Universidad del País Vasco
Euskal Herriko Unibertsitatea
ZIENTZIA ETA TEKNOLOGIA FAKULTATEA
FACULTAD DE CIENCIA Y TECNOLOGÍA

Trabajo de Fin de Grado
Grado en Física

Predicción de la temperatura crítica de superconductores mediante técnicas de Machine Learning

Redes neuronales y algoritmos genéticos

Autor:
Javier Herrero Álvarez
Director:
Javier Echanobe Arias

Leioa, 22 de junio de 2023

Índice

1. Introducción y objetivos

El término Machine Learning (ML) hace referencia a un conjunto de técnicas computacionales cuyo objetivo es que las máquinas aprendan a partir de la experiencia adquirida. Está englobado dentro de otro conjunto de técnicas más amplio, denominado Inteligencia Artificial (IA). En sus inicios, la IA se centró en resolver problemas complicados para la mente humana pero sencillos para los computadores, es decir, cálculos matemáticos y algoritmos. Hoy en día, el reto está en la resolución de problemas que son relativamente fáciles e intuitivos para los humanos, pero que no son sencillos de formular en un contexto computacional [1]. Por ejemplo, cuando un humano pasea por la calle, es capaz de reconocer de inmediato si un objeto que está viendo es un coche, una persona, etc. Encontrar un algoritmo computacional (i.e. escribir el código de un programa de ordenador tradicional) para reconocimiento de objetos a partir de imágenes, sin embargo, no es sencillo. La diferencia clave entre los humanos y las máquinas en este contexto es que los humanos, al contrario que las máquinas, adquieren conocimiento y poseen intuición [1].

ML es un enfoque de IA cuya filosofía es que los computadores adquieran experiencia y conocimiento a partir de una gran cantidad de datos, al igual que hacen los seres humanos a lo largo de sus vidas. Una de las claves para obtener un buen modelo ML es escoger adecuadamente la representación de los datos que se van a utilizar para entrenar (*train*) a la máquina, es decir, cuántas características (*features* o predictores) se le van a proporcionar y en qué formato [1], buscando evitar características redundantes que aumenten la complejidad del modelo innecesariamente. Esto se conoce como selección de características (*feature selection*).

En este trabajo, se aplican distintos modelos ML y técnicas de selección de características al problema de la predicción de temperaturas críticas de superconductores. Los superconductores son materiales para los que a temperaturas menores que una temperatura crítica T_c la resistividad eléctrica se hace cero. A la temperatura T_c se produce una transición de fase, pasando bruscamente de una resistividad no nula a una resistividad nula [2]. Este tipo de materiales, por tanto, permiten el flujo de corriente a través de ellos sin disipación de potencia por efecto Joule, lo cual resulta muy útil en aplicaciones de electrotecnia.

Sin embargo, a día de hoy el uso de los superconductores no está para nada extendido. Esto se debe fundamentalmente a dos motivos [3]:

- Las temperaturas críticas son, por lo general, extremadamente bajas. El mayor valor de T_c registrado hasta Julio de 2017 es de unos 203 K (unos $-70\,^{\circ}$C) [3].
- No existe un modelo teórico que explique la dependencia funcional de la temperatura crítica T_c, lo cual dificulta la búsqueda de posibles materiales superconductores a temperatura ambiente.

Algunos investigadores han desarrollado modelos ML de regresión para predecir la temperatura crítica de superconductores. Por ejemplo, Kam Hamidieh obtiene un modelo con RMSE = 9.5 K (*Root Mean Squared Error*) y $R^2 = 0.92$ (coeficiente de determinación) basado en una técnica ML denominada XGBoost (conjunto de árboles propulsado por gradiente) [3]. Otros modelos de regresión propuestos en la literatura son el modelo *random forest* de Kaname Matsumoto y Tomoya Horide (con $R^2 = 0.92$) [4] o el modelo *bagged trees* de B. Roter y S.V. Dordevic (con RMSE = 8.91 K y $R^2 = 0.93$) [5]. Los tres modelos mencionados parten de la base de datos Supercon, del Instituto Nacional de Japón de Ciencia de Materiales, Japan's National Institute for Materials Science (NIMS), y predicen la temperatura crítica partiendo de la fórmula empírica del material.

En este trabajo se desarrolla un modelo de regresión para predecir T_c partiendo de los predictores extraídos por Kam Hamidieh de la base de datos Supercon (relacionados con la fórmula empírica del material). Sin embargo, el enfoque seguido es distinto: el modelo que se desarrolla está basado en redes neuronales.

El principal objetivo de este trabajo es encontrar un modelo ML que, a partir de la fórmula empírica de un material superconductor, prediga su temperatura crítica. Este modelo podría ser útil para encontrar nuevos superconductores de alta temperatura. En esta línea, se desarrollarán distintos modelos ML, y se compararán sus resultados. En concreto, se utilizarán modelos basados en redes neuronales FFN y redes ELM.

Un segundo objetivo del trabajo es la ganancia de intuición física acerca de qué propiedades de los elementos atómicos tienen mayor influencia sobre la temperatura crítica de los superconductores. Las técnicas genéticas de selección de características utilizadas convergerán hacia modelos con un número reducido de predictores, proporcionando la mencionada intuición física, así como una menor complejidad del modelo. Cabe destacar que los algoritmos genéticos también se utilizan en este trabajo para optimizar los hiperparámetros de los modelos de regresión utilizados.

Como último objetivo, de carácter más amplio, se pretende ilustrar cómo se pueden aplicar los distintos algoritmos ML en la resolución de problemas de física y demostrar su potencia para la resolución de problemas que con métodos analíticos o numéricos tradicionales requerirían de una cantidad excesiva de recursos y tiempo.

El trabajo comienza con una descripción cualitativa del fenómeno de la superconductividad en la sección 2, en base al modelo BCS y las reglas de Matthias.

En la sección 3 se presentan las técnicas ML utilizadas para el desarrollo del trabajo. Tras explicar los aspectos fundamentales del Machine Learning, se explican las técnicas basadas en redes neuronales FFN, redes ELM y algoritmos genéticos.

La sección 4 recoge los experimentos realizados a lo largo del trabajo utilizando las herramientas ML presentadas en la sección previa, así como los resultados obtenidos y un análisis de los mismos. El modelo que se obtiene finalmente es una red neuronal FFN con 35 predictores, reduciendo los 81 que había inicialmente en el *dataset* en un 56.79 %. Los indicadores de eficiencia promedio obtenidos, RMSE = 11.66 K y $R^2 = 0.8849$, no llegan a superar a los de los modelos de referencia presentados en esta introducción. Sin embargo, la reducción del número de predictores conseguida proporciona intuición física sobre qué variables influyen sobre la temperatura crítica, además de que el modelo FFN desarrollado tiene una estructura computacional más sencilla que los utilizados en los otros trabajos mencionados (basados en árboles de decisión).

Finalmente, en la sección 5 se presentan las conclusiones extraídas de los experimentos realizados. También se mencionan posibles líneas futuras de investigación en el ámbito de este trabajo.

Se incluyen también tres apéndices en los que se realizan desarrollos matemáticos que, si bien no son necesarios para una lectura de esta memoria, pueden resultar interesantes como profundización. El apéndice A contiene una introducción a algunos conceptos básicos de estadística, el apéndice B contiene el desarrollo matemático de los algoritmos *backpropagation* y descenso del gradiente y el apéndice C la relación entre el MSE de una variable normalizada y de dicha variable sin normalizar. Además, se proporciona un glosario de siglas y términos en inglés para facilitar una consulta rápida en caso necesario.

2. Superconductividad y temperatura crítica

Los superconductores son materiales que, por debajo de una temperatura crítica T_c, experimentan una transición a una fase de resistividad eléctrica nula. Tradicionalmente, se han dividido en dos grandes grupos: los superconductores de tipo I tienen una transición brusca a la fase superconductora a la temperatura crítica, mientras que los de tipo II presentan una transición gradual. Estos segundos siguen presentando propiedades superconductoras a temperaturas más altas, es por ello que los de tipo I también se denominan de baja temperatura y los de tipo II de alta temperatura [2].

Es importante mencionar que los materiales superconductores presentan propiedades (como el efecto Meissner) que no pueden explicarse modelándolos como un material conductor cuya resistividad tiende a cero [2]. Esto sugiere que es necesario el desarrollo de una teoría de la superconductividad independiente a la conductividad. La teoría de la superconductividad de Bardeen-Cooper-Schrieffer (BCS) ha sido uno de los intentos más relevantes por encontrar una teoría de la superconductividad. Por otra parte, las dificultades mencionadas en la sección 1 para el desarrollo de un modelo teórico para los superconductores han motivado la búsqueda de reglas empíricas que recogen las condiciones para la superconductividad y las variables que afectan sobre la temperatura crítica. Un ejemplo son las reglas de Matthias que, aunque no son rigurosas ni se cumplen siempre, proporcionan intuición sobre las propiedades que influyen sobre T_c.

En la sección 2.1 se realiza una breve introducción cualitativa a la teoría BCS. En la sección 2.2 se presentan las reglas de Matthias para tratar de ganar intuición sobre qué propiedades influyen sobre la superconductividad.

2.1. Teoría BCS de la superconductividad: los pares de Cooper

La teoría BCS explica el fenómeno de la superconductividad para superconductores de tipo I. La idea principal que hay detrás de esta teoría es que, si existe un estado en el que la resistencia es cero, ha de existir un *bandgap* entre dicho estado y el estado conductor normal [6]. En un conductor normal, con resistencia eléctrica finita, los electrones chocan con los iones de la red y, consecuentemente, pierden energía [7] (efecto Joule). En un superconductor, también hay un movimiento de portadores a través de la red, y la única posibilidad para que no haya una interacción con los iones de la red (con su consecuente pérdida de energía) es que no haya ningún nivel energético disponible en el rango de las energías de interacción con la red. Además, como a temperaturas mayores que T_c sí que hay disipación por efecto Joule, el intervalo energético de niveles prohibidos deberá ser del orden de la energía térmica $k_B T_c$, siendo k_B la constante de Boltzmann [6].

Por otra parte, es un resultado bien conocido de la mecánica estadística que en un sistema de fermiones el número de ocupación de los niveles decrece monotónicamente como una función escalón no del todo vertical (estadística de Fermi-Dirac). Además, según el principio de exclusión de Pauli, no es posible que haya dos fermiones en un mismo estado cuántico. Como un electrón individual es un fermión (tiene espín semientero), los electrones quedan descartados como posibles portadores en un superconductor, ya que el principio de exclusión impide que estén todos ellos en el estado superconductor. Así, los portadores han de ser bosones [6]. La teoría BCS propone como portadores del estado superconductor pares de electrones de espín opuesto que quedan emparejados entre sí debido a una fuerza atractiva que es capaz de vencer la repulsión de Coulomb, conocidos como pares de Cooper. Estos pares tienen, en efecto, naturaleza bosónica, ya que

su espín resultante es entero por estar formados por dos partículas de espín semientero [6]. Es precisamente la naturaleza bosónica de los pares de Cooper la que permite que a temperaturas bajas (menores que T_c) todos los portadores de carga formen un condensado de Bose-Einstein en el estado superconductor que se forma por debajo del intervalo prohibido.

En la figura 2.1 se muestra un esquema del estado superconductor y el *bandgap* para ilustrar la explicación de los dos párrafos anteriores. El esquema mostrado es válido solamente para $T < T_c$, esto es, en la fase superconductora.

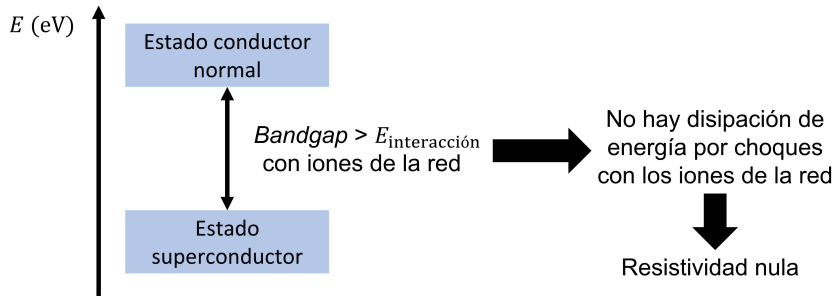


Figura 2.1: Esquema del estado superconductor y el *bandgap* que se forma entre éste y el estado conductor normal del material.

Para terminar con esta breve descripción de la teoría BCS, falta describir la naturaleza de la interacción atractiva entre los electrones:

- En superconductores de tipo I, la fuerza atractiva se debe al intercambio de fonones (cuantos de vibración de la red) entre los dos electrones del par [7].
- En superconductores de tipo II, no está tan clara la naturaleza de la interacción, pero una posibilidad es que se deba a un apantallamiento de la interacción repulsiva de Coulomb. La presencia de un electrón en una red provoca una deformación de la estructura cristalina, atrayendo los iones positivos cercanos hacia él. Esta densidad de carga positiva neta creada en torno a un electrón apantalla su carga negativa, permitiendo que los dos electrones que forman el par experimenten una fuerza atractiva [7].

2.2. Las reglas de Matthias

Las reglas de Matthias son un conjunto de observaciones empíricas que tratan de recoger las condiciones para que un material sea superconductor y la dependencia funcional de T_c [8]. Son más bien una serie de pautas para facilitar la búsqueda de nuevos superconductores que unas reglas universales. A pesar de sus evidentes y muy grandes limitaciones, estas reglas dan pistas de qué propiedades de los elementos atómicos y de las estructuras cristalinas influyen en la superconductividad, y pueden utilizarse para decidir qué variables incluir como predictores en los modelos de *Machine Learning* que se desarrollarán.

2.2.1. Reglas de Matthias para elementos atómicos

Los experimentos con superconductores dan lugar a las siguientes reglas, aunque no siempre se cumplen [8]:

1. Los elementos superconductores tienen un número de electrones de valencia por átomo n comprendido entre 2 y 8.

2. En general, los elementos ferromagnéticos, antiferromagnéticos, no metálicos, semimetálicos o semiconductores no son superconductores.
3. La temperatura crítica es proporcional a una función T(n) del número de electrones de valencia por átomo.
4. La temperatura crítica es proporcional a la siguiente función del volumen atómico V y la masa atómica M: $T_c \propto \frac{V^x}{M}$, siendo $4 < x < 5$.
5. Existen ciertas estructuras cristalinas que son más propensas a la superconductividad.

De cara a este trabajo, lo más interesante de estas reglas empíricas es que la existencia de una fase superconductora para un elemento y, en caso afirmativo, su temperatura crítica, dependen de la estructura electrónica del elemento, de sus parámetros de escala (masa y volumen) y de su estructura cristalina. Otro apunte interesante es que la función $T(n)$ tiene máximos locales para n impar y mínimos para n par en el caso de los metales de transición, mientras que es creciente en el caso del resto de metales. Esta dependencia funcional sugiere que, además de la interacción mediada por fonones propuesta en la teoría BCS, posiblemente haya involucrado un término de interacción espín-espín [8].

2.2.2. Reglas de Matthias para compuestos y aleaciones

Las reglas de Matthias para compuestos y aleaciones son similares a las atómicas, con las siguientes modificaciones [8]:

1. n se refiere ahora al número de electrones de valencia por átomo en la aleación o compuesto.
2. Idéntica al caso atómico, excepto algunos ferromagnéticos de espín de orbitales f.
3. Idéntica al caso atómico, pero el patrón de picos está desplazado para cada material.
4. En este caso, $T_c \propto V^x$.
5. Similar al caso atómico.

El hecho de que estas reglas sean similares al caso atómico sugiere que un modelo de predicción de T_c basado en la fórmula empírica sea viable.

3. Técnicas de Machine Learning

Las técnicas Machine Learning (ML), traducido habitualmente como Aprendizaje Automático, consisten en modelos computacionales que pueden extraer conocimiento a partir de datos, lo cual se conoce como proceso de entrenamiento (*train*). En particular, los modelos ML de aprendizaje supervisado se entrenan con pares entrada-salida reales, y tienen como objetivo que la máquina obtenga una salida lo más próxima posible a la real cuando se le presenten entradas que no ha visto durante la fase de entrenamiento (generalización). Existen otro tipo de modelos en los que únicamente se utilizan las entradas durante el entrenamiento (aprendizaje no supervisado).

Esta sección comienza con una introducción a algunos aspectos fundamentales de las técnicas ML (sección 3.1). A continuación, se presentan los modelos de regresión que se emplearán: redes neuronales FFN (sección 3.2) y redes ELM (sección 3.3). Finalmente, se explican los algoritmos genéticos y su aplicación a la selección de características y la optimización de hiperparámetros (sección 3.4).

3.1. Aspectos fundamentales de las técnicas ML

3.1.1. Clasificación general de los modelos ML

Una primera clasificación de los algoritmos ML se realiza atendiendo al tipo de tarea (*task*) o problema que resuelven [1]. Algunos ejemplos son:

- Algoritmos de clasificación (*classification*): su objetivo es predecir a qué categoría de entre k categorías especificadas pertenece una entrada dada.
- Algoritmos de regresión (*regression*): su objetivo es predecir un valor numérico a partir de una entrada.
- Algoritmos de agrupación (*clustering*): son algoritmos que dividen un conjunto de datos en grupos (*clusters*) de características similares.

Otra clasificación es según los datos utilizados para el entrenamiento [1]:

- Algoritmos supervisados (*supervised*): los datos de entrenamiento contienen características o predictores (*features*) para cada entrada, y también contienen una etiqueta o valor numérico que es la salida que se desea predecir (*target*). Los algoritmos de regresión y de clasificación pertenecen a esta categoría.
- Algoritmos no supervisados (*unsupervised*): los datos de entrenamiento únicamente contienen características o predictores, y el objetivo es encontrar patrones en los datos. Los algoritmos de *clustering* están englobados en esta categoría.

En este trabajo, el problema a resolver es la predicción de la temperatura crítica de superconductores a partir de una serie de características relacionadas con su fórmula empírica. Es decir, partiendo de un vector de características, el objetivo es obtener una temperatura, un escalar. Por lo tanto, los algoritmos ML utilizados son de regresión.

3.1.2. Datos de entrenamiento, validación y test

A la hora de entrenar un modelo ML como una red neuronal se divide la base de datos (de ahora en adelante *dataset*) en tres subconjuntos [9]:

- Datos de entrenamiento (*training data*): los parámetros internos del modelo se actualizan de manera iterativa durante el entrenamiento con el objeto de reducir el error (o aumentar la eficiencia) en sus predicciones para este conjunto de datos.

- Datos de validación (*validation data*): este conjunto de datos se utiliza para monitorizar el *overfitting* (explicado en la sección 3.1.4) durante el entrenamiento. Cuando empieza a aumentar el error para este conjunto de datos, se deja seguir el entrenamiento un número de iteraciones prefijado (paciencia de validación). Si en dichas iteraciones no disminuye el error para los datos de validación, se detiene el entrenamiento, aunque siga disminuyendo el error para los datos de entrenamiento.
- Datos de prueba (*test data*): finalizado el entrenamiento, se utiliza este conjunto para medir la eficiencia del modelo con datos que nunca ha visto (i.e. su capacidad de generalizar, explicado en la sección 3.1.4) y para compararlo con otros modelos.

La proporción de cada uno de los conjuntos utilizada en este trabajo es del 60 % *training data*, 20 % *validation data* y 20 % *test data*.

3.1.3. Indicadores para la eficiencia de los modelos de regresión

A la hora de ajustar los parámetros internos de un modelo (entrenamiento) o de evaluar su eficiencia y compararlo con otros modelos, es necesario utilizar un indicador cuantitativo de su rendimiento. En los modelos de regresión, un indicador típico es el error cuadrático medio (MSE, *Mean Squared Error*), que se define como [1]:

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2 = \frac{1}{N}||\vec{\hat{y}} - \vec{y}||^2 \tag{3.1}$$

donde N es el número de datos sobre los que se calcula el error. La eficiencia del modelo es mayor cuanto menor sea MSE, ya que se puede ver la raíz de dicho indicador ($\text{RMSE} = \sqrt{\text{MSE}}$) como la distancia euclídea total entre las predicciones del modelo para las distintas entradas del *dataset* (vector $\vec{\hat{y}}$) y los valores reales objetivo (vector $\vec{y}$), salvo un factor $\frac{1}{\sqrt{N}}$ [1]. Dependiendo de si el cálculo se realiza sobre los datos de entrenamiento, validación o test el indicador se denomina $\text{MSE}_{\text{train}}$, MSE_{val} o MSE_{test}, respectivamente.

Así, los parámetros de la red se ajustan durante el entrenamiento de manera que se vaya reduciendo $\text{MSE}_{\text{train}}$ y el entrenamiento se detiene cuando MSE_{val} no disminuye durante el número de iteraciones especificado en la paciencia de validación. El modelo se evalúa y compara con otros utilizando MSE_{test}.

Otros indicadores ampliamente utilizados son el coeficiente de correlación de Pearson (R) o su cuadrado, el coeficiente de determinación (R^2). Ambos se explican en detalle en el apéndice A. Un modelo de regresión es mejor cuanto más cercanos sean estos dos indicadores a la unidad. Suele aceptarse el criterio $R^2 > 0.9$, equivalente a $R > 0.95$.

3.1.4. Generalización, *overfitting* y *underfitting* de los modelos

En ML, no basta con que el modelo tenga una buena eficiencia para los datos de entrenamiento, sino que tiene que ser capaz de realizar predicciones satisfactorias para datos que nunca antes ha visto. La capacidad de un modelo de lograr una buena eficiencia con datos no vistos durante el entrenamiento se llama generalización (*generalization*) [1].

La fase de entrenamiento de un modelo no es más que la resolución de un problema de optimización: reducir el error en las predicciones del modelo para los datos de entrenamiento, $\text{MSE}_{\text{train}}$. Sin embargo, el verdadero objetivo de un modelo ML es que el error para datos nunca antes vistos por el modelo, MSE_{test} (también llamado error de generalización) sea lo más bajo posible. Así, un buen modelo ML es aquel con un error de entrenamiento bajo y un error de test lo más próximo posible al error de entrenamiento.

Existe, sin embargo, cierta incompatibilidad entre estas dos condiciones. Si un modelo es excesivamente sencillo, puede tener problemas para obtener un buen error de entrenamiento (*underfitting*). El *underfitting* se soluciona aumentando la complejidad del modelo. Si un modelo es excesivamente complejo, puede llegar a memorizar propiedades de los datos de entrenamiento que no generalizan a los datos de test, agrandando la distancia entre el error de entrenamiento y el de test (*overfitting*). El *overfitting* se soluciona reduciendo la complejidad del modelo o aumentando el tamaño del *dataset* utilizado [1].

En este trabajo, se dispone de una cantidad de datos limitada, luego es de vital importancia no complicar excesivamente los modelos utilizados para evitar el *overfitting*. La principal estrategia que se utiliza para ello es la selección de características (sección 3.4.2), además del uso de un conjunto de datos de validación para determinar el fin del proceso de entrenamiento (sección 3.1.2).

3.1.5. Hiperparámetros de un modelo

Los hiperparámetros (*hyperparameters*) son parámetros de un modelo ML que sirven para controlar su complejidad o el proceso de entrenamiento [1]. Por ejemplo, el número de capas de una red neuronal o el número de neuronas por capa son hiperparámetros que controlan su complejidad, mientras que el *learning rate* o la paciencia de validación controlan el entrenamiento. Algoritmos similares a los utilizados para la selección de características sirven para escoger los valores de los hiperparámetros (sección 3.4.3).

3.1.6. Normalización de los datos

En una gran cantidad de ocasiones, los modelos ML funcionan mejor si se normalizan las distintas características de los datos de entrada y/o los datos de salida. La variable normalizada z correspondiente a una variable a, siendo $\{a_i\}_{i=1}^{N}$ sus valores para los N datos del dataset, se puede obtener mediante:

- El uso de *zscores*, obteniéndose una variable con media 0 y desviación estándar 1:

$$z_i = \frac{a_i - \mu(a)}{\sigma(a)} \tag{3.2}$$

  donde $\mu(a)$ es el valor medio de la variable a y $\sigma(a)$ su desviación típica.
- El escalado al rango $[0, 1]$:

$$z_i = \frac{a_i - \text{mín}\,(a)}{\text{máx}\,(a) - \text{mín}\,(a)} \tag{3.3}$$

  donde $\text{máx}\,(a)$ es el valor máximo que toma la variable a y $\text{mín}\,(a)$ el mínimo.

3.2. Redes neuronales FFN (*feed-forward networks*)

Las redes neuronales (NN, *Neural Networks*) son uno de los modelos ML más potentes y versátiles. Están inspiradas en el funcionamiento del cerebro humano y tratan de imitar su proceso de aprendizaje. El cerebro humano está formado por células denominadas neuronas, conectadas entre ellas mediante extensiones denominadas axones y dendritas. Las regiones de conexión entre los axones de una neurona y las dendritas de otra se denominan sinapsis. Cuando un humano recibe un estímulo externo, la fuerza de algunas de sus conexiones sinápticas se modifica en respuesta a dicho estímulo. Esta modificación

de las intensidades de las conexiones sinápticas constituye el proceso de aprendizaje, y la estructura de neuronas interconectadas se conoce como red neuronal natural [10].

Las redes neuronales artificiales (redes neuronales a partir de ahora) están formadas por una multitud de unidades computacionales denominadas neuronas, conectadas entre ellas mediante valores numéricos denominados pesos que simulan la fuerza de las conexiones sinápticas. Partiendo de un conjunto de entradas, las redes neuronales calculan una salida en base a los pesos. Las redes neuronales están organizadas en capas, de manera que las entradas se propagan desde una capa de entrada hasta una neurona de salida (en el caso de un problema de regresión unidimensional la salida es un único valor numérico) a través de una serie de capas ocultas (*hidden layers*). El proceso de aprendizaje se basa en modificar los pesos según un algoritmo que tiende a disminuir el error entre la salida obtenida y la salida objetivo para los datos de entrenamiento [10].

En este trabajo, se utilizan redes neuronales FFN (*feed-forward networks*), que son aquellas en las que la información fluye desde la capa de entrada hasta la salida en una única dirección [11], sin bucles de realimentación.

3.2.1. Redes neuronales de una capa: el perceptrón

El perceptrón es la red neuronal más sencilla que existe. Consta de una capa de entrada con tantas neuronas como variables de entrada al modelo ML (n) y una capa de salida con una única neurona [10]. Las neuronas de la capa de entrada transmiten las entradas a la siguiente capa sin realizar ninguna computación. Por este motivo, la capa de entrada no se incluye en la cuenta del número total de capas de la red.

La neurona de salida computa una combinación lineal de las entradas ($\vec{x}$, vector columna $n \times 1$) con los pesos de las conexiones sinápticas (W, fijados durante el entrenamiento) y un término constante denominado *bias* (b, fijado también durante el entrenamiento):

$$a = W\vec{x} + b \tag{3.4}$$

Esta combinación lineal se pasa por una función f denominada función de activación [10], que en general es no lineal. Así, la salida del perceptrón es:

$$\hat{y} = f(a) = f(W\vec{x} + b) \tag{3.5}$$

Como la capa computacional tiene una única neurona, únicamente se está computando una combinación lineal en dicha capa. Por tanto, W es una matriz fila $1 \times n$ y b es un único valor numérico. El acento circunflejo en $\hat{y}$ se utiliza para hacer hincapié en que el valor de salida es una predicción y no necesariamente el valor real.

En la figura 3.1 se muestran dos representaciones esquemáticas del perceptrón. En la parte izquierda se muestra una representación por componentes para el caso de 5 variables de entrada, en la que puede verse la capa de entrada y la neurona de salida con los pesos w_i y el término *bias* b. El símbolo de suma hace referencia a la combinación lineal calculada en la neurona, y el símbolo de escalón hace referencia a la función de activación. En la parte derecha se muestra una representación vectorial para un perceptrón genérico, con entradas $\vec{x}$, pesos W, término *bias* b y función de activación f.

El perceptrón no es más que una generalización de un modelo de regresión lineal en el que una función lineal de las entradas se transmite a la salida a través de una función f que habitualmente es no lineal. Visto así, parece que no aporta gran cosa frente a un modelo de regresión lineal, pero la verdadera potencia se obtiene combinando varias capas de computación de varias neuronas una detrás de otra.

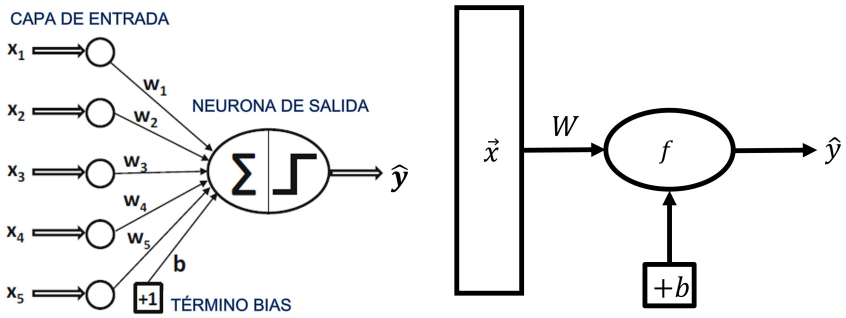


Figura 3.1: Representaciones esquemáticas del perceptrón. *Izquierda* [10]: representación por componentes para el caso de 5 variables de entrada. *Derecha*: representación vectorial para un perceptrón genérico.

3.2.2. Redes neuronales de varias capas

Las redes neuronales multicapa (*multilayer* NNs) son aquellas con más de una capa de neuronas. Las capas situadas entre la de entrada y la de salida se denominan capas ocultas (*hidden layers*) [10]. En la figura 3.2 se muestra una arquitectura típica, en la que todos los nodos de la k-ésima capa están conectados con todos los nodos de la siguiente capa $(k+1)$ por medio de pesos $W^{(k)}$ (matriz). Las redes que se utilizarán seguirán esta arquitectura. Además, todas las neuronas de la capa k incluyen un término *bias*, y dichos *bias* constituyen el vector columna $\vec{b}^{(k)}$. Se utiliza la notación $W^{(\text{out})}$ y $b^{(\text{out})}$ (un único valor numérico en un problema de regresión unidimensional) para los pesos y el *bias* de la capa de salida.

En la parte izquierda de la figura 3.2 se muestra una representación escalar de una FFN con cinco variables de entrada, dos capas ocultas de tres neuronas cada una y una neurona de salida (regresión unidimensional). Las flechas que unen las neuronas de las distintas capas incluyen implícitamente los pesos de las conexiones, y todas las neuronas tienen un término *bias*. En la parte derecha, se puede ver una representación vectorial de una red neuronal con n variables de entrada, dos capas ocultas con m_1 y m_2 neuronas y una neurona de salida. En las conexiones entre capas se indican los pesos $W^{(k)}$, los *bias* $\vec{b}^{(k)}$ y las funciones de activación f_k. Las dimensiones de las matrices $W^{(k)}$ y de los vectores $\vec{b}^{(k)}$ quedan implícitamente definidas fijado el número de neuronas de cada capa.

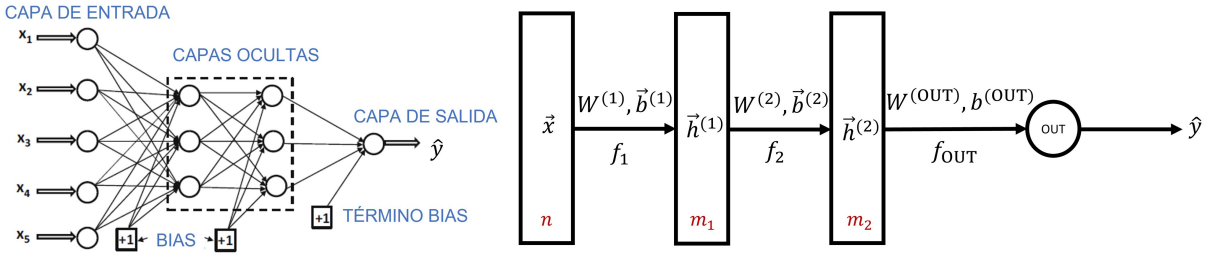


Figura 3.2: *Izquierda* [10]: representación escalar de una FFN con cinco variables de entrada, dos capas ocultas de tres neuronas cada una y una neurona de salida. *Derecha*: representación vectorial de una red neuronal de con n variables de entrada, dos capas ocultas con m_1 y m_2 neuronas y una neurona de salida.

Las neuronas de cada capa computan combinaciones lineales de las salidas de la capa anterior con los pesos especificados en las filas de $W^{(k)}$ y con los *bias* de $\vec{b}^{(k)}$, a las que se aplica la función de activación f_k. Así, las ecuaciones que definen una red FFN con $k = 1, 2, ..., p$ capas ocultas de m_k neuronas son:

$$\begin{aligned}\vec{h}^{(1)} &= f_1\left(\vec{a}^{(1)}\right) = f_1\left(W^{(1)}\vec{x} + \vec{b}^{(1)}\right)\\ \vec{h}^{(k)} &= f_k\left(\vec{a}^{(k)}\right) = f_k\left(W^{(k)}\vec{h}^{(k-1)} + \vec{b}^{(k)}\right)\\ \hat{y} &= f_{\text{out}}\left(a^{(\text{out})}\right) = f_{\text{out}}\left(W^{(\text{out})}\vec{h}^{(p)} + b^{(\text{out})}\right)\end{aligned} \tag{3.6}$$

donde se considera que una función aplicada a un vector es la aplicación de dicha función a cada uno de sus componentes. Para describir de manera compacta la topología o arquitectura de una red, se utiliza la siguiente notación: input $= n$, hidden $= [m_1, m_2, ..., m_k, ..., m_p]$, donde n es el número de variables de entrada, p el número de capas ocultas y m_k el número de neuronas en la capa k. Se sobrentiende que hay una única neurona de salida, por tratarse de un problema de regresión unidimensional.

Una red neuronal de una única capa oculta con función de activación no lineal y una capa de salida con función de activación lineal puede modelar una función de los datos de entrada tan complicada como se desee si se incrementa suficientemente el número de neuronas. La cantidad de neuronas requerida suele ser, sin embargo, muy grande. Las redes multicapa permiten reducir el número global de neuronas y pesos en la red, ya que utilizan un menor número de neuronas por capa. Las claves de la potencia de las arquitecturas multicapa son [10]:

- La no linealidad debida al uso de funciones de activación no lineales en cada capa interna.
- La capacidad de aprender funciones más complejas a partir de la composición de funciones más simples computadas en las sucesivas capas.

3.2.3. La función de activación

Como se acaba de comentar, la potencia de las redes multicapa se debe a la no linealidad de las funciones de activación. Esto se debe a que las funciones no lineales pueden transformar un conjunto de datos que no es linealmente separable en otro que sí lo es [10]. Por otra parte, se comentó en la sección 3.1.6 que los modelos ML suelen funcionar mejor con variables de entrada normalizadas. También parece conveniente que las variables de salida de las capas ocultas estén normalizadas. Por este motivo, las funciones de activación no lineales más utilizadas suelen tener salidas comprendidas entre $[-1, 1]$ o entre $[0, 1]$.

Existe una gran cantidad de funciones de activación. Las no lineales se utilizan en las capas ocultas de las redes, mientras que las lineales se utilizan en la neurona de salida de las redes de regresión. Algunas funciones de activación son (figura 3.3):

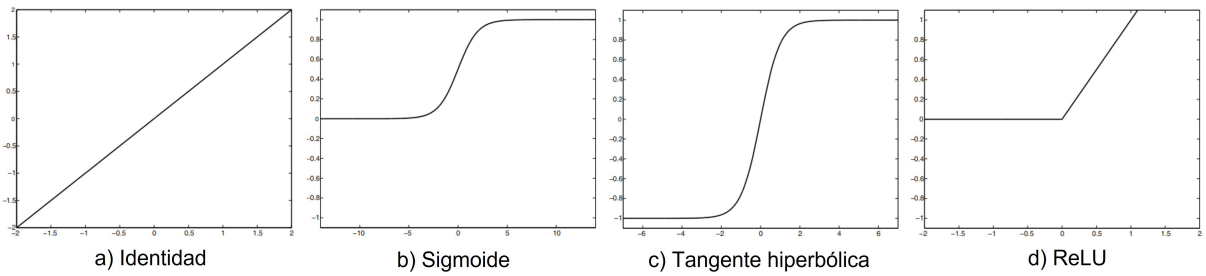


Figura 3.3: Algunas funciones de activación: a) Identidad, b) Sigmoide, c) Tangente hiperbólica, d) ReLU [10].

- Identidad (neurona de salida):

$$f(a) = a \tag{3.7}$$

- Sigmoide en (neuronas de las capas ocultas, 'logsig' en MATLAB):

$$f(a) = \frac{1}{a + \exp(-a)} \tag{3.8}$$

  Tiene tres propiedades que la hacen interesante como función de activación: su salida está acotada al intervalo $(0, 1)$, su gradiente es máximo en valores próximos a cero y satura para valores alejados del cero. Una combinación lineal de un número suficiente de unidades sigmoides en una única capa puede aproximar cualquier función [10].
- Tangente hiperbólica (neuronas de las capas ocultas, 'tansig' en MATLAB):

$$f(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}} \tag{3.9}$$

  Es similar a la sigmoide, salvo un reescalado horizontal y que su salida está acotada a $(-1, 1)$.
- ReLU o *Rectified Linear Unit* (neuronas de las capas ocultas, 'poslin' en MATLAB):

$$f(a) = \begin{cases} 0 & a < 0 \\ a & a \geq 0 \end{cases} \tag{3.10}$$

  Esta función facilita y acelera el proceso de entrenamiento en redes multicapa.

3.2.4. Entrenamiento de la red: el algoritmo *backpropagation*

El entrenamiento de una red comienza asignando valores aleatorios a sus pesos y *bias*. El proceso de entrenamiento subsiguiente es iterativo, y tiene como objetivo minimizar una función error, como el MSE, mediante la modificación de los pesos y *bias* de la red. El principal problema es que, si la red produce una salida incorrecta, no hay manera a priori de saber qué neuronas son las culpables ni cómo se han de modificar sus parámetros [12].

La solución a este problema viene dada por el algoritmo *backpropagation*. Si la función error es derivable con respecto a las salidas y las funciones de activación utilizadas son derivables, entonces la función error es derivable con respecto a los pesos (y los *bias*). Conociendo las derivadas del error con respecto a los pesos, es posible utilizar técnicas de optimización como el descenso del gradiente para actualizar el valor de los pesos [12].

Así, el entrenamiento de una red es un proceso iterativo en el que cada iteración consta de dos partes:

1. Se calculan las derivadas de la función error con respecto a los pesos (y *bias*) [12]. Se comienza calculando las salidas de la red para los datos de entrenamiento (*forward-propagation*) y, con ellas, el MSE con respecto a los valores objetivo. A continuación, se utiliza el algoritmo *backpropagation* para propagar el error desde la salida hacia las neuronas internas de la red, por medio de la regla de la cadena de las derivadas.
2. Se utilizan las derivadas calculadas en la fase anterior para actualizar los pesos de manera que se reduzca el error. Para esta fase, hay una gran cantidad de algoritmos de optimización. Uno de los más utilizados es el "descenso del gradiente" [12].

Para actualizar los pesos, existen dos posibilidades. La primera, denominada *on-line learning* consiste en actualizar los pesos tras el cálculo de las derivadas para una entrada de los datos de entrenamiento. Se denomina iteración a la actualización realizada para cada entrada individual de los datos de entrenamiento, y *epoch* al conjunto de actualizaciones asociadas a una exposición completa a todos los datos de entrenamiento. La segunda posibilidad es utilizar *batch learning*, que consiste en actualizar los pesos tras sumar las

derivadas para cada una de las entradas de todos los datos de entrenamiento. En este caso, una única actualización de los pesos ya constituirá una *epoch*.

En el apéndice B se desarrollan en detalle las matemáticas de *backpropagation* y del descenso del gradiente.

3.3. Redes ELM (*Extreme Learning Machine*)

Las redes ELM (*Extreme Learning Machine*, Máquina de Aprendizaje Extremo) son un tipo de FFN de una única capa oculta. Este enfoque consiste en generar aleatoriamente los pesos de las neuronas ocultas y calcular los de la capa de salida imponiendo que las salidas de la red para todos los datos de entrenamiento coincidan con su *target* [13].

La topología de la red consiste en una capa de entrada de n neuronas, una capa oculta de m neuronas con función de activación no lineal f (sigmoide, por ejemplo) y una neurona de salida (en nuestro caso) con activación lineal (unitaria). Los pesos de cada neurona de la capa oculta se recogen en las filas de la matriz W y los de la neurona de salida en el vector fila β. Los *bias* de la capa oculta constituyen el vector columna $\vec{b}$. Las salidas de la única capa oculta se representan con el vector columna $\vec{h}$. Así, las ecuaciones de la red son:

$$\begin{aligned} \vec{h} &= f\left(W\vec{x} + \vec{b}\right) \\ \hat{y} &= \beta\vec{h} \end{aligned} \tag{3.11}$$

Para entrenar una red ELM, se comienza asignando valores aleatorios a los pesos W y *bias* $\vec{b}$ de la capa oculta. Con dichos valores, se calculan las salidas $\vec{h}^{(i)}$ de la capa oculta para cada dato i del subconjunto de datos de entrenamiento (N datos en total). Con los vectores columna obtenidos para cada dato, se forma una matriz $H = \left[\vec{h}^{(1)} \quad \vec{h}^{(2)} \quad ... \quad \vec{h}^{(N)}\right]$. Por otra parte, se forma una matriz fila $T = \left[y^{(1)} \quad y^{(2)} \quad ... \quad y^{(N)}\right]$ con los *targets* $y^{(i)}$ de los datos de entrenamiento. A continuación, se exige que las salidas de la red para los datos de entrenamiento sean los *targets*, es decir, que:

$$\beta H = T \tag{3.12}$$

Para ello, los pesos de la capa de salida β han de tomar los siguientes valores [13]:

$$\beta = TH^{\dagger} \tag{3.13}$$

donde $H^{\dagger}$ es la inversa generalizada de Moore-Penrose (sección 3.3.1) de la matriz H [13][1].

La principal ventaja de las redes ELM es la rapidez de su entrenamiento. En lugar de un entrenamiento iterativo, como en las redes FFN, basta con realizar un cálculo matricial.

3.3.1. La inversa generalizada de Moore-Penrose

Las matrices no cuadradas, como H, no tienen una operación de inversión H^{-1} bien definida. Existe, sin embargo, una matriz $H^{\dagger}$ llamada pseudoinversa o inversa generalizada de Moore-Penrose que puede utilizarse como aproximación de una hipotética H^{-1} [1].

Un método para obtener la pseudoinversa se basa en la descomposición en valores singulares (SVD), según la cual una matriz H de tamaño $m \times n$ se descompone como [1]:

[1]Nótese que, para mantener la coherencia con el resto del trabajo, la notación utilizada es ligeramente distinta a la habitual en redes ELM (las matrices están traspuestas con respecto a la notación habitual).

$$H = UDV^T \tag{3.14}$$

- U es una matriz $m \times m$ ortogonal, y sus columnas se llaman vectores singulares de salida de H. Dichos vectores se calculan como los vectores propios de HH^T.
- D es una matriz $m \times n$ que sólo tiene elementos no nulos en su diagonal, denominados valores singulares de H. Los valores singulares de H son las raíces cuadradas de los valores propios de HH^T o H^TH.
- V es una matriz $n \times n$ ortogonal, y sus columnas se llaman vectores singulares de entrada de H. Dichos vectores se calculan como los vectores propios de H^TH.

Teniendo en cuenta SVD, la pseudoinversa puede calcularse de forma directa a partir de la generalización de la propiedad $(ABC)^{-1} = C^{-1}B^{-1}A^{-1}$ de la inversión de matrices. Aplicando esta propiedad a la ecuación 3.14:

$$H^\dagger = \left(UDV^T\right)^\dagger = \left(V^T\right)^{-1} D^\dagger U^{-1} = VD^\dagger U^T \tag{3.15}$$

donde la pseudoinversa $D^\dagger$ de la matriz diagonal D se calcula invirtiendo sus elementos no nulos y trasponiendo el resultado [1]. En la última igualdad se ha tenido en cuenta que las matrices U y V son ortogonales, es decir, que $U^{-1} = U^T$ y $V^{-1} = V^T$.

La justificación de que $H^\dagger$ es una aproximación de una hipotética H^{-1} se basa en que si la matriz H tiene mayor número de columnas que de filas, la ecuación matricial $H\vec{x} = \vec{y}$ tiene múltiples soluciones, siendo $\vec{x} = H^\dagger\vec{y}$ la de menor norma euclídea $||\vec{x}||$. Si H tiene mayor número de filas que de columnas, la ecuación matricial no siempre tiene solución, pero $\vec{x} = H^\dagger\vec{y}$ es el vector que minimiza la norma euclídea $||H\vec{x} - \vec{y}||$ [1].

Se verifican resultados análogos considerando que $\vec{x}$ y $\vec{y}$ son vectores fila y la ecuación matricial $\vec{x}H = \vec{y}$, intercambiando las palabras filas por columnas (y viceversa).

3.4. Algoritmos genéticos

En esta sección se realiza una introducción a los algoritmos genéticos (sección 3.4.1). A continuación, se explica cómo este conjunto de técnicas ML puede aplicarse a la selección de características (sección 3.4.2) y a la optimización de hiperparámetros (sección 3.4.3).

3.4.1. Introducción a los algoritmos genéticos

Los algoritmos genéticos son un conjunto de técnicas ML de optimización inspiradas en la teoría de la evolución de Darwin y en la genética molecular. En este trabajo, se utilizan indistintamente los términos algoritmo genético y evolutivo, aunque este último hace normalmente referencia a un conjunto más amplio de algoritmos. Según los principios de la genética molecular, un determinado ser vivo está caracterizado por su genotipo (su conjunto de genes), que codifica su fenotipo (el conjunto de características físicas observables a nivel macroscópico). Cada gen puede tomar diferentes valores o alelos y, en función de ellos el ser vivo en cuestión tendrá un fenotipo u otro. Según la teoría de la evolución de Darwin, si se considera una población de individuos distintos en un entorno concreto, hay algunos con un fenotipo más favorable para la supervivencia y reproducción, y otros con mayores dificultades para sobrevivir y reproducirse. A medida que transcurre el tiempo, la población evoluciona hacia genotipos que codifican fenotipos más aptos para el entorno considerado. Esto es lo que se conoce como selección natural: los individuos "más adaptados" sobreviven y se reproducen, transmitiendo su genotipo a la generación siguiente, mientras que los "menos adaptados" mueren y su genotipo se va perdiendo [14].

Inspirados en esta analogía biológica, surgen los algoritmos evolutivos. En el caso en el que el fenotipo se codifica en forma de genes, estos algoritmos se conocen también como algoritmos genéticos. En este trabajo, los individuos a optimizar son redes neuronales, y la función objetivo a minimizar es el MSE para los datos de *test*. Con respecto a los genes (que no son más que los parámetros del modelo neuronal que el algoritmo evolutivo toma como variables independientes en el proceso de optimización), se consideran dos casos:

- Algoritmos de selección de características: los genes son binarios. Para cada variable del *dataset*, un 1 indica su inclusión como predictor en el modelo de regresión, mientras que un 0 indica su exclusión. El objetivo del algoritmo es encontrar un modelo que incluya como predictores el subconjunto de variables que minimice MSE_{test}.
- Algoritmos de optimización de hiperparámetros: los genes son números enteros o reales, o bien variables categóricas (en este trabajo sólo se consideran genes enteros). Cada gen es un hiperparámetro (sección 3.1.5) del modelo de regresión cuyo MSE_{test} se desea minimizar.

Nótese que el mínimo encontrado por un algoritmo evolutivo no tiene por qué ser un mínimo global, pero puede ser suficientemente bueno para la aplicación considerada [14].

Funcionamiento de un algoritmo genético

Los algoritmos genéticos se basan en dos pilares: la variación de los genotipos y la selección de individuos según la función objetivo. En primer lugar, se inicializa una población de modelos ML con valores aleatorios para sus genes, y se evalúa la eficiencia de cada uno (MSE_{test}). A continuación, comienza el siguiente proceso iterativo [14], en el que la k-ésima iteración constituye una población de individuos llamada k-ésima generación:

1. Selección de los padres de futuros individuos en base a su eficiencia.
2. Recombinación de los genes de los padres para dar lugar a individuos hijos.
3. Mutación con cierta probabilidad de los hijos resultantes del paso anterior.
4. Evaluación de los nuevos candidatos en base a su eficiencia.
5. Selección de los individuos que formarán parte de la siguiente generación.

Componentes de un algoritmo genético

Los componentes fundamentales de un algoritmo genético son [14]:

- Representación o codificación de los genes (binarios, enteros, reales, etc.).
- Función objetivo (por ejemplo, MSE_{test} a minimizar).
- Población: conjunto de individuos en una generación.
- Mecanismo de selección de los progenitores, los correspondientes operadores se aplican a nivel de población. Este mecanismo proporciona una mayor probabilidad de reproducirse a los individuos con mejor valor de la función objetivo.
- Operadores de mutación: operan sobre un individuo resultante del cruce de dos progenitores (hijo). Es un operador que, de forma aleatoria y con una pequeña probabilidad, modifica el valor de algunos de los genes del hijo.
- Operadores de recombinación o cruce: operan sobre dos individuos, los progenitores, para dar uno o más descendientes. El operador, que puede ser aleatorio o determinista, selecciona qué genes de cada progenitor tendrán los hijos. Cruzando dos individuos con buen valor de la función objetivo es esperable obtener eventualmente algún descendiente que mejore dicho valor.

- Mecanismo de selección de supervivientes: opera a nivel de población. Se aplica una vez generados los descendientes de una generación para decidir qué individuos pasarán a formar parte de la siguiente generación. Existen distintas estrategias, algunas de ellas son deterministas (se cogen los individuos con mejor valor de la función objetivo) y otras tienen algún componente aleatorio. Algunas estrategias también dan mayor peso a los individuos más jóvenes.

En la parte izquierda de la figura 3.4 se muestra un posible operador de cruce para un genotipo binario. Las líneas rojas (pivotes) dividen el genotipo en diferentes regiones, y el hijo contiene en cada región la copia del genotipo de uno de sus progenitores de manera alterna. En la parte derecha se puede ver un posible operador de mutación para genotipo binario. De manera aleatoria (según una tasa de mutación), algunos de los genes del hijo (marcados en rojo) cambian su valor. Existen otros tipos de operadores de variación. Para información más detallada puede consultarse el libro de Eiben y Smith [14].

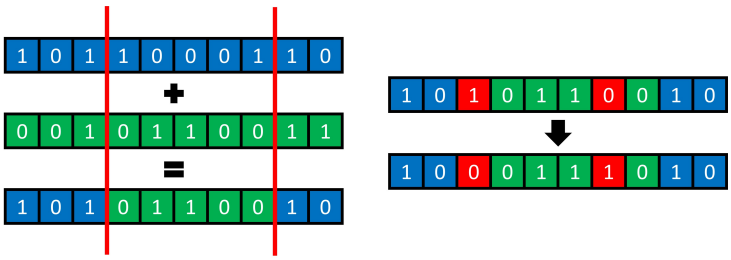


Figura 3.4: Ejemplo de operadores de variación para una representación binaria. *Izquierda*: operador de cruce de dos pivotes. *Derecha*: operador de mutación.

En teoría, si el operador mutación utilizado permite que partiendo de un genotipo dado se pueda obtener cualquier otro (por muy pequeña que sea la probabilidad), entonces el algoritmo evolutivo en cuestión es capaz de encontrar el óptimo global de la función objetivo si se deja correr suficiente tiempo [14].

3.4.2. Selección de características con algoritmos genéticos

Los algoritmos genéticos para selección de características tienen como fin encontrar el subconjunto de predictores del *dataset* que da lugar al modelo ML de regresión con menor error de *test*. Así, cada individuo es un modelo de regresión, y la función objetivo a optimizar es MSE_{test} (o el promedio de MSE_{test} para unos pocos entrenamientos del modelo). El genotipo del modelo consiste en qué predictores incorpora y cuáles no.

La representación utilizada es binaria: se utiliza un vector cuya longitud coincide con el número de predictores en el *dataset*. Cada elemento del vector (gen) corresponde a un predictor del *dataset*. El valor 1 indica que dicho predictor se incluye como variable de entrada en el modelo de regresión, mientras que el valor 0 indica que no se incluye.

Los operadores de cruce y de mutación son similares a los que se muestran en la figura 3.4, aunque existen otras posibilidades [14]. Tras crear una serie de individuos de partida aleatorios y esperar unas cuantas generaciones, la distribución de genotipos que exhiba la población evolucionada indicará qué variables de predicción son más importantes para el problema de regresión que se desea resolver. De esta manera, los algoritmos genéticos proporcionan no sólo una mejora en el modelo de regresión ML, sino que también aportan intuición física sobre el problema que resuelve el modelo.

3.4.3. Optimización de hiperparámetros con algoritmos genéticos

En este caso, los individuos son un modelo de regresión (o el promedio de varios) y la función a optimizar sigue siendo un indicador de eficiencia (como el error). Las variables del algoritmo genético son los hiperparámetros del modelo. Para una red FFN hay algunos hiperparámetros enteros (como el número de neuronas en cada capa oculta), otros categóricos (como la función de activación de cada capa) y otros reales (como el *learning rate*). En este trabajo, a las variables categóricas y reales a optimizar se les asocia una variable entera para simplificar el algoritmo y reducir su tiempo de ejecución:

- A cada una de las n categorías de una variable categórica se le asigna un entero entre 1 y n.
- En el caso de las variables reales se escogen n posibles valores reales para la variable, y a cada uno se le asigna un entero entre 1 y n.

El algoritmo genético de optimización de hiperparámetros tiene un funcionamiento similar al que se ha explicado en la sección 3.4.1, con la salvedad de que los operadores de cruce y mutación ahora actúan sobre genes enteros (en lugar de binarios).

Los operadores de mutación para genes enteros mutan cada gen de manera independiente según una tasa de mutación (probabilidad p), y se subdividen en dos grupos [14]:

- Mutación aleatoria: para cada gen, en base a una probabilidad p, se escoge un nuevo valor de manera aleatoria de entre todos los posibles valores del gen. Este operador es adecuado para variables enteras que representan categorías.
- Mutación *creep*: consiste en sumar una pequeña cantidad (positiva o negativa) a cada gen con una probabilidad p. Los valores a sumar se escogen aleatoriamente de una distribución centrada en cero, por ejemplo, una distribución binomial. Este tipo de operadores es adecuado para genes que son enteros *per se*.

Respecto a los operadores de recombinación, lo normal suele ser utilizar el mismo tipo de operadores que para las representaciones de genes binarias, por ejemplo, el operador basado en pivotes mostrado en la parte izquierda de la figura 3.4 [14].

4. Experimentos y resultados obtenidos

En esta sección se aplican las técnicas explicadas en la sección 3 al desarrollo de un modelo de regresión para predecir la temperatura crítica de un superconductor (T_c) partiendo de su fórmula empírica. En la sección 4.1 se presenta el *dataset* utilizado y los 81 predictores del modelo. En la sección 4.2 se realiza una selección de predictores en base al coeficiente de correlación.

En las secciones 4.3 y 4.4 se entrenan, respectivamente, modelos FFN y ELM con los 52 predictores seleccionados. Se descarta utilizar ELM en base a los resultados obtenidos. En las secciones 4.5 y 4.7 se realiza una selección de características con algoritmos genéticos utilizando FFN como modelo de regresión (dos ejecuciones consecutivas del algoritmo de selección). Las redes FFN con el número de predictores resultante tras la selección (38 y 35, respectivamente) se analizan en las secciones 4.6 y 4.8, respectivamente.

A continuación, se realiza una optimización de los hiperparámetros del modelo FFN con 35 predictores utilizando, para ello un algoritmo genético (sección 4.9). La red FFN resultante se estudia en la sección 4.10.

Finalmente, en la sección 4.11 se realiza una recapitulación de los distintos modelos entrenados a lo largo de la sección 4 y se realiza una comparativa entre sus eficiencias.

4.1. Descripción del *dataset* utilizado

El *dataset* utilizado para el desarrollo de los modelos ML para la predicción de T_c fue creado por Kam Hamidieh siguiendo los pasos descritos en la referencia [3]. Está basado en la base de datos "Superconducting Material Database" mantenida por el Instituto Nacional Japonés de Ciencia de Materiales (NIMS). El *dataset* contiene 21263 entradas o muestras correspondientes a distintos superconductores. Para cada entrada, hay 81 variables o predictores (cada una en una columna) relacionadas con la fórmula empírica del superconductor. La última columna contiene las temperaturas críticas T_c (en Kelvin), que son los *targets* de los modelos desarrollados. El fichero con los datos se llama *train.csv*.

La primera columna contiene el número total de elementos (distintos) en el superconductor. Las otras 80 columnas corresponden a características extraídas a partir de las siguientes 8 propiedades de los elementos atómicos: masa atómica (*atomic_mass*), primera energía de ionización (*fie*), radio atómico (*atomic_radius*), densidad (*density*), afinidad electrónica (*electron_affinity*), calor de fusión (*fusion_heat*), conductividad térmica (*thermal_conductivity*) y valencia (*valence*). Nótese que el autor se ha inspirado en las reglas de Matthias (sección 2.2) para escoger estas propiedades [3].

Para cada una de las 8 propiedades de los elementos, se calculan 10 predictores considerando la fórmula empírica del superconductor (las 80 columnas restantes). Dichos predictores son: media (*mean*), media ponderada (*wtd_mean*), media geométrica (*gmean*), media geométrica ponderada (*wtd_gmean*), entropía (*entropy*), entropía ponderada (*wtd_entropy*), rango (*range*), rango ponderado (*wtd_range*), desviación estándar (*std*) y desviación estándar ponderada (*wtd_std*). Las expresiones para el cálculo de los predictores y un ejemplo pueden verse con detalle en el artículo de Kam Hamidieh [3].

Se decide eliminar cuatro entradas con T_c a más de 3 desviaciones típicas de la distribución de temperaturas críticas (*outliers*): H_2S_1 (185 K), $Hg_{0.66}Pb_{0.34}Ba_2Ca_{1.98}Cu_{2.9}O_{8.4}$ (143 K), $Hg_{0.7}Pb_{0.3}Ba_2Ca_2Cu_3O_8$ (143 K) y $Tl_{0.8}Hg_{0.2}Ba_2Ca_2Cu_3O_9$ (137.4 K). Así, el *dataset* resultante tiene 21259 superconductores. En la figura 4.1 se muestra el histograma de T_c de los superconductores del *dataset*. Como es de esperar, la mayor parte de da-

tos corresponden a T_c próximos al cero absoluto. La proporción de superconductores con $T_c > 95\,\mathrm{K}$ es pequeña. El valor medio es de $34.3991\,\mathrm{K}$ y la desviación típica de $34.2184\,\mathrm{K}$.

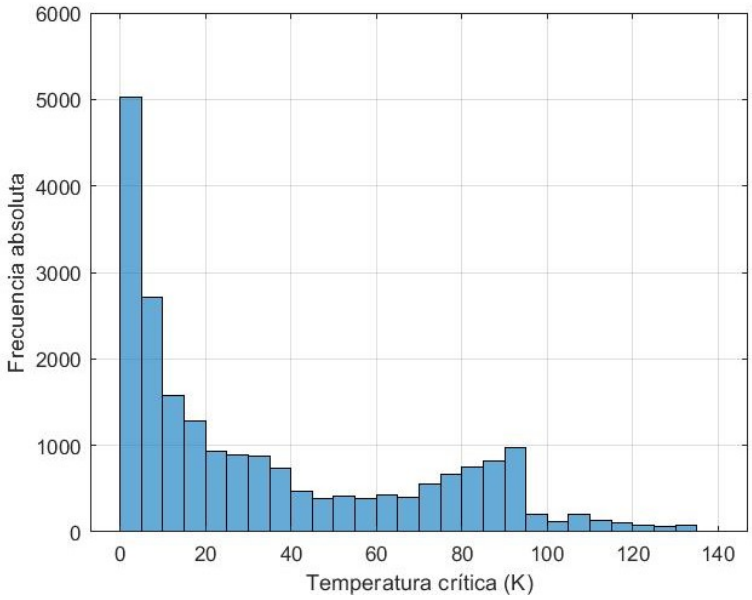


Figura 4.1: Histograma de temperaturas críticas de los superconductores del *dataset*.

Finalmente, cada una de las 81 características y los *targets* se normalizan utilizando *zscores* (ecuación 3.2). Se ha comprobado que este tipo de normalización da mejores resultados que el escalado al rango $[0, 1]$ (ecuación 3.3) mediante pruebas preliminares con unas pocas redes FFN. A partir de ahora se utiliza este tipo de normalización.

4.2. Selección de características con el coeficiente de correlación

Una vez preparado el *dataset*, se utiliza el coeficiente de correlación (apéndice A.2) para reducir el número de predictores. La matriz de correlación para las 81 variables es de tamaño 81×81. En lugar de trabajar con dicha matriz, se buscan correlaciones en las submatrices 10×10 correspondientes a los 10 predictores asociados a cada propiedad atómica. Como criterio para descartar una de dos variables entre a y b, se utiliza $R_{ab} \geq 0.9$.

Tabla 4.1: Submatriz de correlaciones para los predictores relacionados con la masa atómica

atomic_mass	mean	wtd_mean	gmean	wtd_gmean	entropy	wtd_entropy	range	wtd_range	std	wtd_std
mean	1.00	0.82	0.94	0.75	-0.10	-0.10	0.13	0.45	0.20	0.13
wtd_mean	0.82	1.00	0.85	0.96	-0.31	-0.41	-0.14	0.72	-0.06	-0.09
gmean	0.94	0.85	1.00	0.86	-0.19	-0.23	-0.18	0.46	-0.12	-0.17
wtd_gmean	0.75	0.96	0.86	1.00	-0.37	-0.48	-0.35	0.67	-0.27	-0.33
entropy	-0.10	-0.31	-0.19	-0.37	1.00	0.89	0.54	-0.28	0.36	0.41
wtd_entropy	-0.10	-0.41	-0.23	-0.48	0.89	1.00	0.62	-0.54	0.47	0.50
range	0.13	-0.14	-0.18	-0.35	0.54	0.62	1.00	-0.11	0.96	0.92
wtd_range	0.45	0.72	0.46	0.67	-0.28	-0.54	-0.11	1.00	-0.02	-0.06
std	0.20	-0.06	-0.12	-0.27	0.36	0.47	0.96	-0.02	1.00	0.92
wtd_std	0.13	-0.09	-0.17	-0.33	0.41	0.50	0.92	-0.06	0.92	1.00

En la tabla 4.1 se muestra la submatriz de correlaciones para las variables relacionadas con la masa atómica. Se muestran en verde los coeficientes $R \geq 0.90$, en amarillo $0.85 \leq R < 0.9$ y en azul $0.80 \leq R < 0.85$. Como ejemplo, se explica el proceso de eliminación de características redundantes para esta submatriz. Utilizando el criterio $R \geq 0.9$:

- *mean_atomic_mass* está correlacionado con *gmean_atomic_mass*.
- *wtd_mean_atomic_mass* con *wtd_gmean_atomic_mass*.
- *range_atomic_mass* con *std_atomic_mass* y con *wtd_std_atomic_mass*.

- *std_atomic_mass* con *wtd_std_atomic_mass*.

Así, de las 10 variables relacionadas con la masa atómica se seleccionan 6 eliminando redundancias por correlación: *mean_atomic_mass*, *wtd_mean_atomic_mass*, *entropy_atomic_mass*, *wtd_entropy_atomic_mass*, *wtd_range_atomic_mass* y *wtd_std_atomic_mass*.

Tabla 4.2: Variables seleccionadas (marcadas con "X") en base a los coeficientes de correlación.

	mean	***wtd_mean***	***gmean***	***wtd_gmean***	***entropy***	***wtd_entropy***	***range***	***wtd_range***	***std***	***wtd_std***
atomic_mass	X	X			X	X		X		X
fie	X	X			X	X		X		X
atomic_radius	X	X			X	X		X		X
density	X			X	X	X		X	X	
electron_affinity	X	X	X	X	X	X		X	X	X
fusion_heat		X	X		X	X		X		X
thermal_conductivity	X	X	X	X	X	X		X		X
valence				X		X		X		X

En la tabla 4.2 se muestran las variables seleccionadas de la manera explicada en los párrafos anteriores. A estas variables hay que añadir el número de elementos. En total, se han seleccionado 52 características de las 81 que se habían extraído inicialmente.

4.3. Redes FFN con las 52 características seleccionadas

Una vez seleccionados los 52 predictores mencionados, se ha procedido al entrenamiento de redes FFN (objeto "feedforwardnet" del "Deep Learning Toolbox" de MATLAB). Se ha optado por utilizar en un inicio los parámetros mostrados en la tabla 4.3, escogidos tras realizar pruebas preliminares. Para más información sobre estos u otros parámetros de la red consúltese la documentación de los objetos "feedforwardnet" de MATLAB [15].

Tabla 4.3: Parámetros utilizados para el entrenamiento de las redes FFN.

Parámetro	**Valor**	**Descripción**
hidden	[18 10]	Número de neuronas en cada capa oculta.
trainFcn	*'trainscg'*	Entrenamiento con *backpropagation* de gradiente conjugado escalado.
performFcn	*'mse'*	Error cuadrático medio como función error.
trainParam.lr	0.3	*Learning rate*, tasa de aprendizaje.
trainParam.max_fail	300	Máximo número de *epochs* durante el que continúa el entrenamiento si no disminuye MSE_{val}.
trainParam.min_grad	1e-10	Mínimo gradiente para que continúe el proceso de entrenamiento.
trainParam.epochs	10000	Número máximo de *epochs* en el entrenamiento.

Para acelerar el proceso de entrenamiento de las redes, se utiliza de ahora en adelante el valor 'true' para la opción 'UseParallel', que habilita la paralelización del entrenamiento entre los distintos núcleos de la CPU (*Central Processing Unit*) de la computadora utilizada. En los casos en los que se ha utilizado un ordenador que dispone de GPU (*Graphics Processing Unit*) también se ha usado el valor 'true' para la opción 'UseGPU'.

Para evaluar la eficiencia de las redes con estos parámetros y las 52 características, se han entrenado 20 redes con distintas divisiones de datos y distintos valores iniciales de los pesos, con el fin de evitar el sesgo. En la tabla 4.4 se muestran los valores medios de algunos indicadores de su eficiencia para los datos de entrenamiento, validación y *test*: error cuadrático medio para los *targets* normalizados (MSE_{norm}), raíz del error cuadrático medio en Kelvin (RMSE (K), véase apéndice C), coeficiente de correlación R y coeficiente

de determinación R^2. Evidentemente, los menores errores y mayores valores de R^2 se obtienen para los datos de entrenamiento. Los indicadores que realmente miden la eficiencia son los correspondientes a los datos de *test*. Para los datos de *test*, los valores promedio obtenidos son RMSE = 12.56 K y $R^2 = 0.8651$. Todavía hay margen de mejora hasta el umbral $R^2 \geq 0.9$ habitualmente considerado satisfactorio.

Tabla 4.4: Valores medios de algunos indicadores de eficiencia de entrenamiento, validación y *test* para 20 redes FFN con 52 predictores y los parámetros de la tabla 4.3.

	Entrenamiento			**Validación**			**Test**		
	Mejor	Peor	Medio	Mejor	Peor	Medio	Mejor	Peor	Medio
MSE_{norm}	0.0913	0.1274	0.1075	0.1172	0.1489	0.1319	0.1212	0.1567	0.1348
RMSE (K)	10.34	12.21	11.22	11.71	13.20	12.43	11.91	13.55	12.56
R	0.9535	0.9336	0.9448	0.9394	0.9217	0.9319	0.9381	0.9188	0.9301
R^2	0.9092	0.8716	0.8926	0.8825	0.8495	0.8684	0.8800	0.8442	0.8651

En la figura 4.2 se muestra el error a lo largo del proceso de entrenamiento de una de las 20 redes entrenadas para los datos de entrenamiento, validación y *test*. La línea discontinua representa la *epoch* con el menor error de validación, que se utiliza para monitorizar el comienzo del *overfitting*. Tras 300 *epochs* de paciencia se detiene el entrenamiento.

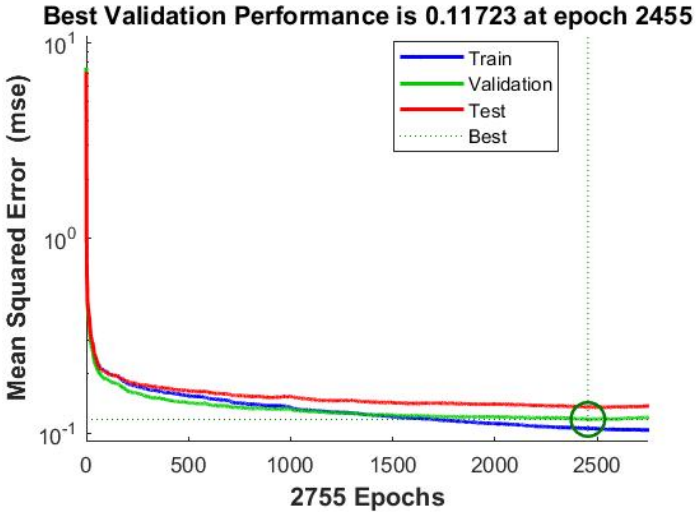


Figura 4.2: MSE_{train}, MSE_{val} y MSE_{test} (para los *targets* normalizados) frente al número de *epochs* para una de las redes con 52 predictores y los parámetros de la tabla 4.3

En la figura 4.3 se muestran las gráficas de regresión (salida del modelo frente a valor real) con los *targets* normalizados para la misma red que en la figura 4.2. Si el modelo de regresión fuera perfecto, se observarían todos los puntos sobre la diagonal señalada con puntos discontinuos en las gráficas. Se observa que, en general, este modelo tiende a predecir menores valores de T_c que los reales para los materiales con mayor T_c.

4.4. Redes ELM con las 52 características seleccionadas

Con vistas a reducir el tiempo de ejecución de los algoritmos evolutivos que se presentan más adelante, se prueba a entrenar redes ELM, ya que su tiempo de entrenamiento es considerablemente menor al de las redes FFN. Se realizan pruebas con entre 1 y 1000 neuronas en la única capa oculta, utilizando los 52 predictores seleccionados. Nótese que, como el entrenamiento de las redes ELM es un cálculo matricial, sin necesidad de realizar

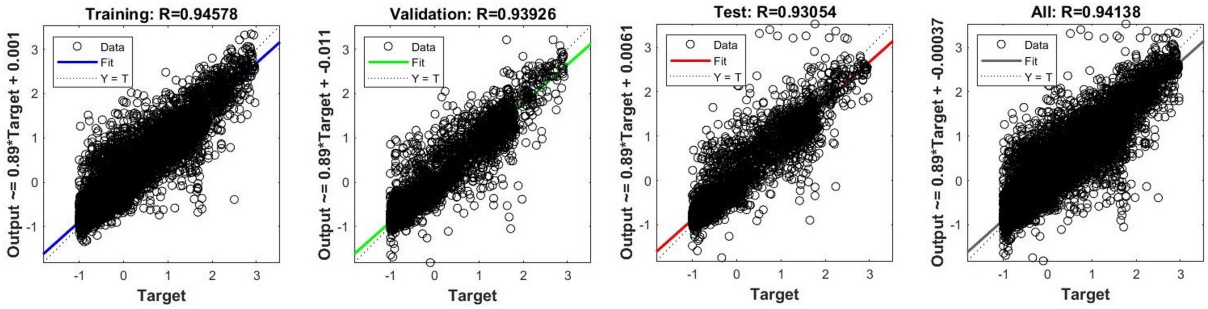


Figura 4.3: Gráficas de regresión para una red con 52 predictores y los parámetros de la tabla 4.3.

varias iteraciones, no es necesario el uso de un subconjunto de validación para monitorizar el *overfitting*. Se utiliza, así, una proporción del 70 % *training data* y 30 % *test data*.

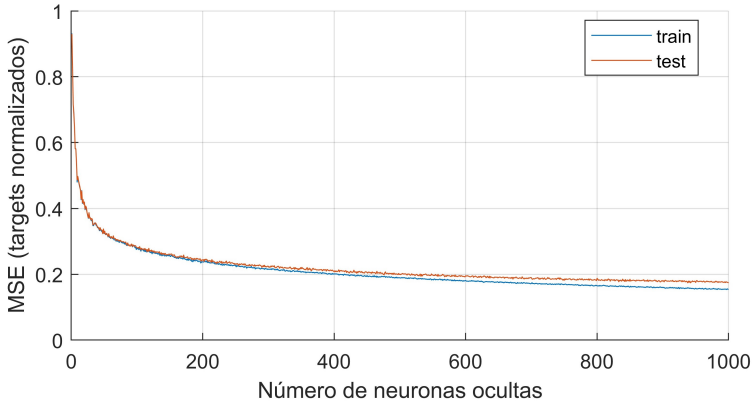


Figura 4.4: Promedio del MSE de entrenamiento y de test de 10 redes ELM en función de su número de neuronas ocultas, utilizando los 52 predictores seleccionados a partir de las correlaciones.

Para cada número de neuronas ocultas se entrenan 10 redes ELM y se toma el promedio de sus indicadores de eficiencia. En la figura 4.4 se muestra el promedio de $\text{MSE}_{\text{train}}$ y MSE_{test} en función del número de neuronas ocultas. El menor MSE_{test} promedio obtenido es de 0.1737, con 1000 neuronas ocultas. Se espera que un aumento del número de neuronas haga disminuir más el error, pero la figura 4.4 muestra que la disminución es lenta.

Tabla 4.5: Valores medios de algunos indicadores de eficiencia de entrenamiento y *test* para 20 redes ELM con 52 predictores y 1000 neuronas ocultas.

	Entrenamiento			**Test**		
	Mejor	Peor	Medio	Mejor	Peor	Medio
$\mathbf{MSE_{norm}}$	0.1511	0.1588	0.1549	0.1679	0.1856	0.1759
RMSE (K)	13.30	13.64	13.47	14.02	14.74	14.35
$\boldsymbol{R}$	0.9214	0.9171	0.9193	0.9124	0.9026	0.9080
$\boldsymbol{R^2}$	0.8490	0.8411	0.8451	0.8325	0.8147	0.8245

A continuación, se entrenan 20 redes ELM de 1000 neuronas ocultas. En la tabla 4.5 se muestran los valores medios de algunos indicadores de su eficiencia para los datos de entrenamiento y *test*: MSE para los *targets* normalizados (MSE_{norm}), RMSE en Kelvin, R y R^2. Los indicadores son peores que para las redes FFN entrenadas en la sección 4.3.

Aunque este modelo no sea el que mejores parámetros de eficiencia arroje, podría caber la posibilidad de que fuera válido para selección de características con algoritmos evolutivos, siempre que el error de *test* disminuya al escoger los subconjuntos de predictores más

importantes para las predicciones. Sin embargo, las pruebas realizadas en esta línea no logran una reducción sustancial del número de predictores, por lo que se abandona esta línea y se opta por utilizar redes FFN como individuos para el algoritmo evolutivo.

4.5. Selección de características con algoritmos genéticos (I)

En esta sección se explica cómo se consigue reducir el número de predictores utilizados por un modelo de regresión FFN de 52 a 38 por medio del uso de algoritmos genéticos. Se parte de una población en la que cada individuo es una red FFN que utiliza un subconjunto de los 52 predictores seleccionados en la sección 4.2. Los parámetros de dicha red son los de la tabla 4.3, salvo *trainParam.epochs* = 2500 y *trainParam.max_fail* = 200. Estas dos últimas elecciones tienen como objetivo reducir el tiempo de ejecución del algoritmo genético, ya de por sí largo. Cada individuo se entrena con una partición diferente (entrenamiento, validación, *test*) del *dataset*, con vistas a reducir el sesgo. Los individuos con menor MSE_{test} tienen mayor probabilidad de reproducirse y transmitir parte de su genotipo a la siguiente generación, según lo explicado en la sección 3.4. Así, aquellos predictores que no son relevantes en la predicción van desapareciendo de los genotipos, mientras que los que sí lo son permanecen en una gran proporción de los individuos de generaciones futuras.

Para implementar el algoritmo genético, se utiliza la función *ga* (*genetic algorithm*) del "Global Optimization Toolbox" de MATLAB. En la tabla 4.6 se muestran los valores de los parámetros con los que se llama a *ga*. El último de los parámetros, *options*, es un objeto creado a partir de la función *optimoptions* [16] mediante el cual se especifican algunas opciones del algoritmo. Los valores escogidos para dichas opciones se recogen en la tabla 4.7, y para el resto se utiliza el valor por defecto. Para más información sobre la función *ga* consúltese su documentación en la página web de MathWorks [16].

Tabla 4.6: Parámetros de *ga* utilizados para el algoritmo evolutivo de selección de características.

Parámetro	**Valor**	**Descripción**
fun	*@miffn*	Función minimizada por el algoritmo genético. Toma como entrada el genotipo de un individuo (qué predictores utiliza y cuáles no), entrena una red FFN y devuelve su MSE_{test}.
nvars	51	Número de variables de la función optimizada por el algoritmo.
options	-	Opciones del algoritmo genético, creadas con la función *optimoptions* y recogidas en la tabla 4.7.

Al lanzar este algoritmo, se cometió el error de fijar $nvars = 51$, en lugar de los 52 predictores de partida. La ejecución del algoritmo duró prácticamente dos meses, y volverlo a correr llevaría un tiempo excesivo. Por este motivo, se decidió simplemente añadir el predictor que quedó fuera al conjunto de predictores seleccionados por el algoritmo y volver a lanzar otro genético con dicho conjunto de variables reducido (sección 4.7).

En la parte izquierda de la figura 4.5 se muestra el número de predictores medio y el mejor número de predictores (el que menor error arroja) para las sucesivas generaciones del algoritmo genético. En la parte derecha se muestra la evolución del valor medio y el menor valor de MSE_{test} para cada generación. El algoritmo se detiene tras 205 generaciones, al observarse convergencia. El valor medio tras la convergencia del MSE_{test} está en torno a 0.138, mientras que el número de características medio es aproximadamente 35.

En la figura 4.6 se muestra el histograma del número de individuos que incluyen cada uno de los 51 predictores tras la finalización del algoritmo genético. Se observa que hay

Tabla 4.7: Opciones de *ga* utilizadas para el algoritmo evolutivo de selección de características.

Opción	Valor	Descripción
'PopulationType'	*'bitstring'*	Genotipo en cadena de bits.
'OutputFcn'	*@ga_save_each_gen* [17]	Función llamada en cada generación, utilizada para guardar los genotipos y las eficiencias de los individuos.
'EliteCount'	Por defecto	El 5 % de los individuos con mayor eficiencia pasan a la siguiente generación.
'FitnessLimit'	Por defecto	El algoritmo se detiene cuando la función objetivo (MSE_{test}) se hace $-\infty$.
'MaxGenerations'	Por defecto	Máximo número de iteraciones o generaciones: $100 \cdot$ (número de variables).
'PopulationSize'	Por defecto	Con más de 5 variables binarias, cada generación tiene 200 individuos.

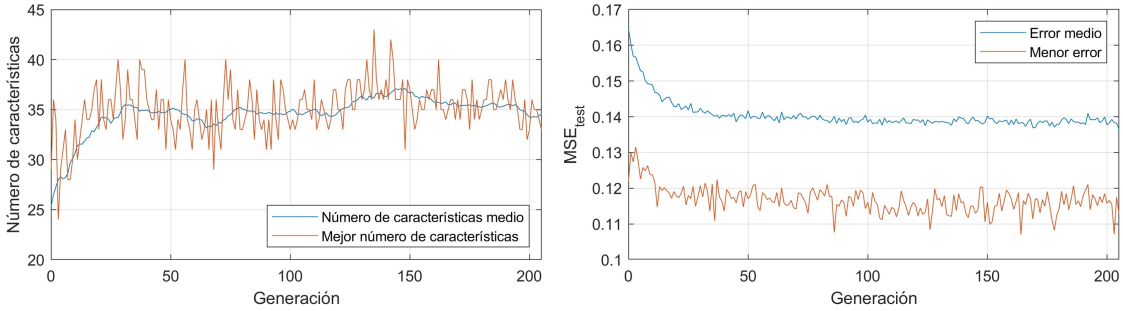


Figura 4.5: *Izquierda*: número de características medio y mejor número de características para cada generación del algoritmo genético. *Derecha*: valor medio y menor valor de MSE_{test} para cada generación.

algunos predictores utilizados por prácticamente todos los individuos, mientras que otros apenas están presentes. Como criterio (arbitrario) para la selección de características, se eliminan los predictores presentes en menos de 1/3 de los individuos de la población. En la tabla 4.8 se muestra el conjunto de variables resultante, además del número de elementos, que no se muestra en la tabla. Así, el número de predictores seleccionados es 38.

Tabla 4.8: Variables seleccionadas (con "X") en base al algoritmo genético con 52 variables de inicio.

	mean	*wtd_mean*	*gmean*	*wtd_gmean*	*entropy*	*wtd_entropy*	*range*	*wtd_range*	*std*	*wtd_std*
atomic_mass	X				X	X				X
fie	X	X			X	X		X		X
atomic_radius	X	X			X	X		X		X
density	X				X	X				
electron_affinity	X			X	X	X		X	X	X
fusion_heat					X	X				
thermal_conductivity	X	X			X	X		X		X
valence				X		X				X

4.6. Redes FFN con 38 características seleccionadas

Una vez seleccionadas 38 características con algoritmos genéticos, se entrenan 20 redes FFN con dichos predictores y con los parámetros de la tabla 4.3. En la tabla 4.9 se muestran los valores medios de algunos indicadores de su eficiencia para los datos de entrenamiento, validación y *test*: MSE_{norm} (*targets* normalizados), RMSE en Kelvin, R y R^2. Prácticamente todos los indicadores son mejores que en el caso de las redes FFN con 52 predictores (tabla 4.4). Sin ser la mejora en eficiencia muy sustancial, se reduce la

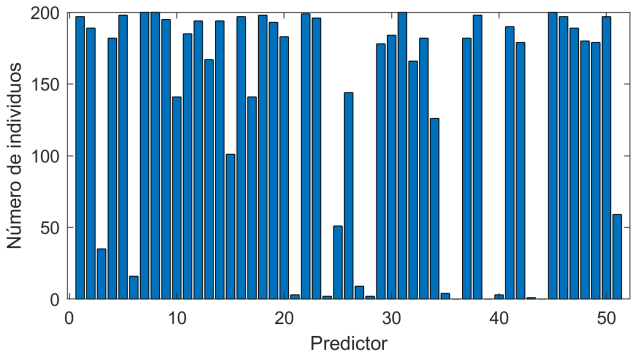


Figura 4.6: Número de individuos que incluyen cada uno de los 51 predictores en la última generación.

complejidad del modelo de regresión debido a la disminución del número de predictores y se gana intuición física sobre qué variables son importantes para la predicción de la temperatura crítica. Conviene destacar los valores medios para los datos de *test*, RMSE = 12.36 K y $R^2 = 0.8699$ (frente a RMSE = 12.56 K y $R^2 = 0.8651$ con 52 predictores).

Tabla 4.9: Valores medios de algunos indicadores de eficiencia de entrenamiento, validación y *test* para 20 redes FFN con 38 predictores y los parámetros de la tabla 4.3.

	Entrenamiento			**Validación**			**Test**		
	Mejor	Peor	Medio	Mejor	Peor	Medio	Mejor	Peor	Medio
$\mathbf{MSE_{norm}}$	0.0877	0.1193	0.0999	0.1070	0.1495	0.1254	0.1180	0.1531	0.1305
RMSE (K)	10.13	11.82	10.82	11.19	13.23	12.12	11.75	13.39	12.36
$\boldsymbol{R}$	0.9555	0.9383	0.9487	0.9445	0.9215	0.9352	0.9387	0.9212	0.9327
$\boldsymbol{R^2}$	0.9130	0.8804	0.9000	0.8921	0.8492	0.8746	0.8812	0.8486	0.8699

4.7. Selección de características con algoritmos genéticos (II)

Tras haber reducido el número de predictores utilizados por la red FFN de 52 a 38, se ejecuta otra vez un algoritmo genético para tratar de reducir su número aún más. El algoritmo utilizado es idéntico al de la sección 4.5, exceptuando que ahora $nvars = 38$ y que cada individuo tiene como valor de la función objetivo el promedio del $\mathrm{MSE_{test}}$ de 3 individuos con el mismo subconjunto de predictores. Aunque esta decisión ralentiza el tiempo de ejecución del algoritmo, reduce el sesgo en el entrenamiento de los modelos ML.

En la parte izquierda de la figura 4.7 se muestra el número de predictores medio y el mejor número de predictores de los individuos para cada generación del algoritmo. En la parte derecha se puede ver el valor medio y el menor valor de $\mathrm{MSE_{test}}$ (el promedio de las tres redes entrenadas por cada individuo) de cada generación. Tras 153 generaciones, se detiene la ejecución del algoritmo, al observarse convergencia. El valor medio de $\mathrm{MSE_{test}}$ tras la convergencia está en torno a 0.131, mientras que el número de características medio es aproximadamente 33 (frente a 0.138 y 35 en el algoritmo de la sección 4.5).

En la figura 4.8 se muestra el histograma del número de individuos que incluyen cada uno de los 38 predictores en la última generación (153) del algoritmo genético. Utilizando el mismo criterio de selección de características que en la sección 4.5, se obtiene el conjunto de predictores de la tabla 4.10, además del número de elementos. Así, se han seleccionado en total 35 predictores. Teniendo en cuenta que el trabajo comenzó con 81 predictores, la reducción en su número ha sido del 56.79 %. En esta ejecución del algoritmo genético sólo se han eliminado tres predictores, por lo que se decide dar por terminada la selección.

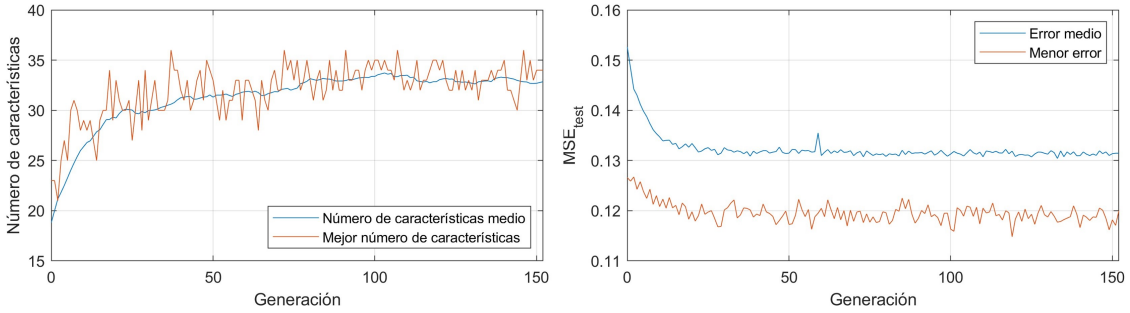


Figura 4.7: *Izquierda*: número de características medio y mejor número de características para cada generación del algoritmo genético. *Derecha*: valor medio y menor valor de MSE_{test} para cada generación.

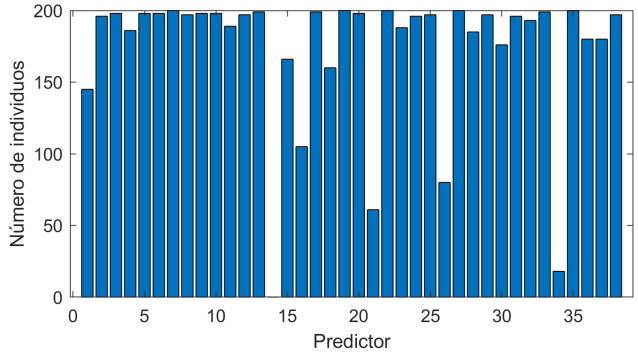


Figura 4.8: Número de individuos que incluyen cada uno de los 38 predictores en la última generación.

Tabla 4.10: Variables seleccionadas (con "X") en base al algoritmo genético con 38 variables de inicio.

	mean	*wtd_mean*	*gmean*	*wtd_gmean*	*entropy*	*wtd_entropy*	*range*	*wtd_range*	*std*	*wtd_std*
atomic_mass	X				X	X				X
fie	X	X			X	X		X		X
atomic_radius	X	X				X		X		X
density	X				X	X				
electron_affinity				X	X	X		X	X	X
fusion_heat					X	X				
thermal_conductivity	X	X			X	X				X
valence				X		X				X

4.8. Redes FFN con 35 características seleccionadas

Tras seleccionar 35 características con algoritmos genéticos, se han entrenado 20 redes FFN con los predictores en cuestión y con los parámetros de la tabla 4.3. En la tabla 4.11 se muestran los promedios de algunos indicadores de eficiencia para los subconjuntos de entrenamiento, validación y *test*: MSE_{norm} (*targets* normalizados), RMSE en Kelvin, R y R^2. Prácticamente todos los indicadores son mejores que en el caso de las redes FFN con 52 predictores (tabla 4.4). La eficiencia es bastante similar a la obtenida con 38 predictores (tabla 4.9), pero se ha reducido la complejidad del modelo al quitar 3 entradas al mismo, además de ganar intuición física. Los valores medios para los datos de *test* son RMSE = 12.21 K y $R^2 = 0.8733$, frente a RMSE = 12.36 K y $R^2 = 0.8699$ con 38 predictores y RMSE = 12.56 K y $R^2 = 0.8651$ con 52 predictores.

4.9. Optimización de hiperparámetros con algoritmos genéticos

Tras reducir a 35 el número de predictores del modelo FFN, se aborda la optimización de los hiperparámetros de dicho modelo. Para ello, se utiliza un algoritmo genético en el que las variables a optimizar toman valores enteros, como se explica en la sección 3.4.3.

Los seis hiperparámetros que se van a optimizar se muestran en la tabla 4.12. Aunque

Tabla 4.11: Valores medios de algunos indicadores de eficiencia de entrenamiento, validación y *test* para 20 redes FFN con 35 predictores y los parámetros de la tabla 4.3.

	Entrenamiento			**Validación**			**Test**		
	Mejor	Peor	Medio	Mejor	Peor	Medio	Mejor	Peor	Medio
$\mathbf{MSE_{norm}}$	0.0839	0.1162	0.0966	0.1143	0.1375	0.1267	0.1107	0.1431	0.1273
RMSE (K)	9.91	11.66	10.64	11.57	12.69	12.18	11.39	12.94	12.21
$\boldsymbol{R}$	0.9571	0.9396	0.9504	0.9423	0.9282	0.9347	0.9436	0.9257	0.9345
$\boldsymbol{R^2}$	0.9160	0.8828	0.9033	0.8879	0.8616	0.8737	0.8904	0.8569	0.8733

los cuatro últimos no toman valores enteros, se escoge un número discreto de posibles valores con el afán de reducir la duración del algoritmo, asignando un entero a cada posible valor y tratando el correspondiente gen como entero. Los parámetros de la red FFN que no se consideran como variables a optimizar toman los siguientes valores: *trainFcn* = *'trainscg'*, *performFcn* = *'mse'*, *trainParam.max_fail* = 150, *trainParam.min_grad* = 1e-10, *trainParam.epochs* = 2000. El porcentaje de *test data* y *validation data* se fijan a partes iguales una vez fijado el porcentaje de *training data* del individuo en cuestión.

Tabla 4.12: Hiperparámetros de la red FFN con 35 características a optimizar.

Hiperparámetro a optimizar	**Posibles valores**
Neuronas 1ª capa oculta	[10, 500]
Neuronas 2ª capa oculta	[3, 300]
Función de activación 1ª capa oculta	'tansig', 'logsig', 'poslin'
Función de activación 2ª capa oculta	'tansig', 'logsig', 'poslin'
learning rate	0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1
Porcentaje de *training data*	40, 45, 50, 55, 60, 65, 70

En el algoritmo genético, cada individuo está constituido por cuatro redes FFN con los valores de los hiperparámetros a optimizar codificados en su genotipo. El valor de la función objetivo es el promedio de $\mathrm{MSE_{test}}$ para las cuatro redes FFN mencionadas.

En la tabla 4.13 se muestran los valores de los parámetros con los que se llama a la función *ga*. Los valores escogidos para las opciones de *ga*, creadas con *optimoptions*, toman los mismos valores que en la tabla 4.7, salvo los mostrados en la tabla 4.14.

Tabla 4.13: Parámetros de *ga* utilizados para el algoritmo genético de optimización de hiperparámetros.

Parámetro	**Valor**	**Descripción**
fun	*@miffn*	Función a optimizar. Entrena cuatro FFNs con el genotipo del individuo (hiperparámetros a optimizar) y devuelve el promedio de sus $\mathrm{MSE_{test}}$.
nvars	6	Número de variables de la función objetivo.
options	-	Opciones del algoritmo genético, creadas con la función *optimoptions* y recogidas en la tabla 4.14.
intcon	[1,2,3,4,5,6]	Las variables en esta lista toman valores enteros.

La gráfica de la figura 4.9 muestra el valor medio y el menor valor de $\mathrm{MSE_{test}}$ para las sucesivas generaciones del algoritmo genético. La ejecución se detiene en la generación 34. Puede verse que, tras la convergencia, el error promedio (para T_c normalizadas) está en torno a 0.12 - 0.13, mientras que el menor error está en torno a 0.11.

En la figura 4.10 se muestran seis diagramas de sectores para los seis genes (hiperparámetros) de los individuos de la última generación (34). Mientras que algunos, como

Tabla 4.14: Opciones utilizadas para el algoritmo genético de optimización de hiperparámetros.

Opción	Valor	Descripción
'PopulationType'	Por defecto	Codificación del genotipo de los individuos. Por defecto, los genes son reales salvo aquellos que se especifiquen en *intcon* como enteros.
'PopulationSize'	Por defecto	$\text{mín}\,(\text{máx}\,(10 \cdot nvars, 40), 100) = 60$

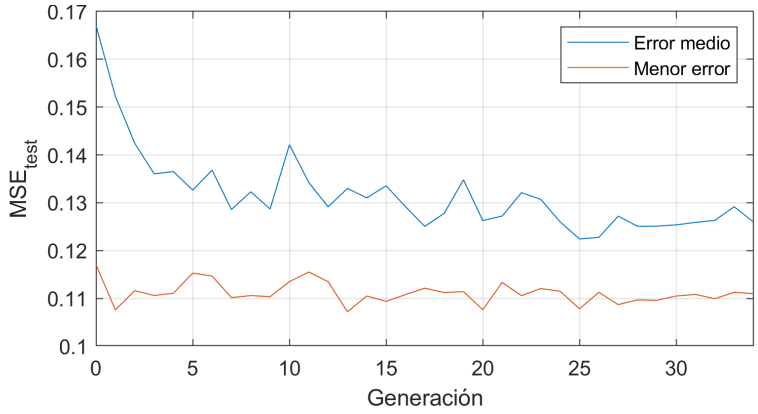


Figura 4.9: Valor medio y menor valor de MSE_{test} para cada generación del algoritmo genético.

la función de activación de la primera capa oculta o el porcentaje de *training data*, están prácticamente homogeneizados para toda la población, hay otros con mayor variabilidad.

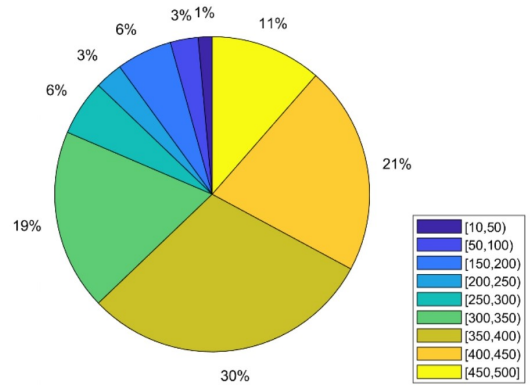
En la tabla 4.15 se muestran los valores (o rangos de valores) de los hiperparámetros para el mayor sector de los gráficos y para el mejor individuo (menor MSE_{test}) de la generación 34. Con respecto a las redes FFN de secciones anteriores, las redes optimizadas:

Tabla 4.15: Valores de los hiperparámetros para el mayor sector del gráfico de sectores y para el mejor individuo (menor MSE_{test}) de la generación 34.

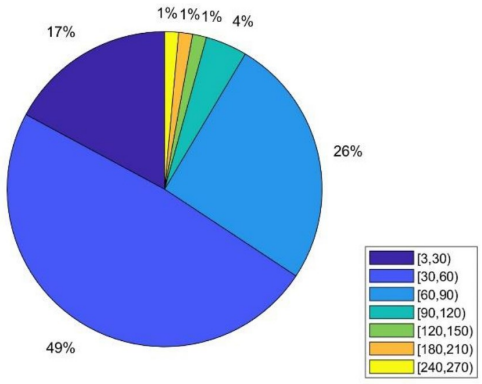
	Hiperparámetro	Mayor Sector	Mejor Individuo
a)	Número de neuronas 1ª capa oculta	[350,400)	422
b)	Número de neuronas 2ª capa oculta	[30,60)	36
c)	Función de activación 1ª capa oculta	'poslin'	'poslin'
d)	Función de activación 2ª capa oculta	'logsig'	'logsig'
e)	*learning rate*	0.1	0.01
f)	Porcentaje de *training data*	70 %	70 %

- Tienen un número de neuronas mayor que las anteriores ($hidden = [18, 10]$).
- Utilizan una primera capa oculta con función de activación 'poslin' (ReLU) en lugar de 'logsig' (sigmoide).
- El *learning rate* utilizado es el mismo que en redes anteriores, 0.1, en el caso del mayor sector, y menor, 0.01, en el caso del mejor individuo.
- Respecto al porcentaje de *training data*, cuantos más datos se utilicen para entrenar el modelo mejor será éste, luego no sorpende que un 97 % de los individuos utilicen el mayor valor considerado, 70 %. Sin embargo, se seguirá utilizando un valor de 60 % por consistencia en la comparación con otros modelos desarrollados en el trabajo.
- El tiempo de entrenamiento de las redes optimizadas es mayor (unos 10 min frente a 3 min en las no optimizadas), por su mayor número de neuronas.

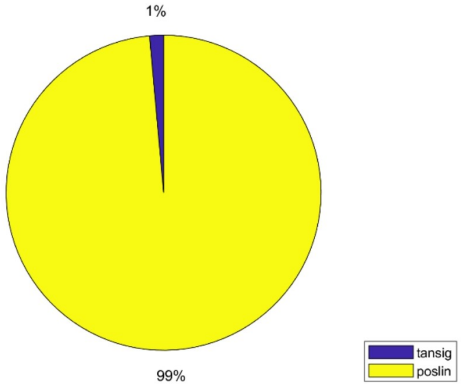
a) Neuronas 1ª capa oculta



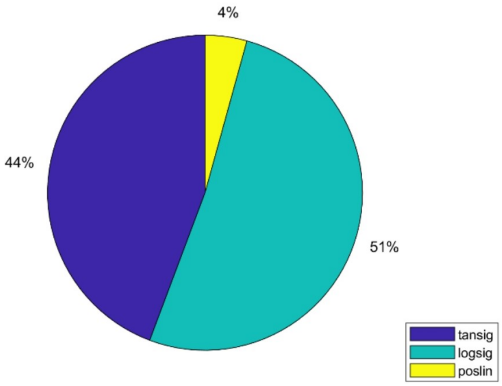
b) Neuronas 2ª capa oculta



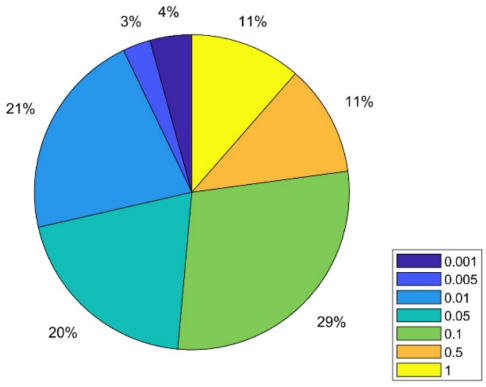
c) Función de activación 1ª capa oculta



d) Función de activación 2ª capa oculta



e) *Learning-rate*



f) Porcentaje de *training data*

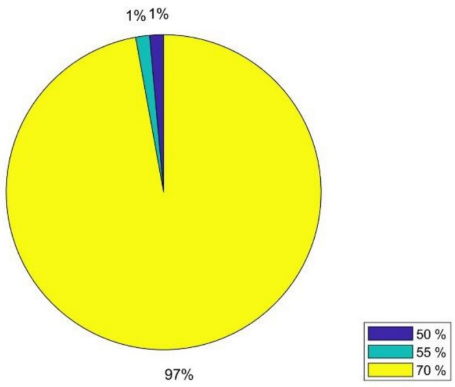


Figura 4.10: Diagramas de sectores de los genotipos de los 70 individuos de la generación 34.

4.10. Redes FFN optimizadas con 35 características

Una vez realizada la optimización de hiperparámetros, se entrenan redes FFN considerando los parámetros optimizados de la tabla 4.15. Para cada conjunto de parámetros, se entrenan 20 redes y se toma el promedio de sus $\mathrm{MSE}_{\mathrm{test}}$ como indicador de su eficiencia. En ambos casos, se utiliza un 60 % de *training data*, por los motivos expuestos en la sección 4.9. Se fija, también, *trainFcn* = *'trainscg'*, *performFcn* = *'mse'*, *trainParam.max_fail* =

300, *trainParam.min_grad* = 1e-10, *trainParam.epochs* = 10000.

En la tabla 4.16 se muestran los valores de los hiperparámetros optimizados para la mejor red obtenida, y en la tabla 4.17 sus indicadores de eficiencia de entrenamiento, validación y *test*. Todos ellos mejoran con respecto a antes de la optimización de hiperparámetros. Es conveniente destacar algunos de estos indicadores:

Tabla 4.16: Valores de los hiperparámetros optimizados para la mejor red obtenida.

Número de neuronas 1ª capa oculta	400
Número de neuronas 2ª capa oculta	36
Función de activación 1ª capa oculta	'poslin'
Función de activación 2ª capa oculta	'logsig'
learning rate	0.01
Porcentaje de *training data*	60 %

Tabla 4.17: Valores medios de algunos indicadores de eficiencia de entrenamiento, validación y *test* para 20 redes FFN con 35 predictores y los parámetros de la tabla 4.16.

	Entrenamiento			**Validación**			**Test**		
	Mejor	Peor	Medio	Mejor	Peor	Medio	Mejor	Peor	Medio
$\mathbf{MSE_{norm}}$	0.0628	0.0817	0.0714	0.1075	0.1338	0.1167	0.1042	0.1291	0.1161
RMSE (K)	8.58	9.78	9.14	11.22	12.52	11.69	11.05	12.29	11.66
$\boldsymbol{R}$	0.9683	0.9581	0.9637	0.9464	0.9298	0.9399	0.9464	0.9338	0.9407
$\boldsymbol{R^2}$	0.9376	0.9180	0.9287	0.8957	0.8645	0.8834	0.8957	0.8720	0.8849

- Para la red con 35 predictores sin optimizar, los promedios para los datos de entrenamiento son RMSE = 10.64 K y $R^2 = 0.9033$. Tras la optimización, se obtiene RMSE = 9.14 K y $R^2 = 0.9287$. Esta mejoría indica que el aumento de la complejidad del modelo (mayor número de neuronas) mejora la eficiencia de entrenamiento. Es decir, que el modelo inicial es demasiado simple para el problema.
- La red con 35 predictores sin optimizar tiene promedios de *test* de RMSE = 12.21 K y $R^2 = 0.8733$. Tras la optimización, se obtiene RMSE = 11.66 K y $R^2 = 0.8849$. El aumento en la complejidad del modelo realizado no disminuye su capacidad de generalización, sino que la aumenta.
- El uso de funciones ReLU en la primera capa oculta mejora la eficiencia.

En la figura 4.11 se muestran las gráficas de regresión (salida predicha frente a valor real) con T_c normalizada para una de las redes optimizadas. En comparación con las obtenidas en el primer modelo FFN (figura 4.3), hay muchos menos puntos alejados de la bisectriz del primer cuadrante, luego la mejoría conseguida ha sido significativa.

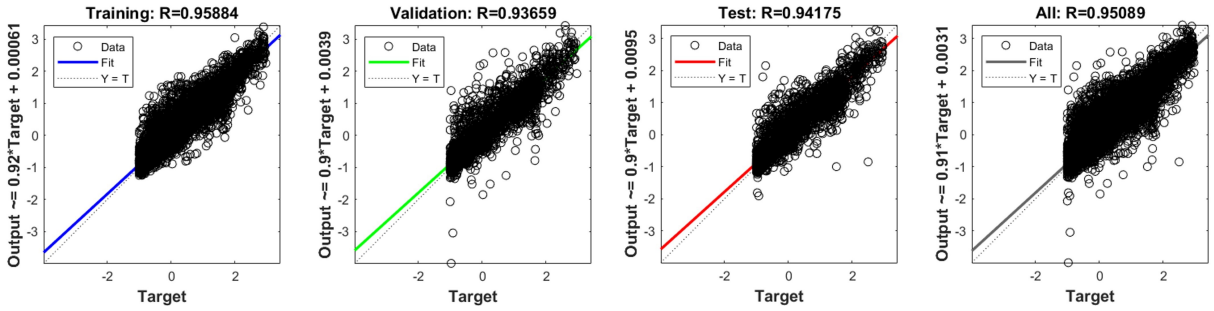


Figura 4.11: Gráficas de regresión con los datos normalizados para una de las redes optimizadas.

4.11. Comparativa y discusión

En la tabla 4.18 se muestra una comparativa de la eficiencia (promedio de 20 redes) de las distintas redes entrenadas en este trabajo, indicando la sección de la memoria correspondiente a dicha red. Además, se indican como filas enteras los algoritmos de selección de características y optimización de hiperparámetros ejecutados, de modo que la tabla sigue un orden cronológico de arriba hacia abajo.

Tabla 4.18: Comparativa del promedio (20 redes) del RMSE y de R^2 de *test* para las distintas redes.

	Sección	$\mathbf{RMSE_{test}}$ **(K)**	$\boldsymbol{R^2}_{\mathbf{test}}$
Selección de características en base al coeficiente de correlación (sección 4.2)			
FFN con 52 características	4.3	12.56	0.8651
ELM con 52 características	4.4	14.35	0.8245
Selección de características con algoritmos genéticos (I) (sección 4.5)			
FFN con 38 características	4.6	12.36	0.8699
Selección de características con algoritmos genéticos (II) (sección 4.7)			
FFN con 35 características	4.8	12.21	0.8733
Optimización de hiperparámetros con algoritmos genéticos (sección 4.9)			
FFN optimizada con 35 características	4.10	11.66	0.8849

Las primeras redes entrenadas, tanto FFN como ELM, tienen 52 predictores, seleccionados en base al coeficiente de correlación. Se observa que ELM arroja un RMSE casi 2 K mayor que FFN y un R^2 0.04 puntos menor.

La primera ejecución del algoritmo genético de selección con redes FFN reduce los predictores a 38. También mejoran los parámetros de eficiencia de *test*: RMSE disminuye en 0.2 K, y R^2 aumenta unos 0.004 puntos. La segunda ejecución del algoritmo reduce los predictores a 35, además de disminuir RMSE en 0.15 K y aumentar R^2 en 0.0034 puntos.

Finalmente, la optimización de hiperparámetros reduce $\mathrm{RMSE_{test}}$ otros 0.55 K y aumenta R^2_{test} otros 0.0116 puntos. Se observa, así, que el mayor aumento de la eficiencia se da con la optimización de hiperparámetros. Aún así, no se llega al umbral $R^2 \geq 0.9$.

Con respecto a la selección de características, finalmente se han seleccionado 35 predictores, los 34 de la tabla 4.10 junto al número de elementos distintos en el superconductor. Los predictores seleccionados proporcionan intuición física sobre qué variables afectan a T_c y sugieren que aquellas propiedades atómicas para las que se incorporan como predictores:

- La media aritmética o la media aritmética pesada (masa atómica, primera energía de ionización, radio atómico, densidad, conductividad térmica) influyen en T_c mediante una relación lineal.
- La media geométrica pesada (afinidad electrónica y valencia) entran en la fórmula de T_c como una raíz n-ésima.
- La entropía o la entropía ponderada (todas) aparecen en la expresión de T_c como sumas de logaritmos.
- El rango ponderado (primera energía de ionización, radio atómico, afinidad electrónica) entran en la función T_c como restas ponderadas de los valores extremos.
- La desviación típica o la desviación típica ponderada (masa atómica, primera energía de ionización, radio atómico, afinidad electrónica, conductividad térmica y valencia) entran en la fórmula de T_c como raíces cuadradas de sumas de términos cuadráticos.

En base a esto, se confirma que la hipotética función matemática para T_c es bastante compleja. Es por ello que los modelos ML resultan de utilidad en contextos como éste, con una solución analítica compleja.

5. Conclusiones y líneas futuras

En este trabajo se ha desarrollado un modelo ML para la predicción de la temperatura crítica de materiales superconductores partiendo de su fórmula empírica. Se han utilizado dos tipos de modelos ML de regresión, las redes FFN y ELM. Estas últimas se han descartado debido a su baja eficiencia. Además de los modelos de regresión, se ha utilizado otro conjunto de técnicas ML, los algoritmos genéticos, para reducir el número de predictores en el modelo de regresión y optimizar sus hiperparámetros.

Para desarrollar el modelo de regresión, se ha utilizado un *dataset* realizado por Kam Hamidieh [3]. Dicho *dataset* contiene 21263 muestras (de las que se han eliminado 4 *outliers*), cada una con 81 predictores extraídos a partir de la fórmula empírica de los superconductores y las temperaturas críticas correspondientes, obtenidas de la base de datos Supercon. Antes de comenzar a entrenar los modelos ML, se ha reducido el número de predictores a 52 eliminando aquellos con coeficiente de correlación $R > 0.9$ con respecto a algún otro predictor.

Tras entrenar varias redes FFN con 52 predictores, se ha reducido el número de predictores a 35 por medio del uso de algoritmos genéticos para selección de características. Como resultado de dicha selección de características se ha obtenido no sólo un modelo más simple sino también una mayor eficiencia, pues dicho modelo con menor número de predictores tiene mayor capacidad de generalización que el modelo inicial con 52 predictores. Además, la selección de características ha servido también para ganar intuición física acerca de qué variables de las consideradas como predictores entrarían y cómo en una hipotética fórmula analítica para la temperatura crítica. Se ha confirmado, de esta forma, la esperable complejidad de esta fórmula que sigue sin encontrarse a pesar de los esfuerzos realizados por la comunidad científica.

Finalmente, se han utilizado algoritmos genéticos similares a los utilizados para la selección de características para optimizar los hiperparámetros del modelo de regresión FFN con 35 predictores. El algoritmo de optimización ha puesto de manifiesto que el número de neuronas necesario es mayor que el considerado inicialmente y que resulta beneficioso para este problema utilizar una función de activación ReLU, en lugar de la sigmoide, en la primera capa oculta. Este modelo optimizado con 35 predictores tiene unos indicadores de eficiencia promedio para los datos de *test* de RMSE = 11.66 K y $R^2 = 0.8849$. Estos indicadores no llegan a superar a los obtenidos con los modelos basados en XGBoost, *random forest* o *bagged trees* mencionados en la introducción, aunque hay que tener en cuenta que el modelo FFN es computacionalmente más sencillo que éstos.

A pesar de no haber superado el umbral $R^2 > 0.9$ y de haber otros trabajos que obtienen mejores indicadores de eficiencia, el modelo desarrollado resulta de interés porque ha conseguido reducir el número de predictores en un 56.79 %, proporcionando la intuición física ya mencionada. Además, se ha ilustrado cómo los algoritmos genéticos para selección de características y optimización de hiperparámetros pueden resultar útiles como modelos auxiliares a la hora de desarrollar un modelo de regresión, con resultados satisfactorios. Gracias a estos algoritmos, es posible explorar un espacio de posibilidades mucho mayor que el que sería viable explorar de manera manual o iterativa (probando una por una todas las posibilidades). Se ha comprobado, por otra parte, que las redes neuronales son también una potente herramienta para tratar de aproximar funciones desconocidas (por su complejidad u otros motivos), como la función para la temperatura crítica. Este trabajo es un ejemplo de cómo las redes neuronales y los algoritmos genéticos, entre otras técnicas ML, pueden resultar de utilidad como herramienta adicional a las herramientas analíticas

y numéricas en la resolución de problemas físicos complejos.

A nivel académico personal, este trabajo me ha servido para sumergirme en el mundo del ML y de la IA, que se está desarrollando a una velocidad cada vez más vertiginosa. Con los modelos desarrollados he aprendido cómo se pueden utilizar las redes neuronales (y otros modelos de regresión) junto con los algoritmos genéticos para resolver problemas de física, y que no sólo se trata de una mera caja negra que predice un resultado, sino que también se puede ganar intuición que pueda servir de ayuda en el desarrollo de una teoría analítica. Además, este trabajo me ha supuesto una oportunidad para aprender sobre el fenómeno de la superconductividad, de gran interés en campos como el almacenamiento y transporte de la energía o la creación de campos magnéticos fuertes que tan en auge están actualmente.

Para dar fin a esta sección, se proponen una serie de líneas de trabajo futuras que podrían ser interesantes para continuar con el estudio realizado en este TFG:

- Desarrollar un modelo ML para predecir la temperatura crítica de superconductores que incluya como predictores parámetros relacionados con la estructura cristalina (que, según las reglas de Matthias, en la sección 2.2, influye en la superconductividad).
- Desarrollar un modelo de clasificación que prediga si un material es superconductor o no (el modelo desarrollado predice T_c considerando que la fórmula empírica proporcionada corresponde a un superconductor).
- Utilizar técnicas de Programación Genética para tratar de encontrar una función analítica para la temperatura crítica en función de los predictores utilizados en el modelo de regresión. Estas técnicas van construyendo funciones en base a unas variables de entrada y un conjunto de operadores, y dichas funciones van evolucionando según un algoritmo genético hacia una que minimice el error.
- Desarrollar un modelo que sepa extraer las características a partir de la fórmula empírica y los datos de los elementos atómicos, para no perder información al realizar la extracción de manera manual. El principal esfuerzo estaría en encontrar una representación de los datos que permitiera, además de la fórmula empírica, proporcionar al modelo la multitud de parámetros atómicos necesarios para la extracción de características.
- Búsqueda de modelos basados en lógica difusa (en términos de conjuntos borrosos, definidos por reglas imprecisas) para ganar intuición sobre qué características influyen más sobre la temperatura crítica.
- Considerando que a día de hoy los únicos superconductores de alta temperatura que se han encontrado requieren también de condiciones de alta presión, desarrollar modelos que también recojan la influencia de la presión.

Está claro que el reto de la superconductividad a temperatura ambiente tiene todavía mucho camino que recorrer. Para terminar de aclarar qué variables influyen sobre la temperatura crítica y sobre si un material es superconductor o no, es necesario un equipo multidisciplinar de expertos en física de materiales y expertos en aplicación de técnicas numéricas y técnicas ML a problemas de física.

Glosario

backpropagation Algoritmo para propagar el error de una red multicapa desde la salida hacia las capas internas.
bandgap Intervalo prohibido de energías comprendido entre dos niveles o bandas permitidos.
batch learning Actualización de los pesos considerando el error total para todos los datos de entrenamiento.
BCS Teoría de la superconductividad de Bardeen-Cooper-Schrieffer.
bias Término constante que se suma a una combinación lineal.

classification Clasificación, referido a un algoritmo.
cluster Cada uno de los grupos de un algoritmo de *clustering*.
clustering Agrupación, referido a un algoritmo.
CPU *Central Processing Unit*, Unidad de Procesamiento Central.

dataset Base de datos.

ELM *Extreme Learning Machine*, es un tipo de NN de una capa oculta.
epoch Conjunto de tantas actualizaciones sucesivas de pesos como datos de entrenamiento, una para cada dato.

feature selection Selección de características.
features Características o variables de una entrada de una base de datos.
FFN *Feed-Forward Network*, red neuronal en la que la información fluye solamente en la dirección de entrada a salida.
forward-propagation Propagación de un dato desde la entrada de la red hacia la salida..

generalization Generalización, capacidad de un modelo de lograr buenos resultados con datos no vistos durante el entrenamiento.
genetic algorithm Algoritmo genético.
GPU *Graphics Processing Unit*, Unidad de Procesamiento de Gráficos.

hidden Oculta, hace referencia a una etapa interna de un modelo ML.
hyperparameters Hiperparámetros de un modelo ML.

IA Inteligencia Artificial.

layer Capa de una red neuronal.
learning rate Tasa de aprendizaje de un modelo ML.

ML Machine Learning, se traduce como Aprendizaje Automático.
MSE *Mean Squared Error*, Error Cuadrático Medio.
multilayer De varias capas, referido a una red neuronal.

NN *Neural Network*, Red Neuronal.

on-line learning Actualización de los pesos considerando cada dato de entrenamiento individualmente.
outlier Dato de un conjunto estadístico muy alejado del resto.
overfitting Cuando un modelo se ajusta tanto a los datos de entrenamiento que no generaliza a datos nuevos.

regression Regresión, referido a un algoritmo.
ReLU *Rectified Linear Unit*, un tipo de función de activación.
RMSE *Root Mean Squared Error*, Raíz del Error Cuadrático Medio.

supervised Supervisado, referido a un algoritmo.
SVD *Singular Value Descomposition*, Descomposición en Valores Singulares.

target Objetivo, valor deseado de la salida en un algoritmo ML.
task Tarea o problema que resuelve un algoritmo ML.
test data Subdivisión de los datos para test o prueba.
train Entrenar un modelo ML.
training data Subdivisión de los datos para entrenamiento.

underfitting Cuando un modelo es tan sencillo que no se ajusta adecuadamente a los datos de entrenamiento.
unsupervised No supervisado, referido a un algoritmo.

validation data Subdivisión de los datos para validación.

zscore Técnica de normalización basada en restar la media y dividir entre la desviación típica.

Referencias

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. Cambridge (EE. UU.): MIT Press, 2016.

[2] C. Kittel, *Introduction to Solid State Physics*, 8th ed. New York: John Wiley and Sons, 2005.

[3] K. Hamidieh, "A data-driven statistical model for predicting the critical temperature of a superconductor," *Computational Materials Science*, no. 154, 2018.

[4] K. Matsumoto and T. Horide, "An acceleration search method of higher Tc superconductors by a machine learning algorithm," *Applied Physics Express*, vol. 12, no. 7, p. 073003, jun 2019. [Online]. Available: https://dx.doi.org/10.7567/1882-0786/ab2922

[5] B. Roter and S. V. Dordevic, "Predicting new superconductors and their critical temperatures using machine learning," *Physica C: Superconductivity and its Applications*, vol. 575, p. 1353689, 2020, iD: 271531. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0921453420301374

[6] "BCS theory of superconductivity lecture notes for engineering physics," 2016. [Online]. Available: https://semesters.in/bardeen-cooper-schriffer-bcs-theory-qualitative-notes-for-engineering-physics-btech/

[7] A. A. Shah and T. Bhatnagar, "Superconductors - "resistance is futile"," in *5th IEEE Integrated STEM Conference*, 2015.

[8] N. Ghazikhanian, "The Matthias Rules," Tech. Rep., December 2017.

[9] L. Marvin, *Neural Networks with MATLAB*, 2016.

[10] C. C. Aggarwall, *Neural networks and deep learning : a textbook*. Cham, Switzerland: Springer, 2018.

[11] A. Zell, *Simulation Neuronaler Netze*, 1st ed. Addison-Wesley, 1994.

[12] C. M. Bishop, *Neural Networks for Pattern Recognition*. New York: Oxford University Press, 1995.

[13] L. L. C. Kasun, H. Zhou, G.-B. Huang, and C. M. Vong, "Extreme learning machines," *IEEE Intelligent Systems*, pp. 31–34, November/December 2013.

[14] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*, 2nd ed. Berlin: Springer, 2015.

[15] "Reference page for feedforwardnet," Último acceso: 17/03/2023. [Online]. Available: https://es.mathworks.com/help/deeplearning/ref/feedforwardnet.html?lang=en

[16] "Reference page for ga," Último acceso: 10/04/2023. [Online]. Available: https://es.mathworks.com/help/gads/ga.html#d124e50955

[17] "How can I save every generation's best position using ga?" Último acceso: 10/04/2023. [Online]. Available: https://es.mathworks.com/matlabcentral/answers/123459-how-can-i-save-every-generation-s-best-position-using-ga

[18] K. F. Riley, M. P. Hobson, and S. J. Bence, *Mathematical Methods for Physics and Engineering*, 3rd ed. Cambridge University Press, 2006.

A. Algunos conceptos de estadística

A continuación se explican algunos conceptos básicos de estadística, con el fin de llegar a definir el coeficiente de correlación de Pearson. Este indicador se utiliza tanto para selección de características (sección 3.4.2) como para evaluación de los modelos de regresión (sección 3.1.3). A lo largo de la explicación, se consideran dos variables a y b (o características) con valores $\{a_i\}_{i=1}^N$ y $\{b_i\}_{i=1}^N$, donde N es el número de datos en el *dataset* o en alguna de sus subdivisiones, dependiendo del contexto.

A.1. Varianza y desviación estándar

La varianza s^2 y la desviación estándar $\sigma = s = \sqrt{s^2}$ son medidas de la dispersión de los N valores $\{a_i\}$ de una variable a del *dataset*. La desviación típica es la raíz de la varianza, y esta última se define como [18]:

$$s^2 = \frac{1}{N}\sum_{i=1}^{N}(a_i - \overline{a})^2 = \overline{a^2} - \overline{a}^2 \tag{A.1}$$

donde $\overline{a}$ es la media de los valores en la muestra y $\overline{a^2}$ la media de los cuadrados de los valores.

A.2. Covarianza y correlación

La covarianza y correlación son dos indicadores que caracterizan la relación entre dos variables a y b del *dataset*. La covarianza se define como [18]:

$$\sigma_{ab} = \frac{1}{N}\sum_{i=1}^{N}(a_i - \overline{a})\left(b_i - \overline{b}\right) = \overline{ab} - \overline{a}\overline{b} \tag{A.2}$$

donde la barra sobre las variables significa valor medio. Una propiedad de la covarianza es que dos variables independientes en sentido estadístico tienen covarianza nula. El recíproco, sin embargo, no es cierto.

Resulta más útil en el contexto de este trabajo el coeficiente de correlación de Pearson, definido de la siguiente manera [18]:

$$R_{ab} = \frac{\sigma_{ab}}{\sigma_a \sigma_b} \tag{A.3}$$

donde σ_{ab} es la covarianza entre las variables a y b, σ_a la desviación estándar de a y σ_b la desviación estándar de b. Se puede demostrar que R_{ab} siempre está comprendido entre -1 y 1 [18]. Por lo tanto, la correlación no es más que un escalado de la covarianza. El coeficiente R_{ab} tiene las siguientes propiedades [18]:

- Dos variables totalmente independientes tienen $R_{ab} = 0$.
- Dos variables relacionadas linealmente entre sí tienen $R_{ab} = 1$ si la pendiente de la relación es positiva y $R_{ab} = -1$ si la pendiente es negativa.

Estas dos propiedades justifican el uso de R como medida de la eficiencia (sección 3.1.3) de un modelo de regresión: cuanto más próximo a la unidad sea el valor de $R_{\hat{y}y}$, mayor será la eficiencia del modelo. Se suele utilizar también el cuadrado de este coeficiente, R^2, denominado coeficiente de determinación. Para que un modelo de regresión se considere bueno, se utiliza el criterio $R^2 > 0.90$, que es equivalente a $R > 0.95$.

Otra aplicación de la correlación es la reducción de predictores de un *dataset* mediante la eliminación de predictores redundantes (aquellos que están correlacionados con otros predictores).

B. Desarrollo matemático del algoritmo *backpropagation* y el descenso del gradiente

En este apéndice se desarrollan las matemáticas del entrenamiento de redes FFN. Se comienza con la deducción de las expresiones para calcular las derivadas de la función error con respecto a los pesos y *bias* utilizando el algoritmo *backpropagation*. Finalmente, se explica cómo se pueden utilizar dichas derivadas para actualizar los pesos de la red con el algoritmo de descenso del gradiente.

La función error E puede expresarse como la suma de la contribución de cada entrada de datos al error total, E_i:

$$E = \frac{1}{N}\sum_{i=1}^{N} E_i \tag{B.1}$$

donde N es el número total de entradas del *dataset*. Para simplificar la notación en el desarrollo matemático, se considerará una única entrada de datos $\vec{x}_i$ y su correspondiente error E_i. Así, en primer lugar, se comienza calculando las salidas de todas las neuronas de la red para dicha entrada (*forward-propagation*), utilizando las ecuaciones 3.6.

Ahora, se calcula la derivada del error E_i con respecto a un peso cualquiera de la red $w_{rs}^{(k)}$ (donde el superíndice k se refiere a la k-ésima capa oculta y los subíndices (r, s) a los elementos de la matriz de pesos). Resultará de ayuda para los cálculos siguientes escribir explícitamente las expresiones para la r-ésima neurona de la k-ésima capa oculta.

$$h_r^{(k)} = f\left(a_r^{(k)}\right) \qquad ; \qquad a_r^{(k)} = \sum_s w_{rs}^{(k)} h_s^{(k-1)} \tag{B.2}$$

Utilizando la regla de la cadena [12]:

$$\frac{\partial E_i}{\partial w_{rs}^{(k)}} = \frac{\partial E_i}{\partial a_r^{(k)}} \frac{\partial a_r^{(k)}}{\partial w_{rs}^{(k)}} = \delta_r^{(k)} \frac{\partial a_r^{(k)}}{\partial w_{rs}^{(k)}} \tag{B.3}$$

donde se ha definido el error de la neurona como [12]:

$$\delta_r^{(k)} \equiv \frac{\partial E_i}{\partial a_r^{(k)}} \tag{B.4}$$

Utilizando la ecuación B.2 es inmediato el cálculo de la derivada de la derecha de la ecuación B.3, y queda:

$$\frac{\partial E_i}{\partial w_{rs}^{(k)}} = \delta_r^{(k)} h_s^{(k-1)} \tag{B.5}$$

Sólo falta calcular los errores de las neuronas, $\delta_r^{(k)}$. Para la capa de salida, el resultado es inmediato [12] considerando que $\hat{y} = f_{\text{out}}\left(a^{(\text{out})}\right)$:

$$\delta^{(\text{out})} = \frac{\partial E_i}{\partial a^{(\text{out})}} = \frac{\partial E_i}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a^{(\text{out})}} = f'_{\text{out}}\left(a^{(\text{out})}\right) \frac{\partial E_i}{\partial \hat{y}} \tag{B.6}$$

Para la r-ésima neurona oculta de la capa k, se aplica la regla de la cadena teniendo en cuenta que hay que sumar a todas las neuronas de la siguiente capa (subíndice t de la suma) [12]:

$$\delta_r^{(k)} = \frac{\partial E_i}{\partial a_r^{(k)}} = \sum_t \frac{\partial E_i}{\partial a_t^{(k+1)}} \frac{\partial a_t^{(k+1)}}{\partial a_r^{(k)}} = f_k' \left(a_r^{(k)}\right) \sum_t w_{tr}^{(k+1)} \delta_t^{(k+1)} \tag{B.7}$$

En la última igualdad se ha utilizado la definición de δ (ecuación B.4) y las ecuaciones B.2. Las tres ecuaciones anteriores (B.5, B.6 y B.7) constituyen las ecuaciones *backpropagation*. Así, dicho algoritmo consiste en ir calculando los errores $\delta_r^{(k)}$ de cada neurona "de atrás hacia delante", y multiplicar el error de la neurona por la salida de la neurona en la capa anterior conectada a través del peso $w_{rs}^{(k)}$ en cuestión.

Para calcular la derivada del error total E (para todos los datos de entrenamiento) respecto a un peso basta con hacer la suma de la derivada para cada uno de los datos [12]:

$$\frac{\partial E}{\partial w_{rs}^{(k)}} = \frac{1}{N} \sum_i \frac{\partial E_i}{\partial w_{rs}^{(k)}} = \frac{1}{N} \sum_i \delta_r^{(k)(i)} h_s^{(k-1)(i)} \tag{B.8}$$

donde se ha reintroducido el índice i para hacer referencia a cada entrada de los datos de entrenamiento, y la suma se extiende a todos los datos de entrenamiento.

Una vez estimada la contribución de cada peso al error, se utilizan las derivadas calculadas para actualizar los valores de los pesos. Uno de los algoritmos de minimización más utilizados es el del descenso del gradiente [12]. Se puede pensar que las derivadas calculadas en las ecuaciones B.5 o B.8 constituyen el vector gradiente de la función E_i o E, respectivamente, en el espacio de los pesos $w_{rs}^{(k)}$. Por tanto, dicho vector apunta hacia la dirección y sentido en el que más aumenta el error en un entorno local al valor actual de los pesos. El descenso del gradiente consiste en actualizar el valor de los pesos "dando un paso" en el sentido opuesto al vector gradiente, que es el de mayor disminución local del error. Se introduce un parámetro η denominado *learning rate* o tasa de aprendizaje que representa el tamaño de dicho paso. Existen dos posibles maneras de actualizar los pesos [12]:

- *On-line learning*: actualizar los pesos tras el cálculo de las derivadas para una entrada de los datos de entrenamiento:

$$\Delta w_{rs}^{(k)} = -\eta \delta_r^{(k)(i)} h_s^{(k-1)(i)} \tag{B.9}$$

  Se denomina iteración a la actualización realizada para cada entrada individual de los datos de entrenamiento, y *epoch* al conjunto de actualizaciones asociadas a una exposición completa a todos los datos de entrenamiento.
- *Batch learning*: actualizar los pesos tras sumar las derivadas para cada una de las entradas de los datos de entrenamiento:

$$\Delta w_{rs}^{(k)} = -\eta \sum_i \delta_r^{(k)(i)} h_s^{(k-1)(i)} \tag{B.10}$$

  En este caso, una única actualización de los pesos ya constituirá una *epoch*. El factor $\frac{1}{N}$ que aparece en la ecuación B.8 se ha absorbido en η.

C. Relación entre el error cuadrático medio de una variable normalizada y sin normalizar

En este apéndice se deduce cuál es la relación entre el error cuadrático medio de una variable normalizada con *zscores* y el error cuadrático medio de dicha variable sin normalizar.

Se considera que los *targets* de la variable son $\{y_i\}$, y que los valores predichos por el modelo son $\{\hat{y}_i\}$. Los *targets* normalizados se denotan $\{z_i\}$, y los valores normalizados predichos por el modelo se escriben como $\{\hat{z}_i\}$. Cuando se entrena el modelo, se están utilizando los *targets* normalizados y el MSE_{test} obtenido es para la variable normalizada (se indicará como $\text{MSE}_{\text{test},z}$). Físicamente, interesa conocer el valor del error cuadrático medio de la variable sin normalizar, que se indicará como $\text{MSE}_{\text{test},y}$.

En el desarrollo de la deducción, se utilizarán las relaciones inversas de la normalización con *zscores* (ecuación 3.2).:

$$y_i = z_i\sigma + \mu \qquad ; \qquad \hat{y}_i = \hat{z}_i\sigma + \mu \tag{C.1}$$

Se parte de la expresión del error cuadrático medio (ecuación 3.1) para los datos sin normalizar (N entradas de datos), y se sustituyen las relaciones de normalización inversas anteriores:

$$\text{MSE}_{\text{test},y} = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2 = \frac{1}{N}\sum_{i=1}^{N}(\hat{z}_i\sigma + \mu - z_i\sigma - \mu)^2 = \sigma^2\frac{1}{N}\sum_{i=1}^{N}(\hat{z}_i - z_i)^2 \tag{C.2}$$

El factor que acompaña a σ^2 en la última igualdad es, precisamente, la definición del error cuadrático medio para la variable normalizada, z, luego:

$$\text{MSE}_{\text{test},y} = \sigma^2\text{MSE}_{\text{test},z} \tag{C.3}$$