Universidad de Valladolid

Facultad de Ciencias

Clasificación de fallos en motores en estado transitorio mediante redes neuronales

Autor:

Miguel Toquero Barón

Tutores:

Miguel Alejandro Fernández Temprano

Alejandro Barón García

Trabajo de fin de grado del
Grado en Estadística

Universidad de Valladolid

Junio 2021

No hay nada que sea más grato
Que ver como lo has logrado
Después de haber trabajado
Años de sol a sol
Asi que sin ofenderte
Mejor no me desees suerte
Desea que sea persistente
En mi lucha por ser mejor.

El Chojin.

Agradecimientos

Quiero agradecer a todos los que me habéis apoyado y motivado para llegar hasta aquí, los que día a día me animáis y dais fuerza para sacar lo mejor de mí.

Resumen

Los motores eléctricos de inducción son una pieza clave en el desarrollo industrial en la actualidad. Cuando un motor falla requiere la detención de toda la producción para repararlo o cambiarlo. De ello, nace la necesidad de detectar los fallos antes de que ocurran, a ser posible mediante técnicas no invasivas que permitan la monitorización del motor mientras está funcionando.

En este trabajo de fin de grado se considera un problema de clasificación de fallos en motores mediante los datos del armónico de la onda superior e inferior de la corriente de alimentación del motor en una situación que, según los expertos, es especialmente compleja ya que se considera la corriente durante el estado transitorio del motor, motores alimentados por distintos inversores, lo cual también complica la clasificación, y se efectúa una clasificación multiestados, ya que se clasifica el estado del motor en cinco estados diferentes según el nivel de degradación del motor.

Se utilizan técnicas de aprendizaje profundo, como son las redes neuronales, comprobando su desempeño en el problema. Se obtiene que, en este problema, las redes neuronales necesitan, al igual que sucedía con las técnicas boosting, un preprocesado de los datos para una mejor solución del problema. La precisión de la clasificación obtenida es considerablemente buena, con valores similares a las técnicas boosting. Se obtienen además conclusiones interesantes desde el punto de vista industrial, como la confirmación de que el nivel de carga alto permite mayor precisión en la clasificación, y que el armónico de la banda inferior resulta más informativo que el armónico de la banda superior. También se hace uso de las técnicas recientemente desarrolladas en la literatura que permiten la interpretabilidad de este tipo de modelos de caja negra, posibilitando extraer información sobre cómo se realizaron las predicciones y cuantificar de esta manera la intuición previa que se tenía sobre cuáles son las variables más interesantes en el problema.

Abstract

Nowadays, electric engines are a key factor in industrial development. Whenever there is an engine failure, the whole production needs to stop in order to repair or change the piece. Hence, there is an arising need to detect faults before they occur. If possible, non-invasive techniques are preferred since they allow monitoring the engine while it is running.

In this Bachelor thesis, we evaluate an engine failure classification problem through both upper-side and lower-side harmonic band power supply. According to the experts, the considered situation is especially complex due to several factors: power supply during the transitory state of the engine is assessed; inverter-fed induction engines are studied, a fact that complicates the analysis; and a multi-state classification takes place, which means the engine is categorized in five different states according to its degradation level.

Deep Learning techniques, such as neural networks, are applied, and their performance solving the problem is evaluated. Obtained results show neural networks need, same as boosting techniques, pre-processed data to improve the solution to the problem. Accuracy of the obtained classification is remarkable, with similar values to those obtained with boosting techniques. Furthermore, we also get interesting conclusions from an industrial point of view, such as the confirmation that a high charge level allows better precision in the classification task and that the lower-side harmonic band turns to be less informative than the upper-side harmonic band. In addition, recently developed techniques that have been previously described in the literature are adopted to interpret the black box model, enabling the extraction of the information on how predictions were made. In such manner, our thoughts on which were the most useful variables to solve the problem can be quantified.

Índice general

Capítulo 1

Introducción

1.1 Descripción del problema

Los motores de inducción son máquinas eléctricas que se encargan de obtener energía mecánica a partir de energía eléctrica por medio de la rotación de los campos magnéticos generados en sus bobinas.

En 2015 el 80 % de los motores utilizados en la industria eran motores de inducción [1]. La rotura de uno de los componentes de un motor de inducción resulta crucial en el proceso de producción, ya que este suceso conlleva el cese de la actividad de producción para reparar la pieza, lo que puede suponer un gran coste en la evaluación de pérdidas. Por todo esto, está en alza el conocido como mantenimiento predictivo, es decir la investigación sobre técnicas y métodos de diagnóstico previos al fallo.

Una de estas técnicas es el denominado *Motor Current Signature Analisis*, MCSA [2], que consiste en la evaluación del estado del motor, a partir de sus corrientes de alimentación. Su principal ventaja es ser una técnica no invasiva, es decir que no requiere interrumpir el funcionamiento para llevar a cabo el análisis. Dentro de este análisis de corriente del motor, en este trabajo de fin de Grado, se utilizan las frecuencias de la tensión de alimentación del motor *upper-sideband harmonic*, USH, y *lower-sideband harmonic*, LSH, se llevará a cabo el proceso de detección y diagnóstico.

El problema que se considera en este trabajo es particularmente interesante puesto que tiene varias características que lo complican. En primer lugar, se consideran motores alimentados por inversores, que es la situación habitual en la práctica, y no motores alimentados directamente de la red. Esto hace que la señal sea menos limpia y más difícil de clasificar. En segundo lugar, se considera la señal emitida por el motor en estado transitorio, es decir, cuando está comenzando a funcionar y no cuando ya está estabilizada (estado estacionario) lo que requiere una transformación funcional de los datos más compleja y también complica el problema de la clasificación. Adicionalmente se considera la clasificación en cinco estados diferentes de deterioro y no simplemente la clasificación en motor sano o motor dañado, que

sería mucho más sencilla de conseguir. También se consideran varias covariables en el problema, como tipo de inversor o el nivel de carga, cuya influencia en la calidad de la clasificación de los fallos se quiere valorar para poder determinar si puede establecerse un único modelo de clasificación o deben considerarse varios en función de los valores de esas covariables.

Tras observar en trabajos previos el buen desempeño de las técnicas boosting en la detección y clasificación de este tipo de fallos [3], en este trabajo se quiere comprobar si las redes neuronales son capaces de mejorar los resultados obtenidos con esa técnica. Se tratará de entrenar y obtener modelos válidos con los que llevar a cabo el diagnóstico con un alto grado de acierto, explorando, en primer lugar, la metodología de redes neuronales y su comportamiento ante este problema, para, a continuación, comprobar si en este problema las redes neuronales son capaces de seleccionar las variables más relevantes para clasificar los fallos o si les es conveniente la ayuda proporcionada por una reducción de dimensión en los datos del problema. A continuación se establecerán uno o varios modelos de clasificación que permitan obtener un diagnóstico preciso y comprobar hipótesis anteriores de expertos y otros trabajos (como que niveles de carga superiores mejoran la precisión en el diagnóstico, que cada modelo de inversor necesita un clasificador propio o si la información que porporcionan las bandas LSH y USH es o no redundante).

Hay alguna referencia en la literatura científica del uso de redes neuronales en la clasificación de fallos en motores de inducción. Tal vez la referencia más interesante sea [4] donde utilizando un método basado en *wavelets* y redes neuronales se analiza la corriente en estado estacionario de motores conectados directamente a la red y se clasifican dichos motores en dos estados (motor dañado o motor sano) logrando un alto porcentaje de acierto. En este trabajo se va más allá puesto que se consideran motores alimentados por distintos inversores y se analiza la señal de su estado transitorio mediante una transformada funcional más compleja y se clasifican los motores en cinco estados diferentes dependiendo del grado de deterioro del motor. No conocemos referencias en la literatura en las que se trate este tipo de problema mediante el procedimiento que aquí se desarrolla.

Empero, la modelización estadística no se realiza únicamente con objeto de predicción, si no de adquirir conocimiento en base a los datos en base a una representación (modelo) de la realidad. Es por esto que, mediante técnicas de interpretación, se pueden extraer los conceptos aprendidos por el clasificador de forma comprensible por el ser humano. Esto permite no solo verificar un correcto aprendizaje del clasificador si no que permite extender el conocimiento previo que se tiene sobre un dominio de aplicación, en este caso, del MCSA.

1.2 Objetivos del trabajo

Las redes neuronales actualmente están presentes en casi todos los ámbitos hasta donde llega la tecnología. Su auge en la actualidad nos motiva a tratar de resolver este problema con su utilización, esperando obtener unos buenos resultados.

Los objetivos que se esperan alcanzar en este trabajo son los siguientes:

- Valorar la capacidad de las redes neuronales para abordar el problema de clasificación de fallos de motores anteriormente descrito.
- Comprobar si las redes neuronales son capaces de seleccionar las variables más interesantes de los datos para la solución del problema o si necesitan de un preprocesado de los datos que reduzca su dimensionalidad.
- Comprobar si las redes neuronales mejoran a los métodos boosting.
- Conocer qué factores o covariables influyen en la clasificación.
- Obtención de modelos interpretables sobre modelos opacos o de caja negra a través de técnicas punteras como SHAP. A través de estos, verificar qué conceptos ha aprendido el clasificador.

1.3 Estructura del documento

Esta memoria se compone de los siguientes capítulos:

- Metodología. Se exponen los conceptos teóricos y el fundamento de los métodos utilizados en este TFG.
- Datos. Descripción, obtención y procesamiento del conjunto de datos.
- Análisis y resultados. Se desarrolla un análisis de los resultados obtenidos y se comparan los resultados obtenidos en este TFG con otros resultados previos.
- Conclusiones. Se resumen los resultados obtenidos en este trabajo y se proponen nuevas lineas de trabajo o mejora.

1.4 Asignaturas relacionadas

En esta sección veremos la relación de técnicas utilizadas en este trabajo y el aprendizaje de estas en diferentes asignaturas del grado.

- Análisis de Datos, Análisis Multivariante y Análisis de Datos Categóricos: en ellas se estudia el fundamento y las bases de los clasificadores y modelos predictivos, así como las métricas para la evaluación de los clasificadores.
- Fundamentos de Programación, Programación Orientada a Objetos y Paradigmas de Programación: se asientan las bases sobre la programación en distintos lenguajes, entre ellos el utilizado en este trabajo (Python).
- Regresión y ANOVA, Modelos Lineales y Modelos Estadísticos Avanzados: en los que se estudia las técnicas de análisis de la varianza.

- Métodos Estadísticos de Computación Intensiva: se explican algunas de las técnicas de regularización utilizadas en el trabajo.
- Técnicas de Aprendizaje Automático: se realiza una introducción a la implementación de diferentes algoritmos de predicción y clasificación con Python.

1.4.1 Ampliación de materia

Los contenidos ampliados respecto a los estudiados en el grado son los siguientes.

- Búsqueda de hiperparámetros. Uso de procesos automáticos para la búsqueda de hiperparámetros, en este caso la búsqueda aleatoria.
- Estudio pormenorizado y empleo de técnicas de *Deep Learning*.
- Métodos de regularización. Inclusión de métodos de regularización como la detención temprana o el *dropout*.
- Interpretabilidad de modelos. Algoritmos de interpretabilidad para modelos de caja negra como SHAP.
- Programación avanzada. Elaboración de código y programas adecuados para el uso de todos los algoritmos pertinentes.

Capítulo 2

Metodología

En esta sección explicaremos en qué consiste el problema de la clasificación con el que tratamos, así como el método que utilizaremos para obtener una respuesta. También, comentaremos la manera de evaluar la precisión de nuestro clasificador y la manera de interpretarlo. Este apartado es puramente teórico y no contiene resultados aplicados al problema, contiene el fundamento teórico base.

2.1 El problema de la clasificación

Uno de los campos de la estadística con más auge en la actualidad es el de la clasificación supervisada. Esta consiste en asignar una etiqueta a una observación tras haber visto un conjunto de observaciones ya etiquetado. A continuación, haremos un repaso de alguno de los procedimientos más utilizados en el tratamiento de este problema a lo largo de la historia.

2.1.1 Discriminante Lineal de Fisher

Linear discriminant analysis (LDA) fue propuesto por Fisher en 1936 [5]. Consiste en encontrar una combinación lineal de características o atributos que separe dos o más clases. De esta forma se obtiene un hiperplano con la máxima separación posible entre los grupos.

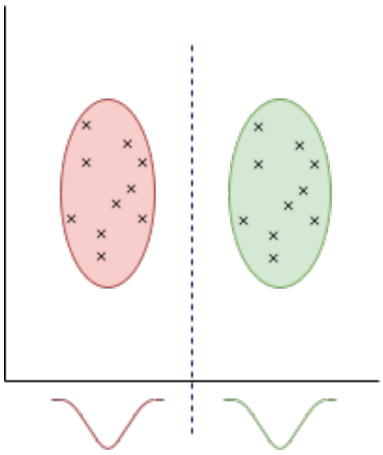

Figura 2.1: Ejemplo LDA.

Es uno de los modelos más sencillos de clasificación y más interpretables. Matemáticamente lo podemos formular generalizando con K clases o grupos. Si $(x-\mu_t)^T\Sigma^{-1}(x-\mu_t) \leq (x-\mu_k)^T\Sigma^{-1}(x-\mu_k), k=1,...,K$ entonces x se clasifica en t.

En la práctica se desconocen los verdaderos valores de los parámetros μ, media, y Σ, varianza, luego se utilizan sus estimadores $\hat{\mu}$ y $\hat{\Sigma}$.

2.1.2 Regresión logística

La regresión logística, modelo logístico o logit se utiliza para modelar la probabilidad de pertenencia a una clase. El método data de 1958 con la publicación de David Cox [6]. La regresión logística se basa en los conceptos de la regresión lineal, donde el modelo es una ecuación lineal que relaciona las características de entrada, X, con la variable objetivo, Y. En este caso la Y será un valor discreto que indica la clase a la que pertenece, el ejemplo típico es una variable de respuesta binaria.

$$logit(Y) = \log\left(\frac{Y}{1-Y}\right) = \beta X \tag{2.1}$$

Se establece un valor límite, a, a partir del cual se asigna a cada clase.

$$\hat{y} = \begin{cases} 0 & si \quad \beta X < a \\ \\ 1 & si \quad \beta X \geq a \end{cases} \tag{2.2}$$

El estimador juega el papel de probabilidad de pertenencia a la clase.

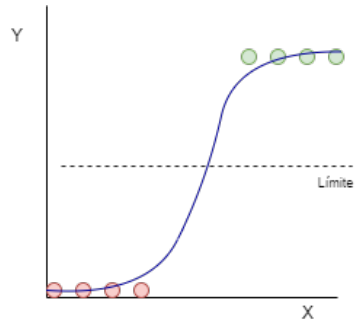

Figura 2.2: Ejemplo de Regresión Logística.

Este modelo también proporciona una amplia interpretabilidad.

2.1.3 Árboles de decisión

Aparecen por primera vez en 1963 en el artículo de James Nelson Morgan y John A. Sonquist [7]. Proponen una algoritmo de clasificación basado en comprobar el cumplimiento de ciertas condiciones. Se presenta un grafo en forma de árbol en cuyos nodos se sitúa cada condición, sobre una única variable, las cuales, de forma enlazada particionan el conjunto de datos en sectores con observaciones de la misma clase. De esta forma, una vez creado el árbol de decisión, para predecir la clase de una nueva observación solo habría que dejarla caer por el árbol y seguir la rama de cumplimiento de cada condición y estimar su clase con la clase asignada a esa partición final.

Es un método con alta interpretabilidad ya que sigue unos pasos similares al razonamiento humano. Sin embargo, es un método poco robusto, pequeñas modificaciones en el conjunto de datos podrían conllevar grandes cambios en la estructura del árbol, y muy sensible al sobreajuste [8]. Una de sus grandes ventajas es que se puede utilizar tanto para variables numéricas como categóricas pues elige las condiciones de los nodos basándose en la entropía.

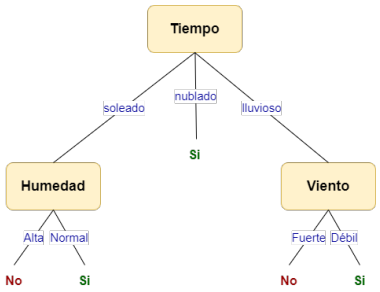

Figura 2.3: Ejemplo de Árbol de Decisión con los datos de *Play Tennis*[9].

2.1.4 Redes neuronales

Es una de las técnicas con mayor aceptación y popularidad en los últimos años debido a los avances computacionales que imposibilitaban su uso en el pasado. Es útil para tratar una gran cantidad de datos de diversas estructuras, no solo datos tabulares. Dado que esta será la metodología que se empleará en este trabajo, se dedicará la sección siguiente a su explicación detallada.

2.2 Redes neuronales

En esta sección veremos con todo lujo de detalles qué es una red neuronal, sus unidades de cómputo o neuronas y cómo se relacionan entre ellas. Explicaremos el proceso de entrenamiento de las mismas y detalles acerca de su configuración o arquitectura. También se comentarán técnicas más avanzadas que se pueden aplicar en las redes neuronales para mejorar su comportamiento como el *Dropout* o la parada temprana.

2.2.1 Neurona

Se llama neurona, neurona artificial o perceptrón simple a las unidades de cómputo de una red neuronal. El concepto fue desarrollado por Frank Rosenblantt, quien se inspiró en el trabajo previo de Warren McCulloch y Walter Pitts [10]. En 1943, se publica su trabajo en el cual, basándose en el comportamiento del sistema nervioso de "todo o nada", se recibe la señal nerviosa o no se recibe y las relaciones entre estas señales nerviosas se describe un modelo matemático, mediante la lógica proposicional, que trata de imitar el comportamiento de una neurona biológica.

Este modelo se basa en la recepción, a través de la sinapsis, de varias señales en la neurona. Ésta procesará las entradas y tendrá una salida en forma de activación del mismo modo "todo o nada". De esta manera, se describe la lógica haciendo que una neurona pueda representar puertas lógicas para su entrada. Esta es la primera descripción de la neurona artificial.

La neurona descrita por McCulloch y Pitts tiene una respuesta binaria.

$$y \in \{0, 1\} \tag{2.3}$$

Donde y indica la activación de la neurona, si esta es igual a 1, o por el contrario si no se activa cuando es igual a 0. También, tiene un número N de excitaciones o señales nerviosas. Estas entradas de la neurona también son binarias.

$$x_k \in \{0, 1\},\ k = 1..N \tag{2.4}$$

La neurona tiene un valor de inhibición i, el cual impide la activación de la neurona cuando está encendido. Por último, encontramos un valor límite Θ que juega el papel de frontera

para la activación. Si la suma de las entradas supera el valor límite la neurona se activará y su salida será $y = 1$, siempre que el inhibidor esté desactivado.

$$\sigma(x) = \begin{cases} 1 \quad si \quad \sum_{k=1}^{N} x_k > \Theta \quad \text{e} \quad i = 0, \\ \\ 0 \qquad\qquad\qquad en\ otro\ caso. \end{cases} \tag{2.5}$$

Con esto podríamos representar algunas puertas lógicas como AND, OR o NOT. Sin embargo, resulta imposible la representación de la puerta XOR con una única neurona.

Podemos observar la representación de un puerta lógica AND con dos entradas. Trazando una línea obtenemos una separación para los dos posibles resultados de salida.

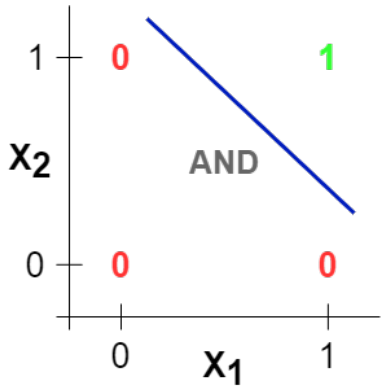

Figura 2.4: Puerta AND.

Nuevamente, la función de una puerta lógica OR resulta linealmente separable.

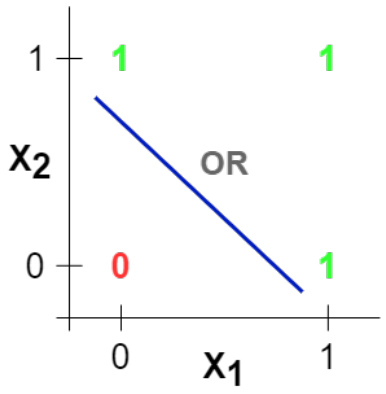

Figura 2.5: Puerta OR.

Por último, mostramos la representación de la función XOR o OR exclusivo. En este caso, sea cual sea la linea recta que tracemos no lograremos separar los conjuntos de respuestas.

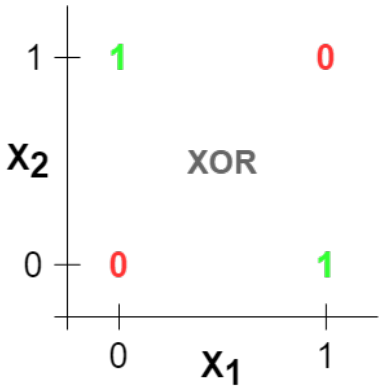

Figura 2.6: Puerta XOR.

Podemos comprobar que una sola neurona artificial no es capaz de representar la función boleana XOR ni ninguna otra que no sea linealmente separable. Debemos fijarnos en que los pesos de cada variable de entrada son iguales entre ellos e iguales a uno.

Partiendo de esta idea, Frank Rosenblantt inventó el perceptrón simple [11] o la neurona artificial tal y como la conocemos hoy en día. El fundamento es similar al empleado por McCulloch y Pitts, sin embargo, ahora las restricciones son menos estrictas: las entradas tienen unos pesos, de forma que juegan el papel de una regresión en la que no todas las entradas tienen la misma contribución, el valor de la inhibición deja de ser absoluto.

$$w_1 \cdot x_1 + w_2 \cdot x_2 + ... + w_N \cdot x_N + b \tag{2.6}$$

Encontramos un símbolo añadido a la suma b, este es el bias o lo que hemos llamado anteriormente Θ, que como ya se ha mencionado es el límite de la activación. Estas modificaciones dieron paso al uso de la neurona artificial para aprendizaje supervisado, permitiendo a la neurona aprender los pesos de las variables de entrada por sí misma.

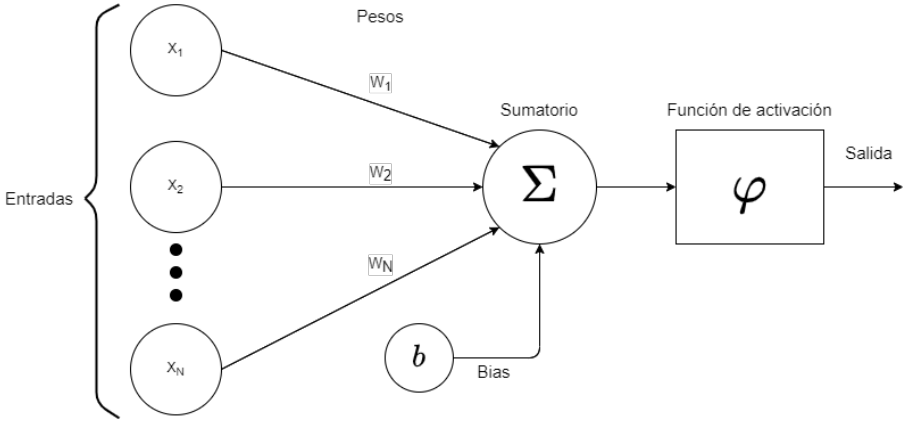

Figura 2.7: Neurona artificial

Como cada variable x_i de entrada tiene un peso asociado, nos permite modificar su influencia en el resultado final libremente.

Tras realizar la suma la señal o resultado se propaga a la función de activación, la cual veremos a continuación con más detalle.

Por todo esto, podemos definir la neurona artificial como nodo computacional, basado en el comportamiento de las neuronas biológicas humanas, que a partir de unas entradas numéricas procesadas con una función matemática devuelven un valor de salida también numérico. La unión de varias neuronas artificiales da lugar a las redes neuronales, como veremos más adelante.

2.2.2 Funciones de activación

Hasta ahora nada diferencia el modelo de la neurona artificial de una regresión múltiple. Sin embargo, la función de activación lo cambia todo. Esta es la encargada de introducir no-linealidades en el modelo. De esta manera, se permite aprender funciones mucho más complejas y adaptarse a aquellas que no sean linealmente separables. Podemos decir que una función de activación es una función matemática aplicada a la salida de la neurona para modificar y añadir deformaciones no lineales [12]. Estas son algunas de las propiedades que buscamos en una función para poder utilizarla como función de activación:

- No lineal: introduce irregularidades en el modelo para poder aproximar una gran variedad de funciones.
- Diferenciable: interesa que sea diferenciable en todos los puntos para poder aplicar métodos de optimización basados en las derivadas, descenso del gradiente.
- Rango: acota los valores de salida o los transforma dentro de un rango de valores.
- Monótona: cuando la función de activación es monótona se garantiza que la superficie de error asociada a un modelo monocapa es convexa y, por tanto, el óptimo local coincide con el óptimo global.
- Simplicidad: deben ser simples para minimizar el coste computacional.
- Se aproximan a la identidad en el origen: necesario para evitar la influencia de los pesos iniciales aleatorios.

Función identidad
Es una función lineal que obtiene el mismo valor que su entrada, retorna un valor idéntico.

$$f(x) = x \tag{2.7}$$

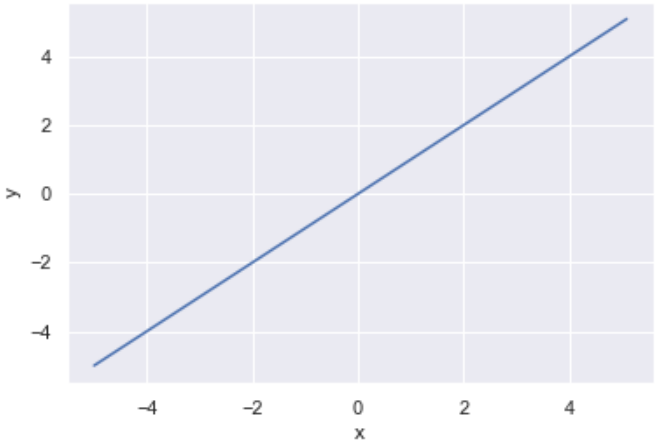

Figura 2.8: Función identidad.

Esta función, aunque puede ser útil en otros problemas, no se adapta a las redes neuronales, ya que no introduce no-linearidades. Por tanto, no nos permitiría adaptarnos a problemas que no sean linealmente separables. Si tuviésemos un modelo de red neuronal muy complejo con muchas capas y neuronas pero todas ellas tuviesen esta función de activación, sería equivalente a tener un modelo con una sola neurona.

Función signo

En las neuronas presentadas previamente se ha utilizado como función de activación la función signo.

$$sgn(x) = \begin{cases} +1 & si\ x \geq 0 \\ -1 & en\ otro\ caso. \end{cases} \tag{2.8}$$

Transforma la entrada en una salida binaria en función del signo. Esto nos da libertad ya que el bias, que hemos presentado con anterioridad, sería el parámetro que nos permitiría ajustar el limite de los dos valores de salida.

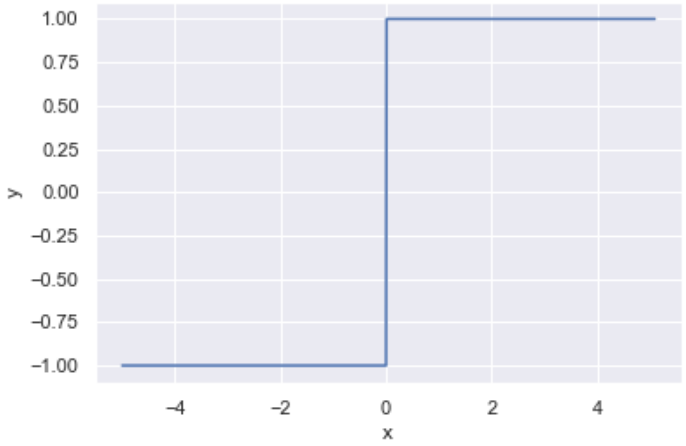

Figura 2.9: Función signo.

Función ReLU

Se conoce con el nombre de rectificador lineal(*Rectified Linear Unit*). Es la función de activación más usada [13]. Tiene una salida igual a cero si su entrada es negativa e igual a la entrada si esta es positiva, es estrictamente creciente.

$$f(x) = \begin{cases} x & si\ x \geq 0 \\ \\ 0 & en\ otro\ caso. \end{cases} \tag{2.9}$$

También se puede expresar como:

$$f(x) = max(0, x) \tag{2.10}$$

La velocidad de construcción, entrenamiento, de modelos con esta función de activación es mayor que otras como la sigmoide. Tiene activación dispersa, es decir, si las neuronas toman valores iniciales aleatorios solo la mitad de ellas se activarían. También tiene desventajas, como que no está centrada en cero y no es diferenciable en 0, pero sí en todos los demás puntos. También encontramos la dificultad de que todos los puntos a la izquierda del cero toman el mismo valor, lo cual nos impide encontrar diferencias. Por desgracia, las unidades $ReLU$ pueden ser frágiles durante el entrenamiento y pueden "morir". Por ejemplo, un gran gradiente que fluya a través de una neurona $ReLU$ puede hacer que los pesos se actualicen de tal manera que la neurona no vuelva a activarse en ningún punto de datos. Si esto ocurre,

el gradiente que fluye a través de la unidad será siempre cero a partir de ese punto. *Dying ReLu* se refiere al problema cuando las neuronas ReLU se vuelven inactivas y sólo dan salida a 0 para cualquier entrada [14]. Podemos observar que no es una función lineal, es lineal a trozos.

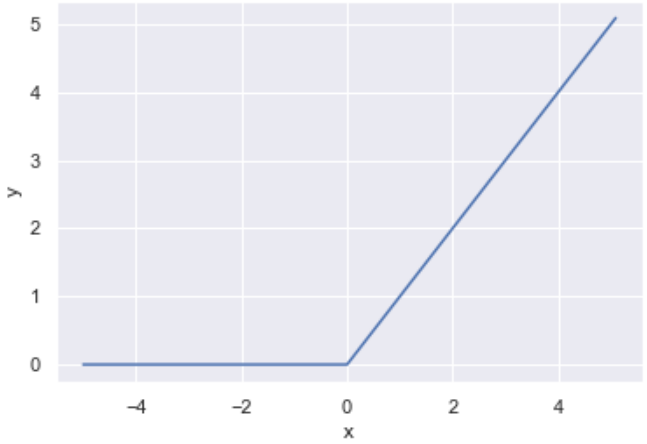

Figura 2.10: Función ReLU

Una de sus mayores desventajas es que no está acotada. Suele ser útil en el reconocimiento de imágenes y redes convolucionales.

Función LeakyReLU
Utilizando la idea anterior del rectificador lineal nace esta variante. Esta función proporciona una corrección para los valores negativos de 2.9, de forma que su valor no sea cero sino un valor muy pequeño que depende de la entrada.

$$f(x) = \begin{cases} x & si\ x \geq 0 \\ 0.01x & en\ otro\ caso. \end{cases} \tag{2.11}$$

Esto corrige el problema de que todos los valores negativos tomasen el mismo valor igual a cero y el problema de *Dying ReLu*.

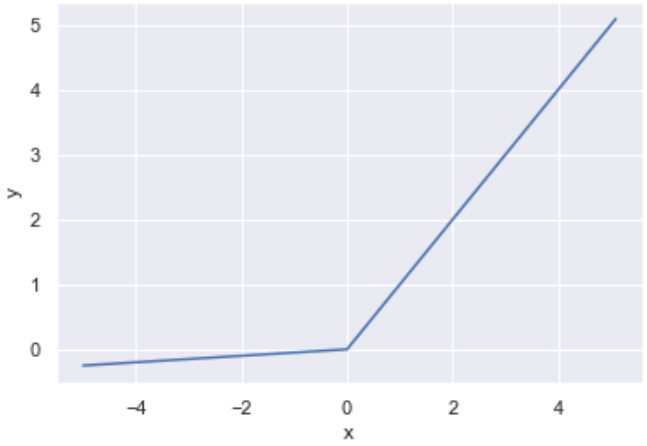

Figura 2.11: Función Leaky ReLU

Existe una versión paramétrica en la que el coeficiente para la x, cuando es negativa, es un valor muy pequeño denominado α.

Al igual que la función ReLU, no está acotada y se comporta bien con imágenes y redes convolucionales.

Función sigmoide
También conocida como función logística. Es especialmente útil en problemas de clasificación binaria ya que su rango de valores es $(0, 1)$, siempre toma valores positivos. Por tanto, el resultado puede interpretarse como una probabilidad. Es una función estrictamente creciente y derivable, tiene una derivada no nula en cada punto, es diferenciable. Por esto, el método del descenso del gradiente puede lograr un mejor resultado en cada paso de la optimización.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.12}$$

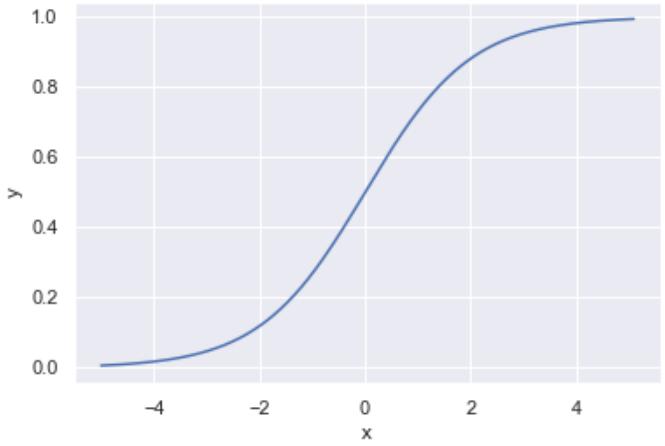

Figura 2.12: Función sigmoide

Esta función tiene lenta convergencia, satura o mata el gradiente, sin embargo, tiene un buen rendimiento en la última capa por su alta interpretabilidad.

Función softmax

Cuando tratamos con problemas de clasificación multiclase utilizamos la función softmax. Con una base matemática muy similar a la de la función sigmoide vista en 2.12, esta función puede tener tantas salidas como se deseen. Por lo general, esto se iguala al número de clases. De esta manera, cada salida con una rango de valores $(0, 1)$ se interpreta como probabilidad de pertenencia a esa clase. Esta función está definida para que la suma de todas sus salidas se igual a uno. Para k clases tenemos la siguiente formulación:

$$f_i(y_i) = \frac{e^{y_i}}{\sum_{j=0}^{k} e^{y_i}} \tag{2.13}$$

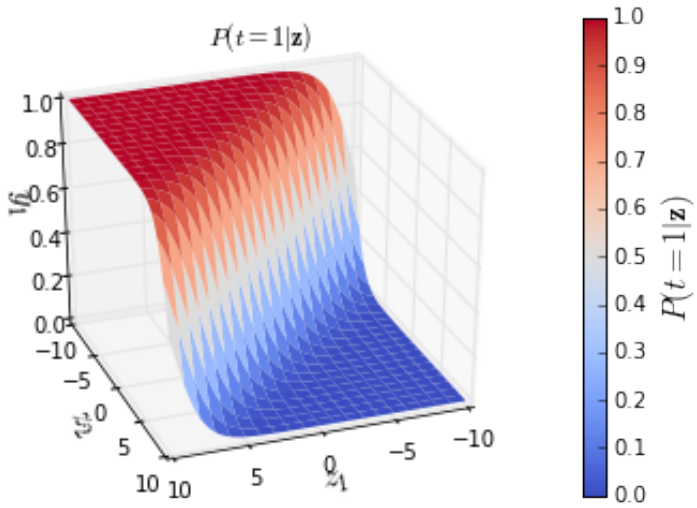

Figura 2.13: Función de activación softmax. Imagen de [15]

Función tangente hiperbólica
También se parece a la función sigmoide. Esta toma valores en el rango $(-1, 1)$. Esta centrada en torno al cero y es estrictamente creciente. En la práctica la optimización es más fácil que para la función logística pero sufre del problema del desvanecimiento del gradiente, problema que se comentará más adelante.

$$f(x) = tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.14}$$

En ocasiones se usa esta otra reformulación, que es equivalente, pero más eficiente computacionalmente al reducir el calculo de exponenciales

$$f(x) = tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{2.15}$$

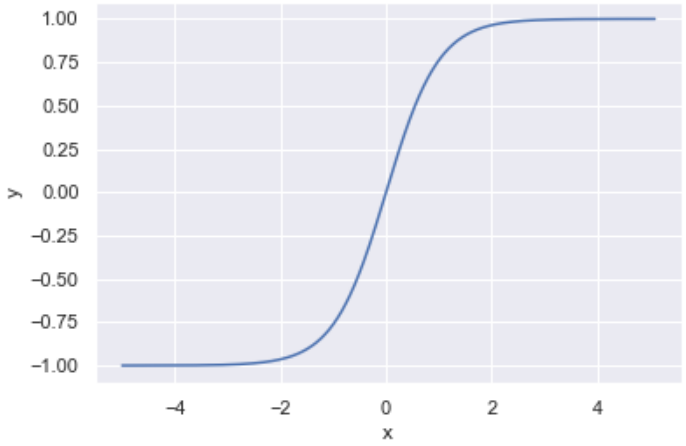

Figura 2.14: Función tangente hiperbólica

Al igual que la función sigmoide tiene lenta convergencia y satura o mata el gradiente. Se utiliza para decidir entre una opción y la contraria por su rango de valores. Tiene buen desempeño en redes recurrentes.

Existen muchas otras funciones de activación y variantes de las aquí presentadas. Sin embargo, en este trabajo solo se usarán las funciones ya mencionadas por su suficiencia, utilidad y popularidad.

Las funciones de activación consituyen uno de los hiperparámetros que se deben ajustar en el modelo.

2.2.3 Arquitectura o topología

Como ya hemos visto, las redes neuronales son combinaciones de neuronas.

Nos referimos a topología o arquitectura cuando hablamos de la forma en que se organizan las neuronas dentro de una red neuronal. En este trabajo se desarrollarán las redes densamente conectadas, con propagación hacia delante. En este tipo de redes, cada neurona toma como entrada las salidas de las neuronas de la capa inmediatamente anterior y su salida se propaga a las todas las neuronas de la capa inmediatamente posterior. Las neuronas se agrupan en capas según su función y su situación dentro de la red. Podemos diferenciar tres capas:

- Capa de entrada (*input layer*): Es la capa que conecta nuestros datos con la red neuronal. Cada neurona perteneciente a esta capa tiene tantas entradas como variables tiene el conjunto de datos utilizado.
- Capa de salida (*output layer*): Recibe como entrada las salidas de las neuronas de la capa anterior y tiene como salida la salida final del sistema. Juega un papel crucial ya

que el número de neuronas en esta capa y el tipo de salida que se desea obtener tiene que adaptarse al problema tratado.

- Capas ocultas (*hidden layers*): Todas las capas que se sitúan entre la capa de entrada y la capa de salida se denominan capas ocultas. Suelen llevar el peso computacional por su gran cantidad de neuronas y porque el número de capas es ilimitado.

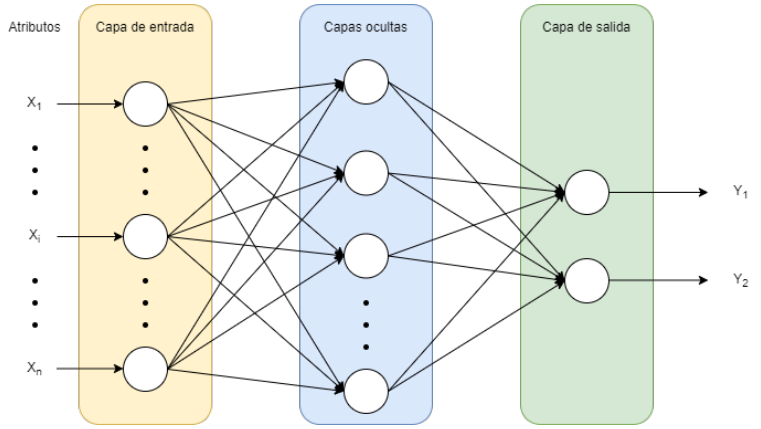

Figura 2.15: Arquitectura de una red neuronal

En la imagen se muestra la topología de una red neuronal con una única capa oculta pero podría tener tantas como se deseen, es decir, puede tener un número arbitrario de capas y un número arbitrario de neuronas dentro de cada capa. El número de neuronas en cada capa no está limitado y se puede adaptar a las condiciones del problema. La posibilidad de encadenar capas con grandes números de neuronas proporciona la posibilidad de representar funciones muy complejas, de esto nace el término de aprendizaje profundo [16].

Encontramos aquí dos hiperparámetros muy interesantes y útiles para modelar diferentes problemas. El teorema de aproximación universal de Hornik nos dice que una red neuronal de una sola capa oculta puede aproximar arbitrariamente bien a cualquier función continua si se le da un número de neuronas suficientes [17].

Como ya hemos comentado, es crucial que el número de neuronas y funciones de activación en la capa de salida se elija cuidadosamente para que se adapte a nuestro problema. Por ejemplo, si tratásemos con una predicción de un valor que juega el papel de tiempo podríamos usar una sola neurona con una función de activación ReLU, que no tiene valores negativos. Sin embargo, en un problema de clasificación binaria deberíamos usar una neurona pero con función de activación sigmoide para poder interpretar la salida como probabilidad de pertenencia a cada clase. Como último ejemplo, si nos encontramos con un problema de clasificación multi-clase debemos utilizar tantas neuronas como clases haya y la función de activación softmax, de forma que también se pueda interpretar la salida como probabilidades de pertenencia.

2.2.4 Entrenamiento

Hemos hablado de los hiperparámetros del modelo, sin embargo, lo que realmente diferencia un modelo de otro con los mismos hiperparámetros son los pesos $w_{i,j}$ siendo i la capa y j la neurona a la que referencia ese peso fórmula 2.6. Como definimos en la sección 2.2.1, estos pesos modificarán completamente el comportamiento y los resultados de la red. Por tanto, la etapa de entrenamiento se centra en obtener unos valores para esos pesos que reproduzcan un buen comportamiento de la red frente a nuestro problema y nuestros datos.

Se trata de convertir el problema en un problema de minimización. Idealmente nos gustaría minimizar el error de clasificación que toma valores de 0 acierto o 1 fallo, por tanto no es una función suave. Por ello, en la práctica, se minimiza otra función que pueda aproximarse a ella y reflejar su información, la función de coste, que veremos en la siguiente sección. Diseñar y entrenar una red neuronal no es muy diferente de entrenar cualquier otro modelo de aprendizaje automático con descenso de gradiente.

La mayor diferencia entre los modelos lineales y las redes neuronales es que la no linealidad de una red neuronal hace que la mayoría de las funciones de pérdida interesantes sean no convexas. No se pueden encontrar mínimos de forma analítica, luego es necesario tomar aproximaciones numéricas basadas en descenso de gradiente. Esto significa que las redes neuronales suelen entrenarse utilizando optimizadores iterativos basados en el gradiente que simplemente llevan la función de coste a un valor muy bajo, en lugar de los solucionadores de ecuaciones lineales utilizados para entrenar modelos de regresión lineal.

La optimización convexa converge a partir de cualquier parámetro inicial. Como funciones de pérdida que usamos no son convexas, pueden presentar puntos de silla que hagan que el gradiente se atasque o muchos mínimos locales peores que el mínimo global . El descenso de gradiente estocástico aplicado a las funciones de pérdida convexas no tiene esa garantía de convergencia y es sensible a los valores de los parámetros iniciales.

2.2.5 Función de coste o pérdida

Esta función representa la suma del error, la diferencia entre el valor predicho y el real. Hablamos de problemas supervisados, es decir, con la variable respuesta conocida. Su función es medir cómo de bien se comporta nuestra red neuronal frente al problema establecido. Consideramos $\hat{y}$ = valor predicho, y = valor real, w = pesos y x = variables de entrada. Calculamos $\hat{y} = w \cdot x$.

Destacamos las siguientes funciones de coste:

- MAE(*Mean Absolute Error*): Se utiliza cuando la salida es un valor escalar. $MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$
- MSE(*Mean Squared Error*): Se utiliza cuando la salida es un valor escalar. $MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$

- *Binary Crossentropy*: Se utiliza cuando estamos ante un problema de clasificación binaria. Sea p la etiqueta real y q la etiqueta predicha.
  $H(x) = \sum_{i=1}^{n} p_i(x) log(q_i(x))$

- *Categorical Crossentropy*: Se utiliza cuando estamos ante un problema de clasificación con varias clases. Para m clases y n muestras, sea y_{ij} la pertenencia de la muestra i a la clase j, valor real, y $\hat{y_{ij}}$ el valor predicho. $L(y, \hat{y}) = \sum_{j=0}^{m} \sum_{i=1}^{n} (y_{ij} \cdot log(\hat{y_{ij}}))$

Las funciones de coste también se deben elegir cuidadosamente según el problema tratado [16].

2.2.6 Descenso del gradiente

La mayoría de los algoritmos de aprendizaje implican algún tipo de optimización. La optimización se refiere a la tarea de minimizar o maximizar alguna función $f(x)$. La mayoría de los problemas de optimización se expresan en términos de minimización de $f(x)$ y la maximización de $-f(x)$. Cuando la minimizamos, también podemos llamarla función de coste, función de pérdida o función de error.

Supongamos que tenemos una función $y = f(x)$, en la que x e y son números reales. La derivada de esta función se denomina $f'(x)$ o $\frac{dy}{dx}$. La derivada $f'(x)$ da la pendiente $f(x)$ en el punto x. En otras palabras, específica cómo escalar un pequeño cambio en la entrada para obtener el cambio correspondiente en la salida: $f(x + \epsilon) \approx f(x) + \epsilon f'(x)$. La derivada es, por tanto, útil para minimizar una función porque nos dice cómo cambia x para conseguir una pequeña mejora en y. Esta técnica, llamada descenso de gradiente [18] consiste en ir moviéndonos pequeños pasos en la dirección opuesta de la derivada.

Cuando la derivada vale cero hemos llegado a un punto en el que no sabemos hacia donde movernos, estamos en un mínimo local. El gradiente $\bigtriangledown f(x)$ se expresa como el vector de derivadas parciales para todas las componentes del vector x. El paso que hemos denominado ϵ, es lo que se conoce como tasa de aprendizaje y es la longitud del paso que damos en la dirección de máximo descenso.

El descenso estocástico del gradiente o *Stochastic gradient descent* (SGD) y sus variantes son probablemente los algoritmos de optimización más utilizados para el aprendizaje automático en general y para el aprendizaje profundo en particular. Es posible obtener una estimación insesgada del gradiente tomando el gradiente medio en un minibloque de muestras extraídas i.i.d. de la distribución que genera los datos. El algoritmo 8.1 muestra cómo seguir esta estimación del descenso del gradiente.

Algoritmo de Descenso del Gradiente Estocástico (SGD)

Requiere: Tasa de aprendizaje fijada $\epsilon_1, \epsilon_2, \ldots$
Requiere: Parámetro inicial θ
$k \leftarrow 1$
while no se cumple el criterio de parada hacer
Tomar una muestra, lote (*batch*), de m ejemplos del conjunto de entrenamiento $\left\{x^{(1)}, \ldots, x^{(m)}\right\}$ con
los objetivos correspondientes $y^{(i)}$.
Calcular la estimación del gradiente: $\hat{g} \leftarrow \frac{1}{m} \nabla_\theta \sum_i L\left(f\left(x^{(i)}; \theta\right), y^{(i)}\right)$
Aplicar la actualización: $\theta \leftarrow \theta - \epsilon_k \hat{g}$
$k \leftarrow k + 1$
end while

Tabla 2.1: Algoritmo de descenso del gradiente [16].

Método de optimización: Se dan unos valores aleatorios iniciales a los parámetros (pesos y bias). Iterativamente estos parámetros se van actualizando en la dirección contraria al gradiente de la función de pérdida. Se actualizan de forma proporcional a un parámetro llamado tasa de aprendizaje. Esto se realiza una vez con cada ejemplo de entrenamiento y constituye una época. El número de épocas es el numero de iteraciones de entrenamiento. En vez de realizar una actualización cada época se pueden realizar varias, utilizando la técnica de lotes. Es decir, se dividen los ejemplos de entrenamiento en varios grupos denominados lotes y la actualización de los pesos se realiza una vez se ha pasado por cada muestra del lote. Una pasada por cada lote constituye una época.

Un parámetro crucial para el algoritmo SGD es la tasa de aprendizaje. Anteriormente, hemos descrito el SGD utilizando una tasa de aprendizaje fija. En la práctica, es necesario disminuir gradualmente la tasa de aprendizaje a lo largo del tiempo, por lo que ahora denotamos la tasa de aprendizaje en la iteración k como l_k. Esto se debe a que el estimador del gradiente SGD introduce una fuente de ruido (el muestreo aleatorio de los ejemplos de entrenamiento) que no desaparece incluso cuando llegamos a un mínimo. En comparación, el verdadero gradiente de la función de coste total se vuelve pequeño y luego 0 cuando nos acercamos y alcanzamos un mínimo utilizando el descenso de gradiente por lotes, por lo que el descenso de gradiente por lotes puede utilizar una tasa de aprendizaje fija [16].

El término de momentum o inercia es un término que se añade a algunas técnicas de optimización para acelerar la rapidez de su convergencia. Consiste en un término que reduce la distancia de movimiento cuando se acerca a los mínimos. También proporciona ventajas en el sentido de lograr una mejor optimización ya que permite salir de mínimos locales aumentando el valor de la inercia [19].

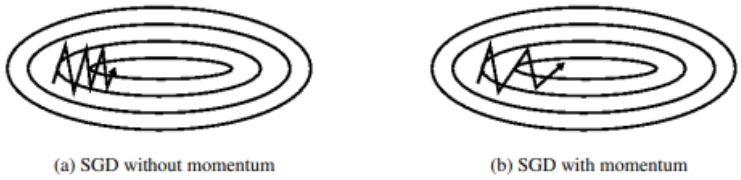

Figura 2.16: Aceleración de convergencia con momentum. Imagen de [19]

2.2.7 Backpropagation-Retropropagación del error

Como ya hemos visto las primeras redes neuronales surgieron sobre 1960 pero no es hasta 1986 [20] cuando se empieza a utilizar el algoritmo de *Backpropagation*. Hasta entonces, el método de entrenamiento de las redes neuronales era algo más rudimentario, consistía en asignar unos pesos de forma aleatoria a las conexiones de las neuronas y comprobar su eficacia en términos de la función de pérdida o de la precisión, a esto lo llamamos pasada hacia delante. Realizando multitud de pruebas y eligiendo aquella cuyos resultados eran los mejores se obtenían los pesos de la red, método conocido como perturbación aleatoria o fuerza bruta. Esto resultaba muy ineficiente computacionalmente.

Con el algoritmo de retropropagación del error cambia el método de calcular los pesos. Si bien es cierto que el punto de partida es el mismo, la asignación de unos pesos iniciales aleatorios, la forma de actualizarlos difiere completamente. Este algoritmo trata de repartir el error entre las neuronas o, dicho de otro modo, asignar a cada neurona un porcentaje del error que ella ha provocado, siguiendo un procedimiento similar al de actualización de pesos como el que se llevaba a cabo en el perceptrón simple. Como vimos en el gradiente, tenemos que obtener con las derivadas parciales como varia el coste o función de pérdida ante un cambio en el parámetro $\frac{\partial C}{\partial w}$. En este caso, la profundidad de las redes neuronales y densidad en cuanto a conexiones hacen que este término sea más complejo de calcular. Por tanto, haremos uso del algoritmo de *Backpropagation* para calcular el vector del gradiente de la red neuronal. Consiste en hacer una evaluación del error hacia atrás, es decir, partiendo por las capas posteriores o últimas capas, hacia las capas anteriores o primeras capas, actualizando los pesos en este proceso.

Veamos a continuación su base matemática. Como ya hemos dicho queremos calcular cómo varía el coste ante un pequeño cambio para cada parámetro de la red neuronal. Sea C la función de coste o de pérdida y w el peso del parámetro.

$$\frac{\partial C}{\partial w} \tag{2.16}$$

También nos interesa para el parámetro de *bias* o sesgo.

$$\frac{\partial C}{\partial b} \tag{2.17}$$

Considerando una red neuronal con L capas, denotamos con un super-índice, $l = 1 \dots L$, a qué capa hace referencia cada parámetro, de forma que las derivadas de los parámetros de la ultima capa serían:

$$\frac{\partial C}{\partial w^L} \quad y \quad \frac{\partial C}{\partial b^L} \tag{2.18}$$

Para calcularlo analizamos la conexión entre el valor del parámetro y el coste final. Recordamos que en una neurona los parámetros w hacen referencia a una suma ponderada que denotaremos como Z.

$$Z^L = W^L + b^L \tag{2.19}$$

Utilizaremos $W^L = w^L \cdot X^{L-1}$, con w^L los pesos en esa capa y X^{L-1} la salida de la capa anterior. El resultado anterior se pasa por la función de activación a.

$$a(Z^L) \tag{2.20}$$

Obteniendo así el resultado de la neurona. El resultado de las activaciones de las neuronas en la última capa conformarían el resultado de la red, que luego sería evaluado por la función de coste C para determinar el error de la red.

$$C(a(Z^L)) \tag{2.21}$$

Para calcular la derivada de una composición de funciones aplicaremos la regla de la cadena. Por tanto:

$$\frac{\partial C}{\partial w^L} = \frac{\partial C}{\partial a^L}\frac{\partial a^L}{\partial z^L}\frac{\partial z^L}{\partial w^L} \tag{2.22}$$

$$\frac{\partial C}{\partial b^L} = \frac{\partial C}{\partial a^L}\frac{\partial a^L}{\partial z^L}\frac{\partial z^L}{\partial b^L} \tag{2.23}$$

Siendo $\frac{\partial C}{\partial a^L}$ cómo varia el coste cuando variamos un poco la activación o la salida de las neuronas en la última capa como vimos en 2.21. Por ejemplo, para una función de coste como el error cuadrático medio: $C(a^L) = \frac{1}{2}\sum_j (y_j - a_j^L)^2$, $\frac{\partial C}{\partial a^L} = (a_j^L - y_j)$.

Por ejemplo, para una función de activación sigmoide $a(z^L) = \frac{1}{1+e^{-z^L}}$ y recordando que $z^L = \sum_i a_i^{L-1} w_i^L + b^L$ tenemos que las derivadas correspondientes serían $\frac{\partial a^L}{\partial z^L} = a^L(z^L)\cdot(1-a^L(z^L))$, para la variación de la salida de la neurona cuando modificamos su suma ponderada, $\frac{\partial z^L}{\partial b^L} = 1$, para la derivada de la suma ponderada frente al *bias* y $\frac{\partial z^L}{\partial w^L} = a_i^{L-1}$ para la derivada de la suma ponderada frente a los pesos, la salida de la activación de las neuronas de la capa anterior. Por tanto, obtendríamos $\frac{\partial C}{\partial z^L} = \frac{\partial C}{\partial a^L}\frac{\partial a^L}{\partial z^L}$ representa la responsabilidad de la neurona en cuanto al error. Es decir, nos indica la responsabilidad de la neurona en el error, también denominado error imputado a la neurona. Lo representamos como δ^L. Con esto podemos simplificar las expresiones anteriores(2.22 y 2.23) de la siguiente manera:

$$\frac{\partial C}{\partial w^L} = \delta^L \tag{2.24}$$

$$\frac{\partial C}{\partial b^L} = \delta^L a^{L-1} \tag{2.25}$$

Para calular las derivadas de las siguientes capas se utiliza la misma intuición, usando la regla de la cadena.

$$\frac{\partial C}{\partial w^{L-1}} = \frac{\partial C}{\partial a^L}\frac{\partial a^L}{\partial z^L}\frac{\partial z^L}{\partial a^{L-1}}\frac{\partial a^{L-1}}{\partial z^{L-1}}\frac{\partial z^{L-1}}{\partial w^{L-1}} \tag{2.26}$$

$$\frac{\partial C}{\partial b^{L-1}} = \frac{\partial C}{\partial a^L}\frac{\partial a^L}{\partial z^L}\frac{\partial z^L}{\partial a^{L-1}}\frac{\partial a^{L-1}}{\partial z^{L-1}}\frac{\partial z^{L-1}}{\partial b^{L-1}} \tag{2.27}$$

Repitiendo el mismo procedimiento podemos simplificarlo de la siguiente manera. Los primeros términos hacen referencia a δ^L. $\frac{\partial z^L}{\partial a^{L-1}}$ es la derivada de la función de activación en la capa $L-1$. $\frac{\partial z^{L-1}}{\partial w^{L-1}} = a^{L-2}$ y $\frac{\partial z^{L-1}}{\partial b^{L-1}} = 1$. Solo nos falta calcular $\frac{\partial z^L}{\partial a^{L-1}}$, que hace referencia a la variación de la suma ponderada de una capa en función de la salida de la capa anterior, es decir, sus entradas. Estamos hablando de la matriz de pesos W^L .

Denominamos $\delta^{L-1} = \frac{\partial C}{\partial z^{L-1}}$ que representa al error de las neuronas en esta capa. Aplicando la misma lógica, esto se extiende a todas las capas anteriores. Con el siguiente algoritmo se pueden calcular todos los errores y las derivadas parciales de nuestra red.

1. Cómputo del error de la última capa.

$$\delta^L = \frac{\partial C}{\partial w^L} \tag{2.28}$$

2. Retropropagación del error a la capa anterior.

$$\delta^{L-1} = W^L \delta^L \cdot \frac{\partial a^{L-1}}{\partial z^{L-1}} \tag{2.29}$$

3. Cálculo de las derivadas de la capa usando el error.

$$\frac{\partial C}{\partial b^{L-1}} = \delta^{L-1} \quad ; \quad \frac{\partial C}{\partial w^{L-1}} = \delta^{L-1} a^{L-2} \tag{2.30}$$

Problema del desvanecimiento del gradiente
Se da en las redes neuronales con muchas capas al calcular el error en las neuronas de las primeras capas. El encadenamiento de derivadas es tan grande que el gradiente se hace muy pequeño, se desvanece. Por ello, no se puede calcular correctamente la actualización de pesos en las neuronas de las primeras capas y esto puede provocar que una elección mala de pesos en las primeras capas influya de manera definitiva en el resultado de la red [21].

Esto podría solucionarse utilizando funciones de activación que no provoquen pequeñas derivadas con la *ReLU* o añadiendo en cada capa un *batch normalization*, que consiste en normalizar las entradas a cada neurona para cada lote de entrenamiento [16].

Se denomina gradiente explosivo cuando tenemos la situación contraria, valores excesivamente grandes de la función gradiente.

Optimizadores
Con el conocimiento obtenido a cerca de la tarea de optimización de los parámetros en la etapa del entrenamiento, veremos a continuación que técnicas se van a utilizar en la práctica. No debemos olvidar que todas estas técnicas hacen uso del descenso del gradiente y del algoritmo de retropropagación.

- Descenso del gradiente estocástico o SGD [22]: optimizador con descenso de gradiente y momento. Puede incluirse la aceleración de Nesterov [23].
- RMSprop [24]: mantiene una media móvil del cuadrado de los gradientes y divide el gradiente por la raíz de esta media. Esta implementación de RMSprop utiliza el impulso simple, no el impulso Nesterov. Utiliza esa media móvil para estimar la varianza.
- Adam (*Adaptative moment estimation*) [25]: la optimización de Adam es un método de descenso de gradiente estocástico que se basa en la estimación adaptativa de momentos de primer y segundo orden. Según [26], el método es eficiente desde el punto de vista computacional, tiene pocos requisitos de memoria, es invariable al reescalado diagonal de los gradientes y se adapta bien a los problemas que son grandes en términos de datos/parámetros.

Es interesante destacar que aunque la convergencia con Adam es más rápida, el algoritmo SGD generaliza mejor [27].

2.2.8 Regularización

La regularización es un conjunto de técnicas que influye en el aprendizaje para que el algoritmo generalice mejor. Dicho de otro modo, son técnicas útiles para evitar o reducir el sobreajuste. De esta manera, nuestro algoritmo se adaptará mejor a los datos que nunca se han visto, así mejorará la precisión cuando se enfrente a datos completamente nuevos del dominio del problema.

La regularización consiste en penalizar los coeficientes de los pesos en los nodos de forma que estos sean más suaves. Es decir, les impide o dificulta tomar valores extremos. Se penalizará más o menos según un valor que será el coeficiente de regularización, parámetro que también se debe optimizar y será un hiperparámetro más de nuestro modelo.

Sobreajuste
Uno de los aspectos más importantes a la hora de entrenar redes neuronales es evitar el sobreajuste. Como es un aspecto muy conocido aqui haremos una breve recapitulación.

El sobreajuste se refiere al fenómeno en el que una red neuronal modela muy bien los datos de entrenamiento pero falla cuando ve nuevos datos del mismo dominio del problema. El

sobreajuste está causado por el ruido en los datos de entrenamiento que la red neuronal capta durante el entrenamiento y lo aprende como un concepto subyacente de los datos.

Este ruido aprendido, sin embargo, es único para cada conjunto de entrenamiento. En cuanto el modelo ve nuevos datos del mismo dominio del problema, pero que no contienen este ruido, el rendimiento de la red neuronal empeora mucho.

Cuando las redes neuronales alcanzan suficiente complejidad se adaptan perfectamente a los datos de entrenamiento aprendiendo asi su ruido en el modelo. Esto significa que la red neuronal en un momento determinado del periodo de entrenamiento no mejora su capacidad de resolver el problema sino que simplemente empieza a aprender alguna regularidad aleatoria contenida en el conjunto de patrones de entrenamiento [28].

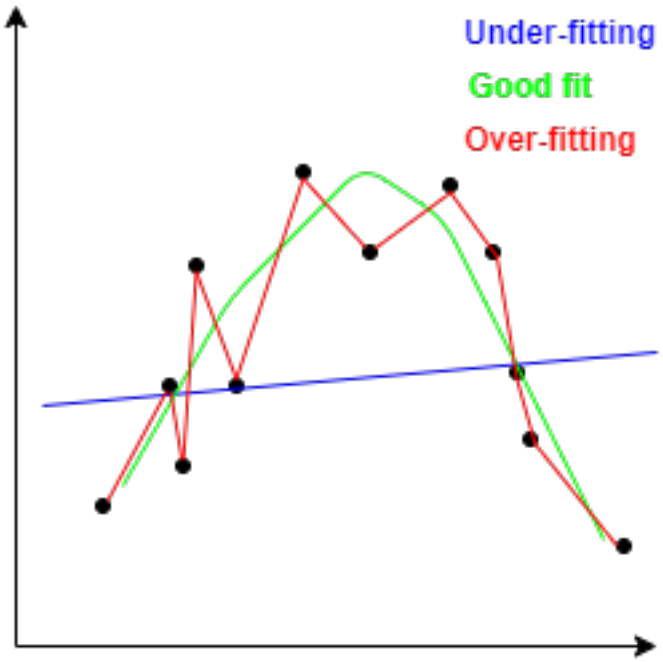

Figura 2.17: Ejemplo de sobreajuste.

Regularización L2
Esta técnica se conoce como decaimiento del peso, *weight decay*, o regresión Ridge.

$$funcion\ de\ coste = funcion\ de\ coste + \lambda \sum_{j=1}^{p} \beta_j^2 \tag{2.31}$$

Cabe mencionar que: λ es el parámetro que decide cuanto queremos penalizar y β son los coeficientes de los pesos que anteriormente hemos denominado w. El término añadido con lambda es la regularización, en este caso se utiliza la norma euclídea sobre los pesos de las conexiones entre neuronas.

Cuando $\lambda = 0$ la regularización queda sin efecto y se obtiene el mismo resultado que si no se aplicase. Sin embargo, cuando $\lambda \to \infty$ la penalización aumenta y los estimadores de los coeficientes tienden a 0.

Regularización L1
La regularización L1, también conocida como regresión Lasso, varía un poco respecto de la regularización L2, cambia el término de regularización.

$$funcion\ de\ coste = funcion\ de\ coste + \lambda \sum_{j=1}^{p} |\beta_j| \tag{2.32}$$

Este realiza selección de variables ya que los coeficientes pueden tomar valor 0, produce modelos dispersos.

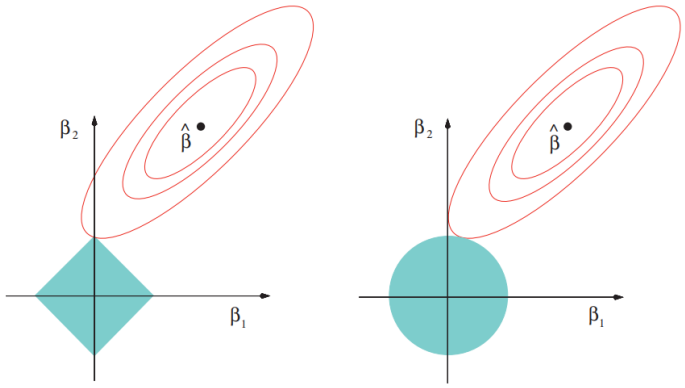

Figura 2.18: Comparación L1(izquierda) y L2(derecha). Imagen de [29]

Intuitivamente, los pesos más pequeños reducen el impacto de las neuronas ocultas. En ese caso, esas neuronas ocultas se vuelven despreciables y la complejidad general de la red neuronal se reduce en ambas regularizaciones L1 y L2. Los modelos menos complejos suelen evitar modelar el ruido en los datos y, por tanto, evitar el sobreajuste. Hay que tener cuidado a la hora de elegir el término de regularización λ. El objetivo es encontrar el equilibrio adecuado entre la baja complejidad del modelo y la precisión.

Si el valor de λ es demasiado alto, el modelo será sencillo, pero corre el riesgo de que infraajuste a los datos. El modelo no aprenderá lo suficiente sobre los datos de entrenamiento para hacer predicciones útiles.

Si el valor de λ es demasiado bajo, el modelo será más complejo y se corre el riesgo de sobreajustar los datos. Aprenderá demasiado sobre las particularidades de los datos de entrenamiento y no podrá generalizar a nuevos datos.
En keras podemos realizar la regularización en distintos puntos [30].

- *kernel_regularizer*: Regularizador para aplicar una penalización en el núcleo de la capa. Esto es, la función con los pesos que realiza la suma ponderada.
- *bias_regularizer*: Regularizador para aplicar una penalización sobre el sesgo de la capa. El término de sesgo, *bias* o término independiente que hemos denominado b.
- *activity_regularizer*: Regularizador para aplicar una penalización a la salida de la capa. En la salida de la neurona tras pasarla por la función de activación.

Dropout
Es una técnica de regularización relativamente sencilla. Consiste en no actualizar los pesos de una neurona, con probabilidad p, en una etapa de entrenamiento. De esta manera, se evita que un nodo o neurona tenga demasiada responsabilidad en la tarea de la clasificación [31].

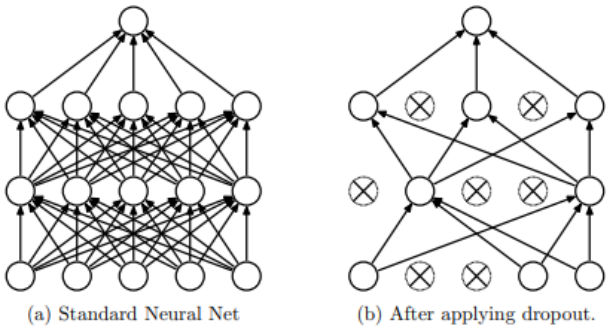

Figura 2.19: A la izquierda: modelo de red neuronal con dos capas ocultas. A la derecha: ejemplo de red regularizada con dropout. Imagen de [32]

En cada iteración el conjunto de neuronas que actualizan sus pesos es diferente. Esto permite capturar más aleatoriedad y generalizar mejor. La probabilidad de que una neurona de una capa se actualice o no es otro hiperparámetro que debemos seleccionar. Esta probabilidad es la misma para todas las neuronas de una misma capa.

Parada temprana
Conocido como *early stop*. Entrenar demasiado tiempo puede provocar un sobreajuste, lo que significa que el óptimo local de la red neuronal sólo puede ser eficaz en sus datos de entrenamiento. La solución es simplemente entrenar la red neuronal durante menos tiempo. Es difícil decidir cuándo es mejor dejar de formarse si sólo se observa la curva de aprendizaje de entrenamiento por sí misma.

El inicio del sobreajuste puede detectarse mediante una validación cruzada en la que los datos disponibles se dividen en subconjuntos de entrenamiento validación y prueba. El subconjunto de entrenamiento se utiliza para calcular el gradiente y actualizar los pesos de la red. Se monitoriza el error de validación con este conjunto durante el entrenamiento.

El error de validación suele disminuir durante la fase inicial del entrenamiento al igual que el error en el conjunto de entrenamiento.

Sin embargo, cuando la red empieza a sobreajustar los datos, el error en el conjunto de validación comienza a aumentar. Cuando el error de validación aumenta durante un número determinado de iteraciones el entrenamiento se detiene, y se devuelven los pesos al mínimo del error de validación. Se detiene el entrenamiento cuando el error de validación del modelo no haya mejorado de forma apreciable durante un número determinado de épocas.

Este es el proceso básico para utilizar la parada temprana:

1. Se pone un contador a 0 y se selecciona una paciencia (un valor entero para el número de épocas que está dispuesto a esperar a que el modelo mejore antes de terminar el entrenamiento).
2. Se entrena la red neuronal durante una época.
3. Se evalúa su rendimiento frente a un conjunto de validación reservado.
4. Si el modelo no tiene más rendimiento que en la última época, incrementa el contador.
5. Si el contador es igual a su paciencia, se detiene el entrenamiento. en caso contrario, se vuelve al paso 1.

Ésta es una técnica sencilla que puede mejorar el rendimiento de la red neuronal con un mínimo esfuerzo[33].

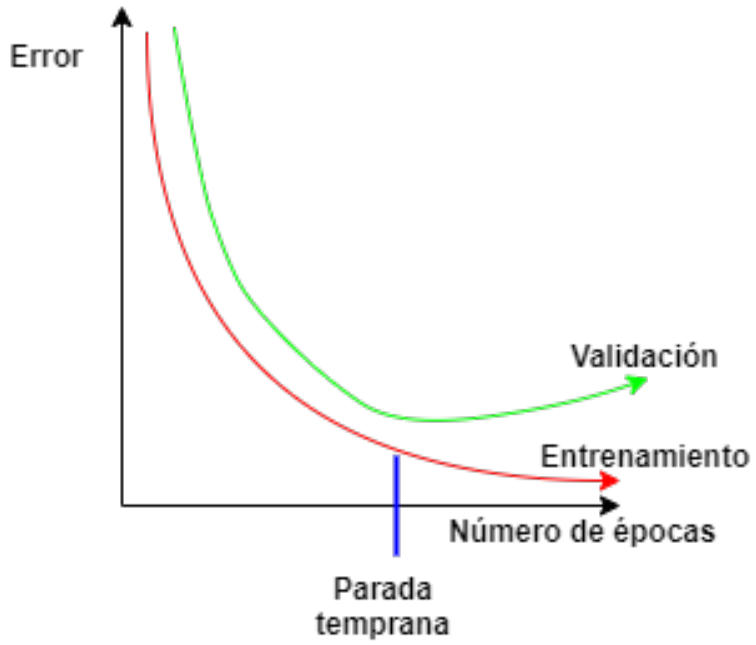

Figura 2.20: Detención temprana del entrenamiento.

A continuación veremos las ventajas e inconvenientes que supone usar esta técnica [28]. Ventajas de la detención temprana:

- Frente a otras regularizaciones, solo se lleva a cabo una vez el proceso de entrenamiento, no una vez para cada posible λ.
- La medida de rendimiento de la validación cruzada se aplica directamente al conjunto real de parámetros de la red que va a utilizar.

Destacamos las siguientes desventajas:

- Método *ad hoc*.
- Depende de los detalles del método de optimización.
- Algunos de los datos de entrenamiento se utilizan sólo para decidir cuándo parar y esto parece un desperdicio.

2.2.9 Hiperparámetros

Ya hemos visto todos los hiperparámetros necesarios para ajustar una red neuronal. También hemos visto que los pesos de las neuronas se entrenan automáticamente con algoritmos de optimización. Ahora debemos elegir que hiperparámetros usar en nuestro modelo.

- Número de capas ocultas.
- Número de neuronas en cada capa.
- Función de activación en cada capa.
- Dropout en cada capa.
- Regularización en cada capa.
- Optimizador.
- Tamaño de lote (*batch*).
- Número de épocas.

Con estos hiperparámetros definimos un espacio de búsqueda. Limitamos el número máximo de capas ocultas a 3. El número de neuronas en cada capa será un múltiplo de 2 entre 2 y 32. Las funciones de activación podrán ser alguna de las definidas en Funciones de activación. El *dropout* tomará un valor distinto en cada capa entre 0 y 0.3 con saltos de 0.05. El coeficiente de regularización tomará valores entre 0 y 0.05. Se elige uno de los optimizadores comentados en la sección 2.2.7. El número de épocas se establece entre 30 y 200. El tamaño de lote se busca en las potencias de 2 hasta un máximo de 2^5.

Búsqueda aleatoria
La búsqueda aleatoria o *Random Search* sustituye la enumeración exhaustiva de todas las combinaciones por una selección aleatoria. Esto puede aplicarse de forma sencilla al entorno discreto descrito anteriormente, pero también se generaliza a los espacios continuos y mixtos. Puede superar a la búsqueda en cuadriculada (*Grid Search*), especialmente cuando sólo un pequeño número de hiperparámetros afecta al rendimiento final del algoritmo de aprendizaje automático [34]. Este artículo muestra como la búsqueda aleatoria es más eficiente en la elección de hiperparámetros que la búsqueda en cuadrícula. La búsqueda aleatoria funciona desplazándose de forma iterativa a mejores posiciones en el espacio de búsqueda, que se muestrean a partir de una hiperesfera que rodea la posición actual [35]. Los experimentos aleatorios son más eficientes, respecto al tiempo, que los experimentos de cuadrícula para la optimización de hiperparámetros porque no todos los hiperparámetros son igualmente importantes. Los experimentos de búsqueda en cuadrícula asignan demasiados ensayos a la búsqueda en dimensiones que no importan y sufren de una pobre cobertura en las dimensiones que son importantes, aunque, al explorar todo el espacio en una cantidad de tiempo mucho mayor, logran encontrar mejores resultados. Cuando se utiliza un conjunto de validación relativamente pequeño, la incertidumbre que conlleva la selección del mejor modelo por validación cruzada puede ser mayor que la incertidumbre en la medición del rendimiento del conjunto de pruebas de cualquier modelo [34]. Nosotros utilizaremos la búsqueda aleatoria por limitaciones en la capacidad de cómputo, sacrificando de esta manera algo de precisión.

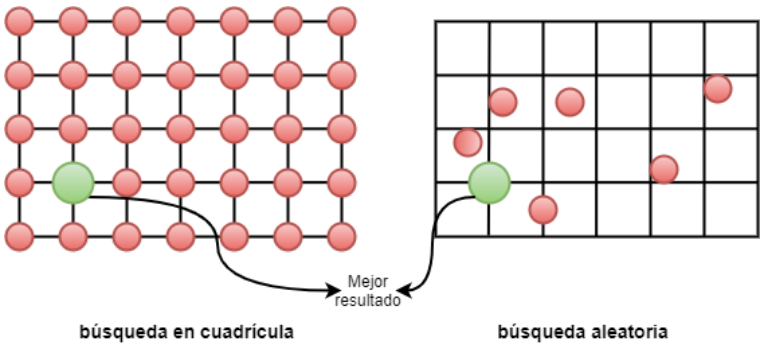

Figura 2.21: Búsqueda aleatoria frente a búsqueda en cuadrícula.

2.3 Estimación del error

Queremos saber como se comporta el modelo que estamos utilizando. Es decir, como de bueno es el modelo en el contexto del problema. Para ello, vamos a evaluar cómo predice la clase de las observaciones. Nos limitamos al ámbito de los clasificadores, aprendizaje inductivo basado en ejemplos.

2.3.1 Tarea de inducción de un clasificador

Dados:

- Descripción de instancias X, conjunto de atributos y valores
- Descripción de las hipótesis H, espacio de árboles de decisión, reglas o demás clasificadores.
- Concepto objetivo $c : X \rightarrow 0, 1$.
- Ejemplos x, ya sean positivos o negativos, que constituyen el subconjunto D, observaciones disponibles del dominio. Pares $(x, c(x))$

La tarea de inducción de un clasificador consiste en determinar la hipótesis que coincida con el concepto en todo el espacio de instancias perteneciente al dominio.

$$h \in H / h(x) = c(x) \quad \forall x \in X \tag{2.33}$$

Hay muchos motivos por los cuales $h(x) \neq c(x)$ para algún $x \in X$.

- El concepto c no pertenece al espacio de hipótesis H.
- No se puede describir con el lenguaje de hipótesis.
- Originalmente $c \in H$, pero hay ejemplos con ruido.
- Aunque pertenezca, entrenamos con un subconjunto de X, los ejemplos de D,

$$card(D) \ll card(X) \tag{2.34}$$

  Especialmente si D es poco representativo del comportamiento de c en X.

En la tarea de clasificación de una observación observamos dos posibles escenarios:

- Acierto: la clase predicha coincide con la clase real de la observación.
- Fallo: la clase predicha no coincide con la clase real de la observación.

Entonces, usamos la tasa de error como medida del rendimiento. La tasa de error es la proporción de errores cometidos sobre todo el conjunto de ejemplos.

Como ya hemos mencionado, no disponemos de todos los datos del dominio del problema, entonces intentaremos estimar esta tasa de error, que llamaremos error verdadero, con diferentes técnicas.

El error verdadero e_D, de una hipótesis h respecto a un concepto objetivo c y distribución de probabilidad del espacio de instancias D es:

$$e_D(h) = Pr_{x \in D}[c(x) \neq h(x)] \tag{2.35}$$

La tasa de error $e_S(h)$ de una hipótesis h respecto a un concepto objetivo c y una muestra $S \subset X$, S es una muestra de observaciones del dominio X, es:

$$e_S(h) = \frac{1}{card(S)} \cdot \sum_{x \in S} \delta(c(x), h(x)) \tag{2.36}$$

Siendo δ:

$$\delta(c(x), h(x)) = 1 \quad sii \quad c(x) \neq h(x) \tag{2.37}$$

$$\delta(c(x), h(x)) = 0 \quad sii \quad c(x) = h(x) \tag{2.38}$$

El error de resubstitución e_r o error de entrenamiento, como su propio nombre indica este error se consigue calculando la tasa de error sobre el conjunto de datos utilizado para entrenar el algoritmo. Es inevitablemente optimista, ya que adaptarse mucho a esos datos proporcionará una buena estimación del rendimiento cuando la generalización sobre el dominio puede ser escasa, utilizar esta medida puede conducir a sobreajuste. El error de resubstitución estima tasas de error menores que el error verdadero.

El error en el conjunto de entrenamiento no es un buen indicador del error sobre datos nuevos. Almacenar todos los datos del dominio sería el clasificador óptimo. Como no tenemos disponibles todos los datos del dominio solo se podrían almacenar un pequeño subconjunto de datos, teniendo una óptima precisión sobre este conjunto pero sin garantías de la capacidad de generalizar.

Nos interesa el rendimiento futuro sobre nuevos datos. Para ello se utiliza un conjunto de prueba o *test* independiente del conjunto de entrenamiento. Se obtiene una buena estimación de la tasa de error cuando la hipótesis h y el conjunto de muestra S son independientes.

Normalmente solo se dispone de un conjunto de datos etiquetados. Para evitar el sobreajuste o la mala estimación del rendimiento del clasificador debemos utilizar un conjunto de datos con ejemplos independientes que no se hayan utilizado de ninguna manera en la construcción del clasificador, ni en la etapa de aprendizaje ni en la etapa de procesamiento (estandarización, escalado...). Por ello, divideremos en dos subconjuntos, entrenamiento D y prueba S, los datos de los que disponemos.

Calcularemos la estimación del error verdadero, e_D, con el error de generalización, e_S. Nos basamos en la suposición que los datos de entrenamiento y de prueba son muestras representativas de un mismo problema subyacente.

En ocasiones se suele hablar de un tercer conjunto de datos, el conjunto de validación. Este debe ser independiente del conjunto de entrenamiento y de prueba. Se utiliza para optimizar los hiperparámetros del clasificador antes de calcular una estimación de su error verdadero con el conjunto de prueba.

Normalmente, cuanto más grande sea el conjunto de entrenamiento mejor será el clasificador dado que, a priori, verá más diversidad de instancias del problema a resolver. Cuanto más grande sea el conjunto de prueba mejor será la estimación del error verdadero. Y cuanto más grande sea el conjunto de validación mejores serán los hiperparámetros elegidos.

2.3.2 Estratificación

Consiste en repartir las observaciones entre las particiones de manera que en todas ellas se respete la distribución inicial de clases.

2.3.3 Hold-out

También conocido como método de reserva. Consiste en reservar un cierto número de observaciones del conjunto de datos. Estas observaciones no se utilizan para entrenar el algoritmo de clasificación. Se usarán en una etapa posterior únicamente para estimar el error de generalización. Existe el dilema de que ambos conjuntos deben ser suficientemente grandes para que el entrenamiento del clasificador sea bueno y también lo sea la estimación del error.

La partición más común usada es $\frac{2}{3}$ para entrenar y $\frac{1}{3}$ para estimar el error.

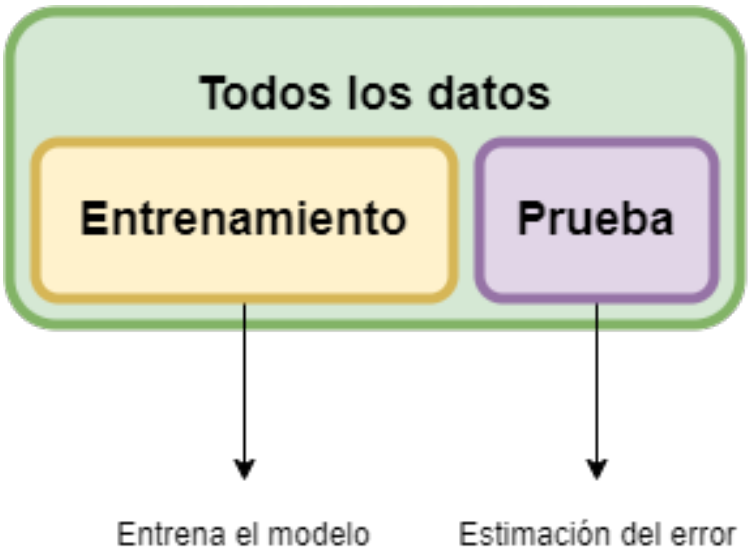

Figura 2.22: Método de reserva.

Su principal inconveniente es la dependencia de los resultados de la partición inicial. Estos subconjuntos podrían no ser representativos, podríamos tener la situación de que una clase no tuviese ningun ejemplo en alguno de los dos conjuntos, ya que la partición se realiza de manera aleatoria. Entra en juego la variante estratifiacada. Esta técnica se puede repetir para obtener unos resultados más consistentes y menos dependientes de la partición inicial.

La tasa de error se estimaría como el promedio de las tasas de error en las repeticiones. No es óptimo ya que existe solapamiento entre los conjuntos en las diferentes repeticiones.

2.3.4 K-fold

Se suele llamar validación cruzada, *XV*. Se reparten los datos con un muestreo aleatorio simple entre las k carpetas, que es un número arbitrario, por lo general 10. Cada carpeta en una partición del conjunto de datos con observaciones independientes de las demás carpetas. En k iteraciones, se entrena el algoritmo con todas las carpetas menos una, que va variando segun la iteración. Finalmente, se estima el error en con la carpeta que no se ha utilizado para entrenar. Esta técnica evita el solapamiento entre subconjuntos y resulta un método más consistente que el anterior. Al mismo tiempo, garantiza que todos los datos se han utilizado tanto para la etapa de entrenamiento como para la etapa de prueba. También se puede utilizar la variante estratificada para que todas las particiones tengan una muestra representativa. La tasa de error estimada se calcula como el promedio de las tasas de error en las k particiones.

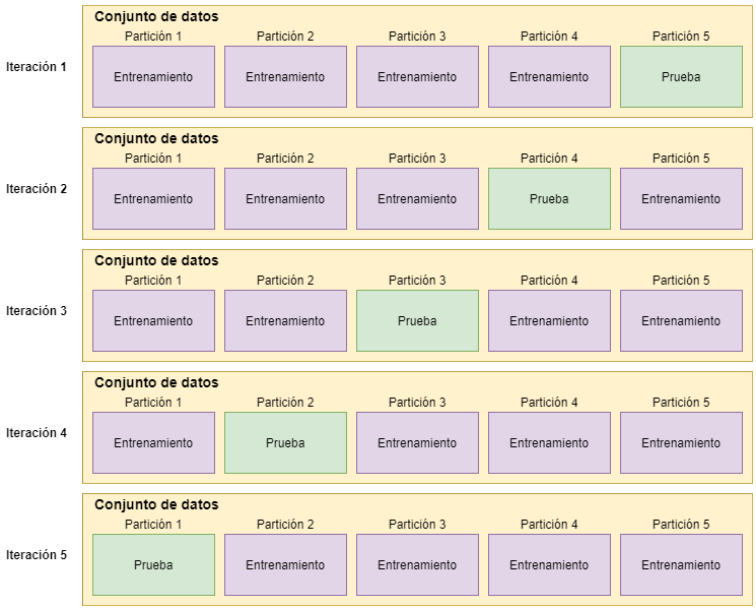

Figura 2.23: Método de validación cruzada con 5 carpetas.

Del mismo modo que con el método de reserva se pueden realizar repeticiones con la validación cruzada. Sin embargo, las repeticiones no son totalmente independientes. Con esta técnica se reduce la variabilidad del error estimado pero aumenta considerablemente el coste computacional.
Para la realización de este apartado se han utilizado los textos [36] y [37].

2.4 Interpretabilidad de modelos complejos

El aprendizaje automático tiene un gran potencial para mejorar productos, procesos e investigaciones. Sin embargo, los ordenadores no suelen explicar sus predicciones, lo que supone una barrera para la adopción del aprendizaje automático. Nos centraremos en algunos métodos de interpretación de modelos de caja negra, como son nuestras redes neuronales, y en la explicación de las predicciones individuales con los valores de Shapley y LIME [38].

Un modelo de caja negra es un sistema que no revela sus mecanismos internos. En el aprendizaje automático, caja negra describe los modelos que no pueden entenderse observando sus parámetros por ejemplo, una red neuronal. Lo contrario de una caja negra se denomina a veces caja blanca o modelo interpretable. Nuestro objetivo es hacer interpretables todos los modelos de aprendizaje automático supervisado.

Una forma de hacer que el aprendizaje automático sea interpretable es utilizar modelos interpretables, como los modelos lineales o los árboles de decisión. La otra opción es el uso de herramientas de interpretación agnóstica de modelos que pueden aplicarse a cualquier modelo de aprendizaje automático supervisado.

Los métodos de diagnóstico de modelos funcionan cambiando la entrada del modelo de aprendizaje automático y midiendo los cambios en el resultado de la predicción. Todos los métodos de diagnóstico de modelos pueden diferenciarse en función de si explican el comportamiento global del modelo en todas las instancias de datos o en predicciones individuales.

Una de las principales desventajas de utilizar el aprendizaje automático es que la información sobre los datos y la tarea que resuelve la máquina queda oculta en modelos cada vez más complejos. Se necesitan millones de números para describir una red neuronal profunda y no hay forma de entender el modelo en su totalidad. Otros modelos, como el bosque aleatorio, *Random Forest*, están formados por cientos de árboles de decisión que votan las predicciones. Para entender cómo se tomó la decisión, habría que mirar los votos y las estructuras de cada uno de los cientos de árboles.

Los modelos ganadores de las competiciones de aprendizaje automático suelen ser conjuntos de modelos o modelos muy complejos, como los árboles potenciados, *boosted trees*, o las redes neuronales profundas.

Los métodos de interpretabilidad de modelos tratan los modelos de aprendizaje automático como cajas negras, aunque no lo sean.

El aprendizaje automático interpretable se refiere a los métodos y modelos que hacen que el comportamiento y las predicciones de los sistemas de aprendizaje automático sean comprensibles para los humanos. Definimos una instancia u observación del conjunto de datos como $x^{(i)}$ cuyo resultado, clasificación, es y^i si este es conocido. Suponemos que las características, atributos, son interpretables por los humanos, es decir, fáciles de entender. Denominamos X

a la matriz con todas las características y todas las instancias. El vector de un atributo con todas las instancias será x_j y el valor de un par instancia-atributo $x_j^{(i)}$. Denominaremos al clasificador como $\hat{f}$ o $\hat{f}(x)$. La predicción es el valor objetivo que asigna el clasificador para unas instancias dadas. Lo denotaremos como $\hat{f}(x^{(i)})$ o $\hat{y}$. Se considera necesario hacer estas aclaraciones por el pequeño cambio de notación utilizado en esta sección.

No existe una definición matemática de interpretabilidad pero una definicion podría ser: Interpretabilidad es el grado en que un humano puede comprender la causa de una decisión [39]. Si un modelo de aprendizaje automático funciona bien, ¿por qué no confiamos en el modelo? Porque quieres saber cómo se hizo la predicción y por qué llegamos a ese resultado. Así nace la demanda de la interpretabilidad.

2.4.1 LIME

LIME, *Local Interpretable Model-Agnostic Explanations* [40], es un modelo de sustituto local. Se utiliza para explicar las predicciones individuales de los modelos de aprendizaje automático de caja negra.

El objetivo es comprender por qué el modelo de aprendizaje automático hizo una cierta predicción. LIME comprueba como varían las predicciones cuando se le dan pequeñas variaciones de los datos al modelo de aprendizaje automático, generando un nuevo conjunto de datos que consta de muestras permutadas y las correspondientes predicciones del modelo de caja negra.

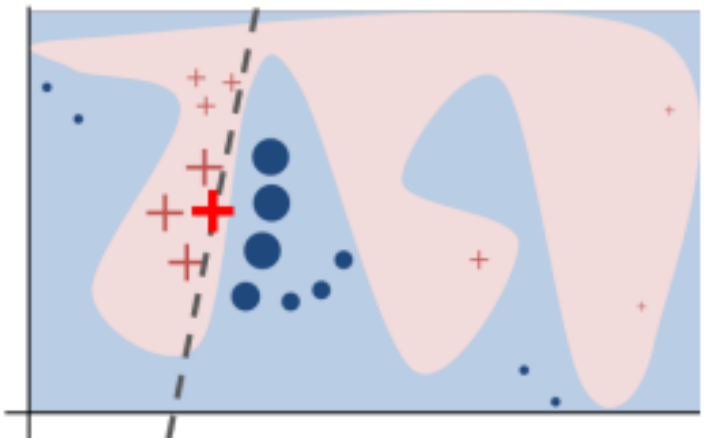

Figura 2.24: Ejemplo de LIME. Imagen de [40]

En la Figura 2.24 aparece un ejemplo simple para presentar de forma intuitiva el significado de LIME. La función de decisión compleja f, desconocida para LIME, está representada por el fondo azul/rosa fondo, que no se puede aproximar bien con un modelo lineal. La cruz roja en negrita es la instancia que se está explicando. LIME muestrea instancias, obtiene sus

predicciones utilizando f y las pondera por proximidad a la instancia que se está explicando, representada aquí por el tamaño de los símbolos en la imagen 2.24. La línea discontinua es la explicación aprendida que es localmente, pero no globalmente, fiel.

Lime debe encontrar un punto entre la complejidad Ω del modelo sustituto g y la fidelidad f, en este caso llevada a verosimilitud L. Este compromiso se puede reflejar con la siguiente fórmula.

$$explanation(x) = arg \min_{g \in G} L(f, g, \pi_x) + \Omega(g) \tag{2.39}$$

Hace referencia a la explicación de la observación x por el modelo g que minimiza la función de pérdida L que mide cuanto de cerca esta del modelo original f. $\Omega(g)$ es la complejidad, que debe mantenerse baja. G es la familia de posibles explicaciones y π_x es el vecindario de la observación x. Supone un compromiso entre la fidelidad y la complejidad del modelo subrogado. En la práctica, se fija esta complejidad y maximiza la verosimilitud.

El modelo interpretable puede ser cualquier modelo interpretable, por ejemplo una regresión Lasso o un árbol de decisión. El modelo aprendido debería ser una buena aproximación de las predicciones del modelo de aprendizaje automático de forma local, no necesariamente de forma global. A esto se le denomina fidelidad local.

El procedimiento para entrenar modelos locales sustitutos:

- Seleccionar la instancia de interés para la que se desea obtener una explicación de la predicción de caja negra.
- Perturbar el conjunto de datos y obtener las predicciones de caja negra para estas nuevas observaciones.
- Ponderar las nuevas muestras según su proximidad a la instancia de interés.
- Entrenar un modelo ponderado e interpretable en el conjunto de datos con las variaciones.
- Explicar la predicción interpretando el modelo local.

LIME crea nuevas muestras al perturbar cada característica individualmente, es decir, muestrea a partir de la observación a interpretar introduciendo un ruido aleatorio.

Ventajas

- Posibilidad de reemplazar el modelo interpretable g subyacente. Pudiendo interpretar el algoritmo que se desee: un árbol de decisión, un SVM... Se puede optar por el modelo interpretable que se desee.
- Usando Lasso o árboles podados, las explicaciones resultantes son cortas, selectivas. Por tanto, las explicaciones son muy fácilmente comprensible para los humanos.

- LIME es uno de los pocos métodos que funciona para datos tabulares, texto e imágenes.
- Ayuda a comprobar la fiabilidad del clasificador y detectar sesgos.

Desventajas

- La definición correcta del entorno local es un gran problema sin resolver cuando se utiliza LIME con datos tabulares. Para cada aplicación, se debe probar diferentes configuraciones de kernel y comprobar si las explicaciones tienen sentido.
- La complejidad del modelo de explicación debe definirse de antemano. Se debe definir el grado que se quiere entre mayor y menor fidelidad a cambio de complejidad.
- Inestabilidad de las explicaciones. Se ha demostrado que las explicaciones de dos puntos muy cercanos variaban mucho en un entorno simulado [41].

2.4.2 Shapley values

Una predicción puede explicarse suponiendo que cada valor de característica de la instancia es un "jugador" en un juego donde la predicción es el pago. Los valores de Shapley es un método de la teoría de juegos de coalición que nos dice cómo distribuir equitativamente el "pago" entre las características o atributos [42].

El valor de Shapley ϕ_j es la contribución marginal promedio de dicha de característica en todas las coaliciones posibles. Es decir, se realizan permutaciones o se eliminan algunas de las características haciendo de nuevo la repartición de responsabilidades. Tras un muestreo de estas alteraciones la responsabilidad asignada o el valor Shapley es el promedio de la responsabilidad de esa característica en cada muestra. El tiempo de cálculo de estos valores aumenta exponencialmente con el número de características.

La interpretación del valor de Shapley para el valor de característica j es: El valor de la característica j contribuyó ϕ_j a la predicción de esta instancia particular en comparación con la predicción promedio para el conjunto de datos.

El valor de Shapley funciona tanto para la clasificación como para la regresión.

Ahora veamos como obtenerlo matemáticamente. En un ejemplo simple de regresión tendríamos:

$$\hat{f}(x) = \beta_0 + \beta_1 x_1 + ... + \beta_p x_p \tag{2.40}$$

Con β_j como el peso del atributo j. La contribución ϕ_j del atributo en la predicción es:

$$\phi_j(\hat{f}) = \beta_j x_j - E(\beta_j x_j) = \beta_j X_j - \beta_j E(X_j) \tag{2.41}$$

El valor de Shapley de un valor de característica es su contribución al pago, ponderado y sumado sobre todas las combinaciones posibles de valor de características:

$$\phi(val) = \sum_{S \subseteq (X_1,...,X_p) \setminus (X_j)} \frac{|S|!(p-|S|-1)!}{p!} val(S \cup \{X_j\}) - val(S) \tag{2.42}$$

Donde S es el subconjunto de atributos usado en el modelo, x el vector de características de la instancia a explicar y $val(S)$ la predicción para los valores de características en el conjunto S que están marginados sobre las características que no están incluidas en el conjunto S:

$$val_x(S) = \int \hat{f}(X_1, ..., X_p) d\mathbb{P}_{X \notin S} - E_X(\hat{f}(X)) \tag{2.43}$$

Se integra sobre cada característica que no pertenece al subconjunto S.

Propiedades

- Eficiencia: La suma de las contribuciones es la diferencia entre el valor esperado y la predicción.

$$\sum_{j=1}^{p} \phi_j = \hat{f}(x) - E_X(\hat{f}(X)) \tag{2.44}$$

- Simetría: cuando dos variables contribuyen de igual manera a todas sus posibles coaliciones sus contribuciones son las mismas.

$$\forall j, k : val(S \cup x_j) = val(S \cup x_i), S \subseteq \{x_1, \ldots, x_p\} \setminus \{x_j, x_k\} \implies \phi_j = \phi_k \tag{2.45}$$

- Una característica que no cambia el valor predicho debe tener un valor Shapley de 0.

$$\forall S \subseteq \{x_1, \ldots, x_p\} \; si \; val(S \cup \{x_j\}) = val(S) \implies \phi_j = 0 \tag{2.46}$$

- Aditividad: Para un juego con varios pagos p^1, p^2, los valores de Shapley respectivos son :

$$\phi_j = \phi_j^1 + \phi_j^2 \tag{2.47}$$

Para obtener una estimación del valor Shapley, se propone una aproximación utilizando el método de Montecarlo [43].

$$\hat{\phi_j} = \frac{1}{M} \sum_{m=1}^{M} \left(\hat{f}(x_{+j}^m) - \hat{f}(x_{-j}^m) \right) \tag{2.48}$$

Donde $\hat{f}(x_{+j}^m)$ es la predicción para x, pero con un número aleatorio de valores de características reemplazados por valores de características de un punto de datos aleatorio z, excepto el valor respectivo de la característica j. El vector x_{-j}^m es casi idéntico a x_{+j}^m, pero el valor x_j^m también se toma de la muestra z. Cada una de estas M nuevas instancias es una creación ensamblada a partir de dos instancias.

Ventajas

- Eficiencia. La diferencia entre la predicción y la predicción promedio está distribuida de manera justa entre los valores de característica de la instancia.
- Permite explicaciones comparativas. En lugar de comparar una predicción con la predicción promedio de todo el conjunto de datos, puede compararse con un subconjunto.
- Base teórica. El valor de Shapley es un método de explicación con una teoría sólida como es la teoría de juegos. Las propiedades dan a la explicación una base razonable.

Desventajas

- Requiere mucho tiempo de cómputo. Un cálculo exacto del valor de Shapley es computacionalmente costoso porque hay 2^k posibles coaliciones de los valores de la variable y la ausencia de una variable. Se reduce limitando el número de iteraciones M. La disminución de M reduce el tiempo de cálculo, pero aumenta la varianza del valor de Shapley.
- Puede malinterpretarse. La interpretación debe ser: Dado el conjunto actual de valores de características, la contribución de un valor de una característica a la diferencia entre la predicción real y la predicción media es el valor estimado de Shapley.
- Siempre se usan todas las variables. El valor de Shapley es el método de explicación incorrecto si se busca explicaciones dispersas.
- Acceso a los datos. No es suficiente acceder a la función de predicción. Se necesitan los datos para reemplazar partes de la instancia de interés con valores de instancias de datos extraídos al azar.

2.4.3 SHAP

SHapley Additive exPlanations [44] es un método para explicar las predicciones individuales, con base en los *Shapley Values*.

El objetivo de SHAP es explicar la predicción de una instancia x calculando la contribución de cada característica a la predicción. El método de explicación SHAP calcula los valores de Shapley a partir de la teoría de juegos de coalición.

Una innovación que SHAP trae es que la explicación del valor de Shapley se representa como un método de atribución de características aditivas, un modelo lineal. Esta visión conecta los valores Shapley y LIME. SHAP especifica la explicación como:

$$g(z') = \phi_0 + \sum_{j=1}^{M} \phi_j z'_j. \tag{2.49}$$

Donde g es el modelo de explicación, $z' \in \{0,1\}^M$ es el vector de coalición, M es el tamaño máximo de la coalición y $\phi_j \in \mathbb{R}$ es la atribución de características para una característica j.

Como tiene su base en los valores Shapley también satisface las propiedades de eficiencia, simetría, simulación y aditividad. SHAP describe las siguientes tres propiedades deseables:

- Precisión local.

$$f(x) = g(x') = \phi_0 + \sum_{j=1}^{M} \phi_j x'_j \tag{2.50}$$

  Se define $\phi_0 = E_X(\hat{f}(x))$ y establece todos x'_j en 1, esta es la propiedad de eficiencia Shapley. Solo con un nombre diferente y usando el vector de coalición.

$$f(x) = \phi_0 + \sum_{j=1}^{M} \phi_j x'_j = E_X(\hat{f}(X)) + \sum_{j=1}^{M} \phi_j \tag{2.51}$$

- Ausencia.

$$x'_j = 0 \Rightarrow \phi_j = 0 \tag{2.52}$$

  Una característica ausente obtiene una atribución de cero, la denominada "propiedad menor de contabilidad" [44].

- Consistencia. Cuando un modelo cambie de modo que la contribución marginal de un valor de un atributo cambia, el valor de Shapley también debe cambiar de la misma manera, ya sea aumentar o disminuir. Con esta propiedad se explica el cumplimiento de las propiedades de linealidad y simetría de Shapley.

Podemos diferenciar entre los siguientes métodos de SHAP:

- KernelSHAP: utiliza la esperanza marginal. Tiene un coste computacional exponencial. Se puede utilizar para aproximar cualquier función.

- TreeSHAP: proporciona una estimación eficiente para modelos basados en árboles. Utiliza la esperanza condicional. Es una alternativa eficiente a *kernel* ya que tiene un coste computacional polinómico. Se beneficia de la propiedad aditiva de SHAP en el ensemble de árboles. es un enfoque basado en la retropropagación que atribuye un cambio a las entradas basado en las diferencias entre las entradas y las referencias correspondientes para las activaciones no lineales.

- DeepSHAP [45]: es un método que utiliza un enfoque basado en la retropropagación para aproximar el valor Shapley a las entradas frente a sus respectivas referencias para funciones de activación no lineales.

Ventajas

- Base teórica sólida. Se basa en la teoría de juegos al igual que los valores Shapley y aplica todas sus ventajas. La predicción está bastante distribuida entre los valores de las características. Se obtiene explicaciones que comparan la predicción con la predicción promedio.
- Conecta los valores Shapley y LIME.
- Implementación rápida para modelos basados en árboles.
- Reutilización de cálculos. Permite calcular los muchos valores de Shapley necesarios para las interpretaciones del modelo global. Las interpretaciones globales son consistentes con las explicaciones locales, ya que los valores de Shapley son la "unidad atómica" de las interpretaciones globales.

Desventajas

- SHAP es lento. Los métodos SHAP globales requieren calcular los valores de Shapley para muchas instancias.
- KernelSHAP asume independencia de las variables.
- TreeSHAP puede dar valores erróneos de Shapleys para variables no relevantes por el muestreo condicional.
- Las desventajas de los valores de Shapley también se aplican a SHAP: Los valores de Shapley pueden malinterpretarse y se necesita acceso a los datos para calcularlos para nuevos datos.

Capítulo 3

Datos

3.1 Descripción

Los datos utilizados para la realización de este trabajo de fin de grado los proporciona el Departamento de Ingeniería Eléctrica de la Universidad de Valladolid, el GIR Adire coordinado por el profesor D. Óscar Duque Pérez, quienes los obtuvieron de forma experimental. A partir de un motor de inducción y las herramientas de medida necesarias, se recogió el resultado de los datos dañando manualmente el motor de forma controlada y obteniendo así su clasificación.

Para obtener los datos [46] se utiliza un banco de pruebas que consistía en un motor de inducción con las siguientes características: potencia nominal de 1,1 kW, conexión en estrella, tensión nominal de 400 V, velocidad nominal de 1410 rpm y corriente nominal de 2,6 A. La carga del motor era un freno magnético electromagnético Lucas Nülle que también incorpora un medidor de torsión mecánica y velocidad magnética. Utilizando diferentes modelos de inversor y a dos niveles de carga se toman las medidas para el motor. Este se va dañando manualmente lo que nos proporciona la etiqueta.

Contamos con 900 observaciones, siendo la mitad de ellas correspondientes al armónico de la banda lateral superior (USH) y la otra mitad al armónico de la banda lateral inferior (LSH). Por tanto, tendríamos 450 observaciones dobles. Se toman 2048 valores para cada frecuencia correspondientes a los instantes de tiempo de 2048 milisegundos medidos cuando el motor está en funcionamiento.

Los datos corresponden a mediciones en las que el motor estaba alimentado por un inversor. Un inversor es un dispositivo que permite controlar la velocidad rotacional de un motor de inducción de corriente alterna, el cual es alimentado por una frecuencia y voltaje constante, y entrega al motor una frecuencia y voltaje variable, por lo que también son denominados variadores de frecuencia [47]. En este experimento se utilizan tres tipos de variadores de frecuencia:

- Inversor *PowerFlex 40 de Allen Bradley* (AB).

- Inversor *ACS355 de ABB* (ABB).
- Inversor *Altivar 66 de Télémécanique* (TM).

Se lleva a cabo el experimento en cada motor con dos niveles de carga para comprobar su posible influencia en el resultado final.

- Nivel de carga bajo (NC1) 35 % del par nominal.
- Nivel de carga alto (NC2) 60 % del par nominal.

Se utiliza un nivel de carga bajo para evitar que sea vacío y un nivel de carga elevado para comprobar su posible influencia.

Por último, se determina la clasificación del estado del motor según el nivel de perforación del mismo provocado antes de realizar el experimento. El motor se probó en cinco condiciones diferentes: desde el motor sano hasta con la barra del rotor totalmente rota. La rotura de la barra se logró perforando un agujero en la unión de una barra y el anillo final de la jaula. Las clases son ordinales en el rango de R1 a R5:

- Motor en buen estado (R1).
- Motor levemente dañado (R2).
- Motor medianamente dañado (R3).
- Motor severamente dañado (R4).
- Motor gravemente dañado (R5).

En la Tabla 3.1 podemos ver la asignación a clases según la perforación, en diámetro y profundidad, de la barra del rotor.

	Perforación	
Estado	Profundidad	Diámetro
R1	0	0
R2	4.2mm	2.5mm
R3	9.4mm	2.5mm
R4	17mm	2.5mm
R5	17mm	3.5mm

Tabla 3.1: Medidas de la perforación según el estado de deterioro.

Tanto la distribución de clases como de los distintos grupos de inversores y niveles de carga es homogénea, es decir, las muestras están balanceadas. Contamos con el mismo número de observaciones para cada combinación de variador de frecuencia, nivel de carga y estado del motor.

3.2 Procesamiento

En este trabajo, el inversor fue configurado para que el transitorio de arranque tuviera una duración de 10 s con una frecuencia final de 50 Hz. La tensión se capturó para calcular la frecuencia fundamental y la velocidad síncrona a lo largo tiempo. También se midió la velocidad para calcular el deslizamiento del motor. Esta información permite calcular la componente fundamental y las trayectorias de los armónicos relacionados con la falla en el plano tiempo-frecuencia. Aunque el arranque dura 10 s, sólo se analizan los seis segundos centrales para evitar los efectos de borde al al principio y al final del transitorio que afectan a la cuantificación de la gravedad del fallo.

Como ya hemos comentado las observaciones son dobles, para un motor, con un cierto inversor y nivel de carga se obtienen dos observaciones, una para los valores del armónico de la banda lateral superior y otra para los valores de la banda lateral inferior. La primera acción sobre el conjunto de datos será redimensionarlo de tal manera que cada motor tenga una sola observación con todos los valores y los nombres de las columnas apropiados.
Observamos de forma gráfica la estructura de los datos iniciales.

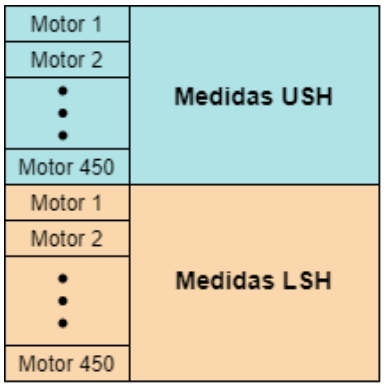

Figura 3.1: Datos iniciales.

Observamos de forma gráfica la estructura de los datos tras la modificación mencionada.

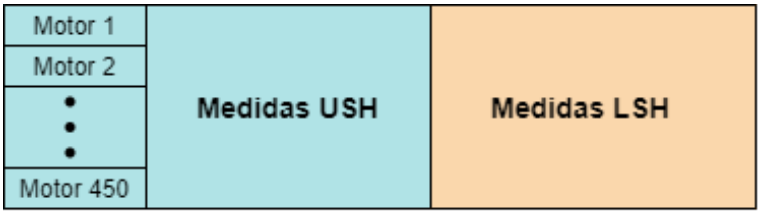

Figura 3.2: Modificación de la estructura de los datos.

Con esto hacemos un cambio de dimensionalidad. Pasamos a tener la mitad de observaciones, 450, pero el doble de atributos. Por tanto, cada observación tendrá 4099 variables:

- Medida de USH: 2048 variables una por cada valor de esta frecuencia en 2048 milisegundos.
- Medida de LSH: 2048 variables una por cada valor de esta frecuencia en 2048 milisegundos.
- Modelo de inversor.
- Nivel de carga.
- Clase.

Esta gran cantidad de variables hace que sea relevante uno de los puntos de interés planteados entre los objetivos de este trabajo. Concretamente el saber si las redes neuronales son capaces de adaptarse, en este problema concreto, a este problema en concreto, a este elevado número de variables seleccionando aquellas más apropiadas para la clasificación o si por el contrario una reducción de la dimensión en el número de variables ayuda a mejorar la tarea de clasificación.

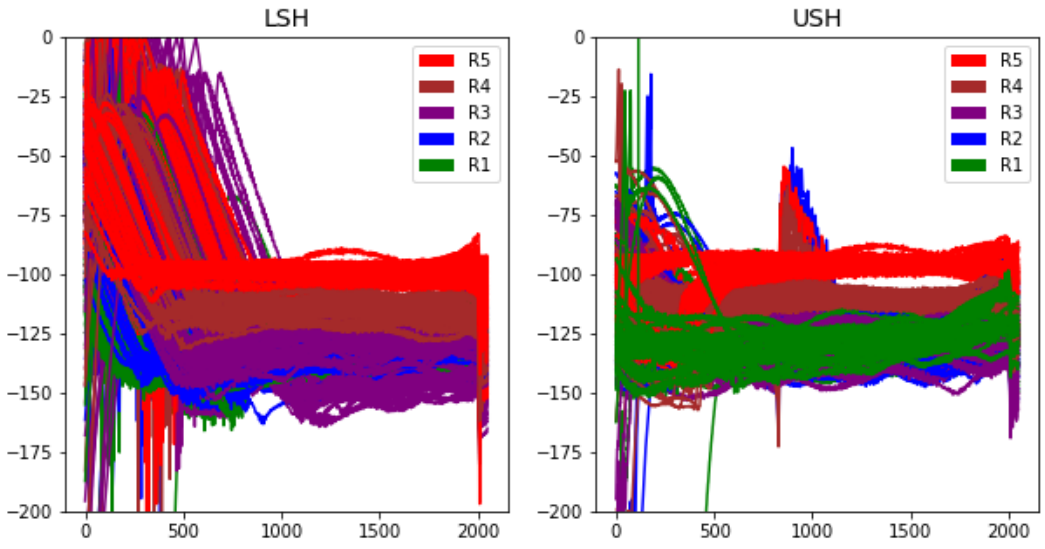

Figura 3.3: Representación de armónicos LSH y USH con los datos originales.

Podemos ver en la Figura 3.3 la representación del conjunto de datos original, las medidas de ambos armónicos frente al tiempo.

3.2.1 Conjunto de datos alternativo

Se utiliza el conjunto de datos propuesto en [3], como conjunto de datos alternativo con reducción de dimensionalidad. En el nuevo conjunto de datos reducido se toman, para cada una de las curvas originales, 7 mediciones construidas a partir de los datos originales para instantes de tiempo entre 250 y 2000 con un intervalo de 250 (250,500, 750, ..., 2000). Cada una de esas 7 mediciones se genera promediando el correspondiente instante con las 5 observaciones inmediatamente anteriores y posteriores al mismo. Constituye una importante reducción de dimensionalidad, es decir, del número de variables empleadas, pasando de 2048 variables por banda a tan solo 7. Lo cual proporcionará una notable reducción del tiempo de ejecución de los algoritmos. Uno de los objetivos de este trabajo es valorar también si se produce una mejora o empeoramiento en los resultados de clasificación con este conjunto de datos reducido.

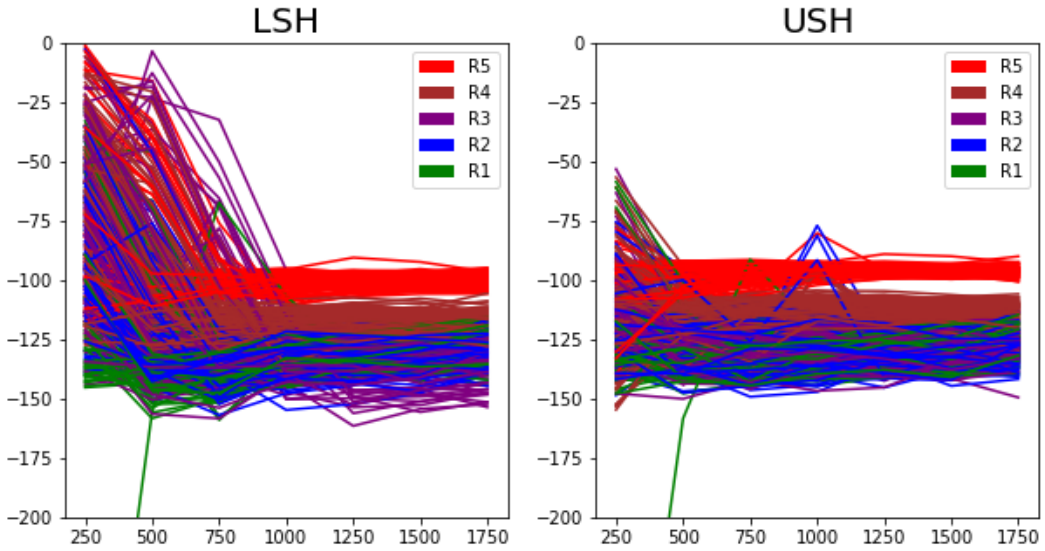

Figura 3.4: Representación de armónicos LSH y USH con los datos alternativos o reducidos.

Podemos ver en la Figura 3.4 la representación del conjunto de datos reducido, las medidas de ambos armónicos frente al tiempo. Respecto al original se observan lineas mucho más rectas porque hay tan solo 7 mediciones por individuo en cada banda.

3.2.2 Subconjuntos de datos

A continuación, describimos los subconjuntos con los que vamos a trabajar en la elaboración de modelos.

- Conjunto agrupado: se utilizarán todas las variables. Las variables categóricas correspondientes al modelo de inversor y nivel de carga se desdoblarán como variables *dummy*

por especificaciones del algoritmo.

- Subconjuntos modelo de inversor-nivel de carga: se crearan seis subconjuntos, uno para cada combinación modelo de inversor-nivel de carga.

3.3 Terminología

Para aclarar las ideas a continuación explicaremos como nos vamos a referir a los siguientes términos de forma sinónima:

- Conjunto de datos original, datos completos o conjunto completo: hace referencia al conjunto con todos los instantes de tiempo midiendo los armónicos de la onda en los 2048 milisegundos segundos.

- Conjunto de datos alternativo, datos reducidos o conjunto reducido: es el conjunto de datos con solo 10 medidas del armónico por banda. Los datos con la reducción de dimensión propuestos en [3].

- Conjunto con variables dummy o conjunto agrupado: representará el conjunto con todos los modelos de inversor y niveles de carga representando esta información en 5 variables dicotómicas. Podrá ser tanto de los datos completos como los reducidos. También nos referimos a este conjunto como datos agrupados ya que solo hay un grupo.

- Por grupos: nos referiremos a los conjuntos separados por grupos según la combinación modelo de inversor y nivel de carga, ya sea para los datos completos o reducidos. Esto dará lugar a seis conjuntos.

Capítulo 4

Análisis y resultados

4.1 Experimento realizado

En primer lugar, se realiza una partición en dos subconjuntos, uno para entrenamiento, *train*, y otro para prueba, *test*. El conjunto de entrenamiento se utilizará para entrenar el algoritmo y seleccionar los hiperparámetros al mismo tiempo. Como queremos evitar el sobreajuste también en la selección de hiperparámetros, realizaremos otra división dentro del conjunto de entrenamiento en 5 carpetas, de esta manera, buscaremos el mejor modelo con validación cruzada *5-fold*. Posteriormente, se entrena el modelo seleccionado con todos los datos de entrenamiento. Los datos de *test* se utilizan para estimar el error solo con el mejor modelo ya entrenado.

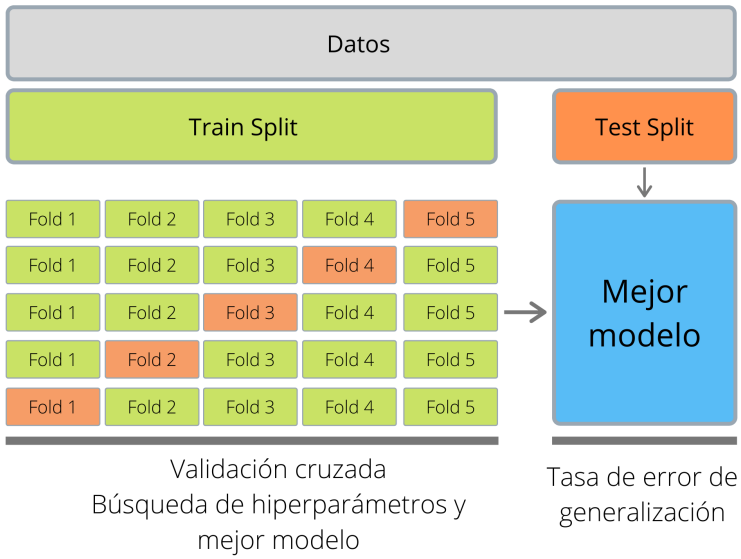

Figura 4.1: Esquema de desarrollo con validación cruzada 5-fold. Imagen de [3]

Con el objetivo de obtener un estimador de error más preciso y con menos variabilidad,

intentando evitar la dependencia de la partición inicial, se repite el proceso previamente descrito. Obtenemos entonces la estimación del error como el promedio de los errores de generalización (e_S) en cada repetición. En este TFG se realizará validación cruzada con 5 particiones y 20 repeticiones. Se toman 5 particiones en vez de 10 por la escasez del número de observaciones.

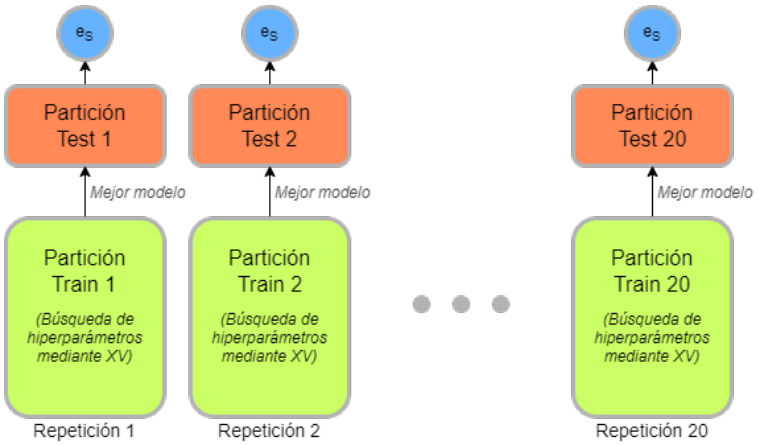

Figura 4.2: Validación cruzada + Test repetido. Imagen de [3]

Se realiza el experimento descrito para los siguientes conjuntos de datos con sus combinaciones:

- Datos completos y datos reducidos.
- Banda: LSH, USH y ambas.
- Nivel de carga: NC1 o NC2.
- Modelo de inversor: AB, ABB y TM

Se entrena un clasificador único para los tres tipos de inversores y los dos niveles de carga, alimentando esta información en forma de variables dicotómicas. Las combinaciones de todas estas posibilidades se tratan como conjuntos de datos separados, se realiza el experimento con estos obteniendo así el mejor modelo para los datos de entrenamiento, con su correspondiente búsqueda de hiperparámetros, realizando validación cruzada con 5 particiones. Empleando el mejor conjunto de hiperparámetros, en cada una de las 20 repeticiones se calculará la tasa de error el conjunto reservado, de prueba, y estimaremos el error verdadero o de generalización como el promedio del error obtenido en estas 20 repeticiones.

4.2 Tasas de error

A continuación se muestran las tablas resumen con las estimaciones del error de generalización obtenidas en cada grupo.

Para el conjunto completo, empleando los registros de los 2048 milisegundos, sin separar grupos por modelo de inversor y nivel de carga obtenemos los resultados mostrados en la Tabla 4.1.

Datos completos				
Modelo de inversor	**Nivel de carga**	**Banda**		
		Ambas	LSH	USH
-	-	0.320	0.313	0.358
-	NC1	0.369	0.319	0.362
-	NC2	0.272	0.307	0.354
AB	-	0.347	0.352	0.401
ABB	-	0.324	0.316	0.361
TM	-	0.29	0.269	0.315
AB	NC1	0.355	0.345	0.347
ABB	NC1	0.426	0.362	0.433
TM	NC1	0.331	0.247	0.307
AB	NC2	0.341	0.358	0.457
ABB	NC2	0.233	0.267	0.285
TM	NC2	0.239	0.291	0.325

Tabla 4.1: Estimación del error con datos completos agrupados.

Se muestra en la primera línea el error global estimado del modelo. En las líneas posteriores se muestra el error estimado para los grupos indicados en las columnas de la izquierda, sobre el mismo modelo global. Podemos observar como los errores producidos por el modelo que solo utiliza la información de la banda LSH son menores que usando USH o ambas bandas. También podemos destacar como el modelo completo con la información de ambas bandas funciona especialmente bien para el nivel de carga 2, destacando los inversores ABB y TM que son los que mejor comportamiento tienen bajo este modelo.

Cabe mencionar que el promedio de los errores no resulta el valor global ya que el reparto de las observaciones es estratificado en cuanto a la clase (R1,R2,...) no así en cuanto al nivel de carga y modelo de inversor.

Los resultados para el conjunto de datos reducido, solo 10 variables, con la misma agrupación se muestran en la Tabla 4.2.

Datos reducidos				
Modelo de inversor	**Nivel de carga**	**Banda**		
		Ambas	LSH	USH
-	-	0.229	0.257	0.298
-	NC1	0.239	0.278	0.331
-	NC2	0.221	0.233	0.268
AB	-	0.272	0.291	0.349
ABB	-	0.212	0.272	0.267
TM	-	0.205	0.206	0.276
AB	NC1	0.295	0.343	0.369
ABB	NC1	0.208	0.289	0.312
TM	NC1	0.212	0.198	0.311
AB	NC2	0.248	0.234	0.331
ABB	NC2	0.216	0.252	0.225
TM	NC2	0.239	0.215	0.239

Tabla 4.2: Estimación del error con datos reducidos agrupados.

En este caso, en la Tabla 4.2 vemos como el modelo que utiliza la información de ambas bandas tiene una estimación del error más baja. Comparándolo con la versión anterior con los datos completos vemos como ahora se reduce el error estimado. Esto nos lleva a pensar que el conjunto de datos reducido se puede comportar igual o mejor en la tarea de clasificación que el conjunto de datos completo, ahorrando así una gran capacidad de cómputo y almacenamiento. Igual que en el caso anterior, en nivel de carga 2 parece resultar más certero en cuanto a la predicción de nuevas observaciones.

Ahora analizaremos los resultados obtenidos en la estimación del error cuando tratamos los grupos (modelo de inversor x banda) como conjuntos de datos independientes y elaboramos un modelo para cada uno de ellos. El objetivo es encontrar modelos específicos para cada inversor, puesto que teóricamente tienen diferente comportamiento, y poder comprobar la hipótesis de simetría de las bandas, la cual nos indicaría que las bandas LSH y USH, por ser armónicos de la corriente de entrada, serían simétricas respecto la onda principal y podrían contener información redundante.

Para los datos completos obtenemos los resultados mostrados en la Tabla 4.3.

Datos completos				
Modelo de inversor	**Nivel de carga**	**Banda**		
		Ambas	LSH	USH
AB	NC1	0.29	0.276	0.337
ABB	NC1	0.313	0.273	0.3
TM	NC1	0.247	0.18	0.223
AB	NC2	0.283	0.229	0.373
ABB	NC2	0.25	0.24	0.28
TM	NC2	0.147	0.267	0.243

Tabla 4.3: Estimación del error con datos completos separados por grupos.

En este caso, para la Tabla 4.3 se obtiene y se entrena un modelo para cada grupo, se estima el error de las observaciones de solo ese grupo. Podemos observar como, por tónica general, los modelos que utilizan solo la información de la banda LSH obtienen menor estimación del error. Esto podría ser beneficioso ya que podríamos reducir el las variables recogidas en el conjunto de datos a la mitad. Sin embargo, encontramos un valor notablemente bajo en el grupo con inversor TM y nivel de carga alto, con ambas bandas, frente a los demás grupos y frente a ese grupo utilizando solo alguna de las dos bandas.

Observamos en la Tabla 4.4 los resultados por grupos con el conjunto de datos reducido.

Datos reducidos				
Modelo de inversor	**Nivel de carga**	**Banda**		
		Ambas	LSH	USH
AB	NC1	0.27	0.26	0.347
ABB	NC1	0.229	0.227	0.303
TM	NC1	0.203	0.219	0.229
AB	NC2	0.207	0.166	0.293
ABB	NC2	0.17	0.247	0.213
TM	NC2	0.256	0.189	0.243

Tabla 4.4: Estimación del error con datos reducidos separados por grupos.

Una vez más, en la Tabla 4.4 vemos como los resultados de las bandas LSH y ambas son notablemente mejores que los obtenidos únicamente con la banda del armónico superior. Hay que señalar que, según los expertos del Departamento de Ingeniería Eléctrica, el inversor TM suele presentar un comportamiento diferente al resto lo que podría explicar la diferencia de resultados que se obtienen para ese caso. Debemos comparar si estas diferencias son significativas para poder elegir la banda LSH y reducir así la información innecesaria en el modelo.

Respecto a los resultados de los grupos con los datos completos vemos como en casi todos los casos el error obtenido es menor con los datos reducidos. Debemos comprobar si estas

diferencias son significativas para poder concluir que los datos reducidos obtienen resultados significativamente mejores. Esto supondría un gran avance ya que pasaríamos de tener 2048 milisegundos medidos a tener 10 variables, lo cual se vería reflejado en un gran aumento en el rendimiento computacional del modelo, tanto en la etapa de entrenamiento como en la etapa de clasificación.

4.3 Análisis de resultados

Para realizar las comparaciones propuestas en la sección anterior utilizaremos técnicas de Análisis de la Varianza (ANOVA). De esta manera, comprobaremos si las diferencias apreciadas a simple vista son diferencias significativas.

Este análisis se realizará únicamente con los resultados obtenidos de los datos por grupos, obviando los modelos completos para todos los inversores y niveles de carga. Tomamos esta decisión en base al análisis previo realizado todos los resultados que podemos ver en la Tabla 4.5. En primer lugar, interesa comprobar si la separación de modelos en grupos según su nivel de carga y modelo de inversor proporciona mejores resultados en la clasificación. En la Tabla 4.5 podemos ver el análisis realizado para contrastar esta hipótesis.

	Df	**Dum Sq**	**Mean Sq**	**F value**	**Pr(>F)**
Por.grupos	1	0.01013	0.010133	3.98	0.0529 .
Residuales	40	0.10185	0.002546		

Tabla 4.5: Anova de un factor: Totales vs. Por grupos.

Primaremos los modelos por grupos por los siguientes motivos:

- Resultados ligeramente mejores. Los resultados del error para los datos por grupos son menores en casi todos los casos.
- Diferencias significativas. Aunque no a nivel 5 %, sí encontramos diferencias significativas a nivel 6 %.
- Tiempo de cómputo. Aunque se deban almacenar más modelos y realizar más procesos de entrenamiento, uno por cada modelo, el tiempo de cómputo se ve reducido frente al entrenamiento de un único modelo con todas las observaciones.
- Especialización. El comportamiento técnico de los diferentes modelos de inversor se podría ver reflejar y capturar mejor en un modelo específico para cada uno de ellos.
- Interpretabilidad. Los modelos con las variables *dummy* producirían problemas de interpretabilidad, al menos en cuanto a esas variables se refiere.

A pesar de todo ello, destacamos el buen comportamiento del modelo con los datos agrupados, es decir, sin separación por grupos debido al modelo de inversor y nivel de carga, ya que un único modelo puede obtener tasas de error similares a las de los conjuntos por grupos. Esto nos podría hacer pensar en un único modelo capaz de clasificar cualquier tipo de inversor y de nivel de carga correctamente, es decir, un modelo universal válido para cualquier máquina, lo cual podría suponer una gran ventaja en la práctica.

A continuación, una vez descartados los modelos agrupados, interesa comprobar la utilidad de los datos completos o si por el contrario supone alguna ventaja utilizar los datos reducidos. A partir de ahora solo se utilizarán los datos por grupos para realizar los siguientes análisis.

	Df	**Dum Sq**	**Mean Sq**	**F value**	**Pr(>F)**
Completos	1	0.00722	0.007225	2.928	0.0962 .
Residuales	34	0.08390	0.002468		

Tabla 4.6: Anova de un factor: Completos vs. Reducidos.

Podemos apreciar diferencias significativas a nivel 10 % en cuanto a la estimación del error entre los conjuntos de datos completos y reducidos. Esto favorece nuestra hipótesis de que los datos reducidos son capaces de explicar la variable de respuesta. Por ello, decidimos a partir de ahora trabajar con este conjunto de datos, favoreciendo así una notable disminución del tiempo de cómputo.

Esto indica también que las redes neuronales, en este problema, se benefician de la reducción de dimensión que se ha hecho al considerar los datos alternativos definidos en la sección 3.2.1. Esta observación resuelve uno de los objetivos del trabajo que era precisamente valorar si las redes neuronales mejoraban o no sus resultados mediante este tipo de reducción de dimensión.

El nivel del factor inversor de una máquina es algo inherente a la misma. Sin embargo, el nivel de carga no, es algo variable y nosotros podríamos seleccionar el nivel de carga óptimo a la hora de medir para obtener menores tasas de fallo y mayor probabilidad de seleccionar correctamente la clase. Observamos los datos y vemos que el nivel de carga dos o alto parece tener menores tasas de error. Como venimos haciendo con las hipótesis anteriores, comprobamos si esta diferencia es significativa con un test de análisis de la varianza. Del mismo modo, encontramos la hipótesis que la información en los armónicos de las bandas es simétrica, por lo cual no necesitaríamos incluir ambas al modelo, podría valer con una de ellas. No interesa contrastar esta hipótesis al tiempo que comprobamos cual es la que mejores resultados obtiene.

Se comprueba ahora realizando un ANOVA multifactorial si, para los datos reducidos y por grupos, son influyentes los factores: inversor, nivel de carga y banda. Los resultados se muestran en la Tabla 4.7.

	Df	Dum Sq	Mean Sq	F value	Pr(>F)
Banda	2	0.010499	0.005249	4.142	0.106
Nivel de carga	1	0.005101	0.005101	4.025	0.115
Inversor	2	0.003768	0.001884	1.487	0.329
Banda:Nivel de carga	2	0.000312	0.000156	0.123	0.887
Banda:Inversor	4	0.006653	0.001663	1.312	0.399
Nivel de carga:Inversor	2	0.005321	0.002661	2.099	0.238
Residuales	4	0.005069	0.001267		

Tabla 4.7: Anova de tres factores con interacciones de orden 2. Banda-Nivel de carga-Modelo de inversor.

Con el objetivo de observar con más claridad este análisis de la varianza se realiza de nuevo obviando las interacciones e incluyendo solo los efectos principales. Se realiza un proceso de eliminación de las interacciones de forma individual, una a una, comprobando que siguen sin ser significativas, se omiten las tablas intermedias.

	Df	Dum Sq	Mean Sq	F value	Pr(>F)
Banda	2	0.010499	0.005249	3.629	0.0585 .
Nivel de carga	1	0.005101	0.005101	3.527	0.0849 .
Inversor	2	0.003768	0.001884	1.303	0.3076
Residuales	12	0.017356	0.001446		

Tabla 4.8: Anova de tres factores sin interacción. Banda-Nivel de carga-Modelo de inversor.

Podemos concluir de forma clara que la banda y el nivel de carga resultan significativos a nivel 10 %. Sin embargo, el modelo de inversor no lo es, esto nos dice que para cualquier modelo de inversor podemos ajustar un modelo estadístico que proporcione resultados sin diferencias significativas, en cuanto a la tasa de error se refiere, frente a otros inversores. Esto es un gran avance ya que el diferente comportamiento de estos dificultaba en algunos casos su clasificación. Hemos podido comprobar como con esta técnica los resultados podrían ser igual de buenos sea cual sea el modelo de inversor que tenga la máquina.

Al no resultar significativa la interacción entre banda y nivel de carga, podemos elegir ambos individualmente de forma que se optimice el resultado. A la vista de los errores obtenidos lo óptimo parece ser elegir el nivel de carga alto (NC2) y la banda inferior (LSH) o ambas bandas en algún otro caso. Esto se especifica en la siguiente sección 4.4 y con los contrastes mostrados a continuación.

Para comprobar el buen funcionamiento del análisis realizado en la Tabla 4.8 mostramos el gráfico de residuales.

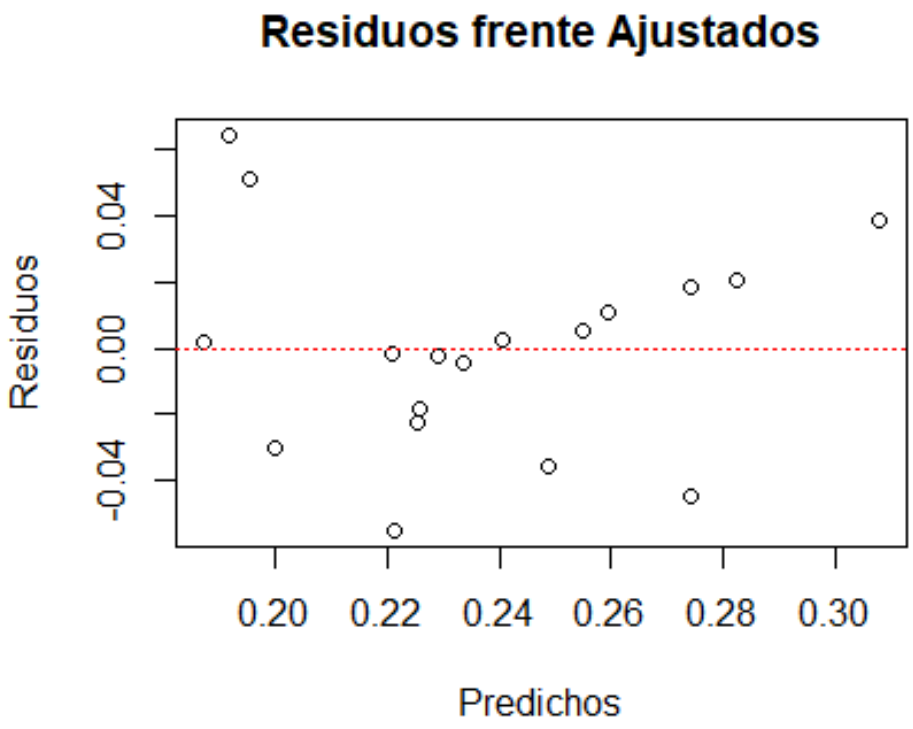

Figura 4.3: Gráfico de residuales del ANOVA de la Tabla 4.8.

Podemos observar como se distribuyen en torno al cero y no parecen tener ningún patrón. Podemos decir que siguen una distribución aleatoria luego el análisis realizado es correcto.

También, realizamos los test *post-hoc* para comprobar en favor de qué covariable encontramos las diferencias significativas aunque podemos sospecharlo a la vista de los resultados obtenidos en la sección 4.2.

Se realiza el test de Duncan a nivel 0.05.

Banda	**Count**	**LS Mean**	**LS Sigma**	**Homogeneous Groups**	
lsh	6	0.218	0.0155259	X	
ambas	6	0.2225	0.0155259	X	
ush	6	0.271333	0.0155259		X

Tabla 4.9: Parte 1 test *post-hoc* para la banda.

Vemos dos grupos separados: las bandas LSH y ambas se encuentran en el mismo grupo, la banda USH se encuentra en otro grupo diferente.

Contrast	Sig.	Difference
ambas - lsh		0.0045
ambas - ush	*	-0.0488333
lsh - ush	*	-0.0533333

Tabla 4.10: Parte 2 test *post-hoc* para la banda.

Podemos observar en la Tabla 4.10 como existen diferencias significativas entre las bandas USH y LSH, entre USH y ambas, pero no hay diferencia significativa entre LSH y ambas bandas. Siendo USH la que peores resultados arroja. Sin diferencia significativa entre LSH y ambas bandas intentaremos priorizar LSH por simplicidad.

Se realiza el test de Duncan a nivel 0.1.

Nivel.de.Carga	Count	LS Mean	LS Sigma	Homogeneous Groups	
NC2	9	0.220444	0.0126769	X	
NC1	9	0.254111	0.0126769		X

Tabla 4.11: Parte 1 test *post-hoc* para el nivel de carga.

Vemos como los dos niveles de carga se encuntran en grupos separados.

Contrast	Sig.	Difference
NC1 - NC2	*	0.0336667

Tabla 4.12: Parte 2 test *post-hoc* para el nivel de carga.

Podemos observar en la Tabla 4.12 que existen diferencias significativas entre el nivel de carga alto y el bajo. Siendo el nivel de carga alto, NC2, el que mejores resultados obtiene, lo cual es coherente con la literatura existente [48].

4.4 Clasificación según modelo de inversor

El objetivo ahora es seleccionar que modelo y que datos incorporaremos a este para que el error cometido sea el menor posible. Buscamos para cada inversor cual serían sus condiciones óptimas tras el análisis realizado en la sección anterior.

4.4.1 Inversor AB

Encontramos su óptimo utilizando los datos reducidos por grupos con nivel de carga alto y utilizando únicamente la banda LSH.

$$\hat{e_g} = 0.166$$

Mostramos la matriz de confusión para el entrenamiento de este modelo promediada en las 20 repeticiones. Se utiliza en cada repetición 60 observaciones para entrenamiento y 15 para prueba.

		Predicho				
		R1	R2	R3	R4	R5
Real	R1	2.0	0.05	0.9	0.05	0.0
	R2	0.05	2.65	0.3	0.0	0.0
	R3	0.6	0.4	2.0	0.0	0.0
	R4	0.15	0.0	0.0	2.85	0.0
	R5	0.0	0.0	0.0	0.0	3.0

Tabla 4.13: Matriz de confusión Inversor AB.

Resulta más interesante observar la matriz anterior condicionada por filas para estimar la probabilidad de clasificar una instancia de clase X en la columna Y. Se muestra la matriz redondeada a 3 decimales, luego puede que sus valores no sumen exactamente 1.

		Predicho				
		R1	R2	R3	R4	R5
Real	R1	0.667	0.017	0.3	0.017	0.0
	R2	0.017	0.883	0.1	0.0	0.0
	R3	0.2	0.133	0.667	0.0	0.0
	R4	0.05	0.0	0.0	0.95	0.0
	R5	0.0	0.0	0.0	0.0	1.0

Tabla 4.14: Matriz de confusión Inversor AB condicionada por filas.

Podemos apreciar una diferenciación casi perfecta el las clases R4 y R5, las que constituyen más daños en el motor. Si agrupamos las clases R1,R2 y R3 frente a estas, haciendo diferenciación entre más dañados y menos dañados obtendríamos una tasa de error muy baja, a pesar de haber utilizado una función de pérdida categórica sin efectos proporcionales.

El caso de motor con más daños R5 predice correctamente el 100 % de las observaciones de prueba. Por la contra, el motor con menos daños R1 y con daños intermedios R3 tan solo aciertan el 66.7 % de las observaciones de prueba.

Interpretabilidad del modelo
En primer lugar mostramos el gráfico obtenido de la importancia de variables.

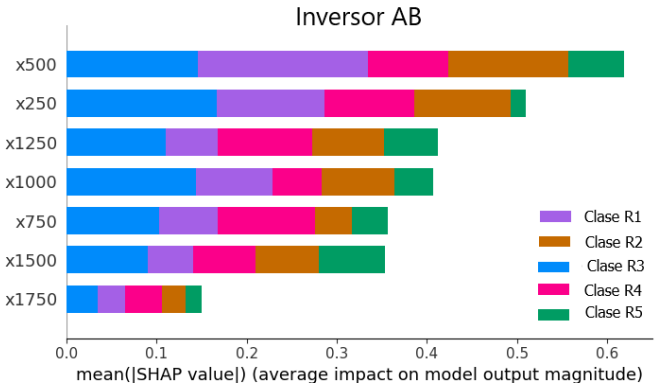

Figura 4.4: Importancia de variables en SHAP para AB.

Podemos ver como para las clases de motores con menos daños importan más los instantes iniciales x500 y x250. Para los motores con más daños, importan más los instantes intermedios x1000, x750 y sobre todo respecto de las clases R1 y R2 importan mas los instantes finales x1500.

A continuación, mostramos para cada clase los gráficos obtenidos con SHAP. Debemos fijarnos en los siguientes puntos para interpretar estos gráficos:

- Situación del punto: Si el punto está a la derecha, SHAP positivo, favorece la clasificación a la clase que estamos estudiando en dicho gráfico. Si está a la izquierda, SHAP negativo, favorece a la no clasificación en esa clase, podría ser cualquiera de las otras.
- Color: Un punto de color rojizo indica que toma valores altos en esa variable. Por el contrario, un color azulado indica que toma valores bajos en esa variable.
- Distribución: se trata de ver si los puntos se agrupan segun colores a izquierda o derecha en cada variable.
- Orden: las variables se muestran en orden de ïmportancia"según los valores SHAP que toman los puntos.

La interpretación de estos gráficos se realiza de forma local variando levemente los valores de la observación y viendo como cambia así su predicción.
En la mayoría de estos gráficos se toman valores bastante extremos que podríamos considerar incluso *outliers*, se toma la decisión de recortar en torno al valor SHAP -0.5, +0.5 para poder apreciar mejor la influencia de las variables y extraer conclusiones a cerca de su comportamiento.
Se muestran a continuación los gráficos sobre valores SHAP recortados, sin *outiers*.

- R1.

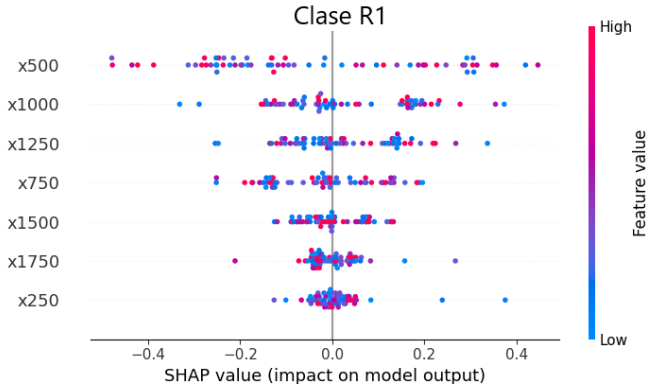

Figura 4.5: SHAP recortado para clase R1 en inversores AB.

La variable x500, que es la que más peso tiene proporciona una información de que los valores altos en esa variable, puntos rojos, se sitúan en los valores SHAP positivos luego favorecen la clasificación de esa variable. También se observan más puntos azules a la izquierda, SHAP negativos, por lo que valores bajos en esa variable favorecen la no clasificación en R1.

Valores bajos de la variable x1000 favorecen a la clasificación como no R1.

- R2.

Figura 4.6: SHAP recortado para clase R2 en inversores AB.

Una vez más la variable x500 toma gran importancia. En este caso se observa perfectamente como valores bajos de esta variable motivan a la clasificación como no R2 y valores altos de estas motivan a la clasificación R2.

La variable x250 tiene una distribución similar a la anterior. Podríamos concluir que valores bajos en los primeros instantes de tiempo motivarían a una clasificación distinta a R2 y valores altos a una clasificación R2.

- R3.

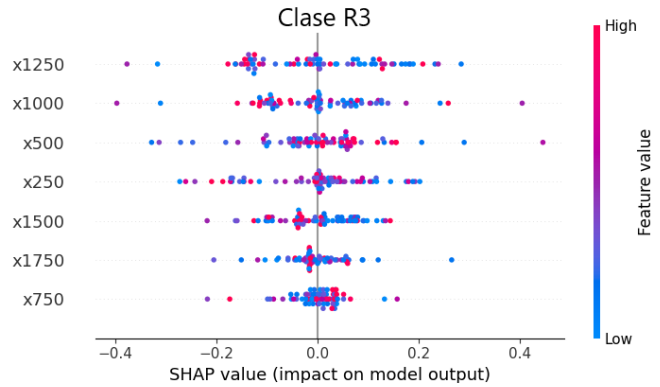

Figura 4.7: SHAP recortado para clase R3 en inversores AB.

Para la clasificación R3, toman más importancia los instantes centrales. Valores bajos en las variables x1250 y x1000 favorecen a la clasificación de la instancia como R3.

- R4.

Figura 4.8: SHAP recortado para clase R4 en inversores AB.

En este caso empiezan a tomar importancia los instantes de x1750 pero también los de x500, tiene en cuenta momentos del tiempo mayores para la clasificación de esta clase.

- R5.

Figura 4.9: SHAP recortado para clase R5 en inversores AB.

Podemos ver en el gráfico como es la que mejor se diferencia, que la interpretación sea tan clara puede estar relacionado con que la clase es fácilmente separable en esa variable. Valores bajos en las variables x1500 y x 1750 favorecen a la clasificación no R5, por el contrario valores altos en estas variables favorecen a la clasificación R5.

Con un enfoque más general vemos como los individuos de esta clase deben tener valores generalmente más altos en todas la variables.

Podemos observar como valores bajos en los primeros instantes de tiempo favorecen clasificaciones de motores con menos daños. Valores altos en los primeros instantes de tiempo favorecen clasificaciones de motores con más daños. Con los instantes superiores sucede justo al contrario, valores altos indican clasificaciones en motores con más daños y valores más bajos favorecerían clasificar en motores con menos daños.

En los motores R1 y R2 importaron más los instantes iniciales, en motores R3 los instantes intermedios y en motores R4 y R5 los instantes finales.

4.4.2 Inversor ABB

Encontramos su óptimo utilizando los datos reducidos por grupos con nivel de carga alto y utilizando la información de ambas bandas.

$$\hat{e_g} = 0.17$$

Mostramos la matriz de confusión para este modelo promediada en las 20 repeticiones. Se utiliza en cada repetición 60 observaciones para entrenamiento y 15 para prueba.

		Predicho				
		R1	R2	R3	R4	R5
Real	R1	1.9	1.1	0.0	0.0	0.0
	R2	1.15	1.65	0.15	0.05	0.0
	R3	0.0	0.0	3.0	0.0	0.0
	R4	0.0	0.05	0.0	2.9	0.05
	R5	0.0	0.0	0.0	0.0	3.0

Tabla 4.15: Matriz de confusión Inversor ABB.

Resulta más interesante observar la matriz anterior condicionada por filas para estimar la probabilidad de clasificar una instancia de clase X en la columna Y. Se muestra la matriz redondeada a 3 decimales, luego puede que sus valores no sumen exactamente 1.

		Predicho				
		R1	R2	R3	R4	R5
Real	R1	0.633	0.367	0.0	0.0	0.0
	R2	0.383	0.55	0.05	0.017	0.0
	R3	0.0	0.0	1.0	0.0	0.0
	R4	0.0	0.0	0.017	0.967	0.017
	R5	0.0	0.0	0.0	0.0	1.0

Tabla 4.16: Matriz de confusión Inversor ABB condicionada por filas.

Al igual que en el caso anterior vemos como el modelo es capaz de separar entre los motores con menos daños, R1 a R3, y los motores con más daños, R4 y R5, casi a la perfección. Sin embargo, a penas es capaz de decidir entre las clases R1 y R2 ya que las confunde en muchas ocasiones.
También destacamos el buen comportamiento en R3, R4 y R5 en las cuales predice correctamente casi todas las observaciones. Sin duda nos encontramos ante un modelo muy preciso y que ha funcionado bien ante los datos del laboratorio. Habría que comprobar su funcionamiento en la práctica con datos reales donde hay mucho ruido.

Interpretabilidad del modelo
A continuación se mostrarán e interpretarán los gráficos SHAP. Al igual que en la sección anterior, encontramos *outliers* o puntos con valores SHAP muy extremos que nos dificultan la interpretación del gráfico. Por ello, mostramos directamente los gráficos recortados en el entorno -0.5,+0.5 para mejorar la interpretabilidad.

- R1.

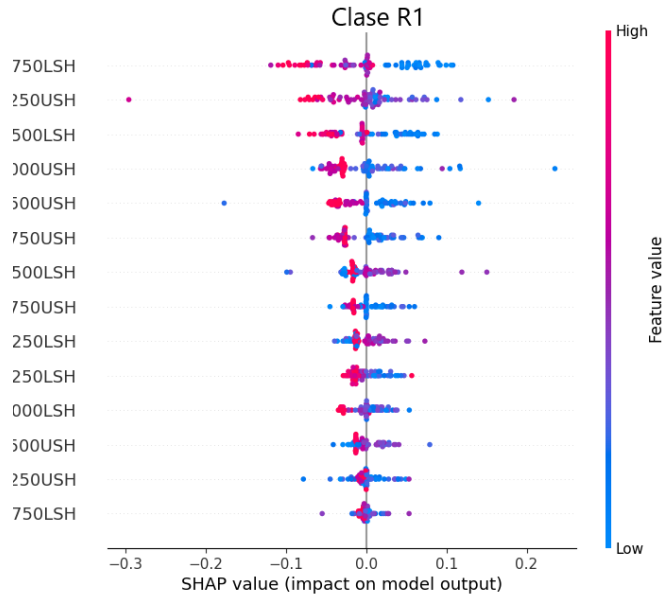

Figura 4.10: SHAP recortado para clase R1 en inversores ABB.

Podemos ver como valores bajos en las variables más importantes favorecen a la clasificación como R1. Por el contrario, valores altos de estas favorecen a la clasificación como no R1. Resultan más importantes los instantes iniciales.

- R2.

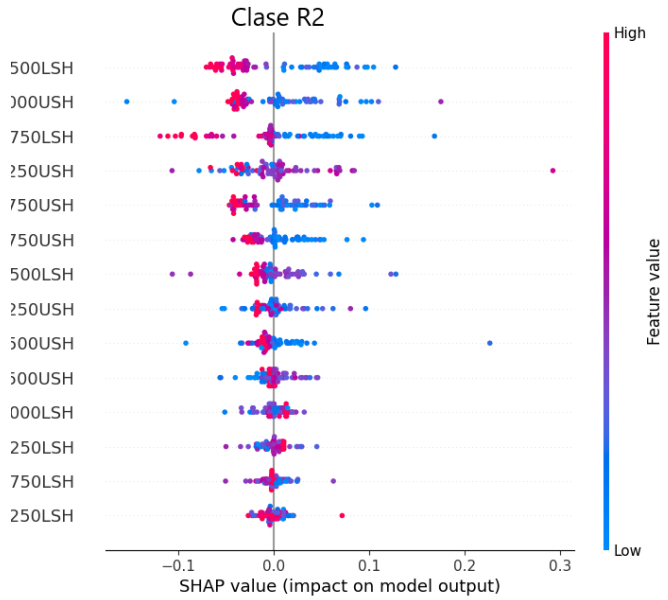

Figura 4.11: SHAP recortado para clase R2 en inversores ABB.

De forma similar a como sucedía con R1 pero en esta ocasión dando más pero al instante x500, los valores bajos de las variables más representativas de esta clase favorecen a la clasificación como R2. Los valores altos lo contrario, motivarían no clasificarlo como R2.

- R3.

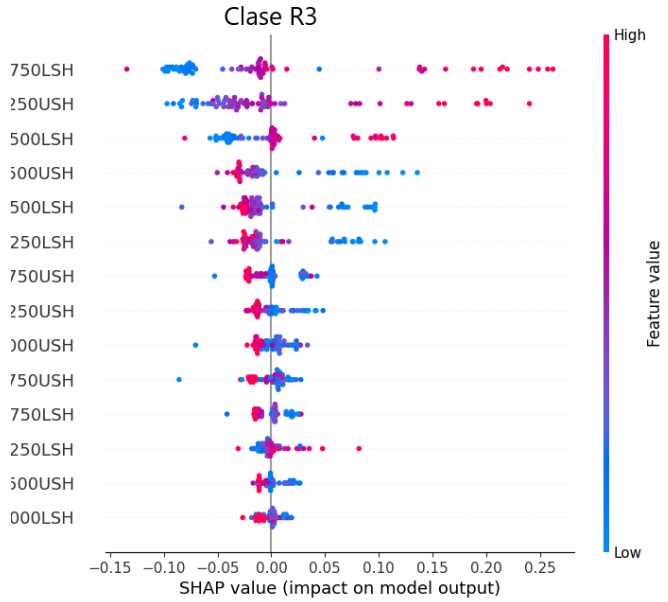

Figura 4.12: SHAP recortado para clase R3 en inversores ABB.

Se observa el cambio notablemente ya que ahora los instantes iniciales toman valores valores extremos favoreciendo la clasificación como R3, incluso hay observaciones con valores altos que tienen SHAP negativo y valores bajos con la motivación opuesta.

- R4.

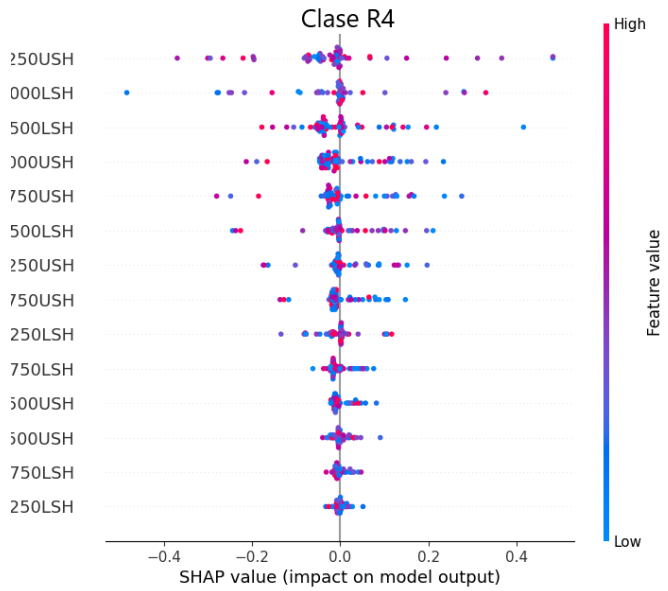

Figura 4.13: SHAP recortado para clase R4 en inversores ABB.

Los puntos están demasiado agrupados en la zona central y se dificulta mucho la interpretación de clasificación en esta clase.

- R5.

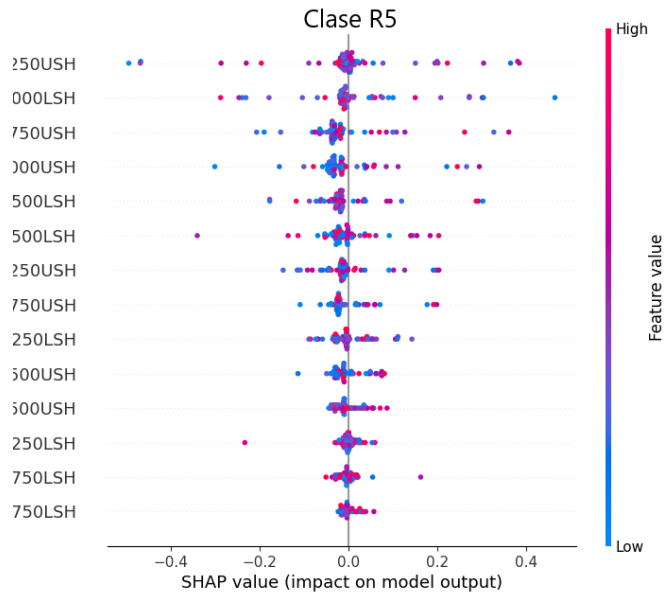

Figura 4.14: SHAP recortado para clase R5 en inversores ABB.

En este caso la interpretación también es compleja pero parece que los valores bajos (azules) se sitúan más a la izquierda luego favorecerían a la clasificación como no R5 si los instantes superiores en el tiempo toman valores bajos.

4.4.3 Inversor TM

Encontramos su óptimo utilizando los datos completos por grupos con nivel de carga alto y utilizando la información de ambas bandas. Sin embargo, hemos descartado estos porque la diferencia no resultaba significativa y supone una reducción notable del tiempo de cómputo. Por ello, utilizaremos su segundo mejor resultado que encontramos con los datos reducidos por grupos a nivel de carga dos con la información de la banda inferior.

$$\hat{e_g} = 0.189$$

Vemos las matrices de confusión para observar con detalle la distribución de la tasa de error en las diferentes clases. Una vez más se muestra la tabla promediada a las 20 repeticiones, en las que se utilizan 60 observaciones para entrenamiento y 15 para prueba.

		Predicho				
		R1	R2	R3	R4	R5
Real	R1	2.85	0.0	0.15	0.0	0.0
	R2	0.05	2.2	0.75	0.0	0.0
	R3	0.1	1.55	1.3	0.05	0.0
	R4	0.0	0.0	0.05	2.8	0.15
	R5	0.0	0.0	0.0	0.0	3.0

Tabla 4.17: Matriz de confusión Inversor TM.

Resulta más interesante observar la matriz anterior condicionada por filas para estimar la probabilidad de clasificar una instancia de clase X en la columna Y. Se muestra la matriz redondeada a 3 decimales, luego puede que sus valores no sumen exactamente 1.

		Predicho				
		R1	R2	R3	R4	R5
Real	R1	0.95	0.0	0.05	0.0	0.0
	R2	0.017	0.733	0.25	0.0	0.0
	R3	0.033	0.517	0.433	0.017	0.0
	R4	0.0	0.0	0.017	0.933	0.05
	R5	0.0	0.0	0.0	0.0	1.0

Tabla 4.18: Matriz de confusión Inversor TM condicionada por filas.

En esta ocasión sorprende la buena clasificación obtenida en R1 y la mala clasificación obtenida en R3 pues la mayoría de estos los asigna la clase R2. Se vuelve a observar un gran desempeño en la clasificación de los motores con más daños R4 y R5 logrando alto grado de acierto.

Interpretabilidad del modelo

En primer lugar podemos mostrar el gráfico obtenido de la importancia de variables. Aunque este carece de una interpretación directa sobre cómo se hacen las predicciones de nuevas observaciones.

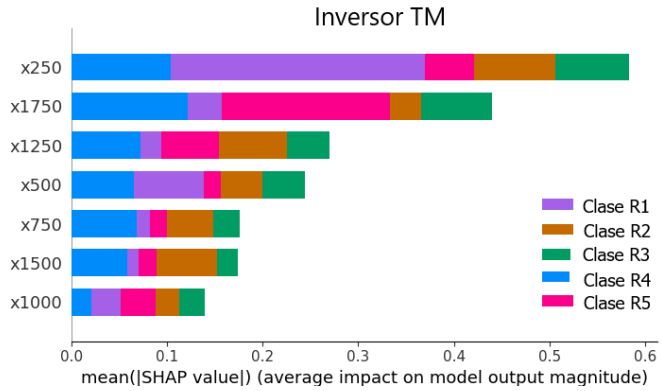

Figura 4.15: Importancia de variables en SHAP para TM.

Podemos apreciar como los instantes iniciales son más influyentes en la clasificación de motores con menos daños y los instantes finales para los motores con más daños. En este caso, mostramos los gráficos sin recortar ya que la interpretación en inmediata.

- R1.

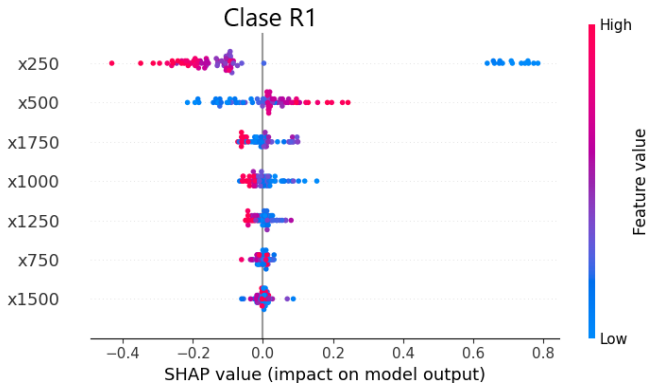

Figura 4.16: SHAP para clase R1 en inversores TM.

Valores altos de la variable x250 favorecen la clasificación como no R1. Valores bajos en esta variable favorecen la clasificación como R1. En x500, valores bajos motivan la clasificación no R1 y valores altos motivan clasificar R1. En variables más avanzadas en el tiempo vemos como valores bajos favorecen más R1, los puntos azules se situan a la derecha de los rojos. Como vimos con el gráfico de importancia de variables los instantes iniciales influyen en la clasificación para R1.

- R2.

Figura 4.17: SHAP para clase R2 en inversores TM.

Vemos claramente como la interpretación de la variable x250 cambia frente a la clase R1, esto podría ser un factor diferencial entre estos grupos. Valores altos favorecen R2, valores bajos no R2. En el resto de variables se comporta similar a R1, valores bajos favorecen la clasificación R2 y valores altos no R2.

- R3.

Figura 4.18: SHAP para clase R3 en inversores TM.

Con un gráfico muy similar al de R2, en este caso vemos como valores bajos de la variable x1750 favorecen la clasificación como no R3. Esta variable podría ser realmente importante para diferenciar este grupo.

- R4.

Figura 4.19: SHAP para clase R4 en inversores TM.

Valores bajos en la variable x1750 favorece la clasificación como R4. Valores altos en ella favorecen no R4. También valores altos en las variables x250 y x750 favorecen la clasificación R4.

- R5.

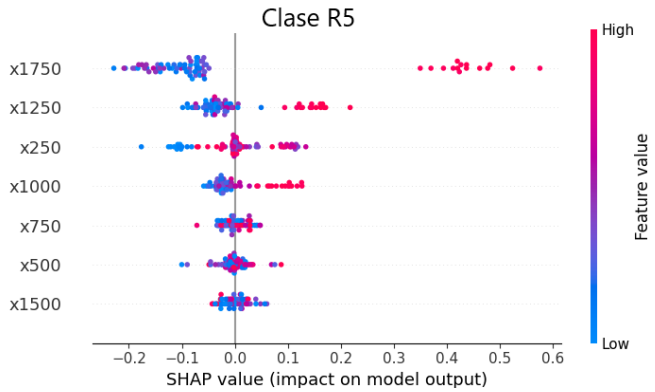

Figura 4.20: SHAP para clase R5 en inversores TM.

Valores bajos en la variable x1750 favorecen la clasificación como no R5, por el contrario, valores altos favorecen R5. Esta variable podría ser la diferenciadora entre los grupos R4 y R5. Vemos como valores altos en todas las variables favorecen la clasificación R5 puesto que los puntos rojos se sitúan siempre en la zona de valores positivos.

Podemos apreciar como a medida que aumentan los daños en el motor las variables que toman más importancia también van siendo mayores en el tiempo.

Para las clases R1, R2, R3 importan más los instantes iniciales e intermedios, en R4 y R5 toman más importancia los instantes finales.

Pese a ser uno de los inversores más conflictivos según la literatura existente, hemos logrado obtener una interpretabilidad muy clara, incluso mejor que para los otros inversores. También hemos logrado ver que variables son las que diferencian clasificar en un grupo u otro más claramente.

4.5 Comparación de métodos

En este capítulo compararemos los resultados obtenidos con los mejores resultados obtenidos sobre estos datos con técninas boosting [3].

Resultados de este trabajo

AB-NC2					
	R1	R2	R3	R4	R5
R1	0.667	0.017	0.3	0.017	0.0
R2	0.017	0.883	0.1	0.0	0.0
R3	0.2	0.133	0.667	0.0	0.0
R4	0.05	0.14	0.0	0.95	0.0
R5	0.0	0.0	0.0	0.0	1.0
Tasa Acierto = 0.834					

ABB-NC2					
	R1	R2	R3	R4	R5
R1	0.663	0.367	0.0	0.0	0.0
R2	0.383	0.55	0.05	0.017	0.0
R3	0.00	0	1.0	0.0	0.0
R4	0.0	0.0	0.017	0.967	0.017
R5	0.0	0.0	0.0	0.0	1.0
Tasa Acierto = 0.83					

TM-NC2					
	R1	R2	R3	R4	R5
R1	0.95	0.0	0.05	0.0	0.0
R2	0.017	0.733	0.25	0.0	0.0
R3	0.033	0.517	0.433	0.017	0.0
R4	0.0	0.0	0.017	0.933	0.05
R5	0.0	0.0	0.0	0.0	1.0
Tasa Acierto = 0.811					

Resultados de [3]

AB-NC2					
	R1	R2	R3	R4	R5
R1	0.78	0.04	0.17	0.0	0.01
R2	0.09	0.76	0.15	0.0	0.0
R3	0.36	0.1	0.54	0.0	0.0
R4	0.01	0.01	0.0	0.98	0.0
R5	0.0	0.0	0.0	0.0	1.0
Tasa Acierto = 0.812					

ABB-NC2					
	R1	R2	R3	R4	R5
R1	0.54	0.11	0.27	0.0	0.08
R2	0.13	0.79	0.01	0.07	0.0
R3	0.16	0.01	0.83	0.0	0.0
R4	0.0	0.0	0.0	1.0	0.0
R5	0.0	0.0	0.0	0.0	1.0
Tasa Acierto = 0.832					

TM-NC2					
	R1	R2	R3	R4	R5
R1	0.94	0.0	0.06	0.0	0.0
R2	0.01	0.66	0.33	0.0	0.0
R3	0.0	0.29	0.61	0.04	0.06
R4	0.0	0.0	0.0	1.0	0.0
R5	0.0	0.0	0.0	0.02	0.98
Tasa Acierto = 0.838					

Tabla 4.19: Comparación de resultados Boosting vs. Redes Neuronales.

Podemos observar, en la Tabla 4.19, unos resultados muy similares para ambas técnicas. Ambas logran una tasa de acierto en torno a 0.82. Siendo el peor el modelo de inversor TM para las redes neuronales y el mejor para boosting y por el contrario el modelo de inversor AB el mejor para redes neuronales y el peor para boosting, siempre refiriéndonos a mejor y peor como mejor y peor tasa de acierto. Si que podemos destacar que para todos los modelos aquí presentados y con unos resultados relativamente buenos se utiliza el nivel de carga alto o 2 ya que obtenía mejores tasas de acierto.

Podemos observar como las redes neuronales son capaces de diferenciar mejor las clases R1 y R2 frente a las técnicas boosting.

Capítulo 5

Conclusiones y trabajo futuro

Aquí se exponen las conclusiones obtenidas con la realización de este trabajo, así como posibles mejoras y trabajos futuros.

5.1 Conclusiones

Se ha comprobado en esta memoria que para un mejor funcionamiento de las redes neuronales en este tipo de problema es conveniente un preprocesado de los datos, igual que ocurría con el boosting. En otras palabras, las redes neuronales no han sido capaces por sí mismas de efectuar una selección de variables suficientemente eficiente.

Los resultados que se obtuvieron con las redes neuronales considreadas en este trabajo son similares a los obtenidos con boosting. Se mejora en la clasificación para el inversor AB y se pierde ligeramente para el inversor TM. Los resultados obtenidos para el inversor ABB son prácticamente idénticos.

Las redes neuronales aquí consideradas obtienen siempre la mejor clasificación a nivel de carga alto (NC2), lo que es coherente con la literatura existente ya que los fallos suelen aparecer de forma más clara a niveles de carga altos [48], este hecho también sucedía con las técnicas boosting. Se vuelve a confirmar la hipótesis de que el nivel de carga alto facilita la detección de fallos y la diagnosis del estado del motor.

También es destacable la superioridad de los modelos que utilizan la banda LSH sobre los de USH que aparecen también en boosting. Esto refuerza el mayor interés que parece tener LSH. Además, se descarta la hipótesis de la simetría entre las bandas. Los resultados parecen decir que LSH es significativamente mejor que USH a la hora de detectar los fallos lo que es una cuestión por investigar más detenidamente desde el punto de vista eléctrico.

Con la interpretabilidad extraída a través de SHAP se puede ver que variables y de que manera influyen en la predicción de nuevas observaciones. En términos generales, podríamos

decir que los instantes iniciales del estado transitorio son más determinantes en las clasificaciones de motores sanos y los instantes finales son más determinantes en la clasificación de los motores más dañados.

Respecto a la interpretabilidad de modelos, podemos destacar la dispersión de puntos en las redes neuronales ya que los resultados son están más mezclados y son menos claros frente a [3] donde los árboles proporcionaban unos resultados muy claros. Es interesante destacar que con las técnicas boosting apenas aparecían dos o tres variables con amplia dispersión (lo que invita a pensar en una mayor relevancia de la variable) en los gráficos SHAP, que eran más mucho claros. Con las redes neuronales parece que encuentra patrones en varias variables. Esto puede ser una ventaja porque las redes infieren más patrones o una desventaja porque los árboles consiguen separan mejor.

Para el inversor TM en boosting se utilizan los dos armónicos (USH y LSH) mientras que en redes neuronales solo se utiliza LSH. Según la información proporcionada por los expertos del Departamento de Ingeniería Eléctrica, este inversor es el más complejo de estudiar. Hay que notar que en este inversor las redes neuronales obtuvieron una tasa de error de 0.147 utilizando los datos originales frente a 0.189 utilizando los datos reducidos y frente a 0.162 que obtuvo boosting. El modelo basado en los datos originales no se ha detallado en este trabajo sobre todo por problemas en la interpretabilidad, ya que aparecían demasiadas variables en escena. Estos resultados refuerzan la opinión de los expertos del Departamento de Ingeniería Eléctrica sobre la complejidad del inversor y ponen de manifiesto la necesidad de un estudio más exhaustivo sobre el inversor TM.

5.2 Trabajo futuro y posibles mejoras

A continuación, se comentan algunas posibles mejoras para este trabajo y trabajos futuros que nacen de este TFG.

Los modelos entrenados en este trabajo son relativamente sencillos por la escasez de datos los cuales no podían satisfacer las necesidades de modelos complejos. Modelos con más neuronas y más capas podrían funcionar mejor para los datos actuales y conseguir mejores predicciones en cuanto al error se refiere, no obstante, siempre a riesgo de sobreajustar. También pueden utilizarse otro tipo de redes neuronales con memoria temporal, como las recurrentes o convolucionales 1D, para tratar los datos como una secuencia temporal.

Por un motivo de restricciones temporales y computacionales la búsqueda de hiperparámetros se realiza con una búsqueda aleatoria en ver de una búsqueda en cuadrícula. Esta primera es mucho más rápida al fijar el número de iteraciones y encuentra conjuntos de hiperparámetros correspondientes con un óptimo local. La búsqueda en cuadrícula es capaz de encontrar el óptimo global a costa de un algoritmo más parecido a la fuerza bruta y probar todas las posibilidades, algo que se salía completamente del tiempo disponible. Puede ser conveniente realizar una búsqueda más exhaustiva de hiper parámetros.

Tras analizar la estructura de las matrices de confusión se tiene la intuición que penalizar la etapa de entrenamiento según la distancia a la clase podría tener beneficios en cuanto a la clasificación. En este sentido podría ser útil el uso de otras funciones de pérdida como el MAE en el desarrollo de las redes.

También puede ser interesante el uso de nuevas técnicas específicas para datos tabulares, como la que se describe en TabNet [49], que es una red neuronal interpretable, y comprobar si la interpretación que se obtenga mediante esta metodología concuerda con la obtenida aquí.

Finalmente, otra labor de interés a completar, de acuerdo a lo sugerido por los expertos del Departamento de Ingeniería Eléctrica, es la publicación de los resultados aquí obtenidos en revistas internacionales de impacto.

Bibliografía

[1] R. Rosa. *Estudio sobre la viabilidad de los estadísticos de orden superior de la corriente de alimentación como indicadores para determinar el estado de un motor de inducción*. Trabajo de Fin de Grado, Universidad de Valladolid, 2015.

[2] Miljković Dubravko and Z Hep. Brief review of motor current signature analysis. *HDKBR Info-CrSNDT Journal*, 15:15–26, 2015.

[3] Alejandro Barón. *Detección y Clasificación de Fallos en Motores mediante Procedimientos Boosting*. Trabajo de Fin de Grado, Universidad de Valladolid, 2020.

[4] Sahar Zolfaghari, Samsul Bahari Mohd Noor, Mohammad Rezazadeh Mehrjou, Mohammad Hamiruce Marhaban, and Norman Mariun. Broken rotor bar fault detection and classification using wavelet packet signature analysis based on fourier transform and multi-layer perceptron neural network. *Applied Sciences*, 8(1), 2018.

[5] R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(7):179–188, 1936.

[6] D. R. Cox. The regression analysis of binary sequences. *Journal of the Royal Statistical Society: Series B (Methodological)*, 20(2):215–232, 1958.

[7] James N. Morgan and John A. Sonquist. Problems in the analysis of survey data, and a proposal. *Journal of the American Statistical Association*, 58(302):415–434, 1963.

[8] Dimitris Bertsimas and Jack Dunn. Optimal classification trees. *Machine Learning*, 106(7):1039–1082, April 2017.

[9] Zhou Yong. Knowledge discovery of interesting classification rules based on adaptive genetic algorithm. *International Journal of Computational Intelligence Systems*, 10 2007.

[10] Warren McCulloch and Walter Pitts. A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:127–147, 1943.

[11] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 65–386, 1958.

[12] Michael A. Nielsen. Neural networks and deep learning, 2018.

[13] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for activation functions. *CoRR*, abs/1710.05941, 2017.

[14] Lu Lu. Dying relu and initialization: Theory and numerical examples. *Communications in Computational Physics*, 28(5):1671–1706, Jun 2020.

[15] The softmax function. `http://akashgit.github.io/2017/03/13/unsaturating_softmax.html`. Último acceso: 2021-03-25.

[16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[17] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, 1991.

[18] A. Cauchy. Methode generale pour la resolution des systemes d'equations simultanees. *C.R. Acad. Sci. Paris*, 25:536–538, 1847.

[19] Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016.

[20] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning Representations by Back-propagating Errors. *Nature*, 323(6088):533–536, 1986.

[21] Garrett B. Goh, Nathan O. Hodas, and Abhinav Vishnu. Deep learning for computational chemistry, 2017.

[22] Keras API reference optimizers sgd. `https://keras.io/api/optimizers/sgd/`. Último acceso: 2021-03-25.

[23] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, page III–1139–III–1147. JMLR.org, 2013.

[24] Keras API reference optimizers rmsprop. `https://keras.io/api/optimizers/rmsprop/`. Último acceso: 2021-03-25.

[25] Keras API reference optimizers adam. `https://keras.io/api/optimizers/adam/`. Último acceso: 2021-03-25.

[26] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[27] Pan Zhou, Jiashi Feng, Chao Ma, Caiming Xiong, Steven HOI, et al. Towards theoretically understanding why sgd generalizes better than adam in deep learning. *arXiv preprint arXiv:2010.05627*, 2020.

[28] H Jabbar and Rafiqul Zaman Khan. Methods to avoid over-fitting and under-fitting in supervised machine learning (comparative study). *Computer Science, Communication and Instrumentation Devices*, pages 163–172, 2015.

[29] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. The elements of statistical learning – data mining, inference, and prediction.

[30] Keras API reference layer weight regularizers. `https://keras.io/api/layers/regularizers/`. Último acceso: 2021-04-7.

[31] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, January 2014.

[32] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[33] Lutz Prechelt. *Early Stopping — But When?*, pages 53–67. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[34] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13:281–305, February 2012.

[35] L. A. Rastrigin. The convergence of the random search method in the extremal control of a many parameter system. *Automaton Remote Control*, 24:1337–1342, 1963.

[36] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.

[37] Francisco Azuaje, Ian Witten, and Frank E. Witten ih, frank e: Data mining: Practical machine learning tools and techniques. *Biomedical Engineering Online - BIOMED ENG ONLINE*, 5:1–2, 01 2006.

[38] Christoph Molnar. *Interpretable Machine Learning*. 2019. `https://christophm.github.io/interpretable-ml-book/`.

[39] Tim Miller. Explanation in artificial intelligence: Insights from the social sciences. *CoRR*, abs/1706.07269, 2017.

[40] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should i trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 1135–1144, New York, NY, USA, 2016. Association for Computing Machinery.

[41] David Alvarez-Melis and Tommi S. Jaakkola. On the robustness of interpretability methods. *CoRR*, abs/1806.08049, 2018.

[42] Lloyd S. Shapley. *A Value for n-Person Games*. RAND Corporation, Santa Monica, CA, 1952.

[43] Erik Štrumbelj and Igor Kononenko. Explaining prediction models and individual predictions with feature contributions. *Knowledge and Information Systems*, 41:647–665, 12 2013.

[44] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[45] Zeon Trevor Fernando, Jaspreet Singh, and Avishek Anand. A study on the interpretability of neural retrieval models using DeepSHAP. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, July 2019.

[46] Vanesa Fernandez-Cavero, Luis A. García-Escudero, Joan Pons-Llinares, Miguel A. Fernández-Temprano, Oscar Duque-Perez, and Daniel Morinigo-Sotelo. Diagnosis of broken rotor bars during the startup of inverter-fed induction motors using the dragon transform and functional anova. *Applied Sciences*, 11(9), 2021.

[47] Raúl Granados. *Diagnóstico de fallos en el rotor de motores eléctricos en estado transitorio mediante técnicas estadísticas*. Trabajo de Fin de Grado, Universidad de Valladolid, 2017.

[48] Sakthivel Ganesan, Prince Winston David, Praveen Kumar Balachandran, and Devakirubakaran Samithas. Intelligent starting current-based fault identification of an induction motor operating under various power quality issues. *Energies*, 14(2), 2021.

[49] Sercan O Arik and Tomas Pfister. Tabnet: Attentive interpretable tabular learning. *arXiv preprint arXiv:1908.07442*, 2019.

Apéndice A

Anexos

A.1 Código

```
# -*- coding: utf-8 -*-
"""TFG maquina uva.ipynb

# Lectura y representacion de datos
"""

import numpy as np
import pandas as pd

#archivos subidos a la carpeta de google drive
originales=pd.read_csv('./Datos/Originales.csv',sep=";")
ambos_armonicos=pd.read_csv('./Datos/ambosarmonicos.csv',sep=',',decimal='.',index_col=0)
armonicosseparados=pd.read_csv('./Datos/armonicosseparados.csv',sep=',',decimal='.',index_col=0)

#Unimos los originales por columnas
originales_LSH=originales[originales.band=="LSH"]
originales_USH=originales[originales.band=="USH"]

originales_USH.index-=150
originales_todo=originales_LSH.merge(originales_USH.iloc[:,4:],how="right",left_index=True,right_index=True, suffixes=('_LSH','_USH'))
originales_todo.index=range(0,originales_todo.shape[0])
#Quitamos la informacion de la banda que ya no necesitamos
#originales_todo = originales_todo.loc[:, originales_todo.columns != 'band']
originales_todo = originales_LSH.loc[:, originales_LSH.columns != 'band']
#originales_todo = originales_USH.loc[:, originales_USH.columns != 'band']


ambos_armonicos= armonicosseparados.loc[armonicosseparados.band=="LSH", armonicosseparados.columns != 'band']
```

```
#ambos_armonicos= armonicosseparados.loc[armonicosseparados.band=="USH",
    armonicosseparados.columns != 'band']

#Preparamos un conjunto de datos utilizando el modelo de inversor y nivel
    de carga como variables dummy
originales_y_dummy = pd.get_dummies(originales_todo, columns=["model", "
    level"], prefix=["model", "level"])
ambos_armonicos_y_dummy = pd.get_dummies(ambos_armonicos, columns=["model"
    , "level"], prefix=["model", "level"])

#Los barajamos aleatoriamente
originales = originales.sample(frac=1)
ambos_armonicos = ambos_armonicos.sample(frac=1)
originales_todo = originales_todo.sample(frac=1)
originales_y_dummy = originales_y_dummy.sample(frac=1)
ambos_armonicos_y_dummy = ambos_armonicos_y_dummy.sample(frac=1)

"""Representacion de los datos"""

import matplotlib.pyplot as plt
import matplotlib.patches as mpatches

"""# Funciones auxiliares"""

import numpy as np
#A partir de los datos en formato data set, obtiene el subgrupo del modelo
    y nivel indicados
def get_subset(df,model,level,verbose=False):
  data= df.loc[(df.model==model) & (df.level==level)]
  if verbose: print(data)
  return data.drop(["model","level"],axis=1)

#Separa el dataFrame en X(data) e Y(target)
def preprocess(df, drop_band=True):
  #Para originales_todo y ambos_armonicos
  if drop_band:
    if 'band' in df.columns:
      df = df.drop("band",axis=1)
  X = df.drop("state",axis=1)
  y = pd.get_dummies(df.state)
  return [X,y]

def preprocess_LSTM(df):
  #datos de entrada en formato original, sin procesamiento despues de la
    lectura y de salida [samples,timestamp,features]
  #originales.csv y ambos_armonicos.csv preprocesados, ya separados en X e
    Y
  if "band" in df.columns:
  #Datos originales
    #XLSH = df.loc[df.band=="LSH"].drop(["state","band"],axis=1)
    #XUSH = df.loc[df.band=="USH"].drop(["state","band"],axis=1)
```

```
    XLSH = df.loc[df.band=="LSH"].drop(["band"],axis=1)
    XUSH = df.loc[df.band=="USH"].drop(["band"],axis=1)
  else:
  #Ambos armonicos
    XLSH = df.filter(regex="LSH|model|level")
    XUSH = df.filter(regex="USH|model|level")
    #XLSH = df.filter(regex="LSH")
    #XUSH = df.filter(regex="USH")


  XLSH = XLSH.values.reshape(XLSH.shape[0],XLSH.shape[1],1)
  XUSH = XUSH.values.reshape(XUSH.shape[0],XUSH.shape[1],1)

  X = np.concatenate([XLSH,XUSH],axis=2)
  #y = pd.get_dummies(df.state)

  #return (X,y)
  return X

def standarize(df,type="normal"):
  if type=="normal":
    #Columns
    return (df-df.mean())/df.std()
  elif type =="series":
    #Rows
    return df.sub(df.mean(1), axis=0).div(df.std(1), axis=0)

def plot_model(model):
  hist=model.history.history

  fig,ax = plt.subplots(1,2)

  ax[0].plot(hist["loss"],label="Train Loss")
  ax[1].plot(hist["accuracy"],label="Train Acc")
  try:
    ax[0].plot(hist["val_loss"],label="Validation Loss")
    ax[1].plot(hist["val_accuracy"],label="Validation Acc")
  except:
    print("No validation Split")
  ax[0].legend()
  ax[1].legend()

#Calcula la tasa de acierto en una matriz de confusion
def prec_mat(mat):
  return np.sum(np.diag(mat))/np.sum(mat)

#Imprime por pantalla los resultados(tasas de acierto) a partir de la
    matriz de confusion
def print_result_mat(m,metric="Error"):#metric Error o Accuracy
  l = m.shape
  if l == (5,5):
```

```
    acc_global = prec_mat(m)
  else:
    acc_global = prec_mat(np.sum(m,axis=0))
  if l[0] == 2:
    acc_NC1 = prec_mat(m[0])
    acc_NC2 = prec_mat(m[1])
  if l[0] == 3:
    acc_AB = prec_mat(m[0])
    acc_ABB = prec_mat(m[1])
    acc_TM = prec_mat(m[2])
  if l[0] == 6:
    acc_AB_NC1 = prec_mat(m[0])
    acc_ABB_NC1 = prec_mat(m[1])
    acc_TM_NC1 = prec_mat(m[2])
    acc_AB_NC2 = prec_mat(m[3])
    acc_ABB_NC2 = prec_mat(m[4])
    acc_TM_NC2 = prec_mat(m[5])

    acc_AB = prec_mat(m[0]+m[3])
    acc_ABB = prec_mat(m[1]+m[4])
    acc_TM = prec_mat(m[2]+m[5])
  if metric == "Error" or metric == "error":
    metrica = 1
  else:
    metric = "Acc"
    metrica = 0
  print(metric," global = ", abs(metrica - acc_global))
  if l == 2 or l == 6:
    print("\n",metric," NC1 = ", abs(metrica - acc_NC1))
    print(metric," NC2 = ", abs(metrica - acc_NC2))
  if l == 3 or l == 6:
    print("\n",metric," AB = ", abs(metrica - acc_AB))
    print(metric," ABB = ", abs(metrica - acc_ABB))
    print(metric," TM = ", abs(metrica - acc_TM))
  if l == 6:
    print("\n", metric, " AB-NC1 = ", abs(metrica - acc_AB_NC1))
    print(metric, " ABB-NC1 = ", abs(metrica - acc_ABB_NC1))
    print(metric, " TM-NC1 = ", abs(metrica - acc_TM_NC1))
    print(metric, " AB-NC2 = ", abs(metrica - acc_AB_NC2))
    print(metric, " ABB-NC2 = ", abs(metrica - acc_ABB_NC2))
    print(metric, " TM-NC2 = ", abs(metrica - acc_TM_NC2))

import numpy as np
from sklearn.model_selection import StratifiedKFold

# Given a set of HP, perform Stratified XV
def crossval(X,y,params,verbose=0,nsplits=10,net="normal",group="no"):
  #tipo puede ser normal, dummy o LSTM
  #group puede ser no: todos los datos, level para agrupar por nivel de
    carga, model para agrupar por modelo de inversor o both para ambos
  skf = StratifiedKFold(n_splits=nsplits,shuffle=True)
```

```
173
174 yr=np.argmax(y.values,axis=1) #From dummy to number
175 skf.get_n_splits(X,yr)
176
177 if verbose: print(skf)
178
179
180 if group =="no":
181   errors=np.zeros(nsplits,dtype=float)
182   n_obs=np.zeros(nsplits,dtype=float)
183   m = np.zeros((5,5),dtype=float)
184 elif group =="level":
185   errors=np.zeros((2,nsplits),dtype=float)
186   n_obs=np.zeros((2,nsplits),dtype=float)
187   m0 = np.zeros((5,5),dtype=float)
188   m1 = np.zeros((5,5),dtype=float)
189 elif group =="model":
190   errors=np.zeros((3,nsplits),dtype=float)
191   n_obs=np.zeros((3,nsplits),dtype=float)
192   m0 = np.zeros((5,5),dtype=float)
193   m1 = np.zeros((5,5),dtype=float)
194   m2 = np.zeros((5,5),dtype=float)
195 elif group =="both":
196   errors=np.zeros((6,nsplits),dtype=float)
197   n_obs=np.zeros((6,nsplits),dtype=float)
198   m0 = np.zeros((5,5),dtype=float)
199   m1 = np.zeros((5,5),dtype=float)
200   m2 = np.zeros((5,5),dtype=float)
201   m3 = np.zeros((5,5),dtype=float)
202   m4 = np.zeros((5,5),dtype=float)
203   m5 = np.zeros((5,5),dtype=float)
204
205
206 for iter,index in enumerate(skf.split(X,yr)):
207   #iter contiene la iteracion
208   #index[0]: train_index
209   #index[1]: test_index
210
211   if verbose: print("Training fold",iter+1)
212
213   X_train, X_test = X.iloc[index[0]], X.iloc[index[1]]
214   y_train, y_test = y.iloc[index[0]], y.iloc[index[1]]
215
216   if net=="normal":
217     X_train = standarize(X_train)
218     X_test = standarize(X_test)
219     model = build_model(X_train,y_train,params=params)
220
221   elif net=="dummy":
222     X_train.iloc[:,:-5] = standarize(X_train.iloc[:,:-5])
223     X_test.iloc[:,:-5] = standarize(X_test.iloc[:,:-5])
```

```
224       model = build_model_dummy(X_train,y_train,params=params)
225
226     elif net=="LSTM":
227       #Para saltarnos las dos primeras columnas en caso de que model y
    level esten
228       aux = 0
229       if "model" in X.columns:
230         aux = aux+1
231       if "level" in X.columns:
232         aux = aux+1
233       X_train.iloc[:,aux:] = standarize(X_train.iloc[:,aux:],"series") #
    standarize data by row (serie with mean 0 and std 1)
234       X_test.iloc[:,aux:] = standarize(X_test.iloc[:,aux:],"series")
235       model = build_LSTM_model(X_train,y_train,params=params)
236
237     if group =="no":
238       test = test_model(model,X_test,y_test,net)
239       errors[iter] = 1 - test["accuracy"]
240       n_obs[iter] = len(test["yhat"])
241       m = m + test["confusion_matrix"]
242       if verbose: print("ACC = ",test["accuracy"])
243     elif group =="level":
244       test0 = test_model(model,X_test.loc[X_test["level_NC1"]==1],y_test.
    loc[X_test["level_NC1"]==1],net)
245       test1 = test_model(model,X_test.loc[X_test["level_NC2"]==1],y_test.
    loc[X_test["level_NC2"]==1],net)
246       errors[0,iter] = 1-test0["accuracy"]
247       errors[1,iter] = 1-test1["accuracy"]
248       n_obs[0,iter] = len(test0["yhat"])
249       n_obs[1,iter] = len(test1["yhat"])
250       m0 = m0 + test0["confusion_matrix"]
251       m1 = m1 + test1["confusion_matrix"]
252       test = [test0,test1]
253       m = [m0,m1]
254       if verbose:
255         print("ACC NC1 = ",test0["accuracy"])
256         print("ACC NC2 = ",test1["accuracy"])
257     elif group =="model":
258       test0 = test_model(model,X_test.loc[X_test["model_AB"]==1],y_test.
    loc[X_test["model_AB"]==1],net)
259       test1 = test_model(model,X_test.loc[X_test["model_ABB"]==1],y_test.
    loc[X_test["model_ABB"]==1],net)
260       test2 = test_model(model,X_test.loc[X_test["model_TM"]==1],y_test.
    loc[X_test["model_TM"]==1],net)
261       errors[0,iter] = 1-test0["accuracy"]
262       errors[1,iter] = 1-test1["accuracy"]
263       errors[2,iter] = 1-test2["accuracy"]
264       n_obs[0,iter] = len(test0["yhat"])
265       n_obs[1,iter] = len(test1["yhat"])
266       n_obs[2,iter] = len(test2["yhat"])
267       m0 = m0 + test0["confusion_matrix"]
```

```
      m1 = m1 + test1["confusion_matrix"]
      m2 = m2 + test2["confusion_matrix"]
      test = [test0,test1,test2]
      m = [m0,m1,m2]
      if verbose:
        print("ACC AB = ",test0["accuracy"])
        print("ACC ABB = ",test1["accuracy"])
        print("ACC TM = ", test2["accuracy"])
    elif group =="both":
      X_test.iloc[:,:-5] = standarize(X_test.iloc[:,:-5])
      test0 = test_model(model,X_test.loc[(X_test["model_AB"]==1) & (
    X_test["level_NC1"]==1)],y_test.loc[(X_test["model_AB"]==1) & (X_test["
    level_NC1"]==1)],net)
      test1 = test_model(model,X_test.loc[(X_test["model_ABB"]==1) & (
    X_test["level_NC1"]==1)],y_test.loc[(X_test["model_ABB"]==1) & (X_test[
    "level_NC1"]==1)],net)
      test2 = test_model(model,X_test.loc[(X_test["model_TM"]==1) & (
    X_test["level_NC1"]==1)],y_test.loc[(X_test["model_TM"]==1) & (X_test["
    level_NC1"]==1)],net)
      test3 = test_model(model,X_test.loc[(X_test["model_AB"]==1) & (
    X_test["level_NC2"]==1)],y_test.loc[(X_test["model_AB"]==1) & (X_test["
    level_NC2"]==1)],net)
      test4 = test_model(model,X_test.loc[(X_test["model_ABB"]==1) & (
    X_test["level_NC2"]==1)],y_test.loc[(X_test["model_ABB"]==1) & (X_test[
    "level_NC2"]==1)],net)
      test5 = test_model(model,X_test.loc[(X_test["model_TM"]==1) & (
    X_test["level_NC2"]==1)],y_test.loc[(X_test["model_TM"]==1) & (X_test["
    level_NC2"]==1)],net)
      errors[0,iter] = 1-test0["accuracy"]
      errors[1,iter] = 1-test1["accuracy"]
      errors[2,iter] = 1-test2["accuracy"]
      errors[3,iter] = 1-test3["accuracy"]
      errors[4,iter] = 1-test4["accuracy"]
      errors[5,iter] = 1-test5["accuracy"]
      n_obs[0,iter] = len(test0["yhat"])
      n_obs[1,iter] = len(test1["yhat"])
      n_obs[2,iter] = len(test2["yhat"])
      n_obs[3,iter] = len(test3["yhat"])
      n_obs[4,iter] = len(test4["yhat"])
      n_obs[5,iter] = len(test5["yhat"])
      m0 = m0 + test0["confusion_matrix"]
      m1 = m1 + test1["confusion_matrix"]
      m2 = m2 + test2["confusion_matrix"]
      m3 = m3 + test3["confusion_matrix"]
      m4 = m4 + test4["confusion_matrix"]
      m5 = m5 + test5["confusion_matrix"]
      test = [test0,test1,test2,test3,test4,test5]
      m = [m0,m1,m2,m3,m4,m5]
      if verbose:
        print("ACC AB NC1 = ",test0["accuracy"])
        print("ACC ABB NC1 = ",test1["accuracy"])
```

```
307         print("ACC TM NC1 = ", test2["accuracy"])
308         print("ACC AB NC2 = ",test3["accuracy"])
309         print("ACC ABB NC2 = ",test4["accuracy"])
310         print("ACC TM NC2 = ", test5["accuracy"])
311
312   #si no hay que hacer la media ponderada cuando no separamos en grupos:
313   #if group == "no"  :
314
315   m_e = np.sum(errors*(n_obs/np.sum(n_obs)))
316   return {"errors":errors,"mean_e":m_e,"model":model, "test":test,"
      confusion_matrix":m, "n_obs":n_obs}
317
318 """Interpretabilidad de modelos complejos (SHAP)"""
319
320 import shap
321 def get_shap(model,X_train, X_test = None, type="deep",s=True):
322   #type: "deep" o "kernel"
323   '''
324   if X_test == None:
325     X_test = X_train
326   '''
327   if type=="deep":
328     explainer = shap.DeepExplainer(model,np.array(X_train))
329     shap_values = explainer.shap_values(np.array(X_train))
330
331     return shap.summary_plot(shap_values, np.array(X_train), plot_type="
      bar",feature_names=X_train.columns, show=s)
332   else:
333     explainer = shap.KernelExplainer(model.predict_proba,X_train)
334     shap_values = explainer.shap_values(X_train)
335
336     return shap.summary_plot(shap_values, X_train, plot_type="bar",
      feature_names=X_train.columns, show=s)
337
338 def get_shap_class(model,X_train, X_test = None, clase = "R1", type="deep"
     , s=True):
339   #type: "deep" o "kernel"
340   c=int(clase[-1])-1
341   '''
342   if X_test == None:
343     X_test = X_train
344   '''
345   if type=="deep":
346     explainer = shap.DeepExplainer(model,np.array(X_train))
347     shap_values = explainer.shap_values(np.array(X_train))
348
349     np.save('./resultados/shaps/shap_R'+str(c+1)+'.npy',shap_values[c])
350     aux = shap_values[c]
351     aux_train = X_train.iloc[np.where(np.all(aux<0.5,axis=1))]
352     aux = aux[np.where(np.all(aux<0.5,axis=1))]
353     aux_train = X_train.iloc[np.where(np.all(aux>-0.5,axis=1))]
```

```
    aux = aux[np.where(np.all(aux>-0.5,axis=1))]
    np.save('./resultados/shaps/shap_clipped_R'+str(c+1)+'.npy', shap_values[c])

    return shap.summary_plot(aux, aux_train,feature_names=X_train.columns, show=s)
  else:
    explainer = shap.KernelExplainer(model.predict_proba,X_train)
    shap_values = explainer.shap_values(X_train)
    #shap.summary_plot(shap_values[c], X_train,feature_names=X_train.columns)
    np.save('./resultados/shaps/shap_R'+str(c)+'.npy',shap_values[c])
    return shap.summary_plot(shap_values[c], X_train,feature_names=X_train.columns, show=s)

"""



Busqueda de hiperparametros"""

from kerastuner.tuners import RandomSearch
import tensorflow as tf
from kerastuner import HyperModel
from tensorflow import keras
from tensorflow.keras import layers
from kerastuner.tuners import RandomSearch
from kerastuner import HyperModel
from keras.regularizers import l1_l2

class MyHyperModel(HyperModel):

  def __init__(self, num_classes, input_shape):
    self.num_classes = num_classes
    self.input_shape = input_shape

  def build(self, hp):
    model = keras.Sequential()
    reg=hp.Choice('regularization',values=[0.0,0.025,0.05])

    model.add(layers.Dense(units= hp.Choice('units_1_0',values = [2,4,8,16,32]),
                           activation=hp.Choice('dense_activation_1_0', values=['relu', 'sigmoid']),#Funcion de activacion
                           input_dim=self.input_shape,
                           activity_regularizer=l1_l2(l1=reg, l2=reg)
                           )
              )
    model.add(layers.Dropout(rate=hp.Float('dropout_0', min_value=0.0, max_value=0.3, default=0.05, step=0.05)))
```

```
    lay=hp.Choice('layers', values = [1,2,3])
    for i in range(1,lay):#Entre 1 y 3 capas Densas
    #incluye 0, 1 o 2 capas mas

      model.add(layers.Dense(units= hp.Choice('units_l_'+str(i),values =
    [2,4,8,16,32]),
                             activation=hp.Choice('dense_activation_l_'+str
    (i), values=['relu', 'sigmoid']),#Funcion de activacion
                             input_dim=self.input_shape,
    activity_regularizer=l1_l2(l1=reg, l2=reg)
                             )
                )
      model.add(layers.Dropout(rate=hp.Float('dropout_'+ str(i), min_value
    =0.0, max_value=0.3, default=0.05, step=0.05)))
    #Capa de salida
    model.add(layers.Dense(self.num_classes, activation='softmax'))

    model.compile(hp.Choice('optimizer',['adam','rmsprop']), '
    categorical_crossentropy', metrics=['accuracy'])#Metodo de optimizacion
    return model


class MyTuner(RandomSearch):#esto lo pruebo yo pero no se si ira con RS
  def run_trial(self, trial, *args, **kwargs):
    # You can add additional HyperParameters for preprocessing and custom
    training loops
    # via overriding `run_trial`
    kwargs['batch_size'] = trial.hyperparameters.Choice('batch_size',
    values=[2,4,8,16,32])
    kwargs['epochs'] = trial.hyperparameters.Int('epochs', 30, 200)
    super(MyTuner, self).run_trial(trial, *args, **kwargs)

def buscar_model(X,y,trials=400,executions=5,val_split=0.15):
  #maximo de intentos: trials
  #ejecuciones por intento(modelo): executions
  #Porcentaje de datos que se usan para la validacion del modelo:
   val_split

  #Se crea el hipermodelo con el espacio de b squeda
  hypermodel = MyHyperModel(num_classes=5,input_shape=X.shape[1])
  #se establecen las caracteristicas de la busqueda y el tipo de busqueda
  #tuner = RandomSearch(
  tuner = MyTuner(#Para probar tunear epocas y batch_size
      hypermodel,
      objective='val_accuracy',
      max_trials=trials,
      executions_per_trial=executions,
      directory='my_dir',
      project_name='helloworld',
      overwrite=True)
  #Se entrenan los modelos buscando el mejor
```

```
  tuner.search(X, y,
    #si tuneamos epoch y batch_size no hay que pasarlo como argumento
    #epochs=100,
    #batch_size = 8,
    #validation_data=(X_test, d_test))
    validation_split=val_split,#validamos con un porcentaje de los mismos datos
    callbacks=[tf.keras.callbacks.EarlyStopping('val_loss', patience=3)],#Se anade para una parada temprana en las epocas si val_loss no decrece
    verbose = 0
    )
  model = tuner.get_best_models(num_models=1)[0]
  params = tuner.get_best_hyperparameters()[0]
  return params , tuner

from sklearn.model_selection import train_test_split
def hold_out(X,y,test_size=0.2,net="normal",t=400,e=5,reps=20,group=False):
  #reps: repeticiones
  #t: trials, e:executions
  #Separacion train-test
  if group:
    #Matrices de confusion por grupos
    m0 = np.zeros((5,5),dtype=float)#AB-NC1
    m1 = np.zeros((5,5),dtype=float)#ABB-NC1
    m2 = np.zeros((5,5),dtype=float)#TM-NC1
    m3 = np.zeros((5,5),dtype=float)#AB-NC2
    m4 = np.zeros((5,5),dtype=float)#ABB-NC2
    m5 = np.zeros((5,5),dtype=float)#TM-NC2
  else:
    m = np.zeros((5,5),dtype=float)

  for r in range(0,20):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size, stratify=y)
    #normalizar los datos
    if net=="dummy":
      X_train.iloc[:,:-5] = standarize(X_train.iloc[:,:-5])
      X_test.iloc[:,:-5] = standarize(X_test.iloc[:,:-5])
    if net=="normal":
      X_train = standarize(X_train)
      X_test = standarize(X_test)
    #Para LSTM no normalizamos

    #Construimos el modelo con los parametros obtenidos automaticamente
    params,tuner = buscar_model(X_train,y_train,trials=t,executions=e)

    model = tuner.hypermodel.build(params)
    model.fit(X_train,y_train,epochs=params.get('epochs'),batch_size=params.get('batch_size'),verbose=0)

```

```
    if group:
      test0 = test_model(model,X_test.loc[(X_test["model_AB"]==1) & (X_test["level_NC1"]==1)],y_test.loc[(X_test["model_AB"]==1) & (X_test["level_NC1"]==1)],net)
      test1 = test_model(model,X_test.loc[(X_test["model_ABB"]==1) & (X_test["level_NC1"]==1)],y_test.loc[(X_test["model_ABB"]==1) & (X_test["level_NC1"]==1)],net)
      test2 = test_model(model,X_test.loc[(X_test["model_TM"]==1) & (X_test["level_NC1"]==1)],y_test.loc[(X_test["model_TM"]==1) & (X_test["level_NC1"]==1)],net)
      test3 = test_model(model,X_test.loc[(X_test["model_AB"]==1) & (X_test["level_NC2"]==1)],y_test.loc[(X_test["model_AB"]==1) & (X_test["level_NC2"]==1)],net)
      test4 = test_model(model,X_test.loc[(X_test["model_ABB"]==1) & (X_test["level_NC2"]==1)],y_test.loc[(X_test["model_ABB"]==1) & (X_test["level_NC2"]==1)],net)
      test5 = test_model(model,X_test.loc[(X_test["model_TM"]==1) & (X_test["level_NC2"]==1)],y_test.loc[(X_test["model_TM"]==1) & (X_test["level_NC2"]==1)],net)
      m0 = m0 + test0["confusion_matrix"]
      m1 = m1 + test1["confusion_matrix"]
      m2 = m2 + test2["confusion_matrix"]
      m3 = m3 + test3["confusion_matrix"]
      m4 = m4 + test4["confusion_matrix"]
      m5 = m5 + test5["confusion_matrix"]
      m = [m0,m1,m2,m3,m4,m5]
    else:
      test = test_model(model, X_test, y_test, net)
      m = m + test["confusion_matrix"]

  return m

"""# Building model funtions"""

from keras.models import Sequential
from keras.layers import Dense,LeakyReLU,Dropout, LSTM, InputLayer
from keras.metrics import Accuracy
from keras.regularizers import l1_l2
from sklearn.metrics import accuracy_score, confusion_matrix



# Builds Keras model given a set of hyperparameters (passed as a dictionary)
def build_model(X,y,params):


  myreg=l1_l2(l1=params["regularizer"][0], l2=params["regularizer"][1])

  model = Sequential()
  #Input layer
```

```
524   model.add(Dense(params["nneurons"][0],activation=params["activations"
       ][0], input_shape=(X.shape[1],),activity_regularizer=myreg))
525   model.add(Dropout(params["dropouts"][0]))
526
527   #Hidden layers
528   for i in range(1,len(params["nneurons"])):
529     model.add(Dense(params["nneurons"][i],activation=params["activations"
       ][i],activity_regularizer=myreg))
530     model.add(Dropout(params["dropouts"][i]))
531
532   #Output layer
533   model.add(Dense(5,activation="softmax",activity_regularizer=myreg))
534
535   model.compile(optimizer=params["optimizer"],loss="
       categorical_crossentropy",metrics=["accuracy"])
536   model.fit(X,y,epochs=params["epochs"],batch_size=params["batch_size"],
       validation_split=params["val_split"],verbose=params["verbose"])
537   return model
538
539 # Builds Keras model given a set of hyperparameters (passed as a
       dictionary) with dummy variables at 6 least columns
540 def build_model_dummy(X,y,params):
541
542   myreg=l1_l2(l1=params["regularizer"][0], l2=params["regularizer"][1])
543
544   model = Sequential()
545   #Input layer
546   model.add(Dense(params["nneurons"][0],activation=params["activations"
       ][0], input_shape=(X.shape[1],),activity_regularizer=myreg))
547
548   model.add(Dropout(params["dropouts"][0]))
549   #Hidden layers
550   for i in range(1,len(params["nneurons"])):
551     model.add(Dense(params["nneurons"][i],activation=params["activations"
       ][i],activity_regularizer=myreg))
552     model.add(Dropout(params["dropouts"][i]))
553
554   #Output layer
555   model.add(Dense(5,activation="softmax"))
556
557   model.compile(optimizer=params["optimizer"],loss="
       categorical_crossentropy",metrics=["accuracy"])
558   model.fit(X,y,epochs=params["epochs"],batch_size=params["batch_size"],
       validation_split=params["val_split"],verbose=params["verbose"])
559   return model
560
561 # Builds Keras RNN model given a set of hyperparameters (passed as a
       dictionary)
562 def build_LSTM_model(X,y,params):
563
564   if "model" in X.columns:
```

```
    X = X.drop(["model"],axis=1)
  if "level" in X.columns:
    X = X.drop(["level"],axis=1)

  X = preprocess_LSTM(X) #reshape data

  lay = len(params["nneurons"])
  ret = [True]*(lay-1)+[False]

  model = Sequential()

  #Input layer
  model.add(LSTM(params["nneurons"][0],return_sequences=ret[0],input_shape
   =(X.shape[1],X.shape[2])))
  model.add(Dropout(params["dropouts"][0]))

  #Hidden layers
  for i in range(1,len(params["nneurons"])):
    model.add(LSTM(params["nneurons"][i],return_sequences=ret[i]))
    model.add(Dropout(params["dropouts"][i]))

  #output layer
  model.add(Dense(5,activation="softmax"))

  model.compile(optimizer=params["optimizer"],loss="
   categorical_crossentropy",metrics=["accuracy"])
  model.fit(X,y,epochs=params["epochs"],batch_size=params["batch_size"],
   validation_split=params["val_split"],verbose=params["verbose"])
  return model

def test_model(model,X,y,net="normal"):
  if net == "LSTM":
    if "model" in X.columns:
      X = X.drop(["model"],axis=1)
    if "level" in X.columns:
      X = X.drop(["level"],axis=1)
    X = preprocess_LSTM(X)
  yhat = np.argmax(model.predict(X),axis=1)
  ytest= np.argmax(y.values,axis=1)
  #La posicion Cij de la matriz de confusion corresponde con los de la
   clase real i asignados a la clase j (real filas, predicho columnas)
  return {"accuracy":accuracy_score(yhat,ytest),"yhat":yhat,"ytest":ytest,
   "confusion_matrix":confusion_matrix(ytest,yhat,labels=[0,1,2,3,4])}

"""# Saving results"""

#Crea un fichero el la ubicacion proporcionada
def create_file(arg):
  f = open(arg,"w+")
  f.close()

```

```
611 #Anade los resultados al final del fichero
612 #Debe ser un fichero ya existente
613 def write_R(title, m, file, metric="Error"):
614   file.write("%s" %title)
615   try:
616     l = m.shape
617   except:
618     l=len(m)
619   if l == (5,5):
620     file.write("Matriz global:\n")
621     np.savetxt(file, m,delimiter=",",fmt="%2.0f")
622     acc_global = prec_mat(m)
623   else:
624     file.write("Matriz global:\n")
625     np.savetxt(file, np.sum(m,axis=0),delimiter=",",fmt="%2.0f")
626     acc_global = prec_mat(np.sum(m,axis=0))
627 
628   if l == 6:
629     file.write("Matriz AB-NC1:\n")
630     np.savetxt(file, m[0],delimiter=",",fmt="%2.0f")
631     acc_AB_NC1 = prec_mat(m[0])
632     file.write("Matriz ABB-NC1:\n")
633     np.savetxt(file, m[1],delimiter=",",fmt="%2.0f")
634     acc_ABB_NC1 = prec_mat(m[1])
635 
636     file.write("Matriz TM-NC1:\n")
637     np.savetxt(file, m[2],delimiter=",",fmt="%2.0f")
638     acc_TM_NC1 = prec_mat(m[2])
639     file.write("Matriz AB-NC2:\n")
640     np.savetxt(file, m[3],delimiter=",",fmt="%2.0f")
641     acc_AB_NC2 = prec_mat(m[3])
642     file.write("Matriz ABB-NC2:\n")
643     np.savetxt(file, m[4],delimiter=",",fmt="%2.0f")
644     acc_ABB_NC2 = prec_mat(m[4])
645     file.write("Matriz TM-NC2:\n")
646     np.savetxt(file, m[5],delimiter=",",fmt="%2.0f")
647     acc_TM_NC2 = prec_mat(m[5])
648 
649     file.write("Matriz AB:\n")
650     np.savetxt(file, m[0]+m[3],delimiter=",",fmt="%2.0f")
651     acc_AB = prec_mat(m[0]+m[3])
652     file.write("Matriz ABB:\n")
653     np.savetxt(file, m[1]+m[4],delimiter=",",fmt="%2.0f")
654     acc_ABB = prec_mat(m[1]+m[4])
655     file.write("Matriz TM:\n")
656     np.savetxt(file, m[2]+m[5],delimiter=",",fmt="%2.0f")
657     acc_TM = prec_mat(m[2]+m[5])
658 
659     file.write("Matriz NC1:\n")
660     np.savetxt(file, m[0]+m[1]+m[2],delimiter=",",fmt="%2.0f")
661     acc_NC1 = prec_mat(m[0]+m[1]+m[2])
```

```
662       file.write("Matriz NC1:\n")
663       np.savetxt(file, m[3]+m[4]+m[5],delimiter=",",fmt="%2.0f")
664       acc_NC2 = prec_mat(m[3]+m[4]+m[5])
665     if metric == "Error" or metric == "error":
666       metrica = 1
667     else:
668       metric = "Acc"
669       metrica = 0
670     file.write("%s global = %s \n" %(metric,str(abs(metrica - acc_global))))
671     if l == 2 or l == 6:
672       file.write("%s NC1 = %s \n" %(metric, str(abs(metrica - acc_NC1))))
673       file.write("%s NC2 = %s \n" %(metric, str(abs(metrica - acc_NC2))))
674     if l == 3 or l == 6:
675       file.write("%s AB = %s \n" %(metric, str(abs(metrica - acc_AB))))
676       file.write("%s ABB = %s \n" % (metric,str(abs(metrica - acc_ABB))))
677       file.write("%s TM = %s \n" %(metric, str(abs(metrica - acc_TM))))
678     if l == 6:
679       file.write("%s AB-NC1 = %s \n" %(metric, str(abs(metrica - acc_AB_NC1)
      )))
680       file.write("%s ABB-NC1 = %s \n" %(metric, str(abs(metrica -
      acc_ABB_NC1))))
681       file.write("%s TM-NC1 = %s \n"%(metric, str(abs(metrica - acc_TM_NC1))
      ))
682       file.write("%s AB-NC2 = %s \n" %(metric, str(abs(metrica - acc_AB_NC2)
      )))
683       file.write("%s ABB-NC2 = %s \n" %(metric, str(abs(metrica -
      acc_ABB_NC2))))
684       file.write("%s TM-NC2 = %s \n" %(metric, str(abs(metrica - acc_TM_NC2)
      )))
685
686 def guarda_foto(archivo,imagen,titulo):
687     plt.title(titulo)
688     fig=imagen
689     plt.savefig(archivo)
690     plt.clf()
691
692 """#Funciones para la ejecucion"""
693
694 def crea_guarda_shaps(X,y,nombre=""):
695     #Nombre seria por ejemplo "AB-NC1"
696     #Ya esta estandarizado
697     #X = standarize(X)
698
699
700     #Construimos el modelo con los parametros obtenidos automaticamente
701     params,tuner = buscar_model(X,y ,trials=400,executions=5)
702
703     model = tuner.hypermodel.build(params)
704     model.fit(X,y,epochs=params.get('epochs'),batch_size=params.get('
        batch_size'),verbose=0)
705
```

```
  f1 = get_shap(model,X,type="deep",s=False)
  guarda_foto(archivo="./resultados/total_"+nombre+".png",imagen=f1,titulo
   =nombre)

  for i in range(5):
    f2 = get_shap_class(model,X,clase="R"+str(i+1),type="deep",s=False)
    guarda_foto(archivo="./resultados/"+nombre+"R"+str(i+1)+".png",imagen=
   f2,titulo=nombre+" clase R"+str(i+1))

import inspect
def retrieve_name(var):
  callers_local_vars = inspect.currentframe().f_back.f_locals.items()
  return [var_name for var_name, var_val in callers_local_vars if var_val
   is var]

def crea_guarda_resultados(datos,nombre,modelo=None, level=None):
  #datos: ambos_armonicos, originales_todo (Para los grupos)
  #modelo: AB, ABB, TM (o dejarlo por defecto si se eligen todos)
  #level: NC1 o NC2 (o dejarlo por defecto si se eligen ambos)
  if modelo == None or level == None:
    data = datos
    data = data.sample(frac=1)
    X,y = preprocess(data)
    m = hold_out(X,y, net="dummy", group=True)
    archivo = "./resultados/resultados.txt"
    f=open(archivo,"a+")
    write_R("Resultados "+nombre+"\n",m,f)
    f.close()
    X.iloc[:,:-5] = standarize(X.iloc[:,:-5])
    crea_guarda_shaps(X,y,nombre)
  else:
    data = get_subset(datos,modelo,level)
    data = data.sample(frac=1)
    X,y = preprocess(data)
    m = hold_out(X,y)
    archivo = "./resultados/resultados.txt"
    f=open(archivo,"a+")
    write_R("Resultados "+nombre+str(modelo)+str(level)+"\n",m,f)
    f.close()
    X = standarize(X)
    crea_guarda_shaps(X,y,nombre+str(modelo)+str(level))
#Hay que cambiar numero de ejecuciones...

#archivo = "./resultados_LSH/resultados.txt"
archivo = "./resultados/resultados.txt"
#archivo = "./resultados_USH/resultados.txt"
create_file(archivo)

"""# Datos por grupos

Armonicos
```

```
"""

crea_guarda_resultados(ambos_armonicos,retrieve_name(ambos_armonicos)[0],'AB','NC1')


crea_guarda_resultados(ambos_armonicos,retrieve_name(ambos_armonicos)[0],'ABB','NC1')


crea_guarda_resultados(ambos_armonicos,retrieve_name(ambos_armonicos)[0],'TM','NC1')


crea_guarda_resultados(ambos_armonicos,retrieve_name(ambos_armonicos)[0],'AB','NC2')


crea_guarda_resultados(ambos_armonicos,retrieve_name(ambos_armonicos)[0],'ABB','NC2')


crea_guarda_resultados(ambos_armonicos,retrieve_name(ambos_armonicos)[0],'TM','NC2')


"""Originales"""

crea_guarda_resultados(originales_todo,retrieve_name(originales_todo)[0],'AB','NC1')


crea_guarda_resultados(originales_todo,retrieve_name(originales_todo)[0],'ABB','NC1')


crea_guarda_resultados(originales_todo,retrieve_name(originales_todo)[0],'TM','NC1')


crea_guarda_resultados(originales_todo,retrieve_name(originales_todo)[0],'AB','NC2')


crea_guarda_resultados(originales_todo,retrieve_name(originales_todo)[0],'ABB','NC2')


crea_guarda_resultados(originales_todo,retrieve_name(originales_todo)[0],'TM','NC2')

```

```
793
794 """# Dummies
795
796 Originales
797 """
798
799 crea_guarda_resultados(originales_y_dummy,retrieve_name(originales_y_dummy
    )[0])
800
801
802 """Armonicos"""
803
804 crea_guarda_resultados(ambos_armonicos_y_dummy,retrieve_name(
    ambos_armonicos_y_dummy)[0])
```

Índice de figuras

Índice de tablas