



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:* José Luis Torrecilla Noguerales y Luis Alberto Rodríguez Ramírez

Curso 2020-2021

## Resumen

Aliquam lorem ante, dapibus in, viverra quis, feugiat a, tellus.

## Abstract

Etiam rhoncus.

# Índice general

La mayor parte de la introducción ha sido obtenida del capítulo 3, *Linear regression*, del libro *An Introduction to Statistical Learning: With Applications in R.* [**?**]

# CAPÍTULO 1
# Introducción

La estadística y el análisis de datos permiten llegar cada vez más lejos en el desarrollo de inteligencia artificial, buscando combinar el razonamiento humano con la velocidad y capacidad de procesamiento de una máquina. En este trabajo se desarrollará un método de ensemble que aprenderá la relación entre un conjunto de datos conocidos para obtener otros desconocidos, es decir, se estudiará el problema de regresión.

Tras explicar el contexto estadístico sobre el que se define el problema de regresión, se explicarán los modelos que compondrán el método de ensemble: el perceptrón simple y la *Support Vector Machine*, que implementan de dos maneras distintas una poderosa idea: la transformación no lineal adecuada permite obtener a partir de unos datos que no pueden explicarse de manera lineal, otros que sí se pueden aproximar satisfactoriamente mediante un modelo lineal.

Para dar apoyo a 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.

## 1.1. Estado del arte: modelos de ensemble

## 1.2. El problema de regresión

Un *problema de regresión* consiste en asignar a una nueva observación de una variable aleatoria un valor numérico, a partir de la información proporcionada por otras variables aleatorias. Esto nos lleva a las primeras definiciones:

**Definición 1.1.** Dado un espacio de probabilidad $(\Omega, \mathcal{A}, P)$, y un espacio medible $(S, \Sigma)$, una *variable aleatoria* X es una función

$$X : \Omega \to S$$

que es $\mathcal{A}/\Sigma$-medible. Si S es $\mathbb{R}$ y $\Sigma$ es $\mathcal{B}(\mathbb{R})$, X es una variable aleatoria real.

Nos referiremos a las distintas variables aleatorias, cuando haya más de una, con superíndices, por ejemplo $X^1, X^2, \ldots, X^p$, y a repeticiones de una misma variable aleatoria, por ejemplo de $X^1$, con subíndices, como $X_1^1, X_2^1, \ldots, X_n^1$.

**Definición 1.2.** Dadas las variables aleatorias reales $X^1, X^2, \ldots, X^p, Y$, definimos una muestra de datos de tamaño $n$ como

$$D_n = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_n, y_n)\} \subset \mathbb{R}^p \times \mathbb{R}, \tag{1.1}$$

donde

$$(\boldsymbol{x}_i, y_i) = (x_i^1, x_i^2, \ldots, x_i^p, y_i)$$

**Definición 1.3.** Dadas $p+1$ variables aleatorias reales, $X^1, X^2, \ldots, X^p, Y$, de las que conocemos una muestra $D_n$, y una función distancia $d : \mathbb{R} \times \mathbb{R} \to \mathbb{R}^+$, un problema de regresión es aquel en el que a partir de la información de la muestra, se busca una función

$$\begin{aligned} f : \mathbb{R}^p &\to \mathbb{R} \\ \boldsymbol{x} &\mapsto f(\boldsymbol{x}) \end{aligned}$$

que minimice $d(f(\boldsymbol{x}_i), y_i)$, con $(\boldsymbol{x}_i, y_i)$ una observación que puede pertenecer o no a la muestra.

Intentamos buscar una solución a un problema de regresión a partir de la hipótesis de que el comportamiento de las variables independientes nos indica en cierta medida el comportamiento de la variable dependiente.

Una primera clasificación entre los problemas de regresión, es aquella que distingue entre problemas de regresión lineal y problemas de regresión no lineal. El tipo de regresión a utilizar para resolver un problema viene determinado por la relación que haya entre los datos: si un porcentaje alto de ésta es lineal, obtendremos buenos resultados con una función de regresión lineal, pero en casos en los que la relación sea altamente no lineal, con una función lineal no conseguiremos buenos resultados, al menos de manera directa. Por lo tanto, antes de buscar ninguna solución, hay que realizar un estudio de los datos. La forma que tengan nos dará una primera intuición sobre si debemos buscar una función lineal o una no lineal.

Muchos métodos para resolver problemas de regresión no lineales se basan en transformar los datos de manera que se puedan resolver con un modelo linea. Muchos modelos no lineales son en realidad un modelo lineal con una pequeña modificación, como por ejemplo el caso de la regresión regularizada. Por este motivo, este trabajo comienza definiendo la regresión lineal simple, que trata el caso de un problema lineal donde $p = 1$, es decir, hay una única variable aleatoria independiente, y la regresión lineal múltiple o regresión lineal, donde se estudia un problema de regresión lineal con $p > 1$.

### Regresión lineal simple

En un problema de regresión lineal simple se busca explicar una variable aleatoria, Y, a partir de otra, X, asumiendo una relación lineal entre ambas más un error que sigue una distribución normal de media 0:

$$Y = f(X) = \beta_0 + \beta_1 X + \epsilon,$$

donde $\beta_0$ es el corte de la recta con el eje OY y $\beta_1$ es la pendiente de la recta, y $f(X)$ es el modelo lineal. Que exista este $\epsilon$ significa que la variable aleatoria X explica un cierto porcentaje de Y, que rara vez será su totalidad.

En el contexto habitual de un problema de regresión lineal, no conocemos los valores de $\beta_0$ ni de $\beta_1$, así que los estimamos mediante $\hat{\beta}_0$ y $\hat{\beta}_1$.

Dado que queremos estudiar una muestra de datos genérica, ya no escribiremos las observaciones como $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, sino que trataremos cada observación como a la variable aleatoria que gobierna su comportamiento. Es decir, escribiremos cada observación $(x_i, y_i)$ como $(X_i, Y_i)$. Por tanto, la muestra de datos es

$$D_n = \{(X_1, Y_1), (X_2, Y_2), \ldots, (X_n, Y_n)\} \subset \mathbb{R} \times \mathbb{R}. \tag{1.2}$$

donde nos referiremos a $X_i$ como la observación $i$, y a $Y_i$ como el *target* de la observación $i$.

Ahora obtendremos valores para $\hat{\beta}_0$ y $\hat{\beta}_1$ minimizando el error cuadrático medio visto como función de $\hat{\beta}_0$ y $\hat{\beta}_1$, sobre la muestra $D_n$:

$$J(\hat{\beta}_0, \hat{\beta}_1) = \frac{1}{n}\sum_{i=1}^{n}(Y_i - (\hat{\beta}_0 + \hat{\beta}_1 X_i))^2. \tag{1.3}$$

Estudiando la forma de esta función, vemos que dicho mínimo siempre existirá:

$$\begin{aligned} J(\hat{\beta}_0, \hat{\beta}_1) &= \frac{1}{n}\sum_{i=1}^{n}(Y_i - (\hat{\beta}_0 + \hat{\beta}_1 X_i))^2 = \\ &= \frac{1}{n}\sum_{i=1}^{n}(Y_i^2 + \hat{\beta}_0^2 + \hat{\beta}_1^2 X_i^2 + 2\hat{\beta}_0\hat{\beta}_1 X_i - 2\hat{\beta}_0 Y_i - 2\hat{\beta}_1 Y_i X_i) = \\ &= E[Y^2] + \hat{\beta}_0^2 + \hat{\beta}_1^2 E[X^2] + 2\hat{\beta}_0\hat{\beta}_1 E[X] - 2\hat{\beta}_0 E[Y] - 2\hat{\beta}_1 E[XY] = \\ &= (\hat{\beta}_0, \hat{\beta}_1)\begin{pmatrix} 1 & E[X] \\ E[X] & E[X^2] \end{pmatrix}\begin{pmatrix} \hat{\beta}_0 \\ \hat{\beta}_1 \end{pmatrix} - 2E[Y]\hat{\beta}_0 - 2E[XY]\hat{\beta}_1 + E[Y^2]. \end{aligned} \tag{1.4}$$

El primer término,

$$(\hat{\beta}_0, \hat{\beta}_1) \begin{pmatrix} 1 & E[X] \\ E[X] & E[X^2] \end{pmatrix} \begin{pmatrix} \hat{\beta}_0 \\ \hat{\beta}_1 \end{pmatrix}, \tag{1.5}$$

es una forma cuadrática definida positiva, pues su determinante es la varianza de $X$, que es positiva. Por esto sabemos que tiene un mínimo.

Los otros dos términos lineales y el término constante no cambian este hecho, ya que al derivar, su efecto es el de sumar constantes a una ecuación lineal, y esto no cambia la cantidad de ceros de la ecuación.

Derivamos respecto de $\hat{\beta}_0$ y de $\hat{\beta}_1$ y obtenemos los valores en que hacen mínimo el error cuadrático:

$$\frac{\partial}{\partial \hat{\beta}_0} J(\hat{\beta}_0, \hat{\beta}_1) = 2\hat{\beta}_0 + 2E[X]\hat{\beta}_1 - 2E[Y] = 0$$

$$\frac{\partial}{\partial \hat{\beta}_1} J(\hat{\beta}_0, \hat{\beta}_1) = 2\beta_0 E[X] + 2E[X^2]\hat{\beta}_1 - 2E[XY] = 0$$

$$\hat{\beta}_0 = E[Y] - E[X]\frac{COV(X,Y)}{\sigma^2(X)} \tag{1.6}$$

$$\hat{\beta}_1 = \frac{E[XY] - E[X]\ E[Y]}{E[X^2] - E[X]^2} = \frac{COV(X,Y)}{\sigma^2(X)}. \tag{1.7}$$

**Regresión lineal múltiple**

La regresión lineal múltiple contempla el problema en el que hay que predecir el valor de una variable dependiente $Y$ a partir de $p$ variables independientes $X^1, X^2, \ldots, X^p$. En este caso volvemos a escribir la muestra en mayúscula, para utilizar las propiedades de las variables aleatorias:

$$D_n = \{(\boldsymbol{X}_1, Y_1), (\boldsymbol{X}_2, Y_2), \ldots, (\boldsymbol{X}_n, Y_n)\} \subset \mathbb{R}^p \times \mathbb{R}, \tag{1.8}$$

donde

$$(\boldsymbol{X}_i, Y_i) = (X_i^1, X_i^2, \ldots, X_i^p, Y_i). \tag{1.9}$$

Una primera aproximación a este problema sería hacer $p$ regresiones lineales independientes, pero habría que decidir cómo predecir el valor de la variable respuesta de cada observación, pues al tener $p$ modelos, obtenemos $p$ predicciones. Además, cada predicción se basaría en una sola variable, ignorando las otras, y esto puede reducir la calidad de las predicciones si las variables tienen correlación unas con otras.

La solución que utilizaremos es adaptar el modelo de regresión simple para que en lugar de tener una función de una sola variable, tengamos una función vectorial

$$Y = \beta_0 + \boldsymbol{\beta}\boldsymbol{X} + \epsilon = \beta_0 + \sum_{i=1}^{p} \beta_i X^i + \epsilon. \tag{1.10}$$

Donde $X^i$ representa la i-ésima variable, y $\beta_i$ cuantifica la asociación entre esa variable y la variable respuesta. La interpretación de $\beta_i$ es el efecto medio sobre Y de un incremento de una unidad sobre la variable $X^i$, manteniendo fijos los demás parámetros.

En la práctica, igual que ocurría con la regresión lineal simple, los coeficientes de regresión son desconocidos, y tenemos que estimarlos. Procedemos de la misma manera, minimizando el error cuadrático esta vez como función de $p$ variables:

$$J(\hat{\beta}_0, \hat{\beta}_1, , \ldots, \hat{\beta}_p) = \sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2 = \sum_{i=1}^{n} (Y_i - \hat{\beta}_0 - \sum_{j=1}^{p} \beta_j X_i^j)^2.$$

Al contrario que con la estimación simple, las fórmulas que nos permiten minimizar esta ecuación y obtener las estimaciones de los $p + 1$ parámetros son bastante complicadas, y, dado que las podemos obtener mediante cualquier software de estadística, no las incluiremos aquí.

**Preprocesamiento de datos: Estandarización**

En el siguiente capítulo estudiaremos la regresión regularizada, el perceptrón multicapa y las máquinas de vectores soporte de regresión. Estos modelos son más complejos que la regresión multilineal, y para utilizarlos es importante regularizar nuestros datos.

Estos modelos estudian la relación que hay entre los valores de las variables independientes y las variables respuestas, y normalmente tienden a dar más peso a las variables que tengan valores más grandes.

Sin embargo, lo grandes que sean los valores de una variable dependerán de las unidades de medida. Si la altura se mide en metros podemos obtener valores entre 1,5 y 2, mientras que si se mide en centímetros, estos estarán entre 150 y 200. No tiene sentido que un modelo de más importancia a la misma información según las unidades en que esté expresada.

La solución a esto es estandarizar los datos: restamos a cada variable su media y dividimos por su desviación típica, de manera que la variable tenga media 0 y varianza 1, y por lo tanto todas las magnitudes estén alrededor del 0, alejándose una media de una unidad.

### 1.2.1. Elección de hiperparámetros: validación cruzada

Como ya hemos dicho, los modelos que vamos a estudiar tienen hiperparámetros, que influyen en gran medida en las predicciones que obtendremos. Para seleccionar qué valor deben tener los hiperparámetros de un modelo, utilizamos la validación cruzada.

Para definir la validación cruzada definiremos primero las particiones *leave-one-out* y luego un modelo de regresión como una aplicación $G$. Además, utilizaremos estas dos proyecciones que harán más sencillas las explicaciones:

$$\begin{aligned} \pi_X : \mathbb{R}^d \times \mathbb{R} &\to \mathbb{R}^d \\ (\boldsymbol{X_i}, Y_i) &\mapsto \boldsymbol{X_i} \end{aligned}, \tag{1.11}$$

$$\begin{aligned} \pi_Y : \mathbb{R}^d \times \mathbb{R} &\to \mathbb{R} \\ (\boldsymbol{X_i}, Y_i) &\mapsto Y_i \end{aligned}. \tag{1.12}$$

Sobre nuestra muestra $D_n$, tomaremos $r$ particiones leave-one-out, donde, como regla general, $r = \#(D_{train})$. A estas $r$ particiones las representaremos mediante la letra $\theta$, como $\theta_1, \theta_2, \ldots, \theta_r$. Esta manera de particionar consiste en dividir $D_{train}$ en dos subconjuntos disjuntos: el conjunto de validación y el conjunto de entrenamiento.

Para la partición $\theta_i$, al conjunto de entrenamiento lo llamamos $\theta_{itrain}$, y al de validación, $\theta_{ival}$. Los elementos que componen cada subconjunto son:

$$\begin{aligned} \theta_{ival} &= \{(\boldsymbol{X_i}, Y_i)\} \\ \theta_{itrain} &= D_{train} \setminus \theta_{ival}. \end{aligned} \tag{1.13}$$

Es importante notar que se tienen que cumplir estas dos condiciones:

1. $\theta_{ival} \neq \theta_{jval}$ siempre que $i \neq j$.
2. $\bigcup_{i=1}^{r} \theta_{ival} = D_{train}$, ya que $r = \#(D_{train})$.

Ahora, para un conjunto de entrenamiento $\theta_{itrain}$ de $m$ observaciones y una partición $\theta_i$, definimos función de regresión

$$g_m : \{\theta_{itrain};\ \pi_X(\theta_{ival})\} \to \mathbb{R} \tag{1.14}$$

como una función que se entrena sobre un conjunto de $m$ observaciones, $\theta_{itrain}$ y da una predicción del target de $\pi_X(\theta_{ival})$.

A partir de $g_m$, definimos modelo de regresión individual

$$G = \{g_i\}_{i=1}^{\infty}, \tag{1.15}$$

donde $\{g_i\}_{i=1}^{\infty}$ es una familia de funciones de regresión de cardinal numerable. El objetivo de $G$ es que se utilice como una función de regresión $g_m$ para un $m$ genérico. $g_m$ se puede aplicar cuando $\#(\theta_{itrain}) = m$, sin embargo, $G$ se puede aplicar para un $\theta_{itrain}$ de cualquier tamaño. Aunque esto suene algo enrevesado, más adelante, en los modelos de ensemble, entrenaremos un mismo modelo sobre distintos conjuntos de aprendizaje, con lo que esta definición es necesaria.

$G(\theta_{itrain}, \pi_X(\theta_{ival}))$ equivale a la salida de la función m-ésima de $G$ para los mismos parámetros, con $m = \#(\theta_{itrain})$.

Volvemos a los hiperparámetros: definimos un conjunto finito de posible valores para los hiperparámetros que queremos calibrar, y definir un modelo con cada posible combinación. Digamos que obtenemos una familia de modelos de regresión $G_1, G_2, \ldots, G_k$.

Ahora damos a cada modelo una puntuación que mide cómo de bien predice ese modelo el target de una observación cuando se ha entrenado con la parte restante del conjunto de entrenamiento. De manera más rigurosa, asignamos a cada modelo la media, para las $r$ particiones, del error que tiene al predecir el target de la observación $\theta_{ival}$ cuando se entrena con $\theta_{itrain}$:

$$VC(G, \{\theta_1, \theta_2, \ldots, \theta_r\}) = \frac{1}{r}\sum_{i=1}^{r}\left(G(\theta_{itrain}, \pi_X(\theta_{ival})) - \pi_Y(\theta_{ival})\right)^2 \tag{1.16}$$

Nos quedamos con los parámetros del modelo que menor puntuación obtenga.

En ocasiones, trabajar con particiones leave-one-out tiene un coste temporal demasiado alto. En estos casos es una buena alternativa particionar según el "Método de agrupamiento para el manejo de datos"(siglas en inglés GMDH), donde la cantidad de particiones, $r$, ya no es $\#(D_{train})$, sino algún divisor. Si tomamos $m = \#(D_{train})$, tendremos:

1. $\#(\theta_{itrain}) = \frac{m}{r}$.
2. $\theta_{itrain} \bigcap \theta_{jtrain} = \{\emptyset\}$ siempre que $i \neq j$.

# CAPÍTULO 2
# Modelos de regresión

## 2.1. Regresión regularizada

La regresión regularizada es un modelo de regresión muy similar al de regresión multilineal, en el que penalizamos los coeficientes de las variables independientes $\beta_1, \beta_2, \ldots, \beta_n$. Escribimos una nueva función de error $\widetilde{J}$, y vemos su relación con la función de error de la regresión multilineal $J$:

(2.1)
$$\widetilde{J}(\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_p) = \sum_{i=1}^{n}(Y_i - \hat{\beta}_0 - \sum_{i=1}^{p}\hat{\beta}_i X_i)^2 + \lambda \sum_{j=1}^{p}\hat{\beta}_j^2 = J(\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_p) + \lambda \sum_{j=1}^{p}\hat{\beta}_j^2.$$

El hiperparámetro $\lambda$ balancea 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.

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 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.

Por último, el coste computacional de la regresión regularizada es prácticamente el mismo que el de la regresión multilineal, con lo que 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, que llegan a dónde las redes neuronales sin capa oculta se quedan cortas, permitiéndonos resolver problemas no lineales.

Una red neuronal consiste en capas con neuronas, donde cada neurona es una función no lineal que recibe una suma ponderada de las salidas de las neuronas de la capa anterior o de los atributos de la observación, y cuya salida es a su vez parte de la suma ponderada que será la entrada de las neuronas de la siguiente capa. La clave de estos modelos es que admiten unos algoritmos relativamente simples, en los que la función no lineal se puede aprender a partir del conjunto de datos.

Uno de los métodos más conocidos para entrenar una red neuronal multilineal es el retropropagación, que estudiamos en cierta profundidad, ya que es un método potente y útil a la vez que relativamente simple, y dará pie a entender otros métodos más complicados como modificaciones de éste.

Vamos a ver que cada problema tiene una arquitectura o topología de red neuronal multilineal propia, y aquí es donde entra en juego el conocimiento que tengamos sobre el dominio del problema: incluso las ideas más informales o heurísticas pueden incorporarse fácilmente a nuestro modelo, modificando el número de capas ocultas y el número de neuronas de cada una de ellas. La simplicidad del método de retropropagación permiten probar distintas alternativas, sin que esto suponga un gran trabajo.

Una de las mayores dificultades a la hora de utilizar redes neuronales es ajustar su complejidad, es decir, seleccionar el modelo. 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.

Antes de empezar a definir las redes neuronales formalmente, hacemos hincapié en que como con cualquier modelo, las técnicas que veremos para redes neuronales no eximen de adquirir un conocimiento profundo del problema.

### 2.2.1. Estructura y alcance de las redes neuronales

Las redes neuronales que vamos a estudiar tienen capa de entrada, capa oculta 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 la capa oculta en cambio deberá calcularla 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.

Las redes neuronales que estudiaremos y utilizaremos tendrán dos funciones: las neuronas de la capa de entrada y de la capa de salida, tendrán la función identidad, y las neuronas de la capa oculta tendrán una función definida a trozos conocida como rectificador o $ReLU$:

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

La entrada de las neuronas, es decir, de sus funciones, será una combinación lineal de las salidas de las neuronas de la capa anterior y la neurona del sesgo, a excepción de las neuronas de la capa de entrada. En estas últimas, la entrada de la neurona $i$ es el valor del atributo $i$ de la observación.

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

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

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

$$\boldsymbol{z} = (z_1, \ldots, z_{n_3}),$$

$$z_k(\boldsymbol{y}) = f\Big(\sum_{j=0}^{n_2} w_{kj} y_j\Big), \tag{2.3}$$

donde $w_{kj}$ son los pesos que unen las neuronas de la capa oculta con la neurona k de la capa de salida. Además, sabemos que la salida de cada neurona de la capa oculta es

$$y_j(\boldsymbol{x}) = f\Big(\sum_{i=0}^{n_1} w_{ji} x_i\Big). \tag{2.4}$$

Con $w_{ji}$ los pesos que unen las neuronas de la capa de entrada con la neurona j de la capa oculta, análogamente al caso anterior. Sustituyendo obtenemos la salida de la red neuronal en función de la entrada:

$$z_k(\boldsymbol{x}) = f\Big(\sum_{j=0}^{n_2} w_{kj} f\Big(\sum_{i=0}^{n_1} w_{ji} x_i\Big)\Big). \tag{2.5}$$

**¿Qué funciones puede representar una red neuronal?**

El teorema de Kolmogorov demuestra que cualquier función continua $g(x)$ cuyo dominio esté restringido a un hipercubo $I^n : I = [0, 1], n \geq 2$ puede expresarse en la forma

$$g(x) = \sum_{j=1}^{2n+1} \Xi_j (\sum_{i=1}^{d} \Psi_{ij}(x_i)) \tag{2.6}$$

para funciones $\Xi_j$ y $\Psi_{ij}$ adecuadas.

Este teorema nos permite asegurar que cualquier función continua puede expresarse en una red neuronal de 3 capas, si ponemos $2n + 1$ neuronas ocultas, cada una con una función $\Xi_j$, y teniendo en cada una de las neuronas de la capa de entrada, que deberán ser $d(2n + 1)$, una función $\Psi_{ij}$. 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.

Sabemos que un conjunto completo de funciones, por ejemplo los polinomios, pueden representar cualquier función, sin embargo, esto también puede conseguirse utilizando una sola función, siempre que utilicemos los parámetros adecuados. Esto es lo que queremos conseguir con las redes neuronales multilineales, cuyas neuronas siempre tendrán la misma función de transferencia.

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.

### 2.2.2. Algoritmo de retropropagación

Las redes neuronales tienen dos modos de funcionamiento: feedforward y aprendizaje. El feedforward consiste en introducir una observación en la capa de entrada para obtener un resultado por la capa de salida, mientras que aprendizaje (supervisado) consiste en introducir una observación en la capa de entrada pero conociendo el resultado correcto, y según sea el producido por la red, ajustar los pesos para que se parezcan lo máximo posible.

Sin embargo, 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 solución.

#### Aprendizaje de la red

Queremos medir la distancia entre cada predicción y el resultado real, así que 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):

$$J(\mathbf{w}) = \frac{1}{2}\sum_{k=1}^{n_3}(t_k - z_k)^2 = \frac{1}{2}(\mathbf{t} - \mathbf{z})^2, \tag{2.7}$$

siendo $\mathbf{t}$ y $\mathbf{z}$ son los vectores esperado (target) y obtenido, 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 como mucho, será 0). Representamos el incremento de **w** así:

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

o, componente a componente:

$$\Delta w_{mn} = -\eta \frac{\partial J}{\partial w_{mn}}, \tag{2.9}$$

donde $\eta$ 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:

$$net_k = \sum_{j=0}^{n_2} w_{kj} y_j \tag{2.10}$$

$$net_j = \sum_{i=0}^{n_1} w_{ji} x_j \tag{2.11}$$

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

$$\frac{\partial J}{\partial w_{kj}} = \frac{\partial J}{\partial net_k} \frac{\partial net_k}{\partial w_{kj}}, \tag{2.12}$$

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

$$\delta_k = -\frac{\partial J}{\partial net_k} = -\frac{\partial J}{\partial z_k} \frac{\partial z_k}{\partial net_k} = (t_k - z_k) f'(net_k). \tag{2.13}$$

Luego, derivando respecto de $w_{kj}$ en la ecuación 2.10, obtenemos

$$\frac{\partial net_k}{\partial w_{kj}} = y_j. \tag{2.14}$$

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

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

$$\boxed{\Delta w_{kj} = \eta \delta_k y_j = \eta (t_k - z_k) f'(net_k) y_j} \,. \tag{2.15}$$

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

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

Desarrollamos el primer término:

$$\frac{\partial J}{\partial y_j} = \left( \frac{\partial}{\partial y_j} \right) \left( \frac{1}{2} \sum_{k=1}^{n_3} (t_k - z_k)^2 \right) = -\sum_{k=1}^{n_3} (t_k - z_k) \frac{\partial z_k}{\partial y_j} =$$
$$= -\sum_{k=1}^{n_3} (t_k - z_k) \frac{\partial z_k}{\partial net_k} \frac{\partial net_k}{\partial y_j} = -\sum_{k=1}^{n_3} (t_k - z_k) f'(net_k) \frac{\partial net_k}{\partial y_j}.$$

Ahora derivamos respecto de $y_j$ en la ecuación 2.10, obteniendo

$$\frac{\partial J}{\partial y_j} = -\sum_{k=1}^{n_3} (t_k - z_k) f'(net_k) w_{kj}. \tag{2.16}$$

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

$$\delta_j = f'(net_j) \sum_{k=1}^{n_3} w_{kj} \delta_k = f'(net_j) \sum_{k=1}^{n_3} w_{kj} (t_k - z_k) f'(net_k) \,. \tag{2.17}$$

Además, derivando respecto de $w_{ji}$ en la ecuación de $net_j$ (ecuación 2.11), obtenemos

$$\frac{\partial net_j}{\partial w_{ji}} = x_i \,, \tag{2.18}$$

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

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

$$\boxed{\Delta w_{ji} = \eta\left(\sum_{k=1}^{n_3}(t_k - z_k)f'(net_k)w_{jk}\right)f'(net_j)x_i}\,. \tag{2.19}$$

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

$$\Delta w_{ji} = \eta\left(\sum_{k=1}^{n_3}\delta_k w_{jk}\right)f'(net_j)x_i$$

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

## 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 que consiste en una función $f(\boldsymbol{x})$ que, en las observaciones del conjunto de aprendizaje, tenga un error menor que un $\varepsilon$ definido, y que al mismo tiempo sea lo más plana posible. En este contexto, que una función sea plana significa que sea lo más parecida posible a una función constante.

Conocemos un conjunto de datos $D_n$ como en la definición 1.2. A partir de estos datos construiremos $f$.

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

$$f(\boldsymbol{x}) = \langle \boldsymbol{w}, \boldsymbol{x}\rangle + b \quad : \boldsymbol{w} \in \mathbb{R}^p, b \in \mathbb{R}. \tag{2.20}$$

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

$$\min \quad \frac{1}{2}\|w\|^2 \quad : \quad \begin{cases} y_i - \langle w, x_i\rangle - b \leq \varepsilon \\ \langle w, x_i\rangle + b - y_i \leq \varepsilon \end{cases}\,. \tag{2.21}$$

Esta $f$ siempre existe ya que es la solución de un problema de optimización convexa.

En ocasiones queremos permitir cierta cantidad de error, que se traduce en introducir las llamadas *slack variables*, $\xi_i, \xi_i^*$:

$$\min \quad \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i^*) \quad : \quad \begin{cases} y_i - \langle w, x_i\rangle - b \leq \varepsilon + \xi_i \\ \langle w, x_i\rangle + b - y_i \leq \varepsilon + \xi_i^* \end{cases}\,. \tag{2.22}$$

La constante $C$ balancea la importancia de que $f$ sea plana y la cantidad hasta la que toleramos desviaciones mayores que $\varepsilon$. Es importante el detalle de que sólo las desviaciones mayores que $\varepsilon$ tienen coste.

En la mayoría de los casos, resultará más sencillo, y es lo que haremos, resolver la ecuación 2.22 en su expresión dual.

Construimos el problema dual hallando el máximo de la función en lugar del mínimo, invirtiendo las inecuaciones y añadiendo el conjunto dual de variables, que son los multiplicadores de Langrange $\eta_i^{(*)}$ y $\alpha_i^{(*)}$:

$$\begin{aligned} L :=& \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i^*) - \sum_{i=1}^{n}(\eta_i\xi_i + \eta_i^*\xi_i^*) \\ &- \sum_{i=1}^{n}\alpha_i(\varepsilon + \xi_i - y_i + \langle \boldsymbol{w}, \boldsymbol{x_i}\rangle + b) \\ &- \sum_{i=1}^{n}\alpha_i^*(\varepsilon + \xi_i^* + y_i - \langle \boldsymbol{w}, \boldsymbol{x_i}\rangle - b) \end{aligned} \quad . \tag{2.23}$$

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

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

$$\partial_{\boldsymbol{w}} L = \boldsymbol{w} - \sum_{i=1}^{n}(\alpha_i - \alpha_i^*)\boldsymbol{x_i} = 0, \tag{2.25}$$

$$\partial_{\xi_i^{(*)}} L = C - \alpha_i^{(*)} - \eta_i^{(*)} = 0. \tag{2.26}$$

Sustituyendo las ecuaciones 2.24, 2.25 y 2.26 en 2.23, obtenemos el problema de optimización dual

$$\begin{aligned} \text{máx} & \begin{cases} -\frac{1}{2}\sum_{i,j=1}^{n}(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)\langle \boldsymbol{x_i}, \boldsymbol{x_j}\rangle \\ -\varepsilon\sum_{i=1}^{n}(\alpha_i + \alpha_i^*) + \sum_{i=1}^{n} y_i(\alpha_i - \alpha_i^*) \end{cases} \\ : \quad & \textstyle\sum_{i=1}^{n}(\alpha_i - \alpha_i^*) = 0 \quad \text{y} \quad \alpha_i, \alpha_i^* \in [0, C] \end{aligned} \quad , \tag{2.27}$$

donde hemos eliminado las variables $\eta_i^{(*)}$ despejando en la ecuación 2.26.

A partir de la ecuación 2.25, podemos despejar $\boldsymbol{w}$ y que quede expresado en función de los $\boldsymbol{x_i}$ por un coeficiente:

$$\boldsymbol{w} = \sum_{i=1}^{n}(\alpha_i - \alpha_i^*)\boldsymbol{x_i}, \tag{2.28}$$

y por tanto

$$f(\boldsymbol{x}) = \sum_{i=1}^{n}(\alpha_i - \alpha_i^*)\langle x_i, \boldsymbol{x}\rangle + b. \tag{2.29}$$

Vemos que $\boldsymbol{w}$ puede expresarse como una combinación lineal de las observaciones $\boldsymbol{x_i}$. Además, el algoritmo consiste únicamente en hacer productos escalares entre observaciones.

Ahora falta calcular b, para lo que utilizaremos las condiciones de Karush-Kuhn-Tucker, que dicen que la solución de nuestro problema también es solución de las siguientes cuatro ecuaciones:

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

$$\alpha_i^*(\varepsilon + \xi_i^* + y_i - \langle \boldsymbol{w}, \boldsymbol{x_i}\rangle - b) = 0, \tag{2.31}$$

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

y

$$(C - \alpha_i^*)\xi_i^* = 0. \tag{2.33}$$

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 una observación $i$ sale del tubo épsilon, entonces o $\xi_i$ o $\xi_i^*$ son distintos de cero. Digamos que $\xi_i$ es distinto de cero y el otro caso es análogo, intercambiando el $*$. Entonces, $\xi_i^*$ 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 $\xi_i$ no es cero, para que se cumpla la tercera condición, tiene que cumplirse $\alpha_i = C$. Además, al ser $\xi_i$ 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 $\alpha$ o $\alpha^*$ 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

$$\begin{array}{lll} \varepsilon - y_i + \langle \boldsymbol{w}, \boldsymbol{x_i} \rangle + b \geq 0 \quad \text{y} \quad \xi_i = 0 & & \text{si } \alpha_i < C \\ \varepsilon - y_i + \langle \boldsymbol{w}, \boldsymbol{x_i} \rangle + b \leq 0 & & \text{si } \alpha_i > 0 \end{array} . \tag{2.34}$$

Haciendo el estudio análogo sobre $\alpha_i^*$, obtenemos

$$\begin{array}{l} \text{máx}\{-\varepsilon + y_i - \langle w, x_i \rangle \mid \alpha_i < C \text{ o } \alpha_i^* > 0\} \leq b \leq \\ \text{mín}\{-\varepsilon + y_i - \langle w, x_i \rangle \mid \alpha_i > 0 \text{ o } \alpha_i^* < C\} \end{array} , \tag{2.35}$$

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

### Funciones kernel

El siguiente paso es conseguir que el algoritmo de SVR sea no lineal. En principio podríamos conseguirlo simplemente aplicando a los datos una función no lineal $\phi : \mathcal{X} \to \mathcal{F}$ y luego aplicar el algoritmo como lo hemos descrito hasta ahora. El problema es que aplicar una función $\phi$ 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 $\phi$ 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. De esta forma, utilizaremos ciertas funciones $K : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ que son iguales a este producto escalar, para alguna $\phi$ no lineal, es decir, que se cumple $K(\boldsymbol{x_i}, \boldsymbol{x_j}) = \langle \phi(\boldsymbol{x_i}), \phi(\boldsymbol{x_i}) \rangle$.

Entonces, en problemas no lineales tenemos que resolver una versión de 2.27 modificada, en la que cambiamos $x$ por $\phi(x)$, es decir, $\langle \boldsymbol{x_i}, \boldsymbol{x_j} \rangle$ por $K(\boldsymbol{x_i}, \boldsymbol{x_j})$:

$$\begin{array}{l} \text{máx} \left\{ \begin{array}{l} -\dfrac{1}{2} \displaystyle\sum_{i,j=1}^{n} (\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*) K(\boldsymbol{x_i}, \boldsymbol{x_j}) \\ \qquad -\varepsilon \displaystyle\sum_{i=1}^{n} (\alpha_i + \alpha_i^*) + \sum_{i=1}^{n} y_i (\alpha_i - \alpha_i^*) \end{array} \right. \\ : \quad \sum_{i=1}^{n} (\alpha_i - \alpha_i^*) = 0 \quad \text{y} \quad \alpha_i, \alpha_i^* \in [0, C] \end{array} . \tag{2.36}$$

Además, tendremos

$$\boldsymbol{w} = \sum_{i=1}^{n}(\alpha_i - \alpha_i^*)\phi(\boldsymbol{x_i}), \tag{2.37}$$

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

$$f(\boldsymbol{x}) = \sum_{i=1}^{n}(\alpha_i - \alpha_i^*)K(\boldsymbol{x_i}, \boldsymbol{x_j}) + b. \tag{2.38}$$

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

Algunos ejemplos de funciones kernel son

$$K(\boldsymbol{x}, \boldsymbol{x'}) = (\langle \boldsymbol{x}, \boldsymbol{x'} \rangle + c)^p \quad : p \in \mathbb{N}, c \geq 0, \tag{2.39}$$

$$K(\boldsymbol{x}, \boldsymbol{x'}) = \widetilde{K}(\boldsymbol{x} - \boldsymbol{x'}) = \exp\left( - \frac{\|\boldsymbol{x} - \boldsymbol{x'}\|^2}{2\sigma^2} \right), \tag{2.40}$$

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 $\sigma$ es la norma de la desviación típica de la variable $X$.

# CAPÍTULO 3
# Modelos de ensemble

## 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 son combinados para obtener mejores resultados.

Utilizaremos el término modelo individual para hacer referencia a un modelo tradicional que no está compuesto por otros modelos, en contraste con un modelo de ensemble, que sí estará compuesto por otros modelos; de hecho, por modelos individuales.

La solución ideal al problema de regresión sería un modelo que tuviese suficientes grados de libertad como para resolver la complejidad subyacente de los datos, pero no demasiados para que no sobreajuste.

El rendimiento de los modelos que sobreajustan suele variar mucho según la forma de los datos sobre los que se hagan predicciones, es decir, tienen varianza alta, y esto es algo no deseable.

Los modelos individuales pueden combinarse entre sí para formar modelos más complejos, que serán los modelos de ensemble. A menudo tenemos modelos individuales que no dan muy buenos resultados por tener mucho sesgo (no tienen suficientes grados de libertad), o que por el contrario tienen demasiada varianza (demasiados grados de libertada). Los métodos de ensemble construyen un método con menor sesgo o menor varianza que obtenga mejores resultados.

Un modelo de ensemble es un modelo que hace predicciones basadas en las predicciones de ciertos modelos individuales. El primer paso para construir un modelo de ensemble es seleccionar los modelos base que vamos a utilizar. La mayoría de las veces (bagging, boosting) utilizaremos un único modelo y variaremos los 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, cosa que puede tener grandes beneficios. Entonces decimos que el modelo de ensemble es heterogéneo.

### 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, cosa que puede suponer un ahorro muy significativo de tiempo.

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

Stacking es un método de ensemble que consiste en entrenar varios modelos individuales que pueden ser heterogéneos. Este es el modelo que empleamos en los experimentos, así que le dedicaremos más tiempo en la siguiente parte de este trabajo.

## 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*.

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 $N \geq 1$.

En validación cruzada, por ejemplo, que es una solución muy popular a este problema, nos quedamos con un único modelo que consideramos el mejor, es decir, que ha obtenido la mejor puntuación según una métrica que hemos definido. Esto es una solución relativamente simple y de bajo coste, pero tiene la desventaja de que descartamos todos los modelos menos uno, desentendiéndonos de cualquier aportación que pudiesen hacer.

Al trabajar con modelos de ensemble, en cambio, involucramos a la familia completa de modelos de regresión individuales, $\{G_j\}$, obteniendo así un modelo más potente. 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 fallo no tiene por qué ser la misma

que para otro modelo. Con cierta elección cuidadosa de los modelos individuales que compondrán nuestro modelo de ensemble, esperamos que no lo sean.

A modo de adelanto para facilitar la comprensión del método stacking, 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 en la cual toma parte cierto subconjunto de los N modelos. A continuación, entrena un modelo sobre los datos de nivel 1, y éste es el que se utiliza para hacer predicciones. Como los datos sobre los que queremos hacer predicciones 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 a $\mathbb{R}^d \times \mathbb{R}$, que es donde viven los elementos de $D_n$, y espacio de nivel 1 a $\mathbb{R}^k \times \mathbb{R}$. Puede haber espacios de otros niveles, aunque para esta definición solo nos hacen falta estos dos.

Los elementos del espacio de nivel 1 también los escribimos con la forma habitual $(\widetilde{\boldsymbol{X}}_i, \widetilde{Y})$. Cada observación de este espacio se construye a partir k modelos de regresión individuales de nuestro conjunto $\{G_j\}$, fijando una partición $\theta_i$ y haciendo k predicciones $G_j(\theta_{itrain}, \pi_X(\theta_{ival}))$. Así obtenemos k valores, que serán la componente $\widetilde{\boldsymbol{X}}_i$ de nuestro elemento $(\widetilde{\boldsymbol{X}}_i, \widetilde{Y})$.

La manera de obtener $\widetilde{Y}$, es mediante una transformación biyectiva $\phi(Y)$. En nuestro caso, utilizaremos la identidad en $\mathbb{R}$, es decir, no modificaremos $Y$:

$$\begin{aligned} \phi : \mathbb{R} &\to \mathbb{R} \\ Y_i &\mapsto Y_i \end{aligned}. \tag{3.1}$$

Estos últimos párrafos se corresponden con la figura 3.1.

Para $\theta_i$ fija, hemos obtenido un elemento del espacio de nivel 1, por lo que rotando $i$, en total obtendremos r elementos. Sobre este nuevo conjunto de entrenamiento, que podemos llamar conjunto de entrenamiento de nivel 1 o reducido, entrenaremos un último modelo de regresión individual $\widetilde{G} \in G_j$ de la manera habitual.

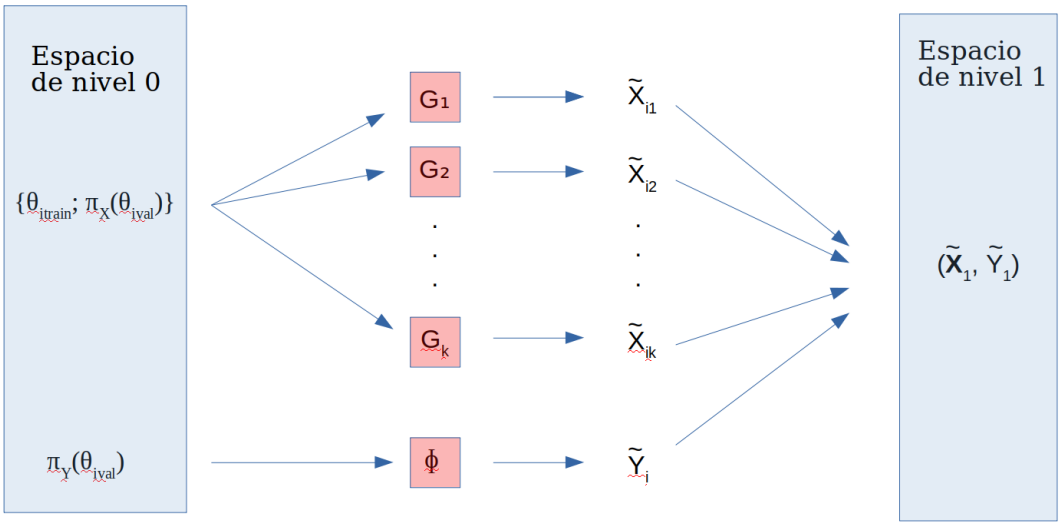



Figura 3.1: En esta imagen vemos como se genera una observación genérica $(\widetilde{\boldsymbol{X}}_i, \widetilde{Y}_i)$ del espacio de nivel 1, a partir de todo $D_{train}$ utilizando una partición $\theta_i$.

Llegados a este punto ya sabemos construir el modelo final $\widetilde{G}$ que se ha entrenado sobre el conjunto de entrenamiento de nivel 1, y que por tanto predice sobre el espacio de nivel 1 (figura 3.1). Solo falta definir cómo llevar una observación a dicho nivel para predecir su target, y cómo traer esta predicción de vuelta al nivel 1.

Digamos que $\boldsymbol{X} \in \mathbb{R}^d$ es una observación en el espacio de nivel 0 y $\widetilde{\boldsymbol{X}} \in \mathbb{R}^k$ su transformación al espacio de nivel 1. Entonces tenemos

$$\widetilde{\boldsymbol{X}} = (G_1(D_n; (\boldsymbol{X}, 0)), G_2(D_n; (\boldsymbol{X}, 0)), \ldots, G_k(D_n; (\boldsymbol{X}, 0))), \tag{3.2}$$

es decir, siguiendo el mismo procedimiento que para obtener el conjunto de entrenamiento de nivel 1 o reducido, solo que esta vez entrenando los modelos $G_1, G_2, \ldots, G_k$ sobre toda la muestra $D_n$, en lugar de sobre un subconjunto de una de las particiones $\theta_i$. Sobre esta observación del espacio de nivel 1, $\widetilde{\boldsymbol{X}}$, ya podemos predecir con $\widetilde{G}$, para obtener $\widetilde{Y}$.

Solo queda llevar $\widetilde{Y}$ del espacio de nivel 1 al espacio de nivel 0. Esto lo hacemos mediante la inversa de la aplicación que llevaba targets del nivel 0 al nivel 1, es decir, con $\phi^{-1}$. Recordando como habíamos definido $\phi$, tenemos

$$\phi^{-1}(\widetilde{Y}) = \widetilde{Y}. \tag{3.3}$$

Y de aquí obtenemos finalmente nuestra predicción (figuras 3.2 y 3.3).

Este es el proceso completo, que puede iterarse para obtener p niveles, con $p \geq 1$.

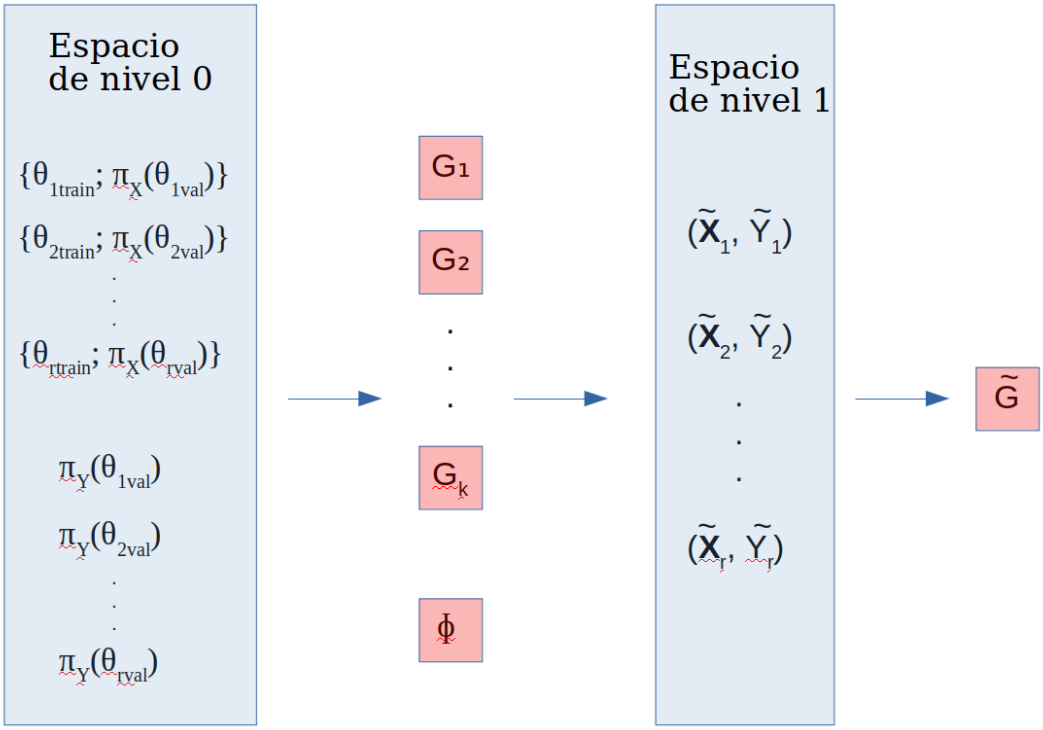



Figura 3.2: En esta imagen tenemos una visión general de un método de ensemble con stacking.

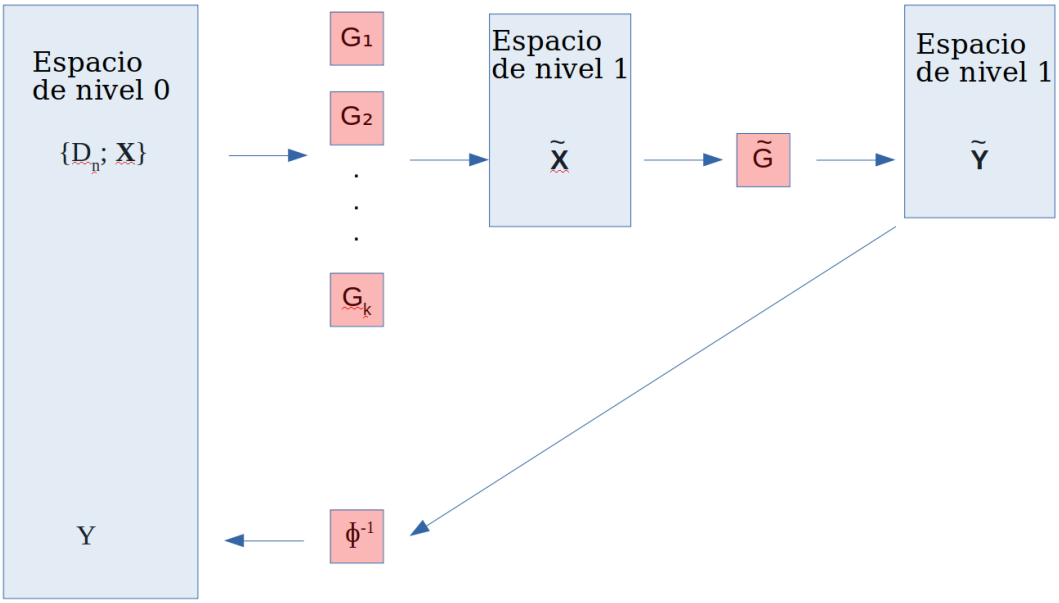



Figura 3.3: Esquema de una predicción utilizando un modelo de ensemble ya entrenado.

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.

# CAPÍTULO 4
# Simulaciones y resultados

## 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.

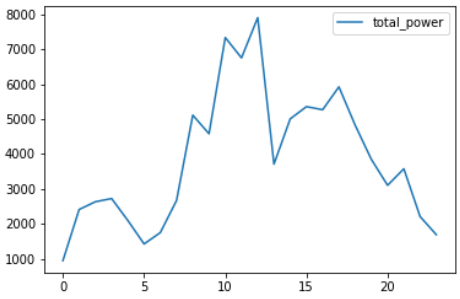
El Parque Eólico de Sotavento, al que a partir de ahora nos referiremos simplemente como parque de Sotavento, está situado en las coordenadas (43.354377º, -7.881213º). Conocemos las medidas atmosféricas que acabamos de explicar en una rejilla de coordenadas de precisión 0,125º, por lo que la coordenada 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, que son:

$$\begin{array}{ccc} (43{,}5,\ -8) & (43{,}375,\ -8) & (43{,}25,\ -8) \\ (43{,}5,\ -7{,}875) & (43{,}375,\ -7{,}875) & (43{,}25,\ -7{,}875) \\ (43{,}5,\ -7{,}75) & (43{,}375,\ -7{,}75) & (43{,}25,\ -7{,}75) \end{array} .$$

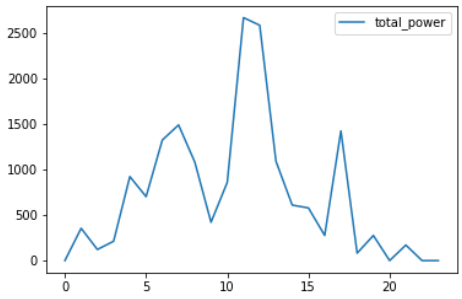
Como en cada una de estas 9 coordenadas tenemos las 8 medidas anteriores, en total tenemos 72 variables independientes.

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

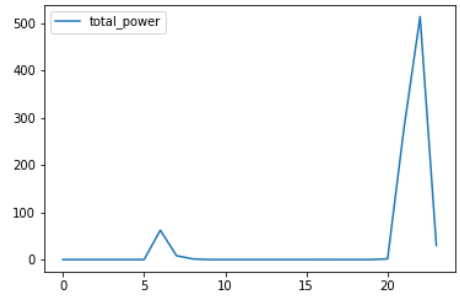



(a) 2 de enero 2018

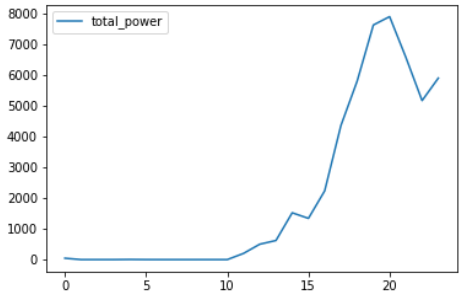



(b) 6 de julio 2018

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.

dirección del viento hacen que predecir la cantidad de energía eólica sea un problema en 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

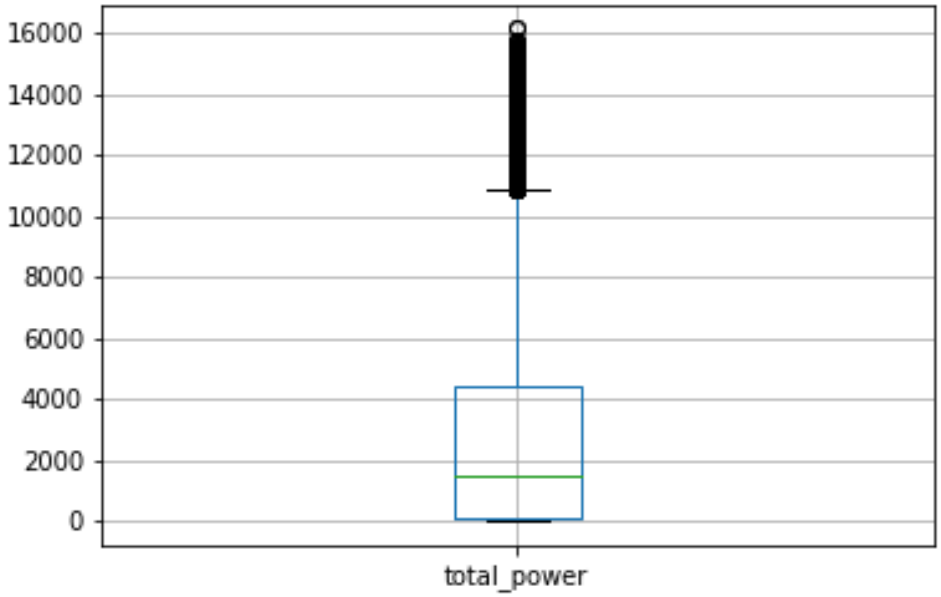



Figura 4.3: Gráfico de caja de la energía obtenida cada hora de cada día de 2016, 2017 y 2018.

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. 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.

Para realizar los experimentos, dividimos los datos en conjunto de entrenamiento y conjunto de test. Esto se ha hecho dejando los datos de los años 2016 y 2017 para conjunto de entrenamiento, y utilizando los de 2018 como conjunto de test. El motivo es que los experimentos se parezcan lo máximo posible al uso que daríamos posteriormente a los modelos: aprender de los datos de días anteriores para predecir sobre el futuro.

## 4.2. 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.

| $\boldsymbol{\alpha}$ | $10^2$ | $10^3$ | $10^4$ | $10^5$ | $10^6$ |
|---|---|---|---|---|---|
| **Ranking** | 4 | 3 | 2 | 1 | 7 |
| **Puntuación** | -2801 | -2800 | -2798 | -2781 | -2822 |

Tabla 4.1: Validación cruzada para el modelo Ridge.

Para la validación cruzada utilizamos la función *GridSearchCV*, del módulo *model_selection*, una vez más de la librería sklearn. El rango de valores para el parámetro de regularización 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 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: $i = 5$.

Con los parámetros obtenidos, el modelo Ridge tiene un error absoluto medio del 8,66 % sobre el máximo de energía que puede obtener el parque: 17560 vatios. Esto son alrededor de 1500 vatios.

El estadístico $R^2$, que nos indica el porcentaje de variación de la energía obtenida que hemos conseguido explicar mediante el modelo Ridge, es del 46,8 %.

Debido a que estamos utilizando un modelo lineal para predecir la energía obtenida a partir de las predicciones meteorológicas, cuando entre estas variables hay una relación no lineal, no son los mejores resultados. Sin embargo, dada la simplicidad del modelo y que el error que obtenemos es relativamente pequeño, como veremos al estudiar los errores del perceptrón multicapa y la SVR, lo incluiremos en el modelo de ensemble.

| $\alpha$ | $10^1$ | $10^2$ | $10^3$ | $10^4$ | $10^5$ | $10^6$ |
|---|---|---|---|---|---|---|
| **ranking** | 5 | 4 | 3 | 1 | 2 | 6 |
| **pts. test** | -1048 | -1044 | -1044 | -1033 | -1043 | -1049 |

Tabla 4.2: Validación cruzada para el perceptrón multicapa.

## 4.3. 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 $\alpha$, y hemos utilizado tres capas ocultas con 100 neuronas cada una.

Para la validación cruzada, hemos probado valores de $\alpha$ según el rango

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

Podemos ver los valores de $\alpha$ y los resultados de la validación cruzada en la tabla 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 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.

Para este modelo obtenemos un error absoluto medio del 6,5 %, es decir, entorno a los 1150 vatios. Hemos conseguido una mejora significativa respecto del modelo Ridge: el error absoluto medio es unos 350 vatios menor. El estadístico $R^2$ es del 71,31 %, que frente al 46,8 % supone una mejora de entorno al 25 %, es decir, otra vez muy significativa.

## 4.4. 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 $\gamma$ como podemos ver en la ecuación 2.40. Para este parámetro, para el parámetro de regularización C, y para el parámetro $\varepsilon$, 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 [1]. Los intervalos que se proponen en dicho artículo son los que vemos en la tabla **??**, donde $\sigma$ 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 independientes, es decir, 72.

| C | $\varepsilon$ | $\gamma$ |
|---|---|---|
| $\{10^k : -1 \leq k \leq 6\}$ | $\{\sigma 2^{-k} : 1 \leq k \leq 6\}$ | $\{4^k d^{-1} : -2 \leq k \leq 3\}$ |

Tabla 4.3: Validación cruzada propuesta en [1].

| **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.

$\sigma$ 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.

Para este modelo obtenemos un error absoluto medio del 6,36 %, y el estadístico $R^2$ es el 72,7 %. Podemos observar que la mejora respecto del perceptrón multicapa es muy pequeña, y no la consideraremos significativa, es decir, que asumiremos un empate entre ambos modelos.

## 4.5. 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.

Con el modelo stacking se obtiene un error absoluto medio del 6,52 %, y un estadístico $R^2$ del 70,97 %. En la tabla 4.4 podemos ver los resultados de todos los modelos.

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.

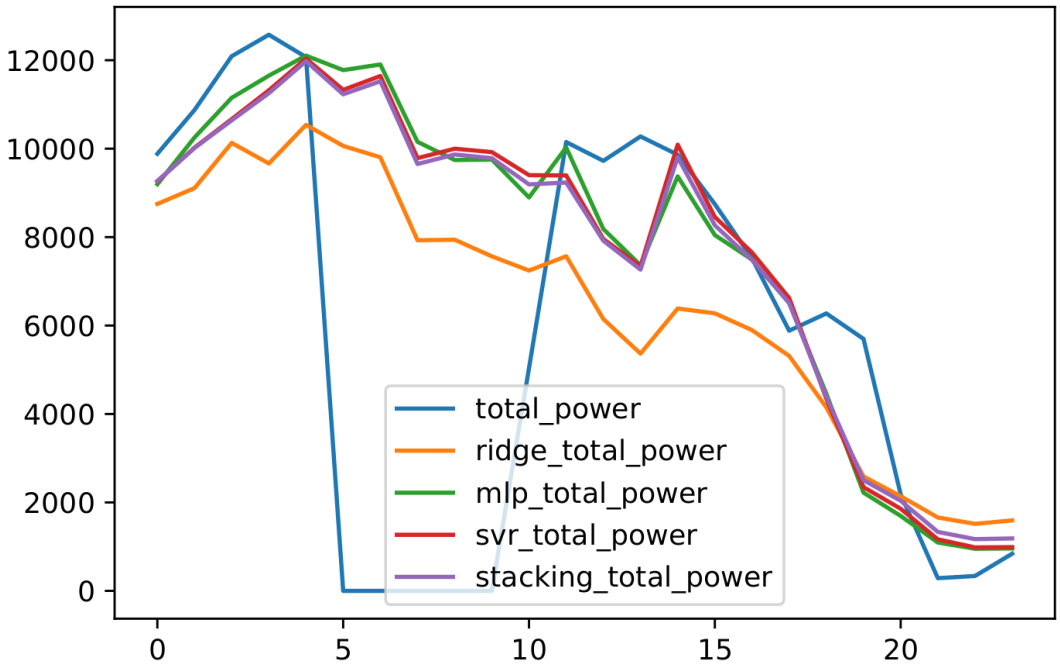



Figura 4.4: Gráfico de la obtención de energía el 4 de abril de 2018.

Otra observación importante es 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.

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 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.

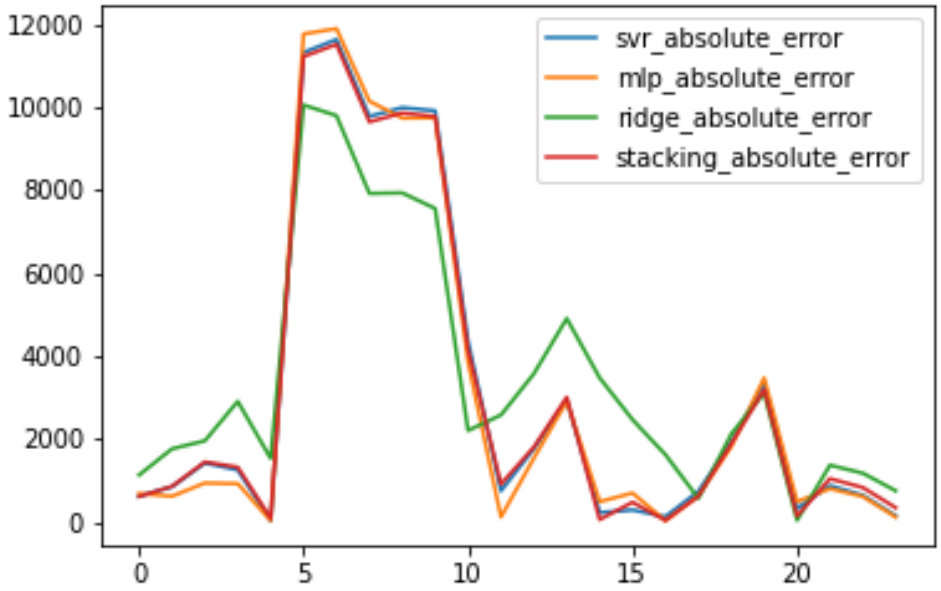



Figura 4.5: Gráfico de la obtención de energía el 4 de abril de 2018.

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, como hemos dicho, ninguno de los modelos es capaz de anticipar dicho periodo, con lo que esta opción no es posible.

# Bibliografía

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