UNIVERSIDAD AUTONOMA DE MADRID

Escuela Politécnica Superior

20|21

Trabajo fin de grado

Predicción de energía eólica con métodos de ensemble

José Benjumeda Rubio

Escuela Politécnica Superior
Universidad Autónoma de Madrid
C\Francisco Tomás y Valiente nº 11

Campus Internacional excelencia UAM CSIC+

www.uam.es

UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Predicción de energía eólica con métodos de ensemble

Autor: José Benjumeda Rubio
Tutora: Ángela Fernández Pascual

junio 2021

José Benjumeda Rubio
Predicción de energía eólica con métodos de ensemble

José Benjumeda Rubio
C\ Francisco Tomás y Valiente Nº 11

IMPRESO EN ESPAÑA – PRINTED IN SPAIN

A mi familia y a mis amigos, por ser un impulso constante en todo lo que hago.

“Investigación es lo que hago cuando no sé lo que estoy haciendo”

Wernher von Braun

Agradecimientos

En primer lugar me gustaría agradecer a Ángela Fernández Pascual que me haya guiado en la realización de este TFG, y que constantemente haya tenido palabras de ánimo para seguir trabajando.

Además, este trabajo no hubiese sido posible sin la ayuda de la Universidad Autónoma de Madrid y el Instituto de Ingeniería del Conocimiento, que han permitido que tenga acceso a los datos de predicciones meteorológicas del Centro Europeo de Previsiones Meteorológicas a Plazo Medio y los datos de energía obtenida en el Parque Eólico Experimental de Sotavento, además de facilitar una cuenta en el Centro de Computación Científica de la UAM. Por esto les estoy tremendamente agradecido.

Finalmente, en estos cinco años en la Universidad Autónoma de Madrid he conocido a personas extraordinarias que tengo la suerte de considerar amigos y amigas. Os agradezco de corazón haber hecho cada día mejor que el anterior.

Resumen

La inteligencia artificial es un área de las ciencias de computación en la que la inversión de trabajo y esfuerzo de investigación esta obteniendo grandes recompensas. Es un campo en el que cada vez parece haber más posibilidades, y en el que en este trabajo pretendemos avanzar para dar una solución a un problema concreto: la predicción de energía eólica en el Parque Eólico Experimental de Sotavento.

La energía eólica es una forma de energía renovable en la que España ha sido pionera, y cada vez es mayor el porcentaje de energía que proviene de esta fuente. Por esto es indispensable conseguir predicciones precisas de la energía que se va a obtener en todo momento, y en este trabajo abordamos el problema de predecirla a partir de datos de predicciones atmosféricas.

Se estudiarán varios modelos de regresión para luego construir un modelo de ensemble stacking que trate de compensar los fallos de los modelos por los que está compuesto, y se hará un análisis de los resultados obtenidos, para dar indicaciones sobre posible trabajo futuro.

Palabras clave

Energía eólica, modelos de regresión, métodos de ensemble, stacking

Abstract

Artificial intelligence is a field of computer science in which the inversion of work and investigation is granted a very decent reward. The posibilities in this field seem to be growing exponentially, and in this assignment we will try to give one more step in its development by finding a solution to a particular problem: the prediction of wind energy in Parque Eólico Experimental de Sotavento.

Wind energy is a source of renewable energy in which Spain has been a pioneer, and the percentaje of energy that comes from this source is becoming larger and larger, with ambitious goals for the years to come. For this reason, getting accurate predictions for the amount of energy that is going to be available at a given time is essential. In this assignment we address the problem of predicting the amount of wind energy obtained by learning its relation with the weather forecast.

We will study individual regression models with which we will then build an ensemble method, and the latter will attempt to fix some of the errors of the models it is composed of.

Keywords

Wind energy, regression models, ensemble methods, stacking

ÍNDICE

Listas

Lista de algoritmos

Lista de códigos

Lista de cuadros

Lista de ecuaciones

Lista de figuras

Lista de tablas

Lista de cuadros

1

Introducción

1.1. Motivación

El uso de la energía eólica está en auge desde hace años, y España es uno de los países pioneros en fomentar su desarrollo. Está claro que la inversión en esta fuente de energía es una apuesta ganadora.

A causa de que la sociedad está cada vez más concienciada con el cuidado del medio ambiente, y que la investigación en fuentes de energía limpias da abundantes frutos, la evolución hacia estas fuentes de energía parece ser imparable. En España hay una gran cantidad de puestos de trabajo en este sector, y a día de hoy hay instalados del orden de 30000 megavatios eólicos, con ambiciosos objetivos para los próximos años.

Sin embargo, la cantidad de energía eólica va ligada a ciertos factores sobre los que no tenemos control alguno: las condiciones atmosféricas. Dado que para cualquier empresa del mercado eólico es indispensable conocer la cantidad de megavatios de los que dispone, no es poca la investigación que se ha llevado a cabo en este campo, aunque aún queda mucho por hacer.

1.2. Objetivos

A lo largo de este trabajo pretendemos comprobar que error de predicción podemos conseguir prediciendo la energía obtenida en el Parque Eólico de Sotavento, en Galicia, utilizando como variables independientes las predicciones meteorológicas del Centro Europeo de Previsiones Meteorológicas a Plazo Medio.

Se estudiará un modelo de regresión regularizada, un perceptrón multicapa, una máquina de vectores soporte de regresión, y finalmente se utilizarán estos para construir un modelo de ensemble de

regresión. Estos tres modelos los entrenaremos individualmente para ver qué error podemos conseguir con cada uno por separado, y para calcular con qué parámetros funcionan mejor, y después se incorporarán también al modelo de ensemble para ver si se obtiene alguna mejora.

1.3. Estructura del documento

Hemos organizado este documento comenzando con una explicación de cada modelo, no en excesiva profundidad, ya que este TFG se centra en los resultados experimentales y en el análisis de los mismos, pero sí lo suficientemente detallada como para poder trabajar con los modelos con cierto criterio y dominio sobre cada uno. Además, cierto conocimiento sobre el modelo que se está utilizando es indispensable para poder configurar sus hiperparámetros adecuadamente, y para interpretar luego qué resultados estamos obteniendo, y qué variaciones podrían llevarnos a mejorarlos.

A continuación se hace una explicación bastante detallada de la naturaleza de los datos con los que tratamos, partiendo de la idea de que un conocimiento profundo del problema es una tarea de la que ningún modelo de predicción nos puede eximir.

Tras la explicación de cada experimento, se hace un análisis de los resultados, para posteriormente, tras los resultados del modelo de ensemble, profundizar aún más en los puntos fuertes y débiles de cada uno de los modelos, y especialmente del modelo de ensemble.

La última parte corresponde a las conclusiones tras haber realizado todo el resto del trabajo, y con algunas indicaciones sobre el trabajo futuro que, tras haber hecho un análisis de las posibles causas de los errores de los modelos, creemos que pueden conducir a mejorar los resultados de este trabajo.

2

Estado del arte

En este capítulo revisamos algunos trabajos que tratan sobre predicción de energía eólica o utilizan métodos de ensemble en problemas que pueden tener cierta similitud con la predicción de energía eólica. Este trabajo no tiene base en todos ellos, pero nos ayudan a tener una idea general de en qué punto está la investigación en cada problema.

Aprendizaje multitarea con máquinas de vectores soporte de regresión para la predicción de energía solar y eólica

El primer trabajo que citaremos es *Multitask Support Vector Regression for Solar and Wind Energy Prediction*, de Carlos Ruiz, Carlos M. Alaíz y José R. Dorronsoro, que aborda el problema de predicción de energía eólica y solar mediante máquinas de vectores soporte de regresión multitarea. El uso de modelos de predicción multitarea se basa en que el problema se puede resolver dividiéndolo en tareas que, a pesar de tener cada una sus distintas particularidades, están relacionadas unas con otras. De esta manera, se entrena un modelo para que resuelva las tres tareas a la vez y la información que el modelo tiene que aprender para resolver cada una de éstas servirá de ayuda para resolver las demás.

En este trabajo se realizan experimentos con tres conjuntos de datos, y uno de ellos son los datos de energía eólica del Parque Eólico Experimental de Sotavento a partir de las predicciones meteorológicas del Centro Europeo de Previsiones Meteorológicas a Plazo Medio, que son precisamente los datos que se utilizan en este trabajo.

Las tasas de error que se han conseguido utilizando máquinas de vectores soporte de regresión multitarea han servido como guía para saber qué rangos de error serían aceptables. Para distintos modelos, obtienen errores absolutos medios que están entre 6,132 y 6,410, con lo que intentaremos conseguir errores también en ese rango, y descartaremos modelos que nos den errores que no se acerquen a estas cantidades.

Otro de los procedimientos que sigue el trabajo de Carlos Ruiz, Carlos M. Alaíz y José R. Dorronsoro y que nosotros seguiremos también es la normalización de los datos. Esto consiste en redimensionar los datos para que la desviación típica sea 1 y la media 0, pero utilizando solo los datos del conjunto de entrenamiento.

Además, se lleva a cabo una validación cruzada para estimar los parámetros de una máquina de vectores soporte de regresión, y se acotan intervalos para los distintos parámetros del modelo. A lo largo de nuestro trabajo, realizaremos también una validación cruzada sobre una máquina de vectores soporte de regresión, y resultará de gran ayuda esta base para encontrar los mejores parámetros con los que calibrar el modelo.

Métodos de ensemble boosting para la predicción de energía solar y eólica: un análisis sistemático.

Otro trabajo en esta línea es *Boosting algorithms in energy research: a systematic review*, de Hristos Tyralis y Georgia A Papacharalampous, en el que se utilizan modelos de ensemble *boosting* para realizar predicciones en energía eólica y fotovoltaica.

Los modelos de ensemble boosting se componen de varias instancias del mismo modelo variando sus hiperparámetros, y se caracteriza y diferencia de otros modelos de ensemble en que el entrenamiento es secuencial, y el entrenamiento de cada modelo influye en el entrenamiento de los modelos posteriores.

En este trabajo se realizan experimentos de predicción de energía eólica y solar, y proporciona una perspectiva útil del trabajo realizado con los métodos de ensemble boosting, de los que hablaremos más adelante.

Predicción de rampas de viento utilizando redes neuronales multicapa y aprendizaje multitarea

Un último trabajo sobre predicción eólica cuyo análisis previo a nuestro trabajo ha sido interesante es *Multi-task learning for the prediction of wind power ramp events with deep neural networks*, que utiliza redes neuronales multitarea para predecir la ocurrencia de rampas de viento.

Las rampas de viento son grandes variaciones en la dirección o velocidad del viento en un periodo de tiempo corto. Este es uno de los elementos que hacen la predicción de energía eólica un problema complicado. Como veremos más adelante en los datos de predicciones meteorológicas, las rampas de viento son muy comunes, y aprender a predecirlas es vital para a partir de ahí estimar la energía que

se obtendrá.

El enfoque que se le da a la predicción de rampas de viento en este trabajo son las redes neuronales multitarea. El aprendizaje multitarea en este caso es algo más intuitivo que en el caso de las máquinas de vectores soporte que explicamos al inicio del capítulo. Básicamente, se añaden neuronas extra en la última capa de la red, cuya salida representa la solución de una tarea secundaria que se intuye relacionada con la tarea principal. Los pesos de la red se calcularán de manera que la red solucione ambas tareas, y se espera que el conocimiento necesario para resolver las tareas secundarias sea beneficioso a la hora de resolver la tarea principal. Este trabajo es de M. Dorado-Moreno, N. Navarin, P. A. Gutiérrez, L. Prieto, A. Sperduti, S. Salcedo-Sanz y C. Hervás-Martínez.

3

Modelos de regresión

3.1. El problema de regresión

La mayoría de esta sección ha sido obtenida a partir del capítulo 3 de [1].

Como ya hemos dicho, nos encontramos frente a un problema de regresión: queremos encontrar la relación que hay entre unas variables aleatorias independientes, que son ciertas predicciones meteorológicas; y una variable aleatoria dependiente, que es la cantidad de energía producida cada hora de cada día en el parque eólico experimental de Sotavento, en Galicia.

Las variables aleatorias independientes serán la velocidad del viento a 10 y 100 metros de altura, la presión sobre la superficie, y la temperatura a 2 metros de altura. No se profundizará más por ahora en el problema de Sotavento, sino que se detallarán los datos y las condiciones del problema en la sección de simulaciones y resultados 5.

En esta sección comenzaremos explicando cómo podemos obtener una solución lineal, viendo en cierta profundidad la regresión multilineal y la regresión regularizada, para, a continuación, estudiar dos modelos más complejos y no lineales: el perceptrón multicapa y la máquina de vectores soporte de regresión.

Regresión lineal simple

En un problema de regresión lineal simple queremos encontrar la expresión de una variable aleatoria, la variable de respuesta, en función de una variable independiente, de manera que la relación que hallemos entre ambas sea lineal. A la variable dependiente la llamaremos X, y a la variable independiente Y. Además, toda la información que obtengamos sobre la relación entre estas vendrá de una muestra de datos D_n, compuesta por n observaciones:

$$D_n = \{(X_1, Y_1), (X_2, Y_2), \ldots, (X_n, Y_n)\}, \tag{3.1}$$

donde (X_i, Y_i) es la observación i, X_i es el atributo de la observación i e Y_i es el *target* de la observación i.

Llamaremos $\widetilde{Y}$ a nuestra predicción del valor de Y, y la escribimos en función de X como

$$\widetilde{Y} = f(X) = \beta_0 + \beta_1 X, \tag{3.2}$$

donde β_0 es el corte de la recta con el eje OY, β_1 es la pendiente de la recta y $f(X)$ es el modelo lineal. A partir de la muestra, construiremos una función cuyos valores se acerquen lo máximo posible a los valores de la variable aleatoria Y, y esta cercanía la mediremos con el error cuadrático medio.

Hallaremos los valores de β_0 y β_1 minimizando el error cuadrático medio visto como función de β_0 y β_1, a partir de D_n:

$$J(\beta_0, \beta_1) = \frac{1}{n}\sum_{i=1}^{n}(Y_i - (\beta_0 + \beta_1 X_i))^2. \tag{3.3}$$

Esta función es convexa, es decir, tiene la forma de una parábola positiva, con lo cual, sabemos que siempre tendrá mínimo. Anulamos entonces las derivadas parciales y obtenemos los valores que hacen mínimo el error cuadrático:

$$\beta_0 = E[Y] - E[X]\frac{COV(X,Y)}{\sigma^2(X)} \tag{3.4}$$

$$\beta_1 = \frac{E[XY] - E[X]\,E[Y]}{E[X^2] - E[X]^2} = \frac{COV(X,Y)}{\sigma^2(X)}, \tag{3.5}$$

donde $E[\,\cdot\,]$ y $\sigma^2(\,\cdot\,)$ son respectivamente esperanza y varianza de una variable aleatoria, y $COV(\,\cdot\,,\,\cdot\,)$ es la covarianza de dos variables aleatorias.

Regresión lineal múltiple

La regresión lineal múltiple o multilineal contempla el problema en el que hay que predecir el valor de una variable dependiente a partir de p variables independientes. Veremos que el modelo se construye siguiendo la misma lógica que para una sola variable, sin embargo será son lo suficientemente complejo como para obtener las primeras predicciones de nuestro problema.

El enfoque que utilizaremos para explicar la regresión multilineal consiste en adaptar el modelo de regresión simple para que en lugar de tener una función de una sola variable, tengamos una función

vectorial

$$\widetilde{Y} = \beta_0 + \boldsymbol{\beta}\boldsymbol{X} = \beta_0 + \sum_{i=1}^{p} \beta_i X_i, \tag{3.6}$$

donde X_i representa la iésima variable, y β_i cuantifica la asociación entre esa variable y la variable respuesta. El valor de cada parámetro β_i nos da una idea del efecto que tiene el incremento de la variable i sobre la variable final.

La manera de obtener los valores de β_i es análoga a la regresión simple: simplemente hay que minimizar una función de p variables:

$$J(\beta_0, \beta_1, , \ldots, \beta_n) = \sum_{i=1}^{n} (Y_i - \widetilde{Y}_i)^2 = \sum_{i=1}^{n} (Y_i - \beta_0 - \sum_{i=1}^{p} \beta_i X_i)^2. \tag{3.7}$$

Sin embargo, para $p > 1$ las fórmulas no son tan compactas y fáciles de escribir como para $p = 1$, y, dado que las podemos obtener mediante cualquier software de estadística, no las incluiremos aquí.

A partir de aquí, los modelos que estudiaremos tienen mayor complejidad: hay que calibrarlos con hiperparámetros, penalizan ciertas características no deseables en la función de regresión obtenida, etc. Para poder utilizarlos y sacarles mayor partido tenemos que estudiar antes dos técnicas muy conocidas: la estandarización de los datos y la validación cruzada.

3.1.1. Preprocesamiento de datos: Estandarización

En los datos eólicos del parque de Sotavento encontramos variables con unidades de medida distintas, por ejemplo velocidad del viento en metros por segundo y presión en pascales, por lo que las magnitudes son muy diferentes unas de otras. Para que un modelo no asuma que una está más relacionada con la cantidad de energía obtenida que otra, por el simple hecho de tener valores más grandes, realizamos la estandarización de los datos.

El proceso de estandarización de los datos es muy sencillo: simplemente tenemos que restar a cada variable independiente su media, y dividirla por su desviación típica. De esta manera, todas las variables tendrán media cero, y sus valores se alejarán de dicho valor una media de una unidad. Ahora todos los datos tienen magnitudes similares, pero no hemos perdido ninguna información: simplemente hemos hecho un cambio de escala.

3.1.2. Elección de hiperparámetros: validación cruzada

Como ya se ha comentado, los modelos que vamos a estudiar tienen hiperparámetros, cuyos valores determinan en gran medida en las predicciones que obtendremos. Para seleccionar qué valor deben tener los hiperparámetros de un modelo, utilizamos la validación cruzada.

Lo primero que hay que hacer es definir un conjunto finito de posible valores para los hiperparámetros que queremos calibrar, y definir un modelo con cada posible combinación. Obtenemos entonces una familia de modelos, y tenemos que ver cual de ellos predice mejor. Esto significa que hay que definir una métrica, una función de medida del error de predicción. En nuestros experimentos hemos utilizado el error cuadrático medio, pero puede ser cualquier otra.

A continuación hay que tomar una partición del conjunto de entrenamiento en K subconjuntos. Fijando uno de estos k subconjuntos como conjunto de validación, entrenamos cada modelo sobre el resto del conjunto de entrenamiento, y calculamos el error de predecir sobre las observaciones del conjunto de validación.

Todo este proceso se repite k veces, de manera que hayamos hecho predicciones sobre todos los subconjuntos de la partición, y por tanto sobre todos los elementos del conjunto de entrenamiento. Calculamos el error cuadrático medio de cada modelo para las predicciones obtenidas y nos quedamos con el que tenga el mínimo.

Un caso particular muy conocido es la validación cruzada utilizando la partición *leave-one-out*, donde los k subconjuntos tienen cada uno un elemento, y por tanto la partición tiene tantos subconjuntos como datos tenga el conjunto de entrenamiento. Tiene la desventaja de que el coste computacional es muy superior a una validación cruzada con 5 subconjuntos por ejemplo (contando desde luego con un conjunto de entrenamiento de tamaño muy superior a 5), pues entrenamos los modelos muchas más veces.

3.2. Regresión regularizada

Esta sección ha sido obtenida a partir de la sección 6.2.1 de [1].

La regresión regularizada es un modelo de regresión muy similar al de regresión multilineal, en el que penalizamos los coeficientes de las variables independientes $\beta_1, \beta_2, \ldots, \beta_n$, basándonos en que los valores muy grandes de estos provocarán sobreajuste, y por tanto predicciones de peor calidad. Tenemos la siguiente ecuación:

$$\widetilde{J}(\beta_0, \beta_1, \ldots, \beta_p) = \sum_{i=1}^{n}(Y_i - \beta_0 - \sum_{i=1}^{p}\beta_i X_i)^2 + \lambda \sum_{j=1}^{p}\beta_j^2 = J(\beta_0, \beta_1, \ldots, \beta_p) + \lambda \sum_{j=1}^{p}\beta_j^2, \qquad (3.8)$$

donde el hiperparámetro λ balancea la importancia relativa de minimizar el error cuadrático y de minimizar los coeficientes. Según el valor de λ, minimizaremos una función u otra, con lo que las soluciones serán distintas. Utilizaremos la validación cruzada para encontrar el valor de λ que produzca la solución de menor error.

Es importante notar que la penalización no se aplica sobre β_0, que es simplemente la media de la variable respuesta cuando todas las variables independientes tienen valor 0.

Normalmente, en problemas que tengan una relación casi lineal entre las variables independientes y la variable respuesta, la regresión multilineal suele tener poco sesgo y varianza relativamente alta. Esto significa que un cambio pequeño en los datos puede provocar una gran diferencia en la estimación de los coeficientes. Cuanto mayor sea la cantidad de variables, p, con respecto a la cantidad de datos, n, mayor será esta diferencia. En estos casos la penalización que aplica la regresión regularizada es una gran ayuda para reducir esta varianza.

Por último, el coste computacional de la regresión regularizada es prácticamente el mismo que el de la regresión multilineal, así que es una técnica que cumplirá las restricciones de tiempo de casi cualquier problema para el que la regresión multilineal las cumpla.

Como veremos en los experimentos, la regresión regularizada es una solución que en relación a su simplicidad, es capaz de conseguir resultados altamente precisos.

3.3. Percetrón multicapa

La mayoría de esta sección ha sido obtenida a partir del capítulo 6 de [2].

En ésta sección veremos la red neuronal multicapa, también conocida como perceptrón multicapa. Este es el primer modelo de los vistos hasta ahora que nos permite resolver problemas no lineales. Sin embargo, también pueden resolver problemas lineales. A cambio de una complejidad mayor y un coste computacional más elevado que la regresión regularizada, nos permiten reducir en gran medida los errores de predicción.

Una red neuronal consiste en capas que son básicamente conjuntos de neuronas, y están conectadas unas con otras ponderadas con pesos. Veremos que los pesos son un factor muy determinante en el rendimiento de la red neuronal, y calcularlos son una de las principales dificultades en el uso de estos modelos.

Hay distintas maneras de entrenar una red neuronal para calcular sus pesos. La más común y la que se ha utilizado en este trabajo es la retropropagación.

Una de las mayores dificultades a la hora de utilizar redes neuronales es ajustar su complejidad, es decir, definir su arquitectura y los valores de sus hiperparámetros. Si la red tiene demasiadas capas ocultas con muchas neuronas, el modelo tendrá sobreajuste, mientras que si no tiene suficientes, generalizará demasiado, es decir, tendrá subajuste.

3.3.1. Estructura y alcance de las redes neuronales

Las redes neuronales que vamos a estudiar tienen capa de entrada, una serie de capas ocultas y capa de salida. Normalmente la cantidad de neuronas de la capa de entrada y de la de salida se obtiene directamente del problema, según la cantidad de atributos de cada dato y la cantidad de variables respuesta. La cantidad de neuronas de cada capa oculta es en cambio algo que tiene que decidir el diseñador.

Las neuronas se conectan por pesos. El sesgo se incluye como una neurona más que se conecta a todas las demás excepto a las de la capa de entrada, que emite un valor constante y no tiene entradas.

La observación se introduce en la primera capa, recibiendo cada neurona el valor de un atributo de la observación. Las neuronas de esta capa transmiten el valor de su entrada a su salida sin modificarlo, es decir, su función es la función identidad.

Como estamos estudiando un problema de regresión, la capa de salida tendrá una sola neurona, y su salida será la predicción para la observación introducida. Tanto las neuronas de la capa de entrada como las de la capa de salida tendrán como función de activación la función identidad, mientras que las neuronas de las capas ocultas tendrán la función ReLU:

$$f(x) = \text{máx}\{x, 0\} = \begin{cases} x & \text{si } x \geq 0 \\ \\ 0 & \text{si } x < 0 \end{cases} . \tag{3.9}$$

Las entradas de las neuronas de la primera capa serán los atributos de cada observación, recibiendo la neurona i el valor del atributo i. La entrada de las demás neuronas será, como ya hemos dicho, una combinación lineal de las salidas de las neuronas de la capa anterior y la neurona del sesgo.

Para una red neuronal de una sola capa oculta podemos escribir la salida en función de la entrada de manera bastante sencilla, y al añadir más capas ocultas el procedimiento es el análogo con más pasos. Llamamos x_i a la salida de las neuronas de la primera capa, y_j a las de la capa oculta k, y z a la salida de la única neurona de la capa de salida:

$$x = (x_1, \ldots, x_{n_1}),$$

$$y = (y_1, \ldots, y_{n_2}).$$

Además, net_j es la entrada de la neurona j de la capa oculta, y net es la entrada de la neurona de la capa de salida.

Escribimos z en función de $\boldsymbol{y}$ como

$$z(\boldsymbol{y}) = f(net) = f\left(\sum_0^n w_j y_j\right). \tag{3.10}$$

donde w_j es el peso que une la neurona j de la capa oculta con la neurona de la capa de salida. Además, sabemos que la salida de cada neurona de la capa oculta es

$$y_j(\boldsymbol{x}) = f(net_j) = f\left(\sum_0^n w_{ji} x_i\right), \tag{3.11}$$

donde w_{ji} es el peso que une la neurona i de la capa de entrada con la neurona j de la capa oculta. Sustituyendo obtenemos la salida de la red neuronal en función de la entrada:

$$z(\boldsymbol{x}) = f\left(\sum_{j=1}^{n_2} w_j f\left(\sum_0^{n_1} w_{ji} x_i\right)\right). \tag{3.12}$$

¿Qué funciones puede representar una red neuronal?

El teorema de Kolmogorov asegura que cualquier función continua puede expresarse en una red neuronal de 3 capas, si ponemos $2n + 1$ neuronas ocultas, y poniendo una función en cada neurona que puede no ser la misma. En la última capa habría una única neurona que sumaría las salidas de todas las neuronas ocultas.

Otra prueba del alcance de las redes neuronales es el teorema de Fourier, que dice que cualquier función continua puede aproximarse arbitrariamente cerca por una suma de funciones armónicas, posiblemente infinita.

Esto sería una red neuronal con la función identidad en las neuronas de la capa de entrada, una cantidad posiblemente muy grande de neuronas en la capa oculta, con dichas funciones armónicas, y una sola neurona en la capa de salida, con una función que sume las salidas de las funciones de la capa oculta.

Ninguno de los teoremas anteriores nos da ninguna pista sobre la cantidad de neuronas ocultas ni sobre los pesos correctos. Además, aunque existiese, una prueba constructiva nos sería de poca

ayuda, ya que normalmente no sabremos cómo es la función buscada. Sin embargo, estos resultados nos ayudan a pensar que el esfuerzo en esta búsqueda es razonable.

3.3.2. Algoritmo de retropropagación

El entrenamiento de una red neuronal consiste en introducir una observación (de la que conocemos el target) en la capa de entrada y según cuánto se aleje la predicción que produzca la red, ajustar los pesos para que se parezcan lo máximo posible.

Sin embargo, aunque podríamos modificar a ojo los pesos de la capa oculta a la capa de salida para que la predicción se pareciese más al valor poblacional, no sabemos cómo modificar los pesos de la capa de entrada a la oculta, ya que no sabemos que salidas deberíamos estar obteniendo por las neuronas de esta capa. Éste es el problema de asignación de crédito (*credit assignment problem*). El algoritmo de retropropagación es una de las soluciones, que vamos a estudiar a continuación.

Aprendizaje de la red

Lo primero que tenemos que hacer para que nuestro método haga buenas predicciones, es cuantificar cómo de buena es una predicción. Queremos medir la distancia entre cada predicción y el resultado poblacional, y para esto utilizamos el error cuadrático. Si fijamos el valor de entrada, el error cuadrático se puede ver como una función de todos los pesos de la red. La llamamos $J(\mathbf{w})$ (multiplicamos por $\frac{1}{2}$ para derivar más cómodamente):

$$J(\mathbf{w}) = \frac{1}{2}(t - z)^2, \tag{3.13}$$

siendo t el valor poblacional (target) y z el valor que ha predicho la red. $\mathbf{w}$ es el vector de todos los pesos de la red (weight). Buscamos el mínimo de esta función, es decir, la combinación de pesos que hay que seleccionar en la red para que las predicciones estén lo más cerca posible de los valores poblacionales.

El algoritmo de retropropagación consiste en intentar llegar a un mínimo local de $J(\mathbf{w})$ haciendo pequeñas modificaciones de $\mathbf{w}$ en la dirección contraria a la del vector gradiente en ese punto, que es aquella en la que más rápido decrece $J(\mathbf{w})$.

Las modificaciones serán proporcionales al error cuadrático, pues suponemos que si este es cercano a cero, ya estamos muy cerca del mínimo (que como mucho, será 0). Representamos el incre-

mento de **w** como

$$\Delta \mathbf{w} = -\eta \frac{\partial J}{\partial \mathbf{w}} \tag{3.14}$$

o, componente a componente:

$$\Delta w_{mn} = -\eta \frac{\partial J}{\partial w_{mn}}, \tag{3.15}$$

donde η indica el tamaño del incremento.

Para una red neuronal de una capa oculta, podemos obtener en pocos cálculos la expresión del incremento de los pesos. Los de la capa oculta a la capa de salida son:

$$\boxed{\Delta w_j = \eta(t-z)f'(net)y_j}\ . \tag{3.16}$$

De manera análoga, obtenemos la expresión del incremento de los pesos de la capa de entrada a la capa oculta:

$$\boxed{\Delta w_{ji} = \eta(t-z)f'(net)w_j f'(net_j)x_i}\ . \tag{3.17}$$

Para una red neuronal con más capas ocultas, los cálculos son idénticos, pero cuantas más capas de neuronas haya entre el peso y el target, más derivadas incluirá su fórmula.

Aunque es interesante tener una idea general de cómo se va incrementando cada peso, la implementación del método de retropropagación está disponible en diversos softwares y no es objeto de este TFG.

3.4. *Support Vector Regressor* (SVR)

La mayoría de esta sección ha sido obtenida de [3] y del capítulo 19 de [4].

Las máquinas de vectores soporte de regresión (siglas en inglés SVR) son otra solución al problema de regresión, que consiste en una función $f(\boldsymbol{x})$ que, en las observaciones del conjunto de aprendizaje, tenga un error menor que un ε definido, y que al mismo tiempo sea lo más cercana posible a una

función constante.

Si imaginamos la función poblacional, que es la que tratamos de aproximar con $f(\boldsymbol{x})$, e imaginamos también un tubo alrededor de su gráfica, de manera que los puntos de este tubo estén a una distancia ε de los puntos de la función, entonces nuestra función $f(\boldsymbol{x})$ estará dentro de este tubo. Esta es la primera de las condiciones que hemos pedido, es decir, que en las observaciones del conjunto de aprendizaje, el error sea menor que ε. La otra condición hace referencia a que la función sea lo más parecida a una recta posible. Esto significa que queremos que la función crezca o decrezca de la forma más constante posible, es decir, que su segunda variable sea cercana a cero, en valor absoluto.

Explicaremos primero el método que siguen las máquinas de vectores soporte para resolver un problema lineal, y luego introduciremos las modificaciones necesarias para resolver problemas no lineales.

Si f es lineal podemos escribirla como

$$f(\boldsymbol{x}) = \langle \boldsymbol{w}, \boldsymbol{x} \rangle + b \quad : \boldsymbol{w} \in \mathcal{X}, b \in \mathcal{R}. \tag{3.18}$$

En este caso, donde, sea cual sea f, su segunda derivada será 0, que sea cercana a una función constante lo entendemos como que $\|w\|$ sea cercano a 0. Encontrar entonces la f sería resolver el siguiente problema de optimización:

$$\text{mín} \quad \frac{1}{2}\|w\|^2 \quad : \quad \begin{cases} y_i - \langle w, x_i \rangle - b \leq \varepsilon \\ \langle w, x_i \rangle + b - y_i \leq \varepsilon \end{cases}, \tag{3.19}$$

donde, de todas las funciones que cumplen las restricciones impuestas, nos quedamos con aquella cuyo vector de coeficientes, w, tenga menor norma. Como la función que buscamos es la solución a un problema de optimización convexa, sabemos que siempre habrá solución.

En ocasiones queremos permitir cierta cantidad de error, es decir, queremos permitir que nuestra función no esté contenida enteramente en el tubo épsilon. En este caso, tenemos que modificar el problema e incluir las llamadas *slack variables*, ξ_i, ξ_i^*:

$$\text{mín} \quad \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N}(\xi_i + \xi_i^*) \quad : \quad \begin{cases} y_i - \langle w, x_i \rangle - b \leq \varepsilon + \xi_i \\ \langle w, x_i \rangle + b - y_i \leq \varepsilon + \xi_i^* \end{cases}. \tag{3.20}$$

La constante C, que es un hiperparámetro que tendremos que ajustar, balancea la importancia de que f sea cercana a una función constante y la cantidad hasta la que toleramos desviaciones mayores que ε. El valor que demos a C define en gran medida las predicciones que obtendrá nuestro modelo, por lo que es imprescindible que su valor sea adecuado. Para encontrarlo utilizaremos la validación cruzada, como definimos al principio del capítulo. Se explica más detalladamente en la sección de experimentos.

Otro detalle importante es que sólo las desviaciones mayores que ε tienen coste. El segundo término de la ecuación no diferencia entre funciones que estén contenidas en el tubo épsilon, sino que únicamente tiene en cuenta la parte de la función que sale del tubo. El primer término, en cambio, será el que nos haga elegir una función u otra de entre las que tienen el mismo error: elegiremos aquella con menores coeficientes, es decir, aquella para la que la norma del vector de coeficientes, w, sea menor.

Para simplificar la resolución del problema ecuación 3.20, se suele modificar su formulación a la denominada expresión dual. Sobre el problema dual únicamente veremos que será una función que construyamos a partir del problema que ya hemos definido, que se llama problema primal. A grandes rasgos, buscamos un máximo en lugar de un mínimo, modificando las restricciones e introduciendo el conjunto dual de variables, que son los multiplicadores de Langrange $\alpha_i^{(*)}$ y $\eta_i^{(*)}$:

$$\begin{aligned} \mathsf{L} :=& \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i) - \sum_{i=1}^{n}(\eta_i\xi_i + \eta_i\xi_i^*) - \\ &- \sum_{i=1}^{n}\alpha_i(\varepsilon + \xi_i - y_i + \langle \boldsymbol{w}, \boldsymbol{x_i}\rangle + b) - \\ &- \sum_{i=1}^{n}\alpha_i^*(\varepsilon + \xi_i^* + y_i - \langle \boldsymbol{w}, \boldsymbol{x_i}\rangle - b). \end{aligned} \tag{3.21}$$

Ahora, derivando respecto de las variables primarias $\boldsymbol{w}$, b y $\xi_i^{(*)}$, anulando y sustituyendo, obtenemos el problema de optimización dual

$$\text{máx} \begin{cases} -\frac{1}{2}\sum_{i,j=1}^{n}(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)\langle \boldsymbol{x_i}, \boldsymbol{x_j}\rangle \\ -\varepsilon\sum_{i=1}^{n}(\alpha_i + \alpha_i^*) + \sum_{i=1}^{n} y_i(\alpha_i - \alpha_i^*) \end{cases} : \quad \textstyle\sum_{i=1}^{n}(\alpha_i - \alpha_i^*) = 0 \quad \text{y} \quad \alpha_i, \alpha_i^* \in [0, C]$$

Y finalmente, con algunos cálculos más en las ecuaciones de las derivadas, podemos expresar $\boldsymbol{w}$ en función de los $\boldsymbol{x_i}$ por un coeficiente:

$$\boldsymbol{w} = \sum_{i=1}^{n}(\alpha_i - \alpha_i^*)\boldsymbol{x_i}, \tag{3.22}$$

y por tanto

$$f(\boldsymbol{x}) = \sum_{i=1}^{n}(\alpha_i - \alpha_i^*)\langle x_i, \boldsymbol{x}\rangle + b. \tag{3.23}$$

Vemos que $\boldsymbol{w}$ puede expresarse como una combinación lineal de las observaciones de nuestra muestra de datos. Además, el algoritmo consiste esencialmente en hacer productos escalares entre observaciones, y mientras conozcamos los resultados de estos, no es necesario conocer el valor de las

observaciones individualmente. Este detalle es importante y más adelante lo utilizaremos para poder resolver problemas no lineales.

Ahora nos falta calcular b. En este punto introducimos las condiciones de Karush-Kuhn-Tucker, a partir de las cuales acotaremos b, y concluiremos algunas observaciones más que revelan por qué las SVRs son tan potentes. Las condiciones de Karush-Kuhn-Tucker dicen que el vector de pesos $\boldsymbol{w}$ que sea solución de nuestro problema también será solución de las siguientes ecuaciones:

$$\alpha_i(\varepsilon + \xi_i - y_i + \langle \boldsymbol{w}, \boldsymbol{x_i} \rangle + b) = 0, \tag{3.24}$$

$$\alpha_i^*(\varepsilon + \xi_i^* + y_i - \langle \boldsymbol{w}, \boldsymbol{x_i} \rangle - b) = 0, \tag{3.25}$$

$$(C - \alpha_i)\xi_i = 0 \tag{3.26}$$

y

$$(C - \alpha_i^*)\xi_i^* = 0. \tag{3.27}$$

A partir de estas condiciones vemos que las observaciones que se salgan del tubo épsilon son las únicas para las que $\alpha_i - \alpha_i^*$ es no nulo, por lo tanto, en la expresión de $\boldsymbol{w}$, las únicas observaciones que no vamos a multiplicar por un coeficiente igual a cero son las que se salgan del tubo. Obviamente no nos hace falta conocer las observaciones que se vayan a multiplicar por cero, con lo que $\boldsymbol{w}$ solo depende las observaciones que salgan del tubo épsilon. A estas observaciones las llamamos vectores soporte, y nos permiten no tener en cuenta el conjunto completo de datos cada vez que queramos realizar una predicción, lo cual supone que podamos aplicar las SVRs en problemas con un conjunto de datos de gran tamaño.

Por otro lado, a partir de estas cuatro condiciones vemos que podemos acotar b en el siguiente intervalo:

$$\begin{gathered}\text{máx}\{-\varepsilon + y_i - \langle w,x_i \rangle \mid \alpha_i < C \text{ o } \alpha_i^* > 0\} \leq \\ \leq b \leq \\ \leq \text{mín}\{-\varepsilon + y_i - \langle w,x_i \rangle \mid \alpha_i > 0 \text{ o } \alpha_i^* < C\},\end{gathered}$$

y en el caso de que algún $\alpha^{(*)}$ pertenezca al intervalo $(0, C)$, las desigualdades pasan a ser igualdades, con lo que tenemos el valor exacto de b.

Ya conocemos un procedimiento para encontrar $f(\boldsymbol{x})$ en un caso lineal, y hemos visto además que solo depende de los vectores soporte. A continuación, vamos a ver como adaptar $f(\boldsymbol{x})$ para problemas no lineales.

Kernel trick

Como se ha hecho notar anteriormente, para escribir $f(x)$ no es necesario conocer el valor de las observaciones individualmente, sino que solo nos interesaba conocer el producto escalar de unas observaciones con otras. Esto nos va a permitir aplicar una transformación no lineal a los datos para convertirlos en unos que sí se puedan aproximar con una función lineal.

Supongamos que queremos aplicar una transformación no lineal ϕ a nuestros datos. Dado que solo necesitamos conocer $\langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle$, tras aplicar la transformación solo necesitamos saber $\langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j) \rangle$, y para ello podemos utilizar las llamadas funciones kernel K, que cumplen $K(\boldsymbol{x}_i, x_j) = \langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j) \rangle$, para determinada ϕ. De esta manera podemos calcular los productos escalares sin conocer la imagen por ϕ, y solo tenemos que calcular $K(\boldsymbol{x}_i, x_j)$.

Hay distintos ejemplos de funciones Kernel, pero la más común y por tanto la que se ha utilizado en este trabajo es la función kernel de base radial, o, según sus siglas en inglés, kernel *rbf*. Tiene la siguiente expresión:

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp\left(-\frac{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2}{2\sigma^2}\right), \tag{3.28}$$

Los valores que toman sus distintos parámetros se hallarán utilizando validación cruzada, como se explica en la sección de simulaciones y resultados, en la sección 5.3, que es la del modelo SVR.

4

Métodos de ensemble

4.1. ¿Qué es un modelo de ensemble?

Los métodos de ensemble de regresión son un tipo de modelo de regresión compuesto de varios modelos individuales, que se entrenan para resolver el mismo problema y son combinados para obtener mejores resultados. En este sentido, con el término modelo individual nos referimos a un modelo que no está compuesto por otros, como sí que lo están los modelos de ensemble. Los modelos ridge, perceptrón multicapa y SVR son modelos individuales.

Los modelos de ensemble dan una solución más elaborada que la de otros métodos al problema de elegir con qué modelos de regresión individuales haremos predicciones, de entre una familia de N modelos $\{G_j\}$, con $N \geq 1$. Por ejemplo, en validación cruzada elegimos cuál de todos los modelos obtiene las mejores predicciones, y se utilizará ese descartando todos los demás.

Los modelos de ensemble en cambio involucran a la familia completa de modelos de regresión individuales, $\{G_j\}$, buscando un modelo más potente a expensas de un coste computacional algo mayor. La idea subyacente es que los fallos de cada modelo se compensen con los demás, ya que la porción de datos donde un modelo comete mayor error no tiene por qué ser la misma que para otro modelo.

Para encontrar una solución a un problema de regresión hacemos un balance de la complejidad del modelo: deberá tener suficientes grados de libertad como para ser capaz de reproducir la complejidad de las relaciones entre las distintas variables, pero no demasiados grados de libertad, o será capaz de replicar con demasiada precisión los resultados de los datos con los que se entrene, resultando esto en sobreajuste. Este balance es uno de los aspectos que buscamos mejorar al componer varios modelos en un modelo de ensemble.

Cuando un modelo sobreajusta, normalmente tiene una varianza muy alta, pues si los datos son muy similares a los del conjunto de entrenamiento, el modelo obtendrá buenas predicciones, pero en cuanto estos sean algo diferentes, el modelo no sabrá generalizar lo suficiente como para obtener

buenos resultados. Por lo tanto, queremos reducir la varianza de los modelos que sobreajusten.

En cambio, cuando un modelo generaliza demasiado, tiene menor varianza, y los resultados suelen ser más parecidos al variar el conjunto de datos, pero al precio de que las predicciones tengan un mayor sesgo. En estos casos, por lo tanto, querríamos reducir el sesgo.

Utilizando los métodos de ensemble podemos construir un método con menor sesgo o menor varianza que obtenga mejores resultados. Tendremos por lo tanto que utilizar el modelo de ensemble que coincida con nuestras necesidades, pues unos están más orientados a reducir la varianza de los modelos que lo componen y otros a reducir el sesgo.

El primer paso para trabajar con métodos de ensemble es seleccionar los modelos base que vamos a utilizar. La mayoría de las veces (bagging, boosting) utilizaremos un único modelo y variaremos sus hiperparámetros y/o el conjunto de aprendizaje. Entonces decimos que el método de ensemble es homogéneo. En los modelos de ensemble stacking en cambio se utilizan distintos modelos individuales. Entonces decimos que el modelo de ensemble es heterogéneo. Esto supone muchas más posibilidades, y por lo tanto pueden obtenerse mayores beneficios, pero hay también una dificultad implícita: no todos los modelos funcionarán bien juntos, y encontrar la combinación adecuada puede resultar complicado. A continuación pasamos a una explicación general de cada uno de ellos.

Bagging

Para definir bagging antes es necesario definir el bootstrapping, que es un método de remuestreo en el que generamos muestras de tamaño k a partir de un conjunto de datos de tamaño n, haciendo extracciones aleatorias con reemplazo. Este método tiene la ventaja de que, bajo las hipótesis de representatividad e independencia, podemos considerar que las extracciones se hacen directamente de la distribución subyacente.

La hipótesis de representatividad consiste en asumir que n es lo suficientemente grande como para reflejar la complejidad de la distribución subyacente.

La hipótesis de independencia consiste en asumir que n es lo suficientemente grande en comparación con k como para poder asumir que las muestras no tendrán correlación. Podremos considerar las observaciones casi independientes idénticamente distribuidas.

Bagging es un método de ensemble que consiste en entrenar varios modelos individuales independientes y promediar sus predicciones para obtener un modelo con menos varianza. Al entrenarse los modelos individuales de manera independiente, tenemos la posibilidad de entrenarlos en paralelo, lo que puede suponer una reducción muy significativa del tiempo de ejecución.

El procedimiento consiste en extraer una muestra bootstrap para cada modelo, que será su con-

junto de aprendizaje. La predicción del modelo bagging se obtiene haciendo un promedio entre las predicciones de los modelos individuales.

Hacer un promedio de las predicciones de los modelos individuales no cambia la respuesta esperada, pero reduce la varianza, de la misma manera que hacer la media de variables aleatorias i.i.d. preserva el valor esperado pero reduce la varianza.

Random forests es el ejemplo más conocido de modelo de ensemble bagging, donde se combinan árboles de decisión.

Boosting

Boosting es un método de ensemble que consiste en entrenar varios modelos individuales secuencialmente, influyendo el entrenamiento de un modelo en el de los modelos posteriores. Conseguimos un modelo con menor sesgo, aunque en ocasiones el coste temporal es muy elevado.

Para que el coste temporal sea razonable, se suelen utilizar modelos individuales poco precisos que se entrenen rápidamente. Cada modelo se entrena dando más prioridad a las observaciones que han sido mal clasificadas previamente.

La predicción del modelo de ensemble bagging será una suma ponderada de las predicciones de los modelos individuales. Normalmente buscar una ponderación que dé buenos resultados no es inmediato.

Adaptative boosting y *gradient boosting* son dos maneras distintas de construir un modelo de ensemble bagging, que se diferencian en la manera de encontrar los coeficientes de la suma ponderada.

Stacking

Dado que emplearemos este modelo en los experimentos, no haremos una explicación tan general como la de bagging y boosting, sino que le dedicaremos más tiempo en la siguiente sección.

4.2. Método de ensemble *stacking*

La mayoría de esta sección ha sido obtenida a partir de [5].

En esta sección vamos a definir una técnica utilizada para construir métodos de ensemble conocida como *stacking* o *stacked generalization*. Como ya hemos visto, los modelos de ensemble stacking tienen la particularidad de que pueden componerse de modelos de distinto tipo.

A modo de adelanto para facilitar la comprensión de este modelo de ensemble, damos la siguiente comparación entre el funcionamiento de un modelo individual y uno stacking:

Un modelo individual tradicional se entrena sobre un conjunto de datos que vive en un espacio, digamos el espacio de nivel 0, y luego es capaz de predecir el target de nuevas observaciones que también vivan en el espacio de nivel 0.

Un modelo stacking que utilice un conjunto de N modelos de regresión individuales, hace una transformación de los datos del espacio de nivel 0 para conseguir datos en un espacio de nivel 1, transformación que se lleva a cabo utilizando cierto subconjunto de los N modelos. A continuación, entrena un modelo sobre los datos de nivel 1, que por lo tanto podrá realizar predicciones sobre datos que vivan en el nivel uno. Como los datos sobre los que queremos utilizar el modelo viven en el espacio de nivel 0, antes de predecir tendremos que llevar estos datos al nivel 1, aplicar el modelo, y finalmente tomar la predicción de nivel 1 que obtengamos y llevarla de vuelta al nivel 0.

Llamaremos espacio de nivel 0 al espacio donde viven los elementos de D_n, y espacio de nivel 1 a donde viven las transformaciones de los datos del nivel 1. Además, de la familia de N modelos de regresión individuales, tomaremos un subconjunto de k modelos, que llamaremos los modelos de nivel 0, y otro modelo, que puede estar incluido en los modelos de nivel 0, será el modelo de nivel 1, y denotaremos por $\widetilde{G}$. Los modelos de nivel 0 se llaman así porque predicen sobre datos del nivel 0, y ocurre igual con el nivel 1.

Para explicar cómo se hace la transformación que genera las observaciones del nivel 1, tomamos un elemento genérico del espacio de nivel 1, digamos $(\widetilde{\boldsymbol{X}}_i, \widetilde{Y}_i)$, con $\widetilde{\boldsymbol{X}}_i = (\widetilde{X}_{i1}, \widetilde{X}_{i2}, \ldots, \widetilde{X}_{ik})$, y veremos como obtenerlo a partir de la observación i del espacio de nivel 1 $(\widetilde{\boldsymbol{X}}_i, \widetilde{Y}_i)$.

El atributo j de $\widetilde{\boldsymbol{X}}_i$, es decir, $\widetilde{X}_{ij}$, es la predicción del modelo j de la familia de modelos de nivel 0 del target de $\boldsymbol{X}_i$, al entrenarse sobre todo el resto de D_{train}. Por lo tanto, para obtener todos los atributos de $\widetilde{\boldsymbol{X}}_i$ aplicamos los k modelos de nivel 0 sobre una observación del espacio de nivel 0.

El target de la observación $(\widetilde{\boldsymbol{X}}_i, \widetilde{Y}_i)$, es decir, $\widetilde{Y}_i$, es simplemente Y_i. En algunas variaciones del método stacking también se hace una transformación de $\widetilde{Y}_i$ para obtener $\widetilde{\boldsymbol{Y}}_i$, pero en este trabajo, Y_i e $\widetilde{Y}_i$ son iguales, o lo que es lo mismo, se utiliza la identidad como transformación.

En la figura 4.1 tenemos un esquema de la transformación de una observación de nivel 0 a una de nivel 1, donde los k modelos de nivel 0 se entrenan, como hemos dicho, con $D_{train} \setminus (\boldsymbol{X}_i, Y_i)$ para obtener $(\widetilde{\boldsymbol{X}}_i, \widetilde{Y}_i)$ a partir de $(\boldsymbol{X}_i, Y_i)$.

Por lo tanto, tendremos tantos elementos en el espacio de nivel 1, que podemos llamar $\widetilde{D_{train}}$, como en el espacio de nivel 1, y tendrán tantos atributos como modelos de nivel 0 tenga nuestro modelo de ensemble.

Hasta aquí se ha definido el conjunto de datos de nivel 1 sobre el que se entrena el modelo de nivel 1. Falta por definir cómo llevar una observación a dicho nivel para predecir su target, y dado que no hemos modificado los targets para pasar del nivel 0 al nivel 1, el target proporcionado por el modelo de nivel 1 será el que buscamos.

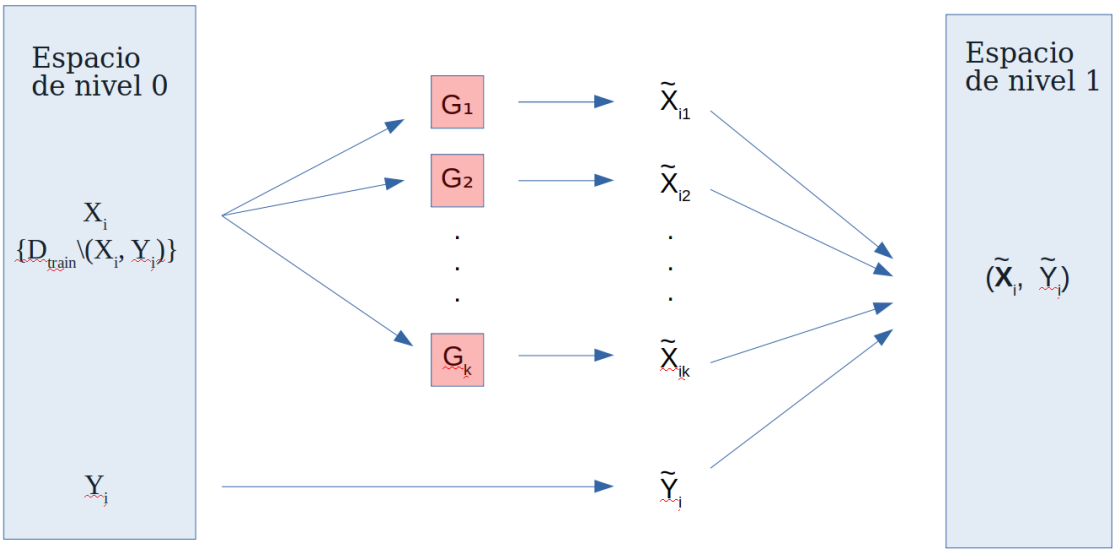

Figura 4.1: Este esquema representa cómo se obtiene una observación del espacio de nivel 1 a partir de una observación del espacio de nivel 0 utilizando los k modelos de nivel 0.

Sea $\boldsymbol{X}$ una observación en el espacio de nivel 0 y $\widetilde{\boldsymbol{X}} = (\widetilde{X}_1, \widetilde{X}_2, \ldots, \widetilde{X}_k)$ su transformación al espacio de nivel 1. Entonces el atributo i de la observación de nivel 1 es la predicción del target de la observación $\boldsymbol{X}$ mediante el modelo i del subconjunto de modelos de nivel 0, cuando se entrena con el conjunto completo de datos, D_n.

En la figura 4.2 vemos el proceso de predicción del target de una observación $\boldsymbol{X}$. Esta vez los k modelos de nivel 0 se entrenan sobre todo D_{train}.

Este es el proceso completo, que puede iterarse para obtener p niveles, con $p \geq 1$. Sin embargo, cuantos más niveles tenga el modelo de ensemble, mayor será el coste computacional.

A día de hoy no hay reglas generales que indiquen qué generalizadores utilizar para cada nivel, ni con qué proceso obtener los k números a partir del conjunto de aprendizaje del nivel i que formen las componentes de entrada del nivel $i + 1$, etc. La manera de proceder habitualmente es aplicar conocimiento específico del problema para seleccionar estos hiperparámetros.

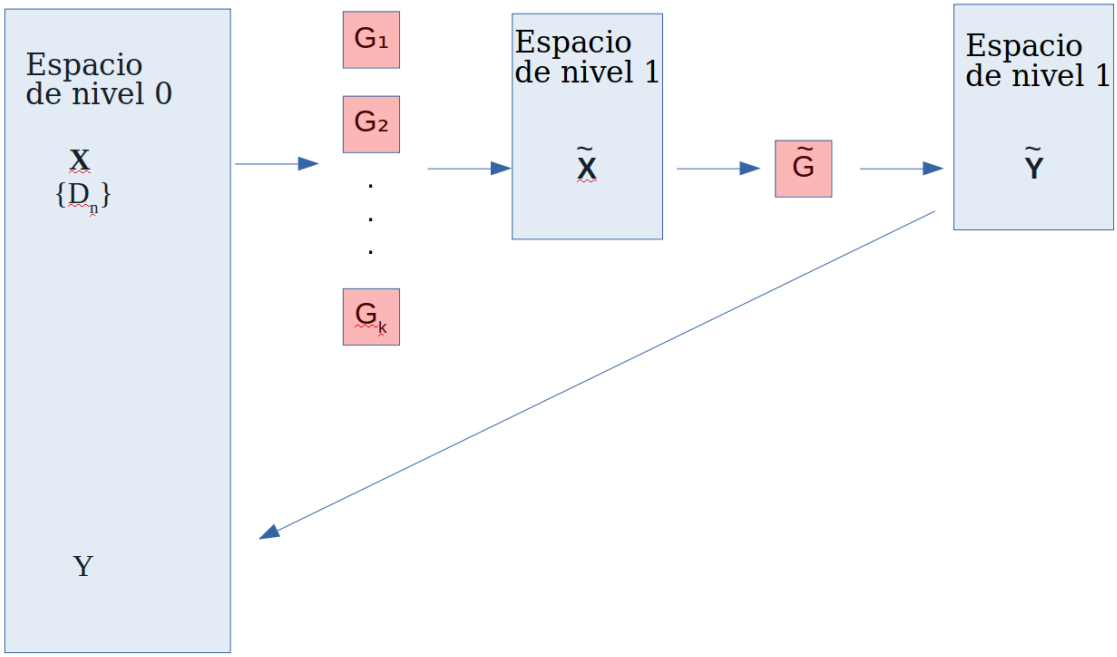

Figura 4.2: Este esquema representa el proceso de predicción una vez entrenado el modelo de ensemble.

5

Simulaciones y resultados

En este capítulo explicamos cómo hemos utilizado cada uno de los modelos vistos hasta ahora para predecir la cantidad de energía eólica obtenida en el Parque Eólico Experimental de Sotavento.

Datos utilizados

La información sobre los datos ha sido obtenida de [6].

Para realizar este trabajo contábamos con una cuenta en el Centro de Computación Científica de la Universidad Autónoma de Madrid, a partir de la cual teníamos acceso a 6 ficheros de datos en los que se encontraban las predicciones atmosféricas del Centro Europeo de Previsiones Meteorológicas a Plazo Medio y las medidas de la energía obtenida en el Parque Eólico de Sotavento de cada hora de cada día de los años 2016, 2017 y 2018.

Los datos de las predicciones atmosféricas consisten en una serie de medidas de la dirección del viento, la velocidad, presión y temperatura, que enseguida concretaremos; tomadas en distintos puntos de Galicia. Estos puntos constituyen una rejilla que está aproximadamente centrada en el parque de Sotavento, y que tiene 435 puntos.

A partir de la intuición de que la cantidad de energía que iba a obtener el parque de Sotavento dependía sobretodo de las condiciones meteorológicas en sus alrededores más próximos, y que no era necesario por lo tanto tener en cuenta estas previsiones en un territorio tan amplio como del que disponíamos en principio, optamos por recortar los datos. Esto nos ha permitido reducir el coste computacional del entrenamiento de los modelos. Nos quedaremos únicamente con los puntos que rodean al parque de Sotavento, que está situado en las coordenadas (43.354377º, -7.881213º).

La coordenada más cercana, dado que la resolución de la rejilla es de 0,125º, es (43.375º -7.875º), y los puntos que estudiaremos, por tanto, son (43.5, -8.), (43.5, -7.875), (43.5, -7.75), (43.375, -8.), (43.375, -7.875), (43.375, -7.75), (43.25, -8.), (43.25, -7.875), (43.25, -7.75).

Las medidas de las que disponemos son:

1.– Componente este de la dirección del viento medida a 10 metros de altura en m/s.

2.– Componente norte de la dirección del viento medida a 10 metros de altura en m/s.

3.– Módulo de la velocidad del viento medida a 10 metros de altura en m/s.

4.– Componente este de la dirección del viento medida a 100 metros de altura en m/s.

5.– Componente norte de la dirección del viento medida a 100 metros de altura en m/s.

6.– Módulo de la velocidad del viento medida a 100 metros de altura en m/s.

7.– Presión en la superficie en pascales.

8.– Temperatura medida a 2 metros de altura en kelvin.

Dado que vamos a realizar una estandarización de los datos, no es necesario prestar mucha atención a las unidades.

En total tenemos 8 medidas tomadas en cada uno de los 9 puntos de la rejilla, es decir, 72 variables. Como la energía obtenida en el parque de Sotavento se mide cada hora, y tenemos los datos de tres años (2016 fue bisiesto), en total tenemos $24 \times 365 \times 3 + 24 = 26304$ observaciones.

En la figura 5.1 vemos un diagrama de caja de la energía obtenida en el año 2018, es decir, del conjunto de test. Vemos que el bigote inferior es muy corto: esto se debe a que hay una alta cantidad de periodos en los que la producción de energía fue nula, debido a que los molinos no estaban funcionando. Desconocemos el motivo por el que esto es así, pero, a la vista de los resultados, parece lógico asumir que no depende de las condiciones atmosféricas, sino que quizá se deba a algún fallo mecánico o a mantenimiento en general. Por lo tanto, los periodos de producción nula, a efectos prácticos, aleatorios, son una desventaja que tenemos que asumir desde el principio.

Continuando con el diagrama de cajas 5.1, vemos que la mediana es muy baja, debido en gran parte al peso de las veces que los molinos no han estado funcionando.

Por último, vemos una gran densidad de valores atípicos elevados. En general la idea que podemos ir sacando en claro de este diagrama es que los datos son bastante cambiantes.

Las gráficas 5.2 y 5.3 son dos ejemplos de días en los que hay producción nula de energía. En el gráfico del día 1 de diciembre de 2018 encontramos un ascenso de 8000 vatios en un periodo de unas 10 horas, mientras que en el segundo gráfico, vemos un ascenso de 5000 vatios en apenas 3 horas.

Lectura de los datos

Para leer los datos, que están en formato csv (valores separados por comas), hemos utilizado la librería pandas, que nos permite trabajar con objetos *dataframe*. Los dataframe tienen una funcionalidad que hemos encontrado muy completa y que ha facilitado enormemente el análisis de los datos y resultados.

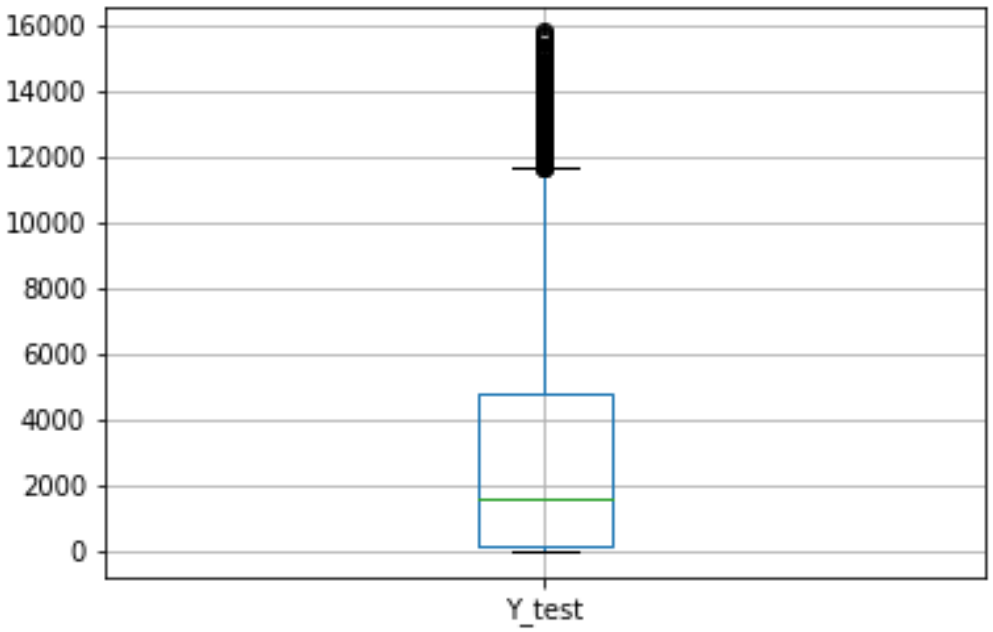

Figura 5.1: Boxplot de los valores de energía obtenida en el año 2018. Este es nuestro conjunto de test.

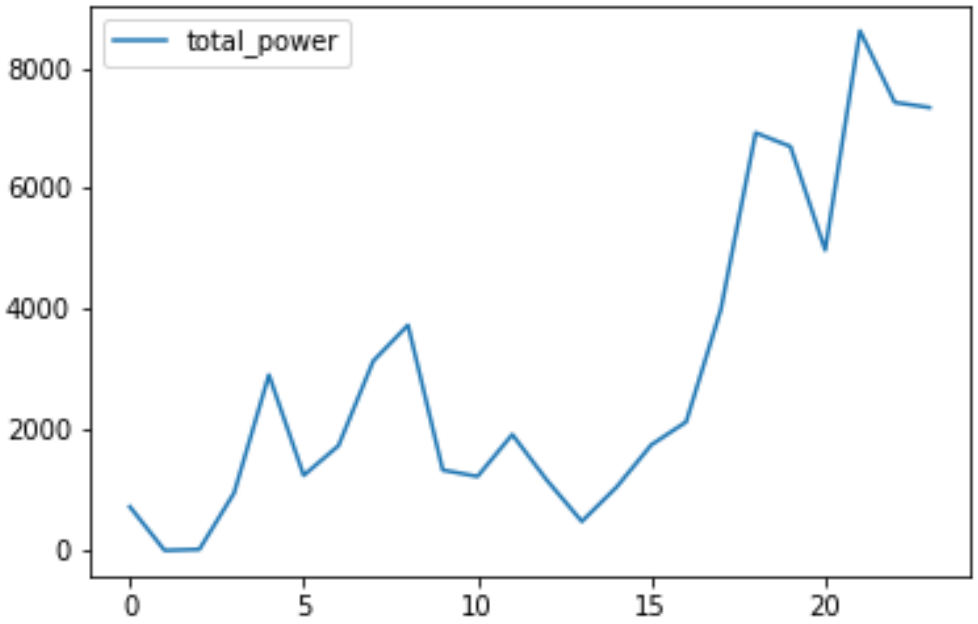

Figura 5.2: Gráfica de los valores de energía obtenida el 1 de diciembre de 2018.

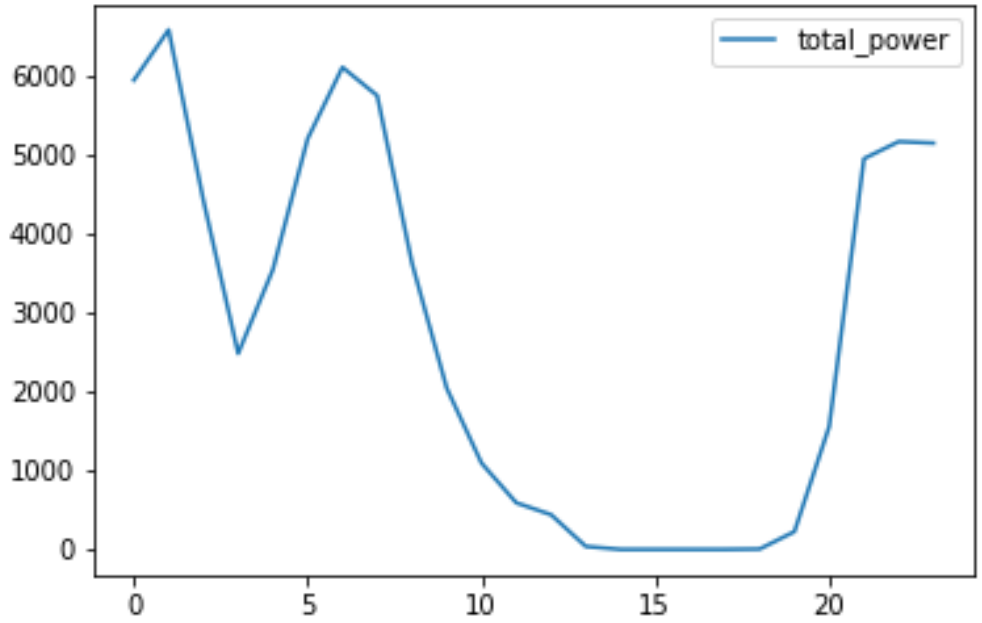

Figura 5.3: Gráfica de los valores de energía obtenida el 30 de diciembre de 2018.

Con la función *read_csv* de pandas, pasando como parámetro el nombre del archivo, creamos un dataframe con tantas columnas como datos. A partir de aquí es fácil realizar un filtrado en el que mantengamos únicamente los atributos cuyas coordenadas pertenezcan al intervalo que hemos recortado. Llegados a este punto tenemos los datos que vamos a utilizar, y hay que separarlos en un subconjunto que utilizaremos como datos de entrenamiento y otro subconjunto que utilizaremos como datos de test.

Conjunto de entrenamiento y conjunto de test

Una vez terminado el modelo de ensemble que construimos a lo largo de este trabajo, se entrenará con todos los datos de los que disponemos, es decir, las predicciones atmosféricas y la correspondiente cantidad de energía eólica obtenida de los años 2016, 2017 y 2018, y se empleará para predecir la cantidad de energía que se obtendrá en el futuro. De esta manera, utilizaremos el modelo entrenándolo con datos que son anteriores a las observaciones para las que queremos hacer predicciones.

Intuimos que este detalle puede ser importante: queremos entrenar un modelo y probarlo en condiciones lo más parecidas posible a aquellas en las que se utilizará. La manera de incorporar esta información a nuestro modelo es utilizar como conjunto de entrenamiento los años 2016 y 2017, y utilizar como conjunto de test los datos del año 2018, de manera que las observaciones sobre las que hagamos predicciones sean posteriores a aquellas sobre las que entrenamos el modelo.

Separamos entonces en los siguientes subconjuntos:

- X_train: predicciones meteorológicas de 2016 y 2017.
- Y_train: energía obtenida en 2016 y 2017.
- X_test: predicciones meteorológicas de 2018.
- Y_test: energía obtenida en 2018.

5.1. Regresión regularizada

Para el modelo de regresión regularizada hemos utilizado la función *Ridge* del módulo *linnear_model* de la librería *scikit learn* (sklearn). Respecto a los parámetros, hemos utilizado como *solver* el mínimo error cuadrático medio, es decir, "lsqr"(del inglés *lowest squares*), ya que es lo que corresponde al modelo que hemos estudiado. Además, suele ser más rápido que las alternativas. Los demás parámetros se han dejado con los valores por defecto, a excepción del parámetro α.

El parámetro α corresponde al peso de la regularización, es decir, al parámetro λ en nuestra explicación del modelo en la sección 3.2. Para seleccionar su valor hemos utilizado la validación cruzada, mediante la función *GridSearchcv* del módulo *model_selection* de la librería sklearn. Los parámetros que hemos utilizado son el modelo ridge que acabamos de describir en el parámetro *estimator*, y como rango de valores hemos utilizado potencias de 10 desde 0 hasta 6.

Al escoger el rango de valores hay que intentar coger un rango no excesivamente amplio, especialmente si el modelo tarda en entrenarse (aunque por ahora, este no es el caso), pero lo suficientemente grande para observar en qué parte del rango están los parámetros que mejor funcionan. Si, por ejemplo, escogemos un rango desde 10 hasta 10000, y el modelo va obteniendo mejores resultados a medida que el valor del parámetro aumenta, y además el mejor resultado se obtiene en el valor 10000, es decir, en el borde del intervalo, es obvio que hay una tendencia a obtener mejores resultados al aumentar el parámetro. En esta situación tenemos que aumentar el intervalo, o al menos moverlo, para incluir valores que hay a la derecha del máximo, pues es muy probable que el modelo siga mejorando. Nos detenemos cuando haya signos claros de que el modelo deja de mejorar.

En nuestro caso, el modelo obtiene resultados prácticamente iguales desde el 10^0 hasta el 10^3, y luego comienzan a mejorar rápidamente en 10^4 y 10^5, para luego obtener el peor resultado de todos en 10^6. Si hubiésemos utilizado un rango de parámetros hasta 10^4, el mejor resultado hubiese estado en el borde del intervalo, hubiésemos renunciado a obtener mejores resultados con 10^5, e incluso si el intervalo llegase solo hasta 10^5, no tendríamos la certeza de que estamos en el mejor valor, ya que no habríamos comprobado que a partir de ese valor los resultados son peores.

Además, hemos utilizado como scoring la estrategia *neg_mean_absolute_error*. Esto significa que

α	10^0	10^1	10^2	10^3	10^4	10^5	10^6
ranking	6	5	4	3	2	1	7
pts. test	-2801,20	-2801,20	-2801,18	-2800,95	-2798,76	-2781,95	-2822,19

Tabla 5.1: Resultados de la validación cruzada para el modelo Ridge sobre el parámetro α.

el ranking y las puntuaciones de los modelos corresponden al error absoluto medio que tiene el método ridge con cada uno de los parámetros del rango que hemos proporcionado. En la tabla 5.1, la tercera fila corresponde a la puntuación media de cada modelo, midiendo el error absoluto en negativo. La razón por la que medimos en negativo es porque la función GridSearchCV siempre maximiza, y como nosotros queremos seleccionar el parámetro para el que el modelo tenga error mínimo, escribimos los errores en negativo (los errores siempre serán una cantidad mayor o igual que cero) y escogemos el mayor.

Un detalle importante es que la validación cruzada se realiza siempre sobre el conjunto de entrenamiento. Esto es para que luego al predecir sobre el conjunto de test los resultados se parezcan a lo que podemos esperar al predecir sobre datos no conocidos. Si incluyésemos el conjunto de test en la validación cruzada, luego no podríamos utilizarlo para evaluar el comportamiento del modelo con datos nuevos.

En la tabla 5.1 vemos que el parámetro que menor error absoluto tiene sobre el conjunto de entrenamiento es $\alpha = 10^5$. Entrenamos entonces un modelo ridge con ese parámetro α, y lo entrenamos con el conjunto de entrenamiento mediante la función *fit*, a la que se le pasan como parámetros las observaciones del conjunto de entrenamiento y sus targets. A continuación, predecimos con él los targets de las observaciones del conjunto de test, y ya podemos medir los errores frente a los targets que tenemos en el conjunto de test.

El error absoluto medio que se ha obtenido prediciendo sobre el conjunto de test, es decir, las predicciones atmosféricas de 2018, es de 0,0866, es decir, $8,66\,\%$. Este porcentaje lo medimos respecto a la potencia máxima que puede obtener el parque en una hora, que son 17560 vatios. El error medio en valor absoluto es entonces entorno a los 1500 vatios.

Obtenemos además un valor para el estadístico R^2 de $46,8\,\%$. Esto quiere decir que la varianza de las predicciones atmosféricas explican un $46,8\,\%$ de la varianza de la energía obtenida.

Obtenemos la distribución de error que vemos en la gráfica 5.4. La información que se muestra es la diferencia *valor real - valor predicción*. Podemos observar que la mediana está por debajo de cero, y que hay valores atípicos negativos de cerca de 17560 vatios (corresponden al -1). Estos son debidos probablemente a los periodos de tiempo de producción nula, que el modelo no ha conseguido aprender. Sin embargo, la caja que corresponde al rango intercuartílico es muy achatada, lo que nos indica que hay una gran cantidad de datos para los que se obtiene un error de $\pm 5\,\%$. Dada la simplicidad del modelo y la complejidad de los datos, consideramos que estos son buenos resultados.

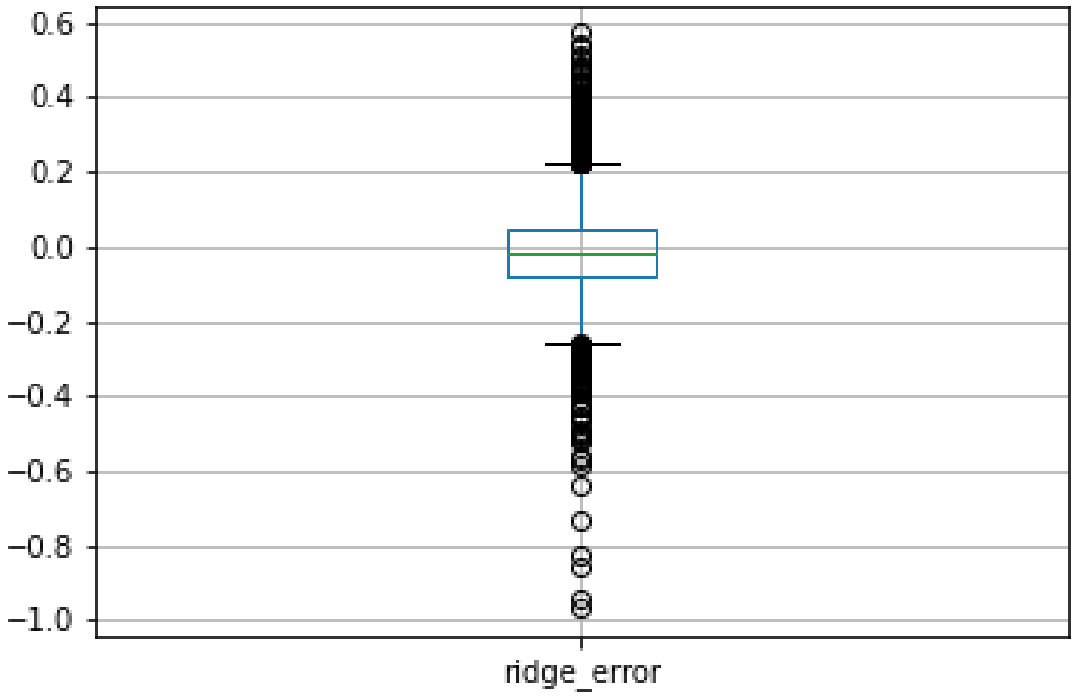

Figura 5.4: Boxplot de los errores medidos como valores reales menos valores predichos del modelo ridge.

5.2. Perceptrón multicapa

Para el perceptrón multicapa hemos utilizado la función *MLPRegressor* del módulo *neural_network* de sklearn. En el parámetro activation, hemos seleccionado funciones de activación ReLU para las neuronas de las capas ocultas. Los pesos se inicializan aleatoriamente. Inicialmente pretendíamos utilizar una red neuronal de una capa oculta, pero tras numerosas pruebas hemos determinado que los resultados con tres capas ocultas eran mejores. Por tanto indicamos tres capas ocultas de 100 neuronas cada una en el parámetro *hidden_layer_sizes*, indicamos 100, 100, 100. Los demás parámetros los dejamos con la configuración por defecto, a excepción del parámetro de regularización α, para el que realizamos una validación cruzada con GridSearchCV.

En la figura 5.2 vemos los resultados de la validación cruzada. El rango es lógico, pues es lo suficientemente grande como para permitir variedad de resultados, y tenemos evidencia para considerar que el valor 10^4 es un mínimo local, ya que si damos al parámetro α un valor mayor o menor, los resultados empeoran según nos alejamos.

$\boldsymbol{\alpha}$	10^1	10^2	10^3	10^4	10^5	10^6
ranking	5	4	3	1	2	6
pts. test	-1048,44	-1044,96	-1044,92	-1033,40	-1043,27	-1049,27

Tabla 5.2: Resultados de la validación cruzada para el perceptrón multicapa sobre el parámetro α.

Para este modelo hemos obtenido un error absoluto medio de $6,5\,\%$, es decir, hemos mejorado considerablemente las predicciones que obteníamos con el modelo ridge, con el que fallábamos en un $8,66\,\%$. Este $6,5\,\%$ equivale a unos 1150 vatios, frente a los 1500 del modelo ridge.

El estadístico R^2 en este caso tiene un valor de $71,31\,\%$, que, frente al $46,8\,\%$ del modelo ridge, supone una mejora de caso el $25\,\%$.

En la figura 5.5 vemos el diagrama de caja con los errores de predicción del perceptrón multicapa sobre las predicciones meteorlógicas de 2018. Podemos apreciar que el rango intercuartílico vuelve a ser muy pequeño, y que la mediana parece ser también muy cercana a cero. Los valores atípicos vuelven a ser más extremos cuando el valor predicho supera al valor real, debido a los preriodos de producción nula. Podemos apreciar también una disminución de la longitud de los bigotes: buena señal ya que indica más cantidad de errores cercanos a cero.

5.3. SVR

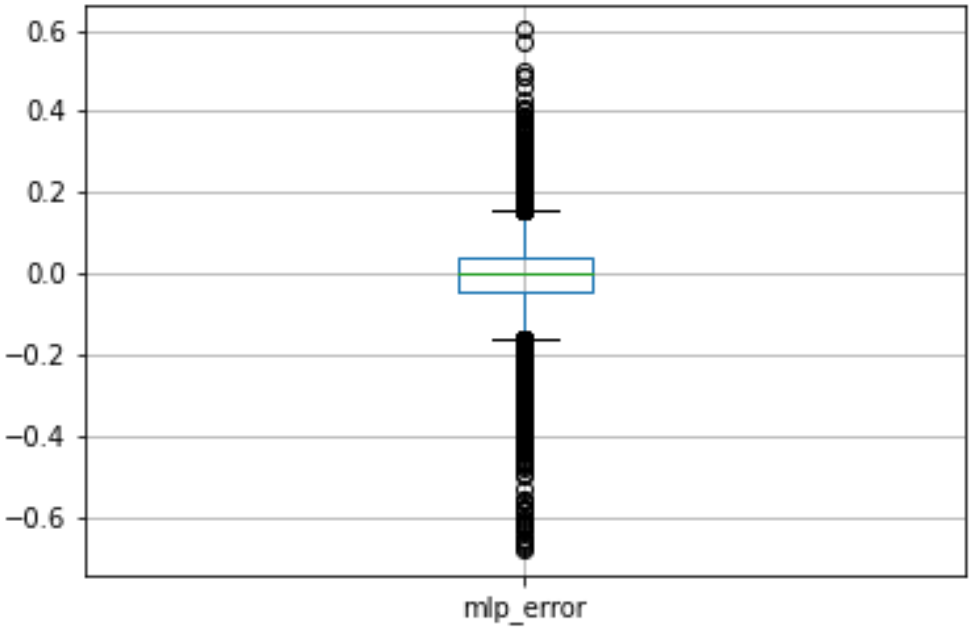

Figura 5.5: Errores del perceptrón multicala medidos como valores predecidos menos valores reales del perceptrón multicapa.

Para el modelo de la máquina de vectores soporte de regresión, SVR, hemos utilizado la función *SVR* del módulo *svm* de la librería sklearn. Hemos utilizado la validación cruzada para los parámetros *epsilon*, *C* y *gamma*, que explicaremos a continuación. Hemos utilizado el kernel *rbf*, es decir, función de base radial (*radial basis function*), sin máximo de iteraciones y con tolerancia 10^{-3}. Los demás parámetros se han dejado con los valores por defecto.

El parámetro C hace referencia a la regularización. Cuanto menor sea su valor, más impacto tendrá la regularización. El parámetro gamma corresponde al parámetro σ de la función de base radial que utilizamos como kernel, según está escrita en la ecuación 3.28. Por último, el parámetro épsilon hace referencia al tamaño del tubo épsilon que describimos en la sección de SVRs 3.4.

Para la validación cruzada hemos configurado GridSearchCV según se hace en el artículo [6]. Los intervalos que se proponen en dicho artículo son los siguientes:

$$C := \{10^k : -1 \leq k \leq 6\}, \tag{5.1}$$

$$\varepsilon := \{\frac{\sigma}{2^k} : 1 \leq k \leq 6\}, \tag{5.2}$$

$$\gamma := \{\frac{4^k}{d} : -2 \leq k \leq 3\}, \tag{5.3}$$

donde σ es la desviación típica de la energía obtenida en el conjunto de entrenamiento, es decir, de 2016 y 2017; y *d* es la cantidad de variables que tiene el problema. σ se puede obtener mediante la función *std()* del objeto DataFrame, y la cantidad de variables o dimensión del problema, con la función *shape()* del objeto dataframe.

Al intentar utilizar estos intervalos, el modelo tardaba demasiado tiempo en ejecutarse, y, dado que el tiempo de ejecución en el servidor del Centro de Computación Científica donde hacíamos las pruebas es limitado, no era posible utilizarlos sin una adaptación.

Haciendo distintas pruebas con varias adaptaciones de los intervalos, la que terminó de ejecutarse siendo más parecida a la original fue para los siguientes rangos:

$$C := \{10^k : 3 \leq k \leq 6\}, \tag{5.4}$$

$$\varepsilon := \{\frac{\sigma}{2^k} : 3 \leq k \leq 6\}, \tag{5.5}$$

$$\gamma := \{\frac{4^k}{d} : -2 \leq k \leq 2\}, \tag{5.6}$$

y los valores obtenidos son

$$C = 10^4 = 10000 \ , \tag{5.7}$$

$$\varepsilon = \frac{\sigma}{2^{-6}} = 53,46 \ , \tag{5.8}$$

$$\gamma = \frac{4^{-1}}{d} = 0,0035 \ , \tag{5.9}$$

No explicaremos esta validación cruzada con la misma tabla que las anteriores ya que tendría demasiadas columnas.

Con este modelo hemos obtenido un error absoluto medio del 6,36 %, que se traduce en alrededor de 1115 vatios, y un estadístico R^2 del 72,7 %. Son resultados mejores que los que obtuvimos para el perceptrón multicapa, que era nuestro mejor modelo hasta ahora, aunque la mejora es menos

significativa que del modelo ridge al perceptrón.

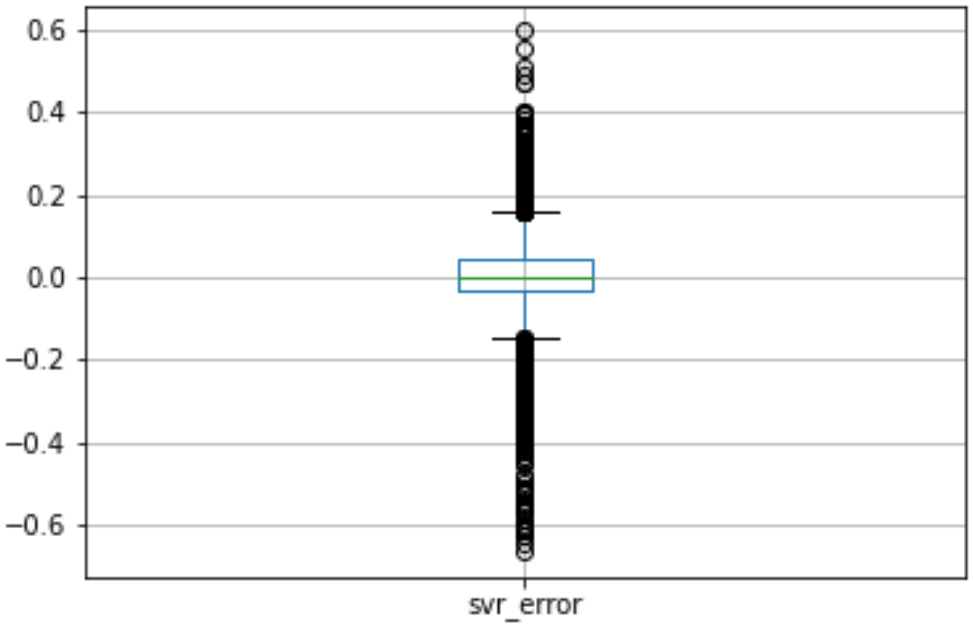

Figura 5.6: Errores del modelo SVR medidos como valores predecidos menos valores reales del perceptrón multicapa.

En la figura 5.6 vemos un diagrama de caja de los errores del modelo SVR prediciendo sobre las predicciones eólicas de 2018. El gráfico es, a simple vista, idéntico al del perceptrón multicapa, lo que nos hace pensar que no solo obtienen una media de error muy parecida, sino que también cometen los mismos errores. A continuación profundizaremos sobre esta idea, y compararemos las predicciones de unos modelos y otros en distintos escenarios, como parte del análisis del modelo de ensemble.

5.4. Stacking

Para el modelo stacking hemos utilizado la función *stackingRegressor* del módulo *ensemble* de la librería sklearn. Los modelos que utilizaremos como modelos de regresión de nivel 0 son los vistos hasta ahora: el modelo ridge, el perceptrón multicapa y la SVR. Estos modelos se introducen en el parámetro *estimators*. Para cada uno de estos utilizaremos los parámetros que calculamos haciendo la validación cruzada de cada uno, y de modelo de regresión de nivel 1, que corresponde al parámetro *final_estimator*, utilizamos una regresión lineal.

Para el modelo de regresión de nivel 1, hemos probado un modelo ridge, un perceptrón multicapa

y una regresión lineal, y los mejores resultados se obtuvieron para el modelo lineal, aunque fueron prácticamente iguales. Por esto, y dado que el modelo lineal es el más simple y rápido, es por lo que lo hemos elegido.

Para el modelo de nivel 1 de regresión lineal, el modelo stacking nos da un error de 6,51 %, es decir, que no consigue mejorar la predicción de sus mejores modelos: el perceptrón multicapa y el modelo SVR.

Para el modelo de nivel 1 ridge obtuvimos un error absoluto medio del 6,52 %, y utilizando un perceptrón multicapa de una capa oculta, un error absoluto medio del 6,53 %.

Vemos que el modelo no ha conseguido mejorar las predicciones de la SVR, aunque los resultados son prácticamente iguales. Esto quiere decir que para los datos donde la SVR cometía mayor error, los otros modelos tenían un error aún mayor, con lo que no se ha podido mejorar la predicción. Vamos a ver un ejemplo de este caso con algunos gráficos.

Estudiando la distribución del error absoluto de cada modelo individual, vemos que el día de mayor error de los tres modelos es el mismo: el 4 de abril de 2018, cuyas producciones de energía vemos en la gráfica 5.7. Es fácil ver cuál es el problema aquí: es un día con una producción relativamente alta (alrededor de un 70 % de la producción máxima) que súbitamente se interrumpe y hay varias horas de producción nula, para luego volver a producir a una tasa bastante cercana a la inicial.

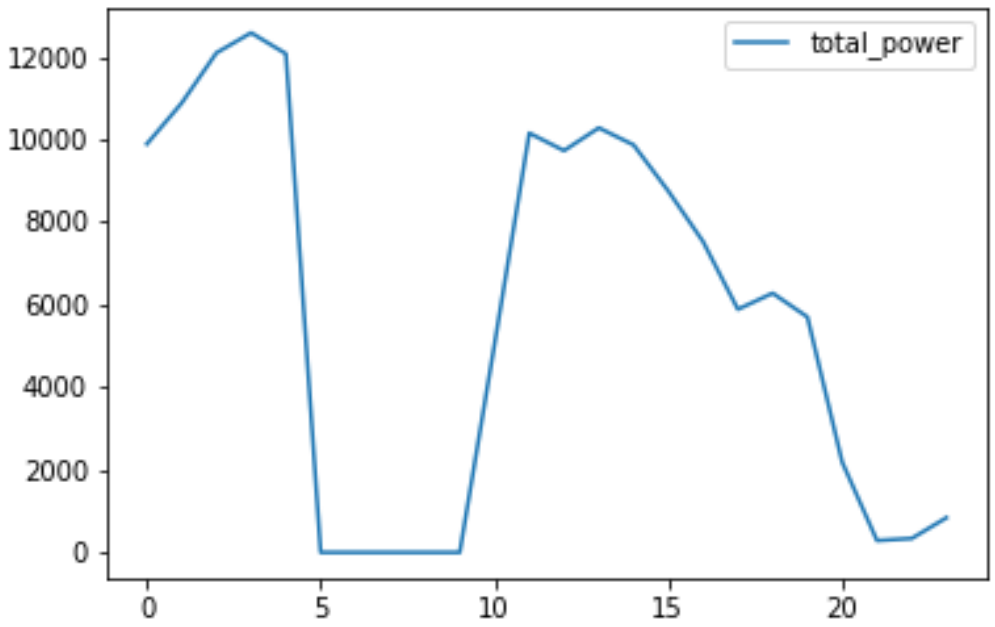

Figura 5.7: Producción de energía el día 4 de abril de 2018, el día en que mayor error cometen los 3 modelos: ridge, perceptrón multicapa y SVR; y por lo tanto también el modelo de enseble.

En la gráfica 5.8 vemos qué producción de energía predice cada modelo este día, y podemos observar claramente que ninguno de los cuatro modelos es capaz de predecir la producción nula. En la gráfica no podemos ver la gráfica del modelo SVR, porque está justo debajo de la gráfica del modelo stacking. Esto quiere decir que para esta sección de los datos, la predicción del modelos stacking se basa completamente en la predicción de la SVR, obviando la del modelo ridge y la del perceptrón.

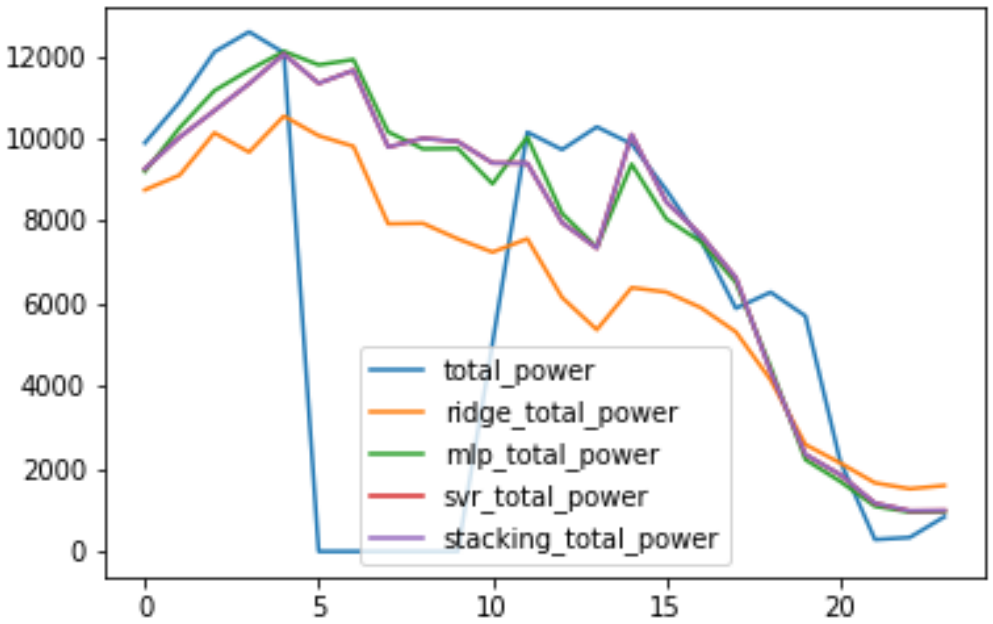

Figura 5.8: Predicción de energía de cada modelo el día 4 de abril de 2018 y producción real.

En la figura 5.9 vemos los errores de todos los modelos en valor absoluto el día 4 de abril de 2018. Al igual que en la gráfica 5.8, las gráficas del modelo SVR y del modelo stacking coinciden, y solo podemos ver la de stacking, que es la que está pintada encima.

La gráfica del error del modelo stacking, es claramente la que menor error tiene, a excepción del periodo en que la producción es nula, donde el modelo ridge tiene menor error. Sin embargo, vemos en el resto del gráfico que esto se debe simplemente a que ese día estaba subestimando las predicciones. Esto nos hace pensar que el modelo stacking no considera apenas las predicciones del modelo ridge debido a que la gran mayoría del tiempo tienen mayor error que las de el modelo SVR y las del modelo mlp, y como además no es capaz de anticipar los periodos de producción nula, no aprovecha la ventaja de que el modelo ridge subestime.

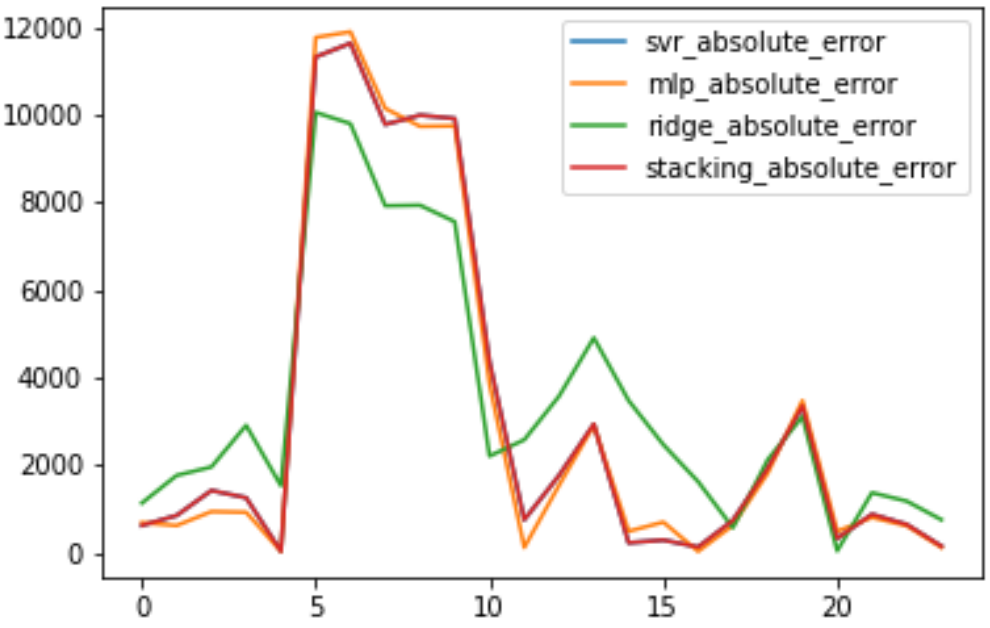

Figura 5.9: Errores de predicción de todos los modelos en valor absoluto para el día 4 de abril de 2018.

6

Conclusiones y trabajo futuro

Nos encontramos ante unos datos en los que es muy difícil mejorar un error del 6,5 % entrenando sobre 2016 y 2017 y prediciendo sobre 2018, y esto es debido a periodos de producción nula que nuestros modelos no han sido capaces de predecir. La conclusión obvia es que los periodos de predicción nula no dependen en absoluto de las predicciones meteorológicas, sino que deben de tener una causa completamente ajena que por lo tanto para nosotros resulta completamente aleatoria.

Respecto al comportamiento de nuestros modelos en periodos en los que no hay producción nula, el perceptrón multicapa y la SVR superan al modelo ridge, con lo que el modelo de ensemble no da ningún peso a las predicciones del modelo ridge.

Vemos además una clara preferencia del modelo de ensemble por las predicciones de la SVR frente a las del perceptrón multicapa, y esto parece deberse a que en general, y aunque no por una gran diferencia, las predicciones de la SVR superan a las del perceptrón, como vemos en todos los gráficos del capítulo anterior. De esta manera, el modelo de ensemble no supone una mejora respecto a la SVR.

Sería interesante encontrar modelos donde las secciones en que uno supere al otro sean algo mayores, para que así tenga más sentido una combinación entre las predicciones de ambos, en lugar de tener dos modelos donde haya una clara superioridad de uno respecto a otro, como nos hemos encontrado a lo largo de nuestros experimentos.

También sería de gran utilidad aumentar el espacio de datos para incluir alguna variable que indique cuándo la producción de energía será nula. Si se consiguiese solucionar este hándicap es muy probable que la mejora fuese altamente significativa, ya que no son pocas las veces en que la cantidad de energía obtenida ha sido nula.

Bibliografía

[1] T. H. y. R. T. Daniela Witten, Gareth James, *An introduction to statistical learning with applications in R*. Springer Texts in Statistics, eighth ed., 2017.

[2] R. O. D. y. D. G. S. Peter E. Hart, *Pattern classification*. John Wiley Sons Inc, second ed., 2000.

[3] A. J. S. y Bernhard Schölkopf, "A tutorial on support vector regression *," *Statistics and computing*, p. 8, Received 2002 and accepted 2003. Descargar.

[4] B. E. y Trevor Hastie, *Computer Age Statistical Inference*. Cambridge University Press, first ed., 2016.

[5] D. H. Wolpert, "Stacked generalization," *ResearchGate*, p. 8, 1992. Leer o descargar.

[6] C. M. A. y. J. R. D. Carlos Ruiz, "Multitask support vector regression for solar and wind energy prediction," *energies*, p. 8, 2020. Leer.


UAM
UNIVERSIDAD AUTONOMA
DE MADRID