Departamento de Matemáticas, Facultad de Ciencias
Universidad Autónoma de Madrid

Combinaciones de expertos aplicados al problema de regresión

Trabajo de fin de grado

Grado en Matemáticas

Autor: José Benjumeda Rubio

Tutores: Luis Alberto Rodríguez Ramírez y José Luis Torrecilla Noguerales

Curso 2020-2021

Resumen

Los *métodos de ensemble* de regresión combinan las predicciones de un conjunto de modelos de regresión buscando reducir el error que estos obtendrían por separado. En este trabajo los utilizamos en el campo de la predicción de energía eólica, donde hasta ahora rara vez se habían empleado.

Construiremos un método de ensemble siguiendo la metodología stacking, y lo utilizaremos para combinar las predicciones de un modelo ridge, un perceptrón multicapa y una máquina de vectores soporte de regresión. Comenzaremos explicando cada uno de estos modelos, para luego hacer una introducción a los principales metodologías para construir modelos de ensemble, viendo las ventajas e inconvenientes de cada uno, y, tras esto, explicaremos el stacking. En la última parte del trabajo utilizaremos los modelos estudiados para predecir la energía eólica obtenida en el Parque Eólico Experimental de Sotavento a partir de las predicciones meteorológicas del Centro Europeo de Previsiones Meteorológicas a Plazo Medio.

Abstract

Ensemble methods are used to combine the predictions of a set of regression models aiming at reducing the error that the latter would have when making predictions individually. In this work we are going to use them in the field of wind power prediction, in which they have been very seldom used.

We will build an ensemble method following the stacking methodology, and we will use it to combine the predictions of a ridge model, a multilayer perceptron and a support vector regressor. We will start by explaing each of these models, then we will introduce the main ensemble methods, underlining the advantages and disadvantages of each one, and, after this, we will explain stacking. In the last part of this work we conduct some experiments with the models that have been studied so far, using them to predict the amount of wind power obtained in the experimental wind farm of Sotavento by learning its relation with the weather forecast data of the European Centre for Medium-Range Weather Forecasts.

Índice general

CAPÍTULO 1
Introducción

Este trabajo está motivado por un problema real, que es la predicción de energía eólica en el Parque Eólico Experimental de Sotavento, en Galicia. Para permitir que la utilización de energía eólica siga creciendo en España, es necesario conocer, con el menor error posible, la cantidad de energía de la que se dispone en todo momento, y en este trabajo trataremos de predecirla a partir de las predicciones meteorológicas del Centro Europeo de Previsiones Meteorológicas a Plazo Medio [1]. Dado que este problema tiene como marco el problema de regresión, comenzaremos definiéndolo.

Hasta ahora, las predicciones que se han hecho en el campo de la energía eólica han empleado algunos modelos como las redes neuronales profundas multitarea [1] o las máquinas de vectores soporte de regresión multitarea [2], pero rara vez se ha estudiado la predicción eólica utilizando *métodos de ensemble*.

En este trabajo se construirá un método de ensemble según la metodología *stacking*, utilizando un modelo de regresión regularizada, un perceptrón multicapa y una máquina de vectores soporte de regresión, y se compararán los resultados obtenidos con los resultados de los modelos entrenados individualmente.

En el Capítulo 1, en el que nos encontramos, se explicará el contexto estadístico sobre el que se define el problema de regresión, que es el problema que estamos estudiando.

Los modelos utilizados se han dividido en dos capítulos: en el Capítulo 2, se hará una introducción a los modelos que luego compondrán el método de ensemble: la regresión regularizada, el perceptrón multicapa y la máquina de vectores soporte de regresión; y en el Capítulo 3, tras una breve introducción general a los modelos de ensemble más conocidos, se explica el stacking.

En el Capítulo 4, con intención de ilustrar las explicaciones teóricas, entrenaremos estos modelos para aprender la relación entre la cantidad de energía eólica obtenida cada hora de cada día en el parque eólico experimental de Sotavento, en Galicia, y las predicciones de los valores de módulo y dirección del vector de viento medido tanto a 10 como a 100 metros de altura, la temperatura medida a 2 metros de altura y la presión en la superficie. Este capítulo termina con algunas conclusiones sobre los resultados obtenidos.

[1] https://meteolo.com/europa/ecmwf/ecmwf-europa

Por último se incluyen algunas conclusiones generales en el Capítulo **??** tras haber realizado el resto del trabajo, se resumen las conclusiones sobre los experimentos y se dan algunas indicaciones sobre posible trabajo futuro.

1.1. El problema de regresión

Como se ha mencionado al principio del capítulo, utilizaremos los datos de predicciones meteorológicas para predecir la energía obtenida en el Parque Eólico Experimental de Sotavento, en adelante Parque de Sotavento. Esto es un problema de regresión, es decir, tratamos de asignar a una nueva observación de una variable aleatoria un valor numérico, a partir de la información proporcionada por otras variables aleatorias. A continuación, daremos algunos resultados y definiciones en el marco de este problema, para los que la referencia principal será el Capítulo 3 de [3].

Llamaremos p a la dimensión del problema, y representaremos como $X^1, X^2, \ldots, X^p$ a las variables aleatorias regresoras, es decir, a las variables cuyos utilizamos para predecir el valor de la variable respuesta. En ocasiones escribiremos $\boldsymbol{X}$ refiriéndonos al vector compuesto por estas p variables, es decir, $\boldsymbol{X} = (X^1, X^2, \ldots, X^p)^T$, donde el superíndice T indica que es un vector transpuesto. A la variable respuesta la representaremos mediante Y. Todas las variables aleatorias serán reales. Representaremos la observación i como $(\boldsymbol{X}_i, Y_i)^T$, donde $\boldsymbol{X}_i = (X_i^1, X_i^2, \ldots, X_i^p)^T$.

Definición 1.1. Dadas las variables aleatorias reales $X^1, X^2, \ldots, X^p, Y$, definimos una muestra de tamaño n como

$$D_n = \{(\boldsymbol{X}_i, Y_i)^T\}_{i=1}^n. \tag{1.1}$$

Definición 1.2. Dados los siguientes términos:

- $\boldsymbol{X}$ vector de p variables aleatorias, que llamamos variables regresoras.
- $\boldsymbol{\beta}$ vector de parámetros escalares.
- Y variable aleatoria, que llamamos variable respuesta.
- ε término de error, que modeliza el ruido y que suponemos se comporta como una normal de media 0 y desviación típica 1.

Definimos modelo de regresión como la función $f(\boldsymbol{\beta}, \boldsymbol{X})$ a partir de la que podemos expresar la variable respuesta Y como

$$Y = f(\boldsymbol{\beta}, \boldsymbol{X}) + \varepsilon. \tag{1.2}$$

El problema de regresión consiste en encontrar una aproximación de la función f, ya que, en el caso habitual, no conocemos su valor exacto. Según sea la forma de esta función, podemos hacer una primera clasificación entre los modelos de regresión, distinguiendo entre modelo de regresión lineal y modelo de regresión no lineal. Empezaremos estudiando la regresión lineal, para luego explicar la regresión regularizada como una modificación de este modelo, y después pasaremos al perceptrón multicapa y a la máquina de vectores soporte de regresión.

Regresión lineal

Un modelo de regresión lineal es aquel en el que la función $f(\boldsymbol{\beta}, \boldsymbol{X})$ se puede expresar como

$$f(\boldsymbol{\beta}, \boldsymbol{X}) = \boldsymbol{\beta}^T \boldsymbol{X} \tag{1.3}$$

donde $\boldsymbol{\beta} = (\beta_0, \beta_1, \ldots, \beta_p)^T$, siendo β_0 la ordenada en el origen y β_i, para $i = 1, \ldots, p$, el coeficiente de la variable i. Además, hacemos un abuso de notación incluyendo en $\boldsymbol{X}$ una primera coordenada igual a 1, que al hacer el producto escalar, multiplicaremos por β_0, es decir, $\boldsymbol{X} = (1, X^1, X^2, \ldots, X^p)^T$. La interpretación de β_i es el efecto medio sobre la variable respuesta, Y, de un incremento de una unidad sobre la variable regresora i, X^i.

En el contexto habitual de un problema de regresión lineal, no conocemos el valor de $\boldsymbol{\beta}$, sino que la única información de la que disponemos es una muestra D_n como vemos en la Definición 1.1. La manera de proceder es, a partir de D_n, estimar sus valores mediante $\hat{\boldsymbol{\beta}} = (\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_p)$, minimizando el error cuadrático medio J para los valores de la muestra:

$$J(\hat{\boldsymbol{\beta}}) = \frac{1}{n} \sum_{i=1}^{n} (Y_i - \hat{\boldsymbol{\beta}}^T \boldsymbol{X}_i)^2. \tag{1.4}$$

En el caso particular en que $p = 1$, y por tanto

$$J(\hat{\boldsymbol{\beta}}) = \frac{1}{n} \sum_{i=1}^{n} (Y_i - (\hat{\beta}_0 + \hat{\beta}_1 X_i^1))^2, \tag{1.5}$$

podemos ver algunos resultados de interés con pocos cálculos, y para $p > 1$ los cálculos son análogos pero algo más largos. En primer lugar, por la forma de la función $J(\boldsymbol{\beta})$, sabemos que el mínimo existe y es único. Desarrollamos, prescindiendo del superíndice de la variable regresora X^1, ya que solo hay una:

$$\begin{aligned} J(\hat{\beta}_0, \hat{\beta}_1) &= \frac{1}{n} \sum_{i=1}^{n} (Y_i - (\hat{\beta}_0 + \hat{\beta}_1 X_i))^2 = \\ &= \frac{1}{n} \sum_{i=1}^{n} (Y_i^2 + \hat{\beta}_0^2 + \hat{\beta}_1^2 X_i^2 + 2\hat{\beta}_0 \hat{\beta}_1 X_i - 2\hat{\beta}_0 Y_i - 2\hat{\beta}_1 Y_i X_i) = \\ &= E[Y^2] + \hat{\beta}_0^2 + \hat{\beta}_1^2 E[X^2] + 2\hat{\beta}_0 \hat{\beta}_1 E[X] - 2\hat{\beta}_0 E[Y] - 2\hat{\beta}_1 E[XY] = \\ &= (\hat{\beta}_0, \hat{\beta}_1) \begin{pmatrix} 1 & E[X] \\ E[X] & E[X^2] \end{pmatrix} \begin{pmatrix} \hat{\beta}_0 \\ \hat{\beta}_1 \end{pmatrix} - 2E[Y]\hat{\beta}_0 - 2E[XY]\hat{\beta}_1 + E[Y^2]. \end{aligned} \tag{1.6}$$

El primer término,

$$(\hat{\beta}_0, \hat{\beta}_1) \begin{pmatrix} 1 & E[X] \\ E[X] & E[X^2] \end{pmatrix} \begin{pmatrix} \hat{\beta}_0 \\ \hat{\beta}_1 \end{pmatrix}, \tag{1.7}$$

es una forma cuadrática definida positiva, pues su determinante es la varianza de X, y por lo tanto siempre tiene un único mínimo. Los otros dos términos lineales y el término constante no cambian esto, ya que al derivar, su efecto es el de sumar constantes a una ecuación lineal, y esto no cambia la cantidad de soluciones.

El segundo resultado que se puede calcular con facilidad es la expresión de $\hat{\beta}_0$ y de $\hat{\beta}_1$ en función de X e Y. Derivando y despejando obtenemos:

$$\hat{\beta}_0 = E[Y] - E[X]\frac{COV(X,Y)}{\sigma^2(X)} \tag{1.8}$$

$$\hat{\beta}_1 = \frac{E[XY] - E[X]\ E[Y]}{E[X^2] - E[X]^2} = \frac{COV(X,Y)}{\sigma^2(X)}. \tag{1.9}$$

En el caso en que $p > 1$, como hemos dicho, los argumentos para probar ambos resultados son análogos. Tomando

$$\boldsymbol{M_Y} = \begin{pmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{pmatrix} \qquad \boldsymbol{M_X} = \begin{pmatrix} 1 & X_1^1 & X_1^2 & \dots & X_1^p \\ 1 & X_2^1 & X_2^2 & \dots & X_2^p \\ \vdots & & & & \\ 1 & X_n^1 & X_n^2 & \dots & X_n^p \end{pmatrix}, \tag{1.10}$$

y suponiendo que ${\boldsymbol{M_X}}^T \boldsymbol{M_X}$ es invertible, podemos expresar $\hat{\boldsymbol{\beta}}$ como

$$\hat{\boldsymbol{\beta}} = ({\boldsymbol{M_X}}^T \boldsymbol{M_X})^{-1} {\boldsymbol{M_X}}^T \boldsymbol{M_Y}. \tag{1.11}$$

1.2. Ajuste de parámetros: validación cruzada

Como ya hemos visto, los modelos que vamos a estudiar tienen parámetros, que influyen en gran medida en las predicciones que obtendremos. Para ajustar su valor utilizamos validación cruzada, que es un método para evaluar los resultados que obtiene un modelo con distintos parámetros.

En primer lugar tenemos que definir un conjunto finito de posibles valores para cada parámetro que queremos ajustar, y un modelo con cada posible combinación. Llamando k a la cantidad de posibles combinaciones, definimos una familia de k modelos $\{G_i\}_{i=1}^k$. El siguiente paso es ver cual de los k modelos obtiene menor error de predicción. Como función de error utilizamos el error absoluto medio.

Para calcular el error de cada modelo, dividimos la muestra D_n en un subconjunto de entrenamiento, D_{train} y un subconjunto de test, D_{test}. Sobre D_{train}, realizaremos otra partición que lo divida en K subconjuntos, que denotaremos como $C_1, C_2, \dots, C_K$. Entonces para realizar una predicción sobre una observación de D_{train}, $\boldsymbol{X}_i \in C_j$, entrenaremos el modelo sobre el resto de D_{train}, es decir, sobre $C_1, \dots, C_{j-1}, C_{j+1}, \dots, C_K$.

De esta manera, los modelos siempre predicen sobre datos desconocidos, que no pertenecían al conjunto sobre el que se ha entrenado, y así evitamos el sobreajuste.

Tras definir estas particiones, solo queda calcular el error de cada modelo, para lo que utilizaremos la siguiente notación: dado un conjunto de datos D_n, y una observación de las variables regresoras $\boldsymbol{X}$, definimos $G(D_n; \boldsymbol{X})$ como la predicción del valor de la variable respuesta de esa observación mediante el modelo G cuando se entrena con el conjunto D_n.

La expresión del error absoluto cada modelo G de la familia $\{G_i\}_{i=1}^k$ es

$$CV(G, D_{train}) = \frac{1}{n} \sum_{i=1}^{k} \sum_{j=1}^{n_i} \left| G\Big(D_{train} \setminus C_i \; ; \; \boldsymbol{X}_j \Big) - Y_j \right|, \tag{1.12}$$

donde el segundo sumatorio itera sobre los elementos de cada subconjunto C_i.

Un caso concreto de validación cruzada es el caso en que tomamos $K = \#(D_{train})$, es decir, particionamos en tantos subconjuntos como datos, y por lo tanto, para predecir el valor de la variable respuesta de la observación $\boldsymbol{X}_i$, entrenamos el modelo con $D_{train} \setminus \{(\boldsymbol{X}_i, Y_i)^T\}$. Esta manera de proceder tiene la ventaja de que los resultados no dependen de cómo hayamos escogido cada uno de los K subconjuntos de D_{train}, sin embargo, hay que entrenar cada modelo tantas veces como observaciones haya en D_{train}, y en consecuencia el coste computacional es mayor que si elegimos un K más pequeño.

CAPÍTULO 2
Modelos de regresión

En este capítulo estudiaremos algunos modelos de regresión más complejos que la regresión lineal, que nos permiten obtener errores pequeños en problemas en los que un modelo de regresión lineal por sí solo no se ajuste bien a los datos. Explicaremos la regresión lineal regularizada, el perceptrón multicapa y la máquina de vectores soporte de regresión, que son los modelos cuyas predicciones combinaremos más adelante con el stacking.

2.1. Regresión lineal regularizada

En problemas en los que utilizando un modelo de regresión multilineal obtengamos un error pequeño, es usual que este modelo tenga poco sesgo pero varianza relativamente alta, debido a que cometa sobreajuste. Esto implicará que obtendremos buenas predicciones en datos cuyo comportamiento se asemeje al de la muestra sobre la que se ha entrenado el modelo, pero que cuando los datos varíen un poco respecto a esta, obtengamos un error demasiado grande. Cuanto mayor sea la dimensión del problema p respecto al tamaño de la muestra n, mayor será el sobreajuste.

La solución a este problema es penalizar la complejidad del modelo, para que éste no sea capaz de representar de manera tan precisa las características de la muestra, y por tanto generalice más. Esto se llama regularización, y veremos distintas formas de hacerlo según el modelo que estemos tratando.

En la regresión lineal, la regularización consiste en penalizar valores altos de los coeficientes de las variables regresoras, y el modelo que implementa esta técnica es la regresión lineal regularizada, o *ridge regression*. Aún buscamos la misma función $f(\boldsymbol{\beta}, \boldsymbol{X}) = \boldsymbol{\beta}^T \boldsymbol{X}$, pero aproximaremos $\boldsymbol{\beta}$ minimizando una modificación la función de error $J(\hat{\boldsymbol{\beta}})$, que ahora llamaremos $\widetilde{J}(\hat{\boldsymbol{\beta}})$, y que escribiremos como

$$\widetilde{J}(\hat{\boldsymbol{\beta}}) = \sum_{i=1}^{n} (Y_i - \hat{\boldsymbol{\beta}}^T \boldsymbol{X}_i)^2 + \lambda \sum_{j=1}^{p} \hat{\beta}_j^2 = J(\hat{\boldsymbol{\beta}}) + \lambda \sum_{j=1}^{p} \hat{\beta}_j^2. \tag{2.1}$$

El parámetro de suavizado, λ, equilibra la importancia relativa de minimizar el error cuadrático y de minimizar los coeficientes. Según el valor de lambda, minimizaremos una función u otra, con lo que las soluciones serán distintas. Cuando mayor sea su valor, mayor será la penalización de los coeficientes altos, y más generalizará el modelo, y viceversa. En el caso particular en el que λ es 0, tenemos un modelo de regresión multilineal.

Es importante notar que la penalización no se aplica sobre $\hat{\beta}_0$, que es simplemente la media de la variable respuesta cuando todas las variables regresoras tienen valor 0. El motivo es que consideramos que ésta es la mejor predicción que podemos hacer sobre el valor de la variable respuesta cuando todas las demás variables son cero, y cambiarla solo nos llevaría a un mayor error.

Otra observación que merece la pena destacar es que añadir la regularización al modelo de regresión multilineal apenas aumenta el coste computacional, y, dado que se mantiene la posibilidad de hacer una regresión multilineal sin regularización, al darle el valor cero al parámetro λ, es difícil encontrar un problema en que la regresión multilineal sea más conveniente que la regresión regularizada.

2.2. Perceptrón multicapa

En ésta sección veremos las redes neuronales multilineales, concretamente el perceptrón multicapa, que es el primer modelo de los vistos hasta ahora que permite resolver problemas no lineales. La referencia principal de la que se han obtenido la mayoría de definiciones y resultados de esta sección es [5].

Comenzaremos definiendo la estructura del perceptrón multicapa. Para mayor simplicidad, realizaremos las explicaciones sobre un perceptrón multicapa de una capa oculta pero las explicaciones para un perceptrón con más capas son análogas. A continuación veremos algunos resultados teóricos que sirven de apoyo o fundamento para el uso de este modelo y, por último, veremos cómo se hace la estimación de sus parámetros a partir de una muestra utilizando el método de retropropagación.

2.2.1. Estructura del perceptrón multicapa

Un perceptrón multicapa consiste en capas compuestas de neuronas. Cada neurona tiene asociada una función, a la que llamamos la función de activación, y su comportamiento consiste en recibir un valor y emitir otro, que será la imagen mediante su función de activación del valor que recibe. Las neuronas están conectadas unas con otras, como se puede ver en la Figura 2.1.

Las entradas de las neuronas de la primera capa son las coordenadas del vector de una observación. Dado que cada observación $\boldsymbol{X}$ es un vector de $\mathbb{R}^p$, la primera capa tendrá p neuronas, y la neurona i recibirá la coordenada i de $\boldsymbol{X}$. Las entradas de las neuronas de las demás capas son las salidas de las neuronas de la capa anterior ponderadas por pesos, como vemos en la Figura 2.1. Además, hay una neurona más,

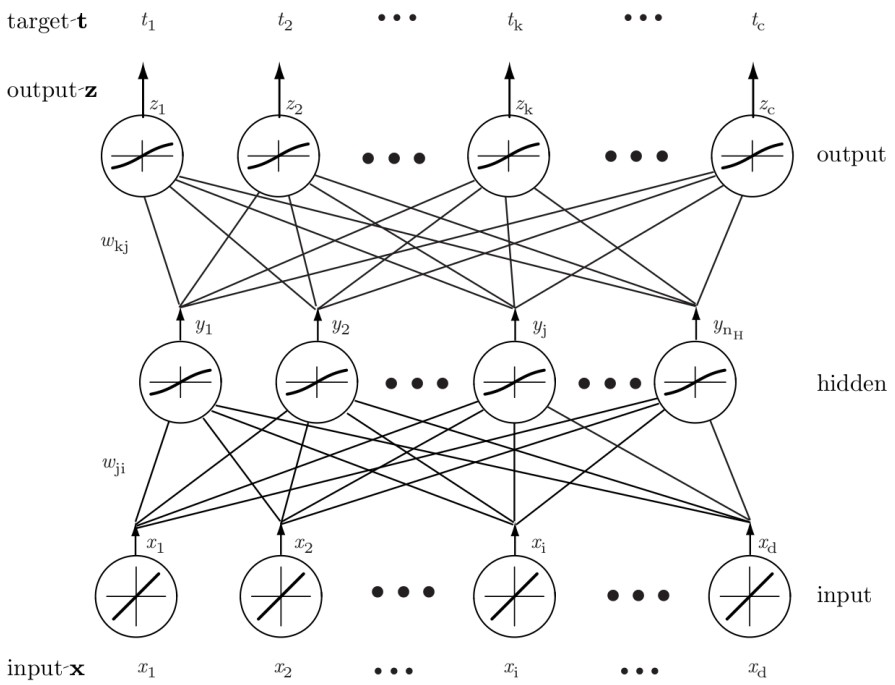


Figura 2.1: En esta figura vemos el esquema de un perceptrón multicapa con una capa oculta.

la neurona del sesgo, que va conectada a todas las demás neuronas excepto a las de la capa de entrada. Esta neurona emite un valor constante y no tiene entradas

Respecto a los pesos, utilizaremos w_{ji} para referirnos al peso que va de la neurona i de la capa de entrada a la neurona j de la capa oculta, y w_{jk} para referirnos al que va de la neurona j de la capa oculta a la neurona k de la capa de salida. El peso que conecta la neurona del sesgo con la neurona j de la capa oculta es w_{0j}, y para la neurona k de la capa de salida es w_{0k}.

La cantidad de neuronas de la capa de entrada, que llamaremos n_i, corresponde a la cantidad de variables regresoras del problema, como ya hemos mencionado. La capa de salida, como estamos en un problema de regresión, tendrá una sola neurona, aunque para ver algunos resultados, a esta cantidad la llamaremos n_o. Estas dos cantidades se obtienen directamente del problema, sin embargo, la cantidad de neuronas de la capa oculta, que será n_h deberá ajustarla el diseñador.

Como función de activación de las neuronas de la primera y la última capa utilizaremos la función identidad. Para las neuronas de la capa oculta emplearemos la función rectificador o *ReLU*. Habitualmente para las neuronas de la capa oculta se han utilizado la función sigmoidal y la tangente hiperbólica, que tienen buenas propiedades de derivación, pero últimamente se han obtenido buenos resultados con la

ReLU, como puede verse en [9]. Esta función es una función definida a trozos, que tiene la siguiente expresión:

$$f(x) = \text{máx}\{x, 0\} = \begin{cases} x & \text{si} \quad x \geq 0 \\ \\ 0 & \text{si} \quad x < 0 \end{cases} . \tag{2.2}$$

Llamaremos x_i a la salida de las neuronas de la primera capa, h_j a las de la capa oculta, y z_k a las de la capa de salida. Decimos además

$$\boldsymbol{x} = (x_1, \ldots, x_{n_i})^T,$$

$$\boldsymbol{h} = (h_1, \ldots, h_{n_h})^T,$$

$$\boldsymbol{z} = (z_1, \ldots, z_{n_o})^T,$$

$$h_j(\boldsymbol{x}) = f\left(\sum_{i=0}^{n_i} w_{ji} x_i\right) \tag{2.3}$$

$$z_k(\boldsymbol{h}) = f\left(\sum_{j=0}^{n_h} w_{kj} h_j\right), \tag{2.4}$$

y, sustituyendo, obtenemos la salida del perceptrón en función de la entrada:

$$z_k(\boldsymbol{x}) = f\left(\sum_{j=0}^{n_h} w_{kj} f\left(\sum_{i=0}^{n_i} w_{ji} x_i\right)\right). \tag{2.5}$$

¿Qué funciones puede representar una red neuronal?

El teorema de Kolmogorov muestra que cualquier función continua de n variables cuyo dominio esté restringido a un hipercubo $I^n : I = [0, 1], n \geq 2$ puede expresarse como sumas y composiciones de un número finito de funciones de una sola variable. Enunciamos el teorema a continuación, que ha sido obtenido de [8]:

Teorema 2.1. ***(Kolmogorov)*** *Para todo entero $n > 1$ existen $n(2n + 1)$ funciones monótonas crecientes Φ_{pq} con $1 \leq p \leq n$ y $1 \leq q \leq 2n+1$ tales que para cada función continua real $f : I^n \mapsto \mathbb{R}$, existen funciones continuas Ψ_q, con $1 \leq q \leq 2n + 1$, tales que*

$$f(x_1, x_2, \ldots, x_n) = \sum_{q=1}^{2n+1} \Phi_q\left(\sum_{p=1}^{n} \Psi_{pq}(x_p)\right), \tag{2.6}$$

donde las funciones Ψ_{pq} son universales (Definición 3 de [8]) y no dependen de f, y las funciones de una sola variable Φ_q son continuas y sí que dependen de f.

Dado que siempre podemos reescalar la parte que nos interesa de una función para que esté contenida en el hipercubo I^n, esto no supone ninguna restricción. En un perceptrón multicapa, representar una función f mediante el teorema de Kolmogorov equivaldría a poner una capa oculta con $2n + 1$ neuronas, donde cada una recibiría como entrada la suma de d funciones de una variable, que a su vez reciben cada una como entrada la correspondiente coordenada del vector de entrada de la red. La salida de las neuronas de la capa oculta sería el resultado de aplicar la correspondiente función Φ_q a su entrada. En la última capa habría una única neurona que sumaría las salidas de todas las neuronas ocultas.

Otra prueba de la utilidad de las redes neuronales es el teorema de Fourier [10], 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.

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

Estimación de parámetros: método de retropropagación

Utilizaremos como función de error el error cuadrático, que, fijando el valor de entrada, se puede ver como una función de todos los pesos de la red. La llamamos $J(\mathbf{w})$:

$$J(\mathbf{w}) = \frac{1}{2}\sum_{k=1}^{n_o}(t_k - z_k)^2 = \frac{1}{2}(\mathbf{t} - \mathbf{z})^2, \tag{2.7}$$

siendo $\mathbf{t}$ el valor poblacional de la variable respuesta de una observación, $\mathbf{z}$ la predicción del valor de la variable respuesta de esa misma observación y $\mathbf{w}$ el vector de todos los pesos de la red (weight). Notar que en este sumatorio ya no incluimos la neurona del sesgo, por lo que empezamos a sumar en $k = 1$. Queremos minimizar esta función; encontrar los pesos para los que la diferencia entre el valor predicho y el valor poblacional es mínima.

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 la imagen.

Las modificaciones serán proporcionales al error cuadrático, pues si este es cercano a cero, ya estamos muy cerca del mínimo (que es mayor o igual a 0). Representamos el incremento de **w** así:

$$\Delta \mathbf{w} = -\eta \frac{\partial J}{\partial \mathbf{w}}, \tag{2.8}$$

donde η indica el tamaño del incremento.

Primero vamos a calcular el incremento para los pesos de entre la capa oculta y la de salida, y luego para los pesos de entre la capa de entrada y la oculta. Llamamos net_k a la entrada de la neurona k de la capa de salida, y net_j a la entrada de la neurona j de la capa oculta:

$$net_k = \sum_{j=0}^{n_h} w_{kj} h_j \tag{2.9}$$

$$net_j = \sum_{i=0}^{n_i} w_{ji} x_j \tag{2.10}$$

Comenzamos a desarrollar la derivada de J respecto del peso w_{kj}:

$$\frac{\partial J}{\partial w_{kj}} = \frac{\partial J}{\partial net_k} \frac{\partial net_k}{\partial w_{kj}}, \tag{2.11}$$

y definimos la sensibilidad de la neurona k de la capa oculta como

$$\delta_k = -\frac{\partial J}{\partial net_k} = -\frac{\partial J}{\partial z_k} \frac{\partial z_k}{\partial net_k} = (t_k - z_k) f'(net_k). \tag{2.12}$$

Luego, derivando respecto de w_{kj} en la ecuación 2.9, obtenemos

$$\frac{\partial net_k}{\partial w_{kj}} = h_j. \tag{2.13}$$

Finalmente, sustituimos en la ecuación del incremento de w_{kj}:

$$\Delta w_{kj} = -\eta \frac{\partial J}{\partial w_{kj}} = -\eta \frac{\partial J}{\partial z_k} \frac{\partial z_k}{\partial net_k} \frac{\partial net_k}{\partial w_{kj}}$$

$$\Delta w_{kj} = \eta \delta_k h_j = \eta (t_k - z_k) f'(net_k) h_j. \tag{2.14}$$

Ahora hacemos lo mismo, con algún paso más, para los pesos de la capa de entrada a la capa oculta:

$$\frac{\partial J}{\partial w_{ji}} = \frac{\partial J}{\partial h_j}\frac{\partial h_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ji}} = \frac{\partial J}{\partial h_j} f'(net_j)\frac{\partial net_j}{\partial w_{ji}}.$$

Desarrollamos el primer término:

$$\frac{\partial J}{\partial h_j} = \left(\frac{\partial}{\partial h_j}\right)\left(\frac{1}{2}\sum_{k=1}^{n_o}(t_k - z_k)^2\right) = -\sum_{k=1}^{n_o}(t_k - z_k)\frac{\partial z_k}{\partial h_j} =$$
$$= -\sum_{k=1}^{n_o}(t_k - z_k)\frac{\partial z_k}{\partial net_k}\frac{\partial net_k}{\partial h_j} = -\sum_{k=1}^{n_o}(t_k - z_k) f'(net_k)\frac{\partial net_k}{\partial h_j}.$$

Ahora derivamos respecto de h_j en la ecuación 2.9, obteniendo

$$\frac{\partial J}{\partial h_j} = -\sum_{k=1}^{n_o}(t_k - z_k) f'(net_k) w_{kj}. \tag{2.15}$$

De manera análoga a como se hizo antes, definimos la sensibilidad de una neurona oculta como

$$\delta_j = f'(net_j)\sum_{k=1}^{n_o} w_{kj}\delta_k = f'(net_j)\sum_{k=1}^{n_o} w_{kj}(t_k - z_k) f'(net_k)\ . \tag{2.16}$$

Además, derivando respecto de w_{ji} en la ecuación de net_j (ecuación 2.10), obtenemos

$$\frac{\partial net_j}{\partial w_{ji}} = x_i\ , \tag{2.17}$$

y haciendo todas estas sustituciones en la ecuación incremento de w_{ji} nos queda

$$\Delta w_{ji} = -\eta\frac{\partial J}{\partial h_j}\frac{\partial h_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ji}}$$

$$\Delta w_{ji} = \eta\left(\sum_{k=1}^{n_o}(t_k - z_k) f'(net_k) w_{jk}\right) f'(net_j) x_i. \tag{2.18}$$

o, en su forma compacta utilizando las sensibilidades que hemos definido,

$$\Delta w_{ji} = \eta\left(\sum_{k=1}^{n_o}\delta_k w_{jk}\right) f'(net_j) x_i$$

$$\Delta w_{ji} = \eta\delta_j x_i$$

2.3. Máquinas de vectores soporte de regresión (SVR)

Las máquinas de vectores soporte de regresión (siglas en inglés SVR) son una solución al problema de regresión en la que buscamos una función f según se describe en la Definición 1.2 que, en las observaciones del conjunto de aprendizaje, tenga un error menor que un ε definido, y que al mismo tiempo sea lo más plana posible. En este contexto, que una función sea plana significa que sea constante. Construiremos esta función a partir de una muestra D_n como en la definición 1.1. Los resultados y definiciones de esta sección pueden encontrarse en [6].

Estudiaremos primero el caso en que f es lineal, y por tanto, tiene la forma

$$f(\boldsymbol{x}) = \langle \boldsymbol{w}, \boldsymbol{x} \rangle + b \quad : \boldsymbol{w} \in \mathbb{R}^p, b \in \mathbb{R}. \tag{2.19}$$

Para esta f, que sea plana se traduce en que $\|w\|$ sea cercano a 0. Encontrar la f entonces sería resolver el siguiente problema de optimización convexa, según vemos en la página 2 de [6]:

$$\min \quad \frac{1}{2}\|\boldsymbol{w}\|^2 \quad \text{sujeto a} \quad \begin{cases} y_i - \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle - b \leq \varepsilon \\ \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b - y_i \leq \varepsilon \end{cases} . \tag{2.20}$$

En ocasiones este problema no se puede resolver, o simplemente queremos permitir cierta cantidad de error: para esto introducimos las llamadas *slack variables*, ξ_i, ξ_i^*:

$$\min \quad \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i^*) \quad \text{sujeto a} \quad \begin{cases} y_i - \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle - b \leq \varepsilon + \xi_i \\ \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b - y_i \leq \varepsilon + \xi_i^* \end{cases} . \tag{2.21}$$

La constante C equilibra la importancia de que f sea plana y la cantidad hasta la que toleramos desviaciones mayores que ε. Destacamos el detalle de que sólo las desviaciones mayores que ε tienen coste, pues son las únicas observaciones para las que $\xi_i \neq 0$ o $\xi_i^* \neq 0$, como vemos en el dibujo de la Figura 2.2, donde además podemos ver el tubo épsilon, representado mediante la línea discontinua.

En la mayoría de los casos, resultará más sencillo resolver la ecuación 2.21 en su expresión dual, como vemos en la página 2 de [6]. Para obtenerla hallamos el máximo de la función en lugar del mínimo, invertimos el signo de las inecuaciones y añadimos el conjunto dual de variables, que son los multiplicadores de Langrange $\eta_i^{(*)}$ y $\alpha_i^{(*)}$:

$$\begin{aligned} L :=& \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i^*) - \sum_{i=1}^{n}(\eta_i\xi_i + \eta_i^*\xi_i^*) \\ &- \sum_{i=1}^{n}\alpha_i(\varepsilon + \xi_i - y_i + \langle \boldsymbol{w}, \boldsymbol{x_i} \rangle + b) \\ &- \sum_{i=1}^{n}\alpha_i^*(\varepsilon + \xi_i^* + y_i - \langle \boldsymbol{w}, \boldsymbol{x_i} \rangle - b) \end{aligned} . \tag{2.22}$$

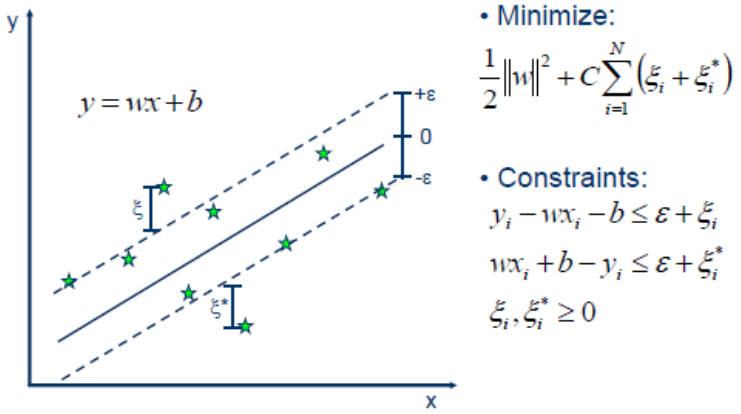


Figura 2.2: En esta figura vemos un dibujo del tubo épsilon [1].

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

$$\partial_b L = \sum_{i=1}^{n}(\alpha_i^* - \alpha_i) = 0, \tag{2.23}$$

$$\partial_{\boldsymbol{w}} L = \boldsymbol{w} - \sum_{i=1}^{n}(\alpha_i - \alpha_i^*)\boldsymbol{x_i} = 0, \tag{2.24}$$

$$\partial_{\xi_i^{(*)}} L = C - \alpha_i^{(*)} - \eta_i^{(*)} = 0. \tag{2.25}$$

Sustituyendo las ecuaciones 2.23, 2.24 y 2.25 en 2.22, obtenemos el problema de optimización dual

$$\text{máx} \begin{cases} -\frac{1}{2}\sum_{i,j=1}^{n}(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)\langle \boldsymbol{x_i}, \boldsymbol{x_j}\rangle \\ -\varepsilon\sum_{i=1}^{n}(\alpha_i + \alpha_i^*) + \sum_{i=1}^{n} y_i(\alpha_i - \alpha_i^*) \end{cases}, \tag{2.26}$$
$$: \quad \textstyle\sum_{i=1}^{n}(\alpha_i - \alpha_i^*) = 0 \quad \text{y} \quad \alpha_i, \alpha_i^* \in [0, C]$$

donde hemos eliminado las variables $\eta_i^{(*)}$ despejando en la ecuación 2.25.

A partir de la ecuación 2.24, podemos despejar $\boldsymbol{w}$ y que quede expresado en función de los $\boldsymbol{x_i}$ por un coeficiente:

$$\boldsymbol{w} = \sum_{i=1}^{n}(\alpha_i - \alpha_i^*)\boldsymbol{x_i}, \tag{2.27}$$

y por tanto

$$f(\boldsymbol{x}) = \sum_{i=1}^{n}(\alpha_i - \alpha_i^*)\langle \boldsymbol{x}_i, \boldsymbol{x}\rangle + b. \tag{2.28}$$

Vemos que $\boldsymbol{w}$ puede expresarse como una combinación lineal de las observaciones $\boldsymbol{x_i}$. Además, las únicas operaciones necesarias en este algoritmo son productos escalares entre observaciones, además de sumas y multiplicaciones por escalares.

Ahora falta calcular b, para lo que, según se indica en la página 3 de [6], utilizaremos las condiciones de Karush-Kuhn-Tucker, que dicen que la solución de nuestro problema también es solución de las siguientes cuatro ecuaciones:

$$\alpha_i(\varepsilon + \xi_i - y_i + \langle \boldsymbol{w}, \boldsymbol{x_i}\rangle + b) = 0, \tag{2.29}$$

$$\alpha_i^*(\varepsilon + \xi_i^* + y_i - \langle \boldsymbol{w}, \boldsymbol{x_i}\rangle - b) = 0, \tag{2.30}$$

$$(C - \alpha_i)\xi_i = 0 \tag{2.31}$$

y

$$(C - \alpha_i^*)\xi_i^* = 0. \tag{2.32}$$

Antes de hablar de b, nos fijamos en las implicaciones de estas condiciones, pues hay que hacer unas cuantas observaciones muy importantes. Lo primero es que si la predicción de una observación i está fuera del tubo épsilon, es decir, la distancia de la predicción al valor poblacional es mayor que ε, entonces o ξ_i o ξ_i^* son distintos de cero. Digamos que ξ_i es distinto de cero y el otro caso es análogo, intercambiando el $*$. Entonces, ξ_i^* es 0, puesto que es parte de lo que queremos minimizar y no hay ninguna otra restricción por la que tenga que ser mayor que 0.

Como ξ_i no es cero, para que se cumpla la tercera condición, tiene que cumplirse $\alpha_i = C$. Además, al ser ξ_i distinto de cero, la primera condición se cumple porque se anula el segundo término, pero la segunda tiene el segundo término no nulo, con lo cual $\alpha_i^* = 0$ para que sea cierta. De esta forma, para todas las observaciones cuya predicción salga del tubo épsilon, sabemos que $\alpha_i\alpha_i^* = 0$ y que o α o α^* es igual a C.

En el caso de que $|y_i - \langle \boldsymbol{w}, \boldsymbol{x}_i\rangle - b| < \varepsilon$, para que se cumplan las dos primeras condiciones, es necesario que $\alpha^{(*)} = 0$.

Lo que concluimos de estas observaciones es que para escribir $\boldsymbol{w}$ solo nos hace falta conocer las observaciones $\boldsymbol{x}_i$ cuya imagen por f salga del tubo épsilon, ya que todas las demás observaciones se multiplican por un coeficiente igual a cero. A las observaciones con coeficiente no nulo las denominamos vectores soporte.

Volviendo al cálculo de b, podemos escribir

$$\begin{array}{llll} \varepsilon - y_i + \langle \boldsymbol{w}, \boldsymbol{x_i}\rangle + b \geq 0 & \text{y} & \xi_i = 0 & \text{si } \alpha_i < C \\ \varepsilon - y_i + \langle \boldsymbol{w}, \boldsymbol{x_i}\rangle + b \leq 0 & & & \text{si } \alpha_i > 0 \end{array}. \tag{2.33}$$

Haciendo el estudio análogo sobre α_i^*, obtenemos

$$\begin{array}{l}\text{máx}\{-\varepsilon + y_i - \langle \boldsymbol{w}, \boldsymbol{x}_i\rangle \mid \alpha_i < C \text{ o } \alpha_i^* > 0\} \leq b \leq \\ \text{mín}\{-\varepsilon + y_i - \langle \boldsymbol{w}, \boldsymbol{x}_i\rangle \mid \alpha_i > 0 \text{ o } \alpha_i^* < C\}\end{array}, \tag{2.34}$$

y si algún $\alpha_i^{(*)} \in (0, C)$, tanto el máximo como el mínimo son b.

Funciones kernel

El siguiente paso adaptar el algoritmo para poder utilizarlo para resolver problemas no lineales. En principio podríamos conseguirlo simplemente aplicando a los datos una función no lineal $\phi : \mathbb{R}^p \to \mathbb{R}^q$ y luego aplicar el algoritmo como lo hemos descrito hasta ahora. El problema es que aplicar una función ϕ directamente a todos los datos tiene un coste computacional demasiado alto para problemas que a día de hoy son comunes.

La solución es que en lugar de calcular la imagen por ϕ de todos los datos de la muestra, aplicaremos la función no lineal de forma implícita. El detalle gracias al cual podemos hacer esto es que en ningún momento nos hace falta conocer $\phi(\boldsymbol{x_i})$ y $\phi(\boldsymbol{x_j})$ individualmente, sino que con conocer $\langle \phi(\boldsymbol{x_i}), \phi(\boldsymbol{x_i})\rangle$ nos basta. Esto lo aprovechamos utilizando ciertas funciones $K : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$, conocidas como funciones kernel, que cumplen $K(\boldsymbol{x_i}, \boldsymbol{x_j}) = \langle \phi(\boldsymbol{x_i}), \phi(\boldsymbol{x_i})\rangle$, siendo ϕ una función no lineal que depende de la función. Para aplicar K únicamente necesitamos saber el valor de $\boldsymbol{x}_i$ y $\boldsymbol{x}_j$, pero no hace falta conocer ϕ. Algunos ejemplos de funciones kernel son

$$K(\boldsymbol{x}, \boldsymbol{x'}) = (\langle \boldsymbol{x}, \boldsymbol{x'}\rangle + c)^p \quad : p \in \mathbb{N}, c \geq 0, \tag{2.35}$$

$$K(\boldsymbol{x}, \boldsymbol{x'}) = \widetilde{K}(\boldsymbol{x} - \boldsymbol{x'}) = \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{x'}\|^2}{2\sigma^2}\right), \tag{2.36}$$

donde la función $K(\boldsymbol{x}, \boldsymbol{x'})$ puede expresarse en términos de la diferencia entre $\boldsymbol{x}$ y $\boldsymbol{x'}$ mediante otra función $\widetilde{K}$, y σ es la norma de la desviación típica de la variable X.

Entonces, en problemas no lineales tenemos que resolver una versión de 2.26 modificada, en la que cambiamos $\boldsymbol{x}$ por $\phi(\boldsymbol{x})$, es decir, $\langle \boldsymbol{x_i}, \boldsymbol{x_j}\rangle$ por $K(\boldsymbol{x_i}, \boldsymbol{x_j})$:

$$\begin{array}{l}\text{máx} \left\{ \begin{array}{l} -\frac{1}{2}\sum_{i,j=1}^{n}(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)K(\boldsymbol{x_i}, \boldsymbol{x_j}) \\ \quad -\varepsilon\sum_{i=1}^{n}(\alpha_i + \alpha_i^*) + \sum_{i=1}^{n} y_i(\alpha_i - \alpha_i^*) \end{array} \right. \\ : \quad \sum_{i=1}^{n}(\alpha_i - \alpha_i^*) = 0 \quad \text{y} \quad \alpha_i, \alpha_i^* \in [0, C]\end{array}. \tag{2.37}$$

Además, tendremos

$$\boldsymbol{w} = \sum_{i=1}^{n} (\alpha_i - \alpha_i^*) \phi(\boldsymbol{x_i}), \tag{2.38}$$

aunque nunca calcularemos su valor explícitamente, sino que hallaremos $f(\boldsymbol{x_j})$ como

$$f(\boldsymbol{x}) = \sum_{i=1}^{n} (\alpha_i - \alpha_i^*) K(\boldsymbol{x_i}, \boldsymbol{x_j}) + b. \tag{2.39}$$

Ahora buscamos la función más plana posible en el espacio aumentado, no en el espacio inicial.

CAPÍTULO 3
Modelos de ensemble

En este capítulo haremos una explicación general sobre los métodos de ensemble, viendo los tres tipos principales: *bagging*, *boosting* y *stacking*. Daremos una explicación de los dos primeros, viendo algunas particularidades, ventajas y desventajas, aunque menos detalladamente que para el método stacking, pues no es tanto el objetivo de este trabajo estudiar los distintos métodos de ensemble como centrarnos en el stacking. Este último se explica tras los dos primeros, y luego realizaremos algunas pruebas con él en el Capítulo 4 para combinar las predicciones de los modelos de regresión vistos en el Capítulo 2.

3.1. ¿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 **CAMBIAR son combinados buscando 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, estudiados en el Capítulo 2, son modelos individuales.

Los modelos de ensemble son una solución al problema de elegir con qué modelos de regresión individuales haremos predicciones, de entre una familia de N modelos $\{G_j\}$, con $N \geq 1$.

Tradicionalmente, cuando utilizamos métodos individuales, elegimos mediante validación cruzada cuál de todos los modelos obtiene las mejores predicciones, y utilizamos 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, es común que tenga una varianza muy alta: 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 un error pequeño. 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 los modelos individuales de los que esté compuesto. Tendremos por lo tanto que utilizar el método de ensemble que coincida con los requerimientos del problema que se esté tratando, pues unos métodos están más orientados a reducir la varianza de sus modelos individuales 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 los métodos, como bagging o boosting, utilizan un único modelo variando sus parámetros 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 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, en la última parte de esta subsección, haremos una explicación general de bagging y boosting. Respecto al método stacking, dado que lo utilizaremos en los experimentos, lo estudiaremos más en detalle en la siguiente subsección.

Bagging

Bagging es un meta-algoritmo para construir métodos de ensemble que reduce la varianza y el sobreajuste de los modelos por los que está compuesto **mediante la combinación de sus predicciones**. La palabra bagging viene de *bootstrap aggregating*, pues este método consiste en promediar las predicciones de un conjunto de k modelos **entrenados/estimados a partir de**cuyos conjuntos de entrenamiento se han obtenido a partir de la muestra principal utilizando bootsrapping.

Bootstrapping es una metodología 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, la distribución subyacente de las nuevas muestras es una buena aproximación de la distribución de la muestra inicial.

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.

Tras generar las k muestras bootstrap, se entrenan los k modelos de forma independiente, y se promedian las predicciones que obtiene cada uno. Este promedio 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.

Bagging tiene la ventaja frente a boosting y stacking de que, al entrenarse los modelos individualmente, se pueden entrenar en paralelo, y esto suele suponer una mejora muy significativa del tiempo de ejecución. El ejemplo más conocido de bagging es random forests, donde se combinan árboles de decisión.

Boosting

Boosting es un meta-algoritmo para construir métodos de ensemble que, al contrario que bagging, está orientado a reducir el sesgo de los modelos por los que está compuesto. Esta vez los modelos se entrenan de forma secuencial, influyendo el entrenamiento de cada uno en el de los modelos posteriores. Esto se hace asignando pesos a los datos, para que cada modelo se centre en las observaciones que el modelo anterior ha predicho con mayor error, tratando así de reducir su sesgo. La predicción del boosting será combinación, esta vez más compleja que en el caso de bagging, de las predicciones de estos modelos: se hará una suma ponderada de las predicciones de cada, asignándoles distintos pesos dependiendo del cuanto error obtengan en sus predicciones.

Dado que los modelos se entrenan secuencialmente, el coste temporal puede llegar a ser muy elevado, por lo que se suelen utilizar modelos individuales poco precisos que se entrenen rápidamente.

La predicción del modelo de ensemble boosting será, como hemos dicho, una suma ponderada de las predicciones de los modelos individuales, y buscar una ponderación que dé buenos resultados no suele ser inmediato. *Adaptative boosting* y *gradient boosting* son dos maneras distintas de construir un modelo de ensemble boosting, que se diferencian en la manera de encontrar los coeficientes de esta suma ponderada, aunque, por restricciones de extensión, en este trabajo no las explicaremos.

3.2. Método de ensemble *stacking*

En esta sección vamos a definir una técnica utilizada para construir métodos de ensemble conocida como *stacking* o *stacked generalization*, utilizando [7] como referencia principal. En un modelo de este tipo formamos varias capas de modelos de forma que los modelos de cada capa se entrenan con las predicciones de los modelos de la capa anterior, que forman un nuevo conjunto de entrenamiento. La principal diferencia con boosting y bagging, aparte del proceso de combinación de las predicciones, es que en stacking los modelos pueden ser de distinto tipo, y esto ofrece un abanico más amplio de posibilidades.

Llamaremos conjunto de aprendizaje de nivel 0 a D_{train}, del que haremos una transformación, que enseguida definiremos, para conseguir el conjunto de aprendizaje de nivel 1, que denotaremos como $\widetilde{D}_{train}$. Además, de la familia de N modelos de regresión individuales, tomaremos un subconjunto de k modelos, que llamaremos los modelos de nivel 0 y denotaremos como $\{G_i\}_{i=1}^{k}$, y otro modelo, que puede estar incluido en los modelos de nivel 0 o no, que será el modelo de nivel 1, y denotaremos por $\widetilde{G}$. Los modelos de nivel 0 se llaman así porque se entrenan sobre el conjunto de entrenamiento de nivel 0, y ocurre igual con el nivel 1.

El enfoque de esta explicación se basa en considerar que el modelo con el que queremos realizar predicciones es el modelo de nivel 1, en lugar de todo el modelo de ensemble. La parte restante de este último, que son los modelos de nivel 0, se utiliza únicamente para hacer una transformación previa de los datos.

Lo primero que explicaremos es cómo se hace la transformación que genera $\widetilde{D}_{train}$ a partir de D_{train}. Tomando un elemento genérico de $\widetilde{D}_{train}$, digamos $(\widetilde{\boldsymbol{X}}_i, \widetilde{Y}_i)$, con $\widetilde{\boldsymbol{X}}_i = (\widetilde{X}_i^1, \widetilde{X}_i^2, \ldots, \widetilde{X}_i^k)$, vamos a ver cómo obtenerlo a partir de la observación i de D_{train}, es decir, $(\boldsymbol{X}_i, Y_i)$.

La coordenada j de $\widetilde{\boldsymbol{X}}_i$, es decir, $\widetilde{X}_i^j$, es la predicción del modelo j de la familia de modelos de nivel 0 del valor de la variable respuesta de la observación de $\boldsymbol{X}_i$, al entrenarse sobre todo el resto de D_{train}, es decir, $D_{train} \setminus \{(\boldsymbol{X}_i, Y_i)\}$. Por lo tanto, para obtener todas las coordenadas de $\widetilde{\boldsymbol{X}}_i$ aplicamos los k modelos de nivel 0 sobre una observación de D_{train}, y la observación obtenida vivirá en $\mathbb{R}^k$.

Volviendo a hacer uso de la notación que utilizamos para explicar la validación cruzada, donde $G(D_n; X)$ es la predicción del modelo G del valor de la variable respuesta de la observación X, cuando se ajustan los parámetros a partir de la muestra D_n, podemos escribir

$$\widetilde{\boldsymbol{X}}_i = \Big(G_1\big(D_{train} \setminus \{(\boldsymbol{X}_i, Y_i)\}; \boldsymbol{X}_i\big), \ldots, G_k\big(D_{train} \setminus \{(\boldsymbol{X}_i, Y_i)\}; \boldsymbol{X}_i\big)\Big), \tag{3.1}$$

El valor de la variable respuesta 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 Y_i para obtener $\widetilde{Y}_i$, pero en este trabajo, Y_i e $\widetilde{Y}_i$ son iguales, o lo que es lo mismo, se utiliza la identidad como transformación.

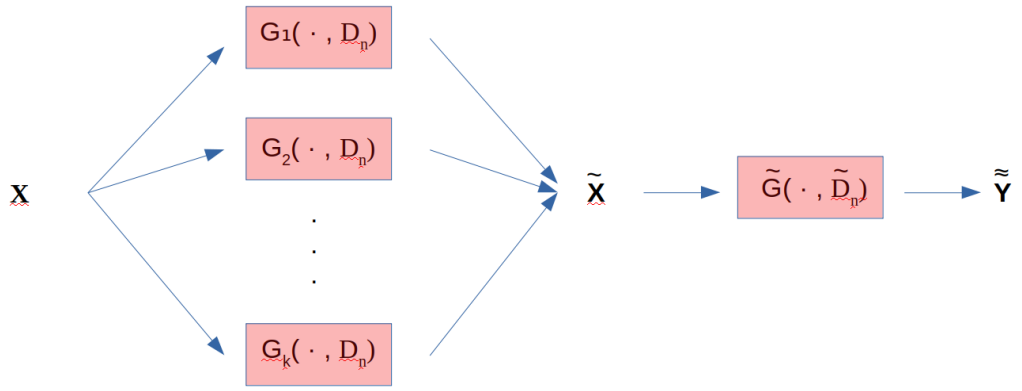


Figura 3.1: En este esquema vemos los pasos que hay que seguir para realizar una predicción sobre la observación $\boldsymbol{X}$.

Dado que con una observación de D_{train} obtenemos otra de $\widetilde{D}_{train}$, ambos conjuntos tendrán la misma cantidad de elementos.

Los modelos de nivel 0 se entrenarán de dos maneras distintas: la primera de ellas es la que acabamos de ver para obtener las observaciones de $\widetilde{D}_{train}$, donde para predecir sobre la observación $(\boldsymbol{X}_i, Y_i)$, se entrenan sobre $D_{train} \setminus \{(\boldsymbol{X}_i, Y_i))\}$. La segunda manera corresponde a cuando predecimos sobre una observación que no pertenece al conjunto de entrenamiento, por ejemplo sobre una observación del conjunto de test. Aunque la veremos un poco más adelante, adelantamos que la única diferencia es que los modelos se entrenarán sobre todo D_{train}, en lugar de sobre $D_{train} \setminus \{(\boldsymbol{X}_i, Y_i))\}$.

Hasta aquí se ha definido $\widetilde{D}_{train}$. Falta por definir, una vez entrenado el modelo de nivel 1, cómo utilizarlo para realizar predicciones de observaciones que esta vez pueden pertenecer a D_{train} o no.

Sea $\boldsymbol{X}$ una observación y sea $\widetilde{\boldsymbol{X}} = (\widetilde{X}_1, \widetilde{X}_2, \ldots, \widetilde{X}_k)$ su transformación, sobre la que podremos aplicar el modelo de nivel 1, la coordenada i de $\widetilde{\boldsymbol{X}}$ es la predicción del valor de la variable respuesta 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:

$$\widetilde{\boldsymbol{X}} = \Big(G_1\big(D_{train}; \boldsymbol{X}\big), \ldots, G_k\big(D_{train}; \boldsymbol{X}\big)\big) \Big), \tag{3.2}$$

Una vez obtenido $\widetilde{\boldsymbol{X}}$, la predicción que hagamos mediante el modelo de nivel 1, que llamaremos $\widetilde{\widetilde{Y}}$, será la predicción que buscamos, es decir, la predicción del modelo de ensemble para la observación $\boldsymbol{X}$. En cambio, en los modelos stacking en que $\widetilde{Y}$ se obtiene modificando Y, a la predicción que obtengamos de $\widetilde{\boldsymbol{X}}$ mediante el modelo de nivel 1 hay que aplicarle la inversa de dicha transformación.

En la Figura 3.1 vemos el proceso de predicción del valor de la variable respuesta 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 $i+1$, pues el procedimiento mostrado aquí es solo una de las posibilidades. La manera de proceder habitualmente es aplicar conocimiento específico del problema para seleccionar estos parámetros.

CAPÍTULO 4
Simulaciones

Este capítulo comienza planteando el problema de predicción eólica en el parque de Sotavento, describiendo los datos de los que disponemos y la manera en que los utilizaremos. Acto seguido se da una explicación general del procedimiento empleado para construir los modelos de regresión, y después se concretan detalles particulares de cada modelo. Finalmente se hace un análisis de los resultados obtenidos, comparando los errores de los distintos modelos y haciendo algunas hipótesis a la vista de estos.

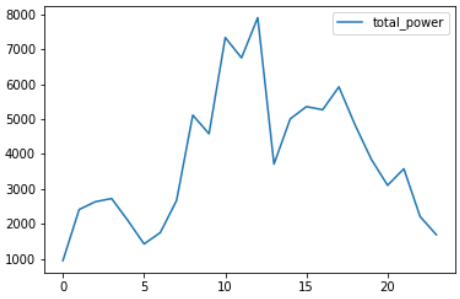
4.1. Parque Eólico Experimental de Sotavento

Vamos a realizar algunas pruebas de los modelos que se han estudiado hasta ahora para predecir la energía obtenida en el Parque Eólico Experimental de Sotavento, en Galicia. Para esto contamos con los datos del Centro Europeo de Previsiones Meteorológicas a Plazo Medio, que miden: velocidad del viento a 10 metros de altura y a 100, presión superficial y temperatura medida a 2 metros de altura. La velocidad del viento se mide en 3 variables: una con la componente este, otra con la componente norte, y otra con el módulo. La mayoría de estos datos se han obtenido a partir de [2].

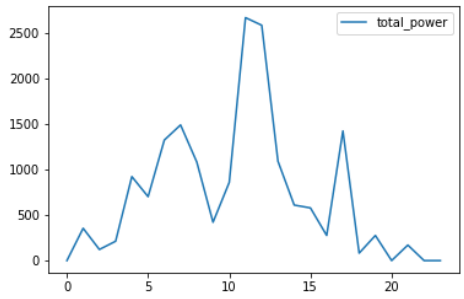
El Parque Eólico de Sotavento, al que a partir de ahora nos referiremos simplemente como parque de Sotavento, está situado en las coordenadas (43.354377º, -7.881213º). Conocemos las medidas atmosféricas que acabamos de explicar en una rejilla cuadrada de coordenadas de precisión 0,125º, por lo que la coordenada más cercana al parque de Sotavento es (43,375, -7,875). Inicialmente, la rejilla tenía 435 coordenadas, pero debido a nuestra capacidad de cómputo limitada, la reduciremos a las 9 coordenadas más cercanas al parque. Como en cada una tenemos las 8 medidas anteriores, en total tenemos 72 variables regresoras.

Tanto de estas variables como de la energía eólica obtenida en el parque de Sotavento, conocemos los valores de cada hora de cada día de los años 2016, 2017 y 2018. Dado que 2016 fue bisiesto, esto equivale a 26304 observaciones.

Una de las dificultades en el uso de energía eólica es su dependencia de factores tan cambiantes como son las condiciones atmosféricas. La variabilidad del módulo y la dirección del viento hacen que predecir la cantidad de energía eólica sea un problema en

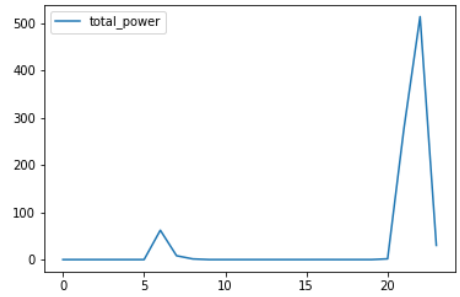


(a) 2 de enero 2018

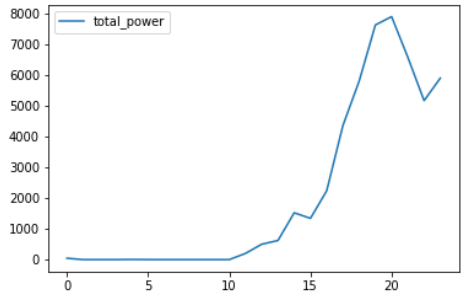


(b) 6 de julio 2018

Figura 4.1: Producción de energía muy poco constante.



(a) 11 de marzo 2018



(b) 26 de marzo 2018

Figura 4.2: Intervalos de tiempo con producción de energía nula.

el que conseguir errores cercanos a cero resulta muy complicado. Mostramos algunas gráficas de días escogidos al azar para mostrar lo poco constante que es la energía obtenida cada día, y para hacer notar que no son datos con un comportamiento fácil de predecir. Esto son las gráficas que vemos en la Figura 4.1.

Además, entre los datos de energía hay ciertas secciones con producción nula, y dado que es un cambio extremadamente drástico que no parece estar relacionado con las predicciones meteorológicas, y sabemos que nuestros datos no contienen valores erróneos ni nos faltan datos, asumimos que se deben a momentos en los que los molinos eólicos dejaron de funcionar, por ejemplo por razones de mantenimiento. Esto es una desventaja que tenemos que asumir al trabajar con estos datos. En la Figura 4.2 podemos ver dos ejemplos de días con producción nula.

Para profundizar algo más en la forma de los datos, pintamos un diagrama de cajas de la energía obtenida durante los tres años, como vemos en la Figura 4.3. Podemos ver que la mediana es muy baja, y que el bigote inferior es también muy pequeño. Esto indica una gran densidad de valores cercanos a cero, probablemente debidos a los periodos de producción nula que acabamos de mencionar y a periodos de poco viento.

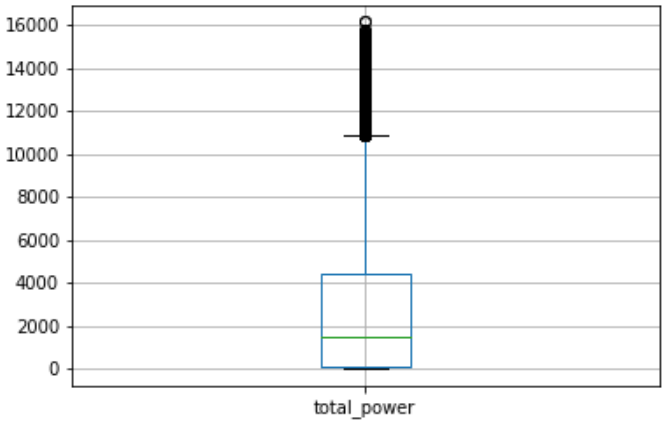


Figura 4.3: Gráfico de caja de la energía obtenida cada hora de cada día de 2016, 2017 y 2018.

Vemos también una alta densidad de valores atípicos elevados, y un bigote superior relativamente largo, lo que indica que los valores altos tienen una menor densidad y están más espaciados.

4.2. Metología

Para realizar los experimentos el primer paso es separar los datos en conjunto de entrenamiento y conjunto de test. Esto se ha hecho dejando los datos de los años 2016 y 2017 como conjunto de entrenamiento, y utilizando los de 2018 como conjunto de test, debido a que en un problema como éste siempre dispondremos de datos anteriores a aquellos sobre los que queremos realizar predicciones.

A continuación, siguiendo lo indicado en [2], estandarizaremos los datos de las variables regresoras, según la fórmula

$$\frac{X-\mu}{\sigma},$$

donde, para cada variable X, μ es su media, que estimamos mediante la media muestral de las observaciones del conjunto de entrenamiento, y σ es su desviación típica estimada como la desviación típica muestral sobre las mismas observaciones. Esto lo hacemos así en lugar de utilizar la muestra completa para las estimaciones para luego poder evaluar los modelos sobre el conjunto de test D_{test} y poder considerar que estos son datos desconocidos. Esto lo hacemos utilizando la función *StandardScaler* del módulo *Preprocessing* de la librería *sklearn*.

Tras haber estandarizado los datos, el siguiente paso es utilizarlos para, mediante la validación cruzada, ajustar los parámetros de los distintos modelos. Para esto utilizaremos la función *GridSearchCV* del módulo *linear_model* de la librería sklearn. Los

α	10^2	10^3	10^4	10^5	10^6
Ranking	4	3	2	1	7
Puntuación	-2801	-2800	-2798	-2781	-2822

Tabla 4.1: Validación cruzada para el modelo Ridge.

parámetros de cada modelo que ajustaremos mediante esta técnica y los respectivos rangos de valores se explican a continuación, donde vemos los aspectos concretos de la construcción de cada uno.

Regresión regularizada

Para entrenar este modelo hemos utilizado la función *Ridge* del módulo *linear_model* de la librería *sklearn*, minimizando el error cuadrático y realizando una validación cruzada sobre el parámetro de regularización. El rango que hemos utilizado es

$$\{10^k : 2 \leq k \leq 6\} \tag{4.1}$$

Al realizar la validación cruzada es importante que, dado un parámetro y su rango de valores, el mejor resultado no se obtenga para el extremo del intervalo. Es decir, si escogiésemos un rango para el que se observase una tendencia creciente, por ejemplo, en el extremo derecho del intervalo, y el mejor valor lo hemos obtenido para el propio extremo, debemos pensar que la tendencia creciente continua, y por lo tanto debemos agrandar el intervalo. Nos detendremos cuando veamos que la tendencia creciente se detiene y se empiezan a obtener peores resultados.

En este caso se ha obtenido los resultados que vemos en la tabla de la Figura 4.1. Vemos que la puntuación va mejorando según i crece. Si el intervalo hubiese terminado en $i = 4$, con el mejor valor de todos en el extremo, habríamos dejado de obtener un valor aún mejor para $i = 5$. Al obtener para $i = 7$ la puntuación más baja de todas, asumimos que la tendencia creciente ha acabado, y escogemos el mejor valor de los que tenemos: $i = 5$.

Perceptrón multicapa

Para realizar pruebas con un perceptrón multicapa, hemos utilizado la función *MLPRegressor* del módulo *neural_network* de la librería sklearn. Una vez más hemos hecho una validación cruzada sobre el parámetro de regularización α, y hemos utilizado tres capas ocultas con 100 neuronas cada una.

Para la validación cruzada, hemos probado valores de α según el rango

$$\{10^k : 1 \leq k \leq 6\}. \tag{4.2}$$

Podemos ver los valores de α y los resultados de la validación cruzada en la tabla de la Figura 4.2. Esta vez vemos, de izquierda a derecha, una tendencia descendente del error (pues la puntuación neg_mean_absolute_error crece) hasta $\alpha = 10^4$. A

α	10^1	10^2	10^3	10^4	10^5	10^6
ranking	5	4	3	1	2	6
pts. test	-1048	-1044	-1044	-1033	-1043	-1049

Tabla 4.2: Validación cruzada para el perceptrón multicapa.

C	ε	γ
$\{10^k : -1 \leq k \leq 6\}$	$\{\sigma 2^{-k} : 1 \leq k \leq 6\}$	$\{4^k d^{-1} : -2 \leq k \leq 3\}$

Tabla 4.3: Validación cruzada propuesta en [2].

partir de este valor comienza una tendencia creciente, por lo tanto, es un intervalo lógico: el mejor valor no queda en ninguno de los dos extremos.

SVR

Construimos un modelo SVR utilizando la función *SVR* del módulo *svm* de la librería sklearn. Como función kernel utilizaremos una función de base radial, y por lo tanto tenemos que dar valor al parámetro γ como podemos ver en la ecuación 2.36. Para este parámetro, para el parámetro de regularización C, y para el parámetro ε, que es la distancia que permitimos entre el valor predicho por el modelo y el valor real en cada observación del conjunto de entrenamiento, realizamos una validación cruzada.

Para seleccionar los rangos de cada parámetro, utilizamos lo indicado en [2]. Los intervalos que se proponen en el mencionado artículo son los que vemos en la tabla de la Figura 4.3, 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 regresoras, es decir, 72. σ se puede obtener mediante la función *std()* del objeto DataFrame que utilizamos para almacenar los datos, y la cantidad de variables o dimensión del problema, con la función *shape()* del objeto DataFrame.

Stacking

Construiremos este modelo utilizando la función *stackingRegressor*, del módulo *ensemble_regressor*, de la librería sklearn. Como ya hemos adelantado, los modelos individuales que compondrán nuestro modelo de ensemble serán el modelo Ridge, el perceptrón multicapa y la SVR.

Como modelo de nivel 1, se han realizado pruebas con un modelo Ridge, un perceptrón multicapa de una capa oculta y una regresión lineal, obteniendo todos ellos los mismos resultados. Por ser la regresión lineal el más simple y rápido de estos modelos, es el que se ha escogido para mostrar los resultados en este trabajo.

Modelos	Ridge	MLP	SVR	Stacking
MAE	8,66 %	6,5 %	6,35 %	6,52 %
$\boldsymbol{R^2}$	46,8 %	71,31 %	72,7 %	70,97 %

Tabla 4.4: Error absoluto medio y estadístico R^2 de cada modelo.

4.3. Resultados y conclusiones

En la tabla 4.4 vemos los resultados de todos los modelos. Observamos que el modelo Ridge es el que peores predicciones hace, pues los modelos SVR y MLP obtienen un error absoluto significativamente menor y un valor del estadístico R^2 alrededor de un 25 % mayor. Sin embargo, no podemos considerar ninguno de estos dos últimos mejor que el otro: la diferencia entre el perceptrón y la SVR no es lo suficientemente significativa, por lo que consideramos que estos modelos empatan, e igual con el stacking.

Vemos que la puntuación obtenida por el modelo stacking se mantiene en las mismas cifras que obteníamos para el perceptrón multicapa y la SVR, es decir, no ha tenido lugar la compensación de los errores de un modelo mediante mejores predicciones del otro, sin embargo, tampoco podemos decir que empeore, ya que la diferencia que hay entre los errores no es significativa. Por lo tanto asumiremos un empate entre el perceptrón, la SVR y el stacking.

Otra observación importante es que parece que las predicciones del modelo stacking se basan principalmente en las de la SVR y el perceptrón multicapa, dando menor peso a las predicciones del modelo Ridge, ya que el error que obtiene ridge es mayor que el que obtienen el perceptrón y la SVR, y que es también el del stacking. Además, enseguida veremos en algunas gráficas que el stacking tiene un error que está continuamente entre el error del perceptrón y el de la SVR, y esto lo podemos extrapolar a las predicciones.

Para estudiar por qué no se han compensado los errores de los modelos individuales, veamos qué ocurre en secciones de los datos donde haya un error significativo.

El mayor error de predicción de los tres modelos ocurre para la misma observación, el día 4 de abril de 2018, como vemos en la gráfica de la Figura 4.4. Podemos ver que este día la producción estaba siendo relativamente alta, entorno a un 70 % de la producción total, para súbitamente interrumpirse y mantenerse en cero alrededor de 5 horas, tras las cuales vuelve drásticamente a una producción muy cercana a la inicial, y luego seguir una tendencia decreciente relativamente suave.

En la misma gráfica vemos también la producción de energía que predice cada modelo, y es fácil ver que ningún modelo es capaz de predecir el periodo de producción de energía nula.

A primera vista, este parece ser un caso bastante claro en que esperaríamos que el error ligeramente menor del modelo Ridge fuese aprovechado por el modelo stacking para compensar un mayor error de los otros dos modelos.

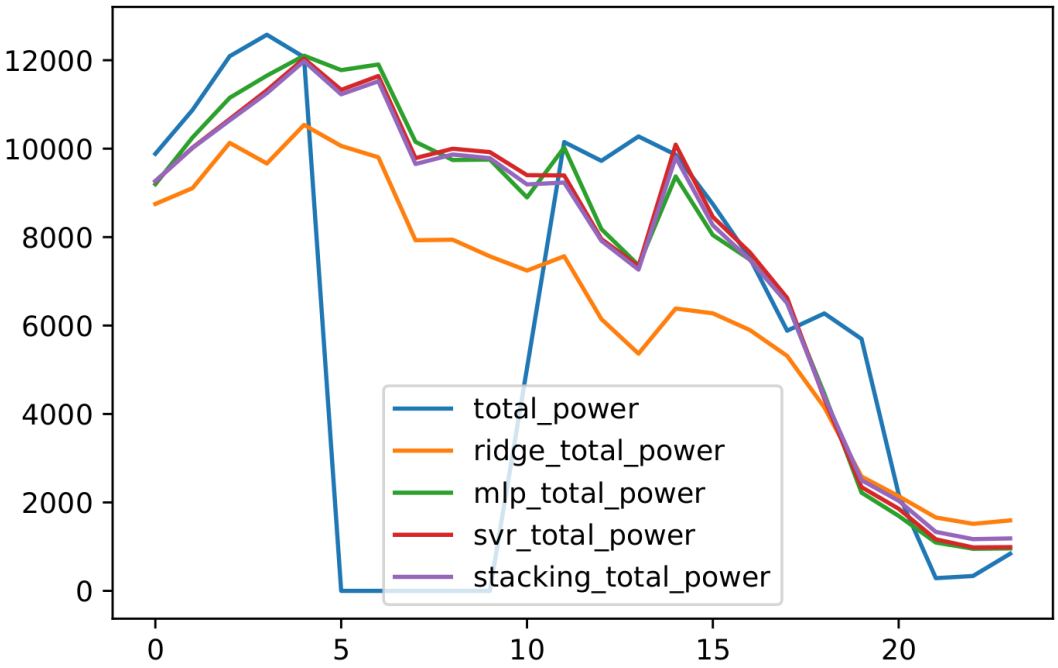


Figura 4.4: Gráfico de la obtención de energía el 4 de abril de 2018.

A partir de lo que se puede observar en el resto de la gráfica, y teniendo en mente que el error absoluto medio del modelo Ridge es significativamente mayor que el del perceptrón multicapa y que el de la SVR, podemos asumir que en general las predicciones de este modelo son peores que las de la SVR y el perceptrón. De esta manera, si el modelo stacking le da más peso a las predicciones del modelo Ridge, mejorará ligeramente la predicción para algunas secciones de producción nula, pero globalmente el error aumentará, por lo que no nos conviene.

Aún podría pensarse que la mejor opción posible sería dar más peso a las predicciones del modelo Ridge únicamente en los periodos de producción nula, para mejorar el error en ellos sin aumentarlo en el resto de casos. El problema es que para esto sería necesario anticipar dichos periodos, pero acabamos de ver que ninguno de los modelos lo consigue, con lo que esta opción no es posible.

Quizá resultaría interesante, a modo de trabajo futuro, probar como modelo de nivel 1 un árbol de decisión, pues así podríamos dar distinto peso a las predicciones de los modelos de nivel 0 dependiendo de la sección de los datos. Además, dado que parece que los periodos de producción nula no dependen de los datos de previsiones meteorológicas, si se pudiesen incorporar más datos en referencia a esto, como por ejemplo paradas de los molinos por mantenimiento planificado, es probable que esto redujese buena parte de los errores de este tipo.

Bibliografía

[1] Dorado-Moreno, M., Navarin, N., Gutiérrez, P. A., Prieto, L., Sperduti, A., Salcedo-Sanz, S. y Hervás-Martínez, C.: Multi-task learning for the prediction of wind power ramp events with deep neural networks. *ScienceDirect* (2020). Descargar.

[2] Carlos Ruiz, Carlos M. Alaíz y José R. Dorronsoro: Multitask Support Vector Regression for Solar and Wind Energy Prediction. *Energies* (2020), 7–8. Leer.

[3] Daniela Witten, Gareth James, Trevor Hastie y Robert Tibshirani: An introduction to statistical learning with applications in R. *Springer Texts in Statistics* (2017).

[4] Daniela Witten, Gareth James, Trevor Hastie y Robert Tibshirani: An introduction to statistical learning with applications in R. *Springer Texts in Statistics* (2017).

[5] Peter E. Hart, Richard O. Duda y David G. Stork: Pattern classification. *John Wiley Sons Inc* (2000).

[6] Alex J. Smola y Bernhard Schölkopf: A tutorial on support vector regression*. *Statistics and computing* (2003). Descargar.

[7] David H. Wolpert: Stacked Generalization. *ResearchGate* (1992). Leer o descargar.

[8] Amela Llobet, Miguel: Aproximación de funciones con redes neuronales. *Universitat Jaume I* (2016). Leer.

[9] Szandała, Tomasz: Review and Comparison of Commonly Used Activation Functions for Deep Neural Networks. *Cornell University* (2020). Descargar.

[10] Hecht-Nielsen: International 1989 Joint Conference on Neural Networks. Pages 593-605 vol.1.