Universidad de Alcalá
Escuela Politécnica Superior

Grado en Ingeniería en Tecnologías Industriales

Trabajo Fin de Grado

Cálculo del límite elástico del acero en piezas de hormigón armado haciendo uso de técnicas de Machine Learning y Deep Learning

Autor: Adrián García Cerrillo

Tutor: Jorge Pérez Aracil

Cotutor: César Peláez Rodríguez

2023

UNIVERSIDAD DE ALCALÁ

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería en Tecnologías Industriales

Trabajo Fin de Grado

Cálculo del límite elástico del acero en piezas de hormigón armado haciendo uso de técnicas de Machine Learning y Deep Learning

Autor: Adrián García Cerrillo

Tutor: Jorge Pérez Aracil

Cotutor: César Peláez Rodríguez

Tribunal:

Presidente: Hilario Gómez Moreno

Vocal 1º: Roberto López Sastre

Vocal 2º: Jorge Pérez Aracil

Fecha de depósito: 20 de septiembre de 2023

Resumen

Hoy en día debido a los nuevos avances tecnológicos y computacionales se permite almacenar y procesar una gran cantidad de información y gracias a ello es posible utilizar los datos y la tecnología en infinidad de aplicaciones. En este caso se centra en el ámbito de la construcción, en el que el hormigón armado es un material que puede presentar comportamientos no lineales. Al presentar estas no linealidades modelizar una respuesta numérica puede ser una tarea compleja. Para ello se presenta una alternativa utilizando varios algoritmos de predicción basados en *Machine Learning* y *Deep Learning*. Para implementar los algoritmos se ha hecho uso del lenguaje de programación Python. Además, se han realizado dos enfoques, un análisis multisalida, que consiste en predecir todas las variables respuesta a la vez, y un análisis de una única variable respuesta. Ambos enfoques han sido comparados, interpretados y analizados, obteniendo resultados de los algoritmos empleados. Finalmente se ha concluido que el enfoque basado en predecir una única variable respuesta es mucho más preciso y que el algoritmo que mejor se ha adaptado a los datos ha sido *Support Vector Regression*. También se han obtenido resultados positivos utilizando redes neuronales en ambos enfoques.

Palabras clave: Machine Learning, Deep Learning, hormigón armado, acero, algoritmo.

Abstract

Nowadays, due to new technological and computational advances, it is possible to store and process a large amount of information and thanks to this it is possible to use data and technology in countless applications. In this case it focuses on the field of construction, in which reinforced concrete is a material that can present non-linear behaviors. When presenting these nonlinearities, modeling a numerical response can be a complex task. For this, an alternative is presented using several prediction algorithms based on Machine Learning and Deep Learning. To implement the algorithms, the Python programming language has been used. In addition, two approaches have been carried out, a multi-output analysis, which consists of predicting all the response variables at the same time, and an analysis of a single response variable. Both approaches have been compared, interpreted and analyzed, obtaining results from the algorithms used. Finally, it has been concluded that the approach based on predicting a single response variable is much more precise and that the algorithm that has best adapted to the data has been Support Vector Regression. Positive results have also been obtained using neural networks in both approaches.

Keywords: Machine Learning, Deep Learning, reinforced concrete, steel, algorithm.

Índice general

Índice de figuras

Índice de tablas

Lista de acrónimos

ML Machine Learning
DL Deep Learning
TCCs Teorías del campo a compresión
IA Inteligencia artificial
KNN K- nearest neighbors
RNP Redes neuronales profundas
LASSO Least Absolute Shrinkage and Selection Operator
SVR Support Vector Machine
SVM Support Vector Regression
RNA Redes neuronales profundas
RI Rango intercuartílico
MAE Mean absolute error
RMSE Root mean square error
NMSE Normalized mean square error

Capítulo 1

Introducción

1.1 Presentación

Desde el Imperio Romano hasta hoy en día el hormigón ha sido un elemento de construcción muy importante en infinidad de edificaciones. Consecuentemente a lo largo de la historia el hormigón ha ido evolucionando, tanto en los materiales utilizados como la forma de emplearlo [11]. Antes de todo convendría tener una idea clara de lo que es este material.

El hormigón es un material de construcción compuesto por una mezcla homogénea de cemento y agua, al que se le añaden una serie de compuestos agregados finos y gruesos que hacen que esta mezcla se mantenga unida. Como agregado fino se tiene la arena y dentro del agregado grueso se encuentra canto rodado o piedra partida. La mezcla del agregado fino, cemento y agua se llama mortero. El hormigón se elabora en dos fases, en la primera se encuentra en estado líquido y se produce la hidratación del cemento. En la segunda fase comienza el fraguado, que es cuando el hormigón pasa del estado líquido a sólido y empieza el endurecimiento del hormigón [12].

En la prehistoria la forma que tenían de construir el hombre era tallando rocas o piedras de modo que se mantuvieran estables. Más tarde las técnicas de construcción mejoraron y fue en el Paleolítico y Neolítico cuando el hombre empezó a construir muros con piedras y las colocó en forma de hilera lo más juntas posible. Después de esto, se comenzó a utilizar arcilla con canto rodado para formar una especie de conglomerante entre las uniones de las piedras y de esta manera favorecer una mejor repartición de las cargas. Posteriormente, ya en la Edad Antigua, los egipcios descubren otro tipo de conglomerante como la cal.

Con la llegada de la romanización, los procesos y técnicas de construcción mejoran y es cuando el hormigón alcanza su punto álgido. El hormigón que utilizaban en esa época estaba formado por cal, piedras naturales y arenas volcánicas, ya que cerca de Roma había gran facilidad para encontrar este tipo de arena, y todo mezclado hacía que se consiguiera unas propiedades similares a las que tiene el hormigón utilizado actualmente. Todo esto y juntado con la gran habilidad de construcción que tenían los romanos hizo que se construyeran edificaciones que todavía hoy perduran [11].

Como consecuencia de la caída del Imperio Romano se produce un cambio radical en la sociedad hasta el momento y una ruralización de esta. Esto hace que las necesidades constructivas cambien y la llegada de la mano de obra esclava provoca que se pierda la técnica de fabricación de cementos, lo que conlleva a una pérdida del uso del hormigón.

No fue hasta finales del siglo XVIII cuando, con el descubrimiento de aglomerantes modernos, empieza una etapa de resurgimiento del hormigón. Con la revolución industrial y el dominante uso del acero

en construcciones aparece el hormigón armado. Fue a partir de 1910 cuando se da cuenta de que el comportamiento del hormigón es parecido al de una piedra. Tiene la capacidad de tener mucha resistencia a compresión, pero, sin embargo, cuando trabaja a tracción la resistencia es menor. Es por ello por lo que se empieza a añadir acero, ya que a demás de tener un coeficiente de dilatación similar al hormigón también es protegido de la oxidación. Monier es la primera persona que explica el papel del acero en el hormigón y es que sirve para resistir las fuerzas de tracción que actúan sobre él. Además, se da cuenta de que la armadura debe estar dispuesta siguiendo la distribución de esfuerzos [13].

En la actualidad el hormigón es un material comúnmente utilizado en infinidad de estructuras y en el ámbito de la construcción. Por ejemplo, en estructuras civiles, tales como puentes, pasarelas y presas. Además, es muy común en todo tipo de edificios. Este material proporciona gran estabilidad y seguridad a todo tipo de construcciones. Como se ha mencionado el hormigón presenta una gran resistencia a compresión además de una elevada impermeabilidad. Una de las mayores desventajas del hormigón es su baja resistencia a tracción. Para contrarrestar este defecto, y poder trabajar con el hormigón, se añade la armadura, resultando en lo que se conoce como hormigón armado.

El comportamiento del acero de armadura es no lineal. Para resolver el problema de deformación de este, es necesario recurrir a métodos numéricos, tales como aquellos de tipo Newton-Raphson. Algunos de estos métodos no lineales para resolver los problemas no aportan soluciones reales de la deformación del acero, ya que estos modelos precisan de un trabajo complicado [14], y como consecuencia se hace más difícil resolverlo. Esto puede conllevar a que se proporcionen soluciones que no sean exactas. Normalmente, el problema se resuelve mediante métodos iterativos, pero al presentar estas no linealidades, en muchas ocasiones no basta con esta estrategia [15]. Para este tipo de casos en los que los métodos numéricos iterativos como el método Newton son incapaces de resolver los problemas no lineales.

Hoy en día con el rápido avance de las tecnologías y la gran cantidad de datos que se manejan y se procesan han aparecido nuevas estrategias computacionales que permiten la mejora de muchos procesos. Con el objetivo de facilitar la resolución del sistema numéricamente, en el desarrollo del presente trabajo, se propone el uso de técnicas de *Machine Learning* (ML) y *Deep Learning* (DL) [14]. Este tipo de técnicas pueden aprender relaciones entre los datos de entrada y la deformación del acero. De esta forma, se pueden evitar los problemas de solubilidad de los sistemas de ecuaciones, especialmente en aquellas zonas más complejas, es decir, aquellas más cercanas al borde de solubilidad y lograr la optimización del problema basado en sistemas de ecuaciones muy complejos y laboriosos de resolver [16].

1.2 Objetivos

El objetivo principal de este trabajo es desarrollar modelos de ML y DL que permitan obtener la deformación aparente de cedencia del acero en elementos de hormigón armado pudiéndolo aplicar en la construcción. De esta forma, se evita el uso de métodos numéricos, que tienen problemas de convergencia en este tipo de problemas.

Para el desarrollo de este trabajo, se hará uso de Python, para el análisis estadístico de las variables, uno de los objetivos es el aprendizaje continuo de este idioma de programación.

Otro de los objetivos es aprender distintas técnicas de ML, que van a ser utilizadas para la predicción de la deformación del acero. Esto conlleva el uso de modelos de regresiones, que son capaces de llegar a una solución donde otros tipos de procesos como los cálculos iterativos no pueden.

Por último, se tiene como objetivo interpretar los resultados y obtener conclusiones claras de ellos.

Capítulo 2

Estudio Teórico

2.1 Introducción

En esta parte se exponen los conocimientos previos y necesarios para el entendimiento del problema y su mejor interpretación. Primeramente, se darán conocimientos de estudios anteriores que reflejan problemas similares a los que se exponen en el presente documento. Y después, se explicará lo relacionado con el campo de la inteligencia artificial, el ML y DL. Este apartado es muy importante ya que sienta las bases del conocimiento necesario para entender cómo funciona este ámbito y cómo se va a llegar a la solución del problema planteado. Además, se hablará sobre la historia, la creación y el recorrido hasta hoy en día de este campo, así como de sus aplicaciones prácticas.

2.2 Estado del Arte

Uno de los problemas más importantes en el ámbito de la construcción es el de poder predecir las propiedades mecánicas de los materiales [17]. Un material fundamental en este campo es el hormigón, el cual trabaja muy bien ante esfuerzos de compresión, sin embargo, presenta desventajas significativas ante esfuerzos a tracción y cortantes [18]. Es por ello por lo que ha sido objeto de numerosos estudios. Desde hace varias décadas algunos de ellos se han centrado en la transmisión de esfuerzos cortantes [19], [20], [21]. Actualmente, se han establecido nuevos enfoques más completos del problema [22], [23], [15], [24]. Todos estos estudios que modelan el comportamiento de los elementos estructurales sometidos a cortante pertenecen a lo que se conoce como Teorías del Campo a Compresión (TCCs) [16]. Esta teoría fue desarrollada inicialmente por Collins [25], [26] aunque, posteriormente se introdujeron algunas modificaciones [19], [27]. La teoría inicial del campo de compresión [26] asumía que no existía tensión alguna una vez agrietado el hormigón, mientras que, el nuevo enfoque [19] tenía en cuenta los esfuerzos de tracción en el hormigón entre las grietas y se calculaban experimentalmente las relaciones tensión – deformación. Esta teoría establecía la condición de que cualquier deformación en el hormigón se traducía en una misma deformación en el acero. Sin embargo, la resolución del sistema de ecuaciones que proponen los modelos convencionales es una tarea compleja de resolver, ya que el hormigón armado introduce relaciones no lineales en los modelos estructurales [16].

Generalmente para el cálculo de estas expresiones se utilizan métodos iterativos, como el método tipo Newton. Encontrar las incógnitas de las ecuaciones planteadas precisa de gran dificultad y es un proceso laborioso en el que se requiere establecer una aproximación inicial y una región de solución para cada

grupo de datos de entrada. Muchas de las veces, las soluciones de estos métodos no convergen debido a que depende de la aproximación inicial de las incógnitas del sistema [28], [16].

Para superar los inconvenientes de los métodos convencionales, actualmente se han implementado técnicas de ML. Numerosos estudios se han centrado en predecir las propiedades mecánicas del hormigón armado tanto a compresión como a tracción [29], [30], [31], [32], [17], [33], [18]. Normalmente, los problemas de predicción de las propiedades del hormigón armado se han abordado empleando modelos de regresión, pero [31] utiliza un enfoque combinado de métodos de regresión y clasificación. Otros estudios enfocan el problema con técnicas de DL basándose en redes neuronales [34], [35].

Cabe mencionar que también se han utilizado estas técnicas de ML para identificar el inicio de la corrosión en las barras de acero del hormigón permitiendo mejorar la durabilidad de las estructuras de hormigón armado. La corrosión afecta a las barras de acero haciendo que se reduzca la sección y que se pierda resistencia a tracción y flexión [36]. También, otro estudio [37] se ha centrado exclusivamente en predecir la capacidad a tracción de las barras de refuerzo de acero empalmadas y no empalmadas.

2.3 Inteligencia artificial, Machine learning y Deep learning

2.3.1 Introducción

La inteligencia artificial (IA) es un área de estudio dentro del mundo de la computación, que tiene como finalidad la resolución de problemas tecnológicos que precisan de inteligencia humana [1]. La IA hace que los ordenadores imiten cómo se comporta el ser humano y que hace que sea posible la resolución de problemas sin ninguna o una mínima intervención humana [38]. Cuando a las computadoras se les otorgan estas habilidades adquieren funciones que son imposibles en el ser humano, como es la velocidad de procesamiento de los datos y la capacidad de no descansar para su funcionamiento [39]. La finalidad, en este caso, del uso de la IA es poder predecir la deformación del acero en elementos de hormigón armado.

Dentro del mundo de la IA, existen distintos campos de estudio como pueden ser el *Machine Learning* (ML) y el *Deep Learning* (DL).

El ML es una rama de la inteligencia artificial que tiene como función el aprendizaje en una computadora a partir de métodos computacionales para lograr resolver un determinado problema utilizando la experiencia sin tener que estar programado específicamente para ello. Estos métodos computacionales están basados en instrucciones matemáticas, llamados algoritmos. Para que el algoritmo pueda funcionar es necesario introducir datos. Estos datos tendrán una estructura determinada, formada por datos de entrada al modelo que se encargarán de ajustar una o múltiples variables de salida, que serán los resultados [1]. El algoritmo aprende en base a todos los datos que le llegan de entrada y proporcionará un resultado que será validado. Estos dos procesos se realizan de forma separada para evitar modelos incorrectos. Para ello los datos son divididos en datos de entrenamiento, que es el proceso de aprendizaje del algoritmo, y datos de validación utilizados para verificar la precisión del modelo. Esto permite predecir resultados, hacer clasificaciones e identificar patrones.

Por otra parte, el DL es un subcampo del ML, pero en este caso está basado en una estructura de redes neuronales similar a un modelo biológico [1]. Se entrará en detalle más adelante.

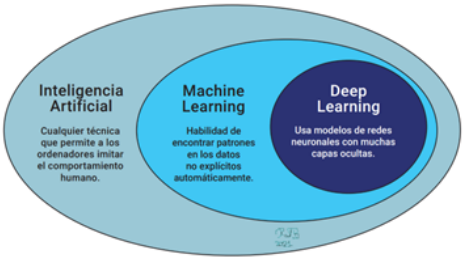

Figura 2.1: Partes del campo de la Inteligencia Artificial y características [1]

2.3.2 Historia y antecedentes

El *Machine Learning* es un subcampo que proviene de la inteligencia artificial y se ha convertido en un avance fundamental para el desarrollo global. Para entender dicha evolución es necesario explicar el inicio de este concepto.

Nos remontamos a el año 1943 cuando se presentó el trabajo realizado por el matemático Walter Pitts y el neurocientífico Warren McCulloch. Este trabajo se enfocaba en la teoría de la comparación del cerebro humano como si fuera una máquina computacional y de la creación de máquinas capaces de trabajar como una red neuronal humana [40]. Sin saberlo, esta suposición era un preámbulo de lo que más tarde se conocería como inteligencia artificial.

El término inteligencia artificial fue nombrado por primera vez en 1956, en una conferencia en la que participaban John McCarthy y Martin Minsky entre otros. Esta conferencia científica tuvo lugar en Darthmouth y a partir de entonces, el término inteligencia artificial, daría nacimiento al nombre de uno de los campos de la informática [40].

Antes de este acontecimiento, concretamente en el año 1950, el matemático británico Alan Turing conocido como ser uno de los precursores de la informática moderna, publicaba un artículo titulado *Computing Machinery and Intelligence* [41]. En este escrito se establecía una pregunta al principio del artículo, ''¿Pueden las máquinas pensar?" [41] y para ello propuso el Test de Turing. La finalidad era comprobar como de inteligente podría ser la máquina tratando de imitar las respuestas humanas. La prueba consistía en mantener una conversación escrita entre el interrogador, una computadora y otra persona, en la que el interrogador intentaría averiguar cuál de los dos es la máquina. Si el interrogador no era capaz de diferenciar al ordenador del ser humano, la máquina pasaría la prueba [42].

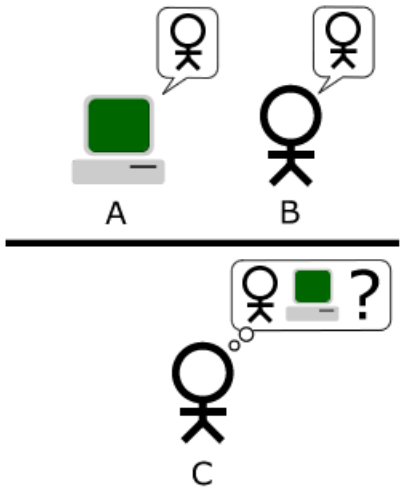

Figura 2.2: Test de Turing [2]

Turing también es conocido por la creación de la Máquina de Turing, capaz de resolver problemas que puedan ser representados por un algoritmo. Además, tuvo un papel muy importante en la Segunda Guerra Mundial, descifrando los códigos nazis de la máquina Enigma [43].

Posteriormente, en 1952, Arthur L.Samuel conocido por ser uno de los pioneros en el campo de la inteligencia artificial, presentó el primer programa que podía aprender por sí mismo y tenía como función jugar a las damas. Debido a la capacidad de información que podía almacenar el programa, era capaz de aprender de los errores que cometía y así mejorar su respuesta [40].

Cinco años más tarde, en 1957, Frank Rosenblatt, un psicólogo estadounidense, creó el *Perceptron* que era una red neuronal capaz de aprender de forma parecida a los sistemas biológicos. Este acontecimiento supuso que recibiera una buena reputación internacional y que se pusieran grandes expectativas en su trabajo. Con la publicación del libro *Perceptrons: an introduction to computational geometry* [44] en 1969, se presentó que el *Perceptron* no podía resolver problemas no lineales. A partir de este hecho, comienza una etapa de 15 años de estancamiento en la investigación sobre la inteligencia artificial, sobre todo en el campo de las redes neuronales, que se conoce como el invierno de la IA [45]. Durante la década de los 70 se producen pocos avances, debido a la falta de presupuesto, pero aun así en 1979 en la universidad de Stanford, se consigue inventar un vehículo autónomo, el *Stanford Car* [40]. El robot estaba formado por un monitor supervisado por control remoto y a partir de imágenes que obtenía del entorno era capaz de moverse evitando obstáculos presentes en su camino [46].

Este hito fue posible gracias al algoritmo *k-nearest neighbors* (KNN). En 1951, Evelyn Fix y Joseph Hodges plantearon un método de clasificación no paramétrico de patrones, pero no fue hasta 1967 cuando Thomas Cover hizo una ampliación del algoritmo, introduciendo las distintas reglas [47]. El KNN tenía la función de reconocer patrones y de esta manera permitió a la computadora predecir una posible solución al problema planteado. Esto sirvió de precedente y la década de los 80 quedó marcada con la aparición de los sistemas expertos, que fueron de interés comercial. Estos sistemas se basaban en reglas, uno de los más conocidos fue el MYCIN creado por Edward Shortliffe en la universidad de Stanford, que fue utilizado en decisiones diagnósticas médicas [43].

Seguidamente aparecieron otros modelos como el creado por Gerald Dejong en 1981, en el que introduce la idea de *Explanation Based Learning* (EBL), que consiste en el aprendizaje de un sistema inteligente mediante ejemplos, que son los datos de entrenamiento [48]. Esta idea sentó las bases de lo que se conoce hoy en día como aprendizaje supervisado.

Más tarde, concretamente en 1985, se produjo otro avance importante de la mano de Terry Sejnowski, el cuál inventó un programa capaz de pronunciar palabras a nivel escolar, llamado *NetTalk* [40].

Un año más tarde, David Rumelhart y James McClelland crean una red neuronal multicapa en la que no se diferencian neuronas de entrada o de salida, sino que todas pueden actuar como entrada y como salida, y están conectadas, por lo que existe una realimentación entre ellas. Esta idea de red neuronal realimentada fue presentada anteriormente por John Hopfield a principios de los años 80 y se conoció por Red de Hopfield [49].

Después de estos años llenos de avances en el campo de la IA y el ML, se produjo una segunda paralización en esta rama de la IA que duró hasta mediados de la década de los 90, época en la que el rumbo del ML se torna más bien al análisis de grandes conjuntos de datos, desvinculándose así, como una herramienta de la IA [40]. Finalmente, a pesar de los pocos avances que se produjeron durante la década de los 90, en 1995, Vapnik y su equipo desarrollaron el algoritmo *Support Vector Machines* (SVM). Dos años más tarde, en 1997, Freund y Schapire presentaron AdaBoost, un modelo de clasificación sencillo dispuesto en secuencia [49]. En el mismo año se produjo otro acontecimiento importante, el ordenador *Deep Blue* desarrollado por la empresa tecnológica IBM logró vencer a Gary Kaspárov, campeón mundial de ajedrez [40].

Posteriormente con la llegada del nuevo milenio, se presenta en 2001 de la mano de Leo Breiman el algoritmo *Random Forests* (RF), que se basaba en un conjunto de árboles de decisión dispuestos aleatoriamente y que presentaba mejores resultados que el modelo propuesto por Freund y Schapire [49].

Como se ha mencionado, en los años 90 el ML empezó a cambiar el rumbo hacia el procesamiento de datos, y con la entrada de los años 2000 y la ayuda de grandes empresas presentes en este sector como IBM, Google y Microsoft, el ML comenzó su expansión en un mundo cada vez más enfocado en el tratamiento y análisis de grandes cantidades de datos. Así fue como en 2006 Geoffrey Hinton acuñaba el término *Deep Learning*, un subcampo del ML basado en redes neuronales. Más tarde en 2008 se presenta la beta de *Azure Machine Learning*, un programa desarrollado por Microsoft que permite guardar aplicaciones a modo de nube para sus usuarios [40].

Tres años después, en 2011, IBM saca a la luz su ordenador Watson que logró vencer a personas en un concurso que consistía en responder preguntas. Era capaz de responder de la manera más precisa posible a preguntas realizadas en tiempo real analizando el lenguaje natural. De esta manera se observó que podía ser utilizado para extraer y analizar grandes cantidades de datos dada su gran capacidad de análisis del lenguaje [50].

Un año más tarde, Jeff Dean y Andrew Ng se unen para formar parte de un proyecto llamado *Google Brain* que trataba de implementar redes neuronales en el reconocimiento de imágenes [40].

De nuevo Geoffrey Hinton en 2012 utilizando estas Redes Neuronales Profundas (RNP) revolucionó el campo del reconocimiento de imágenes y el ML, ganando con ayuda de un estudiante suyo, el desafío Imagenet [45]. Este tipo de arquitectura tan elaborada sirvió como base para el reconocimiento de personas utilizando algoritmos como el *DeepFace* de Facebook, además también se utilizó en temas militares y sistemas de seguridad [40].

En 2014 Google compró la empresa DeepMind, dedicada al DL, la cual sobresalió en su actuación en juegos Atari. Se utilizó un algoritmo basado en RNP que era capaz de reconocer los píxeles de la pantalla como si fuera un humano y tras un tiempo de entrenamiento logró derrotar a jugadores profesionales en este tipo de juegos [40]. Dos años más tarde de nuevo la empresa DeepMind marcó otro hito en el campo de la IA cuando con la ayuda del algoritmo AlphaGo, derrotó al mejor jugador profesional del mundo del juego de mesa chino Go. Este juego es considerado como uno de los más complejos debido a que es más intuitivo, lo que hace que sea más difícil utilizar enfoques algorítmicos de ''fuerza bruta". A diferencia de

Deep Blue utilizado para vencer al jugador de ajedrez Gary Kaspárov, mencionado anteriormente, que analizaba los movimientos posibles para obtener el mejor resultado, AlphaGo se entrenaba así mismo y mejoraba con cada juego que hacía, esto se conoce como aprendizaje por refuerzo [51].

En 2015 Amazon saca su propia plataforma de ML y Microsoft se reinventa creando otra llamada Distributed Machine Learning Toolkit. En el mismo año se funda la compañía Open AI, invirtiendo en ella muchos recursos, dedicada a la investigación y el avance de la IA [40].

Así es cómo ha ido evolucionando el campo de la IA, el ML y DL hasta lo que conocemos hoy en día. Actualmente no solo se utiliza en sectores empresariales o informáticos que acumulan grandes cantidades de datos para su análisis y procesamiento, sino que el ML se utiliza en otros sectores que puede servir de gran utilidad hoy en día.

2.3.3 Aplicaciones del Machine learning y Deep learning

Hasta ahora se han ido mencionando algunas de las aplicaciones que puede tener el ML y el DL y se pueden clasificar de la siguiente manera:

- Visión artificial: este campo de aplicación engloba la detección y reconocimiento de objetos o cualquier elemento que se pueda visualizar.
- Análisis semántico, lenguaje natural y de la información: el análisis semántico se refiere a la capacidad de relacionar de manera escrita párrafos, oraciones y palabras. También es muy importante la capacidad para procesar el lenguaje natural, además de la búsqueda de información y de datos.
- Predicción: en este campo de aplicación destacan la clasificación y análisis de textos e imágenes, así como la predicción de un diagnóstico médico o de ataques en la red en el campo de la seguridad [49].

Visión artificial

Este ámbito forma parte del ML y DL y en él se estudia la capacidad que tienen las máquinas en el reconocimiento y análisis de elementos visuales. Dentro de este campo se encuentra la detección y reconocimiento de objetos.

Hoy en día se ha ampliado con la llegada de nuevas tecnologías, como puede ser el reconocimiento facial y el escáner de iris. El reconocimiento facial ha sido aplicado en temas de seguridad, ayudando a reconocer a ciertas personas que puedan ser peligrosas e interfieran en el bienestar común. De la misma manera también se utiliza para mejorar la accesibilidad reconociendo las caras de las personas y permitiéndolas pasar o controlando su asistencia, por ejemplo, en las empresas [52].

La visión artificial también es utilizada para el reconocimiento de gestos con la mano entre una persona y la máquina. Este campo de aplicación del ML es utilizado alarmas, que presentan un sistema de detección de objetos para avisar de intrusos [49].

Análisis semántico, lenguaje natural y de la información

El campo del análisis semántico se centra en el relacionar las estructuras escritas, mientras que el lenguaje natural busca la forma correcta de programar a la computadora para que sea capaz de procesar y analizar la gran cantidad de datos del lenguaje natural. Además, este ámbito del ML y DL se encarga de la búsqueda de información y datos, ya puedan ser documentos, sonidos e imágenes [49].

Este tipo de aplicaciones se utilizan en ámbitos escolares o educativos, en los que la digitalización de documentos escritos a mano es muy útil y facilita mucho el trabajo. Otro ejemplo es el reconocimiento por voz, que consiste en convertir las palabras habladas a texto. Esto se aplica sobre todo en la creación de asistentes de voz, que proporcionan beneficios a la hora de conducir automóviles o simplemente en la vida cotidiana [52]. También es de gran utilidad a la hora de traducir palabras en diferentes idiomas que la computadora reconozca [39].

En el campo sanitario o de la salud, estos tipos de algoritmos permiten visualizar en tiempo real cualquier variación de los parámetros de una persona mediante el procesamiento de datos [52].

Modelos de predicción

Este campo de aplicación del ML y DL agrupa la clasificación, el análisis de los datos y la recomendación. Se basa en la observación de datos históricos para realizar predicciones.

Una de las aplicaciones más destacada es el ámbito financiero, en el que se utilizan técnicas de ML y DL para detectar y predecir movimientos fraudulentos o estafas en transacciones de dinero digital, analizando patrones en los movimientos bancarios de los clientes o acciones extrañas. También se utiliza este tipo de técnica para predecir el precio de las acciones [52], poder saber qué proyectos pueden generar mayores beneficios, identificar clientes, sus preferencias y comportamientos y prever los ingresos de una empresa [39].

Otro campo importante de aplicación es el de la salud. Las técnicas de predicción permiten realizar pronósticos médicos, como el diagnóstico de enfermedades, identificar grupos de mayor riesgo o predecir la propagación de enfermedades [52]. Todo esto es gracias a la gran cantidad de datos que se obtienen de registros médicos utilizados para el entrenamiento de modelos predictivos. En cambio, si se utilizan datos administrativos es posible predecir la asistencia de los pacientes y poder organizar de manera óptima la gestión. Es posible predecir también la cantidad de fallecimientos que se producirán en un hospital, así como la capacidad de analizar imágenes médicas y poder detectar anomalías en ellas permitiendo la selección de un tratamiento adecuado. El análisis de imágenes médicas requiere mediciones, que en manos de un médico podría depender del observador y podría influir en los resultados [53].

También se utiliza para predecir y detectar intrusiones o ataques en la red, así como la clasificación de textos e imágenes [49].

En el ámbito industrial, como es en este caso, las técnicas de ML y DL se van a utilizar para intentar predecir la deformación del acero en estructuras de hormigón armado. Además, también es utilizado, por ejemplo, para predecir energías generadas en distintos sectores de la industria o prever fallos en maquinaria que podrían afectar en retrasos temporales y costes innecesarios.

2.3.4 Tipos de variables

En cuanto a las variables, hay que tener claro los conceptos y definiciones de los tipos que se van a tratar en el desarrollo del problema. Se pueden clasificar de varias maneras:

Una primera clasificación podría ser separarlas dependiendo de la forma en que se observan. De esta manera se tienen variables cualitativas o categóricas y cuantitativas o numéricas.

- Las cualitativas hacen referencia a las variables que no pueden ser medidas de forma numérica, como por ejemplo el sexo, género, color de ojos, etc. Este tipo de variables son principalmente utilizadas en problemas de clasificación.

- En cambio, las variables cuantitativas pueden ser medidas de forma numérica y expresarse con cifras. En este grupo se encuentran, por ejemplo, la deformación del acero, la tensión que soporta, etc [54]. Las variables numéricas son generalmente empleadas en problemas de regresión.

Una segunda clasificación sería dependiendo de las relaciones de causalidad. De esta forma, se encuentran variables dependientes e independientes.

- Las independientes son las variables de entrada del modelo, es decir, las que explican los resultados [54]. También se las puede llamar como predictoras, antecedentes o regresoras [55].
- Las variables dependientes son las magnitudes objetivo, es decir, las variables explicadas por las magnitudes independientes [54]. Pueden ser nombradas como, consecuentes o predichas [55]. En resumen, las variables independientes son las elegidas para comprobar la causalidad que existe con las magnitudes dependientes [55].

2.3.5 Tipos de aprendizaje

Aprendizaje supervisado

El aprendizaje supervisado consiste en enseñar a la máquina a partir de un modelo basado en ejemplos. Los ejemplos están formados por un grupo de datos llamados datos de entrenamiento, que a su vez contienen las variables de entrada y salida del modelo. De esta manera, el grupo de datos de entrenamiento es utilizado para el ajuste del modelo, ya que se sabe cómo va a ser el resultado y cómo están clasificados a partir de las diferentes variables de entrada. Para la evaluación del modelo se utiliza una parte de los datos, los cuales no han sido utilizados en el proceso de entrenamiento, llamados datos de prueba o test [56]. De esta manera, se intenta establecer una función que relacione las variables de entrada con las de salida [57]. Dentro del aprendizaje supervisado se encuentran los problemas de clasificación y los problemas de regresión. Principalmente, la regresión predice en función de las variables de entrada, un valor numérico o cuantitativo, mientras que la clasificación predice la categoría a la que pertenece un determinado dato [10].

El problema tratado, el cálculo de la deformación del acero, se va a centrar en modelos de predicción basados en el aprendizaje supervisado. Algunos de los algoritmos que se van a emplear son la regresión lineal, árboles de decisión, Lasso, *K-Nearest Neighbors* (KNN), *Support Vector Machine* (SVR), *Random Forest*, *AdaBoost* y *Gradient Boosting*. A continuación, se presentan de manera resumida dichos algoritmos:

Regresión lineal

Este tipo de regresión es uno de los más sencillos, dentro de los algoritmos que presenta el ML. Para llevar a cabo el análisis predictivo se hace uso de una base matemática [58]. La regresión lineal como su nombre indica, hace referencia al cálculo de la ecuación de la línea recta que mejor se ajusta a la nube de puntos explicada por la variable de entrada y la de salida [59]. Para establecer la ecuación lineal que relaciona ambas magnitudes se utiliza el método de mínimos cuadrados. Este método consiste en minimizar los residuos, es decir, la diferencia al cuadrado entre los valores reales y los puntos predichos por la recta estimada a partir de las variables del modelo [60]. Cuando se obtenga la línea que ajusta el modelo, se observará que algunos puntos estarán más alejados que otros, que se encontrarán más cercanos a la recta de regresión. Cuanto menor sean los residuos, es decir, cuanto más cercana sea la distancia entre los puntos reales y la recta, mejor va a ser el resultado del ajuste lineal [59].

Dependiendo del número de variables independientes que conforme el modelo se va a diferenciar entre regresión lineal simple o regresión lineal múltiple.

La regresión lineal simple consta de una única variable independiente, es decir, una sola variable predictora. Este modelo está formado por la ecuación de la recta de regresión:

$$y = \beta_0 + \beta_1 x \tag{2.1}$$

Donde y es la variable dependiente y x es la variable independiente o predictora. La constante β_0 es el valor cuando la línea corta por el eje de ordenadas, es decir, cuando la x es igual a cero y β_1 es la pendiente de la recta. Todo esto se traduce en que la pendiente depende del efecto que tiene la variable regresora sobre la variable predicha [59]. Cada vez que se introduce un nuevo dato varía la pendiente de la recta, es por ello el peligro de incluir valores atípicos en el modelo. Aunque son datos que pertenecen al modelo, pueden suponer grandes variaciones en la pendiente de la recta. Como el método se realiza por mínimos cuadrados y hace la diferencia entre el punto observado y la ecuación de la recta de regresión, provoca cambios significativos en el ajuste del modelo [60]. Cada valor real y_i se puede expresar siguiendo la función:

$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i \tag{2.2}$$

Donde β_0 y β_1 son los parámetros del modelo, x_1 es el punto dónde se observa y ϵ_i son los errores aleatorios que es el responsable de la diferencia entre el dato real y el predicho [59].

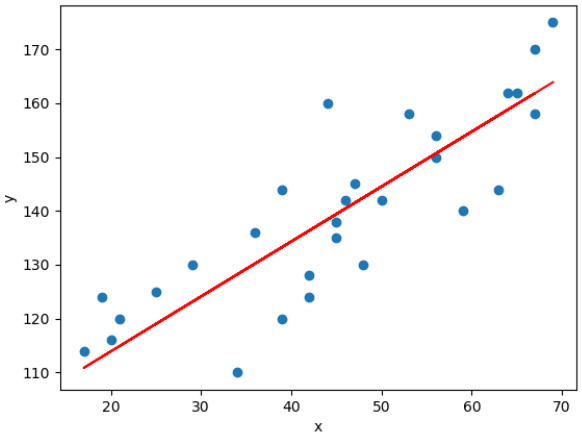

Figura 2.3: Regresión lineal simple [3]

La figura 2.3 muestra un modelo de regresión lineal simple formado por dos variables (una la variable dependiente y la otra la independiente) representadas por los puntos azules, que son los puntos observados reales, mientras que la línea roja representa los datos predichos por la variable predictora. La distancia entre los datos observados reales a la recta de regresión hace referencia a los residuos.

En muchas ocasiones la regresión simple no es suficiente para explicar un modelo en el que pueden influir más de una variable predictora. En este caso se utiliza la regresión lineal múltiple [61]. Este método, a diferencia del simple en el que se encontraba una única variable independiente, consta de dos o más variables predictoras. Este modelo está formado por la siguiente estructura:

$$y = \beta_0 + \beta_1 x_1 + ... + \beta_k x_k + \epsilon \tag{2.3}$$

De nuevo se tiene que, y es la variable dependiente, $(x_1, ..., x_k)$ son las distintas variables predictoras y ϵ es el error aleatorio [59].

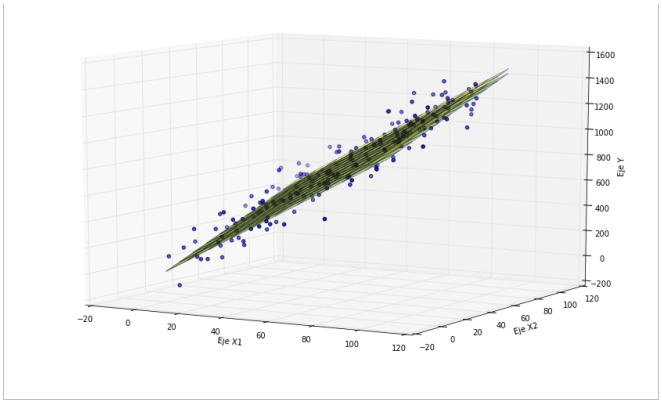

Figura 2.4: Regresión lineal múltiple [4]

La figura 2.4 muestra un modelo de regresión lineal múltiple en el que concretamente se representan dos variables predictoras. En este caso los datos reales están representados en un eje tridimensional y la recta de regresión que se tenía en el modelo simple ha sido sustituida por un plano tridimensional.

Árboles de decisión

Este modelo de predicción utiliza una serie de datos para establecer unas reglas a partir de las características y la información que tiene cada uno de ellos [62]. Representa todos los sucesos posibles que se podrían dar para cada una de las decisiones tomadas en ciertos momentos, ayudándonos a elegir la mejor decisión utilizando un enfoque probabilístico. Se utiliza en modelos de clasificación, segmentación y predicción [63].

La representación gráfica de un árbol de decisión está formada por un conjunto de nodos, hojas y ramas. El modelo parte de un nodo principal llamado nodo raíz, a partir del cual surgen el resto de nodos internos dependiendo de las preguntas y respuestas que se le hagan al atributo. Los nodos hoja son los nodos finales y son los que contienen la decisión tomada de la variable dependiente [64]. La rama hace referencia a los diferentes caminos que se pueden tomar cuando elegimos una determinada decisión [63].

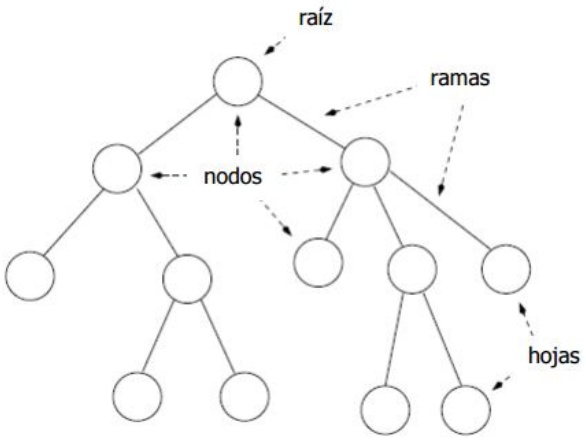

Figura 2.5: Árbol de decisión [5]

La obtención de un árbol de decisión contiene dos etapas. La primera etapa es la construcción del árbol a partir de un subconjunto de datos más pequeño llamado conjunto de entrenamiento. El árbol se comienza con un nodo raíz, que es una de las variables predictoras, y a partir de él se van generando divisiones del conjunto de entrenamiento dependiendo de los valores que pueda tomar el atributo en cada uno de los nodos. Para cada una de estas divisiones se genera de nuevo un nodo interno en el que se encuentra otra variable de predictora. De la misma manera se divide de nuevo el conjunto ya seccionado en subconjuntos y así sucesivamente, hasta que finalmente se forma un nodo hoja. La segunda etapa es la clasificación o predicción con la otra parte de los datos a partir del árbol generado. En esta parte cada dato recorre un camino desde el nodo principal hasta un nodo hoja. Los datos recorrerán distintas ramas dependiendo de las decisiones que se tomen en cada nodo [64].

Las ventajas de utilizar este método de regresión es que al tratarse de un modelo visual hace más sencilla la interpretación de los resultados, así como de las decisiones que toma. Además, puede reducir el número de variables predictoras o independientes [63].

Lasso

Algunos de los métodos de regresión empleados que utilizan un grupo de variables independientes para la construcción de modelos pecan de incluir más variables de las necesarias para explicar dicho modelo. Además, tienden a sobreestimar qué tan bien se ajusta el modelo, ya que funciona bastante mal ante la aparición de valores extremos. Uno de los métodos que intenta solucionar estos problemas es LASSO (*Least Absolute Shrinkage and Selection Operator*).

LASSO se basa en la selección de las variables que hacen que se minimice el error de la predicción [65]. La selección de las variables se lleva a cabo imponiendo una restricción a las magnitudes, que hace que para algunos coeficientes se produzcan estimaciones no nulas, mientras que otros coeficientes de regresión se reduzcan a cero. Al reducir los coeficientes del modelo hace que se reduzca la variabilidad de las predicciones [66]. La reducción de los coeficientes a cero se lleva a cabo haciendo que la suma de los coeficientes en valor absoluto sea menor que un determinado valor λ [65]. Este parámetro normalmente es elegido mediante un proceso automático de validación cruzada y es especialmente importante para que permita el mejor ajuste de los datos, ya que para cada valor de λ, existe un conjunto distinto de coeficientes de regresión [67]. Los datos se dividen aleatoriamente en subconjuntos. Todos los subconjuntos excepto unos son escogidos para el desarrollo del modelo de predicción, mientras que el subconjunto restante es utilizado para la validación del modelo. El proceso se repite de manera que todos los subconjuntos van rotando por el proceso de validación, y los demás de nuevo, para el desarrollo del modelo. Esto hace que finalmente se elija el parámetro λ que mejor convenga de entre todos los obtenidos en las validaciones [65].

Una de las ventajas de este método es que al seleccionar las variables hace que se obtengan modelos más simplificados y fáciles de interpretar. Además, LASSO se espera que sea mejor en modelos con variables predictoras con coeficientes mayores que otras variables con coeficientes cercanos o iguales a cero [66].

K-Nearest Neighbors

K – vecinos más cercanos (KNN, del inglés *K – Nearest Neighbors*) es un método normalmente utilizado como modelo de clasificación, pero también se puede utilizar para modelos de regresión. Su funcionamiento se basa en medir las distancias que hay entre dos datos, el dato de validación o de prueba y los datos de entrenamiento [47]. El valor que se intenta predecir es calculado como una interpolación de los puntos más

cercanos al dato de prueba. El vecino más cercano es obtenido con la distancia Euclidiana más pequeña de entre todos los datos de entrenamiento y el dato de validación [68].

$$d(x_t, x_i) = \sqrt{\sum_{n=1}^{N} w_n (x_{t,n} - x_{i,n})^2} \tag{2.4}$$

Donde x_t es el dato de validación, x_i es el dato de entrenamiento y w_n es el peso [6]. El peso refleja la importancia y es asignado en función de la distancia a la que se encuentren los datos de entrenamiento al dato de prueba. Por tanto, es una magnitud que varía inversamente asignando mayor peso a las distancias menores, que son las que tienen mayor importancia en la validación del modelo. De esta manera se soluciona el problema cuando se elige un número elevado de vecinos para la predicción del nuevo dato, haciendo que los puntos más alejados tengan poca influencia sobre el valor predicho [69].

KNN es un método de aprendizaje perezoso, es decir, el conjunto de entrenamiento es almacenado y utilizado para el aprendizaje en el momento de la regresión [70].

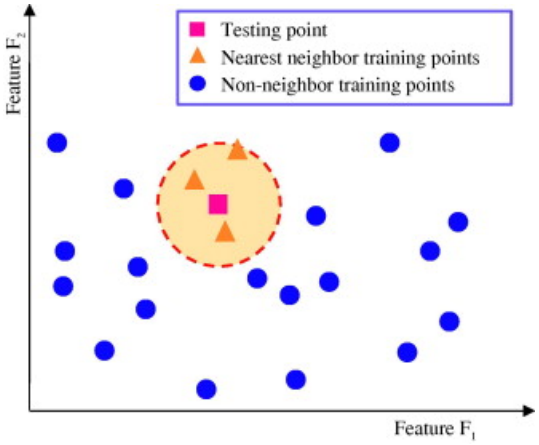

Figura 2.6: K - vecinos más cercanos a un punto de validación [6]

La figura 2.6 muestra una representación del algoritmo K - vecinos más cercanos en el que el cuadrado rosa hace referencia al dato de validación, mientras que los triangulos naranjas representan los vecinos más cercanos al dato de validación medidos por la distancia Euclidiana. El resto de datos fuera del perímetro son excluidos como parte de la predicción.

Support Vector Regression

El método Regresión de Vectores de Soporte hace referencia a las siglas SVR del inglés *Support Vector Regression* y proviene de las Máquinas de Vectores Soporte (SVM, en inglés, *Support Vector Machine*). Fueron creadas en los años 90 para realizar proyectos sobre aprendizaje automático y modelos estadísticos [71]. Además, las SVM se utilizaban en problemas de clasificación binaria en dónde los datos eran agrupados en distintas clases. Pero hoy en día son empleadas para resolver también problemas de regresión en donde los resultados son cuantitativos [7]. Algunas de las ventajas de utilizar este método es que se defiende bastante bien analizando patrones en muestras pequeñas y no lineales [71].

El campo de las SVM centrado en el estudio de la regresión es SVR, que se encarga de analizar tanto problemas lineales como no lineales. En los problemas lineales, que se tiene un conjunto de datos de entrenamiento, los cuales pueden ser ajustados mediante una ecuación lineal o hiperplano, la finalidad

del problema es establecer los parámetros w_i y b que definan la ecuación lineal de ajuste [7].

$$f(x) = (w_1 x_1 + ... + w_d x_d) + b =< w, x > +b \tag{2.5}$$

Al modelo se le asigna una tolerancia, ya que es complicado que todos los valores se puedan ajustar de manera lineal. De esta manera, se permite que exista un cierto error entre el dato predicho y el valor real. Para permitir este error, se establece una función lineal con una zona de una cierta anchura que será de 2ϵ. Los puntos que se encuentren dentro de esta región no serán definidos como vectores de soporte, haciendo que se minimice su número. También son creadas dos variables, que se encargan de la holgura para medir el error, ξ^+ y ξ^-. Por tanto, si $\xi^+ > 0$, el valor predicho será mayor que el real y estará por encima de ϵ. En el caso que $\xi^+ = 0$, el valor predicho será cero. De la misma manera ocurre si $\xi^- > 0$, el valor predicho, esta vez, será menor que el real. Igualmente, si esto no se cumple el valor predicho será cero [7].

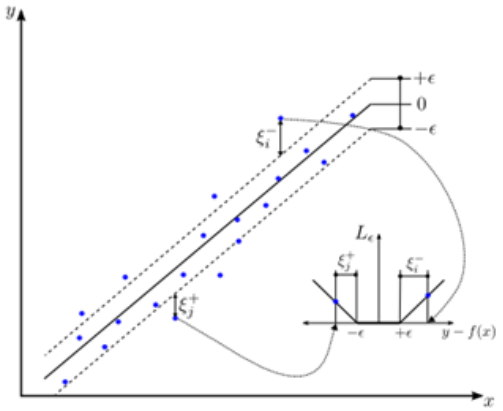

Figura 2.7: Parámetros del algoritmo SVR [7]

El problema asociado es un problema de optimización, en el que, imponiendo una serie de restricciones, trata de resolver el problema de forma dual. Para realizar esta transformación se hace uso de la función de Lagrange, que utiliza la función que se pretende minimizar y las restricciones correspondientes.

$$L(x, \alpha) = f(x) + \sum_{i=1}^{n} \alpha_i g_i(x) \tag{2.6}$$

Donde $f(x)$ es la función original, α_i son los multiplicadores de Lagrange y $g_i(x)$ las restricciones asociadas. De esta manera, la función de predicción de la transformación del problema original al dual quedaría:

$$f(x) = \sum_{i=1}^{n} (\alpha_i^{*-} - \alpha_i^{*+}) < x, x_i > +b^* \tag{2.7}$$

Donde α^+ y α^-, corresponden a los vectores de soporte, que son los puntos que se encuentran en contacto con la anchura máxima de separación del hiperplano y la solución está basada en estos vectores. Cabe mencionar que los datos que se encuentran fuera de la zona de anchura 2ϵ también contribuyen en la formación del hiperplano [14].

Cuando el problema a resolver presenta no linealidades, el conjunto de datos presentes en el espacio original se transforman en el espacio de características. Este espacio permite al modelo transformado ser ajustado de manera lineal [7]. Por tanto, el problema no lineal original se transforma en uno lineal en el espacio de características, y viceversa [71]. La transfromación asociada a los datos dependerá de una

función kernel, que puede ser de varios tipos como: funciones lineales, polinómicas, en la que se elegirá el grado de la función, gaussiano y sigmoidal. Esta es la función lineal que representa los valores predichos en el espacio de características:

$$f(x) = \sum_{i=1}^{n} (\alpha_i^{*-} - \alpha_i^{*+}) K(x, x_i) \tag{2.8}$$

Donde K es la función kernel [7].

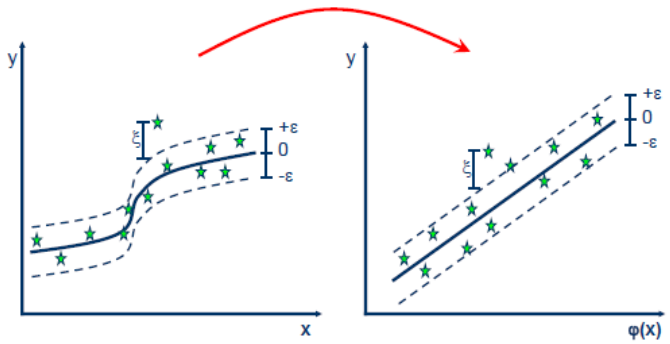

Figura 2.8: Transformación de un modelo no lineal al espacio lineal de características [8]

La figura 2.8 muestra la transformación de un modelo no lineal, utilizando una función kernel, al espacio de características dónde es posible poder ajustar el modelo de manera lineal.

Random Forest

Este modelo de predicción es muy parecido al árbol de decisión, pero la diferencia es que combina distintos árboles de decisión, en vez de utilizar uno solo. Dichos árboles de decisión son totalmente independientes unos de otros y no están correlacionados entre sí [72], debido a que son construidos con diferentes grupos de datos de entrenamiento a partir de un método que se llama embolsado.

El proceso de embolsado es un método de entrenamiento que consiste en dos etapas. En la primera se elaboran grupos de datos independientes de manera aleatoria a partir de la base de datos original. En la segunda etapa, el modelo es entrenado en base a los grupos de datos creados anteriormente [14].

Como se ha mencionado anteriormente, un árbol de decisión está formado por ramas que se forman a partir de las sucesivas divisiones. En cada rama hay presentes subconjuntos de nodos aleatorios, dónde a su vez, se producen más decisiones aleatorias. Este proceso se repite hasta que finalmente queda una hoja, es decir, un nodo terminal. Existe un parámetro en el modelo que determina las muestras por hoja de cada árbol de decisión. De esta manera, y a partir del método de embolsado se realiza el promedio de los valores del nodo hoja obtenidos, para determinar el valor predicho definitivo. El valor de predicción que finalmente se obtendrá, será el promedio de todos los valores predichos de cada uno de los árboles que componen el modelo [9].

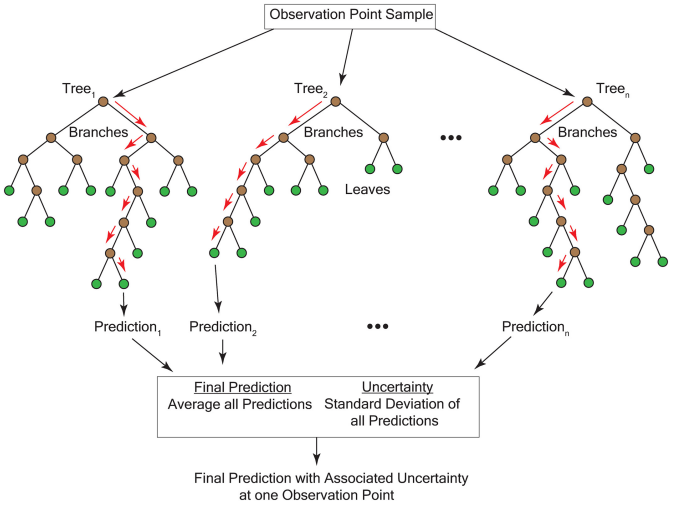

Figura 2.9: Modelo de bosque aleatorio [9]

La figura 2.9 pertenece a un modelo de bosque aleatorio en el que se representan los distintos árboles de decisión y el camino seguido hasta obtener la predicción de cada uno de ellos. Psteriormente se elegirá la saolución como un promedio de cada una de las predicciones de los árboles de decisión.

AdaBoost

Este algoritmo fue creado en 1996 por Freund y Schapire. Inicialmente fue utilizado como clasificación binaria. Posteriormente, en 1997 sus mismos autores ampliaron el algoritmo para adaptarlo a modelos de regresión. Esta primera adaptación tiene por nombre *AdaBoost.R*, aunque finalmente fue Drucker el que desarrolló el algoritmo en el que se basa llamado *AdaBoost.R2* [73].

El modelo de predicción *AdaBoost* está basado en un algoritmo de aprendizaje débil que evoluciona a uno fuerte. Para ello utiliza un método aditivo de etapas hacia adelante que hace que el estimador sea ajustado a partir de una serie de estimadores débiles. Con estas iteraciones lo que se consigue es minimizar una función de pérdida, en base a que los estimadores que se ajustan en el próximo modelo son entrenados por el modelo anterior.

Sin embargo, *AdaBoost.R2* lo que hace es cambiar el peso de los datos del entrenamiento del algoritmo [74]. Es decir, cada iteración que realiza lleva consigo un error asociado a ese dato, el error es calculado y se le asocia un peso. El peso del dato que haya sido mal clasificado aumenta, por el contrario, el peso del dato que haya sido correctamente clasificado disminuye. De esta manera se consigue que aumenten las probabilidades de que, la instancia mal clasificada por la iteración actual, sea clasificada correctamente en la próxima iteración [75]. El peso de cada iteración es calculado a partir de una función de pérdida, que puede ser lineal, cuadrada o exponencial [73]. El resultado final es calculado mediante la mediana ponderada de todos resultados. De esta manera cuanto menor sea el error de los resultados más peso tendrán en el cálculo [74].

Gradient Boosting

Este modelo de predicción está basado en árboles de decisión, llamados árboles de aumento de gradiente. Este tipo de árboles presentan poca profundidad, que hace que las predicciones sean mucho más rápidas y se les conoce como aprendices débiles. El modelo de aumento de gradiente está definido por la conexión de numerosos árboles simples, de modo que cuantos más árboles haya más robusto será. De la misma manera

que en el algoritmo *AdaBoost* se produce un proceso aditivo en el que los predictores son agregados para minimizar el error anterior. La diferencia respecto a *AdaBoost*, que da ponderaciones dependiendo de si la clasificación es correcta o no, es que, en este caso el predictor aprende de los residuos de la iteración anterior. Los resultados de cada árbol de regresión sencillos serán ponderados para calcular el valor final [18].

Aprendizaje no supervisado

El aprendizaje no supervisado, a diferencia del anterior modelo, toma como única referencia los datos de entrada, sin observar los datos de salida, así pues, no tiene en cuenta la clasificación previa de los datos para llevar a cabo el entrenamiento [56]. El aprendizaje no supervisado se puede dividir en dos tipos de problemas. El primero es la asociación, que consiste en agrupar datos que contengan características parecidas. Uno de los algoritmos que se encarga de llevar a cabo este tipo de problemas es el K-Medias (K-Means). El segundo problema es la reducción, que consiste en disminuir el número de variables de entrada mediante la exploración de características repetitivas [10].

Aprendizaje por refuerzo

Existe, además, otro tipo de aprendizaje llamado aprendizaje por refuerzo. Es un modelo basado en la experiencia, en el que se establece un sistema de acción-recompensa. La recompensa está dada por el ambiente y las acciones tomadas y mide el resultado de dichas acciones que tienen sobre el objetivo final. Esto quiere decir que el algoritmo lo que busca es obtener la máxima recompensa en función de las acciones que realiza, teniendo presente el entorno donde se encuentra. Al estar basado en la experiencia no se necesita estar supervisado y aprende por sí mismo, a través de las recompensas, si las decisiones que toma están bien o no [10]. Este tipo de aprendizaje es utilizado para enseñar a los robots a llevar a cabo distintas tareas [56].

2.3.6 Deep learning y las redes neuronales

El Deep Learning o aprendizaje profundo es un subcampo del ML diseñado siguiendo una estructura de redes neuronales similar a un modelo biológico [1]. Proviene de las redes neuronales artificiales (RNA) [76]. Las RNA están basadas en el procesamiento de la información en los sistemas biológicos, es decir en cómo se comporta el cerebro humano. El algoritmo está formado por unidades de procesamiento que pueden ser aproximadas a funciones matemáticas. Estas unidades se llaman neuronas artificiales y se organizan formando capas.

- Capa de entrada: en primer lugar, se encuentra la capa de entrada que la que se encarga de recibir los datos iniciales.
- Capas ocultas: a continuación de la capa de entrada se encuentran las capas ocultas. Están formadas por cero o más capas de neuronas conectadas entre sí. Son las encargadas del procesamiento de la información que relacionan de manera no lineal las neuronas de la capa de entrada con las neuronas de la capa de salida.
- Capa de salida: Después del procesamiento de la información, se proporciona una salida.

En las RNA la fuerza de la conexión entre las neuronas, las cuales realizan operaciones internas y se encargan de devolver una salida, está controlada por un peso que es ajustado durante el aprendizaje. Que

la información sea transmitida de una neurona a otra depende de si se excede o no un umbral impuesto por una función de activación [38], que es la encargada de devolver una salida en cada proceso.

Las RNA simples y los algoritmos descritos de ML en la sección 2.3.5 pertenecen al aprendizaje automático superficial ya que son entrenados por un algoritmo de retropropagación que no puede ser utilizado en el entrenamiento de redes profundas [77]. Este método se encarga de asignar los pesos óptimos a las conexiones de la red neuronal para relacionar de la mejor manera posible la entrada con la salida [76].

Las redes neuronales profundas son las redes pertenecientes al espacio de DL. Están formadas por más de una capa oculta, constituidas por neuronas más complejas que las RNA simples. Tener mayor complejidad neuronal da la posibilidad de emplear operaciones avanzadas o en lugar de usar una única función de activación utilizar activaciones múltiples. Estas propiedades hacen de las redes neuronales profundas la solución perfecta para analizar datos de entrada sin procesar y averiguar automáticamente la representación necesaria para el aprendizaje.

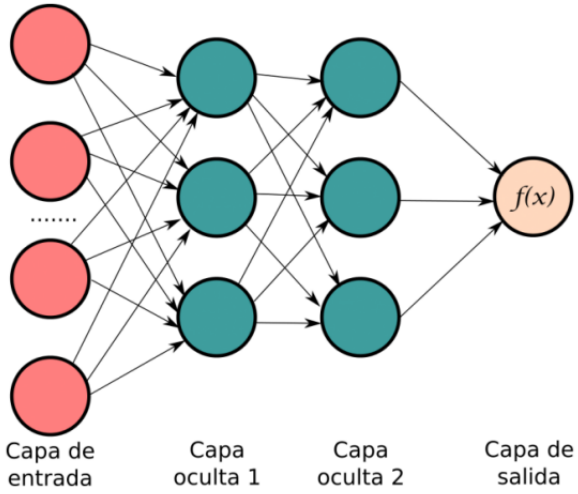

Figura 2.10: Representación esquemática de una red neuronal [10]

La ventaja principal de emplear algoritmos de DL frente a los de ML es que es muy útil cuando se analizan grandes cantidades de datos y de alta dimensión [38]. En cambio, los algoritmos de DL presentan el problema de ‘’caja negra”. Esto quiere decir que al modelo se le suministra entradas y genera una salida, pero se desconoce lo que sucede en el proceso intermedio y las características identificadas. Esto hace que los resultados sean más difíciles de interpretar que los modelos de ML [78].

2.3.7 Implementación de modelos de Machine learning y Deep Learning

En este apartado, se presentan los pasos del proceso a seguir para el desarrollo de los modelos de predicción de ML. Hay que tener en cuenta que no solo es importante la elección de un modelo, entrenarlo y obtener resultados, sino que es conveniente seguir unos pasos para poder obtener la mejor predicción posible. El procedimiento de creación de un modelo sigue una serie de etapas: lo primero de todo es entender el problema, y después se sigue con la recolección de los datos, el preprocesamiento de estos, definir un modelo adecuado que se ajuste de manera óptima a los datos, entrenamiento del modelo y por último la verificación o validación de este [10].

Entendimiento del problema

La primera etapa para la implementación de un modelo de predicción es tener conciencia del tipo de problema al que se enfrenta. Es muy importante saber y entender en que consiste la cuestión que se presenta, ya que las decisiones que se tomarán en base a esto, podrán afectar al resultado y dependerán de ello. En esta etapa se define que tipo de aprendizaje que se llevará a cabo, ya pueda ser supervisado, no supervisado o de refuerzo. También se elegirá el modelo que se va a utilizar como puede ser un modelo de regresión, clasificación, etc. [10].

Recolección de los datos

Es la tarea que consiste en la búsqueda de los datos que van a ser necesarios para realizar el modelo. En esta etapa se establece el número y tipo de datos, así como de dónde provienen. Los datos se pueden buscar en páginas webs, bases de datos o incluso se pueden comprar a organismos externos en caso de que no sean accesibles [10], [79].

Tratamiento de los datos

Esta etapa consiste en realizar una depuración de los datos, es decir, se tiene que establecer un formato adecuado para que puedan ser utilizados. También es preciso identificar valores nulos y puntos atípicos, que puedan influir perjudicialmente en el proceso de cálculo. Además, es importante detectar las características de los datos y las variables más importantes y que más pueden influir para realizar la predicción [79].

Elección del modelo

Esta parte del proceso consiste en implementar y decidir el modelo que mejor se podría ajustar a la cuestión planteada [10]. Los algoritmos que se utilizan están presentes en librerías y se implementan a través del lenguaje de programación Python.

Entrenamiento del modelo

El conjunto de datos ya procesados es dividido en dos subgrupos. Uno de ellos es el subgrupo de datos de entrenamiento, que es utilizado para entrenar al algoritmo, y está formado tanto por los datos de las variables de entrada como por los datos de las variables de salida. De esta manera es posible entrenar al algoritmo, el cual intenta obtener toda la información posible para encontrar los patrones que mejor se ajusten y poder realizar las predicciones [39].

Cada algoritmo de predicción está formado por una serie de hiperparámetros que son modificables. En esta etapa es importante la elección de estos parámetros, ya que dependiendo de los valores que se elijan de cada uno el error de la predicción será distinto en cada caso. Es conveniente encontrar cuáles son los valores de los hiperparámetros que mejor explican el modelo para tener un resultado óptimo.

Validación del modelo

En esta última etapa de la implementación del modelo se utiliza el otro subconjunto de datos, llamado datos de prueba. A su vez, se dividen en otros dos grupos. Un grupo corresponde a los datos de las variables de entrada que se introducen en el modelo para predecir la salida. La evaluación de la precisión

del modelo es realizada midiendo los errores que se cometen entre el valor predicho y el otro grupo de datos de prueba, formado por los datos reales de las variables de salida. Por tanto, se elegirá el modelo que presente menor error en la salida. Si no se está conforme con el resultado es posible volver al paso de entrenamiento del modelo y reajustar los parámetros del algoritmo para reducir el porcentaje de error [39], [79].

Capítulo 3

Desarrollo

3.1 Introducción

En este capítulo del proyecto se redacta una de las partes más importantes que es el camino seguido hasta llegar a los resultados. Dicho resultado es poder predecir las características del acero del hormigón armado cuando es sometido a ciertos estímulos. Para ello se han seguido las partes del proceso de implementación de los modelos de ML y DL expuestas en la sección 2.3.7. El desarrollo del problema se lleva a cabo a través del lenguaje de programación Python, ya que nos da muchas facilidades a la hora de utilizar los algoritmos. Todos ellos están almacenados en distintas librerías. Cabe destacar que el algoritmo de DL es una red neuronal, desarrollada a partir de la librería Keras y el código de su implementación es un poco diferente al que se utiliza para los algoritmos de ML.

En esta parte se darán a conocer los dos tipos de enfoques que se le va a dar al problema para intentar llegar a la solución óptima.

El primero de los enfoques es un análisis multisalida, que consiste en realizar una predicción a la vez de todas las variables. Es decir, se utilizan todos los subgrupos de entrenamiento de las variables a predecir, para entrenar a la vez el algoritmo. De esta manera por cada algoritmo se consigue un único modelo de predicción.

El segundo de ellos es un análisis en el que únicamente se predice una de las salidas. Es decir, se utiliza el subconjunto de entrenamiento de una de las variables a predecir para entrenar al modelo. De esta manera, una vez finalizada la predicción para dicha variable de salida, se procederá a realizar otro modelo de predicción para la siguiente variable dependiente. Realizando este tipo de análisis se obtendrán más modelos de predicción por cada algoritmo, ya que únicamente se centrará en una de las salidas.

3.2 Entendimiento del problema

Dicho problema se centra en calcular la deformación que sufre el acero presente en elementos de hormigón armado. El comportamiento del acero en este tipo de estructuras se define como no lineal. Resolver este tipo de problemas mediante los métodos convencionales es una ardua tarea. Por ello, se han utilizado modelos de ML y DL para tratar de facilitar este tipo de cálculos e intentar predecir las soluciones utilizando un enfoque alternativo.

Como el problema plantea el cálculo la deformación del acero, que se trata de un dato cuantitativo, se va a optar por un modelo de regresión.

3.3 Recolección de los datos

Los datos que se van a utilizar en el desarrollo del modelo han sido obtenidos a partir de una base de datos en Excel. Dicha base de datos consta de veintidós columnas, de las cuales las doce primeras son las variables de entrada al modelo conocidas como las variables independientes o predictoras y las siguientes diez columnas son las variables de salida del modelo o variables dependientes. El algoritmo se va a encargar de predecir las variables dependientes en función de las variables independientes. Además, la base de datos consta de 2000 datos por columna.

Las variables predictoras van a ser las siguientes:

Nº	Variables	Símbolo	Unidades
1	Módulo elástico del acero (*Elastic modulus of the steel*)	E_s	MPa
2	Límite elástico del acero (*Steel yield stress*)	f_y	MPa
3	Diámetro de las barras longitudinales (*Diameter of the longitudinal bars*)	ϕ_x	mm
4	Diámetro de las barras transversales (*Diameter of the transverse bars*)	ϕ_t	mm
5	Deformación impuesta del acero de pretensado (*Imposed prestressing steel strain*)	$\Delta\epsilon_p$	1
6	Resistencia a la compresión del hormigón en un ensayo de cilindro (*Compressive strength of concrete in a cylinder test*)	f_{ck}	MPa
7	Ancho de banda (*Web width*)	b_w	mm
8	Distancia entre estribos (*Stirrups spacing*)	s	mm
9	Área de la sección transversal de los cordones de pretensado (*Cross-sectional area of the prestressing strands*)	A_p	mm^2
10	Parámetro longitudinal λ (*Longitudinal λ parameter*)	λ_x	1
11	Parámetro transversal λ (*Transverse λ parameter*)	λ_t	1
12	Deformación principal de tracción (*Principal tensile strain*)	ϵ_1	1

Tabla 3.1: Variables de entrada

Y las variables predichas son:

Nº	Variables	Símbolo	Unidades
1	Ángulo del esfuerzo principal de compresión (*Angle of the principal compressive stress*)	θ	rad
2	Deformación media en el acero longitudinal (*Average strain in the longitudinal steel*)	ϵ_x	1
3	Fuerza interna cortante (*Shear internal force*)	V	N
4	Deformación media en el acero transversal (*Average strain in the transverse steel*)	ϵ_t	1
5	Deformación principal de compresión (*Principal compressive strain*)	ϵ_2	1
6	Deformación media en los cordones de pretensado (*Average strain in the prestressing strands*)	ϵ_p	1
7	Tensión del acero de pretensado (*Prestressing steel stress*)	σ_p	MPa
8	Tensión principal de compresión (*Principal compressive stress*)	σ_2	MPa
9	Esfuerzo medio de tracción en el acero longitudinal (*Average tensile stress in the longitudinal steel*)	σ_{sx}	MPa
10	Esfuerzo medio de tracción en el acero transversal (*Average tensile stress in the transverse steel*)	σ_{st}	MPa

Tabla 3.2: Variables de salida

3.4 Tratamiento de los datos

Antes de comenzar un modelo de predicción, una tarea muy importante es realizar un análisis exploratorio de los datos. De esta manera se va a entender de una manera más clara la información de cada variable y va a permitir detectar, en caso de que los hubiera, errores.

Esta etapa es el tratamiento de los datos. Para ello se ha llevado a cabo un análisis del archivo Excel dónde se encuentran.

Esto pasa por averiguar si todos los datos incluidos están almacenados correctamente y en un formato adecuado al tipo de análisis que se va a realizar. También es imprescindible detectar si alguna variable contiene valores nulos. Todo esto va a servir como base para interpretar que variables son las que mayor influencia pueden tener en un modelo predictivo.

Por tanto, los pasos que se han llevado en este apartado son los siguientes:

- Realizar un análisis de las variables, atendiendo al formato y la cantidad de datos presentes.
- Identificar, si existen, valores nulos o ausentes.
- Analizar la varianza de las variables.
- Analizar las correlaciones existentes entre cada una de las variables.
- Escalado de los datos.

Con la ayuda del comando de Python, info(), se ha podido visualizar la información presente en cada una de las variables, mostrando su nombre, el número de datos presentes no nulos y el formato de la variable:

Nº	Column	Non-Null Count	Dtype
1	Módulo elástico del acero (*Elastic modulus of the steel*)	2000 non-null	float64
2	Límite elástico del acero (*Steel yield stress*)	2000 non-null	float64
3	Diámetro de las barras longitudinales (*Diameter of the longitudinal bars*)	2000 non-null	float64
4	Diámetro de las barras transversales (*Diameter of the transverse bars*)	2000 non-null	float64
5	Deformación impuesta del acero de pretensado (*Imposed prestressing steel strain*)	2000 non-null	float64
6	Resistencia a la compresión del hormigón en un ensayo de cilindro (*Compressive strength of concrete in a cylinder test*)	2000 non-null	float64
7	Ancho de banda (*Web width*)	2000 non-null	float64
8	Distancia entre estribos (*Stirrups spacing*)	2000 non-null	float64
9	Área de la sección transversal de los cordones de pretensado (*Cross-sectional area of the prestressing strands*)	2000 non-null	float64
10	Parámetro longitudinal λ (*Longitudinal λ parameter*)	2000 non-null	float64
11	Parámetro transversal λ (*Transverse λ parameter*)	2000 non-null	float64
12	Deformación principal de tracción (*Principal tensile strain*)	2000 non-null	float64
13	Ángulo del esfuerzo principal de compresión (*Angle of the principal compressive stress*)	2000 non-null	float64
14	Deformación media en el acero longitudinal (*Average strain in the longitudinal steel*)	2000 non-null	float64
15	Fuerza interna cortante (*Shear internal force*)	2000 non-null	float64
16	Deformación media en el acero transversal (*Average strain in the transverse steel*)	2000 non-null	float64
17	Deformación principal de compresión (*Principal compressive strain*)	2000 non-null	float64
18	Deformación media en los cordones de pretensado (*Average strain in the prestressing strands*)	2000 non-null	float64
19	Tensión del acero de pretensado (*Prestressing steel stress*)	2000 non-null	float64
20	Tensión principal de compresión (*Principal compressive stress*)	2000 non-null	float64
21	Esfuerzo medio de tracción en el acero longitudinal (*Average tensile stress in the longitudinal steel*)	2000 non-null	float64
22	Esfuerzo medio de tracción en el acero transversal (*Average tensile stress in the transverse steel*)	2000 non-null	float64

Tabla 3.3: Información de las variables

En la tabla 3.3 se puede observar, todas las variables presentan 2000 valores no nulos, esto no quiere decir que la variable no pueda contener valores nulos. Además, el formato de los valores es de carácter numérico decimal, es por ello por lo que se ha elegido un problema de regresión.

Como se ha mencionado, uno de los requisitos es eliminar los datos ausentes, o bien las variables que contienen dichos datos. Esto supone que se pierda información a la hora de realizar el modelo y en caso de que tengamos pocos valores puede ser un proceso bastante perjudicial. También se pueden estimar estos valores con la ayuda del resto de información que se dispone. Hay que tener en cuenta sobre qué variables se están eliminando o introduciendo valores, ya que pueden ser magnitudes que tengan un peso significativo sobre el modelo.

Para conocer de verdad si las variables presentan valores faltantes se hace uso del comando insa().sum().sort_values() que va a sacar como resultado una tabla en la que se indicaran el número de datos nulos.

Nº	Variable	Cantidad de valores nulos
1	Módulo elástico del acero (*Elastic modulus of the steel*)	0
2	Límite elástico del acero (*Steel yield stress*)	0
3	Diámetro de las barras longitudinales (*Diameter of the longitudinal bars*)	0
4	Diámetro de las barras transversales (*Diameter of the transverse bars*)	0
5	Deformación impuesta del acero de pretensado (*Imposed prestressing steel strain*)	0
6	Resistencia a la compresión del hormigón en un ensayo de cilindro (*Compressive strength of concrete in a cylinder test*)	0
7	Ancho de banda (*Web width*)	0
8	Distancia entre estribos (*Stirrups spacing*)	0
9	Área de la sección transversal de los cordones de pretensado (*Cross-sectional area of the prestressing strands*)	0
10	Parámetro longitudinal λ (*Longitudinal λ parameter*)	0
11	Parámetro transversal λ (*Transverse λ parameter*)	0
12	Deformación principal de tracción (*Principal tensile strain*)	0
13	Ángulo del esfuerzo principal de compresión (*Angle of the principal compressive stress*)	0
14	Deformación media en el acero longitudinal (*Average strain in the longitudinal steel*)	0
15	Fuerza interna cortante (*Shear internal force*)	0
16	Deformación media en el acero transversal (*Average strain in the transverse steel*)	0
17	Deformación principal de compresión (*Principal compressive strain*)	0
18	Deformación media en los cordones de pretensado (*Average strain in the prestressing strands*)	0
19	Tensión del acero de pretensado (*Prestressing steel stress*)	0
20	Tensión principal de compresión (*Principal compressive stress*)	0
21	Esfuerzo medio de tracción en el acero longitudinal (*Average tensile stress in the longitudinal steel*)	0
22	Esfuerzo medio de tracción en el acero transversal (*Average tensile stress in the transverse steel*)	0

Tabla 3.4: Número de valores nulos encontrados en las variables

La tabla 3.4 muestra el número de valores que hay nulos, en este caso se observa que hay cero valores nulos para todas las variables.

Otro de los requisitos es prestar especial atención a las variables las cuales su varianza sea nula o cercana a cero, esto quiere decir que, los datos almacenados en estas magnitudes no varían o si lo hacen es con muy poca frecuencia y no aportan gran información al modelo. Este proceso es conveniente realizarlo entes de la estandarización de los datos.

El análisis de la varianza se puede ver de forma gráfica realizando histogramas de frecuencias de las distintas variables. En este tipo de gráficos se aprecia la distribución que toman los datos. En el eje de abscisas (eje X) se representa el rango de datos que ocupa la variable que se analiza, mientras que en el eje de ordenadas (eje Y) se representan las frecuencias de los datos, es decir, la altura de las barras depende de cuántas veces aparezca ese dato en la variable.

A continuación, se muestran los histogramas pertenecientes a las variables de entrada.

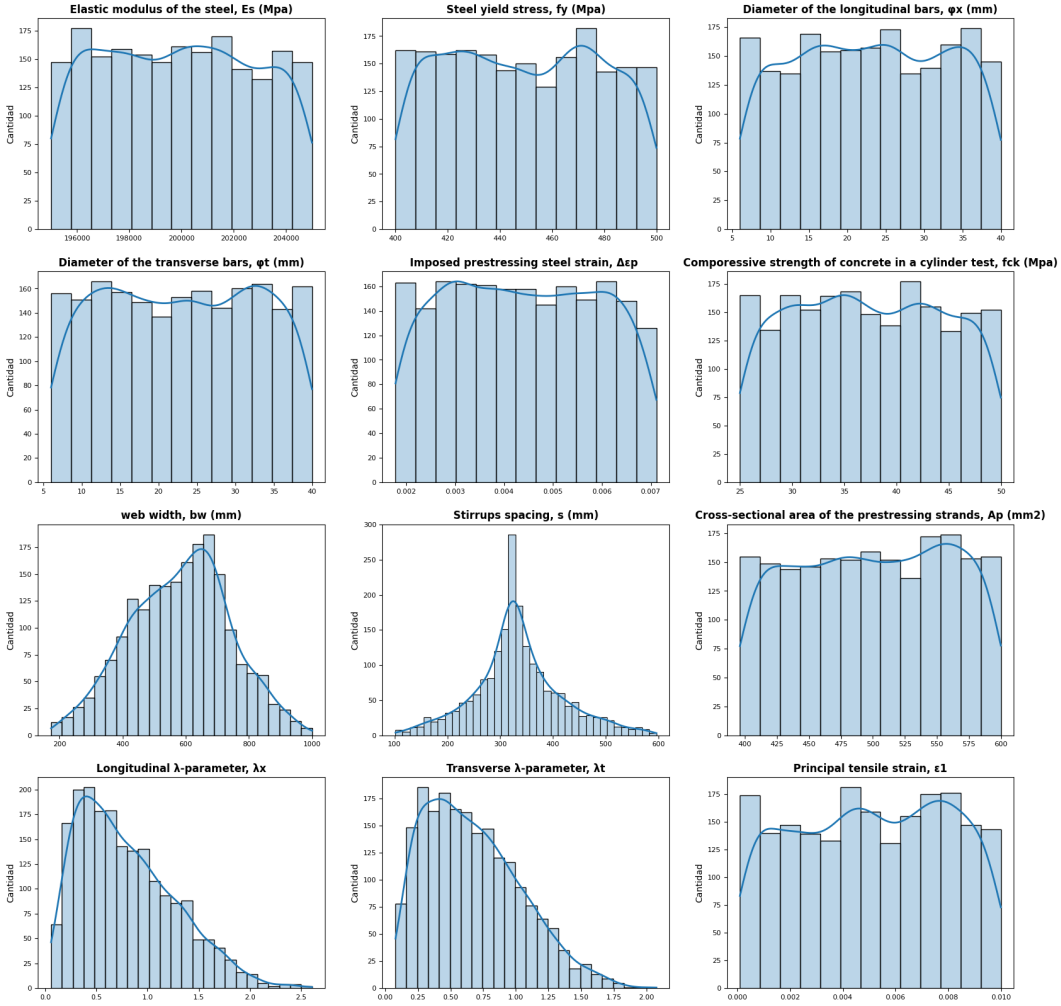

Figura 3.1: Histogramas de las variables de entrada

Los datos de las variables de entrada tienen bastante variabilidad. Se puede observar en la figura 3.1 que las variables ancho de banda, b_w (*Web width*) y distancia entre estribos, s (*Stirrups spacing*) presentan una distribución prácticamente simétrica de los datos, mientras que las variables parámetro longitudinal λ, λ_x (*Longitudinal* λ *parameter*) y parámetro transversal λ, λ_t (*Transverse* λ *parameter*) presentan una distribución más asimétrica de los datos.

A continuación, se muestran los histogramas pertenecientes a las variables de salida.

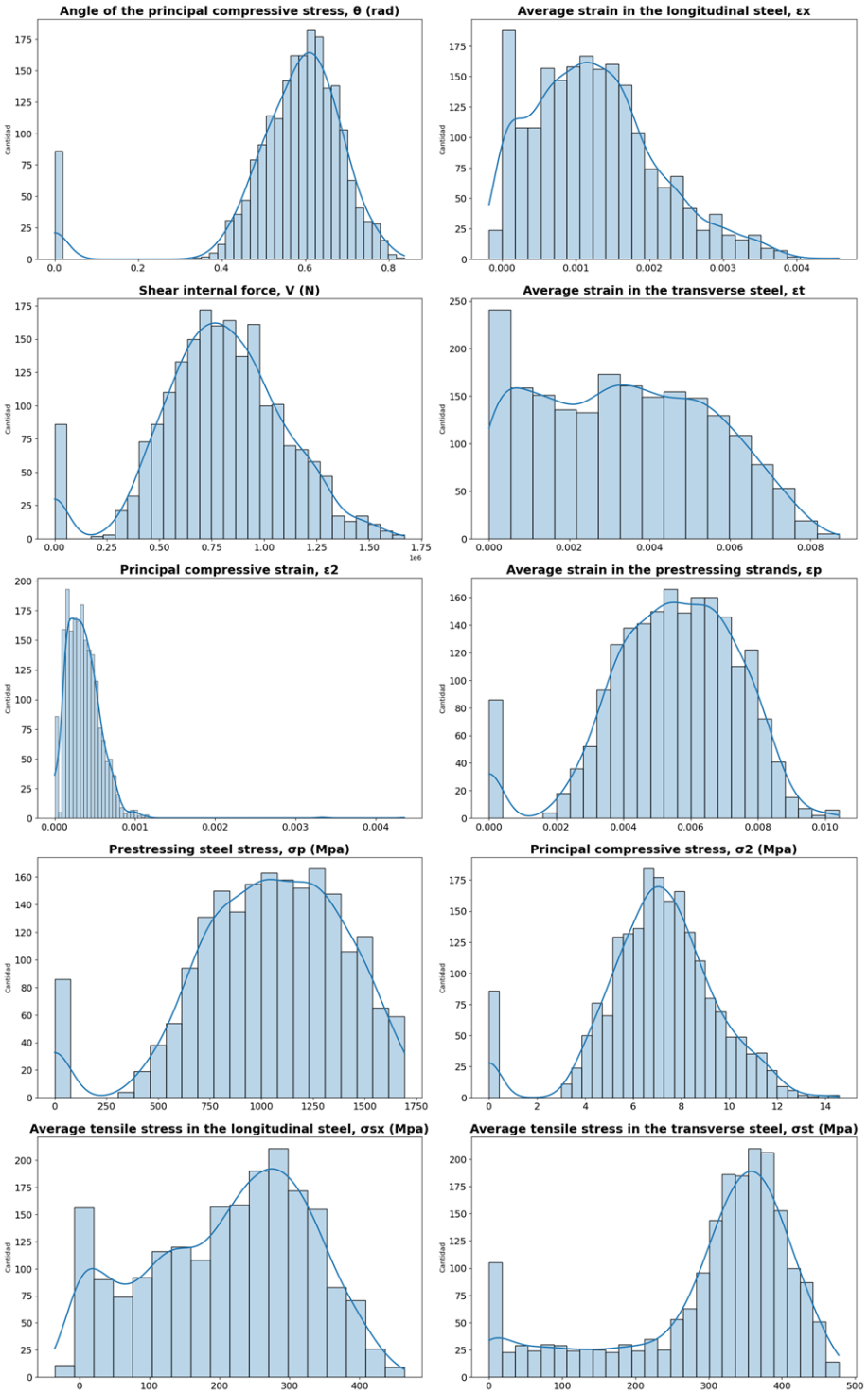

Figura 3.2: Histogramas de las variables de salida

Las variables de salida, en cambio, muchas de ellas presentan forma de campana de Gauss. También se puede observar en la figura 3.2 que bastantes de las variables contienen puntos atípicos que pueden influir en los modelos de predicción.

Otro gráfico importante donde también se puede observar la variabilidad de los datos y mostrar de mejor manera los puntos atípicos son los diagramas de cajas y bigotes. En este caso en el eje de ordenadas (eje Y) se observa el rango de datos de la variable. El diagrama se representa mediante un rectángulo en el que el borde inferior hace referencia al primer cuartil (C_1), mientras que el borde superior es el tercer

cuartil (C_3). De esta manera, dentro del rectángulo se encuentran el 50 % de los datos. Dentro de la caja se encuentra la mediana, que está representada por una línea horizontal. A los lados del rectángulo salen dos líneas horizontales que hacen referencia a los bigotes. Los límites de los bigotes están determinados por las siguientes expresiones:

- El límite inferior del bigote representa el mínimo valor establecido que se considera normal en la distribución y se calcula mediante la fórmula:

$$L_i = C_1 - 1{,}5RI \tag{3.1}$$

  En dónde RI hace referencia al rango intercuartílico, que se trata de una medida de dispersión calculada como la diferencia o distancia entre el tercer y primer cuartil:

$$RI = C_3 - C_1 \tag{3.2}$$

- El límite superior, por el contrario, representa el máximo valor establecido que se considera normal en la distribución y es calculado como:

$$L_s = C_3 + 1{,}5RI \tag{3.3}$$

Los datos que se encuentren dentro del rango entre los límites superior e inferior se considerarán datos normales. En cambio, si los datos se encuentran por debajo del límite inferior o por encima del límite superior son llamados valores atípicos y se encuentran representados en el diagrama mediante puntos aislados. [80]

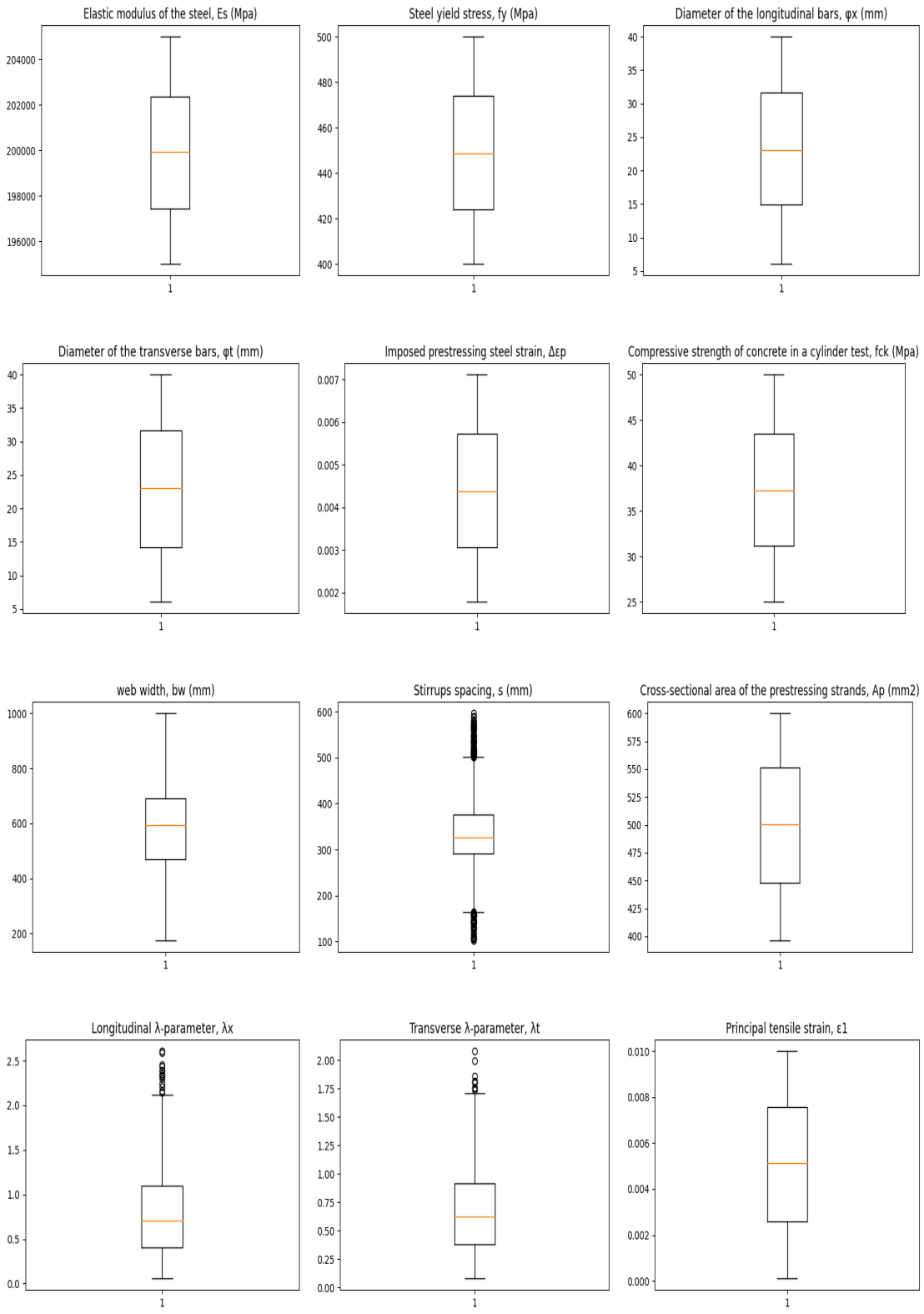

Figura 3.3: Diagramas de cajas y bigotes de las variables de entrada

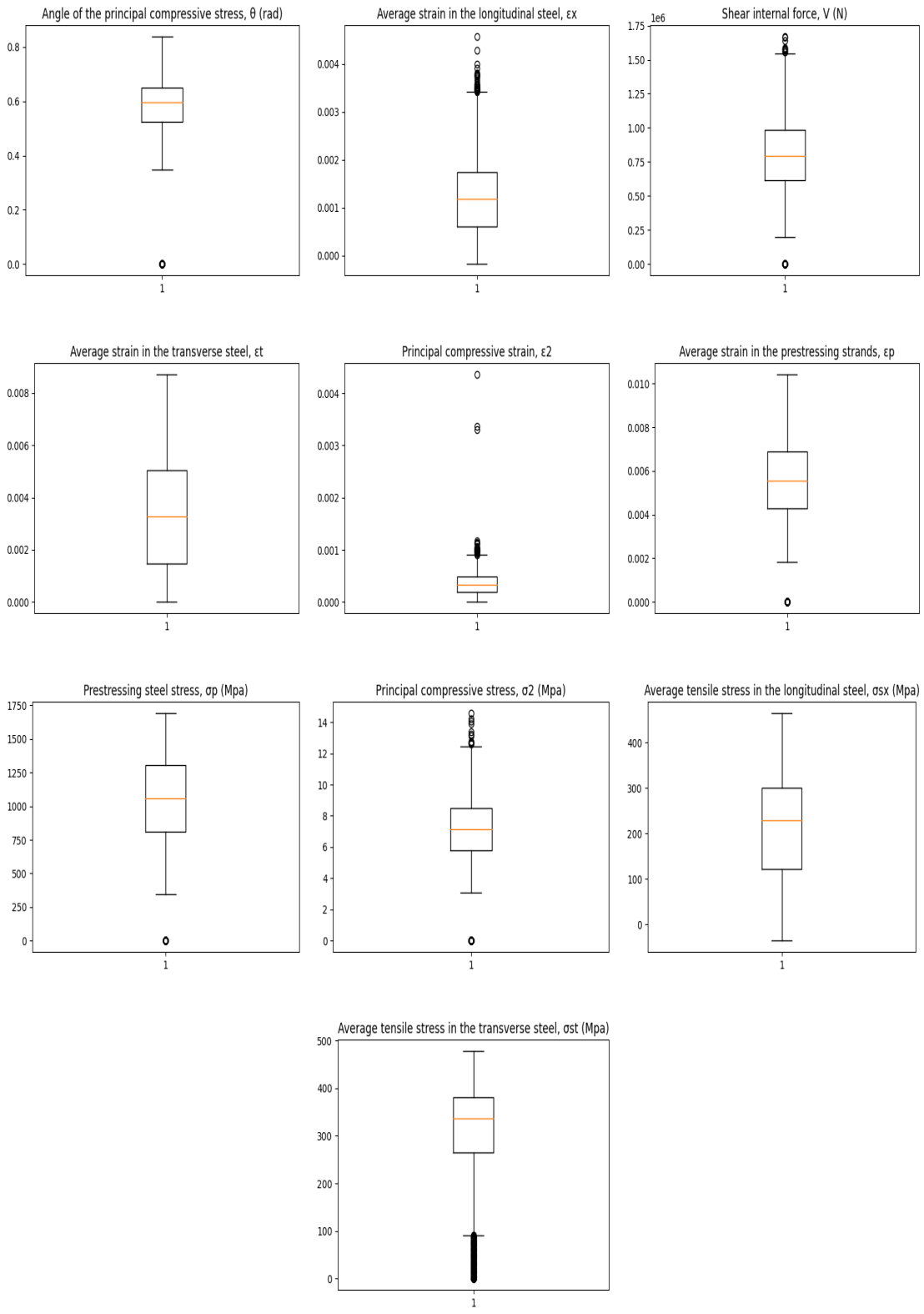

Figura 3.4: Diagramas de cajas y bigotes de las variables de salida

En las figuras 3.3 y 3.4 se representan de una manera más visual los puntos atípicos de las distintas variables. Estos datos influirán en los modelos de predicción, pero tienen que ser tomados en cuenta ya que forman parte de los datos de las variables.

A continuación, se representa la matriz de correlaciones que ayudará a tener una idea global de qué variables podrían tener un mayor peso sobre otras en el modelo de predicción.

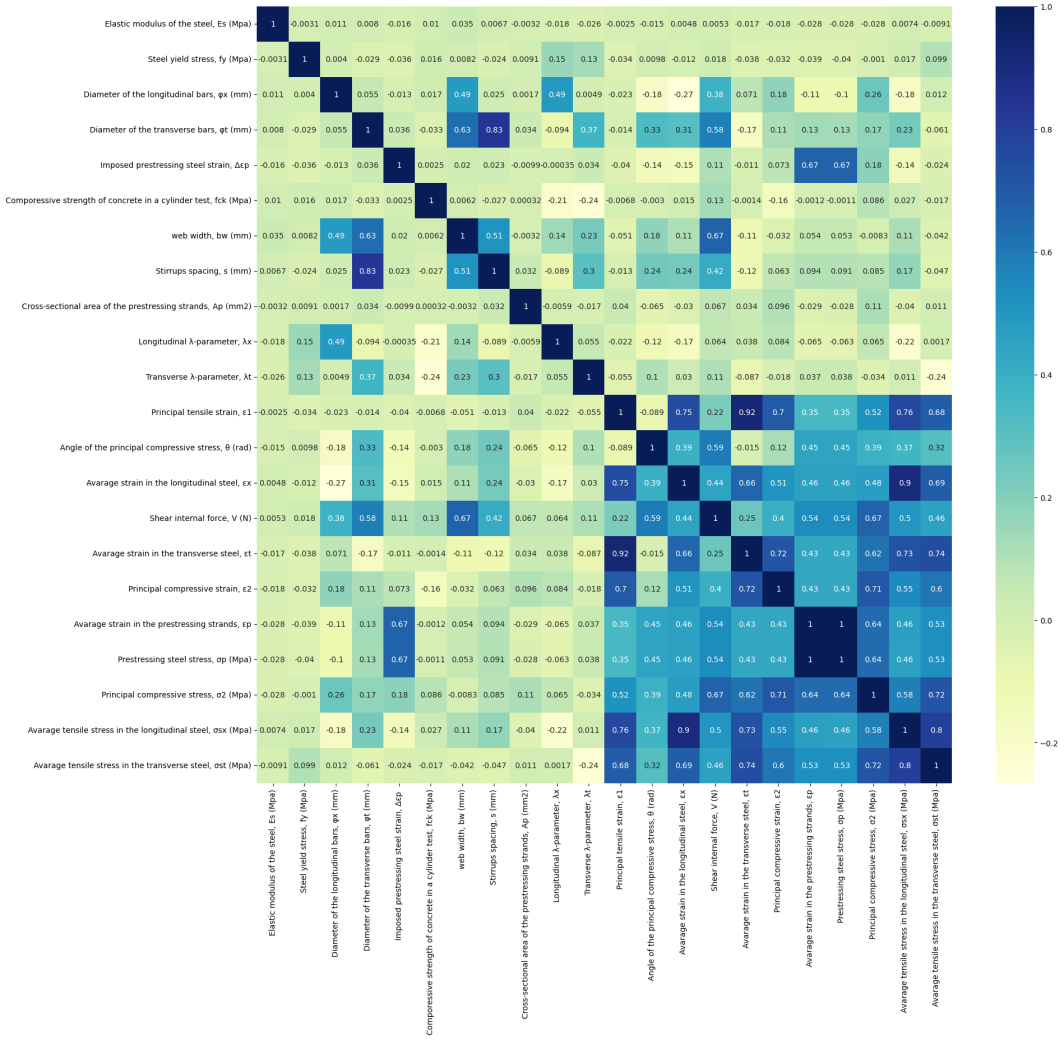

Figura 3.5: Matriz de correlaciones

La figura 3.5 muestra la matriz de correlaciones determinada a partir de los coeficientes de Pearson. Lo que pretende mostrar la matriz es la dependencia lineal que existe entre cada una de las variables. Los coeficientes están calculados a partir de dos variables, en los que la covarianza de cada una es dividida por el producto de sus variables estándar, esto hace que actúe como un proceso de normalización. De esta manera los coeficientes finales se encontrarán todos en un rango entre -1 y 1. El coeficiente está calculado en base a la siguiente fórmula:

$$r_{xy} = \frac{\sum(x_i - \bar{x})\sum(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2}\sqrt{\sum(y_i - \bar{y})^2}} \tag{3.4}$$

Donde x_i e y_i son puntos individuales de la muestra,

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{N} x_i \tag{3.5}$$

$$\bar{y} = \frac{1}{n}\sum_{i=1}^{N} y_i \tag{3.6}$$

hacen referencia a la media muestral de x e y. Como se ha mencionado el rango de valores del coeficiente de Pearson está entre -1 y 1. El signo del coeficiente será positivo si la relación que mantienen ambas variables es directamente proporcional, mientras que si el signo del coeficiente es negativo quiere decir que las variables son inversamente proporcionales. Si el valor del coeficiente de Pearson es igual a cero quiere decir que las variables no tienen relación alguna y cuanto más cercano esté de la unidad más fuerte será la relación lineal [81].

Gracias a la matriz de correlaciones se puede realizar un primer análisis acerca de que variables van a tener una influencia superior sobre otras en los modelos predictivos. La variable predictora deformación principal de tracción ϵ_1 (*Principal tensile strain*) está altamente correlacionada con la deformación media en el acero transversal ϵ_t (*Average strain in the transverse steel*). Además, también presenta un grado importante de correlación con las variables deformación media en el acero longitudinal ϵ_x (*Average strain in the longitudinal steel*), deformación principal de compresión ϵ_2 (*Principal compressive strain*) y esfuerzo medio de tracción en el acero longitudinal σ_{sx} (*Average tensile stress in the longitudinal steel*).

Para que los datos puedan ser interpretados por el algoritmo es necesario establecer un mismo formato que se lleva a cabo mediante la transformación de las magnitudes. La transformación de los datos se lleva a cabo después de dividirlos en los subconjuntos de entrenamiento y de prueba. Para este proyecto se van a dividir los datos siguiendo una proporción del 75 % dedicada a los datos de entrenamiento y el 25 % restante dedicada a los datos de prueba, encargados de la evaluación del modelo.

En este caso como se ha podido observar, la base de datos no dispone de valores ausentes ni de variables con varianza cercana a cero, por lo tanto, facilita en gran medida el preprocesamiento de los datos.

La estandarización de los datos es un proceso muy importante que se realiza para establecer un rango en el que se encuentran todos los datos. Ayuda a que el proceso de cálculo del algoritmo se realice más rápido y a que tengan un formato adecuado para el ajuste del modelo [82].

La estandarización en este caso se ha llevado a cabo utilizando la función de Python StandardScaler() presente en la biblioteca Sklearn. Esta función transforma los datos haciendo que la desviación estándar sea uno y el valor medio de la distribución resultante de la estandarización sea cero. La fórmula matemática empleada en el proceso de estandarización StandardScaler() es la siguiente:

$$z = \frac{x - \mu}{\sigma} \tag{3.7}$$

En donde z es el valor resultante del proceso de transformación, x es el dato original al que se le resta μ, que es la media y es dividido entre la desviación estándar, σ [82].

El proceso de estandarización se ha realizado de manera distinta. En el análisis multisalida la estandarización se produce por partes, es decir, por un lado, se transforman los datos de las variables de entrada y, por otro lado, los datos de las variables de salida. Ambos escaladores se entrenan únicamente con los datos de los subconjuntos de entrenamiento. Así pues, utilizando las características de los datos de entrenamiento de las variables de entrada, se transforman el propio subconjunto de entrenamiento y los datos de prueba. De la misma manera, utilizando las características de los datos de entrenamiento de las variables de salida son transformados dichos datos y los que pertenecen al subconjunto de prueba. Esto se realiza así debido a que al modelo entran múltiples datos de entrada que pueden presentar escalas muy dispares entre sí. De la misma manera ocurre en la salida, pues son distintas variables las que se intentan predecir, las cuales contienen datos que pueden presentar un rango de valores distinto.

Al contrario sucede cuando se intenta predecir una única salida cuya magnitud de los datos no es tan importante. En este caso para entrenar el escalador sólo se tiene en cuenta los datos de las variables de

entrada del subconjunto de entrenamiento, ya que son múltiples datos los que se encargan de entrenar al modelo. Con el resultado de la media del subconjunto entrenamiento se estandarizan los datos de las variables de entrada del subconjunto de prueba, que son los que se van a utilizar para evaluar el modelo.

En el caso del modelo de red neuronal se ha optado por normalizar los datos de entrada. Para realizar el proceso de normalización se ha empleado una función de Python llamada MinMaxScaler(), presente en la librería Sklearn.

Este tipo de normalización escala los datos para que se encuentren en un rango de entre 0 y 1. Para ello la función de Python está basada en la siguiente operación:

$$z = \frac{X - X_{mín}}{X_{máx} - X_{mín}} \tag{3.8}$$

Donde X es el dato original que se pretende normalizar, mientras que $X_{mín}$ y $X_{máx}$, hacen referencia a los valores mínimos y máximos de la variable que se está normalizando. Por su parte, z es el dato resultante del proceso de escalado [82].

Una de las ventajas que tiene utilizar un algoritmo de DL frente a uno de ML es que, la red neuronal puede trabajar con datos de entrada de alta dimensión que no necesitan ser procesados. Su compleja arquitectura hace posible que se extraigan las características de las variables con mucha facilidad, esto hace que pueda operar con datos no estructurados.

3.5 Elección del modelo

Después del proceso de tratamiento de los datos, la siguiente parte de la implementación de un modelo es la elección del algoritmo que se va a emplear. Como ya se ha mencionado se van a utilizar los algoritmos de regresión lineal, árboles de decisión, Lasso, *K-Nearest Neighbors* (KNN), *Support Vector Machine* (SVR), *Random Forest*, *AdaBoost* y *Gradient Boosting*. Estos modelos están presentes en librerías de Python.

El algoritmo de DL que se va a utilizar es una red neuronal que se va a programar desde un entorno virtual Python. Para la programación del algoritmo se va a hacer uso de librería Keras, que permite programar de una manera sencilla una red neuronal profunda. De esta manera se permite seleccionar el número de neuronas que tiene cada capa de la red, así como el número de capas y la dimensión de la entrada, es decir, como se tienen doce variables de entrada pues la primera capa tendrá dimensión doce [38].

3.6 Entrenamiento del modelo

El entrenamiento de estos algoritmos se realiza tanto con los datos de entrenamiento de las variables de entrada como con los de las variables de salida. En el caso del análisis multisalida se utilizan ambos subgrupos estandarizados, mientras que cuando se analiza una única salida el subgrupo de entrenamiento de las variables de entrada es el que se encuentra estandarizado.

3.7 Validación del modelo

Una vez entrenado el modelo, se sigue con la etapa de validación. El algoritmo elegido es evaluado utilizando el subconjunto de datos de prueba pertenecientes a las variables de entrada ya estandarizados.

Este proceso lo que hace es predecir los valores en la salida que a continuación son comparados con los datos reales correspondientes a los ejemplos empleados en la validación.

Esta comparación es utilizada para establecer si un modelo es válido o no. Para ello se ha hecho uso de distintas métricas ampliamente utilizadas en la evaluación de modelos como son el error absoluto medio (MAE, del inglés *mean absolute error*), el error cuadrático medio (RMSE, del inglés *root mean square error*), el error cuadrático medio normalizado (NMSE, del inglés normalized mean square error) y el coeficiente de determinación R cuadrado (R^2).

El MAE es un tipo de error que es calculado como el promedio de la diferencia de las distancias entre los puntos observados y los predichos en valor absoluto. Para medir el error MAE se utiliza la siguiente expresión:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|x_i - \hat{x_i}| = \frac{1}{n}\sum_{i=1}^{n}|e_i| \tag{3.9}$$

Donde, x_i es el conjunto de valores observados, x^i son los valores predichos y n es el conjunto de datos, es decir, el tamaño de la muestra. De esta manera se entiende que $e_i = x_i - \hat{x_i}$ son los llamados residuos [83].

El RMSE en cambio, estima el error calculando la raíz cuadrada de la diferencia media entre los valores observados y los valores predichos al cuadrado, es decir, la media de los cuadrados de los residuos. La expresión que se utiliza para el cálculo del RMSE es:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \hat{x_i})^2}{n}} = \sqrt{\frac{\sum_{i=1}^{n}(e_i)^2}{n}} \tag{3.10}$$

La diferencia entre ambos errores es que el RMSE tiene mayor sensibilidad ante los valores atípicos, ya que la distancia entre el dato real y el predicho está elevada al cuadrado [84].

Cuando se tienen distintos modelos, como en este caso, en los que es necesario comparar errores es conveniente que todos tengan una misma escala. Para ello es muy útil emplear el tipo de error NMSE que consiste la normalización del error MSE. El MSE calcula la media de los cuadrados de los residuos, es decir, es el error RMSE sin aplicar la raíz cuadrada. Para normalizar el MSE, es dividido entre la varianza de los datos observados reales en la salida. De esta manera todos los resultados obtenidos del error estarán entre el rango [0,1]. Esto quiere decir que cuanto menor sea el resultado del NMSE menor será el error entre el valor observado y el predicho [85].

$$NMSE = \frac{MSE}{\sigma_0^2} \tag{3.11}$$

Por último, se ha utilizado también como forma de evaluación del modelo el coeficiente de determinación R^2. Este parámetro determina la bondad de un modelo de regresión, es decir, representa cuanto de bien se ajusta. La expresión que conforma el coeficiente es la siguiente:

$$R^2 = 1 - \frac{SCR}{SCT} \tag{3.12}$$

Donde SCR es la suma de cuadrados de los residuos, que es la suma de cuadrados de las distancias que corresponden a la diferencia entre el valor real y el predicho. Mientras que SCT es la suma de cuadrados total y hace referencia a la suma de cuadrados de la diferencia entre el valor real y la media de los datos de la variable dependiente [86].

A veces no es suficiente entrenar el algoritmo con los valores que vienen por defecto, ya que el resultado podría no ser tan exacto como se esperaba. El algoritmo presenta una serie de variables que pueden ser modificadas en función de los datos y el problema que se tenga, llamadas hiperparámetros. Los

hiperparámetros pueden ser ajustados experimentalmente y con una buena selección de estos el modelo podría mejorar considerablemente. El proceso experimental para encontrar estas variables se le conoce como ajuste de hiperparámetros y es realizado mediante dos tipos de funciones:

- Grid Search: se encarga de buscar los hiperparámetros, previamente definidos por el usuario, y establecer un método de validación para comparar sus errores y seleccionar aquellos que mejor expliquen el modelo.
- Randomized Search: la diferencia que tiene con el método Grid Search es que hace una búsqueda aleatoria de los hiperparámetros, seleccionando el número de iteraciones que se desean realizar.

Cuando el método devuelve los hiperparámetros que ha seleccionado como óptimos son introducidos de nuevo en el modelo y se repite el proceso de entrenamiento y validación.

En el caso de la red neuronal se ha optado por optimizar el algoritmo manualmente y no empelar ningún tipo de función destinada a este fin. El proceso consiste en volver a dividir únicamente los datos de entrenamiento en dos subconjuntos, un subconjunto destinado de nuevo como entrenamiento y otro de validación. De esta manera los hiperparámetros serían elegidos en función de un nuevo modelo destinado exclusivamente para este proceso y eligiendo como resultado final los valores que mejor error obtuvieran. Así se reservarían los datos de test para validar el modelo principal en función del mejor resultado obtenido en la validación del segundo modelo. En resumen, de esta manera se evitaría hacer ‘’trampa” ya que los datos de test son datos desconocidos los cuales no intervienen en el proceso de entrenamiento de un modelo y no sería lógico elegir el mejor resultado utilizando dichos datos.

Dicho esto, se ha generado una tabla en la que se anotaban los errores de cada proceso de validación variando el número de capas y neuronas de cada capa. Finalmente, los datos con los que mejores resultados se han obtenido de este proceso, han sido implementados en el modelo principal.

Capítulo 4

Resultados

4.1 Introducción

Una vez ajustado de nuevo el modelo con los mejores hiperparámetros, es momento de comparar las métricas mencionadas anteriormente 3.7 entre los distintos algoritmos de predicción y de cada variable. No todos los modelos explican de la misma manera los valores que se tienen, y para obtener la predicción óptima es necesario averiguar el algoritmo que mejor se ajusta a la distribución de los datos.

El objetivo de esta parte del proyecto es realizar una comparativa de los errores obtenidos en el proceso de validación de cada modelo y establecer cuál de ellos es el que mejor explica los datos. La estructura que va a seguir la presentación de los resultados es por orden de variables dependientes, de esta manera se van a ir analizando una por una y extrayendo sus conclusiones. Se van a comparar los errores NMSE de los modelos propuestos en la sección 2.3.5, ya que es muy útil a la hora de comparar magnitudes que presentan diferentes escalas. Cuanto menor sea el valor del error NMSE mejores serán los resultados de la predicción del algoritmo.

Primeramente, se van a estudiar los resultados observados en el análisis multisalida y a continuación se estudiarán los obtenidos en el análisis de una única salida, finalizando con una comparación de ambos análisis.

4.2 Resultados experimentales

4.2.1 Análisis multisalida

El objetivo de este tipo de análisis es predecir en función de las variables independientes todas las variables dependientes a la vez. Es decir, la entrada al modelo serán todos los datos de entrenamiento de las variables predictoras y como salida se utilizará para el ajuste del modelo todos los datos de entrenamiento de las variables de salida. De esta manera se dispondrá del mismo algoritmo global ya optimizado para calcular el error de cada variable de salida, en vez de tener un modelo distinto para cada una de ellas.

Los resultados obtenidos para cada una de las variables predichas se expresan en los siguientes gráficos que comparan los errores de los distintos algoritmos utilizados para cada una de ellas.

Para el caso del análisis multisalida, no ha sido posible establecer un modelo de predicción para cada uno de los algoritmos que se disponen, debido que alguno de ellos no permite tener una salida de más de una dimensión.

Angle of the principal compressive stress

Para el caso de la variable *Angle of the principal compressive stress* el algoritmo que mejor explica la distribución de los datos es la red neuronal, prácticamente se han obtenido los mismos resultados en los algoritmos lineales a diferencia de los árboles de decisión que han presentado peores resultados.

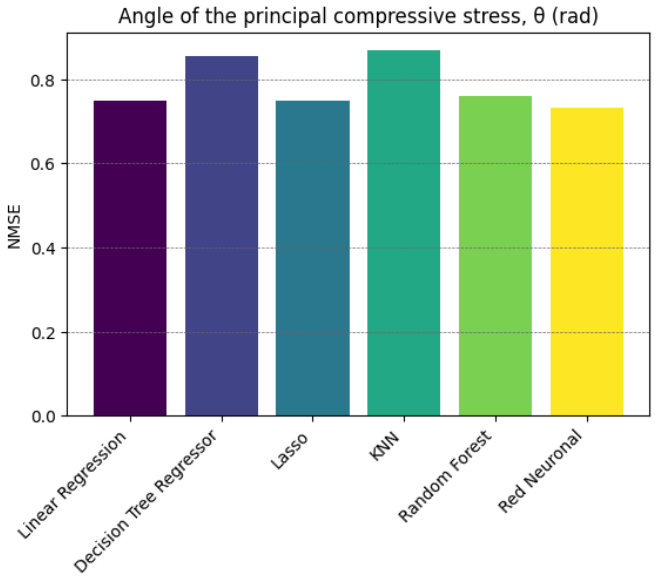

Figura 4.1: Resultados del error NMSE en la variable *Angle of the principal compressive stress* - Multisalida

A continuación, se puede observar cómo se ajusta la predicción del algoritmo de la red neuronal en un gráfico que muestra los valores reales en comparación con los valores que predice el modelo.

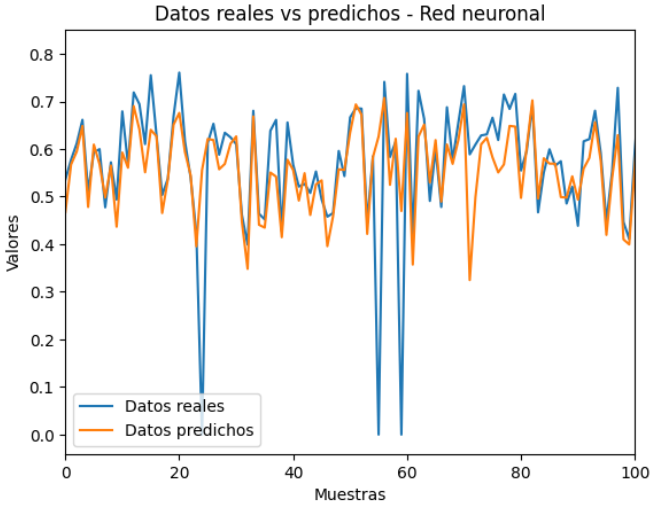

Figura 4.2: Comparación entre los valores reales y los predichos en la red neuronal de la variable *Angle of the principal compressive stress* - Multisalida

Como se puede observar en la figura 4.2 los valores predichos se ajustan en una medida aceptable a los datos reales, pero es una variable difícil de explicar. Especialmente presenta dificultad en predecir los puntos atípicos.

Average strain in the longitudinal steel

En este caso la variable *Average strain in the longitudinal steel* presenta muy buenos resultados con valores del error NMSE muy bajos en la red neuronal, como en el caso anterior. Seguido de la red neuronal el algoritmo Random Forest también presenta buenos resultados en la predicción. Sin embargo, los modelos lineales y sobre todo el KNN predice los valores con mayor imprecisión.

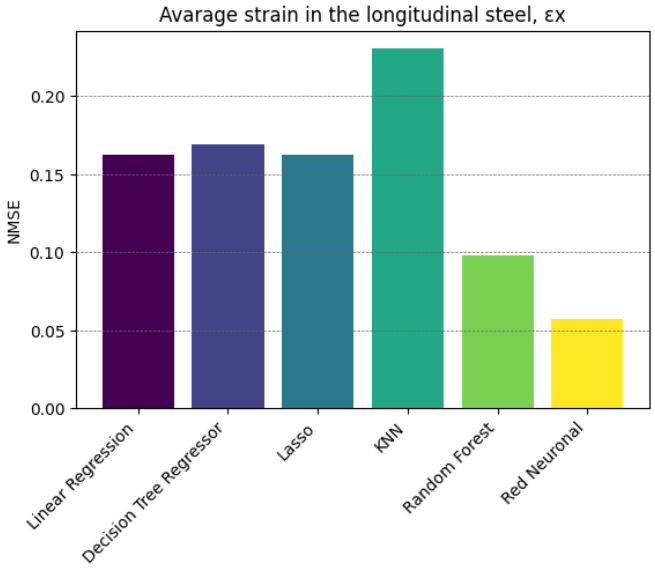

Figura 4.3: Resultados del error NMSE en la variable *Average strain in the longitudinal steel* - Multisalida

A continuación, se muestra una comparación entre los valores reales y los predichos para los algoritmos de la red neuronal y KNN que son los que mejor y peor resultados han obtenido respectivamente.

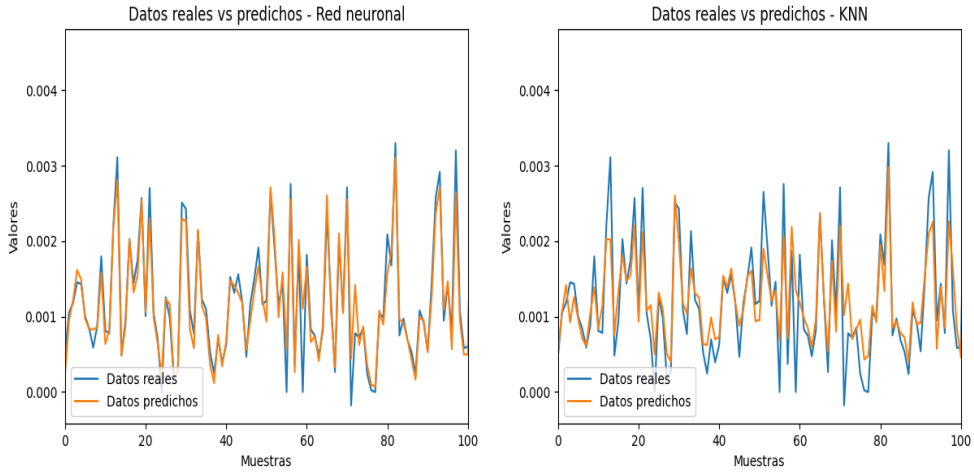

Figura 4.4: Comparación entre los valores reales y los predichos en la red neuronal y en KNN de la variable *Average strain in the longitudinal steel* - Multisalida

En esta comparación 4.4 entre los valores reales y los predichos para la variable *Average strain in the longitudinal steel* se puede apreciar que en el caso de la red neuronal los resultados son claramente más

precisos a los datos y se puede ver como las predicciones en los puntos más alejados presentan mejores resultados que el algoritmo KNN.

Shear internal force

El análisis de la variable *Shear internal force* ha obtenido buenos resultados en el modelo de predicción de la red neuronal, seguido por los algoritmos lineales. Sin embargo, esta vez, el modelo de árbol de decisión es el que ha realizado peores predicciones que el resto de algoritmos.

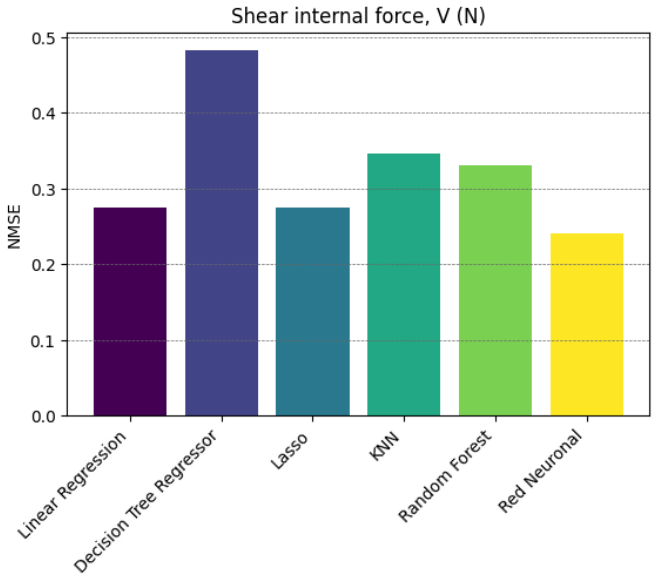

Figura 4.5: Resultados del error NMSE en la variable *Shear internal force* - Multisalida

A continuación, se puede ver cómo es la comparación de las predicciones que realizan el modelo de la red neuronal y el árbol de decisión.

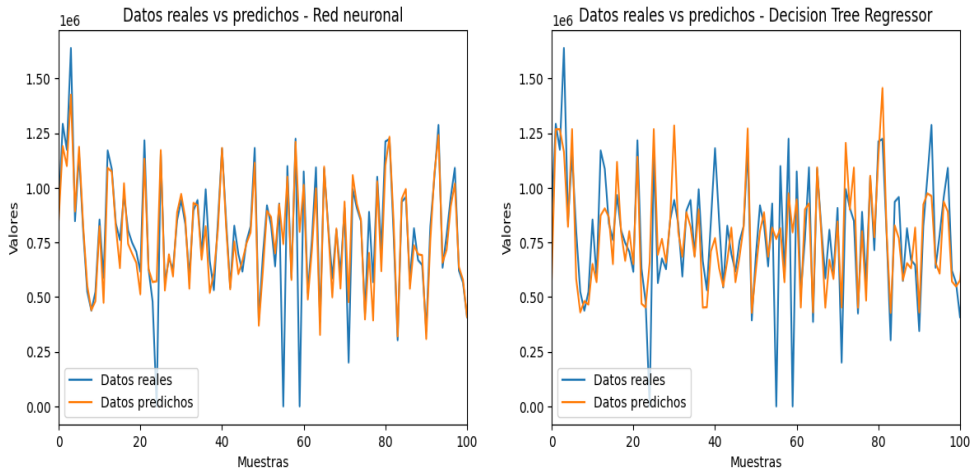

Figura 4.6: Comparación entre los valores reales y los predichos en la red neuronal y en *decision tree regressor* de la variable *Shear internal force* - Multisalida

La figura 4.6 muestra una clara diferencia en cuanto a lo que es capaz de predecir cada algoritmo, siendo la red neuronal la que mejor se ajusta a los valores reales. Es cierto que es complicado predecir para

ambos modelos los puntos atípicos, pero aun así la red neuronal se ajusta bastante bien a la distribución de los datos.

Average strain in the transverse steel

La variable *Average strain in the transverse steel* ha obtenido muy buenos resultados en los algoritmos lineales, a parte de la red neuronal, esto es debido a que como se ha visto anteriormente en la matriz de correlaciones 3.5 se ha observado una fuerte correlación lineal con la variable *Principal tensile strain*. Por otra parte, el algoritmo de árbol de decisión y KNN se han comportado peor, como se puede ver en la siguiente figura.

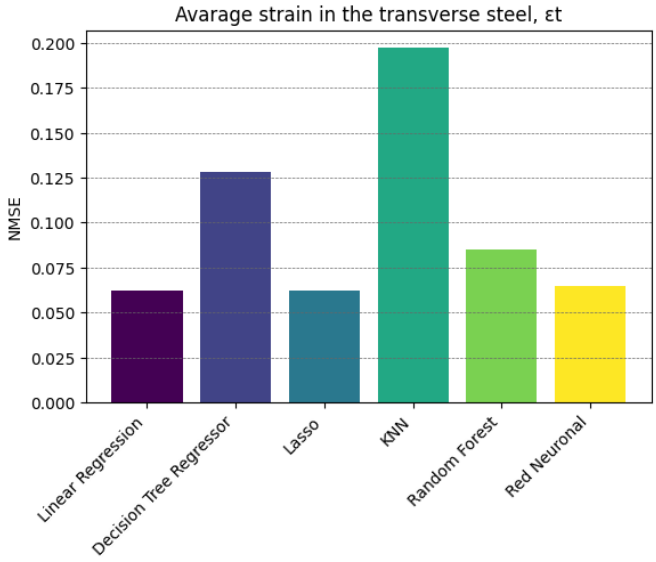

Figura 4.7: Resultados del error NMSE en la variable *Average strain in the transverse steel* - Multisalida

La siguiente imagen muestra la comparación del algoritmo que mejores resultados ha presentado frente al que peor se ha comportado.

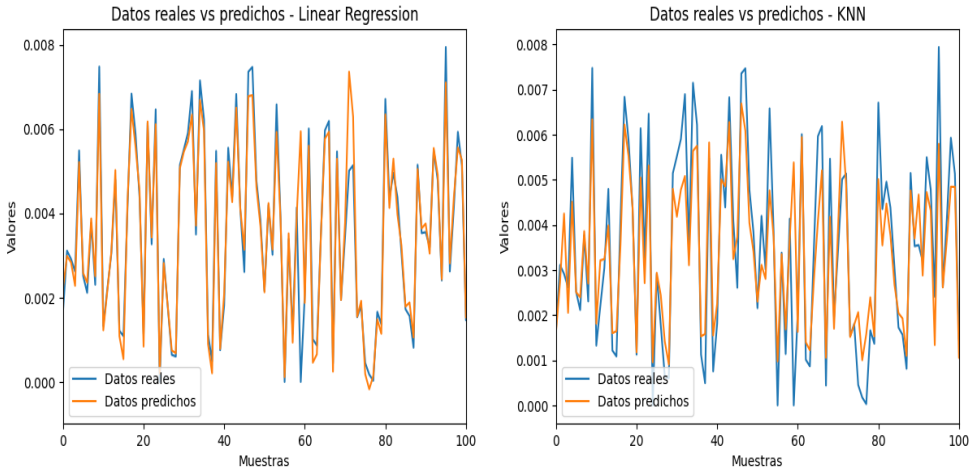

Figura 4.8: Comparación entre los valores reales y los predichos en la regresión lineal y en KNN de la variable *Average strain in the transverse steel* - Multisalida

Como se ha podido observar en la figura 4.8 las predicciones del algoritmo de regresión lineal son las que mejor se ajustan a los datos reales.

Principal compressive strain

La variable *Principal compressive strain* ha obtenido valores más altos del error NMSE que se traduce a una peor predicción que el resto de magnitudes, que puede estar causada por la presencia de numerosos puntos atípicos bastante alejados de la media. A pesar de esto, la red neuronal es la que mejor se ha adaptado y mejores resultados ha obtenido bastante diferenciada del resto de algoritmos que han obtenido peores resultados como KNN y los modelos basados en árboles de decisión. En cuanto a los métodos de regresión lineal han obtenido mejores resultados que estos últimos, que pueden estar debidos a una fuerte correlación lineal con la variable Principal tensile strain.

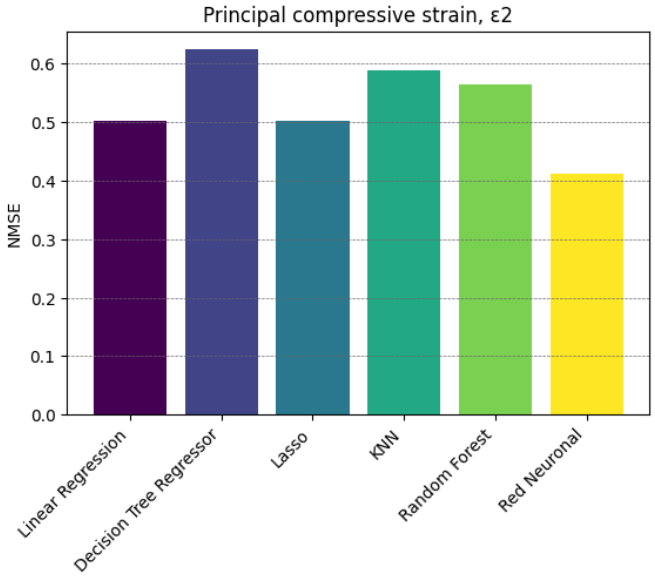

Figura 4.9: Resultados del error NMSE en la variable *Principal compressive strain* - Multisalida

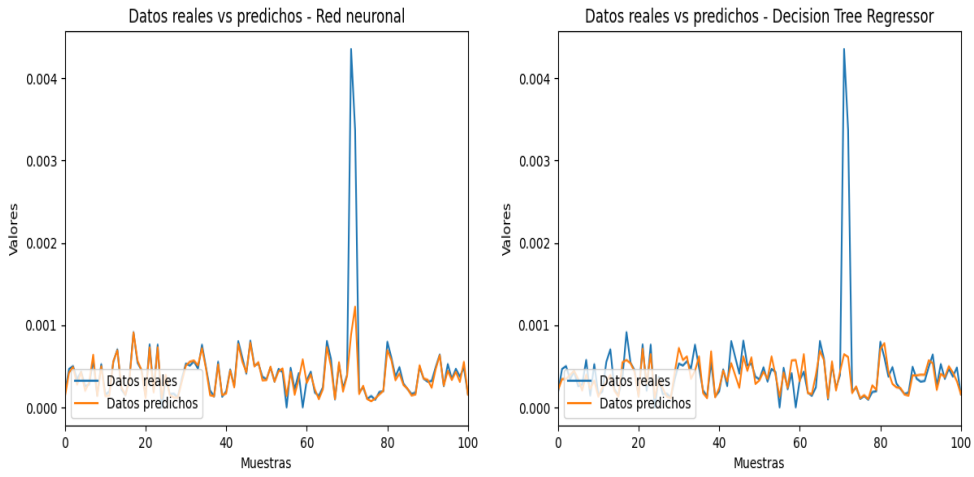

Figura 4.10: Comparación entre los valores reales y los predichos en la red neuronal y en decision tree regressor de la variable *Principal compressive strain* - Multisalida

Como se puede observar en la figura 4.10 la red neuronal se ajusta mucho mejor que el algoritmo *Decision tree regressor*, siendo así los resultados de la predicción mucho más fieles a los datos reales. En las gráficas también se puede apreciar un punto atípico, en el que ambos modelos no logran predecir con exactitud el resultado real. Sin embargo, la red neuronal es la que más se podría acercar a este dato siendo su valor mucho más alto que el que predice el Decision tree regressor.

Average strain in the prestressing strands

Donde mejores resultados se han obtenido en la variable *Average strain in the prestressing strands* ha sido en la red neuronal, sin embargo, los modelos lineales han resultado ser una buena solución, debido a la correlación lineal que mantiene con algunas de las variables. Como era de esperar, los algoritmos basados en árboles de decisión y el KNN representan peor la distribución de los datos.

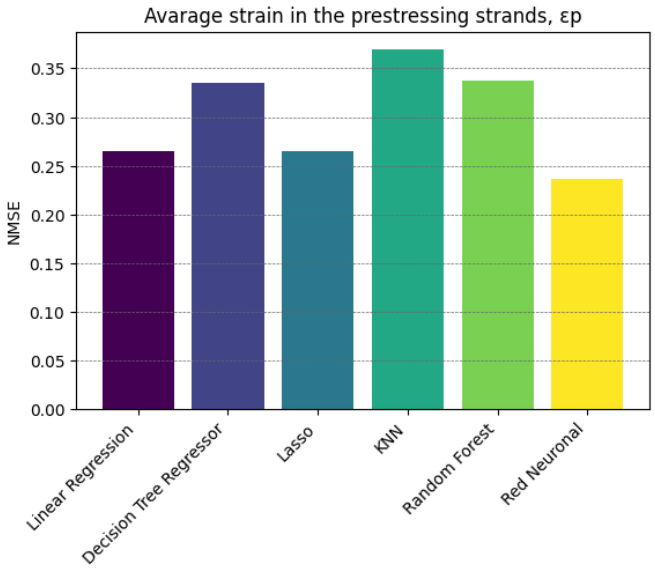

Figura 4.11: Resultados del error NMSE en la variable *Average strain in the prestressing strands* - Multisalida

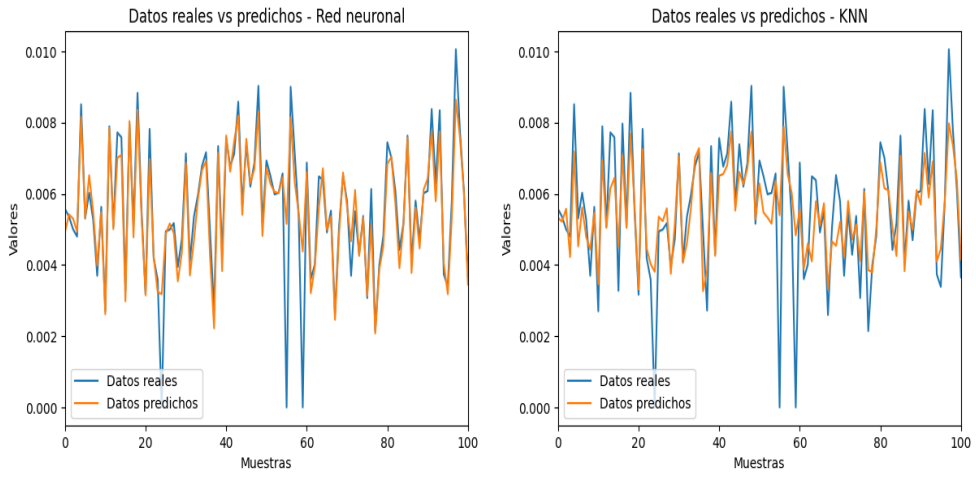

Figura 4.12: Comparación entre los valores reales y los predichos en la red neuronal y en KNN de la variable *Average strain in the prestressing strands* - Multisalida

La figura 4.12 muestra una mejor precisión en la predicción de la red neuronal que se ajusta en mayor medida a la distribución que siguen los datos. Es cierto que los puntos atípicos que se encuentran en el modelo son difíciles de predecir para cualquiera de los dos algoritmos.

Prestressing steel stress

En la variable *Prestressing steel stress* se han obtenido resultados prácticamente idénticos que en el caso anterior, esto es debido a que ambas magnitudes presentan las mismas distribuciones y correlaciones lineales, por tanto es lógico esperar resultados similares.

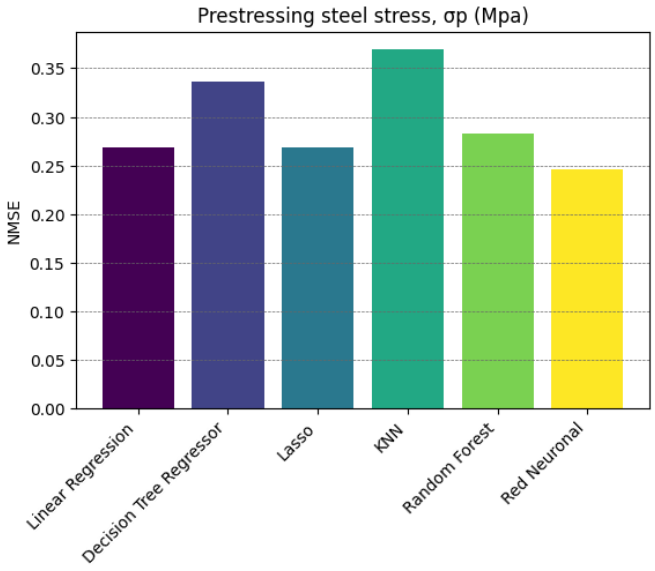

Figura 4.13: Resultados del error NMSE en la variable *Prestressing steel stress* - Multisalida

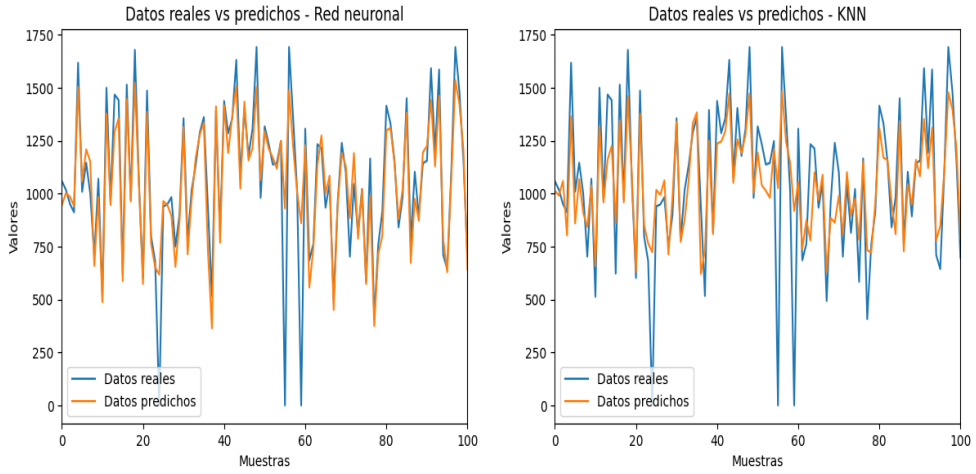

Figura 4.14: Comparación entre los valores reales y los predichos en la red neuronal y en KNN de la variable *Prestressing steel stress* - Multisalida

Las predicciones como eran de esperar, se han obtenido los mismos resultados que para el caso anterior.

Principal compressive stress

La variable *Principal compressive stress* tiene una distribución parecida a los casos anteriores, la diferencia es que las correlaciones lineales son más débiles y por eso salen resultados superiores en el error NMSE en los algoritmos basados en regresión lineal. A pesar de esto, la red neuronal sí que ha obtenido un error NMSE similar a los dos casos anteriores.

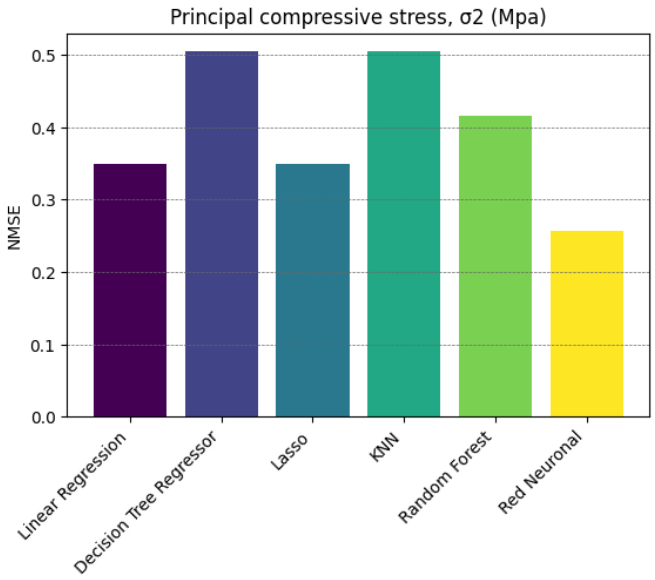

Figura 4.15: Resultados del error NMSE en la variable *Principal compressive stress* - Multisalida

A continuación, se muestra la diferencia en las predicciones de la red neuronal y el algoritmo KNN.

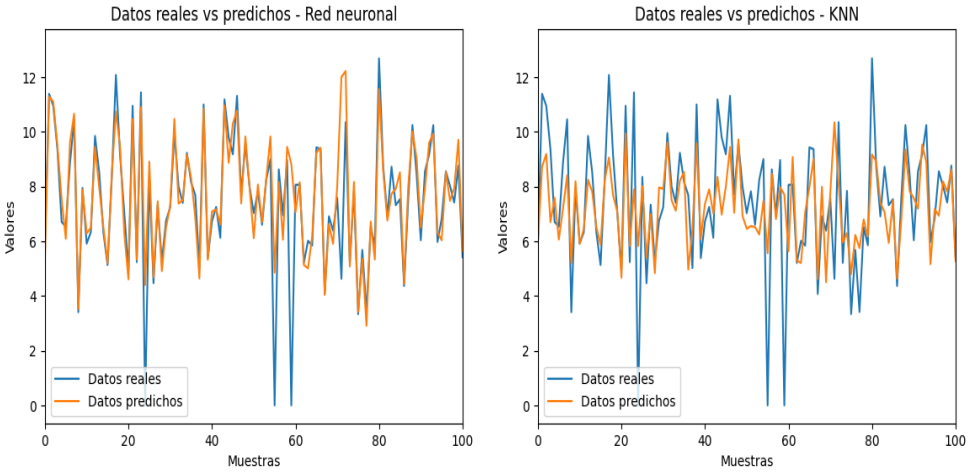

Figura 4.16: Comparación entre los valores reales y los predichos en la red neuronal y en KNN de la variable *Principal compressive stress* - Multisalida

En esta gráfica 4.16 se observa claramente que la red neuronal proporciona una mejor predicción de los datos que el algoritmo KNN. Sin embargo, ninguno de los algoritmos es capaz de predecir con precisión los puntos atípicos, aunque la red neuronal es el que más se acerca a los datos reales.

Average tensile stress in the longitudinal steel

En este caso la variable *Average tensile stress in the longitudinal steel* ha obtenido resultados distintos que las anteriores y es que esta vez, los modelos basados en árboles de decisión presentan mejores resultados que el resto, sin embargo, siguen sin alcanzar los obtenidos de nuevo con la red neuronal.

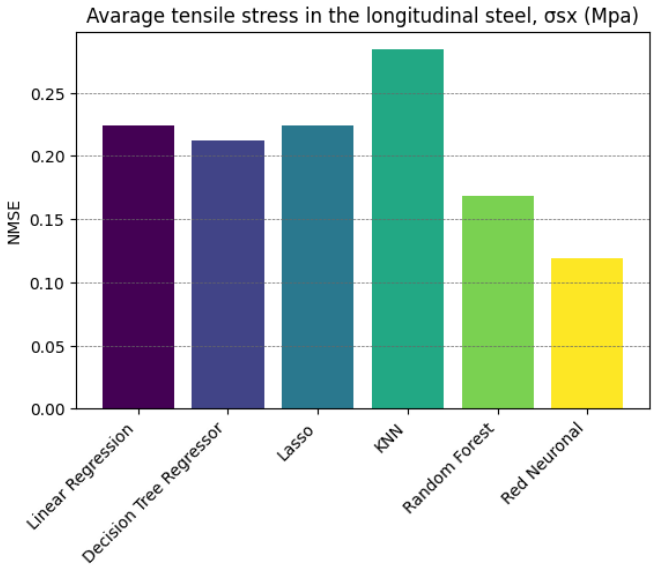

Figura 4.17: Resultados del error NMSE en la variable *Average tensile stress in the longitudinal steel* - Multisalida

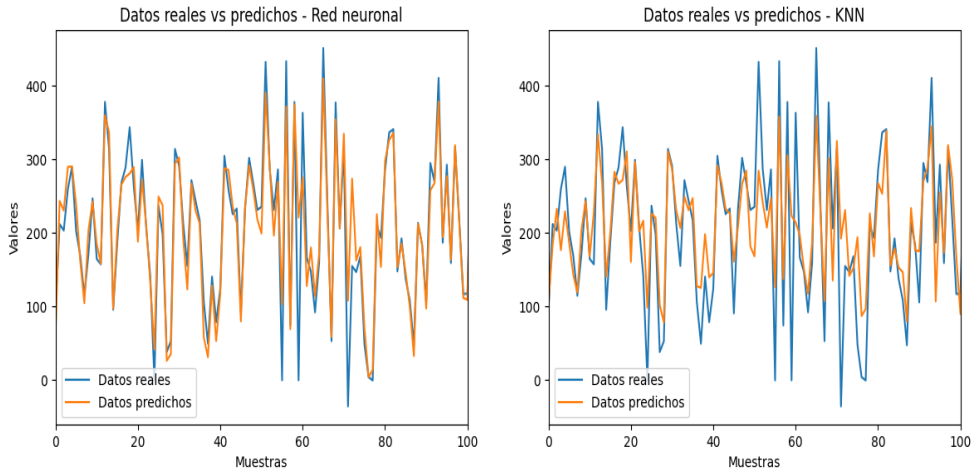

Figura 4.18: Comparación entre los valores reales y los predichos en la red neuronal y en KNN de la variable *Average tensile stress in the longitudinal steel* - Multisalida

De nuevo se diferencia claramente la precisión que se obtiene al utilizar una red neuronal en vez de utilizar cualquier otro modelo.

Average tensile stress in the transverse steel

En el caso de la variable *Average tensile stress in the transverse steel* el mejor modelo sigue siendo la red neuronal, también se han obtenido buenos resultados en los algoritmos basados en árboles de decisión.

Sin embargo, los modelos lineales y el KNN no han llegado a proporcionar tan buenos resultados como los demás.

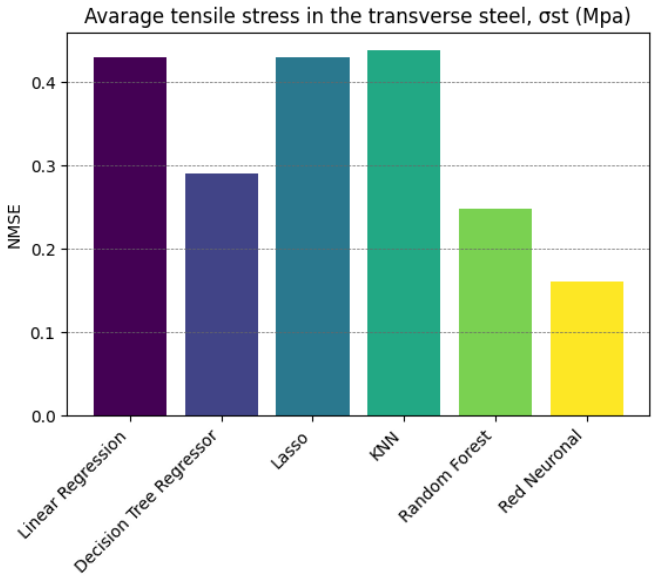

Figura 4.19: Resultados del error NMSE en la variable *Average tensile stress in the transverse steel* - Multisalida

En este caso se va a mostrar una comparación de los algoritmos que mejor han explicado la distribución de los datos y tienen mayor precisión a la hora de predecir.

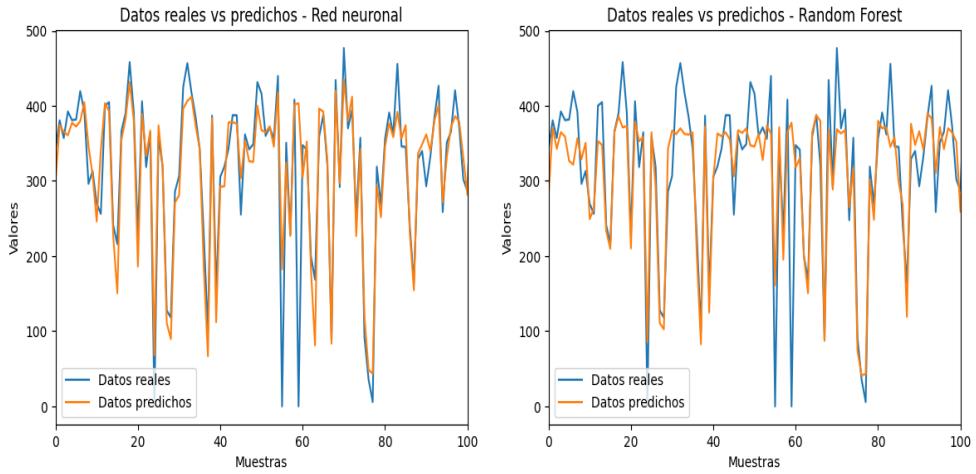

Figura 4.20: Comparación entre los valores reales y los predichos en la red neuronal y en *Random forest* de la variable *Average tensile stress in the transverse steel* - Multisalida

Se puede apreciar en la figura 4.20 que las predicciones del algoritmo *Random forest* son muy similares a las que predice la red neuronal, para datos cuyos valores son más bajos. Sin embargo, el modelo basado en árboles de decisión no predice de manera precisa los valores más altos. En este caso la red neuronal sí que es capaz de predecir dichos valores o por lo menos ajustarse bastante bien al resultado real.

4.2.2 Análisis de una única salida

El objetivo de este análisis es predecir una única variable dependiente en función de todas las variables independientes. Al igual que el análisis multisalida son utilizados como entradas todos los grupos de entrenamiento de las variables predictoras, pero la diferencia es que, como salida para entrenar el algoritmo, sólo se utiliza el grupo de entrenamiento de una única variable predicha. De esta manera se tiene un modelo distinto y personalizado para cada variable de salida y se ajustará mejor a cada una de ellas, ya que no se tiene en cuenta el resto de variables dependientes.

Al igual que en el análisis multisalida se compararán los resultados de los errores NMSE de los distintos algoritmos para cada variable.

En este caso se han podido utilizar todos los algoritmos, ya que en el análisis multisalida no era posible la implementación de alguno de ellos debido a la presencia de varias salidas a la vez. Sin embargo, la red neuronal solo ha obtenido resultados concluyentes en algunas de las variables.

Angle of the principal compressive stress

La variable *Angle of the principal compressive stress* es la que peores resultados ha obtenido en la predicción y prácticamente todos los modelos propuestos se encuentran entre el mismo rango de error NMSE. A pesar de esto, el algoritmo SVR es el que mejor se ha adaptado a la distribución de los datos, seguido muy de cerca de la red neuronal.

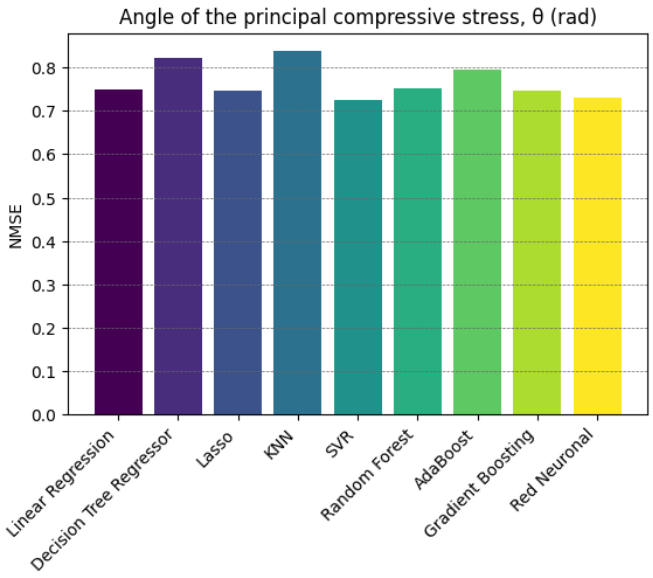

Figura 4.21: Resultados del error NMSE en la variable *Angle of the principal compressive stress*

A continuación, se muestra como es de fiel la predicción del algoritmo SVR respecto a los datos reales.

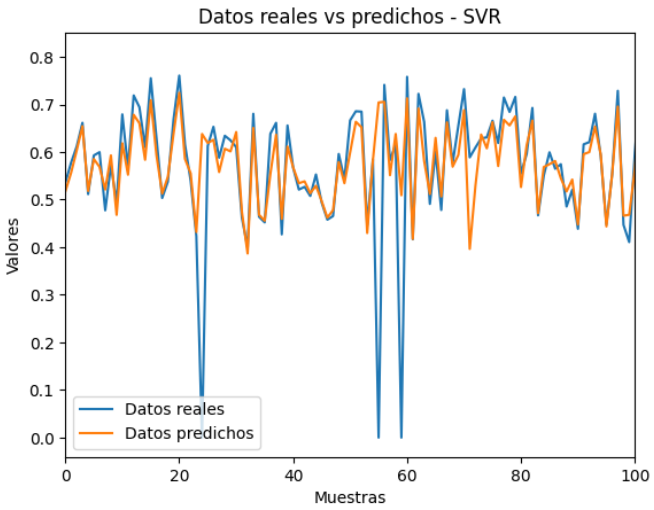

Figura 4.22: Comparación entre los valores reales y los predichos en el modelo SVR de la variable *Angle of the principal compressive stress*

Average strain in the longitudinal steel

En este caso los algoritmos utilizados en la variable *Average strain in the longitudinal steel* presentan resultados más dispares entre sí, habiendo una clara diferencia entre qué modelos son los mejores. De nuevo es el método SVR el que mejor error NMSE ha obtenido, seguido del Gradient Boosting. El modelo de *AdaBoost*, árbol de decisión, KNN y los métodos lineales han obtenido peores resultados.

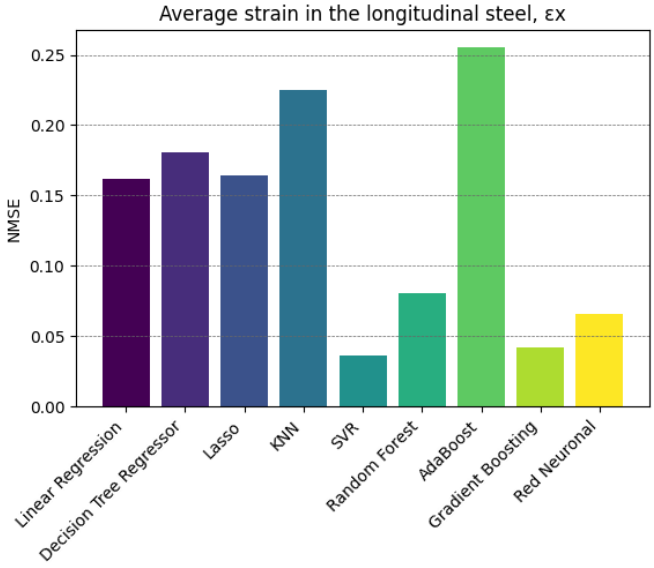

Figura 4.23: Resultados del error NMSE en la variable *Average strain in the longitudinal steel*

A continuación, se muestra una comparativa entre los modelos que mejor y peor NMSE han tenido, para que se aprecie como de bien se ajustan las predicciones a los valores reales.

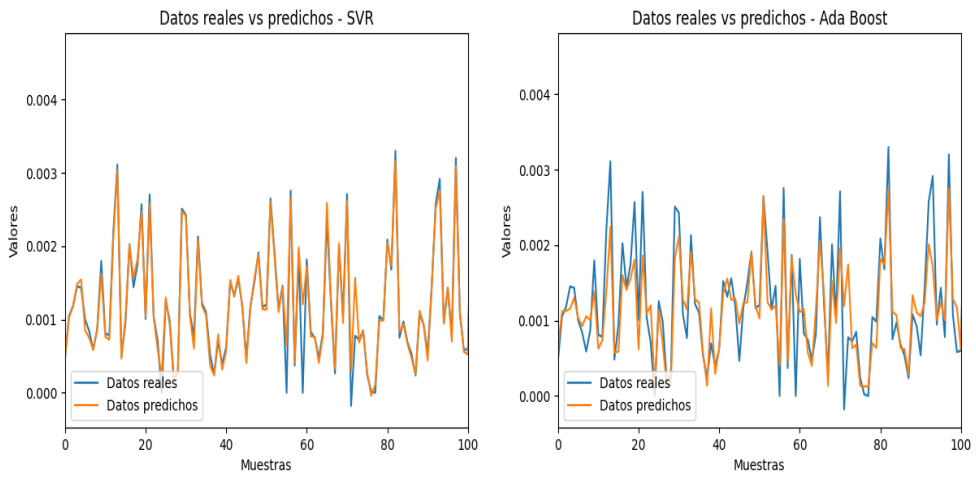

Figura 4.24: Comparación entre los valores reales y los predichos en los algoritmos SVR y *AdaBoost* de la variable *Average strain in the longitudinal steel*

La figura 4.24 muestra cómo las predicciones del método SVR son muy fieles a los datos reales.

Shear internal force

La variable *Shear internal force* sigue la misma tendencia respecto a qué algoritmos son los más representativos. Al igual que antes se tiene que el valor más bajo de NMSE que se ha obtenido ha sido utilizando el algoritmo SVR, mientras que AdaBoost, KNN y el árbol de decisión han obtenido peores resultados.

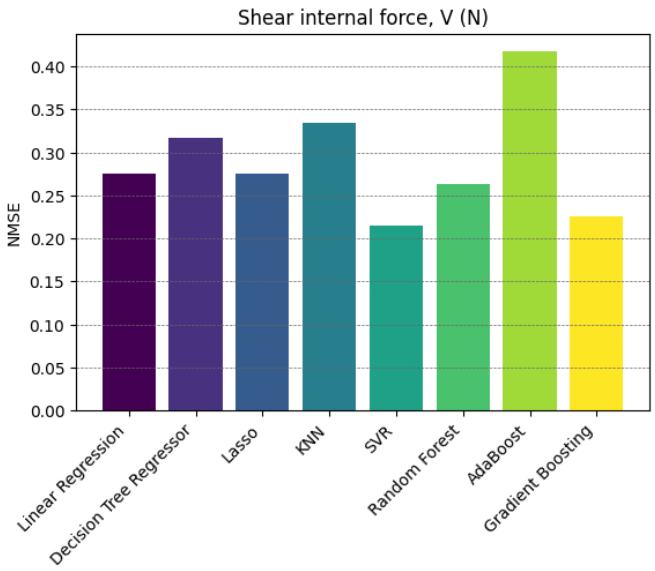

Figura 4.25: Resultados del error NMSE en la variable *Shear internal force*

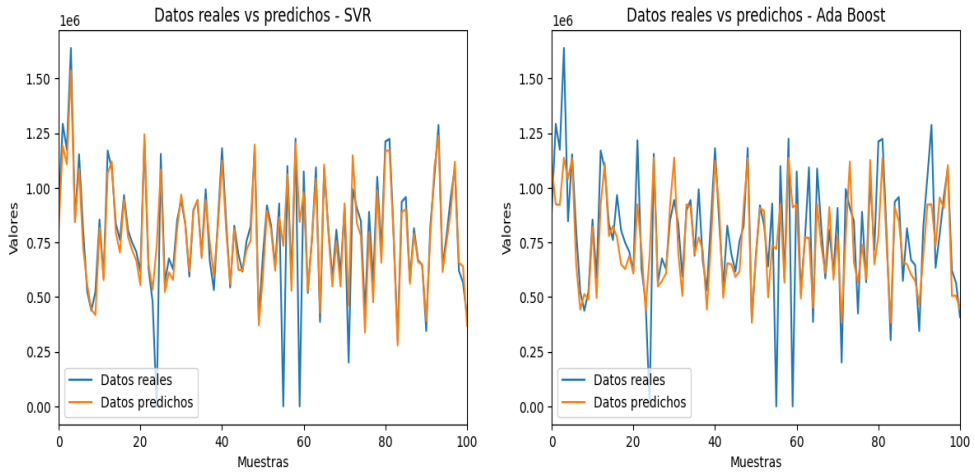

Figura 4.26: Comparación entre los valores reales y los predichos en los algoritmos SVR y *AdaBoost* de la variable *Shear internal force*

Claramente en la figura 4.26 se observa una clara diferencia entre las predicciones del algoritmo Ada Boost y el SVR, el cual se ajusta en gran medida a la distribución de los datos, acercándose bastante bien a los datos reales. Al ser una variable que presenta puntos atípicos es difícil de predecir dichos valores, es por ello por lo que el algoritmo presenta dificultades para predecir los datos de magnitud más baja que el resto.

Average strain in the transverse steel

En este caso la variable *Average strain in the transverse steel* ha obtenido muy buenos resultados en general. Es una variable fácil de predecir y no está influenciada por puntos atípicos. De nuevo, el algoritmo que mejor valor de NMSE ha tenido, ha sido el SVR. Los modelos lineales también han obtenido un buen resultado, debido a que esta variable tiene una alta correlación lineal con la variable Principal tensile strain. Aunque ha sido fácil de predecir, el algoritmo KNN no se ha ajustado tan bien como los demás, obteniendo así un valor de NMSE superior al resto.

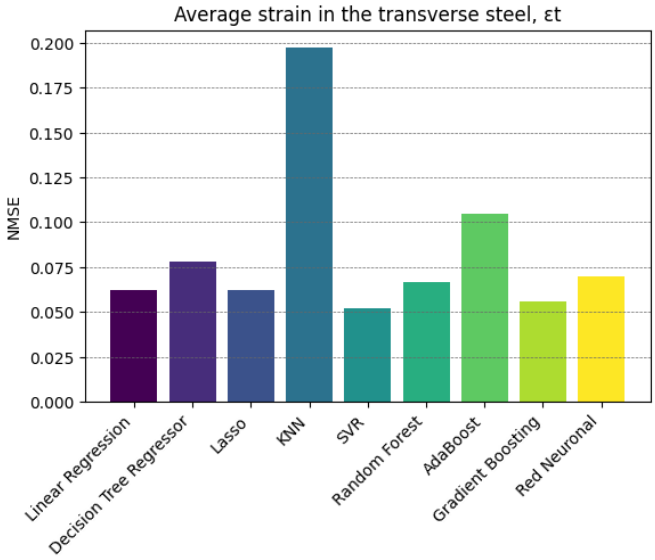

Figura 4.27: Resultados del error NMSE en la variable *Average strain in the transverse steel*

A continuación, se va a comparar la regresión lineal, ya que es una variable con alta correlación lineal, respecto al algoritmo SVR.

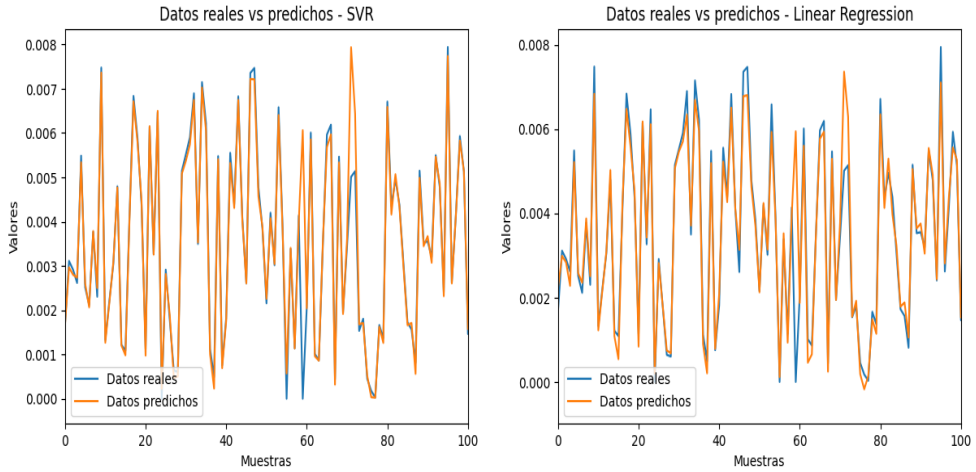

Figura 4.28: Comparación entre los valores reales y los predichos en los algoritmos SVR y Regresión lineal de la variable *Average strain in the transverse steel*

Como se puede ver en la figura 4.28 las predicciones que realiza el método SVR son bastante precisas respecto a los datos reales. A pesar de que, en este caso, la regresión lineal también ha obtenido buenos resultados, no se ajusta tan bien a los datos reales como lo hace SVR, sobre todo en los valores extremos.

Principal compressive strain

La variable *Principal compressive strain* es una de las que mayores dificultades presenta en general para ser predicha. Sin embargo, en este caso la red neuronal es la que mejor resultado ha obtenido, a diferencia del árbol de decisión y *AdaBoost* que han conseguido valores superiores de NMSE.

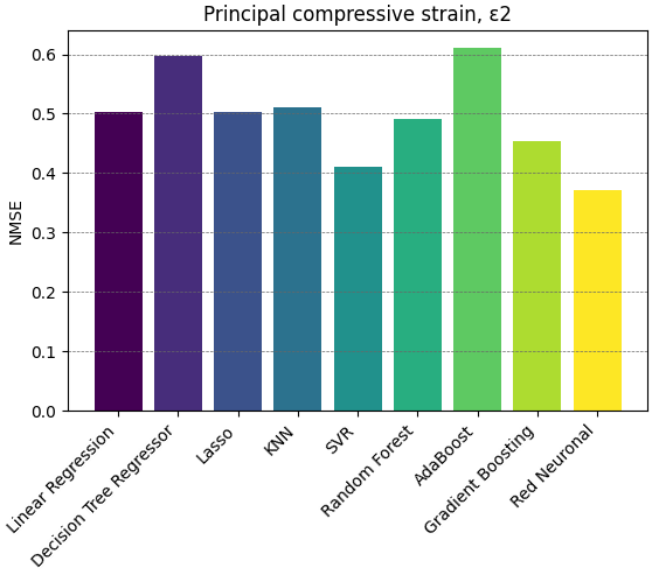

Figura 4.29: Resultados del error NMSE en la variable *Principal compressive strain*

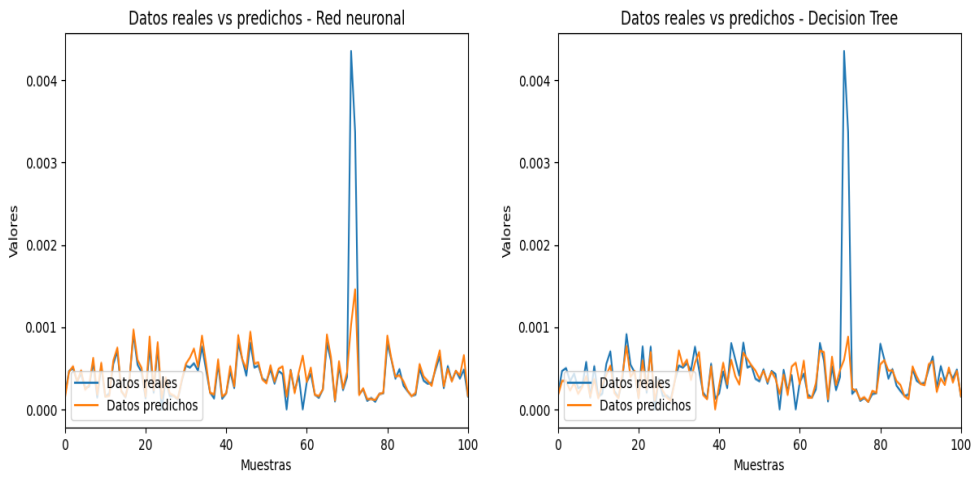

Figura 4.30: Comparación entre los valores reales y los predichos en la red neuronal y el árbol de decisión de la variable *Principal compressive strain*

En este caso la figura 4.30 muestra la comparación entre el mejor resultado obtenido con la red neuronal con los valores predichos por un árbol de decisión, para que se pueda observar la precisión de la red neuronal. Como era de esperar, los valores atípicos que presenta la variable son más difíciles de predecir, aun así, la red neuronal logra predecir que existe un comportamiento fuera de la normalidad.

Average strain in the prestressing strands

En este caso, la variable *Average strain in the prestressing strands* vuelve a obtener mejores resultados utilizando el algoritmo SVR, muy seguido del *Gradient Boosting*. Por el contrario, los valores del error NMSE de los algoritmos *AdaBoost* y KNN han sido bastante mayores que el resto.

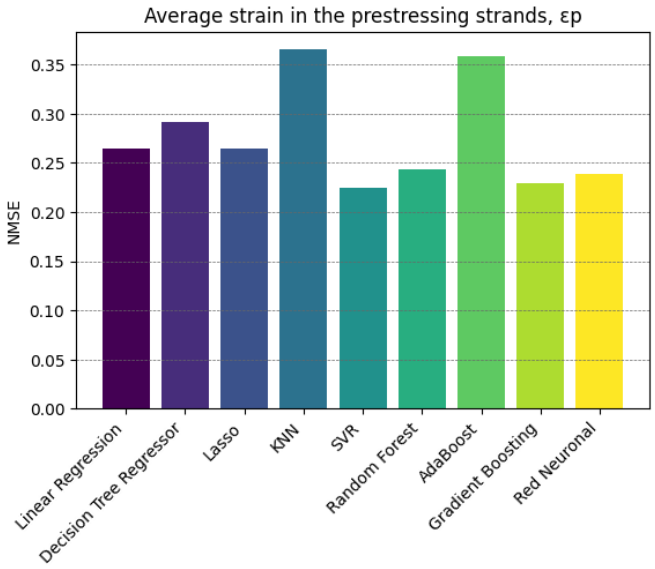

Figura 4.31: Resultados del error NMSE en la variable *Average strain in the prestressing strands*

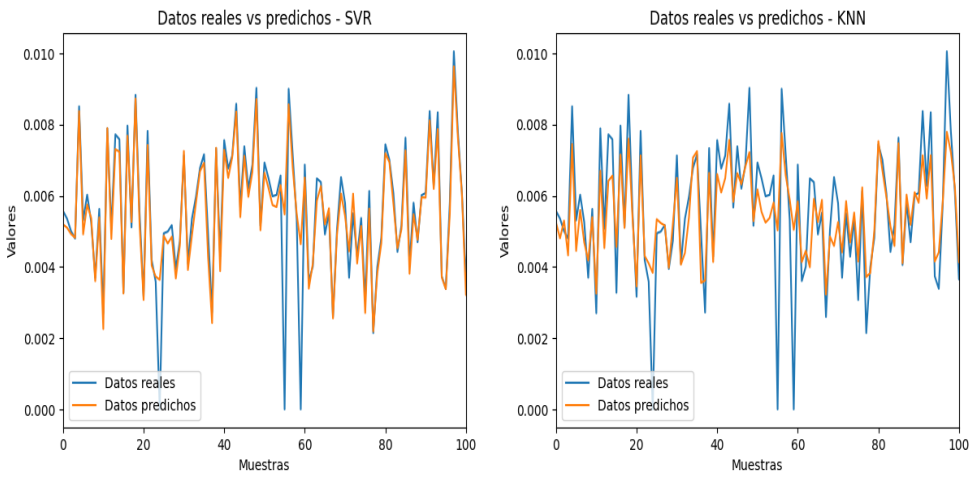

Figura 4.32: Comparación entre los valores reales y los predichos en los algoritmos SVR y KNN de la variable *Average strain in the prestressing strands*

En esta comparación 4.32 se observa claramente que las predicciones obtenidas por el método SVR son mucho más precisas que las del método KNN. Sin embargo, al contener puntos atípicos es muy difícil que las predicciones se ajusten del todo y ambos modelos tienen dificultades en los valores extremos.

Prestressing steel stress

Al igual que sucedía en el análisis multisalida, la variable *Prestressing steel stress* presenta prácticamente los mismos resultados que en la anterior. Ambas magnitudes tienen la misma distribución de datos y correlaciones lineales, por tanto, es de esperar resultados parecidos.

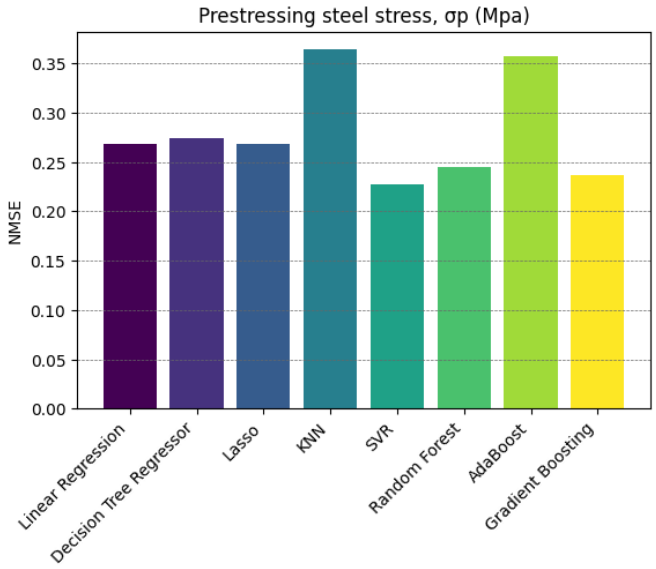

Figura 4.33: Resultados del error NMSE en la variable *Prestressing steel stress*

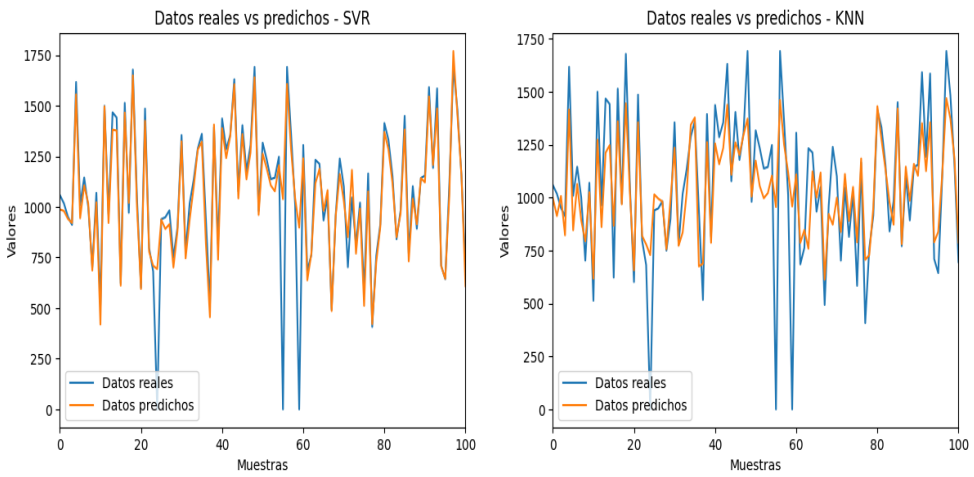

Figura 4.34: Comparación entre los valores reales y los predichos en los algoritmos SVR y KNN de la variable *Prestressing steel stress*

En la figura 4.34 se observa claramente que el algoritmo SVR tiene una buena capacidad de predicción, que destaca sobre el resto de modelos. De nuevo, esta variable presenta puntos atípicos que hacen que sean difíciles de predecir.

Principal compressive stress

En la variable *Principal compressive stress* el modelo que mejores resultados ha obtenido ha sido SVR, seguido de *Gradient Boosting*. Cabe destacar que en este caso *Random Forest* y los algoritmos de regresión lineal también se han ajustado bastante bien a la distribución de los datos.

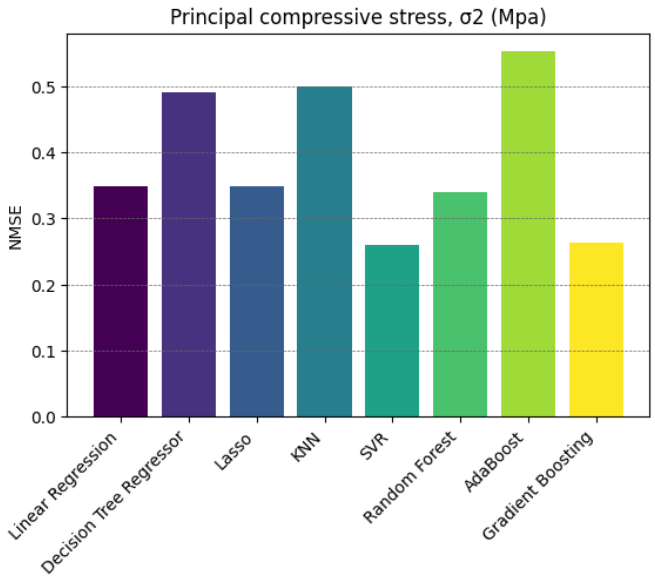

Figura 4.35: Resultados del error NMSE en la variable *Principal compressive stress*

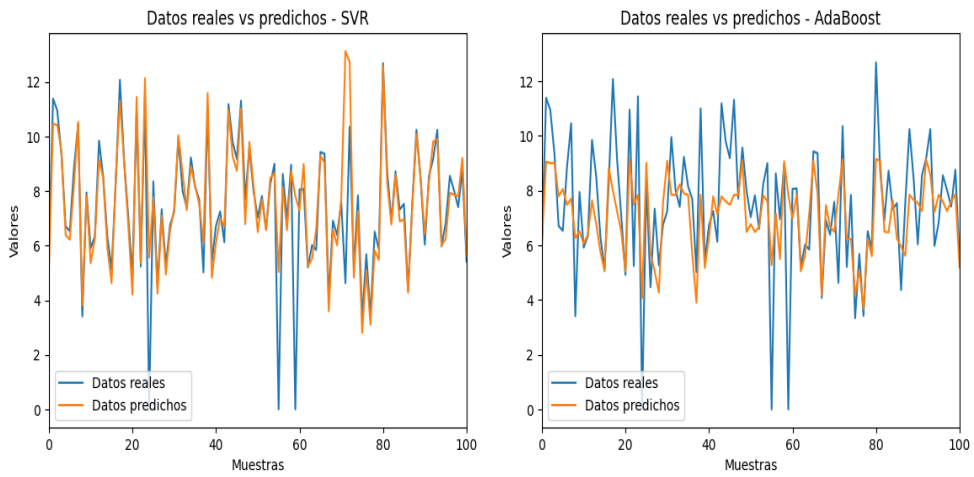

Figura 4.36: Comparación entre los valores reales y los predichos en los algoritmos SVR y *AdaBoost* de la variable *Principal compressive stress*

En la figura 4.36 se muestra la comparación entre las predicciones que realizan los algoritmos SVR y AdaBoost. Hay una diferencia clara en las predicciones que se obtienen de SVR, pues se ajustan en gran medida a los valores reales. Sin embargo, sigue teniendo dificultades en los puntos atípicos que presenta la variable.

Average tensile stress in the longitudinal steel

En general para la variable *Average tensile stress in the longitudinal steel* los errores NMSE no han sido altos, pero sí que ha habido una diferencia clara entre algunos de ellos. El algoritmo SVR es el que menor valor del error NMSE ha obtenido. También los resultados del *Random Forest* y *Gradient Boosting* han sido de los mejores. Los errores NMSE del resto de modelos han sido más altos.

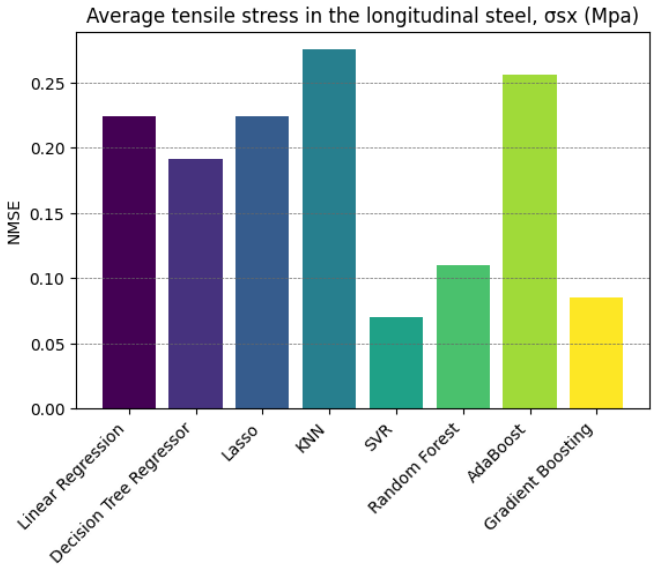

Figura 4.37: Resultados del error NMSE en la variable *Average tensile stress in the longitudinal steel*

A continuación, se muestra una comparación de las predicciones entre los dos mejores algoritmos de predicción.

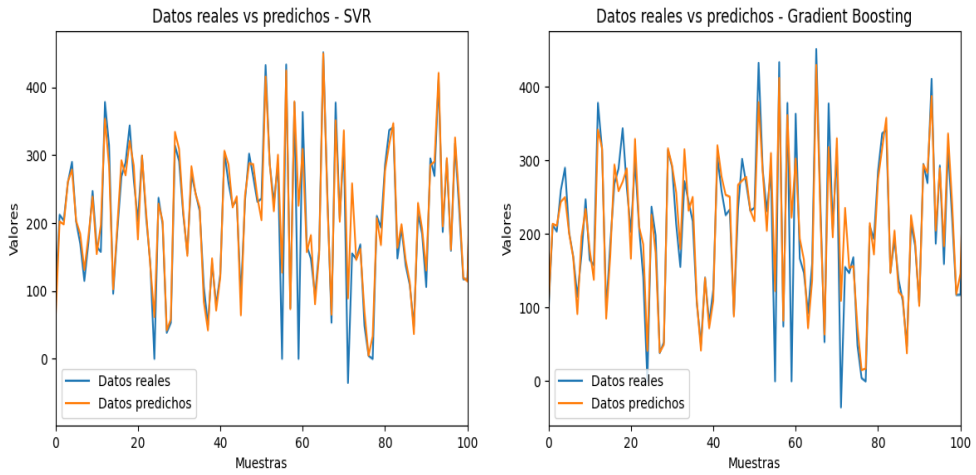

Figura 4.38: Comparación entre los valores reales y los predichos en los algoritmos SVR y *Gradient Boosting* de la variable *Average tensile stress in the longitudinal steel*

Como se puede ver en la figura 4.38 las predicciones del modelo SVR son las que mejor se ajustan a los datos reales. El algoritmo *Gradient Boosting* no consigue ajustarse del todo a los valores más extremos, sin embargo, SVR lo hace de una manera más precisa.

Average tensile stress in the transverse steel

En este caso, para la variable *Average tensile stress in the transverse steel*, el mejor resultado se ha obtenido con el algoritmo *Gradient Boosting*. También se han comportado muy bien SVR y los modelos compuestos en árboles de decisión. Sin embargo, esta vez, los que están basados en la regresión lineal no han conseguido valores aceptables del error NMSE.

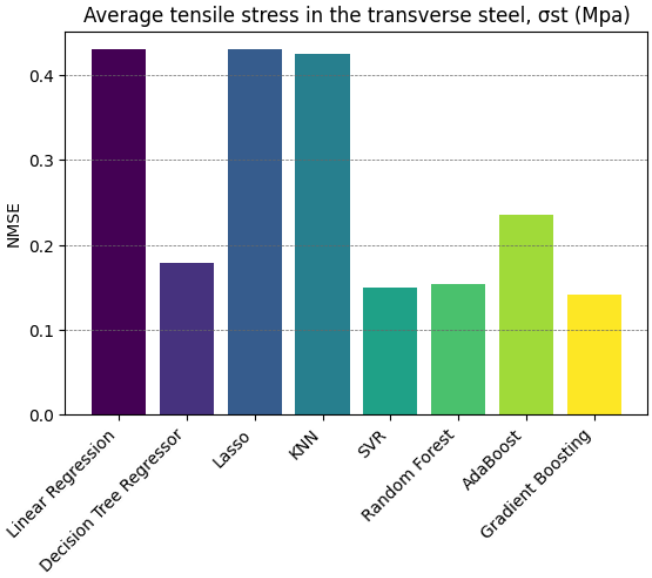

Figura 4.39: Resultados del error NMSE en la variable *Average tensile stress in the transverse steel*

En este caso se van a comparar el mejor modelo, que ha sido el que ha proporcionado el algoritmo *Gradient Boosting*, y *Random Forest*, que también ha obtenido buenos resultados en esta ocasión.

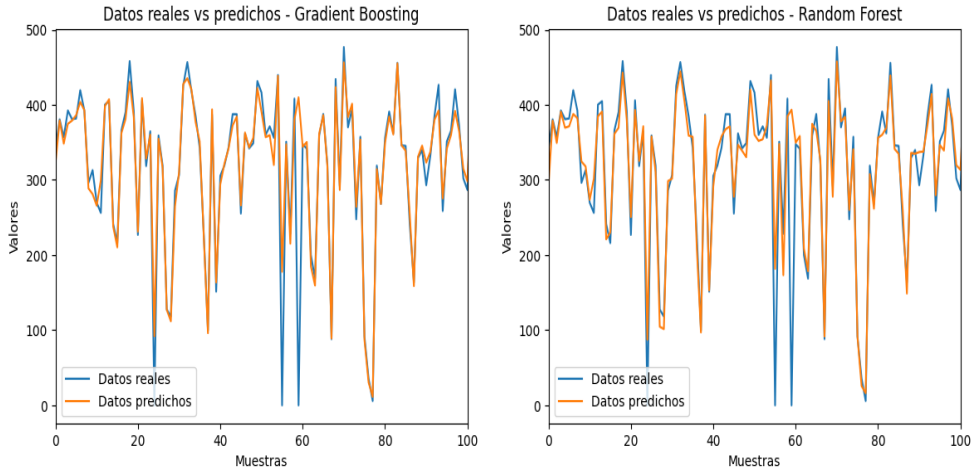

Figura 4.40: Comparación entre los valores reales y los predichos en los algoritmos *Gradient Boosting* y *Random Forest* de la variable *Average tensile stress in the transverse steel*

En la figura 4.40 se comprueba que ambos modelos se comportan bastante bien, y las predicciones que realizan se asemejan en gran medida a los datos reales. Sin embargo, se aprecia que en algunos de los picos no llega a ajustarse tan bien como lo hace *Gradient Boosting*.

4.3 Comparativa de ambos análisis

Una vez presentados todos los resultados que se han obtenido en el análisis multisalida, así como en el de una única salida, se procede a esclarecer cuáles de los dos análisis ha funcionado de mejor manera y qué algoritmos han funcionado mejor.

A continuación, se proporciona una comparativa con un resumen simplificado de todos los resultados de los errores NMSE de cada variable. Se recuerda que cuanto menor sea el error, más precisas serán las predicciones de los modelos.

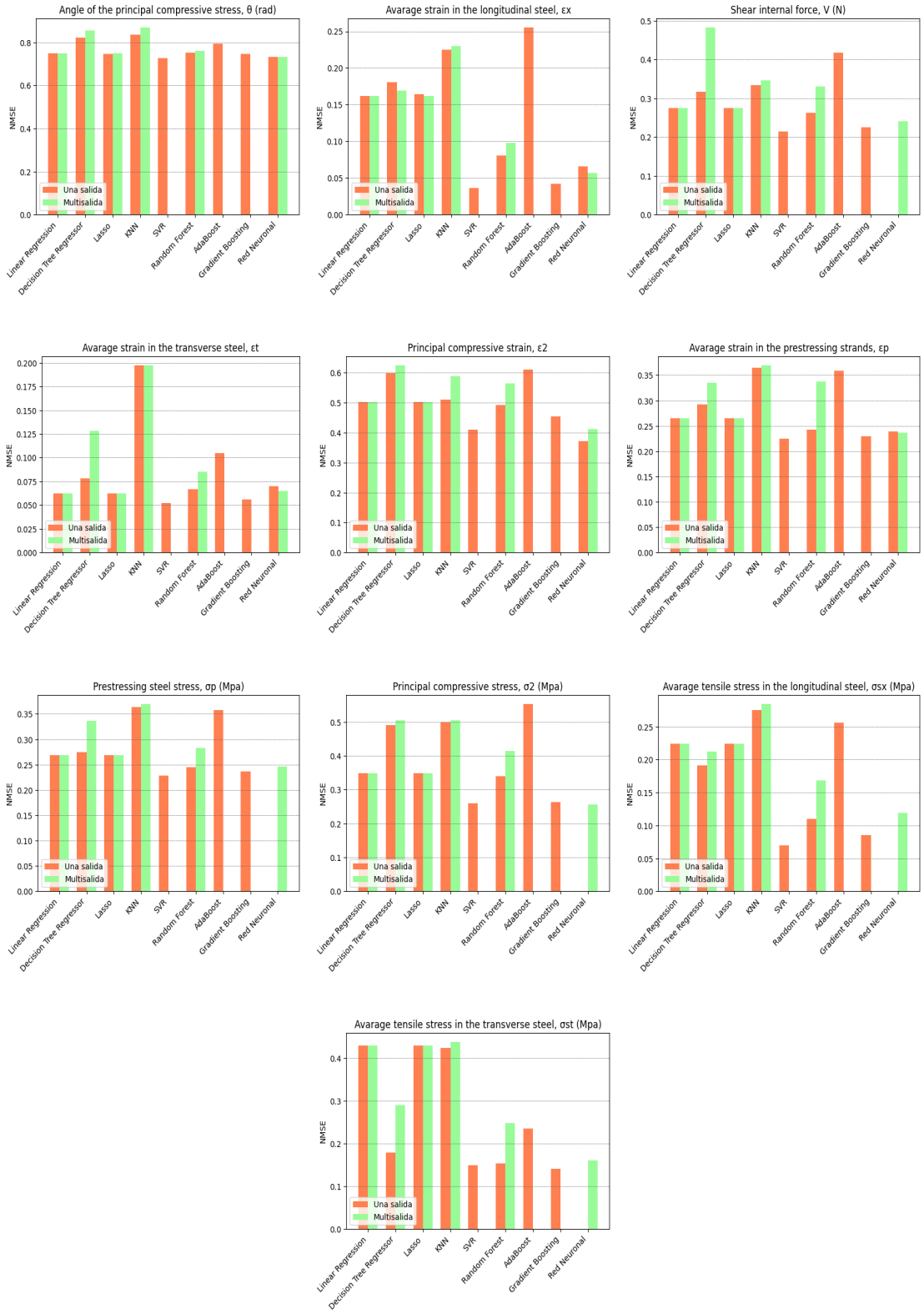

Figura 4.41: Comparativa de los errores NMSE de los algoritmos propuestos para el análisis multisalida y el de una única salida

Se ha establecido, como se puede observar, que el análisis de una única salida supera al realizado con multisalida. En muchas de las variables es una diferencia considerable, aunque sí es cierto que en la

variable Principal compressive stress el mejor resultado ha sido el mismo utilizando el enfoque de una única salida con el algoritmo SVR, que con el enfoque multisalida utilizando la red neuronal.

Es cierto también que, el análisis multisalida ha presentado dificultades a la hora de emplear algunos de los algoritmos de los que se disponían y es que muchos de ellos no soportan este tipo de enfoque. Esto hace que haya una cierta limitación con este análisis.

En cuanto a las limitaciones del análisis de una única salida son pocas. La única limitación que se ha observado ha sido que para algunas de las magnitudes ha sido imposible sacar buenos resultados de la red neuronal. Sin embargo, la red neuronal sí que ha obtenido resultados concluyentes en todas las variables utilizando el análisis multisalida, hasta tal punto que el modelo planteado de DL ha sido el que mejores resultados ha obtenido en todo el enfoque multisalida realizado.

Para mostrar cómo se han comportado los algoritmos en general, se ha realizado la media de los errores NMSE y del R2 de cada algoritmo en todas las variables de salida, de esta manera quedará un error medio NMSE y R2 para cada algoritmo y análisis.

Linear Regression			
Multisalida		Una salida	
R^2	NMSE	R^2	NMSE
0.671	0.329	0.671	0.329

Tabla 4.1: Promedio del R^2 y del error NMSE para el algoritmo *Linear regression*

Decision Tree Regressor			
Multisalida		Una salida	
R^2	NMSE	R^2	NMSE
0.606	0.394	0.658	0.342

Tabla 4.2: Promedio del R^2 y del error NMSE para el algoritmo *Decision Tree Regressor*

Lasso			
Multisalida		Una salida	
R^2	NMSE	R^2	NMSE
0.671	0.329	0.671	0.329

Tabla 4.3: Promedio del R^2 y del error NMSE para el algoritmo *Lasso*

KNN			
Multisalida		Una salida	
R^2	NMSE	R^2	NMSE
0.58	0.42	0.597	0.403

Tabla 4.4: Promedio del R^2 y del error NMSE para el algoritmo *KNN*

SVR	
Una salida	
R^2	NMSE
0.763	0.237

Tabla 4.5: Promedio del R^2 y del error NMSE para el algoritmo *SVR*

Random Forest			
Multisalida		Una salida	
R^2	NMSE	R^2	NMSE
0.673	0.329	0.725	0.276

Tabla 4.6: Promedio del R^2 y del error NMSE para el algoritmo *Random Forest*

AdaBoost	
Una salida	
R^2	NMSE
0.606	0.394

Tabla 4.7: Promedio del R^2 y del error NMSE para el algoritmo *AdaBoost*

Gradient Boosting	
Una salida	
R^2	NMSE
0.752	0.248

Tabla 4.8: Promedio del R^2 y del error NMSE para el algoritmo *Gradient Boosting*

Red neuronal	
Multisalida	
R^2	NMSE
0.737	0.253

Tabla 4.9: Promedio del R^2 y del error NMSE para el algoritmo *Red neuronal*

Hay que destacar que en general el algoritmo que mejor se ha comportado ha sido el SVR en la mayoría de variables. Sin embargo, otros dos algoritmos han destacado obteniendo mejores resultados, como ha sido la red neuronal en la variable *Principal compressive strain* y el modelo de *Gradient Boosting* en la variable *Average tensile stress in the transverse steel*. También, mencionar de nuevo que en la variable *Principal compressive stress* se ha obtenido el mismo resultado utilizando el algoritmo SVR que la red neuronal del análisis multisalida.

Por último, se adjunta a continuación varias tablas que incluyen los resultados de las métricas finales (R^2, MAE, RMSE y NMSE) del algoritmo que ha obtenido mejores resultados para cada variable.

Angle of the principal compressive stress			
SVR			
R^2	MAE	RMSE	NMSE
0.274	0.041	0.112	0.726

Tabla 4.10: Resultados de las métricas del algoritmo final para la variable *Angle of the principal compressive stress*

Average strain in the longitudinal steel			
SVR			
R^2	MAE	RMSE	NMSE
0.964	7.688E-5	1.621E-4	0.036

Tabla 4.11: Resultados de las métricas del algoritmo final para la variable *Average strain in the longitudinal steel*

Shear internal force			
SVR			
R^2	MAE	RMSE	NMSE
0.785	64284.478	132038.62	0.215

Tabla 4.12: Resultados de las métricas del algoritmo final para la variable *Shear internal force*

Average strain in the transverse steel			
SVR			
R^2	MAE	RMSE	NMSE
0.948	1.5E-4	4.712E-4	0.052

Tabla 4.13: Resultados de las métricas del algoritmo final para la variable *Average strain in the transverse steel*

Principal compressive strain			
SVR			
R^2	MAE	RMSE	NMSE
0.613	4.956E-5	1.841E-4	0.371

Tabla 4.14: Resultados de las métricas del algoritmo final para la variable *Principal compressive strain*

Average strain in the prestressing strands			
SVR			
R^2	MAE	RMSE	NMSE
0.775	3.765E-4	8.89E-4	0.225

Tabla 4.15: Resultados de las métricas del algoritmo final para la variable *Average strain in the prestressing strands*

Prestressing steel stress			
SVR			
R^2	MAE	RMSE	NMSE
0.772	69.559	168.471	0.228

Tabla 4.16: Resultados de las métricas del algoritmo final para la variable *Prestressing steel stress*

Principal compressive stress			
SVR			
R^2	MAE	RMSE	NMSE
0.741	0.565	1.145	0.259

Tabla 4.17: Resultados de las métricas del algoritmo final para la variable *Principal compressive stress*

Average tensile stress in the longitudinal steel			
SVR			
R^2	MAE	RMSE	NMSE
0.929	16.172	29.944	0.07

Tabla 4.18: Resultados de las métricas del algoritmo final para la variable *Average tensile stress in the longitudinal steel*

Average tensile stress in the transverse steel			
SVR			
R^2	MAE	RMSE	NMSE
0.859	14.085	41.146	0.141

Tabla 4.19: Resultados de las métricas del algoritmo final para la variable *Average tensile stress in the transverse steel*

Capítulo 5

Conclusiones y lineas futuras

En esta sección se establecen las conclusiones y hallazgos obtenidos en el desarrollo del trabajo. Después de esto, se concluirá con la línea futura que podría seguir del presente trabajo.

5.1 Conclusiones

Este estudio se ha centrado en la predicción de la deformación del acero en elementos de hormigón armado utilizando técnicas de ML y DL, ya que numéricamente es una tarea difícil. Para ello se han seleccionado varios algoritmos y se han probado para determinar cuál de ellos comete el mínimo error en dos tipos de análisis, uno centrado en predecir todas las características del acero a la vez y otro una única característica. Es por ello por lo que el principal objetivo del trabajo es predecir de la manera óptima posible las características del acero.

En base a los resultados obtenidos de este análisis, se ha conseguido concluir que el algoritmo que por lo general mejor se adapta a los datos reales y por consecuencia menor error ha demostrado en las predicciones, ha sido el modelo basado en máquinas de vectores de soporte (SVR) el cuál ha sido empleado en el análisis de una única salida. También, se ha llegado a la conclusión que otros modelos de DL como las redes neuronales ofrecen muy buenos resultados en estos trabajos de predicción.

Cabe destacar que, aunque no era el principal objetivo, ha resultado concluyente mencionar que el análisis de una única salida es mucho más preciso que si se realiza un modelo que contenga todas las variables de respuesta a la vez. Es por ello por lo que se recomienda utilizar un modelo que contenga una única variable a predecir.

Para llevar a cabo todo esto, ha sido necesario realizar una investigación profunda acerca del mundo del ML y DL y comprender como funciona cada una de las técnicas de predicción. Como consecuencia de esto se ha logrado cumplir el objetivo propuesto de aprender conocimientos del amplio campo que es el ML y DL.

A lo largo del proceso de análisis ha sido necesario utilizar como lenguaje de programación Python y se ha conseguido tener un conocimiento más profundo de él, que era otro de los objetivos planteados en la introducción.

Dicho todo esto se concluye que los objetivos propuestos al principio han logrado ser cumplidos satisfactoriamente. Finalmente, se ha llegado también a la conclusión de que la inteligencia artificial es un campo muy grande todavía por explorar y que es totalmente necesario para el futuro próximo, debido

al aumento constante de datos que se manejan y de todas las nuevas tecnologías que harán de este mundo uno más sencillo.

5.2 Lineas futuras

Como líneas futuras de este proyecto se propone principalmente investigar de manera más profunda en cada algoritmo e intentar optimizarlo lo mejor posible incluyendo nuevos hiperparámetros. Los modelos normalmente, son mejorables y es una buena línea de investigación.

En este trabajo se han expuesto un número limitado de algoritmos, puesto que existen más, se propone investigar una mayor cantidad de algoritmos que podrían presentar resultados distintos e interesantes.

El proyecto se ha centrado tanto en técnicas de ML como de DL. Como se ha visto, el DL es un campo complejo y amplio que es fundamental en el futuro próximo y suscita gran interés. Por ello, se propone una investigación exhaustiva acerca de este campo y ver cómo podría mejorar los modelos de predicción.

De la misma manera, se han visto y se han mencionado numerosas aplicaciones que tiene tanto el ML como el DL y sería interesante ampliar el campo de investigación a temas energéticos, financieros y médicos, muy importantes hoy en día.

Bibliografía

[1] R. L. Blanco, M. Á. Labrador-Espinosa, P. Mir, and M. Matarazzo, "Inteligencia artificial y machine learning en trastornos del movimiento," *MANUAL SEN DE*, 2021.

[2] C. de los proyectos Wikimedia. (2002) Prueba de turing - wikipedia, la enciclopedia libre. [Online]. Available: https://es.wikipedia.org/wiki/Prueba_de_Turing

[3] M. Sotaquirá. (2018) Tutorial: la regresión lineal en python codificando bits. [Online]. Available: https://www.codificandobits.com/blog/tutorial-regresion-lineal-en-python/

[4] Álvaro. (2018) Regresión lineal en python. [Online]. Available: https://machinelearningparatodos.com/regresion-lineal-en-python/

[5] F. Román. (2016) Árbol. [Online]. Available: https://estructurasite.wordpress.com/arbol/

[6] C. Hu, G. Jain, P. Zhang, C. Schmidt, P. Gomadam, and T. Gorka, "Data-driven method based on particle swarm optimization and k-nearest neighbor regression for estimating capacity of lithium-ion battery," *Applied Energy*, vol. 129, pp. 49–55, 2014.

[7] E. J. C. Suárez, "Tutorial sobre máquinas de vectores soporte (svm)," *Tutorial sobre Máquinas de Vectores Soporte (SVM)*, vol. 1, pp. 1–12, 2014.

[8] J. A. Camacho. (2020) Support vector regression (svr) jacobsoft. [Online]. Available: https://www.jacobsoft.com.mx/es_mx/support-vector-regression/

[9] J. Graw, W. Wood, and B. Phrampus, "Predicting global marine sediment density using the random forest regressor machine learning algorithm," *Journal of Geophysical Research: Solid Earth*, vol. 126, no. 1, p. e2020JB020135, 2021.

[10] A. Maisueche Cuadrado *et al.*, "Utilización del machine learning en la industria 4.0," 2019.

[11] Á. F. N. Cordero, M. T. R. Abrio, and M. J. R. Maqueda, "El hormigón: Historia, antecedentes en obras y factores identificativos de su resistencia," *Tecnología y desarrollo*, vol. 10, p. 13, 2012.

[12] P. Perles, *Hormigón armado*. Nobuko, 2003.

[13] C. Romea, "El hormigón: breve reseña histórica de un material milenario," *OmniaScience Monographs*, 2014.

[14] W. B. Chaabene, M. Flah, and M. L. Nehdi, "Machine learning prediction of mechanical properties of concrete: Critical review," *Construction and Building Materials*, vol. 260, p. 119889, 2020.

[15] A. Hernández-Díaz and M. García-Román, "Computing the refined compression field theory," *International Journal of Concrete Structures and Materials*, vol. 10, pp. 143–147, 2016.

[16] A. M. Hernández-Díaz, J. Pérez-Aracil, D. Casillas-Perez, E. Pereira, and S. Salcedo-Sanz, "Hybridizing machine learning with metaheuristics for preventing convergence failures in mechanical models based on compression field theories," *Applied Soft Computing*, vol. 130, p. 109654, 2022.

[17] J.-S. Chou, C.-F. Tsai, A.-D. Pham, and Y.-H. Lu, "Machine learning in concrete strength simulations: Multi-nation data analytics," *Construction and Building materials*, vol. 73, pp. 771–780, 2014.

[18] M.-C. Kang, D.-Y. Yoo, and R. Gupta, "Machine learning-based prediction for compressive and flexural strengths of steel fiber-reinforced concrete," *Construction and Building Materials*, vol. 266, p. 121117, 2021.

[19] F. J. Vecchio and M. P. Collins, "The modified compression-field theory for reinforced concrete elements subjected to shear." *ACI J.*, vol. 83, no. 2, pp. 219–231, 1986.

[20] A. Belarbi and T. T. Hsu, "Constitutive laws of concrete in tension and reinforcing bars stiffened by concrete," *Structural Journal*, vol. 91, no. 4, pp. 465–474, 1994.

[21] X.-B. D. Pang and T. T. Hsu, "Behavior of reinforced concrete membrane elements in shear," *Structural Journal*, vol. 92, no. 6, pp. 665–679, 1995.

[22] A. M. Hernández Díaz *et al.*, *Revisión de las teorías de campo de compresiones en hormigón estructural*. Universidad de Granada, 2013.

[23] M. Palermo, L. M. Gil-Martín, T. Trombetti, and E. Hernández-Montes, "In-plane shear behaviour of thin low reinforced concrete panels for earthquake re-construction," *Materials and structures*, vol. 46, pp. 841–856, 2013.

[24] R. M. España, A. M. Hernández-Díaz, J. M. Cecilia, and M. D. García-Román, "Evolutionary strategies as applied to shear strain effects in reinforced concrete beams," *Applied Soft Computing*, vol. 57, pp. 164–176, 2017.

[25] D. Mitchell and M. P. Collins, "Diagonal compression field theory-a rational model for structural concrete in pure torsion," in *Journal Proceedings*, vol. 71, no. 8, 1974, pp. 396–408.

[26] M. P. Collins, "Towards a rational theory for rc members in shear," *Journal of the Structural Division*, vol. 104, no. 4, pp. 649–666, 1978.

[27] F. J. Vecchio and M. P. Collins, "Predicting the response of reinforced concrete beams subjected to shear using modified compression field theory," *ACI Structural Journal*, vol. 85, no. 3, pp. 258–268, 1988.

[28] J. Pérez-Aracil, A. M. Hernández-Díaz, C. M. Marina, and S. Salcedo-Sanz, "Improving numerical methods for the steel yield strain calculation in reinforced concrete members with machine learning algorithms," *Expert Systems With Applications*, vol. 225, p. 119987, 2023.

[29] M.-Y. Cheng, J.-S. Chou, A. F. Roy, and Y.-W. Wu, "High-performance concrete compressive strength prediction using time-weighted evolutionary fuzzy support vector machines inference model," *Automation in Construction*, vol. 28, pp. 106–115, 2012.

[30] E. Slater, M. Moni, and M. S. Alam, "Predicting the shear strength of steel fiber reinforced concrete beams," *Construction and Building Materials*, vol. 26, no. 1, pp. 423–436, 2012.

[31] J.-S. Chou and C.-F. Tsai, "Concrete compressive strength analysis using a combined classification and regression technique," *Automation in Construction*, vol. 24, pp. 52–60, 2012.

[32] H. Nguyen, T. Vu, T. P. Vo, and H.-T. Thai, "Efficient machine learning models for prediction of concrete strengths," *Construction and Building Materials*, vol. 266, p. 120950, 2021.

[33] H.-B. Ly, T.-T. Le, H.-L. T. Vu, V. Q. Tran, L. M. Le, and B. T. Pham, "Computational hybrid machine learning based prediction of shear capacity for steel fiber reinforced concrete beams," *Sustainability*, vol. 12, no. 7, p. 2709, 2020.

[34] H.-G. Ni and J.-Z. Wang, "Prediction of compressive strength of concrete by neural networks," *Cement and Concrete Research*, vol. 30, no. 8, pp. 1245–1250, 2000.

[35] I. B. Topcu and M. Sarıdemir, "Prediction of compressive strength of concrete containing fly ash using artificial neural networks and fuzzy logic," *Computational Materials Science*, vol. 41, no. 3, pp. 305–311, 2008.

[36] B. A. Salami, S. M. Rahman, T. A. Oyehan, M. Maslehuddin, and S. U. Al Dulaijan, "Ensemble machine learning model for corrosion initiation time estimation of embedded steel reinforced self-compacting concrete," *Measurement*, vol. 165, p. 108141, 2020.

[37] H. Dabiri, A. Kheyroddin, and A. Faramarzi, "Predicting tensile strength of spliced and non-spliced steel bars using machine learning-and regression-based methods," *Construction and Building Materials*, vol. 325, p. 126835, 2022.

[38] C. Janiesch, P. Zschech, and K. Heinrich, "Machine learning and deep learning," *Electronic Markets*, vol. 31, no. 3, pp. 685–695, 2021.

[39] L. J. Sandoval Serrano *et al.*, "Algoritmos de aprendizaje automático para análisis y predicción de datos," *Revista Tecnológica; no. 11*, 2018.

[40] D. Hinestroza Ramírez, "El machine learning a través de los tiempos, y los aportes a la humanidad," 2018.

[41] A. M. Turing, *Computing machinery and intelligence*. Springer, 2009.

[42] A. Pinar Saygin, I. Cicekli, and V. Akman, "Turing test: 50 years later," *Minds and machines*, vol. 10, no. 4, pp. 463–518, 2000.

[43] M. M. Mijwel, "History of artificial intelligence yapay zekânın t arihi," *no. April*, vol. 2018, 2015.

[44] M. Minsky and S. A. Papert, *Perceptrons, Reissue of the 1988 Expanded Edition with a new foreword by Léon Bottou: An Introduction to Computational Geometry*. MIT press, 2017.

[45] C. C. Tappert, "Frank rosenblatt, the father of deep learning."

[46] H. P. Moravec, "The stanford cart and the cmu rover," *Proceedings of the IEEE*, vol. 71, no. 7, pp. 872–884, 1983.

[47] L. E. Peterson, "K-nearest neighbor," *Scholarpedia*, vol. 4, no. 2, p. 1883, 2009.

[48] T. Ellman, "Explanation-based learning: A survey of programs and perspectives," *ACM Computing Surveys (CSUR)*, vol. 21, no. 2, pp. 163–221, 1989.

[49] P. P. Shinde and S. Shah, "A review of machine learning and deep learning applications," in *2018 Fourth international conference on computing communication control and automation (ICCUBEA)*. IEEE, 2018, pp. 1–6.

[50] J. M. Bello de Haro, "Inteligencia artificial para la realizacion de procesos comunes mediante watson, nodejs y blue prism," 2018.

[51] S. R. Granter, A. H. Beck, and D. J. Papke Jr, "Alphago, deep learning, and the future of the human microscopist," *Archives of pathology & laboratory medicine*, vol. 141, no. 5, pp. 619–621, 2017.

[52] N. Sharma, R. Sharma, and N. Jindal, "Machine learning and deep learning applications-a vision," *Global Transitions Proceedings*, vol. 2, no. 1, pp. 24–28, 2021.

[53] C. Aracena, F. Villena, F. Arias, and J. Dunstan, "Aplicaciones de aprendizaje automático en salud," *Revista Médica Clínica Las Condes*, vol. 33, no. 6, pp. 568–575, 2022.

[54] D. Cauas, "Definición de las variables, enfoque y tipo de investigación," *Bogotá: biblioteca electrónica de la universidad Nacional de Colombia*, vol. 2, pp. 1–11, 2015.

[55] P. Morales, "Tipos de variables y sus implicaciones en el diseño de una investigación," *Madrid: Universidad Pontificia Comillas. Recuperado de http://web. upcomillas. es/personal/peter/investigacion/Variables. pdf (21/05/05)*, 2012.

[56] G. Ruiz Manosalva, "Modelo de análisis de datos utilizando técnicas de aprendizaje supervisado y no supervisado, para identificar patrones en la información generada por los pacientes, sometidos a juegos diseñados como un instrumento de apoyo terapéutico," 2019.

[57] H. C. Arteaga, "Técnicas de aprendizaje supervisado y no supervisado para el aprendizaje automatizado de computadoras," in *Memorias del primer Congreso Internacional de Ciencias Pedagógicas: Por una educación integral, participativa e incluyente.* Instituto Superior Tecnológico Bolivariano, 2015, pp. 549–564.

[58] D. Maulud and A. M. Abdulazeez, "A review on linear regression comprehensive in machine learning," *Journal of Applied Science and Technology Trends*, vol. 1, no. 4, pp. 140–147, 2020.

[59] A. Carrasquilla-Batista, A. Chacón-Rodríguez, K. Núñez-Montero, O. Gómez-Espinoza, J. Valverde, and M. Guerrero-Barrantes, "Regresión lineal simple y múltiple: aplicación en la predicción de variables naturales relacionadas con el crecimiento microalgal," *Revista Tecnología en Marcha*, vol. 29, pp. 33–45, 2016.

[60] P. M. Alvarez Barco *et al.*, "Predicción del precio del gas natural usando python mediante modelos de aprendizaje automático." Ph.D. dissertation, 2022.

[61] R. M. Granados, "Modelos de regresión lineal múltiple," *Granada, España: Departamento de Economía Aplicada, Universidad de Granada*, 2016.

[62] C. Bouza and A. Santiago, "La minería de datos: árboles de decisión y su aplicación en estudios médicos," *Modelación matemática de fenómenos del medio ambiente y la salud*, vol. 2, pp. 64–78, 2012.

[63] V. Berlanga, M. J. Rubio Hurtado, and R. Vilà Baños, "Cómo aplicar árboles de decisión en spss," *REIRE. Revista d'Innovació i Recerca en Educació, 2013, vol. 6, num. 1, p. 65-79*, 2013.

[64] R. E. B. Martínez, N. C. Ramírez, H. G. A. Mesa, I. R. Suárez, M. Trejo, P. P. León, and S. L. B. Morales, "Árboles de decisión como herramienta en el diagnóstico médico," *Revista médica de la Universidad Veracruzana*, vol. 9, no. 2, pp. 19–24, 2009.

[65] J. Ranstam and J. Cook, "Lasso regression," *Journal of British Surgery*, vol. 105, no. 10, pp. 1348–1348, 2018.

[66] M. Carrasco Carrasco, "Técnicas de regularización en regresión: Implementación y aplicaciones," 2016.

[67] M. B. Allasia, M. D. Branco, M. B. Quaglino *et al.*, "Regresión lasso bayesiana. ajuste de modelos lineales penalizados mediante la asignación de priores normales con mezcla de escala," 2016.

[68] M. Durbin, M. Wonders, M. Flaska, and A. T. Lintereur, "K-nearest neighbors regression for the discrimination of gamma rays and neutrons in organic scintillators," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 987, p. 164826, 2021.

[69] G. Morales España, J. Mora Flórez, and H. Vargas Torres, "Estrategia de regresión basada en el método de los k vecinos más cercanos para la estimación de la distancia de falla en sistemas radiales," *Revista Facultad de Ingeniería Universidad de Antioquia*, no. 45, pp. 100–108, 2008.

[70] L. Jiang, Z. Cai, D. Wang, and S. Jiang, "Survey of improving k-nearest-neighbor for classification," in *Fourth international conference on fuzzy systems and knowledge discovery (FSKD 2007)*, vol. 1. IEEE, 2007, pp. 679–683.

[71] Z. Ming, Y. Zhengbo, Z. Liukun, W. Huijie, and X. Xiaogang, "Energy consumption characteristics analysis of thermal power units based on s-svr," in *2013 International Conference on Materials for Renewable Energy and Environment*, vol. 3. IEEE, 2013, pp. 735–739.

[72] Z. El Mrabet, N. Sugunaraj, P. Ranganathan, and S. Abhyankar, "Random forest regressor-based approach for detecting fault location and duration in power systems," *Sensors*, vol. 22, no. 2, p. 458, 2022.

[73] A. Peerlinck, J. Sheppard, and J. Senecal, "Adaboost with neural networks for yield and protein prediction in precision agriculture," in *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2019, pp. 1–8.

[74] D. Tang, X. Yang, and X. Wang, "Improving the transferability of the crash prediction model using the tradaboost. r2 algorithm," *Accident Analysis & Prevention*, vol. 141, p. 105551, 2020.

[75] C. Seiffert, T. M. Khoshgoftaar, J. Van Hulse, and A. Napolitano, "Resampling or reweighting: A comparison of boosting implementations," in *2008 20th IEEE International Conference on Tools with Artificial Intelligence*, vol. 1. IEEE, 2008, pp. 445–451.

[76] D. I. Q. Yagual, C. C. Yagual, and I. C. Suárez, "Una revisión del aprendizaje profundo aplicado a la ciberseguridad," *Revista Científica y Tecnológica UPSE*, vol. 9, no. 1, pp. 57–65, 2022.

[77] H. Liu and B. Lang, "Machine learning and deep learning methods for intrusion detection systems: A survey," *applied sciences*, vol. 9, no. 20, p. 4396, 2019.

[78] R. Y. Choi, A. S. Coyner, J. Kalpathy-Cramer, M. F. Chiang, and J. P. Campbell, "Introduction to machine learning, neural networks, and deep learning," *Translational vision science & technology*, vol. 9, no. 2, pp. 14–14, 2020.

[79] E. M. Rojas, "Machine learning: análisis de lenguajes de programación y herramientas para desarrollo," *Revista Ibérica de Sistemas e Tecnologias de Informação*, no. E28, pp. 586–599, 2020.

[80] A. E. Madrid, S. M. Valenzuela-Ruiz, C. Batanero, and J. A. Garzón-Guerrero, "Interpretación del diagrama de caja por estudiantes universitarios de ciencias de la actividad física y deporte," *Educación matemática*, vol. 34, no. 3, pp. 275–300, 2022.

[81] H. Zhou, Z. Deng, Y. Xia, and M. Fu, "A new sampling method in particle filter based on pearson correlation coefficient," *Neurocomputing*, vol. 216, pp. 208–215, 2016.

[82] D. Thara, B. PremaSudha, and F. Xiong, "Auto-detection of epileptic seizure events using deep neural network with different feature scaling techniques," *Pattern Recognition Letters*, vol. 128, pp. 544–550, 2019.

[83] D. S. K. Karunasingha, "Root mean square error or mean absolute error? use their ratio as well," *Information Sciences*, vol. 585, pp. 609–629, 2022.

[84] T. Chai and R. R. Draxler, "Root mean square error (rmse) or mean absolute error (mae)," *Geoscientific model development discussions*, vol. 7, no. 1, pp. 1525–1534, 2014.

[85] H. V. Gupta and H. Kling, "On typical range, sensitivity, and normalization of mean squared error and nash-sutcliffe efficiency type metrics," *Water Resources Research*, vol. 47, no. 10, 2011.

[86] S. L. Posada and R. Rosero Noguera, "Comparación de modelos matemáticos: una aplicación en la evaluación de alimentos para animales," *Revista Colombiana de Ciencias Pecuarias*, vol. 20, no. 2, pp. 141–148, 2007.

Apéndice A

Herramientas y recursos

Las herramientas necesarias para la elaboración del proyecto han sido:

- PC compatible
- Sistema operativo Windows
- Entorno de desarrollo Visual Studio Code
- Procesador de textos LaTeX
- Lenguaje de procesamiento matemático Python

Apéndice B

Códigos de programación

En este apartado se adjuntan los códigos en lenguae Python utilizados en el desarrollo del trabajo, tanto como para el modelado de los lagoritmos, como para el procesamiento de la base de datos.

A continuación de los códigos de programación se adjuntan los hiperparámetros utilizados en cada algoritmo.

CÓDIGO PARA LAS CARACTERÍSTICAS DE LAS VARIABLES

```
import pandas as pd

archivo='lista1_con nombres.xlsx'

datos= pd.read_excel(archivo, header=1, sheet_name= 'Sheet1')

datos.info()
```

CÓDIGO PARA LA MATRIZ DE CORRELACIONES

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

archivo='lista1_con nombres.xlsx'

datos= pd.read_excel(archivo, header=1, sheet_name= 'Sheet1')

corr_df = datos.corr(method='pearson')

plt.figure(figsize=(20, 18))
sns.heatmap(corr_df, annot=True, cmap='YlGnBu')
plt.show()
```

CÓDIGO PARA VISUALIZAR LOS VALORES NULOS

```
import pandas as pd

archivo='lista1_con nombres.xlsx'

datos= pd.read_excel(archivo, header=1, sheet_name= 'Sheet1')

datos.isna().sum().sort_values()
```

CÓDIGO PARA LOS DIAGRAMAS DE CAJAS Y BIGOTES

```
#GRÁFICO DE CAJAS

import pandas as pd
import matplotlib.pyplot as plt

# Leer el archivo Excel
archivo='lista1_con nombres.xlsx'

datos= pd.read_excel(archivo, header=1, sheet_name= 'Sheet1')
```

```
# Obtener las columnas de interés
columnas = datos.columns[:22]  # Las primeras 22 columnas (variables)
del DataFrame

# Generar y mostrar el gráfico de cajas para cada variable
for columna in columnas:
    plt.figure(figsize=(6, 4))  # Tamaño de la figura
    plt.boxplot(datos[columna])
    plt.title('{}'.format(columna))  # Título del gráfico
    plt.show()
```

CÓDIGO PARA LOS HISTOGRAMAS

```
import matplotlib.pyplot as plt
from matplotlib import style
import matplotlib.ticker as ticker
import seaborn as sns
import statsmodels.api as sm
import pandas as pd

archivo='lista1_con nombres.xlsx'

datos= pd.read_excel(archivo, header=1, sheet_name= 'Sheet1')

datos.head(2000)
datos.shape
datos.isna().sum().sort_values()
datos.select_dtypes(include=['float64', 'int']).describe()
fig, axes = plt.subplots(nrows=11, ncols=2, figsize=(18, 70))
axes = axes.flat
columnas_numeric = datos.select_dtypes(include=['float64',
'int']).columns

for i, colum in enumerate(columnas_numeric):
    sns.histplot(
        data     = datos,
        x        = colum,
        stat     = "count",
        kde      = True,
        line_kws = {'linewidth': 2},
        alpha    = 0.3,
        ax       = axes[i]
    )
    axes[i].set_title(colum, fontsize = 18, fontweight = "bold")
    axes[i].tick_params(labelsize = 14)
    axes[i].set_xlabel("")
    axes[i].set_ylabel("Cantidad")
```

```
fig.tight_layout()
plt.subplots_adjust(top = 0.9)
```

CÓDIGO PARA EL ALGORITMO SVR

```
from sklearn.model_selection import train_test_split
from sklearn.svm import SVR
from sklearn.metrics import
mean_absolute_error,mean_absolute_percentage_error,mean_squared_error,r
2_score

import pandas as pd
from sklearn.preprocessing import StandardScaler,MinMaxScaler
from sklearn.model_selection import train_test_split
import numpy as np

archivo='lista1_con nombres.xlsx'

datos= pd.read_excel(archivo, header=1, sheet_name= 'Sheet1')

# Seleccionar las variables de entrada y salida
X1 = datos.iloc[:, :12]  # 12 primeras columnas
y1 = datos.iloc[:, 16]  # la columna nº n-1...


# Dividir los datos en conjunto de entrenamiento y prueba
X_train1, X_test1, y_train1, y_test1 = train_test_split(X1, y1,
test_size=0.25, random_state=42, shuffle=True)

#Normalización con StandardScaler

scaler = StandardScaler()

scaler = scaler.fit(X_train1)
X_train_normalizado = scaler.transform(X_train1)
X_test_normalizado = scaler.transform(X_test1)

scaler1 = StandardScaler()

scaler1 = scaler1.fit(np.asarray(y_train1).reshape(-1, 1))
y_train_normalizado = scaler1.transform(np.asarray(y_train1).reshape(-
1, 1))
y_test_normalizado = scaler1.transform(np.asarray(y_test1).reshape(-1,
1))

model1=SVR(C = 10, coef0 = 0.5, degree = 2, epsilon = 0.1, gamma =
'auto', kernel = 'poly', tol = 0.1)
```

```
model1.fit(X_train_normalizado, y_train_normalizado.ravel())

ypred1=model1.predict(X_test_normalizado)

y_test_restaurado=scaler1.inverse_transform(np.asarray(y_test_normaliza
do).reshape(-1, 1))
ypred_restaurado=scaler1.inverse_transform(np.asarray(ypred1).reshape(-
1, 1))

# Evaluar el modelo con los datos de prueba

r2=r2_score(y_test_normalizado, ypred1)
mae=mean_absolute_error(y_test_restaurado, ypred_restaurado)
rmse=mean_squared_error(y_test_restaurado, ypred_restaurado,
squared=False)
mse=mean_squared_error(y_test_restaurado, ypred_restaurado,
squared=True)
nmse = mse / np.var(y_test_restaurado)

print("Puntaje del modelo mean abs.:", mae)
print("Puntaje del modelo mse:", rmse)
print("Puntaje del modelo R2:", r2)
print('nmse:', nmse)
```

CÓDIGO BÚSQUEDA HIPERPARÁMETROS SVR

```
# Elección de los parámetros óptimos utilizando el método gridsearch

from sklearn.model_selection import train_test_split, GridSearchCV,
RandomizedSearchCV
from sklearn.svm import SVR

import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler,MinMaxScaler

archivo='lista1_con nombres.xlsx'

datos= pd.read_excel(archivo, header=1, sheet_name= 'Sheet1')

# Seleccionar las variables de entrada y salida
X1 = datos.iloc[:, :12]  # 12 primeras columnas
y1 = datos.iloc[:, 21]  # la columna nº n-1...

# Dividir los datos en conjunto de entrenamiento y prueba
X_train1, X_test1, y_train1, y_test1 = train_test_split(X1, y1,
test_size=0.25, random_state=42, shuffle=True)

# Normalización con StandardScaler
```

```
scaler = StandardScaler()

scaler = scaler.fit(X_train1)
X_train_normalizado = scaler.transform(X_train1)
X_test_normalizado = scaler.transform(X_test1)


scaler1 = StandardScaler()

scaler1 = scaler1.fit(np.asarray(y_train1).reshape(-1, 1))
y_train_normalizado = scaler1.transform(np.asarray(y_train1).reshape(-
1, 1))
y_test_normalizado = scaler1.transform(np.asarray(y_test1).reshape(-1,
1))


# Modelos

SVRegressor = SVR()

# Selección de parámetros

param_grid_SVRegressor = {'kernel': ['linear', 'poly', 'rbf',
'sigmoid'],
    'degree': [2, 3, 4],
    'gamma': ['scale', 'auto'],
    'coef0': [0.0, 0.5, 1.0],
    'tol': [0.001, 0.01, 0.1],
    'C': [0.1, 1, 10],
    'epsilon': [0.1, 0.2, 0.3]
    }

# Búsqueda de los parámetros óptimos

grid_search_SVRegressor = GridSearchCV(estimator=SVRegressor,
param_grid=param_grid_SVRegressor, cv=5)

grid_search_SVRegressor.fit(X_train_normalizado, y_train_normalizado)

best_params_SVRegressor = grid_search_SVRegressor.best_params_
best_score_SVRegressor = grid_search_SVRegressor.best_score_


print('Los parámetros óptimos del modelo SVR son:',
best_params_SVRegressor)
print('El puntaje del modelo con los parámetros óptimos del modelo SVR
es:', best_score_SVRegressor)
```

CÓDIGO PARA LA IMPLEMENTACIÓN DEL RESTO DE ALGORTIMOS – UNA SALIDA

```
from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.linear_model import LinearRegression,Lasso
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor,NearestNeighbors
from sklearn.ensemble import RandomForestRegressor, AdaBoostRegressor,
GradientBoostingRegressor
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import
mean_absolute_error,mean_absolute_percentage_error,mean_squared_error,r
2_score
import numpy as np

import pandas as pd
from sklearn.preprocessing import StandardScaler,MinMaxScaler

archivo='lista1_con nombres.xlsx'

datos= pd.read_excel(archivo, header=1, sheet_name= 'Sheet1')

# Seleccionar las variables de entrada y salida
X = datos.iloc[:, :12]  # 12 primeras columnas
y = datos.iloc[:, 12]  # la columna nº n-1...

# Dividir los datos en conjunto de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.25, random_state=42, shuffle=True)

# Normalización con StandardScaler
scaler = StandardScaler()

scaler = scaler.fit(X_train)
X_train_normalizado = scaler.transform(X_train)
X_test_normalizado = scaler.transform(X_test)

#para hacer el modelo multisalidas hay que normalizar las columnas de
salida con y_train1

#Crear un modelo de regresión lineal y entrenarlo con los datos de
entrenamiento
model = LinearRegression(copy_X= True, fit_intercept= True, positive=
False)
#model = DecisionTreeRegressor(ccp_alpha = 0.0, criterion =
'absolute_error', max_depth = 10, max_leaf_nodes = None,
min_impurity_decrease = 0.0, min_samples_leaf =5, min_samples_split =
2, min_weight_fraction_leaf = 0.0, random_state = None, splitter =
'best')
```

```
#model = Lasso(alpha = 0.000000001, copy_X = True, fit_intercept =
True, max_iter = 1000, positive = False, precompute = True,
random_state = None, selection = 'cyclic', tol = 0.00001, warm_start =
True)
#model = KNeighborsRegressor(algorithm = 'auto', leaf_size = 10,
n_neighbors = 13, p = 3, weights = 'distance')
#model = NearestNeighbors(n_neighbors=1)
#model = RandomForestRegressor(warm_start = False, oob_score = False,
n_estimators = 100, min_samples_split = 5, min_samples_leaf = 6,
max_features = None, criterion = 'absolute_error')
#model = AdaBoostRegressor(learning_rate = 0.001, loss = 'exponential',
n_estimators = 700)
#model = GradientBoostingRegressor(warm_start = True, subsample = 0.5,
n_estimators = 500, min_samples_split = 5, min_samples_leaf = 6, loss =
'huber', learning_rate = 0.1, criterion = 'friedman_mse')
# Lasso, Polynomial, KNN ,SVR, Random Forest y probar normalizacion y
estandarizacion, neural network keras y multioutput

model.fit(X_train_normalizado, y_train)

ypred=model.predict(X_test_normalizado)


# Evaluar el modelo con los datos de prueba

r2=r2_score(y_test, ypred)
mae=mean_absolute_error(y_test, ypred)
rmse=mean_squared_error(y_test, ypred, squared=False)
mse=mean_squared_error(y_test, ypred, squared=True)
nmse = mse / np.var(y_test)

print("Puntaje del modelo mean abs.:", mae)
print("Puntaje del modelo rmse:", rmse)
print("Puntaje del modelo R2:", r2)
print('nmse:', nmse)
```

CÓDIGO BÚSQUEDA HIPERPARÁMETROS RESTO DE ALGORITMOS – UNA SALIDA

```
# Elección de los parámetros óptimos utilizando el método gridsearch

from sklearn.model_selection import train_test_split, GridSearchCV,
RandomizedSearchCV
from sklearn.linear_model import LinearRegression,Lasso
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor,NearestNeighbors
from sklearn.ensemble import RandomForestRegressor, AdaBoostRegressor,
GradientBoostingRegressor
from sklearn.preprocessing import PolynomialFeatures
```

```
import pandas as pd
from sklearn.preprocessing import StandardScaler,MinMaxScaler

archivo='lista1_con nombres.xlsx'

datos= pd.read_excel(archivo, header=1, sheet_name= 'Sheet1')

# Seleccionar las variables de entrada y salida
X = datos.iloc[:, :12]  # 12 primeras columnas
y = datos.iloc[:, 12]  # la columna nº n-1...

# Dividir los datos en conjunto de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.25, random_state=42, shuffle=True)

# Normalización con StandardScaler
scaler = StandardScaler()

scaler = scaler.fit(X_train)
X_train_normalizado = scaler.transform(X_train)
X_test_normalizado = scaler.transform(X_test)

# Modelos
LinearRegressor = LinearRegression()
TreeRegressor = DecisionTreeRegressor()
Lso = Lasso()
KNRegressor = KNeighborsRegressor()
RandomForest = RandomForestRegressor()
AdaBoost = AdaBoostRegressor()
GradientBoost = GradientBoostingRegressor()

# Selección de parámetros
param_grid_LinearRegressor = {'fit_intercept': [True, False],
    'positive': [True, False],
    'copy_X': [True, False]}
'''param_grid_TreeRegressor = {'criterion': ['squared_error',
'friedman_mse', 'absolute_error', 'poisson'],
    'splitter' :['best', 'random'],
    'max_depth' :[None, 1, 5, 10, 100, 1000],
    'min_samples_split': [ 2, 5, 10, 100],
    'min_samples_leaf' : [1, 2, 5, 10],
    #'min_weight_fraction_leaf': [0.0, 10],
    #'max_features': [None, 'auto', 'sqrt', 'log2'],
    'random_state':[None, 1, 42, 123],
    'max_leaf_nodes':[None,  5, 10],
    #'min_impurity_decrease': [0.0, 100],
    #'ccp_alpha': [0.0, 100]
    }'''
```

```
'''param_grid_Lso = {'alpha': [0.001, 0.01, 0.1, 1, 10, 100],
    'fit_intercept': [True, False],
    'precompute': [True, False],
    'copy_X': [True, False],
    'max_iter': [100, 500, 1000, 5000],
    'tol': [0.00001, 0.0001, 0.001, 0.01, 0.1],
    'warm_start': [True, False],
    'positive': [True, False],
    'random_state': [None, 1, 42, 123],
    'selection': ['cyclic', 'random']}'''
'''param_grid_KNRegressor = {'n_neighbors': [3, 5, 7, 9, 11, 13, 15],
    'weights': ['uniform', 'distance'],
    'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
    'leaf_size': [10, 20, 30, 40, 50],
    'p': [1, 2, 3],
    #'metric': ['minkowski', 'precomputed']
    }'''
'''param_grid_RandomForest = {'n_estimators': [100, 300, 500],
    'criterion': ['squared_error', 'friedman_mse', 'absolute_error',
'poisson'],
    #'max_depth': [None, 5, 10, 20],
    'min_samples_split': [2, 5, 10, 15],
    'min_samples_leaf': [1, 2, 4, 6],
    #'min_weight_fraction_leaf': [0.0, 0.1, 0.2],
    'max_features': ['sqrt', 'log2', None],
    #'max_leaf_nodes': [None, 5, 10, 20],
    #'min_impurity_decrease': [0.0, 0.1, 0.2],
    #'bootstrap': [True, False],
    'oob_score': [True, False],
    #'n_jobs': [None, 1, 2, 4],
    #'random_state': [None, 1, 42, 123],
    #'verbose': [0, 1, 2],
    'warm_start': [True, False],
    #'ccp_alpha': [0.0, 0.1],
    #'max_samples': [None, 1,100]
    }'''
'''param_grid_AdaBoost = {'n_estimators': [50, 100, 200, 500, 700],
'learning_rate': [0.01, 0.1, 1.0, 0.001, 0.0001],
'loss': ['linear', 'square', 'exponential']
}'''
'''param_grid_GradientBoost = {'loss': ['squared_error',
'absolute_error', 'huber', 'quantile'],
'learning_rate': [0.1, 0.01, 1.0],
'n_estimators': [100, 200, 500],
'subsample': [0.5, 1.0],
'criterion': ['friedman_mse', 'squared_error'],
'min_samples_split': [2, 5, 10, 15],
'min_samples_leaf': [1, 2, 4, 6],
'warm_start': [True, False]
```

```
#'max_depth': [3, 5, None],
#'max_features': ['auto', 'sqrt', 'log2', None],
#'ccp_alpha': [0.9, 0.5],
#'random_state': [42]
}'''


# Búsqueda de los parámetros óptimos
grid_search_LinearRegressor = GridSearchCV(estimator=LinearRegressor,
param_grid=param_grid_LinearRegressor, cv=5)
#grid_search_TreeRegressor = RandomizedSearchCV(n_iter=2000,
estimator=TreeRegressor, param_distributions=param_grid_TreeRegressor,
cv=5)
#grid_search_Lso = GridSearchCV(estimator=Lso,
param_grid=param_grid_Lso, cv=5)
#grid_search_KNRegressor = GridSearchCV(estimator=KNRegressor,
param_grid=param_grid_KNRegressor, cv=5)
#grid_search_RandomForest = RandomizedSearchCV(n_iter=800,
estimator=RandomForest, param_distributions=param_grid_RandomForest,
cv=5)
#grid_search_AdaBoost = GridSearchCV(estimator=AdaBoost,
param_grid=param_grid_AdaBoost, cv=5)
#grid_search_GradientBoost = RandomizedSearchCV(n_iter=1600,
estimator=GradientBoost, param_distributions=param_grid_GradientBoost,
cv=5)

grid_search_LinearRegressor.fit(X_train_normalizado, y_train)
#grid_search_TreeRegressor.fit(X_train_normalizado, y_train)
#grid_search_Lso.fit(X_train_normalizado, y_train)
#grid_search_KNRegressor.fit(X_train_normalizado, y_train)
#grid_search_RandomForest.fit(X_train_normalizado, y_train)
#grid_search_AdaBoost.fit(X_train_normalizado, y_train)
#grid_search_GradientBoost.fit(X_train_normalizado, y_train)

best_params_LinearRegressor = grid_search_LinearRegressor.best_params_
best_score_LinearRegressor = grid_search_LinearRegressor.best_score_

#best_params_TreeRegressor = grid_search_TreeRegressor.best_params_
#best_score_TreeRegressor = grid_search_TreeRegressor.best_score_

#best_params_Lso = grid_search_Lso.best_params_
#best_score_Lso = grid_search_Lso.best_score_

#best_params_KNRegressor = grid_search_KNRegressor.best_params_
#best_score_KNRegressor = grid_search_KNRegressor.best_score_

#best_params_RandomForest = grid_search_RandomForest.best_params_
#best_score_RandomForest = grid_search_RandomForest.best_score_
```

```
#best_params_AdaBoost = grid_search_AdaBoost.best_params_
#best_score_AdaBoost = grid_search_AdaBoost.best_score_

#best_params_GradientBoost = grid_search_GradientBoost.best_params_
#best_score_GradientBoost = grid_search_GradientBoost.best_score_


print('Los parámetros óptimos del modelo LinearRegression son:',
best_params_LinearRegressor)
print('El puntaje del modelo con los parámetros óptimos del modelo
LinearRegression es:', best_score_LinearRegressor)

#print('Los parámetros óptimos del modelo DecisionTreeRegressor son:',
best_params_TreeRegressor)
#print('El puntaje del modelo con los parámetros óptimos del modelo
DecisionTreeRegressor es:', best_score_TreeRegressor)

#print('Los parámetros óptimos del modelo Lasso son:', best_params_Lso)
#print('El puntaje del modelo con los parámetros óptimos del modelo
Lasso es:', best_score_Lso)

#print('Los parámetros óptimos del modelo KNeighborsRegressor son:',
best_params_KNRegressor)
#print('El puntaje del modelo con los parámetros óptimos del modelo
KNeighborsRegressor es:', best_score_KNRegressor)

#print('Los parámetros óptimos del modelo RandomForestRegressor son:',
best_params_RandomForest)
#print('El puntaje del modelo con los parámetros óptimos del modelo
RandomForestRegressor es:', best_score_RandomForest)

#print('Los parámetros óptimos del modelo AdaBoost son:',
best_params_AdaBoost)
#print('El puntaje del modelo con los parámetros óptimos del modelo
AdaBoost es:', best_score_AdaBoost)

#print('Los parámetros óptimos del modelo GradientBoost son:',
best_params_GradientBoost)
#print('El puntaje del modelo con los parámetros óptimos del modelo
GradientBoost es:', best_score_GradientBoost)
```

CÓDIGO RED NEURONAL – UNA SALIDA

```
#PROGRAMACIÓN DE UNA RED NEURONAL KERAS

import pandas as pd
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers.core import Dense
```

```
import tensorflow as tf
from sklearn.metrics import
mean_absolute_error,mean_absolute_percentage_error,mean_squared_error,r
2_score
from sklearn.preprocessing import StandardScaler,MinMaxScaler
import matplotlib.pyplot as plt

archivo='lista1_con nombres.xlsx'

datos= pd.read_excel(archivo, header=1, sheet_name= 'Sheet1')

# Seleccionar las variables de entrada y salida
X = datos.iloc[:, :12]  # 12 primeras columnas
y = datos.iloc[:, 12]  # la columna nº n-1...

# Dividir los datos en conjunto de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.25, random_state=42, shuffle=True)

X_train1, X_val, y_train1, y_val = train_test_split(X_train, y_train,
test_size=0.25, random_state=42, shuffle=True)

# Normalización con StandardScaler
scaler = MinMaxScaler()

scaler = scaler.fit(X_train)
X_train_normalizado = scaler.transform(X_train)
X_test_normalizado = scaler.transform(X_test)

# Crear el modelo de la red neuronal
model = Sequential()
model.add(Dense(32, input_dim=12, activation='relu'))
model.add(Dense(4, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

# Compilar el modelo
model.compile(loss='binary_crossentropy',
optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3))

callback = tf.keras.callbacks.EarlyStopping(monitor='val_loss',
patience=50)
history = model.fit(verbose =1,
                        x = X_train_normalizado,
                        y = y_train,
                        batch_size=32,
                        epochs=4000,
                        validation_split=0.1,
                        shuffle=True,
                        validation_steps=None,
```

```
                        validation_freq=1,
                        workers=4,
                        use_multiprocessing=False,
                        callbacks=[callback])

plt.plot(history.history['loss'])

plt.plot(history.history['val_loss'])

plt.title('loss')

plt.ylabel('loss')

plt.xlabel('epoch')

plt.legend(['train','test'], loc='upper left')

plt.show()

import numpy as np
ypred=model.predict(X_test_normalizado)

r2=r2_score(y_test, ypred)
mae=mean_absolute_error(y_test, ypred)
rmse=mean_squared_error(y_test, ypred, squared=False)
mse=mean_squared_error(y_test, ypred, squared=True)
nmse = mse / np.var(y_test)

print("Puntaje del modelo mean abs.:", mae)
print("Puntaje del modelo rmse:", rmse)
print("Puntaje del modelo R2:", r2)
print('nmse:', nmse)
```

CÓDIGO PARA LA ELECCIÓN DEL MODELO DE RED NEURONAL – UNA SALIDA

```
#ELECCIÓN MODELO RED NEURONAL

import pandas as pd
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers.core import Dense
import tensorflow as tf
from sklearn.metrics import
mean_absolute_error,mean_absolute_percentage_error,mean_squared_error,r
2_score
from sklearn.preprocessing import StandardScaler,MinMaxScaler
import matplotlib.pyplot as plt
```

```
archivo='lista1_con nombres.xlsx'

datos= pd.read_excel(archivo, header=1, sheet_name= 'Sheet1')

# Seleccionar las variables de entrada y salida
X = datos.iloc[:, :12]  # 12 primeras columnas
y = datos.iloc[:, 16]  # la columna nº n-1...

# Dividir los datos en conjunto de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.25, random_state=42, shuffle=True)

X_train1, X_val, y_train1, y_val = train_test_split(X_train, y_train,
test_size=0.25, random_state=42, shuffle=True)

# Normalización con StandardScaler
scaler = MinMaxScaler()

scaler = scaler.fit(X_train1)
X_train_normalizado1 = scaler.transform(X_train1)
X_val_normalizado = scaler.transform(X_val)

# Crear el modelo de la red neuronal
model = Sequential()
model.add(Dense(32, input_dim=12, activation='relu'))
model.add(Dense(4, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

# Compilar el modelo
model.compile(loss='mean_squared_error',
optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3))

callback = tf.keras.callbacks.EarlyStopping(monitor='val_loss',
patience=50)
history = model.fit(verbose =1,
                        x = X_train_normalizado1,
                        y = y_train1,
                        batch_size=32,
                        epochs=4000,
                        validation_split=0.1,
                        shuffle=True,
                        validation_steps=None,
                        validation_freq=1,
                        workers=4,
                        use_multiprocessing=False,
                        callbacks=[callback])

plt.plot(history.history['loss'])
```

```
plt.plot(history.history['val_loss'])

plt.title('loss')

plt.ylabel('loss')

plt.xlabel('epoch')

plt.legend(['train','test'], loc='upper left')

plt.show()

import numpy as np
ypred=model.predict(X_val_normalizado)

r2=r2_score(y_val, ypred)
mae=mean_absolute_error(y_val, ypred)
rmse=mean_squared_error(y_val, ypred, squared=False)
mse=mean_squared_error(y_val, ypred, squared=True)
nmse = mse / np.var(y_val)

print("Puntaje del modelo mean abs.:", mae)
print("Puntaje del modelo rmse:", rmse)
print("Puntaje del modelo R2:", r2)
print('nmse:', nmse)
```

CÓDIGO IMPLEMENTACIÓN ALGORITMOS – MULTISALIDA

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression,Lasso
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor,NearestNeighbors
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import PolynomialFeatures
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import
mean_absolute_error,mean_absolute_percentage_error,mean_squared_error,r
2_score
import numpy as np

import pandas as pd
from sklearn.preprocessing import StandardScaler,MinMaxScaler

archivo='lista1_con nombres.xlsx'

datos= pd.read_excel(archivo, header=1, sheet_name= 'Sheet1')
```

```
# Seleccionar las variables de entrada y salida
X = datos.iloc[:, :12]  # 12 primeras columnas
y = datos.iloc[:, 12:]  # la columna nº n-1...

# Dividir los datos en conjunto de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.25, random_state=42, shuffle=True)

#Normalización con StandardScaler
scaler = StandardScaler()
scaler1=StandardScaler()

scaler = scaler.fit(X_train)
X_train_normalizado = scaler.transform(X_train)
X_test_normalizado = scaler.transform(X_test)

scaler1=scaler1.fit(y_train)
y_train_normalizado = scaler1.transform(y_train)
y_test_normalizado = scaler1.transform(y_test)

#Crear un modelo de regresión lineal y entrenarlo con los datos de
entrenamiento
model = LinearRegression(copy_X = True, fit_intercept = False, positive
= False)
#model = DecisionTreeRegressor(splitter = 'best', random_state = 123,
min_samples_split = 10, min_samples_leaf = 10, max_leaf_nodes = None,
max_depth = None, criterion = 'absolute_error')
#model=Lasso(alpha = 0.001, copy_X = True, fit_intercept = False,
max_iter = 100, positive = False, precompute = True, random_state =
None, selection = 'cyclic', tol = 0.01, warm_start = True) #alpha=0.001
#model=KNeighborsRegressor(algorithm = 'auto', leaf_size = 10,
n_neighbors = 11, p = 2, weights = 'distance')
#model=RandomForestRegressor(warm_start = True, oob_score = True,
n_estimators = 500, min_samples_split = 2, min_samples_leaf = 4,
max_features = None, criterion = 'squared_error')


model.fit(X_train_normalizado, y_train_normalizado)

ypred=model.predict(X_test_normalizado)

y_test_restaurado=scaler1.inverse_transform(y_test_normalizado)
ypred_restaurado=scaler1.inverse_transform(ypred)

# Evaluar el modelo con los datos de prueba

#cambiar los números para mostrar cada salida
r2_1=r2_score(y_test_restaurado[:, 2], ypred_restaurado[:, 2])
```

```
mae_1=mean_absolute_error(y_test_restaurado[:, 2], ypred_restaurado[:,
2])
rmse_1=mean_squared_error(y_test_restaurado[:, 2], ypred_restaurado[:,
2], squared=False)
mse_1=mean_squared_error(y_test_restaurado[:, 2], ypred_restaurado[:,
2], squared=True)
nmse_1=mse_1 / np.var(y_test_restaurado[:, 2])

print("Puntaje del modelo mean abs.:", mae_1)
print("Puntaje del modelo rmse:", rmse_1)
print("Puntaje del modelo R2:", r2_1)
print('nmse:', nmse_1)
```

CÓDIGO BÚSQUEDA HIPERPARÁMETROS ALGORITMOS – MULTISALIDA

```
# Elección de los parámetros óptimos utilizando el método gridsearch

from sklearn.model_selection import train_test_split, GridSearchCV,
RandomizedSearchCV
from sklearn.linear_model import LinearRegression,Lasso
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor,NearestNeighbors
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestRegressor, AdaBoostRegressor,
GradientBoostingRegressor
from sklearn.preprocessing import PolynomialFeatures
from sklearn.neural_network import MLPRegressor

import pandas as pd
from sklearn.preprocessing import StandardScaler,MinMaxScaler

archivo='lista1_con nombres.xlsx'

datos= pd.read_excel(archivo, header=1, sheet_name= 'Sheet1')

# Seleccionar las variables de entrada y salida
X = datos.iloc[:, :12]  # 12 primeras columnas
y = datos.iloc[:, 12:]  # la columna nº n-1...

# Dividir los datos en conjunto de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.25, random_state=42, shuffle=True)

# Normalización con StandardScaler
'''scaler = StandardScaler()
scaler1=StandardScaler()

scaler = scaler.fit(X_train)
X_train_normalizado = scaler.transform(X_train)
```

```
X_test_normalizado = scaler.transform(X_test)

scaler1=scaler1.fit(y_train)
y_train_normalizado = scaler1.transform(y_train)
y_test_normalizado = scaler1.transform(y_test)'''

# Modelos
LinearRegressor = LinearRegression()
TreeRegressor = DecisionTreeRegressor()
Lso = Lasso()
KNRegressor = KNeighborsRegressor()
RandomForest = RandomForestRegressor()

# Selección de parámetros
param_grid_LinearRegressor = {'fit_intercept': [True, False],
    'positive': [True, False],
    'copy_X': [True, False]}
'''param_grid_TreeRegressor = {'criterion': ['squared_error',
'friedman_mse', 'absolute_error', 'poisson'],
    'splitter' :['best', 'random'],
    'max_depth' :[None, 1, 5, 10, 100, 1000],
    'min_samples_split': [ 2, 5, 10, 100],
    'min_samples_leaf' : [1, 2, 5, 10],
    #'min_weight_fraction_leaf': [0.0, 10],
    #'max_features': [None, 'auto', 'sqrt', 'log2'],
    'random_state':[None, 1, 42, 123],
    'max_leaf_nodes':[None,  5, 10],
    #'min_impurity_decrease': [0.0, 100],
    #'ccp_alpha': [0.0, 100]
    }'''
'''param_grid_Lso = {'alpha': [0.001, 0.01, 0.1, 1, 10, 100],
    'fit_intercept': [True, False],
    'precompute': [True, False],
    'copy_X': [True, False],
    'max_iter': [100, 500, 1000, 5000],
    'tol': [0.00001, 0.0001, 0.001, 0.01, 0.1],
    'warm_start': [True, False],
    'positive': [True, False],
    'random_state': [None, 1, 42, 123],
    'selection': ['cyclic', 'random']}'''
'''param_grid_KNRegressor = {'n_neighbors': [3, 5, 7, 9, 11, 13, 15],
    'weights': ['uniform', 'distance'],
    'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
    'leaf_size': [10, 20, 30, 40, 50],
    'p': [1, 2, 3],
    #'metric': ['minkowski', 'precomputed']
    }'''
'''param_grid_RandomForest = {'n_estimators': [100, 300, 500],
```

```
    'criterion': ['squared_error', 'friedman_mse', 'absolute_error',
'poisson'],
    #'max_depth': [None, 5, 10, 20],
    'min_samples_split': [2, 5, 10, 15],
    'min_samples_leaf': [1, 2, 4, 6],
    #'min_weight_fraction_leaf': [0.0, 0.1, 0.2],
    'max_features': ['sqrt', 'log2', None],
    #'max_leaf_nodes': [None, 5, 10, 20],
    #'min_impurity_decrease': [0.0, 0.1, 0.2],
    #'bootstrap': [True, False],
    'oob_score': [True, False],
    #'n_jobs': [None, 1, 2, 4],
    #'random_state': [None, 1, 42, 123],
    #'verbose': [0, 1, 2],
    'warm_start': [True, False],
    #'ccp_alpha': [0.0, 0.1],
    #'max_samples': [None, 1,100]
    }'''


# Búsqueda de los parámetros óptimos
grid_search_LinearRegressor = GridSearchCV(estimator=LinearRegressor,
param_grid=param_grid_LinearRegressor, cv=5)
#grid_search_TreeRegressor = RandomizedSearchCV(n_iter=2000,
estimator=TreeRegressor, param_distributions=param_grid_TreeRegressor,
cv=5)
#grid_search_Lso = GridSearchCV(estimator=Lso,
param_grid=param_grid_Lso, cv=5)
#grid_search_KNRegressor = GridSearchCV(estimator=KNRegressor,
param_grid=param_grid_KNRegressor, cv=5)
#grid_search_RandomForest = RandomizedSearchCV(n_iter=800,
estimator=RandomForest, param_distributions=param_grid_RandomForest,
cv=5)

grid_search_LinearRegressor.fit(X_train_normalizado,
y_train_normalizado)
#grid_search_TreeRegressor.fit(X_train_normalizado,
y_train_normalizado)
#grid_search_Lso.fit(X_train_normalizado, y_train_normalizado)
#grid_search_KNRegressor.fit(X_train_normalizado, y_train_normalizado)
#grid_search_RandomForest.fit(X_train_normalizado, y_train_normalizado)


best_params_LinearRegressor = grid_search_LinearRegressor.best_params_
best_score_LinearRegressor = grid_search_LinearRegressor.best_score_

#best_params_TreeRegressor = grid_search_TreeRegressor.best_params_
#best_score_TreeRegressor = grid_search_TreeRegressor.best_score_
```

```
#best_params_Lso = grid_search_Lso.best_params_
#best_score_Lso = grid_search_Lso.best_score_

#best_params_KNRegressor = grid_search_KNRegressor.best_params_
#best_score_KNRegressor = grid_search_KNRegressor.best_score_

#best_params_RandomForest = grid_search_RandomForest.best_params_
#best_score_RandomForest = grid_search_RandomForest.best_score_


print('Los parámetros óptimos del modelo LinearRegression son:',
best_params_LinearRegressor)
print('El puntaje del modelo con los parámetros óptimos del modelo
LinearRegression es:', best_score_LinearRegressor)

#print('Los parámetros óptimos del modelo DecisionTreeRegressor son:',
best_params_TreeRegressor)
#print('El puntaje del modelo con los parámetros óptimos del modelo
DecisionTreeRegressor es:', best_score_TreeRegressor)

#print('Los parámetros óptimos del modelo Lasso son:', best_params_Lso)
#print('El puntaje del modelo con los parámetros óptimos del modelo
Lasso es:', best_score_Lso)

#print('Los parámetros óptimos del modelo KNeighborsRegressor son:',
best_params_KNRegressor)
#print('El puntaje del modelo con los parámetros óptimos del modelo
KNeighborsRegressor es:', best_score_KNRegressor)

#print('Los parámetros óptimos del modelo RandomForestRegressor son:',
best_params_RandomForest)
#print('El puntaje del modelo con los parámetros óptimos del modelo
RandomForestRegressor es:', best_score_RandomForest)
```

CÓDIGO RED NEURONAL – MULTISALIDA

```
#PROGRAMACIÓN DE UNA RED NEURONAL KERAS

import pandas as pd
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers.core import Dense
import tensorflow as tf
from sklearn.metrics import
mean_absolute_error,mean_absolute_percentage_error,mean_squared_error,r
2_score
from sklearn.preprocessing import StandardScaler,MinMaxScaler
import matplotlib.pyplot as plt
```

```
archivo='lista1_con nombres.xlsx'

datos= pd.read_excel(archivo, header=1, sheet_name= 'Sheet1')

# Seleccionar las variables de entrada y salida
X = datos.iloc[:, :12]  # 12 primeras columnas
y = datos.iloc[:, 12:]  # la columna nº n-1...

# Dividir los datos en conjunto de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.25, random_state=42, shuffle=True)

X_train1, X_val, y_train1, y_val = train_test_split(X_train, y_train,
test_size=0.25, random_state=42, shuffle=True)

# Normalización con StandardScaler
scaler = MinMaxScaler()
scaler1 = MinMaxScaler()

scaler = scaler.fit(X_train)
X_train_normalizado = scaler.transform(X_train)
X_test_normalizado = scaler.transform(X_test)

scaler1=scaler1.fit(y_train)
y_train_normalizado = scaler1.transform(y_train)
y_test_normalizado = scaler1.transform(y_test)

# Crear el modelo de la red neuronal
model = Sequential()
model.add(Dense(64, input_dim=12, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(10, activation='sigmoid'))

# Compilar el modelo
model.compile(loss='mean_squared_error',
optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3))

callback = tf.keras.callbacks.EarlyStopping(monitor='val_loss',
patience=50)
history = model.fit(verbose =1,
                        x = X_train_normalizado,
                        y = y_train_normalizado,
                        batch_size=32,
                        epochs=4000,
                        validation_split=0.1,
                        shuffle=True,
                        validation_steps=None,
                        validation_freq=1,
                        workers=4,
```

```
                        use_multiprocessing=False,
                        callbacks=[callback])

plt.plot(history.history['loss'])

plt.plot(history.history['val_loss'])

plt.title('loss')

plt.ylabel('loss')

plt.xlabel('epoch')

plt.legend(['train','test'], loc='upper left')

plt.show()

import numpy as np
ypred=model.predict(X_test_normalizado)

y_test_restaurado=scaler1.inverse_transform(y_test_normalizado)
ypred_restaurado=scaler1.inverse_transform(ypred)

# Evaluar el modelo con los datos de prueba

r2=r2_score(y_test_normalizado, ypred)
mae=mean_absolute_error(y_test_normalizado, ypred)
rmse=mean_squared_error(y_test_normalizado, ypred, squared=False)
mse=mean_squared_error(y_test_normalizado, ypred, squared=True)
nmse = mse / np.var(y_test_normalizado)

print("Puntaje del modelo mean abs.:", mae)
print("Puntaje del modelo rmse:", rmse)
print("Puntaje del modelo R2:", r2)
print('nmse:', nmse)
```

CÓDIGO PARA LA EVALUACIÓN DE CADA SALIDA DE LA RED NEURONAL – MULTISALIDA

```
import numpy as np
ypred=model.predict(X_test_normalizado)

y_test_restaurado=scaler1.inverse_transform(y_test_normalizado)
ypred_restaurado=scaler1.inverse_transform(ypred)

#cambiar los números para cambiar la salida
r2_1=r2_score(y_test_restaurado[:, 7], ypred_restaurado[:, 7])
mae_1=mean_absolute_error(y_test_restaurado[:, 7], ypred_restaurado[:,
7])
```

```
rmse_1=mean_squared_error(y_test_restaurado[:, 7], ypred_restaurado[:,
7], squared=False)
mse_1=mean_squared_error(y_test_restaurado[:, 7], ypred_restaurado[:,
7], squared=True)
nmse_1=mse_1 / np.var(y_test_restaurado[:, 7])


print("Puntaje del modelo mean abs.:", mae_1)
print("Puntaje del modelo rmse:", rmse_1)
print("Puntaje del modelo R2:", r2_1)
print('nmse:', nmse_1)
```

CÓDIGO PARA LA ELECCIÓN DEL MODELO DE RED NEURONAL – MULTISALIDA

```
#ELECCIÓN MODELO RED NEURONAL

import pandas as pd
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers.core import Dense
import tensorflow as tf
from sklearn.metrics import
mean_absolute_error,mean_absolute_percentage_error,mean_squared_error,r
2_score
from sklearn.preprocessing import StandardScaler,MinMaxScaler
import matplotlib.pyplot as plt

archivo='lista1_con nombres.xlsx'

datos= pd.read_excel(archivo, header=1, sheet_name= 'Sheet1')

# Seleccionar las variables de entrada y salida
X = datos.iloc[:, :12]  # 12 primeras columnas
y = datos.iloc[:, 12:]  # la columna nº n-1...

# Dividir los datos en conjunto de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.25, random_state=42, shuffle=True)

X_train1, X_val, y_train1, y_val = train_test_split(X_train, y_train,
test_size=0.25, random_state=42, shuffle=True)

# Normalización con StandardScaler
scaler = MinMaxScaler()
scaler1 = MinMaxScaler()

scaler = scaler.fit(X_train1)
X_train_normalizado1 = scaler.transform(X_train1)
X_val_normalizado = scaler.transform(X_val)
```

```
scaler1=scaler1.fit(y_train1)
y_train_normalizado1 = scaler1.transform(y_train1)
y_val_normalizado = scaler1.transform(y_val)

# Crear el modelo de la red neuronal
model = Sequential()
model.add(Dense(16, input_dim=12, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(10, activation='sigmoid'))

# Compilar el modelo
model.compile(loss='mean_squared_error',
optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3))


callback = tf.keras.callbacks.EarlyStopping(monitor='val_loss',
patience=50)
history = model.fit(verbose =1,
                        x = X_train_normalizado1,
                        y = y_train_normalizado1,
                        batch_size=32,
                        epochs=4000,
                        validation_split=0.1,
                        shuffle=True,
                        validation_steps=None,
                        validation_freq=1,
                        workers=4,
                        use_multiprocessing=False,
                        callbacks=[callback])

plt.plot(history.history['loss'])

plt.plot(history.history['val_loss'])

plt.title('loss')

plt.ylabel('loss')

plt.xlabel('epoch')

plt.legend(['train','test'], loc='upper left')

plt.show()

import numpy as np
ypred=model.predict(X_val_normalizado)

y_val_restaurado=scaler1.inverse_transform(y_val_normalizado)
```

```
ypred_restaurado=scaler1.inverse_transform(ypred)

# Evaluar el modelo con los datos de prueba

r2=r2_score(y_val_normalizado, ypred)
mae=mean_absolute_error(y_val_normalizado, ypred)
rmse=mean_squared_error(y_val_normalizado, ypred, squared=False)
mse=mean_squared_error(y_val_normalizado, ypred, squared=True)
nmse = mse / np.var(y_val_normalizado)


print("Puntaje del modelo mean abs.:", mae)
print("Puntaje del modelo rmse:", rmse)
print("Puntaje del modelo R2:", r2)
print('nmse:', nmse)
```

CÓDIGO PARA LA EVALUACIÓN DE CADA SALIDA DE LA RED NEURONAL EN EL PROCESO DE LA ELECCIÓN DE MODELO – MULTISALIDA

```
import numpy as np
ypred=model.predict(X_val_normalizado)
y_val_restaurado=scaler1.inverse_transform(y_val_normalizado)
ypred_restaurado=scaler1.inverse_transform(ypred)

#cambiar los números para cambiar la salida
r2_1=r2_score(y_val_restaurado[:, 2], ypred_restaurado[:, 2])
mae_1=mean_absolute_error(y_val_restaurado[:, 2], ypred_restaurado[:,
2])
rmse_1=mean_squared_error(y_val_restaurado[:, 2], ypred_restaurado[:,
2], squared=False)
mse_1=mean_squared_error(y_val_restaurado[:, 2], ypred_restaurado[:,
2], squared=True)
nmse_1=mse_1 / np.var(y_val_restaurado[:, 2])


print("Puntaje del modelo mean abs.:", mae_1)
print("Puntaje del modelo rmse:", rmse_1)
print("Puntaje del modelo R2:", r2_1)
print('nmse:', nmse_1)
```

CÓDIGO PARA VISUALIZAR LA GRÁFICA DE VALORES PREDICHOS – UNA SALIDA

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

plt.figure()
plt.plot(np.arange(0,len(y_test)),y_test, label = 'Datos reales')
plt.plot(np.arange(0,len(y_test)),ypred, label = 'Datos predichos')
```

```
plt.xlim([0,100])
plt.ylabel('Valores')
plt.xlabel('Muestras')
plt.title('Datos reales vs predichos - Decision Tree')
plt.legend(loc="lower left")
```

CÓDIGO PARA VISUALIZAR LA GRÁFICA DE VALORES PREDICHOS – MULTISALIDA

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

plt.figure()
plt.plot(np.arange(0,len(y_test_normalizado)),y_test_restaurado[:, 8],
label = 'Datos reales')#cambiar los números para cambiar de variable de
salida
plt.plot(np.arange(0,len(y_test_normalizado)),ypred_restaurado[:, 8],
label = 'Datos predichos')
plt.xlim([0,100])
plt.ylabel('Valores')
plt.xlabel('Muestras')
plt.title('Datos reales vs predichos - Random Forest')
plt.legend(loc="lower left")
```

AJUSTES DE LOS HIPERPARÁMETROS FINALES DE CADA ALGORITMO, PARA CADA ANÁLISIS Y VARIABLE

12. Angle of the principal compressive stress, θ (rad)	
LINEAR REGRESSION	`copy_X= True, fit_intercept= True, positive= False`
DECISION TREE REGRESSOR	`ccp_alpha = 0.0, criterion = 'friedman_mse', max_depth = 1000, max_leaf_nodes = 10, min_impurity_decrease = 0.0, min_samples_leaf =10, min_samples_split = 2, min_weight_fraction_leaf = 0.0, random_state = None, splitter = 'random'`
LASSO	`alpha = 0.001, copy_X = False, fit_intercept = True, max_iter = 500, positive = False, precompute = False, random_state = None, selection = 'random', tol = 0.1, warm_start = False`
KNN	`algorithm = 'auto', leaf_size = 10, n_neighbors = 15, p = 3, weights = 'distance'`
SVR	`C = 0.1, coef0 = 1.0, degree = 3, epsilon = 0.3, gamma = 'auto', kernel = 'poly', tol = 0.001`
ADA BOOST	`learning_rate = 0.001, loss = 'exponential', n_estimators = 50`
GRADIENT BOOST	`warm_start = True, subsample = 0.5, n_estimators = 200, min_samples_split = 2, min_samples_leaf = 6, loss = 'huber', learning_rate = 0.1, criterion = 'squared_error'`
RANDOM FOREST	`warm_start = True, oob_score = False, n_estimators = 500, min_samples_split = 5, min_samples_leaf = 6, max_features = None, criterion = 'friedman_mse'`

13. Average strain in the longitudinal steel, εx	
LINEAR REGRESSION	`copy_X= True, fit_intercept= True, positive= False`
DECISION TREE REGRESSOR	`ccp_alpha = 0.0, criterion = 'absolute_error', max_depth = 10, max_leaf_nodes = None, min_impurity_decrease = 0.0, min_samples_leaf =2, min_samples_split = 10, min_weight_fraction_leaf = 0.0, random_state = 1, splitter = 'random'`
LASSO	`alpha = 0.000001, copy_X = False, fit_intercept = True, max_iter = 500, positive = False, precompute = False, random_state = None, selection = 'random', tol = 0.1, warm_start = False`
KNN	`algorithm = 'auto', leaf_size = 10, n_neighbors = 9, p = 2, weights = 'distance'`
SVR	`C = 0.1, coef0 = 1.0, degree = 3, epsilon = 0.1, gamma = 'auto', kernel = 'poly', tol = 0.001`

ADA BOOST	`learning_rate = 0.01, loss = 'exponential', n_estimators = 700`
GRADIENT BOOST	`warm_start = False, subsample = 0.5, n_estimators = 500, min_samples_split = 15, min_samples_leaf = 6, loss = 'huber', learning_rate = 0.1, criterion = 'friedman_mse'`
RANDOM FOREST	`warm_start = False, oob_score = False, n_estimators = 500, min_samples_split = 5, min_samples_leaf = 4, max_features = None, criterion = 'squared_error'`

14. Shear internal force, V (N)	
LINEAR REGRESSION	`copy_X= True, fit_intercept= True, positive= False`
DECISION TREE REGRESSOR	`ccp_alpha = 0.0, criterion = 'absolute_error', max_depth = None, max_leaf_nodes = None, min_impurity_decrease = 0.0, min_samples_leaf =10, min_samples_split = 2, min_weight_fraction_leaf = 0.0, random_state = 123, splitter = 'best'`
LASSO	`alpha = 0.00001, copy_X = False, fit_intercept = True, max_iter = 1000, positive = False, precompute = False, random_state = None, selection = 'random', tol = 0.1, warm_start = True`
KNN	`algorithm = 'auto', leaf_size = 10, n_neighbors = 11, p = 1, weights = 'distance'`
SVR	`C = 1, coef0 = 0.5, degree = 2, epsilon = 0.3, gamma = 'auto', kernel = 'poly', tol = 0.01`
ADA BOOST	`learning_rate = 0.01, loss = 'exponential', n_estimators = 50`
GRADIENT BOOST	`warm_start = True, subsample = 0.5, n_estimators = 200, min_samples_split = 15, min_samples_leaf = 6, loss = 'huber', learning_rate = 0.1, criterion = 'squared_error'`
RANDOM FOREST	`warm_start = False, oob_score = False, n_estimators = 100, min_samples_split = 10, min_samples_leaf = 6, max_features = None, criterion = 'friedman_mse'`

15. Avarage strain in the transverse steel, εt	
LINEAR REGRESSION	`copy_X= True, fit_intercept= True, positive= False`
DECISION TREE REGRESSOR	`ccp_alpha = 0.0, criterion = 'absolute_error', max_depth = 1000, max_leaf_nodes = None, min_impurity_decrease = 0.0, min_samples_leaf =5, min_samples_split = 5, min_weight_fraction_leaf = 0.0, random_state = None, splitter = 'random'`
LASSO	`alpha = 0.000001, copy_X = False, fit_intercept = True, max_iter = 1000, positive = False, precompute = False, random_state = None, selection = 'random', tol = 0.1, warm_start = True`
KNN	`algorithm = 'auto', leaf_size = 10, n_neighbors = 9, p = 2, weights = 'distance'`
SVR	`C = 0.1, coef0 = 1.0, degree = 2, epsilon = 0.1, gamma = 'auto', kernel = 'poly', tol = 0.001`
ADA BOOST	`learning_rate = 0.001, loss = 'square', n_estimators = 50`
GRADIENT BOOST	`warm_start = True, subsample = 0.5, n_estimators = 200, min_samples_split = 10, min_samples_leaf = 6, loss = 'huber', learning_rate = 0.1, criterion = 'squared_error'`
RANDOM FOREST	`warm_start = False, oob_score = False, n_estimators = 500, min_samples_split = 5, min_samples_leaf = 6, max_features = None, criterion = 'friedman_mse'`

16. Principal compressive strain, ε2	
LINEAR REGRESSION	`copy_X= True, fit_intercept= True, positive= False`
DECISION TREE REGRESSOR	`ccp_alpha = 0.0, criterion = 'absolute_error', max_depth = 10, max_leaf_nodes = None, min_impurity_decrease = 0.0, min_samples_leaf =5, min_samples_split = 2, min_weight_fraction_leaf = 0.0, random_state = None, splitter = 'best'`
LASSO	`alpha = 0.000000001, copy_X = True, fit_intercept = True, max_iter = 1000, positive = False, precompute = True, random_state = None, selection = 'cyclic', tol = 0.00001, warm_start = True`
KNN	`algorithm = 'auto', leaf_size = 10, n_neighbors = 3, p = 2, weights = 'distance'`
SVR	`C = 10, coef0 = 0.5, degree = 2, epsilon = 0.1, gamma = 'auto', kernel = 'poly', tol = 0.1`
ADA BOOST	`learning_rate = 0.001, loss = 'exponential', n_estimators = 700`

GRADIENT BOOST	`warm_start = True, subsample = 0.5, n_estimators = 500, min_samples_split = 5, min_samples_leaf = 6, loss = 'huber', learning_rate = 0.1, criterion = 'squared_error'`
RANDOM FOREST	`warm_start = True, oob_score = True, n_estimators = 100, min_samples_split = 5, min_samples_leaf = 2, max_features = None, criterion = 'friedman_mse'`

17. Average strain in the prestressing strands, εp	
LINEAR REGRESSION	`copy_X= True, fit_intercept= True, positive= False`
DECISION TREE REGRESSOR	`ccp_alpha = 0.0, criterion = 'absolute_error', max_depth = 10, max_leaf_nodes = None, min_impurity_decrease = 0.0, min_samples_leaf =10, min_samples_split = 5, min_weight_fraction_leaf = 0.0, random_state = 123, splitter = 'random'`
LASSO	`alpha = 0.000000001, copy_X = True, fit_intercept = True, max_iter = 1000, positive = False, precompute = True, random_state = None, selection = 'cyclic', tol = 0.00001, warm_start = True`
KNN	`algorithm = 'auto', leaf_size = 10, n_neighbors = 13, p = 3, weights = 'distance'`
SVR	`C = 0.1, coef0 = 1.0, degree = 3, epsilon = 0.2, gamma = 'auto', kernel = 'poly', tol = 0.01`
ADA BOOST	`learning_rate = 0.001, loss = 'linear', n_estimators = 100`
GRADIENT BOOST	`warm_start = True, subsample = 0.5, n_estimators = 100, min_samples_split = 15, min_samples_leaf = 6, loss = 'huber', learning_rate = 0.1, criterion = 'friedman_mse'`
RANDOM FOREST	`warm_start = True, oob_score = False, n_estimators = 500, min_samples_split = 15, min_samples_leaf = 6, max_features = None, criterion = 'friedman_mse'`

18. Prestressing steel stress, σp (Mpa)	
LINEAR REGRESSION	`copy_X= True, fit_intercept= True, positive= False`
DECISION TREE REGRESSOR	`ccp_alpha = 0.0, criterion = 'absolute_error', max_depth = None, max_leaf_nodes = None, min_impurity_decrease = 0.0, min_samples_leaf =10, min_samples_split = 2, min_weight_fraction_leaf = 0.0, random_state = None, splitter = 'best'`
LASSO	`alpha = 0.000000001, copy_X = True, fit_intercept = True, max_iter = 1000, positive = False, precompute = True, random_state = None, selection = 'cyclic', tol = 0.00001, warm_start = True`
KNN	`algorithm = 'auto', leaf_size = 10, n_neighbors = 13, p = 3, weights = 'distance'`
SVR	`C = 0.1, coef0 = 1.0, degree = 3, epsilon = 0.2, gamma = 'auto', kernel = 'poly', tol = 0.001`
ADA BOOST	`learning_rate = 0.0001, loss = 'square', n_estimators = 200`
GRADIENT BOOST	`warm_start = True, subsample = 1.0, n_estimators = 100, min_samples_split = 10, min_samples_leaf = 1, loss = 'huber', learning_rate = 0.1, criterion = 'squared_error'`
RANDOM FOREST	`warm_start = False, oob_score = False, n_estimators = 100, min_samples_split = 2, min_samples_leaf = 6, max_features = None, criterion = 'squared_error'`

19. Principal compressive stress, σ2 (Mpa)	
LINEAR REGRESSION	`copy_X= True, fit_intercept= True, positive= False`
DECISION TREE REGRESSOR	`ccp_alpha = 0.0, criterion = 'absolute_error', max_depth = 100, max_leaf_nodes = None, min_impurity_decrease = 0.0, min_samples_leaf =2, min_samples_split = 100, min_weight_fraction_leaf = 0.0, random_state = 42, splitter = 'best'`
LASSO	`alpha = 0.000000001, copy_X = True, fit_intercept = True, max_iter = 1000, positive = False, precompute = True, random_state = None, selection = 'cyclic', tol = 0.00001, warm_start = True`
KNN	`algorithm = 'auto', leaf_size = 10, n_neighbors = 13, p = 2, weights = 'distance'`
SVR	`C = 10, coef0 = 1.0, degree = 2, epsilon = 0.3, gamma = 'scale', kernel = 'poly', tol = 0.1`
ADA BOOST	`learning_rate = 0.01, loss = 'exponential', n_estimators = 50`
GRADIENT BOOST	`warm_start = False, subsample = 1.0, n_estimators = 500, min_samples_split = 5, min_samples_leaf = 6,`

	`loss = 'huber', learning_rate = 0.1, criterion = 'friedman_mse'`
RANDOM FOREST	`warm_start = False, oob_score = False, n_estimators = 100, min_samples_split = 5, min_samples_leaf = 4, max_features = None, criterion = 'friedman_mse'`

20. Average tensile stress in the longitudinal steel, σsx (Mpa)	
LINEAR REGRESSION	`copy_X= True, fit_intercept= True, positive= False`
DECISION TREE REGRESSOR	`ccp_alpha = 0.0, criterion = 'absolute_error', max_depth = 1000, max_leaf_nodes = None, min_impurity_decrease = 0.0, min_samples_leaf =10, min_samples_split = 10, min_weight_fraction_leaf = 0.0, random_state = None, splitter = 'random'`
LASSO	`alpha = 0.000000001, copy_X = True, fit_intercept = True, max_iter = 1000, positive = False, precompute = True, random_state = None, selection = 'cyclic', tol = 0.00001, warm_start = True`
KNN	`algorithm = 'auto', leaf_size = 10, n_neighbors = 9, p = 2, weights = 'distance'`
SVR	`C = 1, coef0 = 1.0, degree = 3, epsilon = 0.1, gamma = 'scale', kernel = 'poly', tol = 0.01`
ADA BOOST	`learning_rate = 0.01, loss = 'exponential', n_estimators = 200`
GRADIENT BOOST	`warm_start = True, subsample = 0.5, n_estimators = 500, min_samples_split = 15, min_samples_leaf = 6, loss = 'huber', learning_rate = 0.1, criterion = 'squared_error'`
RANDOM FOREST	`warm_start = True, oob_score = True, n_estimators = 100, min_samples_split = 2, min_samples_leaf = 4, max_features = None, criterion = 'squared_error'`

21. Average tensile stress in the transverse steel, σst (Mpa)	
LINEAR REGRESSION	`copy_X= True, fit_intercept= True, positive= False`
DECISION TREE REGRESSOR	`ccp_alpha = 0.0, criterion = 'absolute_error', max_depth = None, max_leaf_nodes = None, min_impurity_decrease = 0.0, min_samples_leaf = 10, min_samples_split = 2, min_weight_fraction_leaf = 0.0, random_state = 42, splitter = 'best'`
LASSO	`alpha = 0.000000001, copy_X = True, fit_intercept = True, max_iter = 1000, positive = False, precompute = True, random_state = None, selection = 'cyclic', tol = 0.00001, warm_start = True`
KNN	`algorithm = 'auto', leaf_size = 10, n_neighbors = 13, p = 1, weights = 'distance'`
SVR	`C = 10, coef0 = 0.5, degree = 3, epsilon = 0.1, gamma = 'scale', kernel = 'poly', tol = 0.01`
ADA BOOST	`learning_rate = 0.01, loss = 'exponential', n_estimators = 100`
GRADIENT BOOST	`warm_start = True, subsample = 0.5, n_estimators = 500, min_samples_split = 5, min_samples_leaf = 6, loss = 'huber', learning_rate = 0.1, criterion = 'friedman_mse'`
RANDOM FOREST	`warm_start = False, oob_score = False, n_estimators = 100, min_samples_split = 5, min_samples_leaf = 6, max_features = None, criterion = 'absolute_error'`

Análisis multisalida	
LINEAR REGRESSION	`copy_X = True, fit_intercept = False, positive = False`
DECISION TREE REGRESSOR	`splitter = 'best', random_state = 123, min_samples_split = 10, min_samples_leaf = 10, max_leaf_nodes = None, max_depth = None, criterion = 'absolute_error'`
LASSO	`alpha = 0.001, copy_X = True, fit_intercept = False, max_iter = 100, positive = False, precompute = True, random_state = None, selection = 'cyclic', tol = 0.01, warm_start = True`
KNN	`algorithm = 'auto', leaf_size = 10, n_neighbors = 11, p = 2, weights = 'distance'`
RANDOM FOREST	`warm_start = True, oob_score = True, n_estimators = 500, min_samples_split = 2, min_samples_leaf = 4, max_features = None, criterion = 'squared_error'`

Universidad de Alcalá
Escuela Politécnica Superior

ESCUELA POLITECNICA
SUPERIOR

Universidad
de Alcalá