Universidad del País Vasco
Euskal Herriko Unibertsitatea

ZIENTZIA ETA TEKNOLOGIA FAKULTATEA
FACULTAD DE CIENCIA Y TECNOLOGÍA

Trabajo Fin de Grado
Grado en Ingeniería Electrónica

Estudio, desarrollo y evaluación de técnicas de aprendizaje automático y su aplicación a la predicción de consumo energético

Autor:
David de Miguel Tercero
Director:
Luis Javier Rodriguez Fuentes

Leioa, junio de 2021

Índice

Nota: A excepción de las figuras en las que se indique su procedencia, el resto han sido creadas para este trabajo.

1. Introducción

1.1. Contextualización

Este trabajo de fin de grado (TFG) se plantea con la intención de responder a las preguntas:

¿Qué es el aprendizaje automático (*Machine Learning* en inglés) y qué técnicas se engloban bajo este método? ¿Qué es el aprendizaje profundo (*Deep Learning*) sobre redes neuronales y qué diferencias tiene con otros métodos de Machine Learning?

El *Machine Learning* (ML) es una rama de la inteligencia artificial (IA), que es un subcampo de las ciencias de la computación. La IA surgió en la década de 1940 [1] queriendo dotar a las máquinas de capacidades que poseen los humanos —capacidades inteligentes—.

Con el fin de saber si este objetivo se alcanza, hay que responder a la pregunta de si la máquina tiene realmente un comportamiento inteligente o si simplemente lo está emulando. Pero ello, en primer lugar hay que establecer qué es la inteligencia, tarea nada trivial.

Sin llegar a definir un concepto tan abstracto, podemos partir de algo que se considere como una condición necesaria para la inteligencia: la capacidad de aprender.

Esto es precisamente en lo que se basa el ML, en el desarrollo de algoritmos para que las máquinas puedan aprender de manera autónoma sin tener que programarlas explícitamente. La ventaja de este enfoque se puede apreciar en el caso de enfrentar un problema en el que hay muchas variables involucradas (como que la máquina juegue una partida de ajedrez o mantenga una conversación con una persona), donde la solución no es única. En un caso así, tendría que considerarse un número de posibilidades casi infinito. Desde el punto de vista del ML, la manera de plantear el problema es diferente: se diseña un algoritmo que minimiza una medida de error y que adquiere experiencia según realiza la tarea especificada.

Además de ser un área de estudio en sí misma, el ML es también una poderosa herramienta. Sin embargo, para poder hacer uso de ella, han tenido que darse dos condiciones: la existencia de un poder computacional suficiente y un grado mayor de sofisticación a la hora de plantear el problema del aprendizaje.

Tras una serie de éxitos iniciales, se observó que no se cumplían los requisitos necesarios para que la disciplina pudiera seguir desarrollándose, iniciando así un periodo de desilusión que resultó en la falta de fondos para seguir investigando. A esta etapa se la denominó *AI winter* (o invierno de la IA), y duró desde 1970 hasta principios de los 2000. Fue en 2005 cuando el llamado aprendizaje profundo (*Deep Learning*, DL) empezó a dar los resultados que se habían estado persiguiendo desde hacía décadas [2].

El DL es un campo de estudio dentro del ML que trata de diseñar arquitecturas conocidas como redes neuronales (*Neural Networks*, NN), estructuras inspiradas en las co-

nexiones que se encuentran en el cerebro humano. Gracias al DL, el campo de la IA ha vuelto a cobrar protagonismo, asentándose como una disciplina a tener en cuenta para el desarrollo tecnológico en muchos campos.

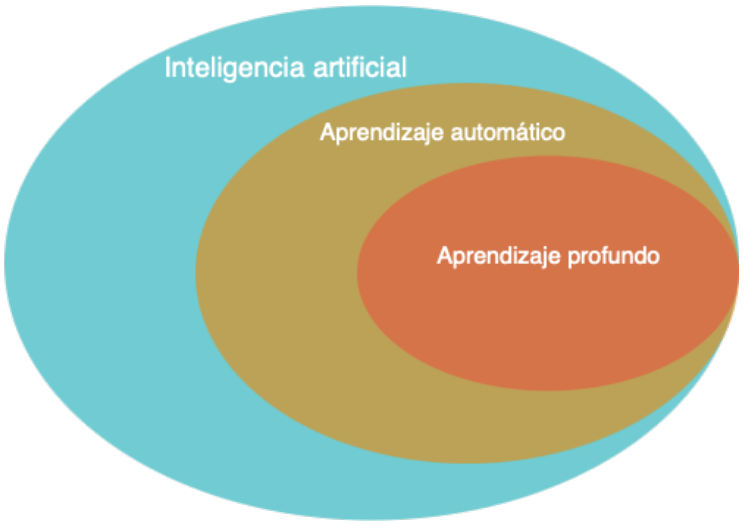

Figura 1: Relación entre la inteligencia artificial, el aprendizaje automático y el aprendizaje profundo.

Tras situar ambos conceptos, ML y DL, dentro del panorama tecnológico actual, en los siguientes apartados se describe la estructura de este trabajo, junto con los objetivos a los que se pretende llegar.

1.2. Motivación, objetivos, herramientas y alcance del TFG

En la industria eléctrica, la generación y el consumo suceden casi simultáneamente —dado que la electricidad no puede almacenarse para el futuro—. Para suministrar electricidad ininterrumpidamente de la manera más eficiente posible, las empresas dentro del sector utilizan técnicas de predicción para anticiparse a la demanda. De esta manera, se minimizan las pérdidas y se evitan periodos de desabastecimiento [3].

Así pues, este TFG tiene como objetivo entender varios modelos de ML y aplicarlos a la predicción del consumo energético.

Para ello, en primer lugar hay que conocer las herramientas que se emplean en esta disciplina: lenguajes de programación, librerías etc.

Se utilizará el entorno de trabajo de Jupyter, y se trabajará en Python para el estudio y generación de datos, y la estimación y evaluación de modelos predictivos.

Será imprescindible estudiar las librerías —las orientadas a la ciencia de datos— disponibles en este lenguaje. Principalmente, se utilizarán las siguientes:

- Pandas: Una herramienta de código abierto para el análisis y la manipulación de datos.

- NumPy: Una biblioteca enfocada a la computación numérica.
- Matplotlib: Una biblioteca orientada a la visualización de datos.
- Statsmodels: Un módulo para llevar a cabo el análisis estadístico de datos.
- scikit-learn: Una biblioteca de ML.
- Keras: Una biblioteca de DL.
- Tensorflow: Una biblioteca que permite trabajar con redes neuronales profundas.

Los cálculos se realizarán utilizando un servidor de cómputo gestionado por el grupo de investigación GTTS del Departamento de Electricidad y Electrónica de la UPV/EHU. A continuación se muestran sus especificaciones:

- Procesador: 2 Xeon® 5550, 2.4GHz, 20M caché, 4 cores 16 threads.
- RAM: 32GB

Los datos del problema seleccionado se han tomado de *Kaggle*, una comunidad web de científicos de datos que comparte código, conjuntos de datos y organiza competiciones de ML.

Por último, se desea recalcar que más que la obtención de unos resultados excelentes, lo que se pretende es enfrentarse al proceso de desarrollo de un sistema de ML y comprobar cómo las metodologías de análisis, modelado y evaluación habituales dentro de este campo realmente sirven para mejorar los modelos de los que se haya partido inicialmente.

1.3. Estructura del trabajo

En primer lugar, se describirá el problema que se va a estudiar y la manera de enfocarlo desde el punto de vista del ML. Después, se mostrará cómo emplear varios modelos para afrontarlo. Cada proceso que se lleve a cabo estará precedido por la explicación teórica correspondiente. La presentación del primer modelo contará con una explicación más exhaustiva del procedimiento que se lleve a cabo. En el resto de los modelos, se omitirá esta parte, ya que se desarrollarán de la misma manera.

En segundo lugar, se tratará el problema desde el punto de vista del DL.

A continuación, se compararán los resultados obtenidos, y se analizarán la eficacia y la eficiencia de cada modelo.

Por último, se reflexionará sobre el trabajo realizado y se plantearán varias posibles maneras de mejorarlo.

2. Desarrollo

2.1. Descripción del problema

El problema escogido para este trabajo es la predicción del consumo energético en una región. Para ello, se ha usado un conjunto de datos (*dataset*) seleccionado de *Kaggle* [4].

En un inicio, el país escogido fue Alemania, ya que la cantidad de datos de consumo que ofrecía era mayor que la del resto de países —en torno a 6 años de instancias tomadas cada 15 minutos, llegando a un tamaño del orden de $2 \cdot 10^5$ muestras—.

Sin embargo, para predecir el consumo, resulta útil disponer de otra información como, por ejemplo, las condiciones meteorológicas o los métodos de generación de electricidad.

Por ello, el país escogido ha sido España, ya que en el dataset disponible se encuentran datos meteorológicos, energéticos y de consumo tomados entre el 31 de diciembre de 2014 a las 23:00 y el 31 de diciembre de 2018 a las 22:00 en intervalos de una hora —un total de 35064 muestras—.

Dentro del amplio campo del aprendizaje automático, la tarea que queremos llevar a cabo se describiría como "**Predicción de series temporales empleando un sistema de aprendizaje supervisado no incremental**".

Se estará trabajando con aprendizaje supervisado ya que los modelos se estimarán a partir de los datos que pretenden predecir. El hecho de que el sistema sea no incremental implica que el entrenamiento se hará con todos los datos disponibles, y no con datos que van llegando de manera secuencial (lo que se conoce como *online learning*).

2.2. EDA: Análisis exploratorio de datos

En esta etapa del proceso de diseño del sistema de ML, se analizan los datos disponibles con el objetivo de convertirlos en información útil. Esto en inglés se conoce como "*Exploratory Data Analysis*" (EDA), es decir, análisis exploratorio de datos.

Idealmente, no habría que utilizar el conjunto de test bajo ningún concepto. Sin embargo, antes de realizar la división de datos en los conjuntos de entrenamiento y test, puede haber situaciones en las que sea inevitable analizar el dataset en su totalidad. Se tratarán los siguientes casos: **tratamiento de datos incompletos o duplicados** y **supresión de datos repetitivos**. Tras lidiar con estas situaciones, **se analizará únicamente el conjunto de datos de entrenamiento**.

En primer lugar, se comienza importando el dataset de *Kaggle*. Vemos que está compuesto por dos archivos CSV. El primero contiene datos relacionados con la energía generada y consumida en España y se le llamará dataset energético. El segundo contiene datos relacionados con la meteorología y se le llamará dataset meteorológico.

Para cada uno de los conjuntos de datos (energético y meteorológico) se analizan los

formatos de cada dato disponible. En el apéndice B (Tabla 9) se muestran las columnas que componen cada uno de los datasets.

Interpretación, formato y unidades del dataset energético

Entre las 29 columnas que componen el dataset energético, las que tienen la palabra "*generation*" en su nombre hacen referencia a diferentes medios de generación de energía (solar, nuclear, eólica etc.). Las que tienen la palabra "*forecast*" o "*ahead*" en su nombre se refieren a las predicciones realizadas por el Gestor de Red de Transporte (TSO) de España. Por último, encontramos tres columnas más que no siguen este esquema: *time* (la fecha en la que se ha tomado el dato, en formato ISO: año-mes-día-zona horaria), *total load actual* (la energía consumida en España cada hora) y *price actual* (el coste de la energía cada hora, en unidades de Euros/MW·hora).

A excepción de la columna *time*, que contiene datos en forma de objeto tipo *String*, el resto de columnas contienen números en formato de coma flotante.

Toda la información de este conjunto de datos representa variables continuas. Todas las variables relacionadas con la generación o el consumo de energía tienen unidades de MW.

Interpretación, formato y unidades del dataset meteorológico

Este dataset está conformado por 17 columnas. En él encontramos una columna llamada *dt iso* que contiene fechas en el mismo formato que la columna *time* del dataset energético.

El resto de columnas, que contienen información de variables continuas, tienen formato de número de coma flotante. Algunos ejemplos de estas variables son la presión, la temperatura y la humedad.

Entre las variables continuas, las que están relacionadas con la temperatura tienen unidades de grados Kelvin; las que están relacionadas con la presión, de hPa; las relacionadas con algún tipo de velocidad, de m/s; las relacionadas con cualquier tipo de precipitación, en mm; la humedad se expresa en porcentajes y la dirección del viento en grados sexagesimales. También hay una variable que representa qué porcentaje del cielo está cubierto de nubes.

La diferencia principal entre este conjunto de datos y el energético es la presencia de variables categóricas. Estas variables —discretas—, contienen información en forma de número o de texto. Encontramos las siguientes:

- *city name*: Como su propio nombre indica, contiene el nombre de la ciudad en la que se han tomado los datos de las variables. Encontramos 5 cadenas de caracteres: Bilbao, Madrid, Barcelona, Sevilla y Valencia.
- *weather main*: Contiene una breve descripción del estado meteorológico en cada instante. En total hay 12 valores diferentes, como por ejemplo: "clear", "rain" y

"thunderstorm".

- *weather description*: Contiene una descripción más detallada del estado meteorológico que la variable *weather main*. Se encuentran 41 valores diferentes en total, como: "few clouds", "light rain" y "proximity thunderstorm".
- *weather icon*: Contiene los códigos que utiliza el TSO para etiquetar el estado meteorológico actual. En total hay 24 valores diferentes, entre ellos: "01n", "10d" y "04". No se proporciona más información sobre estos valores, por lo que su significado se desconoce.
- *weather id*: Contiene etiquetas en forma de números enteros, en un rango entre 200 y 804. Hay 38 valores diferentes en total. Al igual que con la variable *weather icon*, se desconoce el significado de estas etiquetas.

2.2.1. Tratamiento de datos incompletos o duplicados

En este apartado se identifican los datos nulos en cada columna para todo el dataset, y se decide qué hacer con ellos. Después, se comprueba que no existan datos duplicados. En caso de existir, se eliminan.

Dataset energético

En primer lugar, se descartan las columnas relacionadas con las predicciones hechas por el TSO de España, ya que no son de interés para el proyecto. Además, se renombra la variable que queremos predecir —*total load actual*— a "Consumo MW".

A continuación, empleando los datos de la fecha en la que se ha tomado cada instancia (obtenidos de la columna *time*) a modo de índice, se observa que no existen instantes repetidos. Asimismo, se observa que en total hay 35064 instancias repartidas entre el 31 de diciembre de 2014 a las 23:00 y el 31 de diciembre de 2018 a las 22:00, en intervalos de una hora.

Después se cuenta la cantidad total de datos de cada variable del dataset energético, y como resultado se observa que la columna *generation hydro pumped storage aggregated* está completamente vacía. Por lo tanto, esta variable es descartada.

Además, se observa que a excepción de la columna *price actual*, hay instantes en los que para el resto de variables no se ha tomado ningún dato. Entre ellas se encuentra la variable que se desea predecir —*Consumo MW*—, que tiene 36 instantes sin ningún valor asignado.

En la mayoría de los casos, los datos que faltan no son consecutivos o sólo lo son para pocos instantes. A continuación se muestra la mayor cantidad de datos consecutivos que faltan para el consumo:

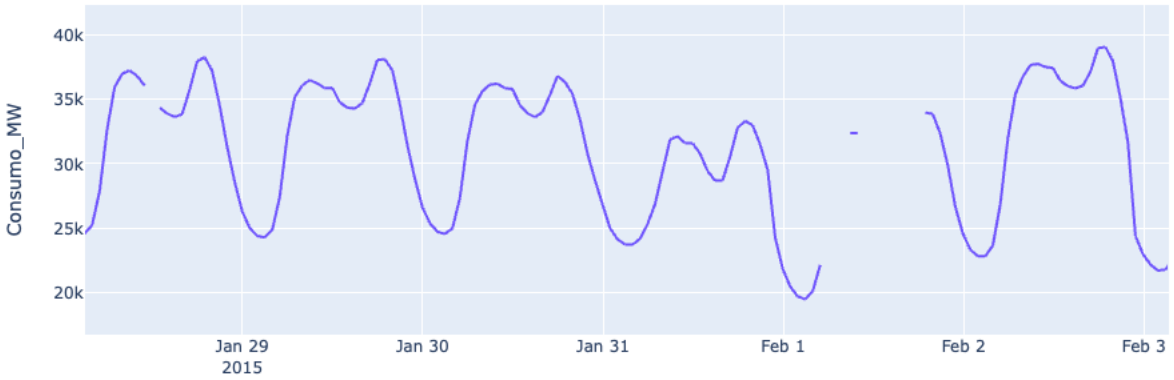

Figura 2: Ejemplo de instantes en los que faltan datos de consumo.

Al resto de columnas les faltan entre 17 y 19 valores. Para ver el rango de fechas en las que faltan valores, se examina la columna *generation biomass* (a la que le faltan 19 valores) y se observa que el resto de variables también tienen instancias vacías en las mismas fechas. A continuación se muestra la mayor cantidad de datos consecutivos que faltan para la variable *generation biomass*:

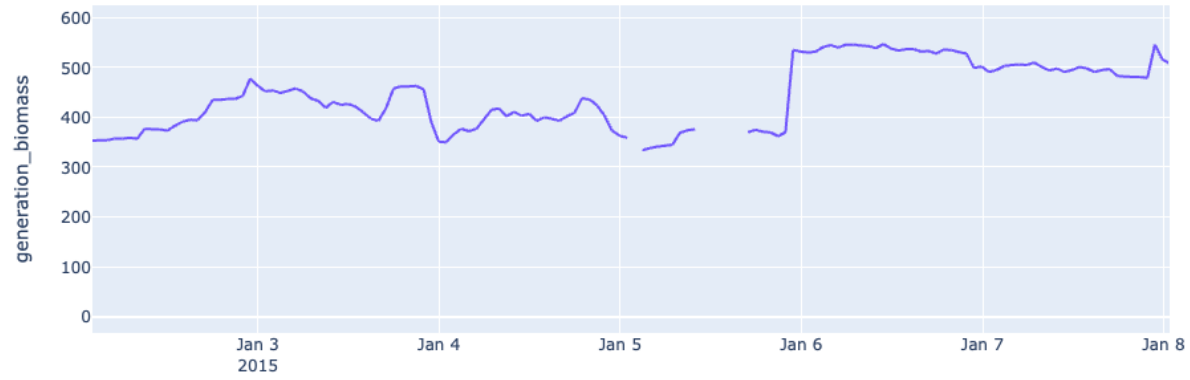

Figura 3: Ejemplo de instantes en los que faltan datos de la variable *generation biomass*.

Para solventar este problema, se ha decidido que los datos que faltan se completarán con el valor del instante anterior (lo que se conoce como *forward fill*). Esto puede ser problemático para casos en los que falten muchos datos consecutivos. Tras aplicar el método forward fill sobre todo el conjunto de datos, para las variables *Consumo MW* y *generation biomass*, se obtiene lo siguiente:

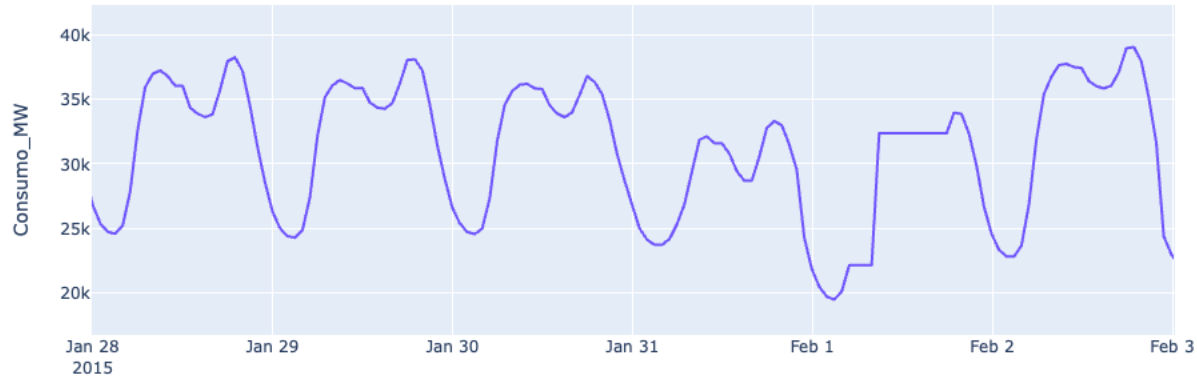

Figura 4: Ejemplo de aplicación del forward fill sobre instantes en los que faltaban datos de consumo.

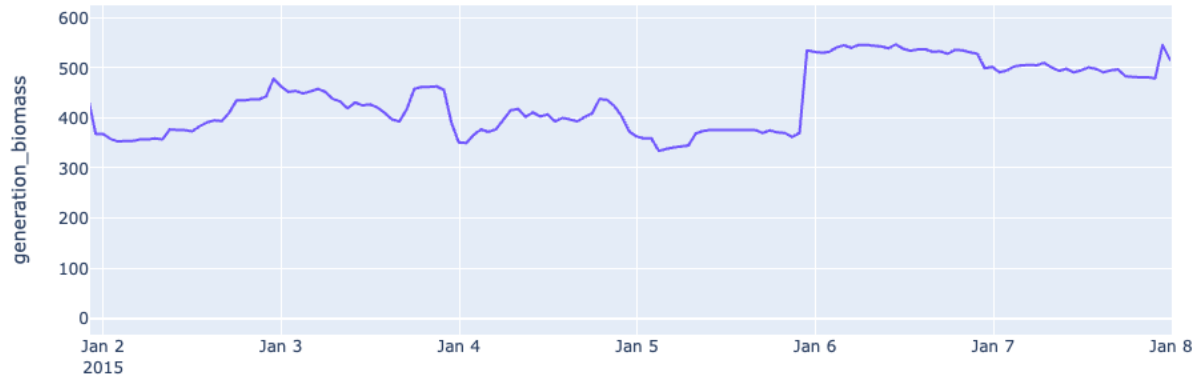

Figura 5: Ejemplo de aplicación del forward fill sobre instantes en los que faltaban datos de la variable *generation biomass*.

Como puede observarse al comparar la Figuras 2 y 4, rellenar los datos de consumo que faltaban con los valores anteriores resulta en una forma algo antinatural para el día 1 de febrero de 2015. Sin embargo, para el día 28 de enero la corrección es prácticamente imperceptible. Lo mismo ocurre tras rellenar los datos que faltaban de la columna *generation biomass*, que al ser pocos, dan lugar a una curva mucho más realista (como puede observarse comparando las Figuras 3 y 5).

Al rellenar los datos en las situaciones anteriores, se han obtenido resultados razonables para los casos en los que faltaban más instancias consecutivas. Por ello, la aplicación del *forward fill* se ha considerado adecuada en este caso.

Dataset meteorológico

Del mismo modo que para el dataset energético, se ha utilizado la columna *dt iso* —que contiene los datos de las fechas para este dataset— a modo de índice para estudiar el conjunto de datos meteorológicos.

Tras estudiar todos los datos, se aprecia que no falta ningún valor.

A diferencia del caso anterior, se encuentran 178396 instancias, comprendidas en el mismo rango de fechas que el dataset energético. Esto se debe a que los datos de este data-

set están tomados para cinco ciudades de España, por lo que se dispone de información acerca de lo que sucede en 5 lugares del país simultáneamente.

Es por eso que se han reagrupado las columnas del dataset en función de la ciudad. Para diferenciar qué variables pertenecen a cada ciudad, se han añadido los sufijos: "Bi", "Ma", "Ba", "Se" y "Va" (refiriéndose a Bilbao, Madrid, Barcelona, Sevilla y Valencia respectivamente).

De esta manera, la variable que antes se llamaba "*pressure*" se divide en 5 variables diferentes: "*pressure Bi*", "*pressure Ma*", "*pressure Ba*", "*pressure Se*" y "*pressure Va*". Así, se puede obtener el valor de presión en las 5 ciudades al mismo tiempo. El mismo método de división se aplica al resto de variables y se obtiene un nuevo conjunto de datos de meteorología con 5 veces más variables. Además, se prescinde de la variable categórica *city name*, ya que su información está contenida en las nuevas variables.

Seguidamente, comienza la búsqueda de valores duplicados. Se observa que para Bilbao, hay 887; para Madrid, 1203; para Barcelona, 412; para Sevilla, 493 y para Valencia, 81.

Esto concuerda con las 35064 instancias que se esperaba obtener, ya que si sumamos todos los valores duplicados y los restamos a la cantidad de datos contada inicialmente, se llega exactamente a $5 \cdot 35064$ instancias (lo que se explica sabiendo que tenemos datos de 5 ciudades). Por ello, se eliminan los datos duplicados.

2.2.2. Supresión de datos repetitivos

Tener variables cuyo valor es prácticamente constante a lo largo del tiempo no aporta información sobre la relación que guarda con el resto de variables. Por ello, se eliminan las columnas que tienen un único valor.

En el dataset energético, las columnas que presentan esta condición son: *generation fossil coal derived gas*, *generation fossil oil shale*, *generation fossil peat*, *generation geothermal*, *generation marine* y *generation wind offshore*. Por ese motivo, son eliminadas del dataset.

En el dataset meteorológico se observa que las variables *snow 3h Ba* y *snow 3h Se* tienen un único valor en todo el dataset. Por ello, también se suprimen del conjunto de datos.

Por último, se unen los dos datasets (energético y meteorológico) para conformar un único conjunto de datos que puede representarse matricialmente de la siguiente manera:

$$\left[\begin{array}{ccccc|c} x_1^{(1)} & x_2^{(1)} & x_3^{(1)} & \cdots & x_n^{(1)} & y^{(1)} \\ x_1^{(2)} & x_2^{(2)} & x_3^{(2)} & \cdots & x_n^{(2)} & y^{(2)} \\ x_1^{(3)} & x_2^{(3)} & x_3^{(3)} & \cdots & x_n^{(3)} & y^{(3)} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ x_1^{(m)} & x_2^{(m)} & x_3^{(m)} & \cdots & x_n^{(m)} & y^{(m)} \end{array}\right]_{m \text{ x } (n+1)}$$

donde $m = 35064$ es el número de instancias, correspondientes a los instantes en los que se han tomado los datos; y es el valor que se desea predecir o *target variable* —el consumo energético— y $n = 88$, es el número de características, denotadas como x_i $(i = 1, 2, \ldots, n)$ —el resto de columnas del dataset—.

2.2.3. Tratamiento de variables categóricas y de texto

Tras finalizar el apartado anterior, se obtiene un conjunto de datos que contiene variables en diversos formatos: números de coma flotante, números enteros y cadenas de caracteres.

Como la mayoría de algoritmos de ML necesitan variables numéricas para poder aprender, en este apartado se muestra el proceso de transformación de variables en formato de texto a variables numéricas.

Una manera de hacer este cambio sería tomar todas las instancias diferentes de la variable categórica en orden, y asignar a cada una un valor numérico. Para aclarar este proceso se toman 7 instancias de la variable "*weather main Bi*":

El vector $\begin{bmatrix} \text{“clear”} \\ \text{“clouds”} \\ \text{“rain”} \\ \text{“mist”} \\ \text{“thunderstorm”} \\ \text{“drizzle”} \\ \text{“fog”} \end{bmatrix}$ se convertiría en: $\begin{bmatrix} 0. \\ 1. \\ 2. \\ 3. \\ 4. \\ 5. \\ 6. \end{bmatrix}$, y una secuencia de cinco instantes, como por ejemplo:

$\begin{bmatrix} \text{“mist”} \\ \text{“rain”} \\ \text{“rain”} \\ \text{“thunderstorm”} \\ \text{“clear”} \end{bmatrix}$ pasaría a ser: $\begin{bmatrix} 3. \\ 2. \\ 2. \\ 4. \\ 0. \end{bmatrix}$. El problema de realizar esta transformación es que los modelos de ML interpretarán que los valores próximos son más parecidos que los lejanos. Esto puede ser un inconveniente ya que se estaría aprendiendo que "rain"y "thunderstorm" se parecen más que "rain" y "drizzle" —aunque parece razonable pensar que la lluvia y la llovizna se asemejan más que la lluvia y una tormenta eléctrica, en la que puede no llover—.

Por este motivo, se ha decidido que las instancias de las variables categóricas de texto se transformarán en variables binarias independientes.

Por lo tanto, la secuencia de cinco instantes anterior se convierte en una secuencia de 5 instantes para 7 variables:

“clear”	“clouds”	“rain”	“mist”	“thunderstorm”	“drizzle”	“fog”
0.	0.	0.	1.	0.	0.	0.
0.	0.	1.	0.	0.	0.	0.
0.	0.	1.	0.	0.	0.	0.
0.	0.	0.	0.	1.	0.	0.
1.	0.	0.	0.	0.	0.	0.

Para realizar la transformación, se aplican las funciones de la clase OneHotEncoder de scikit-learn sobre todo el dataset (para las variables categóricas de las 5 ciudades), y se obtienen 308 variables binarias. Tras este proceso, se descartan las variables categóricas anteriores.

El número de características del dataset resultante es ahora $n = 381$.

Sin embargo, al crear estas nuevas características, sucede que algunas de ellas sólo tienen valores nulos. Por ese motivo son descartadas, resultando en un conjunto de datos con $n = 373$.

2.2.4. Partición y análisis del conjunto de datos

Como en todos los procesos de aprendizaje supervisado, comenzamos dividiendo nuestro dataset en dos subconjuntos a los que desde ahora llamaremos: training set (conjunto de entrenamiento) y test set (conjunto de evaluación).

El proceso de aprendizaje se llevará a cabo en el training set, y la evaluación del funcionamiento del modelo se hará con el test set. Es por esto que el test set no será utilizado hasta que se haya terminado de diseñar cada modelo. Típicamente, el 80 % de los datos van para entrenamiento y el 20 % para evaluación [5].

Con esto, se pretende preparar al modelo para enfrentarse a datos que no conoce.

Además, al dividir el dataset también se intenta evitar una situación conocida como **overfitting**. En un caso así, el modelo ha aprendido tanto de la información conocida que ha perdido la capacidad de generalizar a la hora de evaluar datos nuevos; por lo que predice resultados con errores muy grandes. En el caso contrario tenemos el **underfitting**, donde el modelo no es capaz de aprender de los datos disponibles. Esto sucede cuando se dispone de pocos datos o cuando se emplea un modelo muy sencillo para el problema seleccionado.

Para ilustrar estas dos situaciones, se han entrenado dos modelos diferentes sobre un conjunto de datos de prueba —generado con una función polinómica de tercer grado a la que se le ha añadido ruido aleatorio—.

En la Figura 6 se ve que el modelo 1 es demasiado sencillo como para adaptarse a los datos y el modelo 2, más complejo, se ha sobre-ajustado. Ninguno de los dos es un modelo de ML apropiado para el problema seleccionado.

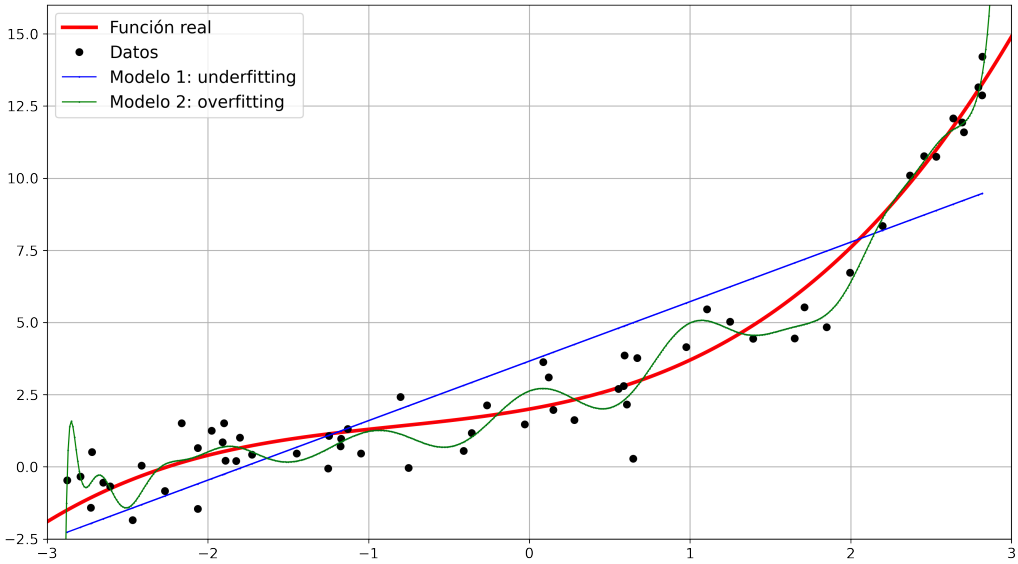

Figura 6: Ejemplo de dos modelos de ML en situaciones de overfitting y underfitting.

Estos son los dos extremos entre los que hay que moverse cuando se trabaja en ML. Para evitar ambas situaciones (overfitting y underfitting), una práctica muy común es reservar una parte del corpus de entrenamiento para probar y refinar los modelos antes de evaluarlos. A este subconjunto **dentro** del training set se le conoce como **validation set**, y contiene un 20 % de los datos totales [6].

Sin embargo, a la hora de trabajar con series temporales, la partición del dataset no puede hacerse tomando elementos al azar como en otros problemas de ML. Esto se debe a que los datos recopilados están tomados en instantes consecutivos, por lo que no se desea perder la causalidad. Por ese motivo, los datos que se reservarán para el test set serán los de los instantes más cercanos al momento actual; mientras que los datos más antiguos, se utilizarán para el entrenamiento.

Así, la partición final se ha realizado de la siguiente manera:

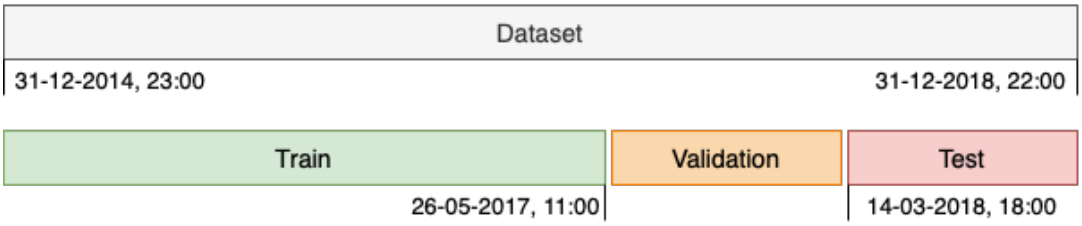

Figura 7: Partición del conjunto de datos.

- Training set: contiene aproximadamente un 60 % de los datos.
- Validation set: contiene aproximadamente el 20 % de los datos. El error que se cometa en él proporcionará información de cara a la obtención del modelo más apropiado para el problema.
- Test set: contiene aproximadamente el 20 % de los datos. El error que se cometa en él determinará la elección del modelo, ya que representa una buena aproximación del error que es esperable cometer en una situación real.

Una vez hecha la división, se procede a analizar los datos del training set.

Se comienza visualizando el consumo:

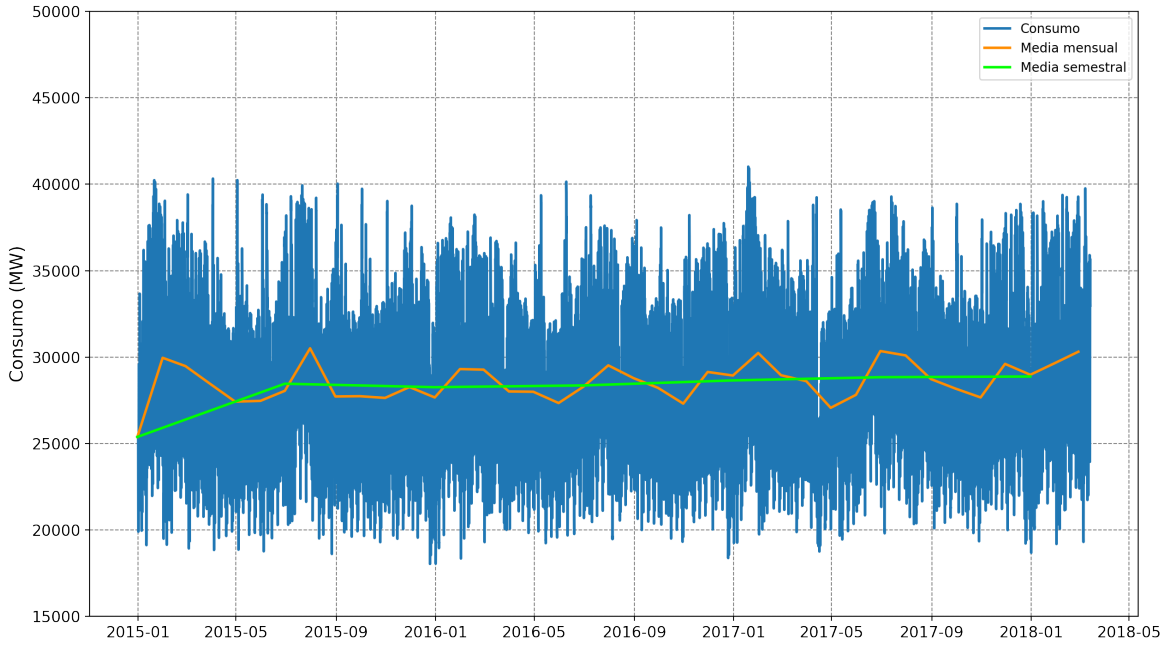

Figura 8: Consumo energético de España entre 2015 y principios de 2018.

A primera vista, parece que el consumo es una serie temporal cuya tendencia permanece constante. Sin embargo, al visualizar también la media de cada semestre se observa que el consumo energético aumenta lentamente según avanza el tiempo. En cuanto a las medias mensuales, se repara en que presentan una estructura similar, con máximos en invierno y en verano.

Por estos motivos, no se realizarán transformaciones que expresen el consumo mediante una serie estacionaria.

Para estudiar la relación lineal entre el instante actual y los instantes previos del consumo, en la Figura 9 se visualiza su función de autocorrelación:

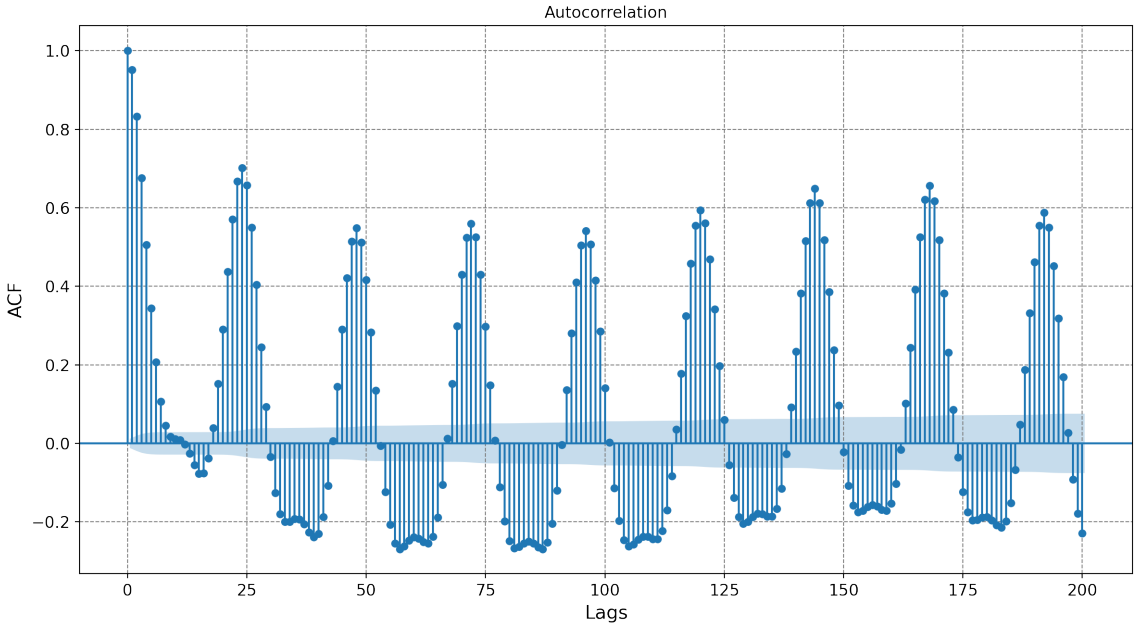

Figura 9: Función de autocorrelación del consumo para 199 instantes previos.

En esta figura se muestran las correlaciones lineales entre un instante de la serie y los que le preceden —conocidos como *lags*—. Concretamente, se repara en que los máximos de la función de autocorrelación se dan para instantes previos en múltiplos de 24, revelando ciclos diarios. Además, también se observa un máximo para un lag de 168, mostrándose ciclos semanales.

Así, se concluye que para predecir el consumo a corto plazo, los datos de consumo más relevantes son aquellos que pertenecen al siguiente rango de instancias previas a la de interés: $[1, 24] \cup [168, 192]$ (instantes de las 24 horas previas, y de las 24 horas previas a la semana anterior).

Una vez analizada la variable que se desea predecir, se pasa a analizar las características —el resto de variables— y la relación que guardan con el consumo energético. A la hora de estudiar y seleccionar la variables, se han tomado los valores absolutos de las correlaciones.

Para estudiar de manera rápida la correlación entre muchas variables es común hacer uso de los mapas de calor. En ellos, se relaciona el grado de correlación entre dos variables con un número dentro de una escala de colores.

Un ejemplo de los mapas de calor empleados en este análisis es el siguiente:

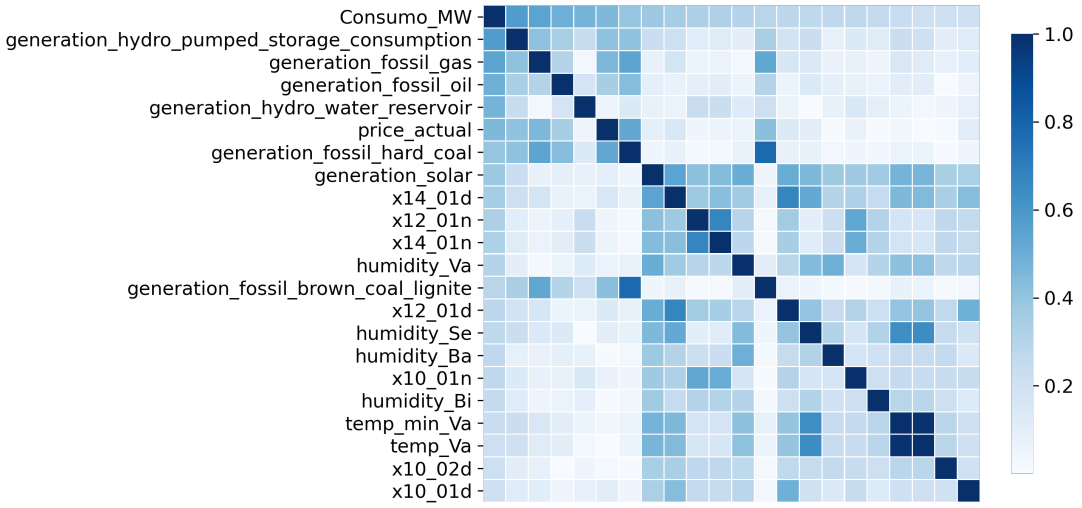

Figura 10: Ejemplo de mapa de calor empleado durante el EDA.

Gracias a él, puede verse de manera rápida que el consumo y la variable *generation hydro pumped storage consumption* tienen un grado de correlación más alto (alrededor de 0,6) que el consumo y la variable *temp Va* (alrededor de 0,2).

Antes de dar por finalizada esta etapa del proceso, hay que decir que el análisis exploratorio de datos es una disciplina en sí misma. Profundizar en este tema bien podría valer para un TFG entero. Sin embargo, este TFG tratará de abarcar y comprender todos y cada uno de los pasos que se siguen en un proyecto de ML de principio a fin.

2.3. Ingeniería de características

En esta fase se obtienen intuiciones para poder diseñar características específicas para los modelos de ML —proceso conocido como *Feature Engineering*—.

Para medir de forma directa cuán útiles son las variables para predecir el consumo energético, se calculará el valor del coeficiente de Pearson (r_{xy}) entre cada una de ellas y el consumo —este es el coeficiente que se utiliza para crear la función de autocorrelación y el mapa de calor del apartado anterior—.

Sobre un conjunto de muestras, el coeficiente de Pearson se define como:

$$r_{xy} = \frac{\sum_{i=1}^{m}(x^{(i)} - \bar{x})(y^{(i)} - \bar{y})}{\sqrt{\sum_{i=1}^{m}(x^{(i)} - \bar{x})^2 \sum_{i=1}^{m}(y^{(i)} - \bar{y})^2}} \ , \tag{1}$$

donde $\bar{x} = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$, y análogamente para $\bar{y}$.

r_{xy} es un valor dentro del rango $[-1, 1]$ que indica la fuerza de la correlación lineal entre dos variables aleatorias.

Los valores cercanos a 1 indicarán variables correlacionadas de manera lineal y positiva, mientras que los valores cercanos a -1 indicarán variables correlacionadas de manera

lineal y negativa. Cuando el coeficiente se trate de un valor cercano a 0, indicará que las variables no estarán correlacionadas de manera lineal, lo que no implica que no guarden otro tipo de relación. Esto puede verse en los conjuntos de datos de la tercera fila en la Figura 11, donde se aprecia que aunque existe una relación no lineal clara, el coeficiente de Pearson no es capaz de percibirla.

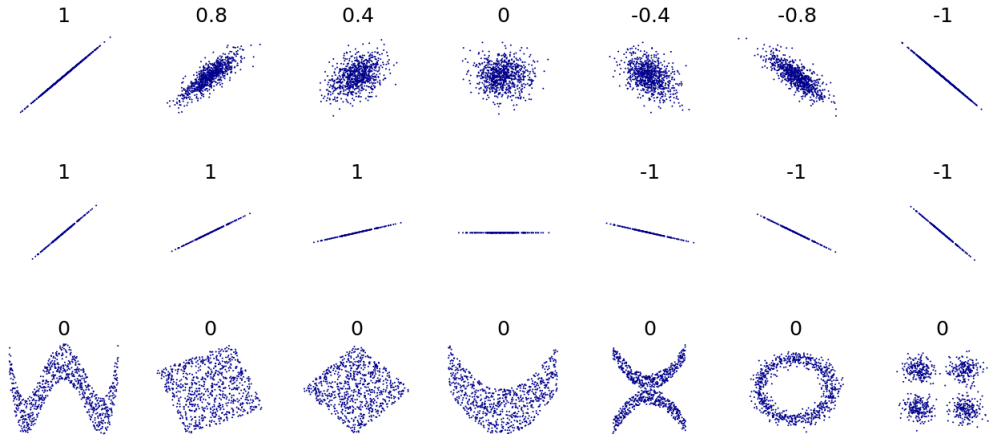

Figura 11: Valores del coeficiente de Pearson para varios conjuntos de datos bidimensionales. Fuente: `https://es.wikipedia.org/`

Una práctica habitual a la hora de diseñar variables nuevas es tomar las características disponibles y mapearlas con una función conocida (como un logaritmo o alguna función trigonométrica) para luego comprobar si la característica modificada y la variable objetivo están correlacionadas.

Otra manera es crear variables directamente, bien utilizando la opinión de un experto en la materia o bien de manera intuitiva. Así, se crean las siguientes nuevas características:

- *year*: una variable categórica en formato de número entero que contiene el año en el que se ha tomado el dato.
- *month*: una variable categórica en formato de número entero que contiene valores entre 1 y 12.
- *day*: una variable categórica en formato de número entero con valores entre 1 y 31.
- *hour*: una variable categórica en formato de número entero con valores entre 0 y 23.
- *weekday*: una variable categórica en formato de número entero con valores entre 1 y 7.

Además, también se prueba a crear variables nuevas combinando diferentes columnas:

- *total generation*: Esta nueva característica se ha creado sumando todas las variables relacionadas con los diferentes tipos de generación de energía. Por ello, contiene información de la energía total generada en el país en cada instante (en MW).
- *money spent*: Esta variable se obtiene tras multiplicar las columnas *price actual* y la recién creada *total generation*. Por ello, contiene información de todo el dinero empleado en generar energía en cada instante (en unidades de Euros/hora).

A continuación, se vuelve a estudiar la correlación entre el consumo y las características —incluyendo las creadas en este apartado— para el training set, y se obtiene lo siguiente:

Tabla 1: Porcentajes de variables y sus correlaciones con el consumo.

Porcentaje de variables (%)	Correlación con el consumo
$\sim 3{,}4$	$> 0{,}3$
$\sim 6{,}8$	$> 0{,}2$
$\sim 17{,}4$	$> 0{,}1$

Como el objetivo es predecir el consumo energético, se tomarán las variables que más relación guarden con él, y se comenzará descartando aquellas cuya correlación sea menor que 0,3. Esto es un punto de partida razonable, ya que además de simplificar el problema, un número menor de variables contribuirá a un funcionamiento más rápido de los modelos de ML.

En la Tabla 2 se muestran las características que cumplen con esta condición:

Tabla 2: Características con una correlación mayor que 0,3.

Características	Correlaciones con el consumo
total generation	0.5874
money spent	0.5394
hour	0.4024
generation hydro pumped storage consumption	0.4003
price actual	0.3825
generation solar	0.3808
generation fossil gas	0.3698
x14 01d	0.3560
x12 01n	0.3229
generation hydro water reservoir	0.3216
x14 01n	0.3196
humidity Va	0.3119
humidity Ba	0.3013

Como puede observarse, entre las variables más correlacionadas con el consumo en cada instante se encuentran tres de las creadas con *feature engineering* (marcadas en verde). El resto de características están compuestas por variables categóricas obtenidas en el apartado 2.2.3 (marcadas en azul) y variables continuas.

Sin embargo, lo que se pretende predecir no es el consumo energético actual, sino el futuro. Por ello, tras visualizar con un mapa de calor las correlaciones entre las variables

seleccionadas y los valores de consumo de las 24 horas siguientes (Apéndice A, Figura 29), se concluye que **los valores actuales de las variables seleccionadas junto con el valor actual de consumo son útiles para predecir los valores de consumo de instantes posteriores**.

Así que, como último paso antes de pasar al modelado, se modifican los conjuntos de datos de entrenamiento y de evaluación de modo que ambos contengan las mismas características y variables objetivo.

2.4. Predicción mediante aprendizaje automático

En este apartado se diseñarán varios modelos de ML con un horizonte de predicción $p = 1$, a excepción de las redes neuronales, que también podrán tener $p = 12$ (en el apartado 2.4.5 se verá cómo pueden aumentarse los horizontes de modelos con $p = 1$).

Esto significa que los primeros modelos intentarán predecir el consumo con un instante —una hora— de antelación, mientras que los segundos tratarán de predecir la secuencia de 12 instantes —12h— posteriores.

Antes que nada, conviene familiarizarse con la notación que se va a utilizar:

- m : Número de muestras/instancias del conjunto de datos.
- n : Número de características del modelo.
- $\mathbf{X}$: *Feature matrix* (matriz de características). Su dimensión es $n \cdot m$. Se trata de la entrada al modelo.
- $\mathbf{x}$: *Feature vector* (vector de características). Es una fila de la matriz de características.
- $\mathbf{y}$: *Target vector* (vector objetivo). Es el dato o conjunto de datos que el modelo tiene que predecir.
- $\hat{\mathbf{y}}$: Valor o valores predichos por el modelo.
- $\boldsymbol{\theta}$: Vector de parámetros. Contiene los pesos (θ_i) de cada característica junto con el término de sesgo (θ_0).
- h : La función hipótesis.

En este proyecto, $m = 35064$ y $n = 14$.

Los modelos diseñados —a excepción de las redes neuronales— utilizarán los datos de las 14 características en un instante tanto para predecir el próximo valor de consumo, como la secuencia de las 12 horas siguientes. Las redes neuronales utilizarán los datos de las últimas 24 horas para hacer las mismas predicciones.

Antes de comenzar la fase de diseño, se explicará la forma en la que se evaluarán los resultados.

2.4.1. Medidas de error y modelos de referencia

Medidas de error

Para saber si los modelos que se diseñen están aprendiendo, hay que definir una medida del error de predicción. Una elección típica a la hora de evaluar el rendimiento de modelos de regresión es el *MSE* (error cuadrático medio), que se define de la siguiente manera:

$$MSE = \frac{1}{m}\sum_{i=1}^{m}(\hat{y}^{(i)} - y^{(i)})^2 \quad . \tag{2}$$

Otra medida común es el coeficiente de determinación (R^2). Lo usaremos como medida complementaria. Su cálculo se realiza como sigue:

$$R^2 = 1 - \frac{\sum_{i=1}^{m}(y^{(i)} - \hat{y}^{(i)})^2}{\sum_{i=1}^{m}(y^{(i)} - \bar{y})^2} \ , \quad \text{con} \quad \bar{y} = \frac{1}{m}\sum_{i=1}^{m} y^{(i)} \ . \tag{3}$$

Cuanto más cercano a 1 sea R^2, más se parecerán las predicciones a los datos reales.

Modelos de referencia

Resulta útil disponer de algunas medidas de referencia con las que poder comparar los resultados que se obtengan. De esta manera, podremos saber si se está progresando o no.

Un procedimiento común es intentar vencer a un modelo que predice el último valor observado en la serie ($\hat{y}_{t+1} = y_t$). A esto se le conoce como *naïve forecasting*, y puede ser difícil de mejorar. Por lo tanto, este será el modelo de referencia que se utilizará para $p = 1$. Para $p = 12$, el modelo de referencia devolverá la secuencia de valores correspondiente al día anterior.

2.4.2. Regresión Lineal

El modelo de regresión lineal realiza las predicciones de la siguiente manera:

$$\hat{y} = h_\theta(\mathbf{x}) = \ \mathbf{x} \cdot \ \boldsymbol{\theta} \ = \begin{bmatrix} 1 & x_1 & x_2 & \cdots & x_n \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix} = \theta_0 + x_1\theta_1 + x_2\theta_2 + ... + x_n\theta_n \ , \tag{4}$$

donde $x_0 = 1$ por convenio.

Como se puede observar, el resultado dado por la función de hipótesis de este modelo es el producto escalar del vector de parámetros y el vector de características. Los parámetros θ pueden interpretarse como "pesos" que regulan la importancia de cada característica. Será el propio modelo el que mediante un proceso de entrenamiento deberá aprender estos parámetros.

Finalmente, para hacer las predicciones sobre un conjunto de m datos, el modelo de regresión lineal hace la siguiente operación:

$$\begin{bmatrix} \hat{y}^{(1)} \\ \hat{y}^{(2)} \\ \vdots \\ \hat{y}^{(m)} \end{bmatrix} = \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \cdots & x_n^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & \cdots & x_n^{(2)} \\ 1 & x_1^{(3)} & x_2^{(3)} & \cdots & x_n^{(3)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(m)} & x_2^{(m)} & \cdots & x_n^{(m)} \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix} . \tag{5}$$

Entrenamiento del modelo

Como se ha dicho en la introducción, los modelos tendrán que minimizar una función de error o coste (J). Este proceso de estimación (o re-estimación) de parámetros con un objetivo de optimización es a lo que se llama *entrenamiento*.

Cuando el problema de minimización no tiene solución analítica, se utilizan métodos iterativos. Por ello, **conviene que todos los datos que se proporcionen al modelo estén en el mismo rango de valores** ya que, así, se consigue minimizar la función de coste respecto a todos los parámetros θ_i de igual manera. Por ello, se escalan todos los valores a un rango de $[0, 1]$ utilizando la clase MinMaxScaler de scikit-learn.

Entre los métodos iterativos de minimización más conocidos, se encuentra el **descenso por gradiente** [7]:

$$\boldsymbol{\theta}^{(k+1)} := \boldsymbol{\theta}^{(k)} - \eta \boldsymbol{\nabla}_\theta \, J(\boldsymbol{\theta}) \, . \tag{6}$$

En él, se inicializan los parámetros θ de manera aleatoria. Después, se calcula el gradiente de la función de coste respecto a cada parámetro para encontrar la pendiente más pronunciada. Finalmente, se sustrae el valor del gradiente multiplicándolo por un hiperparámetro η conocido como *learning rate* que se usa para controlar la velocidad con la que se desciende por la función de coste.

Para favorecer la comprensión del método, se muestra una imagen ilustrando cómo evoluciona la solución al aplicar el descenso por gradiente:

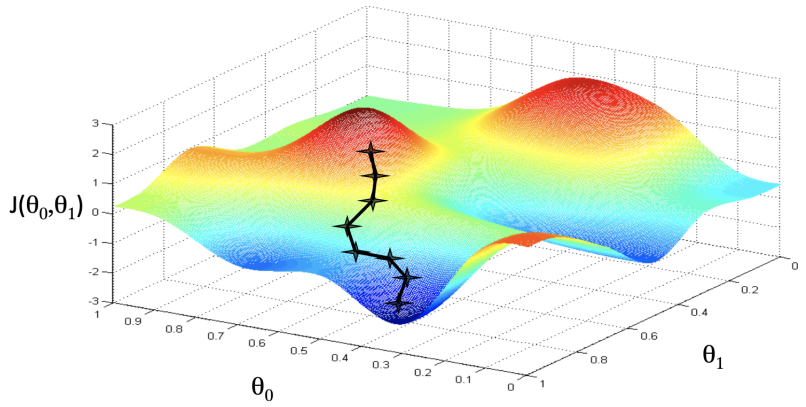

Figura 12: Iteraciones del descenso por gradiente para un vector de parámetros de dos dimensiones. Fuente: `https://medium.com`

Para entrenar la regresión lineal se toma como función de coste el *MSE*, y para minimizarlo utilizando la expresión (6) se calcula su gradiente:

$$\boldsymbol{\nabla}_\theta \, J(\boldsymbol{\theta}) = \boldsymbol{\nabla}_\theta \Big(\frac{1}{m} (\mathbf{X}\boldsymbol{\theta} - \mathbf{y})^2 \Big) = \frac{2}{m} \mathbf{X}^T (\boldsymbol{\theta}\mathbf{X} - \mathbf{y})$$

Ahora, ya puede aplicarse la expresión (6) para entrenar el modelo.

Sin embargo, en el caso de la regresión lineal existe solución analítica:

$$\hat{\theta} = \left(\mathbf{X}^T \mathbf{X}\right)^{-1} \mathbf{X}^T \, , \tag{7}$$

donde $\hat{\theta}$ son los parámetros que minimizan la función de coste.

Resultados

Para analizar los resultados que obtengan los modelos —que no sean redes neuronales—, se estudiará el error cometido en los conjuntos de entrenamiento y de validación —lo que se conoce como curvas de aprendizaje o, en inglés, *learning curves*— en función del tamaño de muestras del conjunto de entrenamiento. Esto resulta útil para comprobar si los datos de los que se dispone son suficientes para el modelo escogido.

En primer lugar, se prueba el modelo de referencia y se obtiene que los errores cometidos en los conjuntos de entrenamiento y validación son respectivamente $3{,}9706 \cdot 10^{-3}$ y $3{,}7730 \cdot 10^{-3}$.

A continuación, se entrena el modelo únicamente en el training set, se evalúa su rendimiento en el validation set y se muestran sus curvas de aprendizaje:

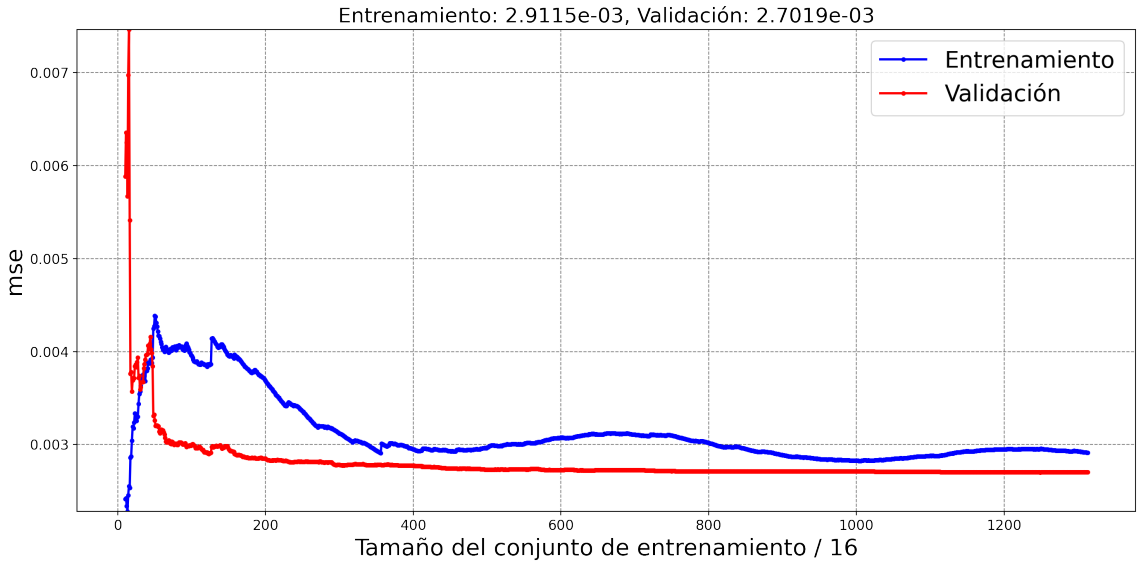

Figura 13: Curvas de aprendizaje para el modelo de regresión lineal.

Como puede observarse, el error es menor que el que comete el modelo de referencia para ambos conjuntos de datos. Observando la Figura 13, se repara en que el modelo no se encuentra en una situación de overfitting, ya que el error en ambos conjuntos de datos es muy similar —en caso de haber overfitting, el error en el conjunto de entrenamiento sería mucho menor que en el conjunto de validación—.

2.4.3. Árboles de decisión

Los árboles de decisión son unos modelos de ML muy versátiles, capaces de llevar a cabo tareas de clasificación y de regresión, con la posibilidad de predecir también secuencias de valores. Además, pueden ajustarse a conjuntos de datos muy complejos, por lo que corren el peligro de sufrir overfitting.

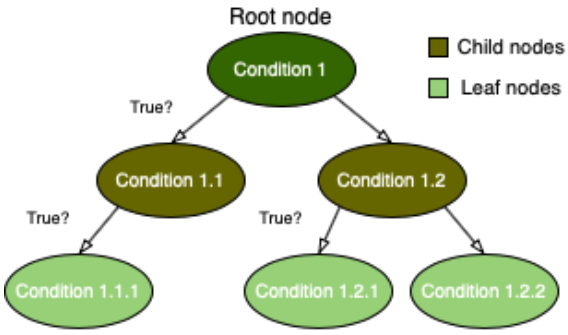

Figura 14: Árbol de decisión binario y sus componentes principales.

Como su nombre indica, el algoritmo —implementado mediante scikit-learn— devuelve una estructura de árbol binario. Cada nodo —que solo puede dividirse en dos—, contiene una condición de acceso junto con una predicción. Si esta condición se cumple, se accede al subnodo de la izquierda; si no, al de la derecha.

Para obtener la predicción, hay que recorrer los nodos del árbol y para cada uno, comprobar si se verifica la condición de acceso. De ser así, la predicción del modelo es la que contiene el último nodo accedido.

Este modelo ofrece la posibilidad de modificar la estructura del árbol por medio de hiperparámetros como la profundidad máxima del árbol —número de generaciones tras

el nodo de partida—, el cambio mínimo requerido para dividir el nodo, y el número de características y muestras que considerar antes de realizar la división.

Entrenamiento del modelo

El algoritmo empleado por scikit-learn se llama CART (Classification and Regression Tree) y consiste en dividir el espacio de características de manera recursiva, agrupando los vectores objetivo con valores similares. Para realizar una división, se recorren todas las parejas (θ, t_θ) —característica y valor umbral/threshold de la característica— y se escoge aquella que minimice la siguiente función de coste:

$$J_i(\theta, t_\theta) = \frac{m_{left}}{m_i} MSE_{left} + \frac{m_{right}}{m_i} MSE_{right} \ , \tag{8}$$

donde J_i, m_i, $m_{left/right}$ y $MSE_{left/right}$ son la función de error, el número de parejas que se evalúan —compuestas por vectores objetivo ($\mathbf{y}$) y vectores de características ($\mathbf{x}$)—, el número de datos que contiene el nodo hijo de la izquierda/derecha y el error cometido en el conjunto de datos del nodo hijo de la izquierda/derecha, respectivamente, para un cierto nodo i.

Cuanto más cercano a 0 sea el valor de J_i, más puro se dice que es el nodo. El entrenamiento se detendrá cuando el árbol alcance la profundidad deseada por el usuario o cuando no se pueda realizar una división más pura que la anterior.

La predicción que realiza el modelo se almacena en cada nodo, y es la siguiente:

$$\hat{y}_i = \frac{1}{m_i} \sum_{j=1}^{m_i} y^{(j)} \ , \tag{9}$$

que no es más que el valor medio de las muestras que se consideran en el nodo i.

Resultados

En primer lugar, se ha entrenado un árbol de decisión con los valores por defecto que recomienda scikit-learn. Tras analizar los resultados, se ha entrenado un nuevo modelo modificando el hiperparámetro que controla la profundidad del árbol (*max depth*). El primer árbol habría crecido sin restricciones hasta asentarse en una profundidad de 31. La profundidad del segundo se ha limitado a 8.

A continuación se muestran las curvas de aprendizaje del árbol modificado y se comentan los resultados obtenidos por ambos modelos:

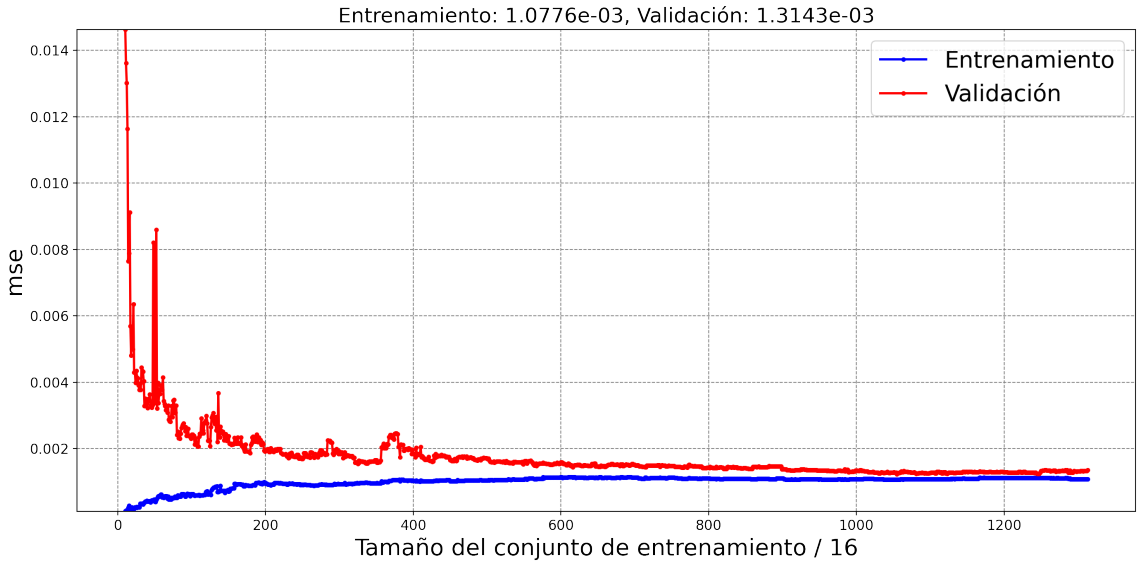

Figura 15: Curvas de aprendizaje para el árbol de decisión modificado.

Los errores cometidos en los conjuntos de entrenamiento y validación por el árbol no modificado son respectivamente $1{,}0622 \cdot 10^{-7}$ y $1{,}8994 \cdot 10^{-3}$. Estos resultados indican claramente un modelo sobreajustado, ya que el error cometido en el conjunto de entrenamiento no sólo es muy pequeño, sino que también está muy lejos del error en el conjunto de validación.

Por otra parte, la Figura 15 muestra que el árbol modificado, aunque se ajusta menos al conjunto de datos de entrenamiento, tiene un error de validación más pequeño. Además, ambas curvas (Entrenamiento y Validación) se encuentran muy próximas, signo de un ajuste más apropiado.

Con este nuevo árbol se obtiene un resultado más prometedor que con el modelo de regresión lineal.

En vez de seguir probando nuevos modelos, se estudiarán dos maneras de potenciar los árboles de decisión para obtener un modelo aún más robusto.

2.4.4. Aprendizaje conjunto

La idea detrás del aprendizaje conjunto es emular la llamada sabiduría de la multitud (o "wisdom of the crowd" [8]), ya que en muchas ocasiones la respuesta de la mayoría es más acertada que la de un único experto. De esta manera, es posible combinar las predicciones realizadas por muchos modelos de ML para construir un modelo mejor.

Aunque a la hora de acoplar modelos las combinaciones son infinitas, en este apartado se estudiarán dos tipos de aprendizaje conjunto muy populares: Random Forests y XGBoost (Extreme Gradient Boosting). En ellos, se unen varios árboles de decisión de diferentes maneras, lo que da lugar a modelos más potentes y robustos.

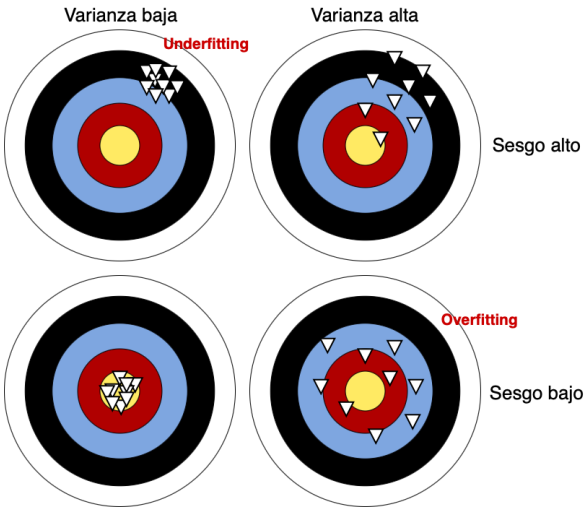

Figura 16: Intercambio entre sesgo y varianza.

Esto está relacionado con el concepto de "intercambio entre sesgo y varianza" ("Bias Variance Tradeoff" [9]), y a su vez con los conceptos de overfitting y underfitting comentados en el apartado 2.2.4. Para entenderlo, a continuación consideramos un ejemplo ilustrativo.

Supongamos un modelo que está intentando predecir una variable $Y = f(x) + \epsilon$, donde ϵ es el término de error, que sigue una distribución normal con media igual a cero. El error cuadrático esperado —que comete el modelo— en un punto x puede expresarse de la siguiente manera:

$$\text{Error(x)} = (\text{Sesgo})^2 + \text{Varianza} + \text{Error irreductible} \ , \tag{10}$$

donde el error irreductible no puede eliminarse aunque el modelo sea bueno.

Un modelo con la capacidad de sobreajustarse al conjunto de datos de entrenamiento —como un árbol de decisión— tendrá un sesgo bajo y una varianza alta a la hora de realizar predicciones, sufriendo de overfitting. Por otra parte, un modelo sencillo que no pueda ajustarse a los datos tendrá generalmente un sesgo alto y una varianza baja, sufriendo de underfitting. La Figura 16 ilustra lo que sucede en cada uno de los casos.

Por ello, si se estima un modelo complejo buscando reducir el sesgo, la varianza aumentará, y viceversa: simplificando el modelo, la varianza disminuirá, mientras que el sesgo aumentará.

El objetivo del aprendizaje conjunto es combinar varios modelos individuales (generalmente sencillos) para obtener un modelo final con la proporción sesgo-varianza adecuada.

2.4.4.1 Random Forests

Creación y entrenamiento del modelo

Los bosques aleatorios (*Random Forests*, en inglés) son métodos creados a partir de la unión de árboles de decisión, entrenados por un método llamado "Bagging" (Bootstrap aggregating).

En este proceso, se elaboran tantos subconjuntos a partir del conjunto de datos como árboles se desee que tenga el modelo combinado. Cada subconjunto se crea tomando muestras del conjunto original de manera aleatoria. Por este motivo, diferentes subconjuntos pueden tener muestras en común (este proceso de tomar muestras y reemplazarlas después se conoce como "Bootstraping").

A continuación, se entrena un árbol de decisión para cada subconjunto, con la posibilidad de considerar a su vez un subgrupo aleatorio de las características antes de realizar la división óptima. Esta aleatoriedad, junto con la del Bootstraping, da lugar a diversos árboles de decisión. Esto resulta en un modelo que intercambia un sesgo mayor por una varianza menor.

Finalmente, el resultado predicho por el modelo es el promedio de las predicciones realizadas por los árboles que lo componen.

Resultados

En la Figura 17 se muestran las curvas de aprendizaje de un modelo compuesto por 10 árboles de decisión con una profundidad máxima de 10:

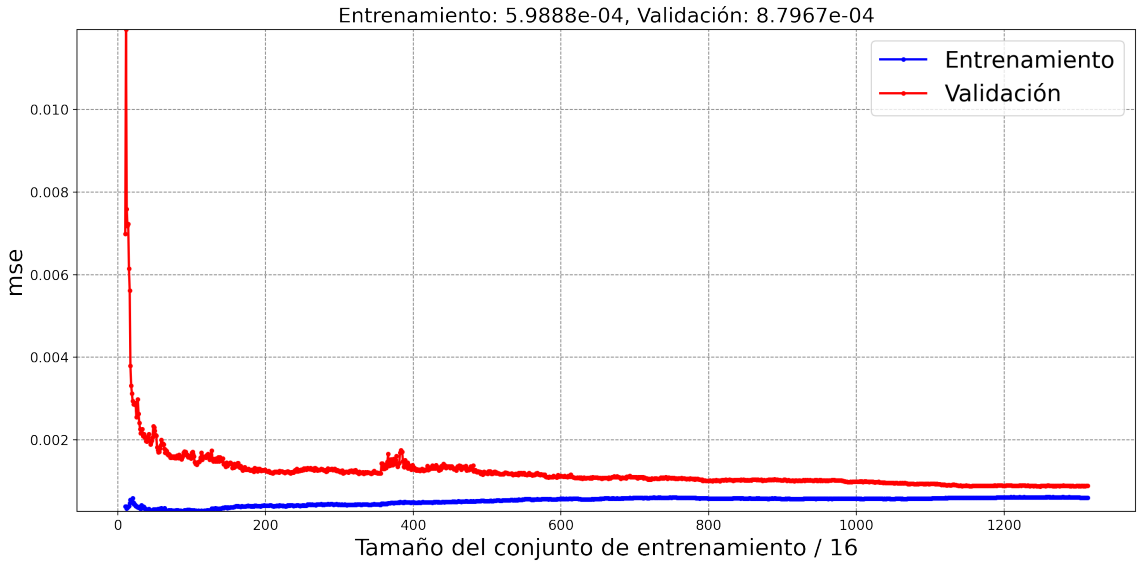

Figura 17: Curvas de aprendizaje de un modelo de tipo random forest.

Comparando los errores cometidos con los del árbol de decisión (Figura 15), el modelo combinado no solo mejora el resultado en ambos conjuntos, sino que además se obtienen curvas más pegadas al estabilizarse, indicando un ajuste mejor.

2.4.4.2 XGBoost

Creación y entrenamiento del modelo

El llamado boosting (que podría traducirse como "impulsión") consiste en combinar varios modelos "débiles" para conformar un modelo mejor. La idea básica es, por tanto, muy similar a la de los random forests.

El entrenamiento se realiza de la siguiente manera:

1. Se añade un árbol al modelo combinado (inicialmente vacío) y se estiman sus parámetros a partir del conjunto de entrenamiento. En este punto del proceso, la predicción del modelo es la del único árbol que lo compone.

2. Se añade un nuevo árbol al conjunto y se entrena sobre la diferencia entre los datos de entrenamiento y las predicciones realizadas —los residuos— por el modelo anterior. En este punto, la predicción del modelo es la combinación de las predicciones de los árboles ya presentes anteriormente en el modelo con la del recién añadido. A continuación, se calcula el error que comete el modelo.

3. A partir de ahora, se repite el paso 2 hasta que el modelo combinado tenga el número de árboles deseado.

Como puede observarse, la diferencia principal entre el random forest y el xgboost se encuentra en los datos que se emplean durante el entrenamiento. El random forest utiliza subgrupos aleatorios dentro del conjunto de datos de entrenamiento, mientras que el xgboost se entrena con los errores cometidos por la versión anterior del modelo.

Una implementación optimizada de este método puede encontrarse en la librería de Python XGBoost [10].

Resultados

Tras entrenar un modelo compuesto por 10 árboles de decisión con una profundidad máxima de 8, se han obtenido las siguientes curvas de aprendizaje:

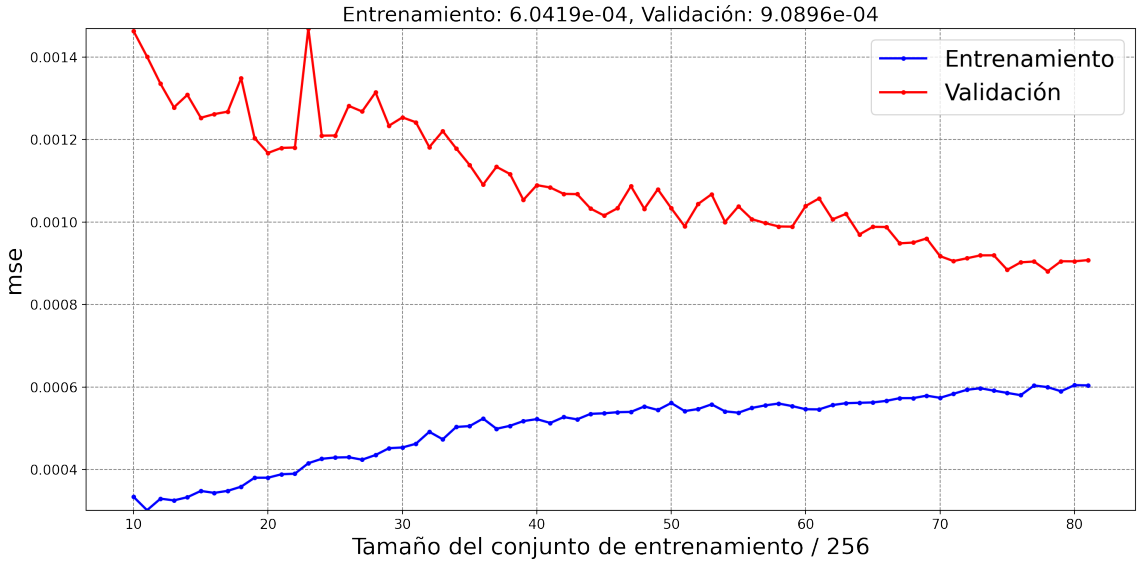

Figura 18: Curvas de aprendizaje de un modelo XGBoost.

Como puede observarse en la Figura 18, el rendimiento es similar al del modelo basado en random forests. Sin embargo, como el tiempo de entrenamiento requerido es mucho mayor, se han mostrado las curvas de aprendizaje en función del tamaño del conjunto de entrenamiento en fracciones de 256 instancias.

2.4.5. Técnicas para extender el horizonte de predicción

Los procedimientos más comunes para extender el horizonte de predicción son:

- Propagar el resultado predicho realimentándolo al modelo hasta llegar al horizonte de predicción p deseado.
- Entrenar p modelos independientemente para que cada uno prediga un instante consecutivo.
- Disponer de un modelo que sea capaz de predecir secuencias, como por ejemplo: árboles de decisión, bosques aleatorios o redes neuronales.

Algunos modelos, debido a su arquitectura, no son capaces de predecir secuencias. Un ejemplo claro de esto es la regresión lineal, ya que, como puede observarse en el apartado 2.4.2, las predicciones realizadas por el modelo son vectores $\hat{\mathbf{y}}$ de dimensión 1. Sin embargo, un modelo tipo árbol de decisión no tiene restricciones en cuanto a la dimensión de los valores que trata de predecir.

Tras realizar una búsqueda exhaustiva del modelo más apropiado para predecir los 12 instantes siguientes de consumo, se ha tomado la decisión de usar un modelo que pueda predecir secuencias. Concretamente, se ha escogido un random forest (compuesto por 50 árboles con una profundidad máxima de 10), con los siguientes resultados:

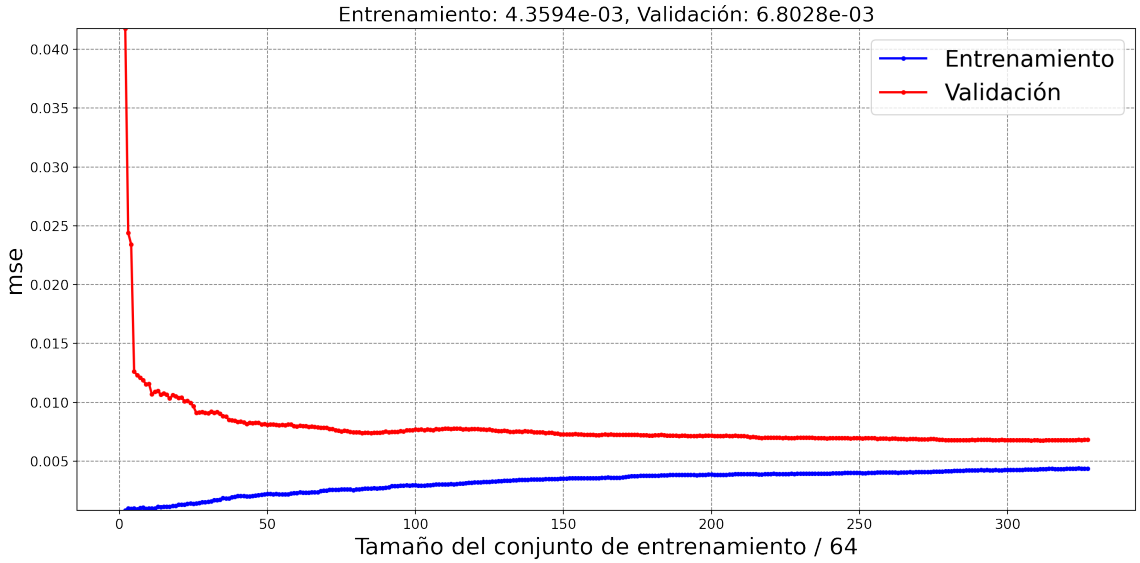

Figura 19: Curvas de aprendizaje para un random forest para la predicción de las 12 horas siguientes.

Hay que tener en cuenta que este modelo utiliza únicamente la información disponible en el instante actual para predecir las 12 horas siguientes.

2.4.6. Redes neuronales

Las redes neuronales son modelos de ML compuestos por varias capas de procesado que aprenden representaciones de datos con múltiples niveles de abstracción [11]. Originalmente se distinguía entre "shallow" y "deep neural networks" (DNN); el término "deep" hacía referencia a un modelo que contuviera muchas capas, aunque hoy en día hablar de redes neuronales —tengan las capas que tengan— es casi sinónimo de modelo de aprendizaje profundo.

A continuación, se muestra un ejemplo de arquitectura de una red neuronal:

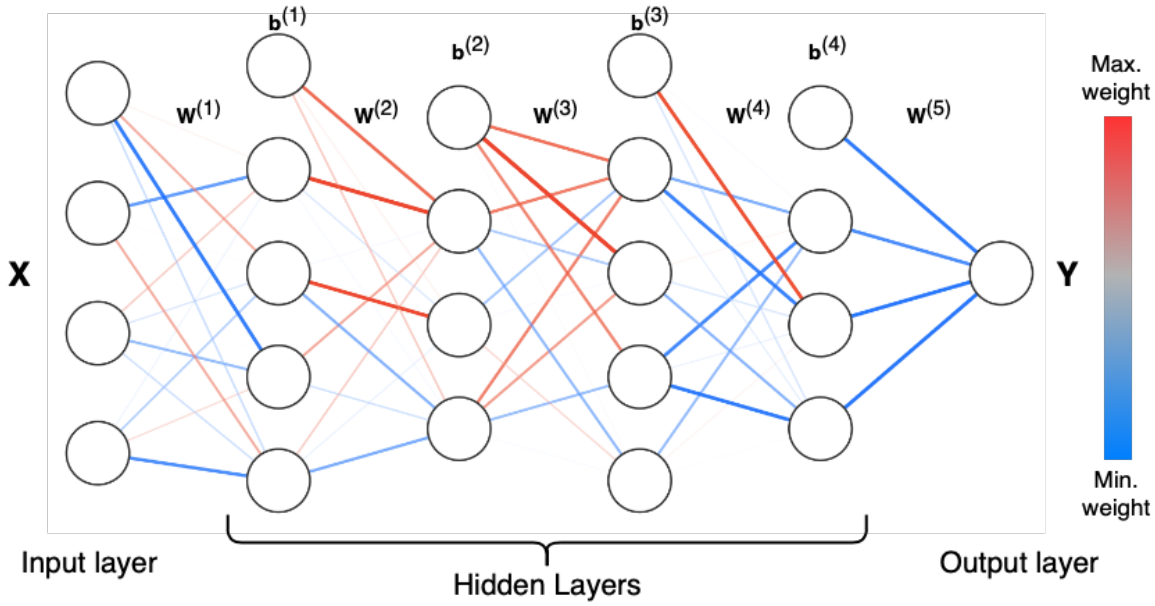

Figura 20: Ejemplo de una red neuronal profunda totalmente conectada.

En la Figura 20 se muestra una red neuronal con una capa de entrada de 4 neuronas, una capa de salida de una neurona y cuatro capas ocultas de 5, 4, 5 y 4 neuronas respectivamente. El modelo recibe unos datos de entrada contenidos en la matriz de características $\mathbf{X}$, y devuelve la predicción en la matriz $\mathbf{Y}$. Conectando las capas se encuentran las matrices de peso $\mathbf{W}$, de dimensiones n x m, siendo n el número de neuronas de la capa siguiente y m el número de neuronas de la capa anterior. Además, las capas ocultas cuentan con lo que se conoce como "neuronas de sesgo" ($\mathbf{b}$, de dimensión n x 1). Estas neuronas llevan a cabo la misma función que el término de sesgo (θ_0) del modelo de regresión lineal estudiado en el apartado 2.4.2.

En una red, la neurona j de la capa l devuelve el valor resultante (a_j^l) de la expresión:

$$a^l(z_j^l) = a^l\left(\sum_{k=1}^{m} w_{jk}^l a_k^{l-1} + b_j^l\right) \ , \tag{11}$$

donde m el el número de neuronas de la capa anterior, a^l es el valor de la función de activación de la capa l (que se utiliza para dar un carácter no lineal al valor de salida de la neurona), y z_j^l es la suma del término de sesgo (b_j^l) y la suma ponderada del producto resultante del valor de salida de la capa anterior (a_k^{l-1}) por los pesos (w_{jk}^l) correspondientes a la matriz $\mathbf{W^l}$ (las matrices de pesos llevan la etiqueta l de la capa siguiente). El resultado de la capa es $\mathbf{a}^l$, de dimensión n x 1.

El proceso de aprendizaje de estos modelos consiste en actualizar los valores w que contienen todas las matrices de pesos, junto con los términos de sesgo de cada capa (en caso de haberlos) para ajustarse al resultado deseado.

En sus inicios, los modelos de red neuronal profunda y sus variantes no tenían una forma eficiente de realizar el entrenamiento, dada la gran cantidad de parámetros que contienen. Fue en los años 80 del siglo XX cuando se introdujo el algoritmo conocido como **"Retropropagación"** (o "backpropagation"). En él, se calcula la dependencia entre el error cometido respecto de la función de error J y cada parámetro (de sesgo y de peso) de la red, realizando un barrido desde la última capa hasta la primera, empleando la regla de la cadena sobre el error cometido al aplicar la expresión (11):

$$\frac{\partial J}{\partial w_{jk}^l} = \frac{\partial J}{\partial a^l}\frac{\partial a^l}{\partial z_j^l}\frac{\partial z_j^l}{\partial w_{jk}^l} = \frac{\partial J}{\partial a^l}\frac{\partial a^l}{\partial z_j^l}\ a_k^{l-1} = \delta_j^l\ a_k^{l-1} \tag{12}$$

$$\frac{\partial J}{\partial b_j^l} = \frac{\partial J}{\partial a^l}\frac{\partial a^l}{\partial z_j^l}\frac{\partial z_j^l}{\partial b_j^l} = \delta_j^l\ . \tag{13}$$

A δ_j^l se le llama gradiente local, y depende de la función de coste empleada y de la función de activación de cada capa.

A continuación, se calcula lo mismo para todos los pesos de la capa $l-1$ y se llega a:

$$\frac{\partial J}{\partial w^{l-1}} = \delta^l\ \mathbf{W}^l\ \frac{\partial a^{l-1}}{\partial z^{l-1}}\ a^{l-2} = \delta^{l-1}\ a^{l-2} \tag{14}$$

$$\frac{\partial J}{\partial b^{l-1}} = \delta^l\ \mathbf{W}^l\ \frac{\partial a^{l-1}}{\partial z^{l-1}} = \delta^{l-1}\ . \tag{15}$$

Nótese que el término $\frac{\partial a^{l-1}}{\partial z^{l-1}}$ sólo depende de la función de activación de la capa $l-1$.

Por lo tanto, la retropropagación se resume de la siguiente manera:

1. Calcular el error de la última capa $\delta^l = \frac{\partial J}{\partial a^l}\frac{\partial a^l}{\partial z_j^l}$, considerando su función de activación.
2. Propagar el error a la capa anterior, calculando $\delta^{l-1} = \delta^l\ \mathbf{W}^l\ \frac{\partial a^{l-1}}{\partial z^{l-1}}$.
3. Calcular los errores de los parámetros de la capa $l-1$.

4. Aplicar los pasos 2 y 3 retrocediendo hasta llegar a la primera capa.

Este algoritmo permite obtener un vector gradiente de la función de coste respecto de los parámetros. De este modo, empleando el descenso por gradiente (6) se actualizan todos los parámetros, entrenando así la red neuronal.

Resultados

La obtención de curvas de aprendizaje en función del tamaño del conjunto de entrenamiento requiere mucho más tiempo para redes neuronales que para los modelos empleados previamente. Por ello, en estos apartados se obtendrán las curvas de aprendizaje en función de las veces que el algoritmo se ajuste al corpus de entrenamiento (lo que se conoce como "epochs"). Estas curvas también sirven para analizar cómo de adecuado es el aprendizaje de un modelo.

A continuación se muestran las curvas de aprendizaje de una red neuronal densa (véase su arquitectura en el apéndice A, Figura 30).

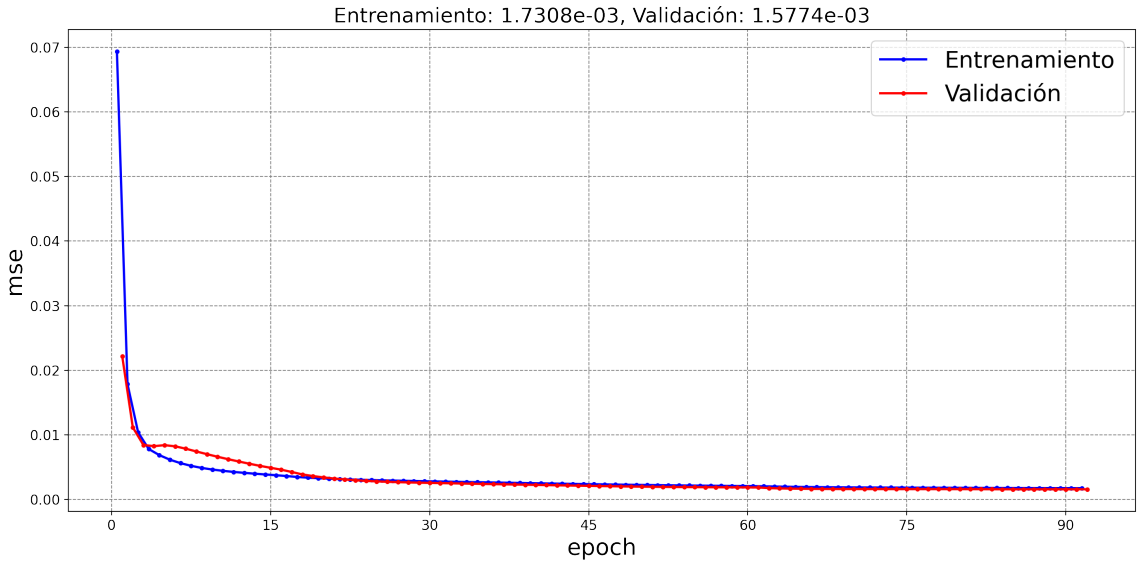

Figura 21: Curvas de aprendizaje de una red neuronal densa (Feed-Forward Neural Network, FFNN).

La Figura 21 muestra las curvas de aprendizaje del modelo en función del número de epochs. El aumento de epochs implica un aumento en la experiencia adquirida por el modelo. Es decir, el modelo aprende, reduciendo los errores que comete en ambos conjuntos. No se observa overfitting, ya que las curvas permanecen unidas.

En caso de haber overfitting, se esperarían unas curvas en las que el error en el conjunto de entrenamiento seguiría disminuyendo con las epochs, mientras que, llegado a un punto, el error en el conjunto de validación comenzaría a aumentar.

2.4.6.1 Redes neuronales recurrentes

Este tipo de redes se especializa en aprender secuencias que se extienden en el tiempo. Por ello, aplicarlas a una tarea de predicción de una variable física como el consumo energético parece una decisión lógica.

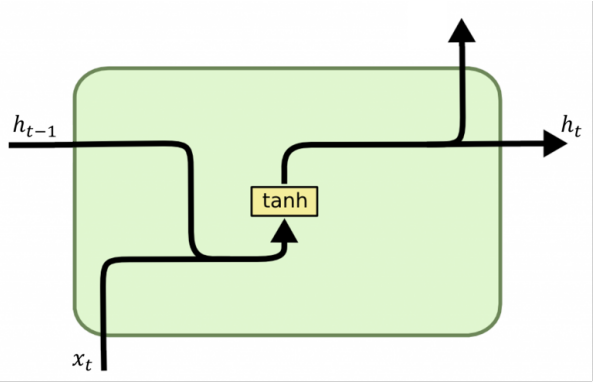

Figura 22: Esquema de una celda de red neuronal recurrente.
Fuente: `http://dprogrammer.org`

La diferencia entre las redes neuronales recurrentes (RNN) y las redes neuronales del apartado anterior se encuentra en la "neurona". En este tipo de redes, la unidad mínima de procesado se llama celda, y puede componerse de varias neuronas. A su vez, pueden encadenarse varias capas de celdas. En la Figura 22 se muestra la arquitectura de una celda. El valor de salida de la celda ($y_t = h_t$) depende del estado de ella misma en un instante anterior (h_{t-1}) —por lo que se trata de una celda realimentada— y del valor de entrada al modelo (x_t) en cada instante. h_t —llamado estado oculto—, podría interpretarse como la memoria de la celda en el instante t. De esta manera, se guarda información de diferentes momentos de la secuencia que se desea predecir.

El entrenamiento de este modelo se lleva a cabo mediante un proceso conocido como "retropropagación a través del tiempo" (BPTT), que consiste en extender la neurona en el tiempo y aplicar la retropropagación habitual (Figura 23).

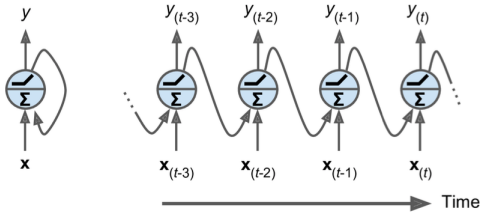

Figura 23: Esquema de la BPTT para una celda.
Fuente: Página 498, referencia [7].

Los resultados del modelo obtenido (Apéndice A, Figura 31) son los siguientes:

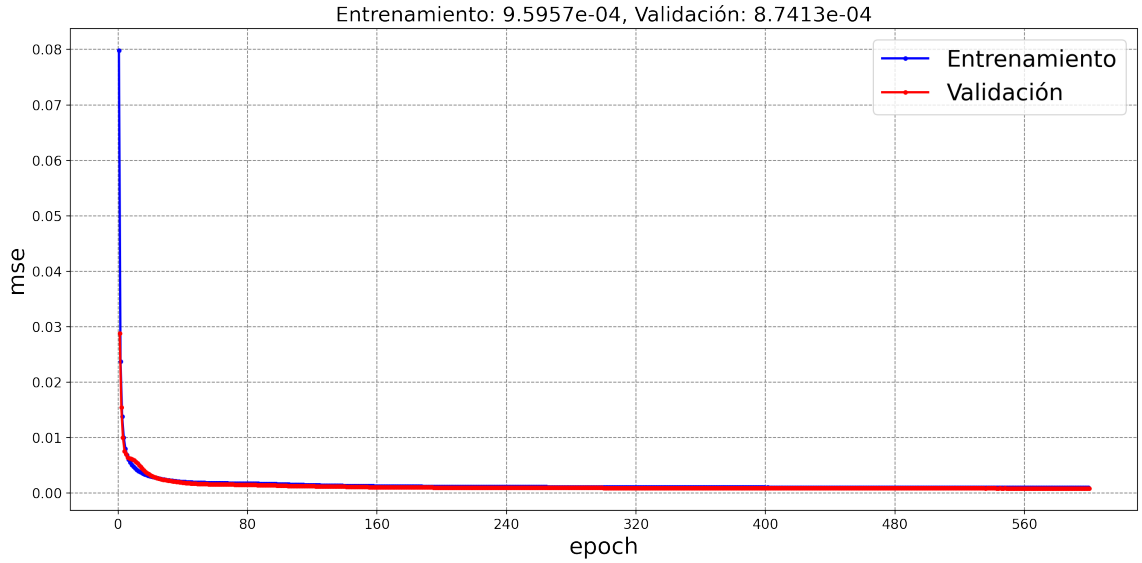

Figura 24: Curvas de aprendizaje de una red neuronal recurrente.

Además, también se ha entrenado una RNN (véase su arquitectura en el Apéndice A, Figura: 33) para predecir la secuencia de 12 instantes posteriores de consumo, con un error de entrenamiento de $7{,}80 \cdot 10^{-3}$ y un error de validación de $7{,}74 \cdot 10^{-3}$.

2.4.6.2 Redes neuronales recurrentes con memoria de largo plazo (LSTM, Long Short-Term Memory)

Los modelos conocidos como Long Short-Term memory (LSTM) son un tipo de redes neuronales recurrentes que nacieron con el objetivo de reducir el efecto de la situación conocida como "vanishing gradients" (gradientes que se van anulando en el tiempo) que sufrían sus antecesoras . En él, las variaciones en los parámetros que llegan a las primeras capas son tan pequeñas que éstas no aprenden. Por ello, se dice que las redes neuronales recurrentes sufren de memoria a corto plazo, limitadas a aprender secuencias cortas.

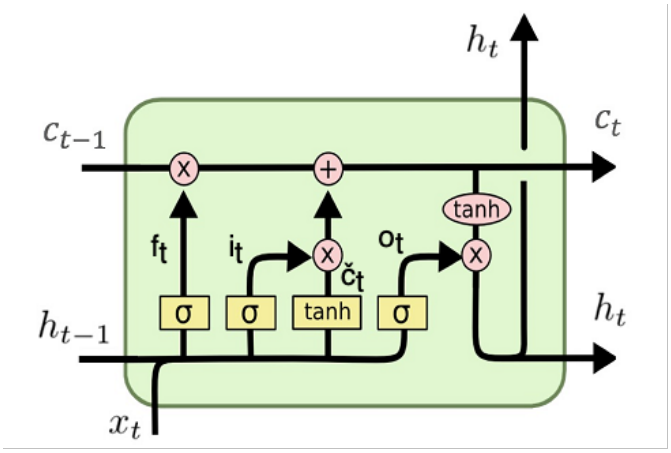

Figura 25: Esquema de una celda LSTM.
Fuente: `http://dprogrammer.org`

Las redes LSTM hacen frente al problema de la memoria a corto plazo con términos adicionales que computan la importancia de cada elemento de la secuencia a predecir:

- Los elementos f_t —llamado "forget gate"— y (i_t x $\check{c}_t$) —conocido como "input gate"— seleccionan los valores más importantes de los instantes previos y actuales, respectivamente.
- El elemento o_t —llamado "output gate"— se encarga de actualizar el estado oculto h_t, que contiene información de las entradas x_i previas.
- Finalmente, c_t —o estado de la célula— transmite la información relevante de los instantes previos y actual a la próxima célula/instante.

Todos estos términos adicionales resultan en un modelo con más parámetros y más costoso de entrenar que una RNN.

Del mismo modo que una RNN, el entrenamiento de este modelo se lleva a cabo empleando la retropropagación a través del tiempo.

Aplicando una red LSTM a la tarea de predicción de consumo, se obtienen las siguientes curvas de aprendizaje:

Figura 26: Curvas de aprendizaje en una red LSTM.

La arquitectura del modelo se muestra en el Apéndice A (Figura 32).

Siguiendo el mismo procedimiento que en el apartado anterior, se ha entrenado una LSTM (véase su arquitectura en el Apéndice A, Figura 34) para predecir la secuencia de 12 instantes posteriores de consumo, obteniendo errores de validación de $6{,}63 \cdot 10^{-3}$ y de $7{,}73 \cdot 10^{-3}$, respectivamente.

3. Resultados

Considerando los resultados obtenidos en el conjunto de validación, los 3 modelos más prometedores —en orden descendente— para predecir el próximo valor del consumo son:

1. El Random Forest del apartado 2.4.4.1.
2. La RNN del apartado 2.4.6.1.
3. El modelo XGBoost del apartado 2.4.4.2.

Y para los modelos destinados a predecir los próximos 12 instantes:

1. El Random Forest del apartado 2.4.5.
2. La red LSTM del apartado 2.4.6.2.
3. La RNN del apartado 2.4.6.1.

Con el objetivo de analizar y evaluar los modelos estudiados durante el trabajo, se lleva a cabo el siguiente procedimiento:

- Se entrenan los modelos sobre el conjunto de entrenamiento, y se toman datos de los errores cometidos (MSE) en los conjuntos de entrenamiento y validación. Además, también se mide la duración de los procesos de entrenamiento y de predicción. Lo realizado en esta fase se etiqueta con el número “1”.

- Se entrenan de nuevo los modelos sobre el conjunto de entrenamiento completo (esto incluye a los conjuntos de entrenamiento y validación anteriores) y se miden los errores cometidos (MSE y R^2) sobre los conjuntos de entrenamiento y evaluación. Lo realizado en esta fase se etiqueta con el número "2".

A continuación, se muestran los resultados finales:

Tabla 3: Tabla de errores para modelos que predicen el siguiente instante de consumo.

Modelo	**train 1 mse**	**val mse**	**train 2 mse**	**test mse**	**test R^2**	**δ mse 1**	**δ mse 2**
xgb	5.863e-04	8.990e-04	6.250e-04	8.207e-04	9.788e-01	3.127e-04	1.957e-04
rfr	5.989e-04	8.827e-04	6.268e-04	8.360e-04	9.784e-01	2.838e-04	2.092e-04
dt3	6.821e-04	1.307e-03	7.178e-04	1.074e-03	9.722e-01	6.253e-04	3.563e-04
etr	1.052e-03	1.163e-03	1.079e-03	1.094e-03	9.717e-01	1.113e-04	1.520e-05
dt2	1.078e-03	1.348e-03	1.082e-03	1.193e-03	9.692e-01	2.705e-04	1.107e-04
rnn	1.070e-03	9.914e-04	1.050e-03	1.205e-03	—	7.842e-05	1.550e-04
lstm	1.322e-03	1.119e-03	1.271e-03	1.320e-03	—	2.030e-04	4.895e-05
dnn	1.439e-03	1.236e-03	1.388e-03	1.384e-03	—	2.027e-04	3.377e-06
dt1	0.000e+00	1.807e-03	0.000e+00	1.772e-03	9.542e-01	1.807e-03	1.772e-03
lr	2.911e-03	2.702e-03	2.858e-03	2.614e-03	9.324e-01	2.097e-04	2.433e-04
naive 1h	3.971e-03	3.773e-03	—	3.706e-03	9.042e-01	—	—

Tabla 4: Tabla de tiempos para los modelos que predicen el siguiente instante de consumo.

Modelo	**train 1 (s)**	**train 1 pred (s)**	**val pred (s)**	**train 2 (s)**	**train 2 pred (s)**	**test pred (s)**
lr	1.047e-02	3.045e-03	8.638e-04	1.727e-02	1.138e-03	6.313e-04
etr	1.149e-01	1.055e-01	1.049e-01	1.139e-01	1.052e-01	1.044e-01
dt2	1.582e-01	2.184e-03	9.212e-04	2.102e-01	2.909e-03	9.279e-04
dt3	1.939e-01	2.635e-03	1.071e-03	2.582e-01	3.570e-03	1.064e-03
xgb	2.266e-01	7.170e-03	2.777e-03	2.948e-01	8.527e-03	1.977e-03
rfr	3.176e-01	1.057e-01	1.047e-01	3.161e-01	1.062e-01	1.048e-01
dt1	3.427e-01	5.930e-03	1.926e-03	4.375e-01	8.217e-03	2.104e-03
lstm	1.534e+02	6.858e-01	1.255e-01	4.563e+00	3.416e-01	1.226e-01
dnn	1.892e+01	2.439e-01	8.895e-02	2.460e+01	2.260e-01	8.818e-02
rnn	2.001e+02	6.352e-01	9.269e-02	2.366e+02	2.162e-01	8.736e-02
naive 1h	—	—	—	—	—	—

Tabla 5: Datos de complejidad para los modelos que predicen el siguiente instante de consumo.

Modelo	**Complejidad**
xgb	10 árboles de profundidad 10
rfr	10 árboles de profundidad 10
dt3	Un árbol de profundidad 10
etr	10 árboles de profundidad 10
dt2	Un árbol de profundidad 8
rnn	$\sim$ 10.01e+02 parámetros. Epochs(Train1,Train2)=(600,600). Early stop: 0
lstm	$\sim$ 68.36e+02 parámetros. Epochs(Train1,Train2)=(245,6). Early stop: 5
dnn	$\sim$ 3.85e+02 parámetros. Epochs(Train1,Train2)=(78,110). Early stop: 5
dt1	Un árbol de profundidad 30
lr	15 parámetros
naive 1h	—

Tabla 6: Tabla de errores para modelos que predicen secuencias de 12 horas de consumo.

Modelo	**train 1 mse**	**val mse**	**train 2 mse**	**test mse**	**test R^2**	**δ mse 1**	**δ mse 2**
rfr	4.358e-03	6.804e-03	4.662e-03	6.375e-03	8.353e-01	2.446e-03	1.713e-03
lstm	6.427e-03	6.274e-03	6.386e-03	7.393e-03	—	1.530e-04	1.006e-03
rnn	7.219e-03	6.902e-03	7.137e-03	7.631e-03	—	3.168e-04	4.937e-04
mlr	1.875e-02	1.857e-02	1.863e-02	1.820e-02	5.298e-01	1.710e-04	4.335e-04
naive 12h	2.395e-02	2.308e-02	—	2.331e-02	0.000e+00	—	—

Tabla 7: Tabla de tiempos para los modelos que predicen secuencias de 12 horas de consumo.

Modelo	**train 1 (s)**	**train 1 pred (s)**	**val pred (s)**	**train 2 (s)**	**train 2 pred (s)**	**test pred (s)**
mlr	1.764e+00	6.984e-01	3.868e-02	2.050e-01	1.335e-01	4.007e-02
rfr	1.044e+00	1.084e-01	1.076e-01	1.458e+00	1.086e-01	1.080e-01
lstm	1.290e+03	1.326e+01	4.003e+00	2.626e+02	1.586e+01	4.006e+00
rnn	2.093e+02	1.795e+00	5.364e-01	2.663e+02	2.003e+00	5.350e-01
naive 12h	—	—	—	—	—	—

Tabla 8: Datos de complejidad para los modelos que predicen secuencias de 12 horas de consumo.

Modelo	**Complejidad**
rfr	50 árboles de profundidad 10
lstm	$\sim$ 22.24e+03 parámetros. Epochs(Train1,Train2)=(38,6). Early stop: 5
rnn	$\sim$ 2.41e+03 parámetros. Epochs(Train1,Train2)=(50,50). Early stop: 10
mlr	180 parámetros.
naive 12h	—

Notas sobre las tablas anteriores:

Las Tablas 3 y 6 están ordenadas según el error cometido en el conjunto de evaluación, en orden ascendente. Esto quiere decir que los primeros modelos de estas tablas han obtenido los mejores resultados. Las Tablas 4 y 7 están ordenadas según el tiempo empleado para realizar el segundo entrenamiento en orden ascendente. Esto significa que, para los primeros modelos de estas tablas, el entrenamiento ha sido más rápido.

El valor "δ" reflejado en las dos últimas columnas de las Tablas 3 y 6 es la diferencia entre el error cometido en el conjunto de entrenamiento y el error cometido en el conjunto de validación (o en el de evaluación, si tiene la etiqueta "2"). La palabra "pred" indica un dato tomado sobre la acción de predicción del modelo. La palabra "dt" indica un árbol de decisión. "etr" es un tipo de Random Forest con un grado más de aleatoriedad: el umbral aplicado para dividir cada árbol del conjunto se escoge de manera aleatoria.

La cadena "mlr" (Tabla 8) se refiere a un modelo de regresión lineal múltiple compuesto por 12 regresiones lineales entrenadas independientemente para cada instante que se desea predecir —como se comenta en el apartado 2.4.5—. En la Tabla 8, "Early stop: t" indica que el entrenamiento se detiene de manera automática tras un número "t" de epochs en las que el error de validación no mejora.

Importante: Los modelos entrenados en este apartado no son los mismos que los de los apartados anteriores, pero tienen la misma arquitectura.

En las siguientes Figuras (27 y 28) se representa el consumo real frente al predicho por el mejor modelo en cada caso (predicción del próximo valor y predicción de los próximos 12 valores). En ellas, el consumo está normalizado —para conocer el consumo real habría que desescalar la predicción realizada, llevando a cabo el proceso inverso del apartado 2.4.2—.

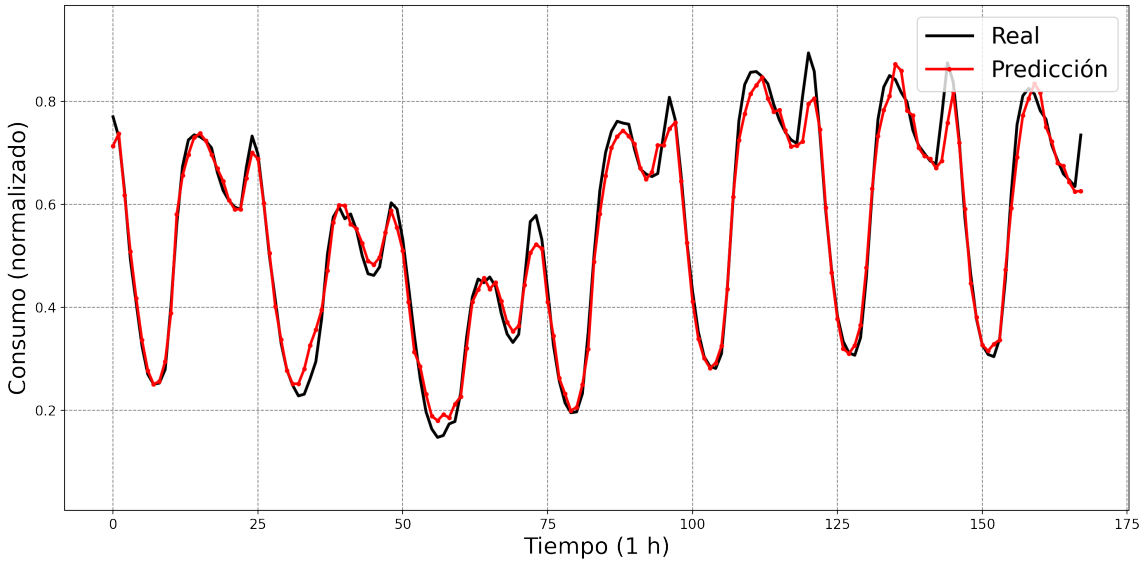

Figura 27: Ejemplo de predicción del próximo instante de consumo energético por el modelo XGBoost para el conjunto de evaluación.

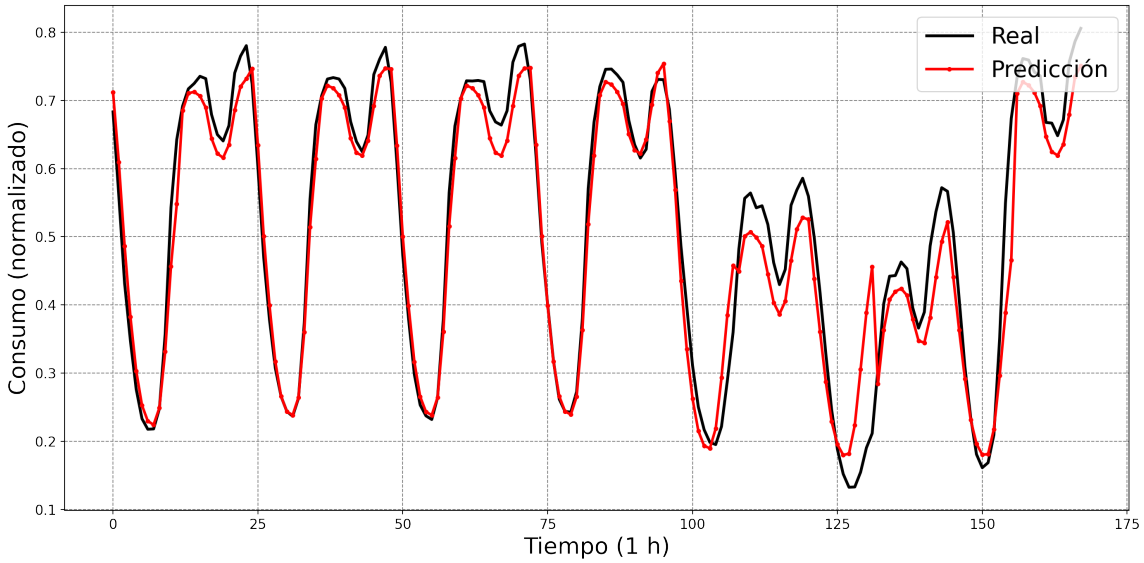

Figura 28: Ejemplo de predicción de los próximos 12 instantes de consumo energético por el modelo Random Forest para el conjunto de evaluación. Nota: La curva roja está compuesta uniendo predicciones en bloques de 12.

Los gráficos anteriores muestran las predicciones realizadas en una semana escogida al azar del conjunto de evaluación. Como puede observarse, los modelos no sólo son capaces de advertir la tendencia semanal, sino que también han aprendido a identificar ciclos diarios.

4. Conclusiones

En la Tabla 3 se observa que todos los modelos superan al de referencia.

Además, se aprecia que el modelo con el error más pequeño en el conjunto de evaluación (test) para las dos medidas de error es XGBoost, compuesto por 10 árboles de profundidad 10.

Este resultado es realmente interesante, ya que además de confirmar que la intuición de situar al modelo entre los tres favoritos era buena, indica que **el error cometido en el conjunto de validación no es lo único a tener en cuenta a la hora de escoger un modelo**.

Si se comparan las curvas de aprendizaje de los 3 modelos favoritos, se observa que tanto el Random Forest (Figura 17) como la RNN (Figura 24) han llegado a una meseta en la que el error de validación no sigue disminuyendo. Sin embargo, si nos fijamos en las curvas de aprendizaje del XGBoost (Figura 18), se observa que si se aumentase el número de muestras de entrenamiento, el error de validación seguiría disminuyendo, hasta casi alcanzar al error de entrenamiento (en la Tabla 3, "δ mse 2" disminuye respecto a "δ mse 1"). Esto es precisamente lo que se hace en la fase 2 en la generación de resultados: se aumentan las muestras de entrenamiento uniendo los conjuntos de entrenamiento y validación.

Así, se comprueba que **las curvas de aprendizaje** de un modelo —expresadas en función del numero de muestras de entrenamiento— **tienen un papel importante a la hora de escoger un modelo final**. Este mismo hecho de aumentar las muestras de entrenamiento también explica por qué ha sido menor el error de test que el de validación.

Pese a que el coste temporal del modelo de regresión lineal se encuentra entre los más pequeños, no es un modelo viable en ningún caso, ya que cuenta con los peores resultados en las tablas de errores (Tablas 3 y 6).

También se observa que los modelos relacionados con árboles de decisión tienen generalmente los mejores resultados en cuanto a coste temporal se refiere —a excepción del modelo dt1, que se ha dejado como ejemplo de overfitting—, mientras que las redes neuronales ofrecen los peores tiempos (es decir, los más altos). Esto puede ser un problema ya que son precisamente las redes neuronales las que más datos necesitan para entrenar sus miles de parámetros (Tablas 5 y 8).

En lo que respecta a los modelos de redes neuronales, se intuye que pueden estar sufriendo una situación de underfitting. Esto puede verse comparando sus resultados con los de un modelo que sí que haya tenido un buen rendimiento, como por ejemplo el xgb (Tabla 3): el error de entrenamiento aumenta cuando aumentan las muestras de entrenamiento, ya que el modelo tiene que adaptarse a datos nuevos. Sin embargo, el error de validación es mayor que el error de test, indicando que los datos de test son una muestra representativa del conjunto y que el modelo está preparado para enfrentarse a datos nuevos. Sin embargo, a un modelo como la red LSTM (Tabla 3) le ocurre lo contrario que al xgb: el error de entrenamiento disminuye al aumentar el volumen de datos

de entrenamiento, indicando que el modelo aún no se ha ajustado lo máximo posible y que todavía está aprendiendo.

Aunque no haya sido una cuestión decisiva, se ha visto que el hecho que de que los modelos evaluados no sean los mismos que los originales repercute especialmente en el caso de las redes neuronales. El motivo es que la inicialización de los parámetros de las redes tiene un gran componente de aleatoriedad. Esto explica las diferencias entre resultados en los conjuntos de entrenamiento y validación del apartado 3 y los anteriores.

Por los motivos mencionados anteriormente, se decide que los modelos más apropiados para la tarea de predicción de consumo energético son: el XGBoost para la predicción del próximo instante de consumo, y el Random Forest (rfr en la Tabla 8) para predecir la secuencia de los 12 instantes siguientes.

En cuanto a la partición del conjunto de datos, se ha observado que es apropiada, ya que da lugar a modelos con buenos resultados. Esto implica que el consumo en España durante las fechas comprendidas en el conjunto de evaluación sigue una distribución similar a la del consumo en los conjuntos de entrenamiento y validación.

Hay que destacar la importancia de los pasos anteriores a la fase de diseño de los modelos (EDA, feature engineering etc), ya que de ellos depende la construcción de un buen conjunto de datos con unas características apropiadas. Y es que **un modelo de ML está condicionado por la calidad —y por supuesto, también por la cantidad— de los datos con los que aprende**.

4.1. Mejoras, aplicaciones y futuras líneas de investigación

Mejoras

Para solucionar el underfitting de las redes neuronales, sería recomendable disponer de más datos con los que pudieran entrenar. Otra manera de disminuir este efecto sería escoger mejor las características que utiliza el modelo o eliminar las que no son tan relevantes. Esto parece una buena solución, ya que como se ve en la Figura 9, los instantes de consumo anteriores más cercanos están más correlacionados con el consumo en el instante siguiente.

En caso de disponer de más datos, sería recomendable llevar a cabo procesos como la validación cruzada [12] antes de elegir el modelo final. De esta manera, se obtendría un error de validación más fiable.

Además, podrían llevarse a cabo técnicas de búsqueda de los hiperparámetros adecuados (como por ejemplo Grid search y Randomized Search [13]) para asegurarse de obtener el modelo óptimo antes de la evaluación final.

Aplicaciones y futuras líneas de investigación

En pocas palabras, ser capaz de predecir el valor de ciertas variables es siempre útil en cualquier sistema de control y automatización. Por ello, las ideas de este trabajo pueden extrapolarse a una multitud de casos.

Desde luego, aplicaciones de este trabajo guardan una relación directa con empresas del ámbito de la generación y suministro de energía. Si se es capaz de anticipar el consumo en una cierta región, se minimizarán las pérdidas de energía —generando únicamente la necesaria—, resultando en un sistema más eficiente y sostenible.

Además, también puede utilizarse el modelo entrenado con los datos de un país para intentar predecir el consumo en otro diferente. Si las predicciones son válidas, significará que ambos países tienen hábitos similares, abriendo las puertas a un mercado energético compartido.

Siguiendo con este último ejemplo, un futuro trabajo podría extender lo realizado mediante aprendizaje supervisado a un sistema que incluyera también aprendizaje no supervisado. Así, podrían buscarse mercados energéticos similares de manera automática, y aplicar algunas de las técnicas de consumo estudiadas en este TFG.

Referencias

[1] Michael Haenlein and Andreas Kaplan. A brief history of artificial intelligence: On the past, present, and future of artificial intelligence. *California Management Review*, 61(4):5–14, 2019.

[2] Yoshua Bengio. Machines who learn. *Scientific American*, 314(6):44–51, 2016.

[3] Anshul Bansal, Susheel Kaushik Rompikuntla, Jaganadh Gopinadhan, Amanpreet Kaur, and Zahoor Ahamed Kazi. Energy consumption forecasting for smart meters. *arXiv preprint arXiv:1512.05979*, 2015.

[4] Western Europe Power Consumption. https://www.kaggle.com/francoisraucent/western-europe-power-consumption. Accedido: 19/03/2021.

[5] Sebastian Raschka and Vahid Mirjalili. *Python Machine Learning*, page 124. Packt Publishing, third edition, 2019.

[6] deeplearning.ai: Train/Dev/Test Sets (C2W1L01). https://www.youtube.com/watch?v=1waHlpKiNyY.

[7] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*, pages 118–128. O'Reilly Media, second edition, 2019.

[8] Wikipedia: Wisdom of the crowd. https://en.wikipedia.org/wiki/Wisdom$_o f_t he_c rowd$.

[9] Towards Data Science: Understanding the bias-variance tradeoff. https://towardsdatascience.com/understanding-the-bias-variance-tradeoff-165e6942b229.

[10] Xgboost documentation. https://xgboost.readthedocs.io/en/latest/index.html.

[11] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[12] Cross-validation strategies for time series forecasting. https://hub.packtpub.com/cross-validation-strategies-for-time-series-forecasting-tutorial/.

[13] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*, pages 76–78. O'Reilly Media, second edition, 2019.

A. Figuras

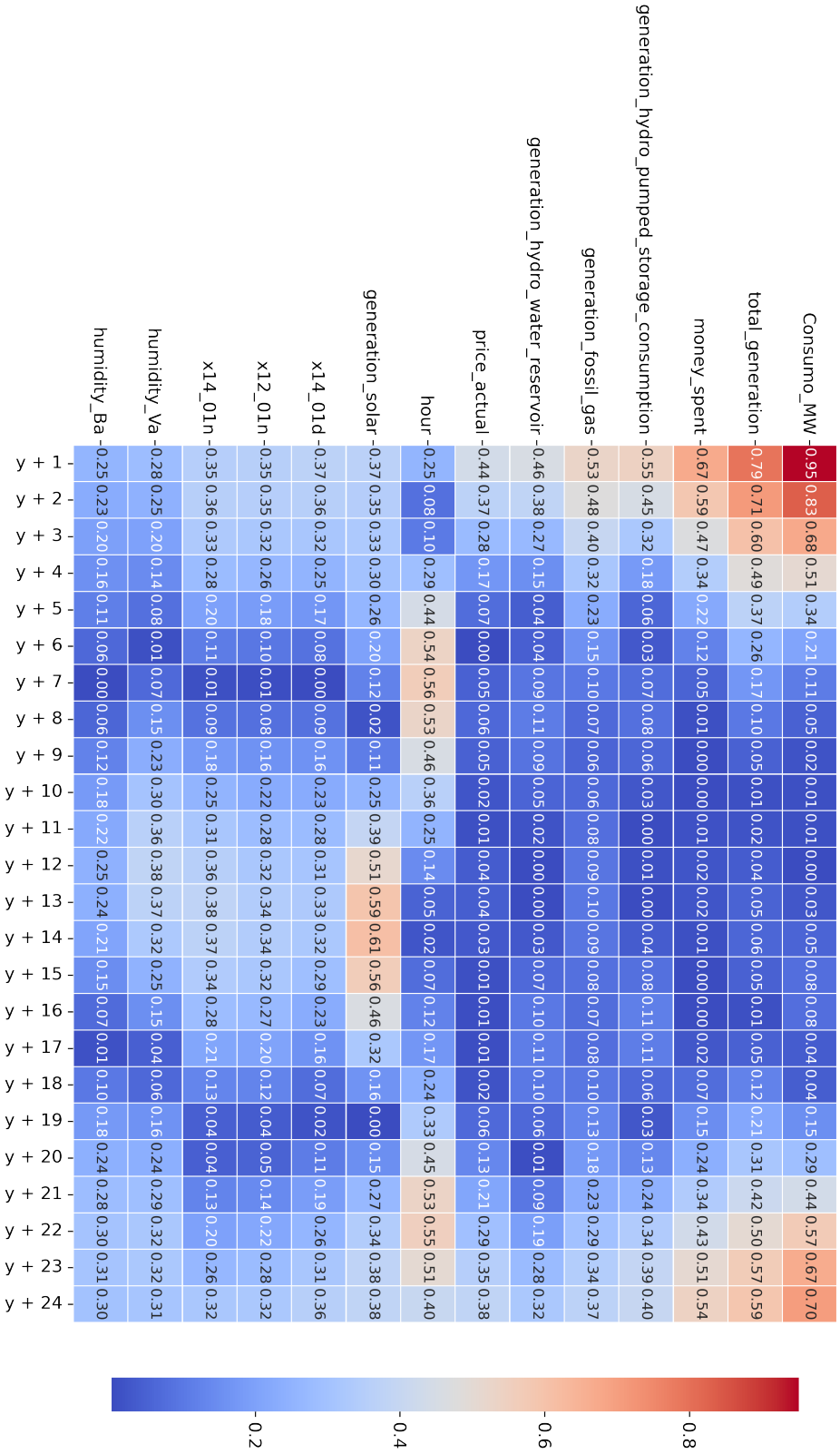

Figura 29: Mapa de calor para las variables seleccionadas y los datos que se desean predecir.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 1, 24)             360
_________________________________________________________________
dense_1 (Dense)              (None, 1, 1)              25
=================================================================
Total params: 385
Trainable params: 385
Non-trainable params: 0
_________________________________________________________________
```

Figura 30: Resumen de la FFNN.

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn_5 (SimpleRNN)     (None, 1, 14)             406
_________________________________________________________________
simple_rnn_6 (SimpleRNN)     (None, 1, 10)             250
_________________________________________________________________
simple_rnn_7 (SimpleRNN)     (None, 1, 10)             210
_________________________________________________________________
simple_rnn_8 (SimpleRNN)     (None, 1, 7)              126
_________________________________________________________________
simple_rnn_9 (SimpleRNN)     (None, 1)                 9
=================================================================
Total params: 1,001
Trainable params: 1,001
Non-trainable params: 0
```

Figura 31: Resumen de la RNN.

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 1, 20)             2800
_________________________________________________________________
lstm_1 (LSTM)                (None, 1, 15)             2160
_________________________________________________________________
lstm_2 (LSTM)                (None, 1, 15)             1860
_________________________________________________________________
dense_2 (Dense)              (None, 1, 1)              16
=================================================================
Total params: 6,836
Trainable params: 6,836
Non-trainable params: 0
_________________________________________________________________
```

Figura 32: Resumen de la LSTM-RNN.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn_2 (SimpleRNN)     (None, 24, 24)            936
_________________________________________________________________
simple_rnn_3 (SimpleRNN)     (None, 24)                1176
_________________________________________________________________
dense_1 (Dense)              (None, 12)                300
=================================================================
Total params: 2,412
Trainable params: 2,412
Non-trainable params: 0
_________________________________________________________________
```

Figura 33: Resumen de la RNN para predecir los próximos 12 instantes de consumo.

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, None, 48)          12096
_________________________________________________________________
lstm_1 (LSTM)                (None, None, 24)          7008
_________________________________________________________________
lstm_2 (LSTM)                (None, None, 12)          1776
_________________________________________________________________
lstm_3 (LSTM)                (None, 12)                1200
_________________________________________________________________
dense_2 (Dense)              (None, 12)                156
=================================================================
Total params: 22,236
Trainable params: 22,236
Non-trainable params: 0
```

Figura 34: Resumen de la LSTM-RNN para predecir los 12 instantes próximos de consumo.

B. Tablas

Tabla 9: Columnas de los conjuntos de datos de energía y meteorología.

Dataset Energético	**Dataset Meteorológico**
time	dt iso
generation biomass	city name
generation fossil brown coal/lignite	temp
generation fossil coal-derived gas	temp min
generation fossil gas	temp max
generation fossil hard coal	pressure
generation fossil oil	humidity
generation fossil oil shale	wind speed
generation fossil peat	wind deg
generation geothermal	rain 1h
generation hydro pumped storage aggregated	rain 3h
generation hydro pumped storage consumption	snow 3h
generation hydro run-of-river and poundage	clouds all
generation hydro water reservoir	weather id
generation marine	weather main
generation nuclear	weather description
generation other	weather icon
generation other renewable	-
generation solar	-
generation waste	-
generation wind offshore	-
generation wind onshore	-
forecast solar day ahead	-
forecast wind offshore day ahead	-
forecast wind onshore day ahead	-
total load forecast	-
total load actual	-
price day ahead	-
price actual	-

C. Código

Los archivos creados durante la realización de este proyecto se encuentran en el siguiente repositorio de GitHub:

`https://github.com/divendor/energy-consumption-forecasting-with-ML`

Nota: Se recomienda leer el archivo "README.md". En él, se explica el orden de lectura recomendado para la comprensión del código.