UNIVERSIDAD PONTIFICIA DE
SALAMANCA
FACULTAD DE INFORMÁTICA

Trabajo Fin de Grado

Estudio, clasificación y predicción de consumo en contadores inteligentes

Pedro Juiz Díaz

Dr. Manuel Martín-Merino Acera

Salamanca, Julio de 2023

Agradecimientos

Me gustaría dedicar este proyecto en primer lugar a mis padres y a mi hermano, por todo el esfuerzo y el apoyo que siempre he tenido por su parte, porque a pesar de todo, siempre han estado ahí para darme un pequeño empujón cuando no me quedaban fuerzas. Ha sido un camino duro, en el que de no haber ido de su mano, nunca podría haber llegado al final.

También me gustaría mencionar a Amaia, mi compañera de vida, un apoyo durante gran parte de este trayecto y una gran motivación para seguir siempre adelante y no rendirme. Siempre dispuesta a brindarme su mano cuando más la necesito y agradecerle por resistir en los momentos duros y de alto estrés.

Por último, mencionar a mi tutor, Manuel. Sin llegar a cumplir tan siquiera los 18 años, me embauque en un gran reto como es la universidad. Por casualidad, la primera de muchísimas clases a lo largo de la carrera fue con él, y con él también termina esta etapa. Un profesor que siempre ha estado dispuesto a ayudar a cualquiera y a tender su mano. Ha sido un orgullo para mí poder compartir mi etapa universitaria con él.

Resumen

Este Trabajo de Fin de Grado explora la relación entre las condiciones meteorológicas y el consumo energético de los hogares en Londres, con el objetivo de comprender los patrones de consumo y mejorar la eficiencia energética. Para su desarrollo, se implementan 3 tipos de modelos predictivos avanzados con el fin de realizar una comparativa entre ellos y ver cuál es el que mejor se ajusta a nuestros datos y nos ofrece mejores resultados. A mayores, se realiza un modelado del perfil de consumo de los usuarios utilizando modelos de clustering, con el fin de segmentar a los usuarios en grupos en función de su consumo. Como resultados, obtenemos que para la parte predictiva del proyecto el modelo que mejor se ajusta a nuestros datos es una red neuronal LSTM, mientras que, por la parte de modelado de perfiles, obtenemos 3 grupos distintos de usuarios que analizaremos para tener más información acerca de estos.

Abstract

This Final Year Project explores the relationship between weather conditions and household energy consumption in London, with the aim of understanding consumption patterns and improving energy efficiency. For its development, three types of advanced predictive models are implemented with the purpose of comparing them and determining which one best fits our data and offers superior results. Furthermore, we model the consumption profile of users using clustering models, in order to segment users into groups based on their consumption. As results, we find that for the predictive part of the project the model that best fits our data is a Long Short-Term Memory (LSTM) neural network, while for the user profile modeling, we identify three distinct user groups, which we will analyze to gain more information about them.

Descriptores

Machine Learning, Series temporales, clustering, modelos predictivos.

Índice

Índice de Figuras

Índice de tablas

1. Marco conceptual

1.1 Introducción

Actualmente, vivimos en la era de la información, en la que la gestión de los datos es clave para su correcto funcionamiento. La forma de vida actual y la economía en constante cambio nos llevan a un momento de gran importancia en el que la gestión de los datos es sin duda alguna, uno de los avances más importantes de la historia.

En el ámbito empresarial, es necesario continuar con el desarrollo existente de numerosos softwares para el procesamiento y análisis de grandes cantidades de datos, ya que la información es poder, y se considera uno de los mayores activos relevantes de una empresa. Este procesamiento de datos por empresas, las ayuda a seleccionar la información relevante, convirtiéndola en un activo, lo que mejora al mismo tiempo la competitividad en el sector al que pertenezca.

En este estudio, se realiza un análisis acerca del consumo energético de miles de hogares que participaron en un proyecto conocido como "Low Carbon London" llevado a cabo por UK Power Networks entre noviembre de 2011 y febrero de 2014 (1).

Este estudio se basa en el desarrollo de varios modelos predictivos, con el fin de obtener un pronóstico acerca del consumo futuro de los hogares londinenses que participaron en el proyecto previamente mencionado.

1.1.1 Motivación del proyecto

Tras la privatización del mercado eléctrico en Europa a finales de los años 80, son las empresas las que toman decisiones de inversión y por lo que aumenta el interés del pronóstico de la demanda que electricidad que pueda existir. Además, en una sociedad en la que se evoluciona constantemente hacia las nuevas tecnologías, que a su vez dependen de la energía eléctrica, ha incrementado la relevancia de los pronósticos (2).

Además, la inviabilidad de poder almacenar altas cantidades de energía eléctrica por parte de las empresas y su alto coste implican que la misma energía que se genera se ha de consumir al mismo tiempo. Esto genera en un desequilibro tanto en la producción como en el consumo creando una relación de oferta y demanda poco compensada. Actualmente, las empresas del mercado energético están persiguiendo un mejor aprovechamiento de los recursos primarios del planeta y una mayor eficiencia energética.

En este proyecto se toma el sistema londinense como tema de estudio para el cual se usará un modelo predictivo para poder pronosticar la demanda eléctrica.

1.1.2 Objetivos

El objetivo principal de este proyecto ayudar a las empresas de la industria energética a tomar decisiones basándose en sus datos. Para ello, como objetivo principal se establece el realizar una comparativa entre diferentes modelos de inteligencia artificial que trabajen con series temporales. En este caso, se implementará un modelo Prophet, un modelo de aprendizaje supervisado conocido como Random Forest y una red neuronal LSTM.

En este caso, se comparan estos modelos entre sí, con el fin de determinar cuál es el mejor para predecir el consumo energético de los hogares, cuál es el que mejor se adapta a los datos y cuál es el que tiene un mejor desempeño en las métricas de evaluación.

Además, como objetivo secundario, está el modelar el perfil de consumo de usuario para realizar ofertas más personalizadas basadas en su consumo utilizando técnicas de clustering de aprendizaje no supervisado.

2. Marco teórico

En este capítulo, se realizará una revisión bibliográfica acerca de determinado material teórico, con el fin de dar un punto de vista más conceptual acerca de la práctica realizada.

2.1 Series temporales

"Una serie de tiempo es una secuencia de puntos de datos que ocurren en orden sucesivo durante un período de tiempo. Esto se puede contrastar con los datos transversales, que capturan un punto en el tiempo.

Al invertir, una serie de tiempo rastrea el movimiento de los puntos de datos elegidos, como el precio de un valor, durante un período de tiempo específico con puntos de datos registrados a intervalos regulares. No hay una cantidad mínima o máxima de tiempo que deba incluirse, lo que permite que los datos se recopilan de manera que brinden la información que busca el inversionista o analista que examina la actividad" (3).

"Se puede tomar una serie de tiempo sobre cualquier variable que cambie con el tiempo. Al invertir, es común usar una serie de tiempo para rastrear el precio de una acción a lo largo del tiempo. Esto se puede monitorear a corto plazo, como el precio de una acción cada hora durante un día hábil, o a largo plazo, como el precio de una acción al cierre del último día de cada mes en el transcurso de cinco años. El análisis de series temporales puede ser útil para ver cómo cambia un determinado activo, valor o variable económica a lo largo del tiempo. También se puede utilizar para examinar cómo los cambios asociados con el punto de datos elegido se comparan con los cambios en otras variables durante el mismo período de tiempo" (4).

2.1.1 Análisis de las series temporales

"El análisis de series de tiempo puede ser útil para ver cómo un determinado activo, seguridad o variable económica cambia con el tiempo. También se puede usar para examinar cómo los cambios asociados con el punto de datos elegido se comparan con los cambios en otras variables durante el mismo período de tiempo.

Por ejemplo, supongamos que deseamos analizar una serie temporal de precios diarios de cierre de acciones para una acción determinada durante un período de un año. Obtendremos una lista de todos los precios de cierre de las acciones de cada día del año pasado y los enumerará en orden cronológico. Esta sería una serie temporal de precios de cierre diario de un año para la acción.

Profundizando un poco más, puede interesarnos saber si la serie temporal de la acción muestra alguna estacionalidad para determinar si atraviesa picos y valles en momentos regulares cada año. El análisis en esta área requeriría tomar los precios observados y

correlacionarse con la temporada elegida. Esto puede incluir temporadas de calendario tradicionales, como verano e invierno, o temporadas de venta minorista, como temporadas de vacaciones. Alternativamente, puede registrar los cambios en el precio de las acciones de una acción en relación con una variable económica, como la tasa de desempleo. Al correlacionar los puntos de datos con información relacionada con la variable económica seleccionada, puede observar patrones en situaciones que muestran dependencia entre los puntos de datos y la variable elegida" (5).

2.1.2 Componentes del análisis de series temporales

Las razones o fuerzas que modifican los atributos de una serie temporal se conocen como componentes de la serie temporal. Los siguientes son los componentes de la serie de tiempo:

- La tendencia muestra una tendencia común de los datos. Puede moverse hacia arriba o aumentar o bajar o disminuir durante un cierto período de tiempo prolongado. La tendencia es una tendencia general estable ya largo plazo del movimiento de los datos. Para ser tendencia, no es obligatorio que los datos se muevan en la misma dirección. La dirección o el movimiento pueden cambiar durante el período a largo plazo, pero la tendencia general debe permanecer igual en una tendencia. Una tendencia podría ser lineal o no lineal.

- Variaciones estacionales son cambios en las series temporales que se producen a corto plazo, normalmente en menos de 12 meses. Suelen mostrar el mismo patrón de crecimiento ascendente o descendente en el período de 12 meses de la serie temporal. Estas variaciones a menudo se registran como programaciones horarias, diarias, semanales, trimestrales y mensuales.

Las variaciones estacionales se producen debido a fuerzas o variaciones naturales o provocadas por el hombre. Las numerosas estaciones y variaciones hechas por el hombre juegan un papel vital en las variaciones estacionales.

- Variaciones en las series de tiempo que ocurren en el lapso de más de un año se denominan variaciones cíclicas. Tales movimientos oscilatorios de tiempo graves suelen tener una duración de más de un año. Un período completo de operación se denomina ciclo o "Ciclo comercial". Las variaciones cíclicas contienen cuatro fases: prosperidad, recesión, depresión y recuperación. Puede ser de naturaleza regular o no periódica. Por lo general, las variaciones cíclicas ocurren debido a una combinación de dos o más fuerzas económicas y sus interacciones.

- Movimientos aleatorios o irregulares. Hay otro tipo de movimiento que se puede ver en el caso de las series de tiempo. Es puro Movimiento Irregular y Aleatorio. Como sugiere el nombre, no se puede utilizar ninguna hipótesis o tendencia para sugerir movimientos irregulares o aleatorios en una serie de tiempo. Estos resultados son de naturaleza imprevista, errática, impredecible e incontrolable (6).

2.1.3 Pronóstico

"El pronóstico de series cronológicas utiliza información sobre valores históricos y patrones asociados para predecir la actividad futura. En la mayoría de los casos, esto se relaciona con el análisis de tendencias, el análisis de fluctuaciones cíclicas y cuestiones de estacionalidad. Como con todos los métodos de pronóstico, el éxito no está garantizado" (7).

En esta parte, se pueden destacar 2 vertientes, modelos predictivos basados en técnicas estadísticas tradicionales y modelos de aprendizaje automático que se entrenan a partir de ejemplos.

Modelos que utilizan técnicas estadísticas tradicionales:

- Modelos ARIMA (Modelos de media móvil integrada autorregresiva): Son una categoría común de modelos que emplean diferencias y retrasos de los valores observados para identificar patrones en los datos de series temporales.
- Modelos de suavizado exponencial: Son útiles cuando los datos presentan una tendencia o patrón estacional. Estos modelos emplean un promedio ponderado de observaciones anteriores para realizar predicciones.
- Modelos SARIMA (Modelos de media móvil integrada autorregresiva estacional): Son una extensión de los modelos ARIMA que incorporan elementos estacionales.
- Modelos de regresión: Son útiles cuando existen variables externas que se cree que afectan la serie temporal que se está modelando.

Modelos de aprendizaje automático que se entrenan a partir de ejemplos:

- Redes neuronales: Las redes neuronales, como las redes neuronales recurrentes (RNN) y las redes de memoria a largo plazo (LSTM), pueden ser muy eficaces para modelar series temporales, ya que son capaces de aprender a partir de secuencias de datos.
- Máquinas de soporte vectorial (SVM): Las SVM pueden ser utilizadas para el pronóstico de series temporales al tratar el problema como un problema de regresión.
- Árboles de decisión y bosques aleatorios: Estos modelos pueden ser útiles para capturar relaciones no lineales en los datos.
- Modelos de aumento de gradiente: Los modelos de aumento de gradiente, como XGBoost y LightGBM, pueden ser muy eficaces para el pronóstico de series temporales, especialmente cuando se tienen múltiples características de entrada.

2.2 Machine Learning

El aprendizaje automático es considerado una rama tanto de la inteligencia artificial (IA) como de la informática que se concentra en el uso de datos y algoritmos para imitar la forma en que los humanos aprenden, mejorando gradualmente su precisión.

“La modelación estadística es la formalización de la unión entre variables en el estado de ecuaciones matemáticas” (8). Durante las últimas dos décadas, los avances tecnológicos en el almacenamiento y la potencia de procesamiento han permitido algunos productos innovadores basados en el aprendizaje automático, como el motor de recomendaciones de Netflix y los automóviles autónomos.

El aprendizaje automático es uno de los componentes más importantes del creciente campo de la ciencia de datos. Mediante el uso de métodos estadísticos, estos algoritmos se entrenan para poder hacer clasificaciones o predicciones y así descubrir información clave en proyectos de minería de datos. Estos conocimientos impulsan con posterioridad la toma de decisiones que se puedan producir dentro de las aplicaciones y los negocios, lo que idealmente es impactante con respecto a las métricas de crecimiento clave. A medida que el Big Data continúa expandiéndose y creciendo, la demanda del mercado de científicos de datos aumentará. Se les pedirá que ayuden a identificar las preguntas comerciales más relevantes y los datos para responderlas.

Según UC Berkeley la técnica del Machine Learning es separar el sistema de aprendizaje del algoritmo de aprendizaje automático en tres partes principales (9).

Un proceso de decisión: en general, los algoritmos de aprendizaje automático se utilizan para hacer una predicción o clasificación. En función de algunos datos de entrada, que se pueden etiquetar o no, su algoritmo producirá una estimación sobre un patrón en los datos.

Una función de error: una función de error evalúa la predicción del modelo. Si se dan ejemplos conocidos, una función de error podría hacer una comparación para llegar a evaluar la precisión del modelo.

Un proceso de optimización del modelo: si el modelo puede ajustarse mejor a los puntos de datos en el conjunto de entrenamiento, entonces los pesos se ajustan para así reducir la diferencia entre el ejemplo conocido y la estimación que realiza el modelo. El algoritmo repetirá este proceso de "evaluar y optimizar", actualizando los pesos de forma autónoma hasta alcanzar un umbral de precisión.

2.2.1 Arquitectura de un modelo de análisis de series temporales

Para poder realizar un proyecto de Machine Learning es necesario llevar a cabo una serie de fases:

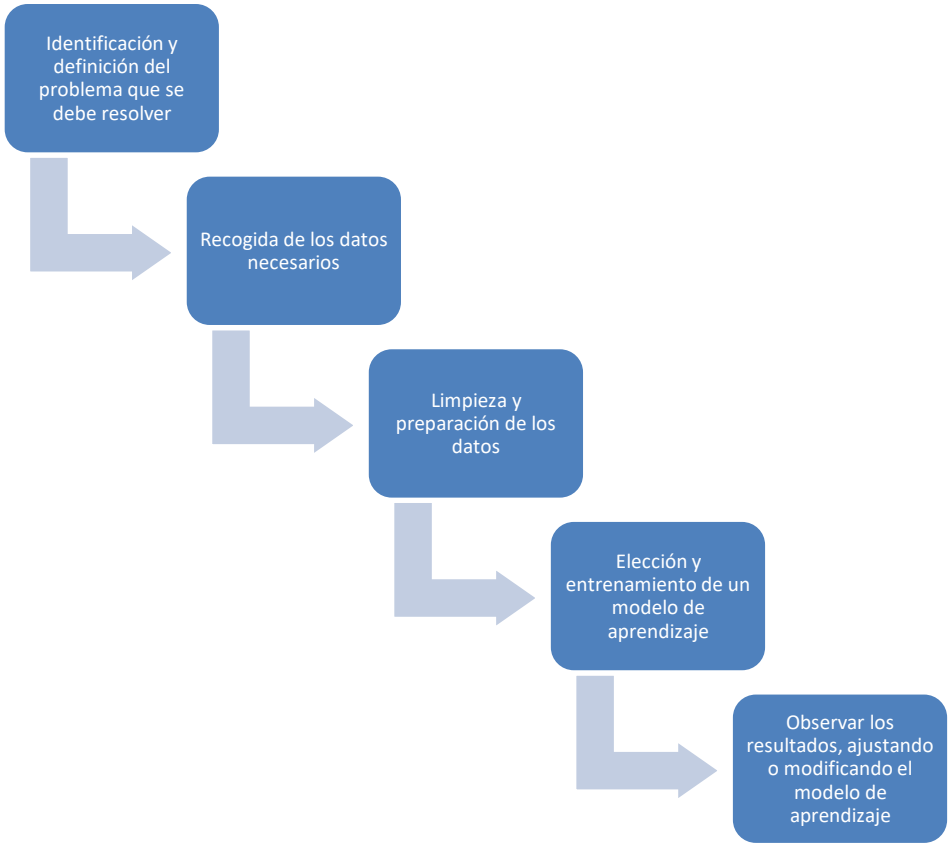


Figura 2-1 Diagrama de flujo (elaboración propia)

2.2.1.1 Identificación y definición del problema que se debe resolver

Es muy importante entender el problema y ser capaces de describir el mismo de forma breve y concisa. Se debe empezar desde un lenguaje más básico a un lenguaje más específico.

Cada proyecto comienza con un problema que debes resolver. Idealmente, una definición clara del problema debe describirse numéricamente. Los números no solo brindan la capacidad de saber dónde está su punto de partida, sino que también le permiten rastrear el efecto de los cambios más adelante. Explicando el problema de esta manera ya se tiene un primer paso bien asentado.

2.2.1.2 Recogida de los datos necesarios

Como hemos mencionado anteriormente, los datos son poder y un gran activo para las empresas. Esta fase de preparación de datos es una de las fases más complejas del Machine Learning debido al esfuerzo necesario y a los desafíos que nos podemos encontrar tales como datos incompletos, ya que es probable que no lleguemos a obtener todos los datos que desearíamos tener. Por ejemplo, al realizar una encuesta donde hay casillas no obligatorias, habrá personas que no rellenen todos los campos. Hay varias formas de lidiar con este problema:

- Eliminar los datos que no estén completos, quedándonos solo con aquellos que si lo estén.
- Imputarlos con un valor razonable, es decir, cuando un valor sea inexistente, se pondrá automáticamente un valor que tenga sentido, como una media entre el resto de encuestados.
- Otra opción es rellenar las variables nulas con el número 0, siempre y cuando se trate de valores numéricos.
- También se puede crear un modelo de Machine Learning que pueda predecir cuál es el valor que nos falta, aprendiendo de aquellos datos que si tenemos.

La procedencia de los datos puede ser de varias fuentes, como bases de datos internas de las propias empresas, también es probable que los clientes compartan con las empresas sus datos en un intercambio de intereses, y, por último, la procedencia externa, en las que, por ejemplo, empresas recopilan datos de diversas poblaciones dentro mercado así pudiendo aprovechar las tendencias para aumentar la rentabilidad.

Nuestro objetivo es recopilar tantos datos relevantes como sea posible. Esto generalmente implica obtener datos para un período de tiempo amplio si hablamos de datos tabulares. Recuerda: cuantas más muestras tengas, mejor será tu futuro modelo (10).

2.2.1.3 Limpieza y preparación de los datos

Los datos recopilados tienden a estar desordenados por lo que los ingenieros de aprendizaje automático se enfrentan a grandes problemas cuando tratan de procesarlos. Los problemas más comunes que se pueden encontrar serían:

- Los datos deben ser filtrados, dejando solo los relevantes y eliminando los irrelevantes.
- Las muestras tanto con ruido como erróneas deben ser identificados y eliminadas.
- Aquellos valores atípicos deben ser reconocidos y eliminados
- Los valores faltantes deben ser detectados y eliminados, imputándose mediante los métodos adecuados
- Y, por último, los datos deben ser convertidos a su formato adecuado

2.2.1.4 Analizar los datos

Los científicos de datos deben ser minuciosos con los datos en los que están trabajando para desarrollar conocimientos sobre su importancia y utilidad. Examinar el tipo y la distribución de datos en cada variable, las relaciones entre ellos y cómo se diferencian del resultado previsto o esperado son todos ejemplos de exploración de datos.

Esta etapa puede revelar problemas como la colinealidad, o variables que se mueven en tándem, o instancias donde se requiere estandarización de conjuntos de datos y otras transformaciones. Puede brindar oportunidades para mejorar el rendimiento del modelo, como reducir la dimensionalidad de un conjunto de datos (11).

2.2.1.5 Limpiar y validar datos

Los equipos de análisis pueden encontrar y corregir inconsistencias, valores atípicos, anomalías, datos faltantes y otros problemas utilizando una variedad de herramientas de limpieza y validación de datos. Las tecnologías de imputación, por ejemplo, pueden lidiar con frecuencia con valores de datos faltantes llenando campos vacíos con alternativas estadísticamente apropiadas. También es útil crear una categoría especial para registrar la importancia de los valores faltantes o establecer deliberadamente los valores faltantes como neutrales.

Para garantizar una alta calidad de los datos, limpiar y validar los datos para el aprendizaje automático, se crean una variedad de herramientas de código abierto, incluidas Pandera y Great Expectations, para verificar la validez de los marcos de datos que se usan con frecuencia para organizar los datos analíticos en dos dimensiones. Hay varias herramientas disponibles para validar flujos de trabajo para el procesamiento de datos y la programación, como pytest (12).

2.2.1.6 Estructuración de datos

Las técnicas de regularización de datos, como el agrupamiento de datos y el suavizado de características continuas, reducen la varianza del modelo al eliminar pequeñas fluctuaciones estadísticas. Se puede utilizar un enfoque equidistante (cada contenedor tiene el mismo "ancho") o un método equiestadístico (cada contenedor tiene aproximadamente la misma cantidad de muestras) para agrupar datos en varias categorías.

También puede servir como condición previa en la optimización local de los datos en cada contenedor para ayudar a crear modelos de bajo sesgo. El suavizado continuo de características puede ayudar a "eliminar el ruido" de los datos sin procesar. También es posible imponer suposiciones causales sobre el proceso de generación de datos al representar relaciones en conjuntos de datos ordenados como funciones monótonas que mantienen el orden de las piezas de datos.

La creación de conjuntos de datos separados para entrenar y probar modelos, la reducción de datos a través de métodos como el muestreo de atributos o registros y la agregación de datos, la normalización de datos mediante el uso de reducción de dimensionalidad y el reescalado de datos son otras formas de estructurar datos para el aprendizaje automático (13).

2.2.1.7 Selección e ingeniería de características

El último paso en la preparación de datos antes de crear un modelo de aprendizaje automático es la ingeniería y selección de características. La ingeniería de características implica agregar o desarrollar variables adicionales para mejorar la salida de un modelo. Los ejemplos incluyen la separación de variables en características distintas, la agregación de variables y el cambio de características de acuerdo con las distribuciones de probabilidad.

La selección de funciones implica elegir funciones útiles para estudiar y eliminar las irrelevantes. Muchas funciones que inicialmente parecen prometedoras pueden limitar la capacidad de un modelo para evaluar con precisión los datos nuevos debido al ajuste excesivo y al entrenamiento prolongado del modelo. Técnicas como la regresión de lazo y la evaluación de relevancia algorítmica también ayudan en la selección de funciones (14).

2.2.1.8 Elección y entrenamiento de un modelo de aprendizaje

La efectividad en el mundo real de un modelo de aprendizaje automático depende de su capacidad para generalizar, para aplicar la lógica aprendida de los datos de entrenamiento a datos nuevos e invisibles.

Para contrarrestar esto, los datos preparados generalmente se dividen en datos de entrenamiento y de prueba. La mayoría del conjunto de datos se reserva como datos de entrenamiento (alrededor del 80 % del conjunto de datos general) y también se crea un subconjunto de datos de prueba. Luego, el modelo puede entrenarse y construirse a partir de los datos de entrenamiento, antes de medirse con los datos de prueba.

A la hora de entrenar un modelo, debe tenerse en cuenta que cuando el modelo se entrena durante tiempo excesivo con datos de muestra o cuando el modelo es demasiado complejo, puede comenzar a reconocer el "ruido" o información que es irrelevante dentro del conjunto de datos. Cuando el modelo reconoce el ruido y se ajusta excesivamente al conjunto de entrenamiento, el modelo llega a "sobre ajustarse" y por lo tanto, no puede generalizar de forma correcta los nuevos datos. Si un modelo no puede universalizarse bien a nuevos datos, ya no podrá realizar las distintas tareas de clasificación o predicción para las que fue diseñado. Las mínimas tasas de error y la alta varianza se consideran buenos indicadores de sobreajuste.

Los datos de prueba actúan como datos nuevos e invisibles, lo que permite evaluar la precisión y los niveles de generalización del modelo.

El proceso se denomina validación cruzada en el aprendizaje automático, ya que valida la eficacia del modelo frente a datos ocultos. Hay una variedad de técnicas de validación cruzada, categorizadas como enfoques exhaustivos y no exhaustivos. Las técnicas exhaustivas de validación cruzada probarán todas las combinaciones e iteraciones de un conjunto de datos de entrenamiento y prueba. Las técnicas de validación cruzada no exhaustivas crearán una partición aleatoria de subconjuntos de entrenamiento y prueba. El enfoque exhaustivo proporcionará una visión más profunda del conjunto de datos, pero requerirá mucho más tiempo y recursos en contraste con un enfoque no exhaustivo (15).

2.2.1.9 Observar los resultados, ajustando o modificando el modelo de aprendizaje

La última parte de un proyecto de Machine Learning consiste en observar y evaluar los resultados obtenidos mediante el uso de diferentes técnicas y métodos. Esto incluye la comparación de los resultados obtenidos con los objetivos previstos, así como la evaluación de la precisión y rendimiento del modelo utilizado. También se pueden utilizar técnicas de validación cruzada para evaluar el rendimiento del modelo en diferentes conjuntos de datos. Además, se realizan análisis de los resultados, con el fin de identificar cualquier posible problema o limitación del modelo y proponer soluciones para mejorar el rendimiento de este.

2.2.2 Tipos de aprendizaje

Cabe destacar, que dentro de lo que conocemos como Machine Learning, se diferencian tres tipos de aprendizaje, el aprendizaje supervisado, aprendizaje no supervisado y el aprendizaje por refuerzo o reforzado.

2.2.2.1 Aprendizaje supervisado.

El aprendizaje supervisado es una de las ramas de Machine Learning en la que se utilizan algoritmos que aprenden de forma iterativa con los datos que se le introducen, para lograr que los ordenadores sean capaces de localizar información oculta sin necesidad de programar dicho ordenador para que sepa dónde debe buscar. Los algoritmos de aprendizaje supervisado tratan

de modelar relaciones y dependencias entre las características de entrada de los datos y el resultado predictivo objetivo, con el fin de poder predecir valores de salida (16).

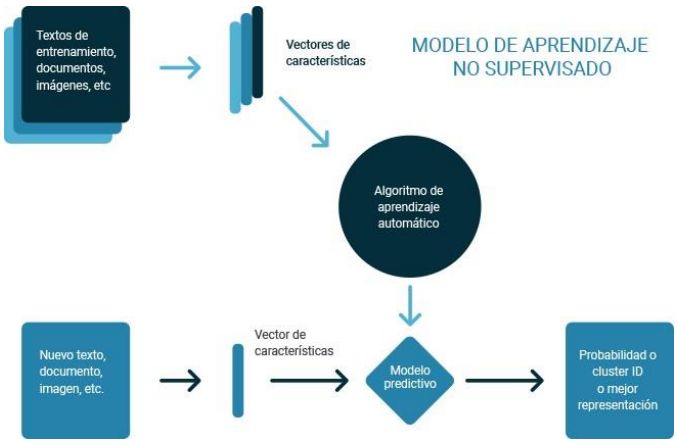


Figura 2-2 Ejemplo de cómo funciona el aprendizaje supervisado (48)

2.2.2.2 Aprendizaje no supervisado

El término no supervisado, hace referencia a que los algoritmos no son guiados como los utilizados en el aprendizaje supervisado. El aprendizaje no supervisado, se basa en algoritmos que deducen ciertos patrones de un conjunto de datos sin etiquetar. Estos métodos no pueden ser aplicados, a diferencia de los algoritmos de aprendizaje supervisado, a problemas de clasificación o regresión, dado que no tiene idea de cuales puedenllegar a ser los valores de los datos de salida. Sin embargo, estos pueden utilizarse con el fin de obtener la estructura subyacente de los datos, es decir, para agrupar los datos en función de patrones y similitudes (17).

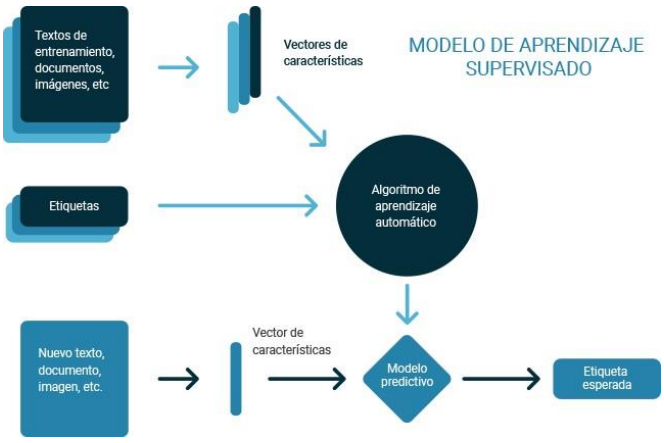


Figura 2-3 Ejemplo de cómo funciona el aprendizaje no supervisado (47)

2.2.2.3 Aprendizaje por refuerzo

El aprendizaje por refuerzo es una rama del machine learning en la que la propia máquina se encarga de dirigir su propio aprendizaje basándose en castigos y recompensas. Dicha máquina, está en una búsqueda constante de decisiones en la que obtenga premio, del mismo modo que trata de evitar aquellas que le supongan una penalización (18).

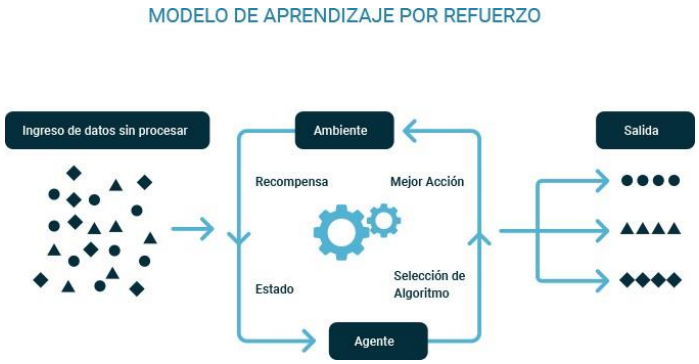


Figura 2-4 Ejemplo de cómo funciona el aprendizaje por refuerzo (49)

2.2.3 Tipos de algoritmos

2.2.3.1 Algoritmos de regresión

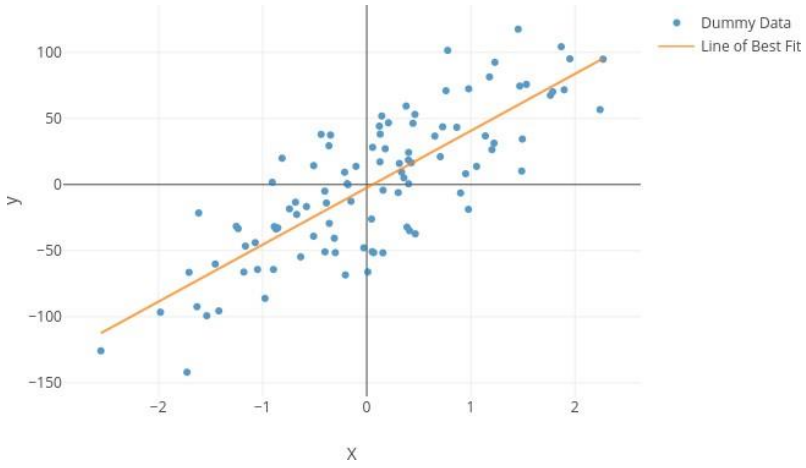


Figura 2-5 Ejemplo de un algoritmo de regresión lineal

Los algoritmos de regresión se encargan de modelar la relación entre diferentes variables. Estos algoritmos hacen uso de una medida de error que se tratará de minimizar con un proceso iterativo, con el fin de lograr hacer predicciones lo más precisas posibles. Son utilizados sobre todo en análisis estadístico.

A pesar de que existen múltiples tipos de regresión, los más utilizados son la regresión lineal y la regresión logística (19).

2.2.3.2 Árboles de decisión

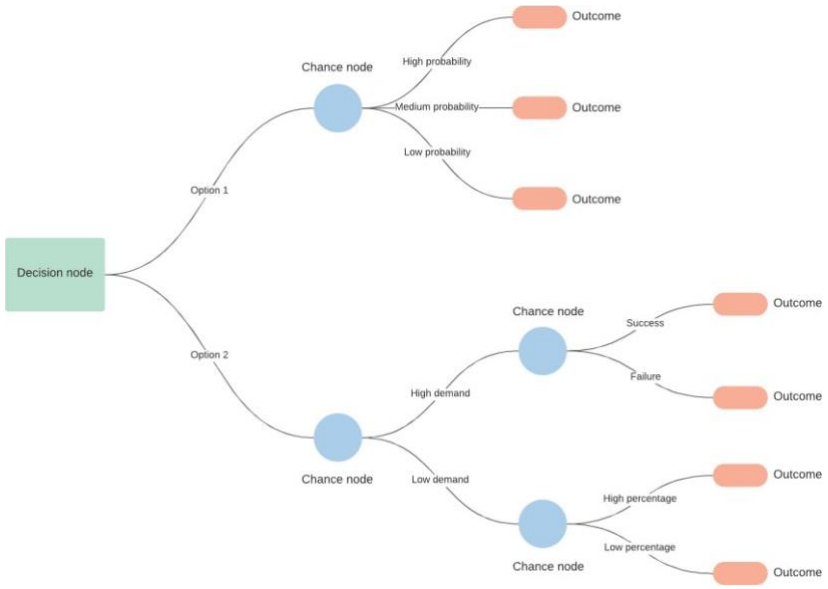


Figura 2-6 Ejemplo de un árbol de decisión (41)

Un árbol de decisión es un algoritmo que se utiliza para categorizar o hacer predicciones en función de cómo se respondió un conjunto anterior de preguntas. El modelo es una forma de aprendizaje supervisado, lo que significa que el modelo se entrena y prueba en un conjunto de datos que contiene la categorización deseada.

Es posible que el árbol de decisiones no siempre proporcione una respuesta o decisión clara. En cambio, puede presentar opciones para que el usuario pueda tomar una decisión fundamentada por su cuenta. Los árboles de decisión replican el pensamiento humano, por lo que generalmente es sencillo comprender e interpretar los resultados obtenidos.

La base del árbol es el nodo raíz (root node). Del nodo raíz derivan una serie de nodos de decisión (decision nodes), los cuales representan las decisiones que se pueden tomar. De los nodos de decisión derivan nodos de hoja (leaf nodes), los cuales representan las consecuencias de cada una de las decisiones. Cada ndo de decisión representa una pregunta o punto de división, y los nodos hoja que brotan de un nodo de decisión representan las posibles respuestas (20).

2.2.3.3 Random Forest

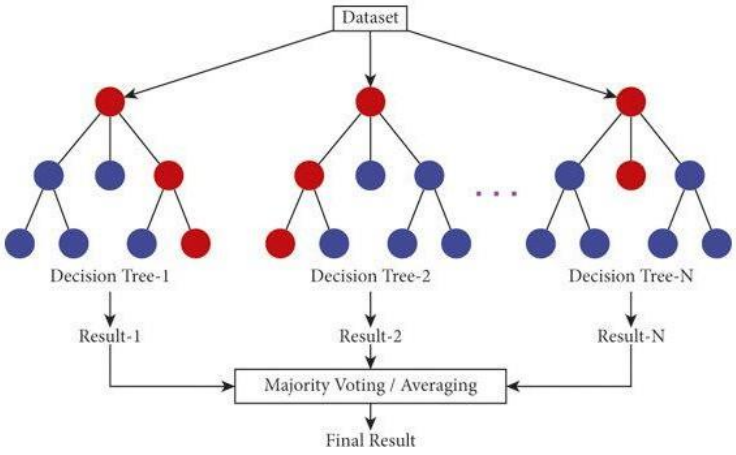


Figura 2-7 Ejemplo de un random Forest (42)

Random Forest es un algoritmo de aprendizaje automático de uso común con marca registrada de Leo Breiman y Adele Cutler, que combina la salida de múltiples árboles de decisión para llegar a un único resultado. Su fácil uso y gran flexibilidad han impulsado su implantación, aun que maneja problemas de clasificación y regresión.

Dado que el modelo de bosque aleatorio se compone de múltiples árboles de decisión, sería útil comenzar describiendo brevemente el algoritmo del árbol de decisión. Los árboles de decisión empiezan con una pregunta simple, cómo "¿Debería navegar?" A partir de ahí, puede hacer una serie de preguntas para determinar una respuesta, cómo "¿Es un oleaje de larga duración?" o "¿Está soplando el viento en alta mar?". Estas cuestiones conforman los diferentes nodos de decisión dentro del árbol y trabajan como un medio para fraccionar los datos. Cada pregunta ayuda a un individuo a llegar a una decisión final, que se denotará con el nodo hoja.

Las observaciones que se acerquen a los criterios deberán seguir la rama "Sí" y las que no, seguirán una ruta alternativa. Los árboles de decisión buscan encontrar la mejor división para dividir los datos en subconjuntos y, por lo general, se entrenan a través del algoritmo del árbol de clasificación y regresión (CART). Las métricas tales como la impureza de Gini, el enriquecimiento de información o el error cuadrático medio (MSE), podrían usarse para evaluar la calidad de la división. Este árbol de decisión es un ejemplo de un error de clasificación, donde las etiquetas de clase que aparecen son "navegar" y "no navegar".

Si bien los árboles de decisión son algoritmos de aprendizaje supervisado comunes, pueden ser propensos a problemas, como sesgos y sobreajuste. Sin embargo, cuando varios árboles de decisión forman un conjunto en el algoritmo de bosque aleatorio, predicen resultados más precisos, especialmente cuando los árboles individuales no están correlacionados entre sí (21).

Algunos de los parámetros a tener en cuenta en este modelo son:

- n_estimators: Número de árboles en el bosque.
- max_features: Número de características a tener en cuenta para encontrar la mejor división.
- max_depth: Profundidad máxima del árbol.
- min_samples_split: Número mínimo de muestras requeridas para dividir un nodo interno.

- min_samples_leaf: Número mínimo de muestras requeridas para ser una hoja del nodo.
- bootstrap: Si se deben usar muestras de bootstrap para construir árboles.

2.2.3.4 *Support Vector Machine (SVM)*

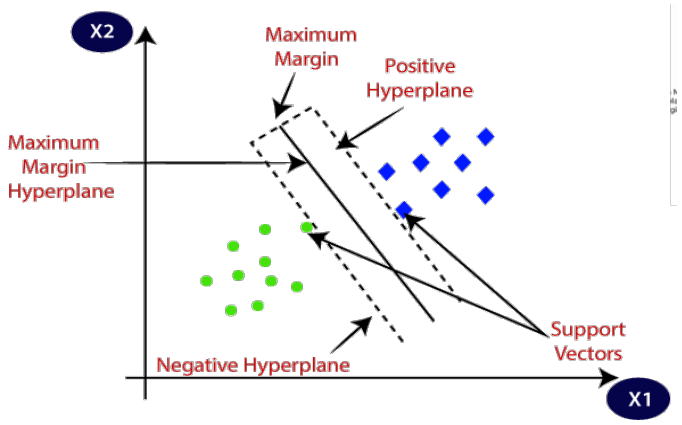


Figura 2-8 Ejemplo del SVM (43)

La máquina de vectores soporte o SVM es uno de los algoritmos de aprendizaje supervisado más populares, que se utiliza tanto para problemas de clasificación como de regresión. Sin embargo, se utiliza principalmente para problemas de clasificación en el aprendizaje automático. El objetivo del algoritmo SVM es crear la mejor línea o límite de decisión que pueda segregar el espacio n-dimensional en clases para que podamos poner fácilmente el nuevo punto de datos en la categoría correcta en el futuro. Esta mejor frontera de decisión se denomina hiperplano. SVM elige los puntos/vectores extremos que ayudan a crear el hiperplano. Estos casos extremos se denominan vectores de soporte y, por lo tanto, el algoritmo se denomina máquina de vectores de soporte. Considere el siguiente diagrama en el que hay dos categorías diferentes que se clasifican utilizando un límite de decisión o hiperplano (22).

Las Máquinas de Vectores Soporte, más conocidas por su capacidad para realizar tareas de clasificación, también pueden utilizarse para problemas de regresión, conocidos como SVR (Support Vector Regression). En SVR, el objetivo es encontrar una línea o un hiperplano que pueda predecir un valor continuo, intentando mantener tantos datos de entrenamiento como sea posible dentro de un margen de error predefinido, ε. Los datos que caen fuera de este margen son los que definen el modelo, llamados vectores de soporte. La SVR también puede manejar relaciones no lineales mediante el uso de kernels, pero la elección de los parámetros ε y C (penalización) y la escala de los datos pueden afectar significativamente los resultados. Además, SVR puede ser computacionalmente intensivo para conjuntos de datos grandes.

2.2.3.5 K-means

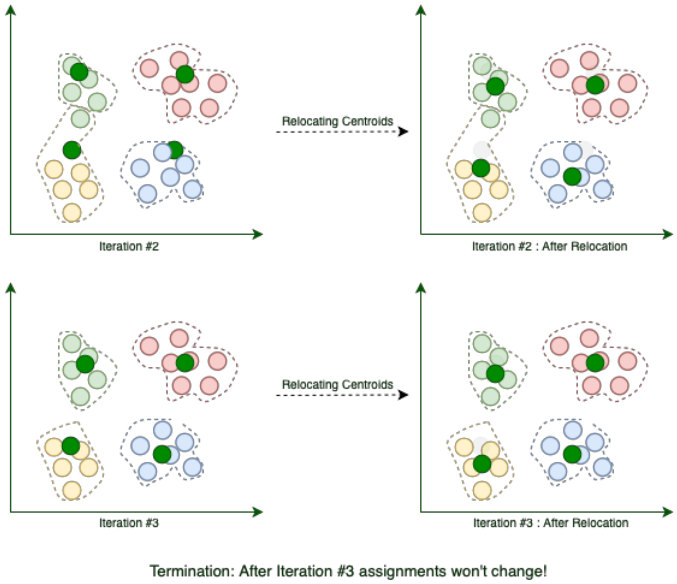


Figura 2-9 Agrupamiento K-means (44)

El algoritmo K-means explora un número planificado previamente de grupos en un conjunto de datos multidimensional sin etiquetar, lo concluye a través de una interpretación fácil de cómo se puede expresar un grupo optimizado. Principalmente, el concepto sería en dos pasos:

- En primer lugar, el centro del conglomerado es la media aritmética (AM) de todos los puntos de datos asociados con el conglomerado.
- En segundo lugar, cada punto es adjunto a su centro de agrupación en comparación con otros centros de agrupación. Estas dos interpretaciones son la base del modelo de agrupamiento de k-medias.

Puede tomar el centro como un punto de datos que describe los medios del grupo, y es posible que no sea un miembro del conjunto de datos. En términos simples, el agrupamiento de k-means nos permite agrupar los datos en varios grupos al detectar las distintas categorías de grupos en los conjuntos de datos sin etiquetar por sí mismo, incluso sin la necesidad de entrenar los datos.

Este es el algoritmo basado en el centroide, de modo que cada grupo está conectado a un centroide mientras sigue el objetivo de minimizar la suma de distancias entre los puntos de datos y sus grupos correspondientes.

Como entrada, el algoritmo consume un conjunto de datos sin etiquetar, divide el conjunto de datos completo en un número k de grupos e itera el proceso para cumplir con los grupos correctos, y el valor de k debe ser predeterminado. Realizando específicamente dos tareas, el algoritmo k-means

Calcula el valor correcto de los puntos centrales K o centroides mediante un método iterativo

Asigna cada punto de datos a su centro k más cercano, y los puntos de datos, más cerca de un centro k particular, forman un grupo. Por lo tanto, los puntos de datos, en cada grupo, tienen algunas similitudes y están muy separados de otros grupos (23).

K-means es uno de los algoritmos más utilizados debido a su eficiencia computacional, facilidad de implementación y uso, los resultados son fácilmente interpretables y se adapta a diferentes formas de datos.

2.2.3.6 Gradient Boosting

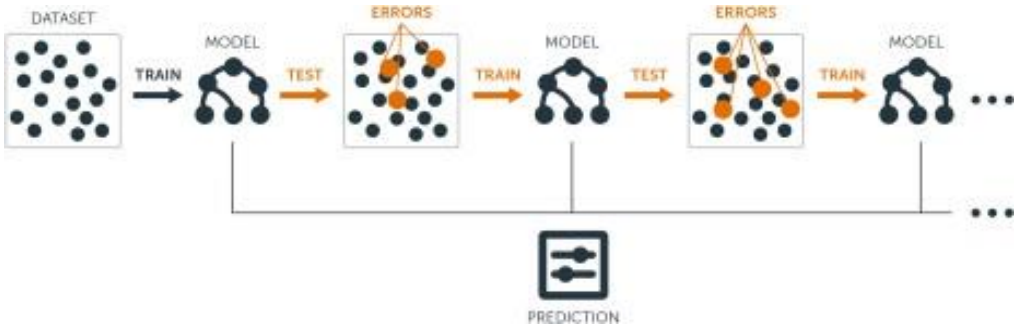


Figura 2-10 Ejemplo de 'Gradient Boosting' (45)

Gradient boosting es una técnica de aprendizaje automático para problemas de regresión y clasificación que produce un modelo de predicción en forma de conjunto de modelos de predicción débiles.

Esta técnica construye un modelo por etapas y generaliza el modelo permitiendo la optimización de una función de pérdida diferenciable arbitraria.

El Gradient boosting básicamente combina aprendices débiles en un único aprendiz fuerte de forma iterativa. A medida que se añade cada aprendiz débil, se ajusta un nuevo modelo para proporcionar una estimación más precisa de la variable de respuesta. Los nuevos aprendices débiles se correlacionan al máximo con el gradiente negativo de la función de pérdida, asociada a todo el conjunto (24).

Como todo modelo, tiene ventajas e inconvenientes a la hora de su utilización, entre las ventajas que podemos encontrar, vemos las siguientes:

- Rendimiento
- Flexibilidad
- Manejo de datos faltantes
- Interpretabilidad

En el caso de los inconvenientes, podemos observar:

- Tiempo de entrenamiento alto
- Sobreajuste
- Complejidad de la configuración de parámetros
- Sensibilidad a valores atípicos

2.2.3.7 Redes neuronales convolucionales

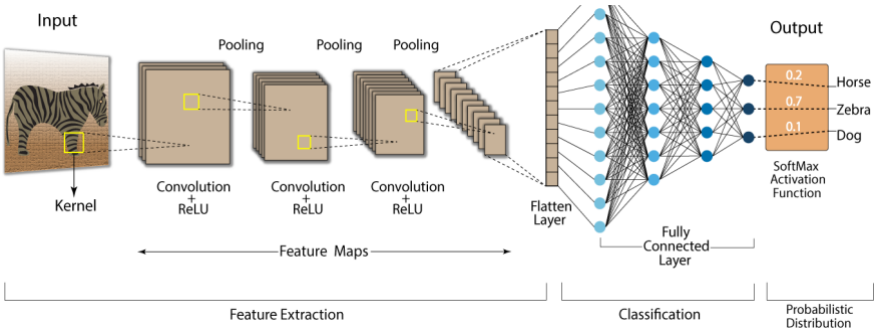


Figura 2-11 Red Neuronal Convolucional (CNN) (46)

"Las redes neuronales son un subconjunto del aprendizaje automático y están en el corazón de los algoritmos de aprendizaje profundo. Se componen de capas de nodos, que contienen una capa de entrada, una o más capas ocultas y una capa de salida. Cada nodo se conecta a otro y tiene un peso y un umbral asociados. Si la salida de cualquier nodo individual está por encima del valor de umbral especificado, ese nodo se activa y envía datos a la siguiente capa de la red. De lo contrario, no se pasan datos a la siguiente capa de la red.

Las redes neuronales recurrentes se utilizan comúnmente para el procesamiento del lenguaje natural y el reconocimiento de voz, mientras que las redes neuronales convolucionales (ConvNets o CNN) se utilizan con mayor frecuencia para tareas de clasificación y visión artificial. Antes de las CNN, se usaban métodos manuales de extracción de características que consumían mucho tiempo para identificar objetos en las imágenes. Sin embargo, las redes neuronales convolucionales ahora brindan un enfoque más escalable para la clasificación de imágenes y las tareas de reconocimiento de objetos, aprovechando los principios del álgebra lineal, específicamente la multiplicación de matrices, para identificar patrones dentro de una imagen. Dicho esto, pueden ser computacionalmente exigentes y requieren unidades de procesamiento gráfico (GPU) para entrenar modelos" (25).

2.2.3.8 Redes neuronales Long Short-Term Memory (LSTM)

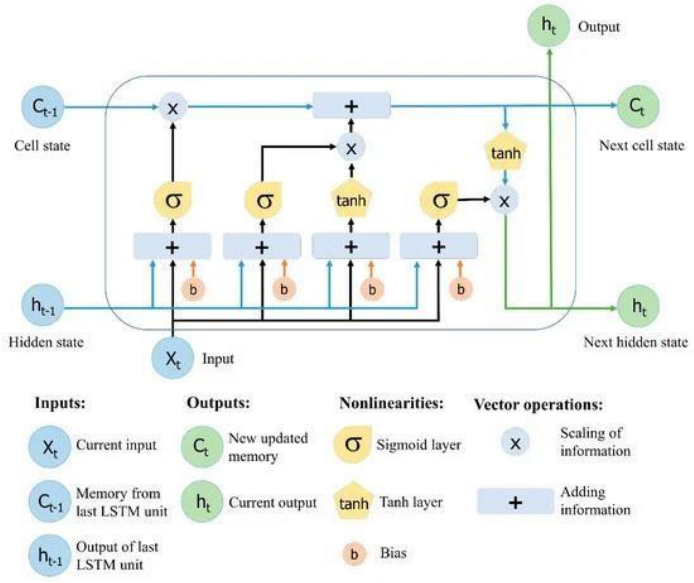


Figura 2-12 Red neuronal Long Short-Term Memory (40)

Las unidades o las agrupaciones de memoria a corto plazo (LSTM) se consideran parte de una estructura de red neuronal recurrente. Las redes neuronales recurrentes están constituidas para usar determinados tipos de procesos de memoria artificial que pudiesen ayudar a diversos programas de inteligencia artificial a reproducir de manera más efectiva el pensamiento humano

La red neuronal recurrente usa agrupaciones de memoria a corto plazo a fin de proporcionar contexto sobre la manera en que el programa toma entradas y crea salidas. La gran agrupación de memoria a corto plazo es una entidad compleja con diversos componentes, como entradas ponderadas, tareas de activación, acceso de bloques anteriores y salidas casuales.

La unidad se denomina agrupación de memoria a corto plazo porque el programa utiliza una estructura basada en procesos de memoria a corto plazo para crear memoria a largo plazo. Estos procedimientos se utilizan con frecuencia, por ejemplo, en el proceso del lenguaje natural. La red neuronal recurrente usa los bloques de memoria a breve plazo para coger una palabra o fonema singularmente y evaluarlo en el entorno de otros dentro de una cadena, donde la memoria puede ser conveniente para clasificar y categorizar este tipo de entradas. Generalmente, LSTM es una idea aceptada y común en redes neuronales recurrentes pioneras (26).

Arquitectura del modelo:

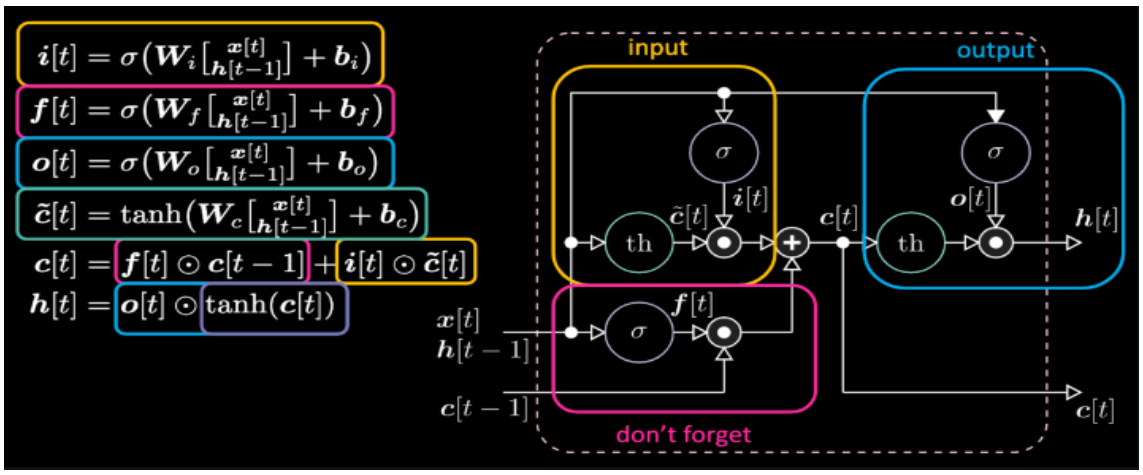


Figura 2-13 Arquitectura red LSTM

"A continuación hay ecuaciones que expresan una LSTM. La compuerta de entrada se resalta con cuadros amarillos, que será una transformación afín. Esta transformación de entrada multiplicará que es nuestra compuerta candidata.

La compuerta "*no-olvidar*" *f* (*don't forget*) está multiplicando el valor anterior de la celda de memoria *c* [*t*−1]. El valor total de la celda *c*[*t*] es el de la compuerta *no-olvidar* más el de la compuerta de entrada. La representación oculta final es la multiplicación elemento a elemento entre la compuerta de salida *o*[*t*] y la tangente hiperbólica aplicada a la celda *c*[*t*], de modo que los límites superior e inferior estén acotados. Finalmente, la compuerta candidata *c*~[*t*] es simplemente una red recurrente. Entonces tenemos una *o*[*t*] para modular la salida, una *f*[*t*] para modular la compuerta de no olvidar y una *i*[*t*] para modular la compuerta de entrada. Todas estas interacciones entre la memoria y las compuertas son interacciones multiplicativas. *i*[*t*], *f*[*t*] y *o*[*t*] son todas funciones sigmoides, que van de cero a uno. Por lo tanto, cuando se multiplica por cero, se tiene una compuerta cerrada. Al multiplicar por uno, se obtiene una compuerta abierta." (27)

Algunos de los parámetros a tener en cuenta para este modelo en problemas de regresión son:

- Número de capas
- Número de neuronas en cada capa
- Función de activación
- Optimizador
- Función de pérdida
- Épocas de entrenamiento
- Batch size o número de muestras de datos a propagar a través de la red
- Tamaño de la secuencia de entrada

2.3 Herramientas utilizadas

En este subcapítulo se definen las herramientas de las que se ha hecho uso para el desarrollo del proyecto, tanto el software como el lenguaje.

2.3.1 Anaconda

Anaconda es una distribución de código abierto de los lenguajes de programación Python y R para la ciencia de datos que pretende simplificar la gestión y el despliegue de paquetes. Las versiones de los paquetes en Anaconda son gestionadas por el sistema de gestión de paquetes, conda, que analiza el entorno actual antes de ejecutar una instalación para evitar interrumpir otros marcos y paquetes.

La distribución de Anaconda viene con más de 250 paquetes instalados automáticamente. Se pueden instalar más de 7500 paquetes adicionales de código abierto desde PyPI, así como el gestor de paquetes y entornos virtuales conda. También incluye una GUI (interfaz gráfica de usuario), Anaconda Navigator, como alternativa gráfica a la interfaz de línea de comandos. Anaconda Navigator está incluido en la distribución de Anaconda, y permite a los usuarios lanzar aplicaciones y gestionar paquetes, entornos y canales conda sin utilizar comandos de línea de comandos. Navigator puede buscar paquetes, instalarlos en un entorno, ejecutar los paquetes y actualizarlos (28).

2.3.2 Jupyter Notebook

Jupyter Notebook es una aplicación web de código abierto que puedes utilizar para crear y compartir documentos que contengan código en vivo, ecuaciones, visualizaciones y texto.

Jupyter Notebook es un proyecto derivado del proyecto IPython, que a su vez tenía un proyecto IPython Notebook. El nombre, Jupyter, viene del núcleo de lenguajes de programación que soporta: Julia, Python y R. Jupyter viene con el núcleo IPython, que te permite escribir tus programas en Python, pero actualmente hay más de 100 núcleos más que también puedes utilizar (29).

2.3.3 Python

“Python es un lenguaje de código abierto, interpretado, de alto nivel y proporciona un gran enfoque para la programación orientada a objetos. Es uno de los mejores lenguajes utilizados por los científicos de datos para diversos proyectos/aplicaciones de ciencia de datos. Python proporciona una gran funcionalidad para hacer frente a las matemáticas, la estadística y la

función científica. Proporciona grandes bibliotecas para hacer frente a la aplicación de la ciencia de datos.

Una de las principales razones por las que Python se utiliza ampliamente en las comunidades científicas y de investigación es su facilidad de uso y su sintaxis sencilla. También es más adecuado para la creación rápida de prototipos.

En términos de áreas de aplicación, los científicos de ML también prefieren Python. Cuando se trata de áreas como la creación de algoritmos de detección de fraudes y la seguridad de la red, los desarrolladores se inclinan por Java, mientras que para aplicaciones como el procesamiento del lenguaje natural (NLP) y el análisis de sentimientos, los desarrolladores optan por Python, ya que proporciona una gran colección de librerías que ayudan a resolver problemas empresariales complejos con facilidad, construir sistemas sólidos y aplicaciones de datos" (30).

2.3.3.1 Librerías utilizadas

Las librerías en Python son a grandes rasgos un conjunto de funcionalidades que dan la posibilidad al usuario de desarrollar nuevas tareas que antes no se podían realizar. Responden al conjunto de implementaciones que posibilitan codificar este lenguaje, con el objetivo de crear una interfaz independiente. Las librerías utilizadas en este proyecto son:

2.3.3.1.1 Pandas

Es una librería de código abierto de Python que proporciona herramientas de análisis y manipulación de datos de alto rendimiento utilizando sus potentes estructuras de datos. Sus principales características son:

- Posibilita lectura y escritura de manera sencilla de ficheros con formato Excel, CSV y bases de datos SQL, entre otros
- Permite acceder a datos a través de nombres o índices para columnas y filas
- Permite trabajar con series temporales
- Ofrece métodos para dividir, combinar y reordenar conjuntos de datos. (31)

2.3.3.1.2 Numpy

Es una librería de Python que proporciona un objeto array multidimensional, varios objetos derivados (como arrays enmascarados y matrices), y un surtido de rutinas para operaciones rápidas sobre arrays, incluyendo operaciones matemáticas, lógicas, manipulación de formas, ordenación, selección, E/S, transformadas discretas de Fourier, álgebra lineal básica, operaciones estadísticas básicas, simulación aleatoria y mucho más (32).

2.3.3.1.3 Random

La librería Random es un módulo integrado para generar variables pseudoaleatorias. Se puede utilizar para realizar alguna acción aleatoriamente, como obtener un número aleatorio, seleccionar elementos aleatorios de una lista, barajar elementos aleatoriamente, etc (33).

2.3.3.1.4 Matplotlib

Matplotlib es una librería de trazado 2D en Python que produce figuras con calidad de publicación en diversos formatos impresos y entornos interactivos en todas las plataformas (34).

2.3.3.1.5 Seaborn

Seaborn es una librería de visualización de datos de Python basada en matplotlib. Proporciona una interfaz de alto nivel para dibujar gráficos estadísticos atractivos e informativos (35).

2.3.3.1.6 Scikit-learn

Scikit-learn (Sklearn) es la librería más útil y robusta para el aprendizaje automático en Python. Proporciona una selección de herramientas eficientes para el aprendizaje automático y el modelado estadístico, incluidas la clasificación, la regresión, la agrupación y la reducción dimensional a través de una interfaz coherente en Python (36).

2.3.3.1.7 Statsmodel

Statsmodels es un módulo de Python que proporciona clases y funciones para la estimación de muchos modelos estadísticos diferentes, así como para la realización de pruebas estadísticas, y la exploración de datos estadísticos (37).

2.3.3.1.8 Pmdarima

Pmdarima es una librería estadística diseñada para llenar el vacío en las capacidades de análisis de series temporales de Python. Incluye:

- El equivalente de la funcionalidad auto.arima de R
- Una colección de pruebas estadísticas de estacionalidad
- Utilidades de series temporales, como diferenciación y diferenciación inversa
- Numerosos transformadores y caracterizadoras endógenas y exógenas, incluyendo transformaciones Box-Cox y Fourier
- Descomposiciones estacionales de series temporales (38)

2.3.3.1.9 Keras

"Keras es una biblioteca de código abierto (con licencia MIT) escrita en Python, que se basa principalmente en el trabajo de François Chollet, un desarrollador de Google, en el marco del proyecto ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System). La primera versión de este software multiplataforma se lanzó el 28 de marzo de 2015. El objetivo de la biblioteca es acelerar la creación de redes neuronales: para ello, Keras no funciona como un framework independiente, sino como una interfaz de uso intuitivo (API) que permite acceder a varios frameworks de aprendizaje automático y desarrollarlos. Entre los frameworks compatibles con Keras, se incluyen Theano, Microsoft Cognitive Toolkit (anteriormente CNTK) y TensorFlow" (39).

3. Metodología

3.1 Introducción

En este capítulo se detallará el proceso que se ha sido seguido con el fin de crear un modelo predictivo del consumo de energía en los hogares londinenses. Como previamente hemos comentado, para la implementación del modelo predictivo se ha hecho uso de un dataset que proporciona Kaggle en el que podemos encontrar información acerca del consumo energético de ciertos hogares que han participado en un proyecto liderado por UK Power Networks, conocido como Low Carbon London entre noviembre de 2011 y febrero de 2014.

El dataset está formado por varios archivos con formato CSV, entre los que se han seleccionado el "daily_dataset.csv", "weather_daily_darksky.csv" y "uk_bank_holidays.csv".

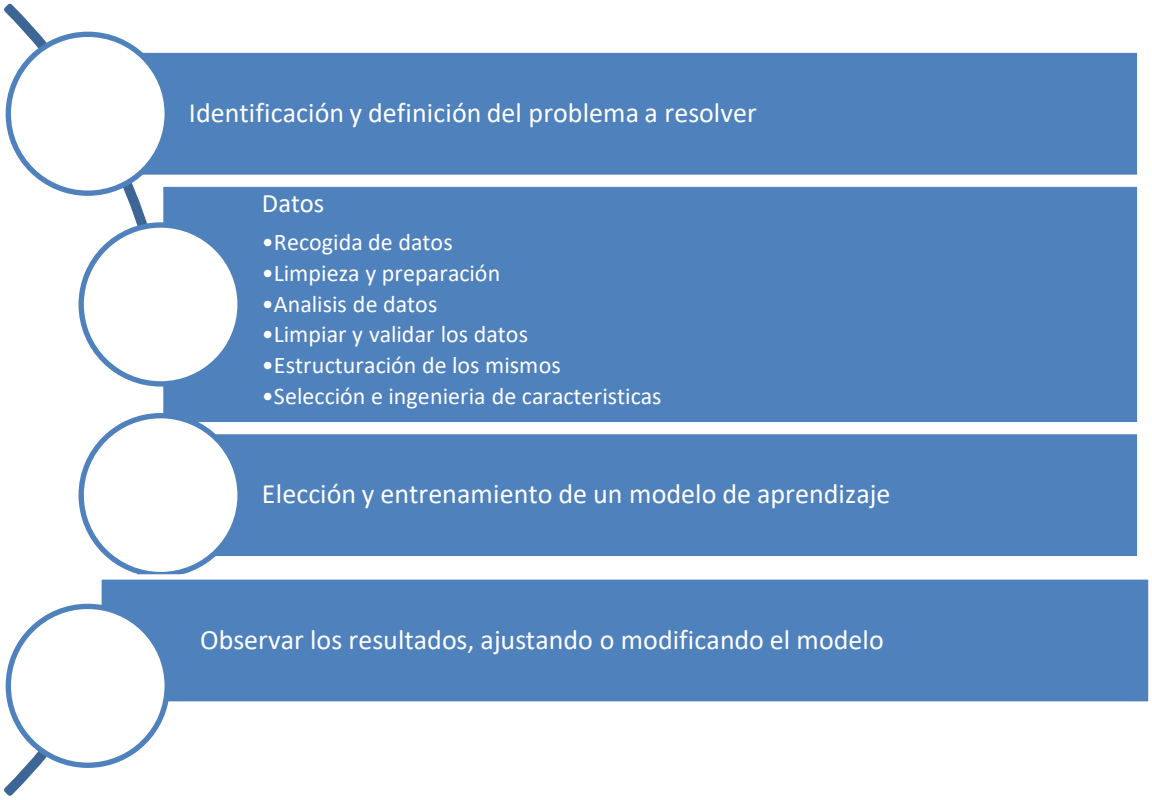
Dentro del archivo "daily_dataset.csv" se encuentra información acerca de los hogares, los cuales vienen identificados por la variable LCLid, el día en el que se han registrado los valores de consumo por los contadores inteligentes y una serie de variables estadísticas acerca del consumo, como el consumo medio o el consumo total del día.

Dentro del archivo "weather_daily_darksky.csv" se encuentra información de carácter meteorológico obtenido a través del api de darksky. En el podemos encontrar información como la temperatura máxima en el día, la temperatura mínima o porcentaje de humedad.

Por último, dentro del archivo "uk_bank_holidays.csv" se encuentra información acerca de los días festivos en Londres.

Con toda esta información, se busca implementar un modelo que nos permita pronosticar el consumo total diario del conjunto de hogares que participan en el proyecto. Para ello, se implementarán diferentes tipos de modelos con el fin de compararlos entre sí y ver cuál es el que ofrece mejores resultados para el conjunto de datos del que disponemos.

A continuación, se presenta un esquema de los pasos que hemos seguido para la obtención de los resultados:



3.2 Preparación de los datos

Una vez que ya tenemos el dataset a nuestra disposición, lo primero que hay que realizar es una preparación o preprocesado de los datos para poder construir nuestros modelos ML.

Lo primero que se hace es cargar los datos en un dataframe a través de la librería pandas. Una vez que tenemos nuestro dataframe cargado con los datos, pasamos a realizar una parte sumamente importante de la preparación de los datos, que es el análisis exploratorio de datos (EDA).

Un primer paso del análisis exploratorio de datos es ver qué tipos de datos tenemos, para ello, hacemos uso de 2 métodos que facilita la librería pandas como el .info() y el .head().

```
In [2]: df_daily = pd.read_csv("daily_dataset.csv")
        df_daily.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3510433 entries, 0 to 3510432
Data columns (total 9 columns):
 #   Column         Dtype
---  ------         -----
 0   LCLid          object
 1   day            object
 2   energy_median  float64
 3   energy_mean    float64
 4   energy_max     float64
 5   energy_count   int64
 6   energy_std     float64
 7   energy_sum     float64
 8   energy_min     float64
dtypes: float64(6), int64(1), object(2)
memory usage: 241.0+ MB
```

```
In [3]: df_daily.head()
```

Out[3]:

	LCLid	day	energy_median	energy_mean	energy_max	energy_count	energy_std	energy_sum	energy_min
0	MAC000131	2011-12-15	0.4850	0.432045	0.868	22	0.239146	9.505	0.072
1	MAC000131	2011-12-16	0.1415	0.296167	1.116	48	0.281471	14.216	0.031
2	MAC000131	2011-12-17	0.1015	0.189812	0.685	48	0.188405	9.111	0.064
3	MAC000131	2011-12-18	0.1140	0.218979	0.676	48	0.202919	10.511	0.065
4	MAC000131	2011-12-19	0.1910	0.325979	0.788	48	0.259205	15.647	0.066

Figura 3-1 Visualización de los datos df_daily

Con esto, obtenemos una visión general acerca de nuestros datos, en la que observamos que el dataframe está compuesto por 9 columnas y 3510433 de instancias o filas. Como hemos comentado previamente, en la columna LCLid se encuentra el identificador de cada hogar, en la columna day se encuentran los días en los que se han registrado los datos y el resto de columnas son estadísticos acerca del consumo energético del hogar. Otra parte importante a destacar del método .info() es que nos indica el tipo de dato que tenemos en cada columna, en este caso obtenemos que tanto las columnas LCLid y day son de tipo object, mientras que el resto de columnas son de carácter numérico.

Lo siguiente que hacemos es comprobar el número de hogares independientes que tenemos y si todos a todos se les instalo su contador inteligente al principio del proyecto:

```
In [5]: len(df_daily.LCLid.unique())
```

```
Out[5]: 5566
```

Probablemente no se pusieron todos los medidores el mismo día, sino que se fue realizando de manera gradual.

```
In [7]: house_cont = df_daily.groupby('day')[['LCLid']].nunique()
        house_cont.plot(figsize=(25,7))
```

```
Out[7]: <AxesSubplot:xlabel='day'>
```

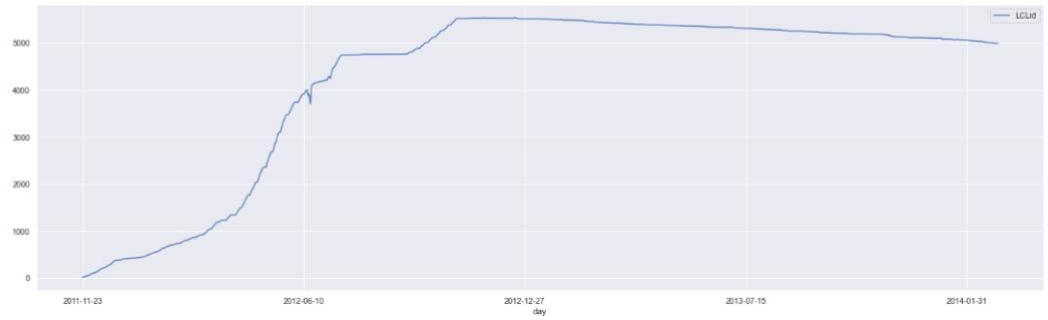


Figura 3-2 Gráfico del aumento de contadores inteligentes

Como se puede observar en la gráfica, el proyecto no comenzó con los 5566 hogares, sino que estos se fueron incrementando gradualmente. Por lo tanto, como no es homogéneo,

hacemos ingeniería de características definiendo una nueva variable, que será la energía media consumida por usuario.

Para ello, de nuestro dataframe nos quedamos con las columnas que contienen el identificador del hogar, el día y el total de energía consumida por el hogar. Con el objetivo de obtener la energía media consumida por usuario, agrupamos en base al día, el número de hogares de los que disponemos datos y su energía total consumida. Con esto, reducimos el dataframe a 829 filas (una por día) y así el coste computacional sin perder información.

```
In [9]: df_daily = df_daily.groupby('day')[['energy_sum']].sum()
        df_daily = df_daily.merge(house_cont, on = ['day'])
        df_daily = df_daily.reset_index()

In [10]: df_daily
```

Out[10]:

	day	energy_sum	LCLid
0	2011-11-23	90.385000	13
1	2011-11-24	213.412000	25
2	2011-11-25	303.993000	32
3	2011-11-26	420.976000	41
4	2011-11-27	444.883001	41
...	...	...	...
824	2014-02-24	51994.547004	4994
825	2014-02-25	51423.508001	4995
826	2014-02-26	50943.305995	4993
827	2014-02-27	51678.185998	4990
828	2014-02-28	1042.266000	4987

829 rows × 3 columns

Figura 3-3 Agrupamiento de hogares en base al día

Procedemos a transformar nuestra variable day a tipo fecha, lo que nos facilitará el trabajo a la hora de trabajar con la serie temporal, y a obtener la energía media consumida por hogar, que será la variable que buscaremos pronosticar con nuestros modelos, la cual obtenemos dividiendo la energía total consumida entre el número total de hogares independientes:

```
df_daily.day = pd.to_datetime(df_daily.day,format='%Y-%m-%d').dt.date
df_daily['avg_energy'] =  df_daily['energy_sum']/df_daily['LCLid']

df_daily
```

	day	energy_sum	LCLid	avg_energy
0	2011-11-23	90.385000	13	6.952692
1	2011-11-24	213.412000	25	8.536480
2	2011-11-25	303.993000	32	9.499781
3	2011-11-26	420.976000	41	10.267707
4	2011-11-27	444.883001	41	10.850805
...	...	...	...	...
824	2014-02-24	51994.547004	4994	10.411403
825	2014-02-25	51423.508001	4995	10.294997
826	2014-02-26	50943.305995	4993	10.202945
827	2014-02-27	51678.185998	4990	10.356350
828	2014-02-28	1042.266000	4987	0.208997

829 rows × 4 columns

Figura 3-4 Cálculo de consumo de energía media por hogar

De esta forma, obtenemos un dataframe con el día en el que se recogen los datos, la energía total consumida por los hogares, el número de hogares independientes y la energía media consumida por cada hogar. Para continuar con el análisis exploratorio de los datos, visualizamos la energía consumida media por hogar a lo largo de la serie temporal, lo que dará una idea de la distribución de nuestros datos y ver si existe estacionalidad en la serie.

```
df_daily.plot(x="day",y="avg_energy", figsize=(25,7))
```

```
<AxesSubplot:xlabel='day'>
```

Figura 3-5 Gráfico del consumo energético medio por hogar

Procedemos a cargar el archivo "weather_daily_darksky.csv", el cual nos proporciona información meteorológica de Londres en los días en los que se ha realizado el proyecto. Realizamos los mismos pasos que con el archivo "daily_dataset.csv" para visualizar la información que contiene el archivo:

```
df_weather = pd.read_csv("weather_daily_darksky.csv")
df_weather.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 882 entries, 0 to 881
Data columns (total 32 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   temperatureMax              882 non-null    float64
 1   temperatureMaxTime          882 non-null    object
 2   windBearing                 882 non-null    int64
 3   icon                        882 non-null    object
 4   dewPoint                    882 non-null    float64
 5   temperatureMinTime          882 non-null    object
 6   cloudCover                  881 non-null    float64
 7   windSpeed                   882 non-null    float64
 8   pressure                    882 non-null    float64
 9   apparentTemperatureMinTime  882 non-null    object
 10  apparentTemperatureHigh     882 non-null    float64
 11  precipType                  882 non-null    object
 12  visibility                  882 non-null    float64
 13  humidity                    882 non-null    float64
 14  apparentTemperatureHighTime 882 non-null    object
 15  apparentTemperatureLow      882 non-null    float64
 16  apparentTemperatureMax      882 non-null    float64
 17  uvIndex                     881 non-null    float64
 18  time                        882 non-null    object
 19  sunsetTime                  882 non-null    object
 20  temperatureLow              882 non-null    float64
 21  temperatureMin              882 non-null    float64
 22  temperatureHigh             882 non-null    float64
 23  sunriseTime                 882 non-null    object
 24  temperatureHighTime         882 non-null    object
 25  uvIndexTime                 881 non-null    object
 26  summary                     882 non-null    object
 27  temperatureLowTime          882 non-null    object
 28  apparentTemperatureMin      882 non-null    float64
 29  apparentTemperatureMaxTime  882 non-null    object
 30  apparentTemperatureLowTime  882 non-null    object
 31  moonPhase                   882 non-null    float64
dtypes: float64(16), int64(1), object(15)
memory usage: 220.6+ KB
```

Figura 3-6 Visualización de los datos de df_weather

Nos encontramos con un dataframe que contiene todo tipo de información meteorológica y que está compuesto por 882 filas y 32 columnas. Dentro de la columna time, se encuentra el día en el que se recogieron los datos, por lo tanto, la procesamos para cambiar el tipo de dato a fecha y almacenarlos en otra columna a la que llamamos day, con el objetivo de concatenar este dataframe con el que teníamos previamente con la información acerca del consumo.

Finalmente, obtenemos un dataframe con los datos de consumo y los datos de meteorología recogidos en cada día:

```
df_energy_weather = df_daily.merge(df_weather, on="day")
df_energy_weather.head()
```

	day	energy_sum	LCLid	avg_energy	temperatureMax	temperatureMaxTime	windBearing	icon	dewPoint	temperatureMinTime	...	temperatureHigh	sur
0	2011-11-23	90.385000	13	6.952692	10.36	2011-11-23 14:00:00	229	fog	6.29	2011-11-23 07:00:00	...	10.36	2(
1	2011-11-24	213.412000	25	8.536480	12.93	2011-11-24 12:00:00	204	partly-cloudy-night	8.56	2011-11-24 02:00:00	...	12.93	2(
2	2011-11-25	303.993000	32	9.499781	13.03	2011-11-25 05:00:00	243	partly-cloudy-day	7.24	2011-11-25 23:00:00	...	12.27	2(
3	2011-11-26	420.976000	41	10.267707	12.96	2011-11-26 14:00:00	237	wind	6.96	2011-11-26 01:00:00	...	12.96	2(
4	2011-11-27	444.883001	41	10.850805	13.54	2011-11-27 10:00:00	256	wind	5.76	2011-11-27 23:00:00	...	13.54	2(

5 rows × 35 columns

Figura 3-7 df_daily y df_weather concatenados

Dentro de este dataframe, observamos que las columnas icon y precipType son de carácter categórico, por lo tanto damos uso de la librería scikit-learn para importar la función LabelEncoder(), que nos permite codificar estas variables en valores numéricos entre 0 y el número de categorías menos 1 (51).

```
df_energy_weather['icon'] = LabelEncoder().fit_transform(df_energy_weather['icon'])
df_energy_weather['precipType'] = LabelEncoder().fit_transform(df_energy_weather['precipType'])
df_energy_weather.head(4)
```

	day	energy_sum	LCLid	avg_energy	temperatureMax	temperatureMaxTime	windBearing	icon	dewPoint	temperatureMinTime	...	temperatureHigh	sunris
0	2011-11-23	90.385	13	6.952692	10.36	2011-11-23 14:00:00	229	2	6.29	2011-11-23 07:00:00	...	10.36	201 0(
1	2011-11-24	213.412	25	8.536480	12.93	2011-11-24 12:00:00	204	4	8.56	2011-11-24 02:00:00	...	12.93	201 0(
2	2011-11-25	303.993	32	9.499781	13.03	2011-11-25 05:00:00	243	3	7.24	2011-11-25 23:00:00	...	12.27	201 0(
3	2011-11-26	420.976	41	10.267707	12.96	2011-11-26 14:00:00	237	5	6.96	2011-11-26 01:00:00	...	12.96	201 0(

4 rows × 36 columns

Figura 3-8 Preprocesado de variables icon y precipType con LabelEncoder

Para continuar con nuestro análisis exploratorio de los datos, realizamos una serie de visualizaciones acerca de ciertas variables meteorológicas que pueden estar relacionadas con el consumo energético en los hogares:

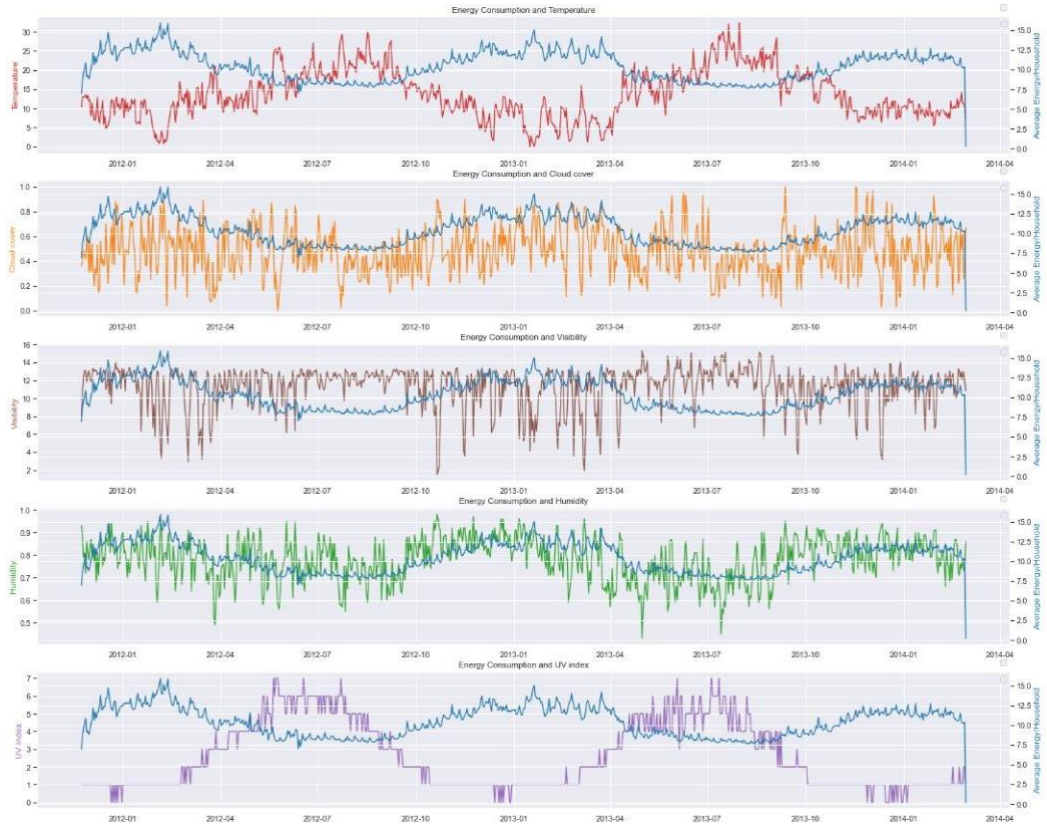


Figura 3-9 Gráficos de correlación de consumo medio por hogar y variables meteorológicas

En estos gráficos, podemos observar cierta correlación entre el consumo enérgico en los hogares y las distintas variables meteorológicas. Se aprecia que cuanto menor es la temperatura y el índice ultravioleta, se produce un mayor consumo energético, por lo tanto, podemos decir que estas variables tienen una correlación inversa. En el caso de la humedad, observamos que cuanta más alta es la humedad, se produce mayor consumo energético, por lo tanto, estas variables tienen correlación directa.

Realizamos también una matriz de correlación para observar de manera más numérica los valores de correlación entre variables:

```
plt.figure(figsize=(16, 6))
cor_matrix = sns.heatmap(df_energy_weather[['avg_energy','temperatureMax','dewPoint', 'cloudCover', 'windSpeed','pressure',
```

Figura 3-10 Matriz de correlación de variables

Para estimar una correlación alta entre variables, establecemos que los valores deben ser inferiores a -0,75 para correlación inversa y superiores a 0,75 para una correlación directa.

Por último, procedemos a cargar el archivo "uk_bank_holidays.csv" que contiene información acerca de los días festivos en la ciudad londinense. Realizamos el mismo método que con los otros 2 archivos, damos uso del método .info() para ver el contenido del dataframe:

```
df_holliday = pd.read_csv("uk_bank_holidays.csv")
df_holliday.info()
df_holliday.rename(columns={"Bank holidays":"day"})
df_holliday.columns = ["day","Type"]
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25 entries, 0 to 24
Data columns (total 2 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Bank holidays  25 non-null     object
 1   Type           25 non-null     object
dtypes: object(2)
memory usage: 528.0+ bytes
```

Figura 3-11 Visualización de datos de df_holiday

Observamos que está compuesto por 25 filas y 2 columnas. En la columna Bank holidays, se encuentra información acerca de la fecha en la que está establecido el festivo, mientras en la columna Type, se encuentra el nombre de la festividad.

Como apreciamos en la figura, la columna Bank holidays, la renombramos como day para darle un nombre que proporcione más información y describa mejor la información que contiene. Acto seguido, procedemos a cambiar el tipo de la columna day a tipo fecha, como hemos hecho con los anteriores dataframes, y a concatenar, en base a la columna day, este dataframe con el df_energy_weather, en el que almacenamos toda la información relacionada con el consumo y la meteorología.

Al concatenar los 2 dataframes, la columna Type solo contiene información en los días festivos, mientras que en el resto de la columna se almacena un Not a Number (NaN). Para solucionar esto, creamos una nueva columna a la que llamaremos Isholiday, en la que almacenaremos un 0 en los días que no hay festividad y un 1 en los días en los que sí existe festividad. Para ello, hacemos uso de la librería Numpy, la cual nos permite identificar las filas

en las que se encuentra un NaN y rellenarlas con el valor que le indiquemos, mientras que en las que identifique un valor, lo substituya por un 1.

```
df_holliday['day'] = pd.to_datetime(df_holliday['day'],format='%Y-%m-%d').dt.date
df_energy_weather_holliday = df_energy_weather.merge(df_holliday, left_on = 'day',right_on = 'day',how = 'left')
df_energy_weather_holliday['Isholiday'] = np.where(df_energy_weather_holliday['Type'].isna(),0,1)
df_energy_weather_holliday = df_energy_weather_holliday.drop(["Type"], axis=1)
```

```
df_energy_weather_holliday.Isholiday.unique()
```

```
array([0, 1])
```

Figura 3-12 Generación de columna binaria Isholiday

En base a la información que tenemos, se nos presenta la cuestión de si en los días festivos, el consumo energético de los hogares puede aumentar debido a que son días en los que se suele estar en casa. Para responderla, realizamos un gráfico de puntos, en los que destacamos los días festivos para ver si el consumo aumenta:

```
plt.figure(figsize=(25, 10))
sns.scatterplot(data=df_energy_weather_holliday, x="day",y="avg_energy", hue="Isholiday")
```

```
<AxesSubplot:xlabel='day', ylabel='avg_energy'>
```

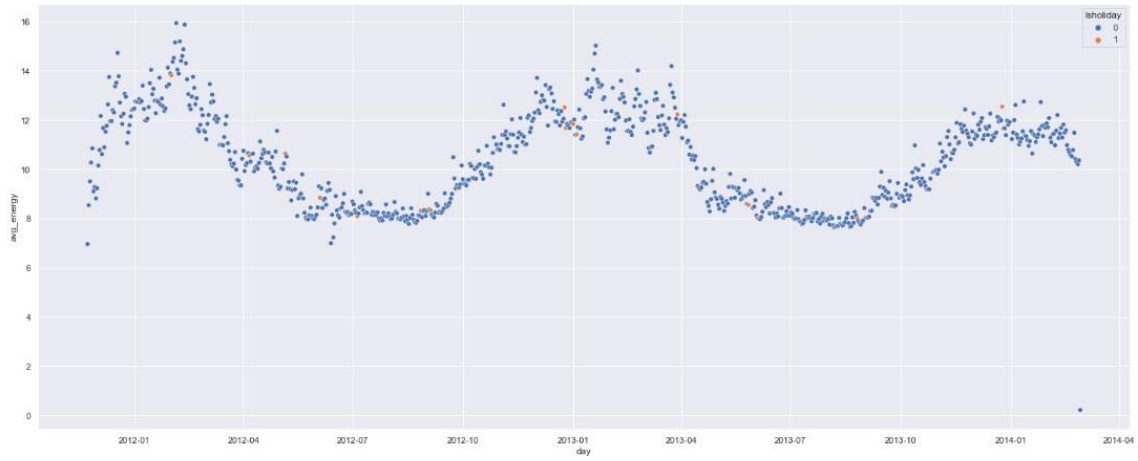


Figura 3-13 Gráfico de consumo energético medio por hogar en días festivos

Como podemos observar en la figura, el consumo en los días festivos frente al resto de días se mantiene en la línea de consumo general.

Una vez que ya tenemos toda la información preprocesada y analizada de los 3 archivos utilizados en un único dataframe llamada df_energy_weather_holiday, procedemos a implementar el algoritmo de aprendizaje no supervisado conocido como K-means.

Antes de pasarle nuestros datos al algoritmo K-means, realizamos un escalado de los datos de carácter numérico. Para ello, damos uso del método MinMaxScaler() de la librería scikit-learn, el cual nos permite darle la misma escala a todos los datos en base a los máximos y los mínimos de cada variable, siendo 1 el máximo y 0 el mínimo. Este método se aplica, como previamente hemos comentado, a las variables de carácter numérico que contienen información meteorológica, ya que los grupos o clusters que buscamos crear con K-means son en base a la meteorología.

```
scaler = MinMaxScaler()
weather_scaled = scaler.fit_transform(df_energy_weather_holliday[['temperatureMax','humidity','windSpeed', 'windBearing', 'icon',
                                                                  'cloudCover', 'pressure','apparentTemperatureHigh', 'precipType'
                                                                  'apparentTemperatureLow', 'apparentTemperatureMax', 'uvIndex', '
                                                                  'temperatureLow', 'temperatureMin', 'temperatureHigh','apparentT
```

Figura 3-14 Escalado de datos con MinMaxScaler

Con nuestros datos ya escalados, procedemos a pasárselos al algoritmo k-means. Lo que buscamos al utilizar K-means, es principalmente reducir la dimensionalidad y complejidad de nuestro dataset, sin perder toda la información de las columnas.

Para utilizar dicho algoritmo, es conveniente utilizar algún método para determinar el número óptimo de clusters en los que se van a dividir nuestros datos. En este caso, se utiliza un método conocido como Elbow Method (Método de codo), el cual utiliza la distancia media entre las observaciones y su centroide.

```
kmeans = [KMeans(n_clusters=i) for i in range(1, 15)]
score = [kmeans[i].fit(weather_scaled).score(weather_scaled) for i in range(len(kmeans))]
plt.figure(figsize=(25, 10))
plt.plot(range(1, 15),score)
plt.xlabel('Number of Clusters')
plt.ylabel('Score')
plt.title('Elbow Curve')
plt.show()
```

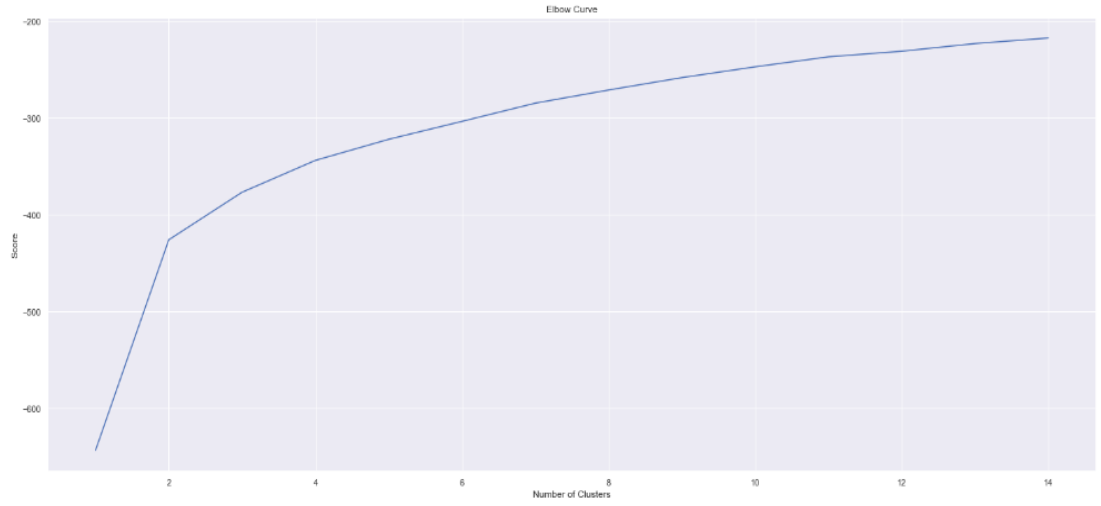


Figura 3-16 Utilización del Elbow Method

En este caso, basándonos en el gráfico que nos proporciona el método de codo y conociendo la naturaleza de nuestros datos, lo cual es una parte muy importante a la hora de la toma de decisiones, decidimos tomar 4 clusters teniendo en cuenta las 4 estaciones del año.

Acto seguido, aplicamos el algoritmo K-Means sobre nuestras variables meteorológicas para realizar el clustering y añadimos a nuestro dataframe una nueva columna a la que llamaremos “weather_cluster”, la cual contendrá el cluster al que pertenece cada día en base a condiciones meteorológicas.

```
kmeans = KMeans(n_clusters=4, max_iter=600, algorithm = 'auto', random_state=50)
kmeans.fit(weather_scaled)
df_energy_weather_holliday['weather_cluster'] = kmeans.labels_
```

```
df_energy_weather_holliday['weather_cluster'].unique()
```

```
array([1, 0, 3, 2], dtype=int32)
```

Figura 3-17 Aplicación del algoritmo K-Means

Una vez terminado todo el preprocesado necesario para darle la forma que nos interesa a nuestros datos, pasamos a crear un nuevo dataframe al que llamaremos "df_modelos", en el que nos quedaremos con las columnas que nos interesan para pasarles a nuestros modelos predictivos, las cuales son energy_sum, LCLid, avg_energy, weather_cluster e Isholiday.

```
df_modelos
```

day	energy_sum	LCLid	avg_energy	weather_cluster	Isholiday
2011-11-23	90.385000	13	6.952692	1	0
2011-11-24	213.412000	25	8.536480	1	0
2011-11-25	303.993000	32	9.499781	0	0
2011-11-26	420.976000	41	10.267707	0	0
2011-11-27	444.883001	41	10.850805	0	0
...	...	...	...	...	...
2014-02-24	51994.547004	4994	10.411403	1	0
2014-02-25	51423.508001	4995	10.294997	1	0
2014-02-26	50943.305995	4993	10.202945	1	0
2014-02-27	51678.185998	4990	10.356350	1	0
2014-02-28	1042.266000	4987	0.208997	1	0

828 rows × 5 columns

Figura 3-18 Creación del dataframe "df_modelos"

Antes de pasar a realizar la partición de nuestros datos en entrenamiento y test, vamos a realizar un pequeño análisis de nuestra serie temporal. Primero, descompondremos los componentes de nuestra serie temporal para ver su tendencia, estacionalidad y residuo. Para ello, hacemos uso de la librería seasonal_descompose de statsmodel, la que nos permitirá visualizar nuestra serie temporal original, su tendencia, estacionalidad y residuo.

```
from statsmodels.tsa.seasonal import seasonal_decompose
decomposition = seasonal_decompose(df_modelos['avg_energy'], period=12, model='additive')
decomposition.plot()
plt.show();
```

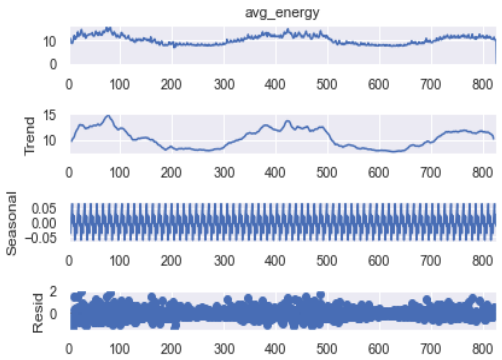


Figura 3-19 Uso de la librería seasonal_descompose de statsmodel

Una vez visualizados los componentes de nuestra serie, realizaremos también la prueba de Dickey-Fuller, la cual es un método estadístico que nos permite verificar si nuestra serie temporal es estacionaria o no. Que una serie temporal sea estacionaria significa que sus propiedades estadísticas, como la media o la varianza, son constantes en el tiempo.

Para aplicar este método, damos uso a la librería adfuller de statsmodel, la cual nos permitirá obtener el adf (valor de la estadística de prueba) y el p valor.

```
from statsmodels.tsa.stattools import adfuller
result = adfuller(df_modelos['avg_energy'])
print(result[0])
print(result[1])
```

```
-1.7529035736567935
0.4040814901345789
```

Figura 3-20 Prueba de Dickey-Fuller

El adf es un número que la prueba utiliza para decidir si rechazar o no la hipótesis nula. En el caso de la prueba de Dickey-Fuller, la hipótesis nula es que la serie temporal tiene una raíz unitaria, es decir, no es estacionaria.

El valor p es la probabilidad de que, dado que la hipótesis nula es verdadera, obtengamos una estadística de prueba al menos tan extrema como la que hemos calculado. Si el valor p es pequeño (generalmente se utiliza un umbral de 0.05), entonces rechazamos la hipótesis nula. En nuestro caso, no podemos rechazar la hipótesis nula debido a que nuestro valor p es superior a 0.05, por lo tanto, determinamos que la serie no es estacionaria.

3.3 Implementación de modelos predictivos

Una vez tenemos nuestros datos preparados para que los modelos puedan trabajar con ellos, procedemos a implementarlos. Como hemos comentado previamente, vamos a realizar una comparativa entre Random Forest, una red neuronal LSTM, ARIMAX y Prophet, con el fin de ver qué modelo se adapta mejor a nuestros datos y nos ofrece mejores resultados.

Para ello, calcularemos diferentes métricas de evaluación para ver el error cuadrático medio de cada uno de ellos.

3.3.1 RANDOM FOREST

Procedemos a dar uso de este modelo para predecir nuestra variable target que es avg_energy. Para ello, lo primero es realizar una partición de nuestro dataframe en entrenamiento y test, para la cual utilizaremos el método train_test_split que nos proporciona sklearn. Cabe destacar, que este método de por si realiza particiones aleatorias de los datos, pero en nuestro caso nos interesa que divida nuestro dataframe en orden para mantener nuestra serie temporal y poder comparar las predicciones con los valores originales que se encuentran almacenados en y_test. Para ello, damos uso de un parámetro llamado shuffle que incluye este método, el cual si ponemos su valor en false, nos permite segmentar el dataframe en el orden en el que está y no me manera aleatoria.

Una vez que la partición de nuestros datos está hecha, procedemos a instanciar el modelo con sus hiperparámetros, en los que indicaremos que cree un bosque con 10 árboles y el hiperparametro n_jobs = -1, para que el algoritmo utilice todos los procesadores disponibles, el

resto de hiperparámetros los dejamos como vienen en el modelo por defecto. Una vez hecho esto, podemos pasar a entrenar nuestro con los datos de entrenamiento.

```
X_train, X_test, y_train, y_test = train_test_split(
                                       df_modelos.drop(columns = "avg_energy"),
                                       df_modelos['avg_energy'],
                                       shuffle=False)

modelo = RandomForestRegressor(
            n_estimators = 10,
            criterion    = 'mse',
            max_depth    = None,
            max_features = 'auto',
            oob_score    = False,
            n_jobs       = -1,
            random_state = 123)

modelo.fit(X_train, y_train)
```

Figura 3-21 Instancia del modelo con sus hiperparámetros

Una vez que nuestro modelo ya está entrenado, pasamos a hacer las predicciones y a comparar los resultados con los valores reales de y_test obteniendo las métricas de evaluación MSE (Mean Absolute Error) y RMSE (Root Mean Absolute Error).

```
predicciones = modelo.predict(X = X_test)

mse = mean_squared_error(
        y_true  = y_test,
        y_pred  = predicciones,
        squared = False)

print(f"El error (mse) de test es: {mse}")
print(f"El error (rmse) de test es: {np.sqrt(mse)}")
```

```
El error (mse) de test es: 0.533356226127915
El error (rmse) de test es: 0.7303124167970274
```

Figura 3-22 Realización de las predicciones

Observamos que las métricas de evaluación de nuestro modelo no son malas desde un primer momento, aun así, es recomendable realizar hiperparametrización del modelo para encontrar los hiperparámetros óptimos con el fin de mejorar el rendimiento de nuestro modelo. Para ello, vamos a dar uso de la librería GridSearchCv, esto implica definir una cuadrícula de posibles valores para cada hiperparámetro que se desea optimizar. Luego, se entrena un modelo para cada combinación de hiperparámetros en la cuadrícula, utilizando un conjunto de datos de entrenamiento, y se evalúa su rendimiento en un conjunto de datos de validación. La combinación de hiperparámetros que produce el mejor rendimiento en el conjunto de datos de validación se elige como la mejor.

```
from sklearn.model_selection import RepeatedKFold
import multiprocessing
from sklearn.model_selection import GridSearchCV

param_grid = {'n_estimators': [150],
              'max_features': [5, 7, 9],
              'max_depth'   : [None, 3, 10, 20]
             }

grid = GridSearchCV(
        estimator  = RandomForestRegressor(random_state = 123),
        param_grid = param_grid,
        scoring    = 'neg_root_mean_squared_error',
        n_jobs     = multiprocessing.cpu_count() - 1,
        cv         = RepeatedKFold(n_splits=5, n_repeats=3, random_state=123),
        refit      = True,
        verbose    = 0,
        return_train_score = True)

grid.fit(X = X_train, y = y_train)

resultados = pd.DataFrame(grid.cv_results_)
resultados.filter(regex = '(param.*|mean_t|std_t)') \
    .drop(columns = 'params') \
    .sort_values('mean_test_score', ascending = False) \
    .head(4)
```

	param_max_depth	param_max_features	param_n_estimators	mean_test_score	std_test_score	mean_train_score	std_train_score
0	None	5	150	-0.397270	0.086202	-0.148222	0.011571
1	None	7	150	-0.397270	0.086202	-0.148222	0.011571
2	None	9	150	-0.397270	0.086202	-0.148222	0.011571
9	20	5	150	-0.397281	0.086198	-0.148222	0.011571

Figura 3-23 Resultados GridSearch

Esto nos devuelve un dataframe con diferentes valores para cada hiperparámetro. Para obtener los mejores valores, grid nos proporciona un método llamado best_params_, en el que nos devuelve la mejor combinación de hiperparámetros para nuestro modelo.

```
print("----------------------------------------")
print("Mejores hiperparámetros encontrados (cv)")
print("----------------------------------------")
print(grid.best_params_, ":", grid.best_score_, grid.scoring)
```

```
----------------------------------------
Mejores hiperparámetros encontrados (cv)
----------------------------------------
{'max_depth': None, 'max_features': 5, 'n_estimators': 150} : -0.397269815510078 neg_root_mean_squared_error
```

Figura 3-24 Mejor combinación de hiperparametros

Una vez obtenidos los mejores valores para los hiperparámetros, procedemos a realizar el mismo proceso que antes, pero cambiando los valores por los nuevos obtenidos. Cabe destacar que GridSearch puede ser computacionalmente costoso, por lo tanto, en este caso no le hemos pasado una gran cantidad de posibles valores.

Ahora, realizamos las predicciones con nuestro nuevo modelo hiperparametrizado y entrenado y obtenemos sus métricas de evaluación.

```
predicciones = modelo1.predict(X = X_test)

mse = mean_squared_error(
        y_true  = y_test,
        y_pred  = predicciones,
        squared = False
       )
print(f"El error (mse) de test es: {mse}")
print(f"El error (rmse) de test es: {np.sqrt(mse)}")
```

```
El error (mse) de test es: 0.5692160109815979
El error (rmse) de test es: 0.7544640554602968
```

Figura 3-25 Predicciones con el modelo hiperparametrizado y entrenado

Observamos que las métricas han empeorado ligeramente, esto puede deberse a que ninguno de los posibles valores que le hemos dado a GridSearch mejora los que teníamos previamente y hemos incrementado el coste computacional de manera considerable. Por último, vamos a ver gráficamente como han ido nuestras predicciones:

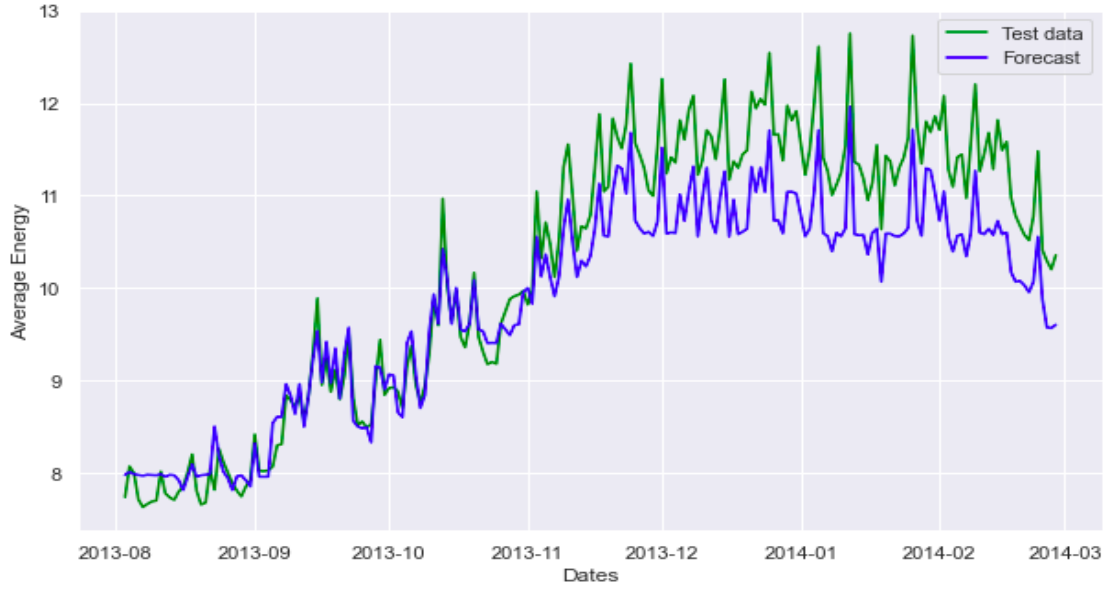


Figura 3-26 Gráfica de las predicciones

Vemos que el modelo al principio es capaz de ajustarse bien a nuestros datos y que interpreta de manera correcta la tendencia de nuestra serie, pero hacia el final de la serie predice unos valores un tanto más bajo de los reales.

3.3.2 LSTM

Procedemos a implementar nuestro siguiente modelo que será una red neuronal LSTM. Estas son una clase de redes neuronales recurrentes diseñadas para trabajar con secuencias de datos, como series de tiempo o texto. Lo primero que haremos será escalar nuestros datos para evitar sesgos en nuestra red, ya que, si las características tienen escalas muy diferentes, las características con una escala mayor podrían dominar el aprendizaje y así sesgar nuestro modelo. Para ello, procedemos a dar uso del escalador MinMaxScaler.

Lo siguiente es preparar nuestros datos de entrenamiento y test para la entrada de nuestro modelo, ya que las redes LSTM esperan una entrada en la forma de número de muestras, número de pasos de tiempo, número de características por paso de tiempo. Para ello, creamos una función a la que llamaremos créate_dataset(), que nos permitirá pasarle una matriz de datos de entrada y un valor de look_back, que es el número de pasos de tiempo anteriores que se deben considerar al predecir el siguiente paso de tiempo.

Esta función recorre el conjunto de datos y, para cada punto, toma el número look_back de pasos de tiempo anteriores y los agrega a dataX. Luego, agrega el paso de tiempo actual (es decir, el paso de tiempo que sigue a los pasos de tiempo en dataX) a dataY. El resultado es dos listas, dataX y dataY, que contienen secuencias de pasos de tiempo y el paso de tiempo siguiente, respectivamente.

Aplicamos esta función para crear nuestros conjuntos de datos trainX, trainY, testX y testY y por último los redimensionamos para que tengan las dimensiones de entrada que espera nuestra LSTM.

```
def create_dataset(dataset, look_back=1):
    dataX, dataY = [], []
    for i in range(len(dataset)-look_back-1):
        a = dataset[i:(i+look_back), :]
        dataX.append(a)
        dataY.append(dataset[i + look_back, 0])
    return np.array(dataX), np.array(dataY)

look_back = 1
trainX, trainY = create_dataset(train, look_back)
testX, testY = create_dataset(test, look_back)

trainX = np.reshape(trainX, (trainX.shape[0], look_back, trainX.shape[2]))
testX = np.reshape(testX, (testX.shape[0], look_back, testX.shape[2]))
```

Figura 3-27 Preparado de datos de entrada para LSTM

Una vez que nuestros datos están preparados para la entrada de la LSTM, procedemos a implementar nuestra red neuronal, a la cual definiremos como un modelo secuencial, le añadiremos una capa LSTM con 4 neuronas y le indicaremos que los datos de entrada serán el número de pasos en el tiempo en cada muestra de entrada y el número de características en cada paso de tiempo. Las neuronas de las redes LSTM, tienen una estructura más compleja que el resto de las redes, ya que estas incluyen tres "puertas" (la puerta de entrada, la puerta de olvido y la puerta de salida) que controlan el flujo de información dentro y fuera de la neurona.

También incluimos una capa Dense, que será la capa de salida de nuestra red, con una neurona, la cual estará conectada a todas las neuronas de la capa anterior y no se especifica ninguna función de activación, por lo que se utiliza la función de identidad por defecto (es decir, la salida es igual a la entrada).

Continuamos dando uso a la función compile, la cual se utiliza para configurar el aprendizaje del modelo y le especificamos que la función de pérdida sea el error cuadrático medio, debido a que estamos en un problema de regresión.

Por último, añadimos un callback a nuestra red llamado EarlyStopping, el cual se utiliza para evitar el sobreajuste de nuestro modelo, deteniendo el entrenamiento en caso de que una métrica de rendimiento deje de mejorar a lo largo de las épocas.

Una vez configurada nuestra red, procedemos a entrenarla pasándole nuestro conjunto de entrenamiento, le indicamos el número de épocas, que es la cantidad de veces que recorrerá el conjunto de datos, el batch_size, que es el número de muestras que se propagarán por la red a la vez, los datos de validación que debe tener en cuenta para evaluar el rendimiento y nuestro callback previamente implementado.

```
model = Sequential()
model.add(LSTM(4, input_shape=(look_back, trainX.shape[2])))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
early_stop = EarlyStopping(monitor='val_loss', patience=10, verbose=1)

history = model.fit(trainX, trainY, epochs=100, batch_size=1, verbose=2, validation_data=(testX, testY), callbacks=[ear
```

Figura 3-28 Instancia de la red y entrenamiento

Una vez entrenado nuestro modelo, pasamos a realizar las predicciones para poder calcular las métricas de evaluación de nuestra red y ver como se ajusta a nuestros datos. Primero vamos a redimensionar tanto las predicciones obtenidas como los datos que le hemos pasado previamente a la red para después hacer un reescalado de los datos y así poder visualizar una gráfica con los valores reales.

```
trainPredict = model.predict(trainX)
testPredict = model.predict(testX)

trainPredict_dataset_like = np.zeros(shape=(len(trainPredict), 4))
trainPredict_dataset_like[:,0] = trainPredict[:,0]
testPredict_dataset_like = np.zeros(shape=(len(testPredict), 4))
testPredict_dataset_like[:,0] = testPredict[:,0]

trainPredict = scaler.inverse_transform(trainPredict_dataset_like)[:,0]
testPredict = scaler.inverse_transform(testPredict_dataset_like)[:,0]

trainY_dataset_like = np.zeros(shape=(len(trainY), 4))
trainY_dataset_like[:,0] = trainY
testY_dataset_like = np.zeros(shape=(len(testY), 4))
testY_dataset_like[:,0] = testY

trainY = scaler.inverse_transform(trainY_dataset_like)[:,0]
testY = scaler.inverse_transform(testY_dataset_like)[:,0]

trainScore = math.sqrt(mean_squared_error(trainY, trainPredict))
print('Train Score: %.2f RMSE' % (trainScore))
testScore = math.sqrt(mean_squared_error(testY, testPredict))
print('Test Score: %.2f RMSE' % (testScore))
```

```
21/21 [==============================] - 0s 800us/step
6/6 [==============================] - 0s 1ms/step
Train Score: 0.55 RMSE
Test Score: 0.52 RMSE
```

Figura 3-29 Predicción y métricas LSTM

Observamos que las métricas de evaluación tanto para el conjunto de entrenamiento como para el conjunto de prueba son bastante buenas. Otra cosa para tener en cuenta es que la diferencia entre el error de entrenamiento y el de prueba son están bastante parejos, lo que es un indicador de que la red no está sobreentrenada. Por último, realizamos una visualización de las predicciones que ha hecho la red con los valores de prueba para ver que rendimiento nos ofrece.

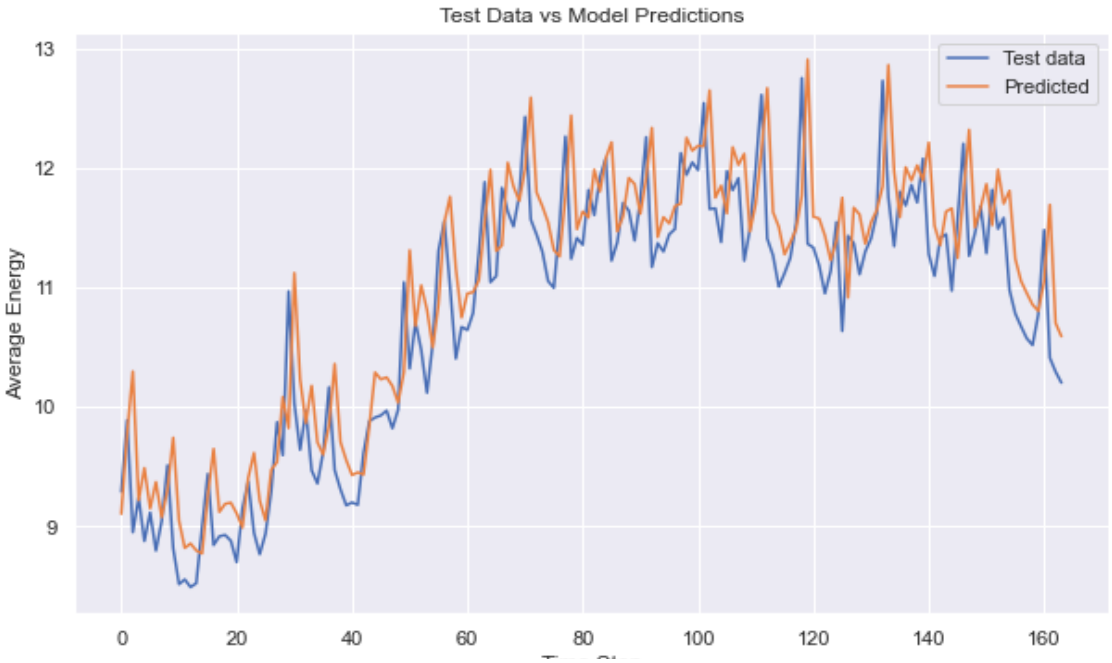


Figura 3-30 Comparativa predicción vs test

Como vemos, la red se ajusta bastante bien a nuestros datos y nos ofrece unos resultados bastante buenos.

3.3.3 Prophet

Procedemos a implementar el modelo Prophet. Este modelo, desarrollado por Facebook, se caracteriza por ser una herramienta para el análisis de series temporales con el fin de predecir patrones de datos futuros. Para instanciarlo, hacemos uso de la librería Prophet.

Debido a que nuestro df_modelos tiene como índice la variable day, lo primero que haremos será resetear el índice de nuestro dataframe, ya que una de las características de Prophet es que la variable temporal tiene que llamarse “ds” y la variable target “y”. Una vez renombradas estas

```
df_prophet = df_prophet.rename(columns={'day' : 'ds',
                                        'avg_energy' : 'y'})
df_prophet['y'] = np.log(df_prophet['y'])
df_prophet
```

	ds	LCLid	y	weather_cluster	Isholiday
0	2011-11-23	13	1.939129	1	0
1	2011-11-24	25	2.144349	0	0
2	2011-11-25	32	2.251269	0	0
3	2011-11-26	41	2.329004	0	0
4	2011-11-27	41	2.384239	1	0
...	...	...	...	...	...
823	2014-02-24	4994	2.342902	1	0
824	2014-02-25	4995	2.331658	1	0
825	2014-02-26	4993	2.322676	1	0
826	2014-02-27	4990	2.337600	1	0
827	2014-02-28	4987	-1.565437	3	0

2 variables, realizamos una transformación logarítmica de la variable “y” para estabilizar la varianza y mejorar la normalidad de nuestros datos.

Figura 3-31 Renombre de variables y transf log Prophet

Otra característica a tener en cuenta de Prophet, es que nos permite añadir días de vacaciones o días que puedan tener cierto valor especial para que el modelo se adapte mejor a nuestros a datos. En nuestro caso, como disponemos de una variable que son precisamente días festivos, le pasamos a nuestro modelo un dataframe con las fechas de los días festivos para que los tenga en cuenta a la hora de hacer predicciones.

```
holidays = pd.DataFrame({
    'holiday' : 'holiday',
    'ds' : pd.to_datetime(['2012-02-01', '2012-04-06', '2012-05-06', '2012-06-04',
                           '2012-07-05', '2012-08-27', '2012-09-04', '2012-12-25',
                           '2012-12-26', '2013-01-01', '2013-01-04', '2013-03-29',
                           '2013-05-27', '2013-06-05', '2013-08-26', '2013-12-25',
                           '2013-12-26'])
})
```

Figura 3-32 DataFrame días festivos

Ahora ya podemos pasar a montar nuestro modelo. Instanciamos Prophet y le pasamos como parámetro holidays nuestro dataframe previamente creado. Acto seguido, Prophet nos permite añadir otras variables como regresores con el fin de obtener más información acerca de los datos y poder realizar mejores predicciones. En este caso, le pasamos el número de contadores inteligentes y la variable weather_cluster como regresores.

Otra característica de Prophet es que nos permite modelar la estacionalidad de nuestra serie temporal, lo cual nos ayudará a mejorar el rendimiento del modelo. Como hemos visto previamente, en nuestros datos de consumo se repiten ciertos ciclos a lo largo del año, se consume más en invierno y menos en verano, por lo tanto, le añadimos a nuestro modelo un componente de estacionalidad anual.

Una vez añadidas todas estas características a nuestro modelo, procedemos a entrenarlo con los datos de entrenamiento.

```
prophet = Prophet(holidays = holidays)

prophet.add_regressor('weather_cluster')
prophet.add_regressor('LCLid')

prophet.add_seasonality(name='yearly', period=365, fourier_order=10)

prophet.fit(train_data)
```

Figura 3-33 Configuración modelo Prophet

Antes de realizar predicciones, vamos a evaluar el rendimiento de nuestro modelo utilizando el método cross_validation o validación cruzada. Para ello, le pasamos los parámetros que el modelo necesita, los cuales son initial = 365, que es el tamaño del período de entrenamiento inicial, period = 100, que es el tamaño del paso entre los cortes de validación cruzada y horizon = 206, que es el horizonte de pronóstico. Acto seguido, damos uso del método performance_metrics() para calcular métricas de rendimiento en las predicciones hechas durante la validación cruzada y así obtener información acerca de cómo se comporta nuestro modelo.

	horizon	mse	rmse	mae	mape	mdape	smape	\
0	20 days	0.018941	0.137626	0.129913	0.050049	0.056457	0.051483	
1	21 days	0.019092	0.138173	0.131586	0.050725	0.056457	0.052172	
2	22 days	0.018712	0.136790	0.127948	0.049276	0.056457	0.050692	
3	23 days	0.018397	0.135634	0.125351	0.048255	0.056457	0.049645	
4	24 days	0.018062	0.134395	0.123245	0.047491	0.056457	0.048857	
..	...	...	...	...	...	...	...	
182	202 days	1.665470	1.290531	1.290026	0.620132	0.622701	0.899288	
183	203 days	1.678298	1.295491	1.294979	0.622234	0.627888	0.903739	
184	204 days	1.693563	1.301370	1.300867	0.624723	0.629724	0.909033	
185	205 days	1.706972	1.306511	1.306044	0.627045	0.630026	0.913968	
186	206 days	1.721683	1.312129	1.311737	0.629713	0.631557	0.919596	

Figura 3-34 Métricas Cross Validation

Estas son las métricas que hemos obtenido haciendo la validación cruzada. Observamos que el modelo haciendo predicciones a corto plazo es bueno, pero conforme las predicciones van siendo más longevas a lo largo del tiempo, el modelo empeora.

Ahora, procederemos a realizar las predicciones para nuestro conjunto de test y ver cómo se comporta el modelo. El método predict de Prophet nos devuelve un dataframe con multitud de columnas informativas acerca de la predicción, pero en este caso nos quedaremos simplemente con la fecha, yhat, que son los valores de las predicciones, yhat_upper e yhat_lower, que son los intervalos de confianza en los que podrían moverse las predicciones.

```
forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].tail(10)
```

	ds	yhat	yhat_lower	yhat_upper
197	2014-02-18	2.482251	2.329809	2.619397
198	2014-02-19	2.467335	2.315971	2.610414
199	2014-02-20	2.468263	2.309090	2.608780
200	2014-02-21	2.473087	2.311471	2.622722
201	2014-02-22	2.497861	2.344599	2.647858
202	2014-02-23	2.540423	2.370573	2.688300
203	2014-02-24	2.481217	2.321029	2.626918
204	2014-02-25	2.465153	2.299272	2.619326
205	2014-02-26	2.449619	2.297063	2.603423
206	2014-02-27	2.449745	2.285686	2.592510

Figura 3-35 Predicciones e intervalos Prophet

Prophet nos facilita una visualización en la que se pueden observar estas predicciones con sus intervalos de confianza y el resto de la serie temporal previa a la predicción.

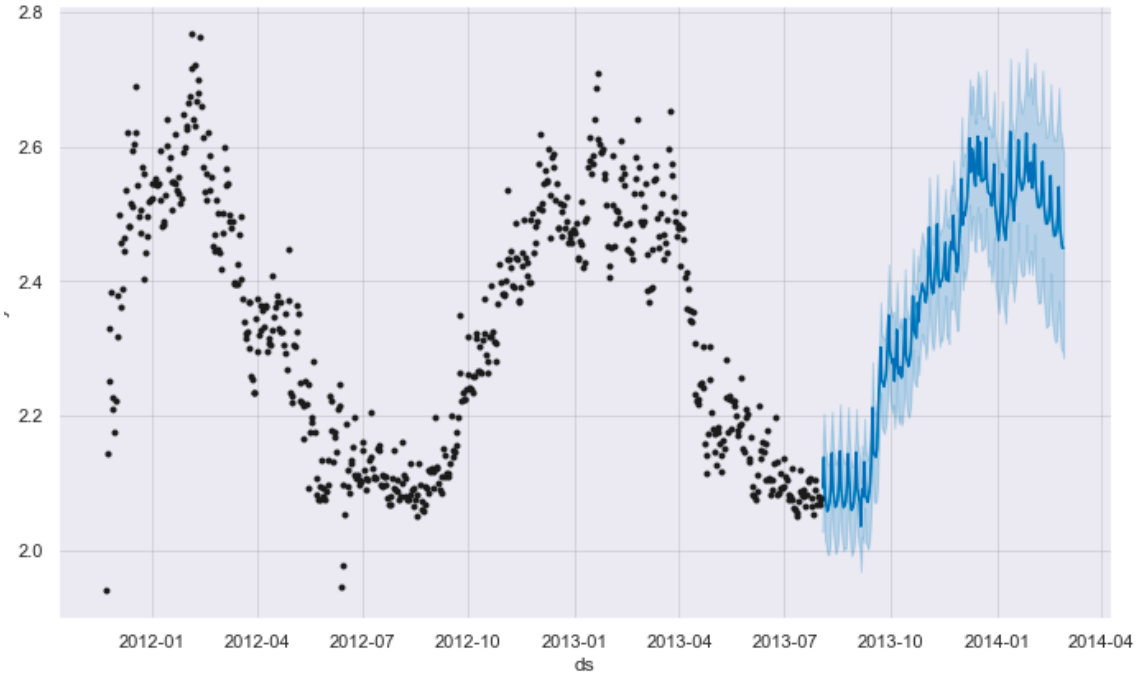


Figura 3-36 Gráfico yhat e intervalos Prophet

Por lo que vemos, nuestro modelo ha captado bastante bien la tendencia de nuestra serie y sigue una distribución bastante acorde a la de nuestra serie. Ahora vamos a cotejar los resultados de las predicciones con los datos de prueba y calcular la métrica de error para ver realmente cómo se comporta nuestro modelo. Para ello, en nuestro dataframe de prueba añadimos los valores predichos que tenemos almacenados en forecast, deshacemos la transformación logarítmica para ver los resultados sobre los valores reales y los visualizamos en un gráfico.

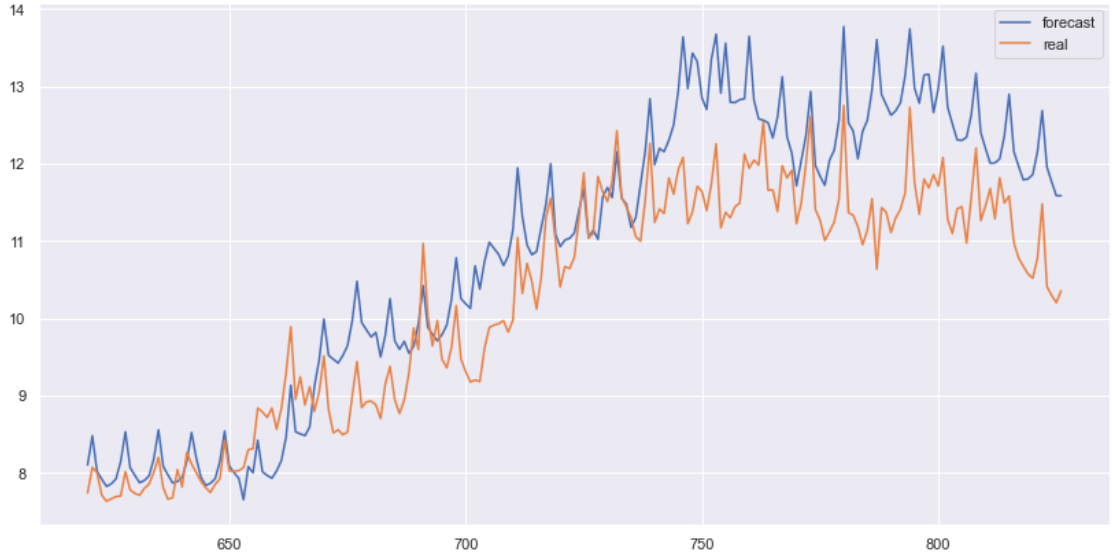


Figura 3-37 Comparativa yhat vs test

Observamos que el resultado concuerda con los resultados obtenidos en la validación cruzada, el modelo predice mejor en el corto espacio temporal y a medida que la predicción tiene un mayor horizonte, tiene una mayor tasa de error. Obtenemos un RMSE de 0.91 con un porcentaje de error del 6.5% del rango total de nuestros datos.

3.3.4 Comparativa de modelos

Tabla 3-1 Comparativa de modelos

	RANDOM FOREST	LSTM	PROPHET
RMSE	0.73	0.52	0.91
% ERROR	5.21%	3.71%	6.5%

Como podemos observar, el modelo que mejor se ajusta a nuestros datos es la red neuronal LSTM, con un Root Mean Squared Error (RMSE) de 0.52 y un porcentaje de error de 3.71%, lo que significa que las predicciones del modelo se desvían de los valores reales dicho porcentaje del rango total de nuestros datos.

En general, como hemos visto a lo largo de la implementación de los modelos, los 3 se ajustan de manera adecuada a los datos, sobre todo al corto espacio temporal, pero a medida que la predicción se alarga en el tiempo, los modelos pierden un poco de eficacia salvo la red LSTM, que vemos que se comporta adecuadamente a lo largo de toda la serie. Esto nos indica que ha captado bien los componentes de la serie temporal como su tendencia y estacionalidad.

3.3.5 Segmentación de clientes

Debido a la naturaleza de nuestros datos y que tenemos información acerca de contadores que son individuales y conocemos sus identificadores, podemos realizar un proyecto de segmentación de clientes. Este proyecto se basa en tratar de segmentar en diferentes grupos a los clientes en base a su consumo. Para ello, haremos uso de técnicas de clustering que nos permitirán encontrar patrones comunes en los datos y agruparlos en clusters.

Empezamos cargando otra vez nuestro archivo daily_dataset.csv en un dataframe con pandas. Para este caso, tan solo vamos a quedarnos con las variables day, LCLid y energy_sum, ya que el clustering lo haremos en función del consumo total diario de cada cliente.

Como hemos visto en la Figura 3-2, los contadores han sido instalados de forma progresiva a lo largo del tiempo, y no tenemos en ningún momento todos los contadores en funcionamiento, sino que cuando llega al pico de contadores instalados, este empieza a descender. Por lo tanto, filtramos el dataframe para quedarnos con los valores que pertenecen al año 2013, que es donde vemos un poco más estable el recuento de contadores en funcionamiento. En total en ese año, tenemos 5528 contadores distintos.

Para continuar con el preprocesado, observamos que hay 7 valores nulos (NaN) en nuestra variable energy_sum. No podemos trabajar con valores nulos en nuestros datos por lo tanto asumimos que esos días el consumo energético fue nulo y por lo tanto rellenamos esos valores con un 0. De esta forma, tenemos todos los valores de nuestro dataframe rellenos.

Para poder realizar el clustering por cliente, la forma de nuestro dataframe no es la adecuada, ya que tenemos varias filas de datos con el mismo id de contador, y lo que nos interesa es tener una fila por cada id único que tengamos. Para ello, pivotamos la tabla estableciendo como índice los id de los contadores, como columnas los días del año y como valores los valores del consumo de cada contador. Esto nos deja un dataframe con este aspecto:

```
df_clustering = df_clustering.pivot(index='LCLid', columns='day', values='energy_sum')
df_clustering
```

day	2013-01-01	2013-01-02	2013-01-03	2013-01-04	2013-01-05	2013-01-06	2013-01-07	2013-01-08	2013-01-09	2013-01-10	...	2013-12-22	2013-12-23	2013-12-24	2013-12-25	2013-12-26	2013-12-27	2013-12-28	2013-12-29	
LCLid																				
MAC000002	10.800	13.300	10.074	9.857	11.171	10.293	9.439	11.640	15.259	10.740	...	12.025	6.721	4.781	4.650	4.706	9.122	13.658	14.982	1
MAC000003	18.734	19.980	23.522	20.949	22.061	39.806	40.185	40.282	19.656	40.258	...	11.767	10.288	34.059	33.797	31.277	31.533	10.564	9.746	1
MAC000004	1.753	1.916	1.722	1.747	1.614	1.858	1.607	1.743	1.712	1.741	...	1.619	1.734	1.826	1.706	1.869	1.758	1.918	1.734	
MAC000005	4.339	5.700	5.177	6.188	5.369	11.436	5.689	3.753	5.092	5.240	...	5.280	4.784	5.241	9.048	3.257	6.935	5.680	2.976	
MAC000006	0.500	0.496	0.494	0.495	0.499	0.498	0.489	1.854	3.184	4.451	...	3.865	3.815	4.327	4.947	3.936	3.412	2.940	3.120	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
MAC005561	8.899	8.204	6.287	7.732	8.689	9.017	9.460	11.039	8.050	10.177	...	9.791	12.178	9.732	15.736	11.592	9.446	9.443	11.572	
MAC005562	10.841	9.727	8.290	9.681	9.314	9.202	9.145	7.469	8.173	8.599	...	12.912	13.259	24.519	14.370	13.373	14.532	10.706	14.832	1
MAC005564	4.926	5.126	4.017	4.214	3.493	5.064	3.526	4.953	5.419	3.833	...	3.569	5.088	6.646	4.439	4.173	5.206	3.482	3.529	
MAC005566	5.318	5.427	5.295	5.259	5.275	5.334	5.278	5.255	5.335	5.502	...	4.903	4.827	4.789	4.858	4.847	5.834	4.833	4.798	
MAC005567	5.760	6.186	5.820	7.803	3.919	6.618	6.910	6.318	7.284	3.565	...	5.302	6.697	5.416	12.650	4.443	9.548	7.131	8.716	

5528 rows × 365 columns

Figura 3-38 Dataframe clustering

Una vez hecho esto, en nuestra tabla vuelven a aparecer valores nulos en algunos días, eso significa que ese contador a partir de ese día ya no se tiene información, por lo tanto,

interpretamos que se ha desinstalado o ha dejado de funcionar. Aun así, estos valores vamos a rellenarlos con la media del resto de valores de consumo de su fila, ya que, si lo rellenamos con 0, estaríamos sesgando el consumo de ese cliente y podría incluirse en un cluster que no le corresponde.

Nos falta un último paso para poder pasarle nuestros datos a un algoritmo de clustering, y es estandarizarlos, ya que los algoritmos de clustering como K-Means son sensibles a la escala de los datos y siempre es recomendable normalizar los datos antes de trabajar con ellos. Para ello, utilizamos el escalador StandardScaler, que hará que nuestros datos tengan media 0 y varianza 1. Esto lo logra restando la media a cada característica y dividiendo por su desviación estándar.

Antes de aplicar nuestros datos un modelo K-Means, vamos a realizar el método del codo (previamente explicado) para determinar cuál es el número óptimo de clusters.

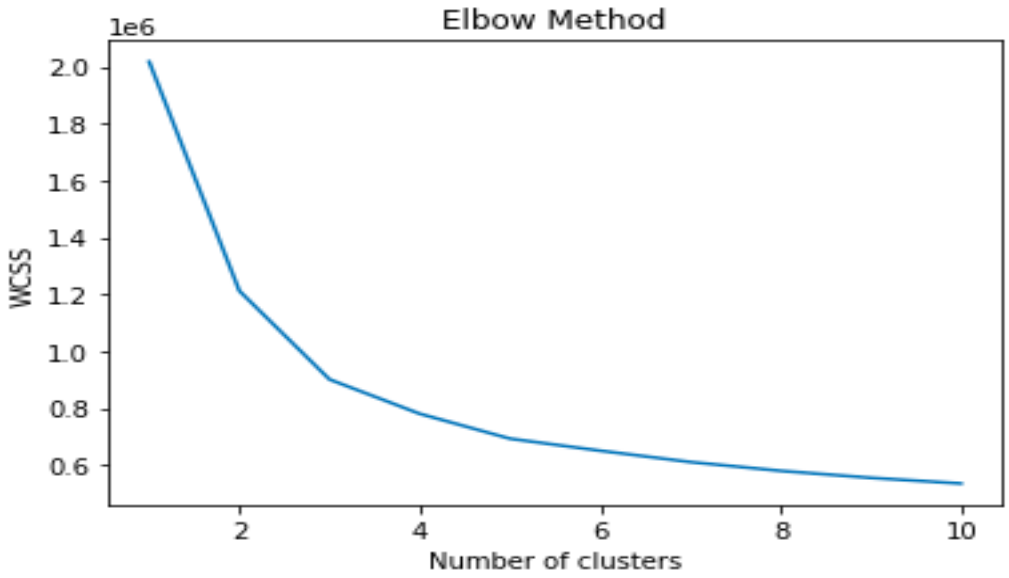


Figura 3-39 Método de codo

Observando el gráfico de codo, podemos determinar que un número adecuado de clusters para nuestros datos es 3.

Debido a la forma de nuestro dataframe, utilizar K-Means simplemente nos resultaría muy complicado a la hora de hacer un profiling o análisis posterior de cada cluster debido a que nuestro dataframe tiene 365 dimensiones. Para ello, daremos uso de mapas auto organizativo (SOM), que nos permitirán realizar clustering y a la vez reducir la dimensionalidad de nuestro dataframe para así poder hacer un análisis de los clusters más correcto.

Un SOM se compone de una serie de nodos o neuronas, cada uno de los cuales tiene un vector de pesos del mismo tamaño que los vectores de datos de entrada. Durante el proceso de entrenamiento, cada vector de datos de entrada se compara con todos los vectores de pesos en el SOM. El nodo cuyo vector de pesos es el más cercano al vector de datos de entrada (según alguna medida de distancia, como la distancia euclidiana) se denomina nodo ganador. Luego, los pesos del nodo ganador y sus nodos vecinos se ajustan para acercarse al vector de datos de entrada. Este proceso se repite muchas veces para todos los vectores de datos de entrada.

Al final del entrenamiento, los nodos del SOM se habrán ajustado de tal manera que representan un mapa de todas las entradas de datos, donde la ubicación de cada nodo en el mapa refleja las propiedades de los datos que representa. Esto permite visualizar la estructura de los datos de entrada de una manera que preserva las relaciones topológicas.

Para implementarlo, daremos uso de la librería MiniSom. Lo primero será instanciar el objeto MiniSom, al que le daremos un tamaño de 30x30, es decir, estará compuesto por 900 neuronas, le indicamos que la dimensión de los vectores de entrada será el número de columnas de los datos, ajustamos el hiperparámetro sigma a 1, que es el radio de los vecinos que se van a ajustar para cada vector de entrada, y el learning_rate a 0.5, que controla cuánto se ajustan los pesos en cada paso. Acto seguido, entrenamos nuestro SOM pasándole nuestros datos y el número de veces que realizará el entrenamiento, en este caso 10000.

Una vez entrenado el modelo, recuperamos los pesos de las neuronas y los remodelamos convirtiéndolos en una matriz bidimensional para que puedan ser procesados por el algoritmo K-Means.

Ahora podemos instanciar el modelo K-Means indicándole el número de clusters que hemos obtenido con el método de codo y entrenamos nuestro modelo con los pesos del SOM que hemos transformado previamente.

Por último, remodelamos los resultados del clustering para que coincidan con la forma del SOM original, lo que nos permitirá visualizar de una forma más fácil los clusters.

```
data = df_clustering_scaled.values

som_size = 30
som = MiniSom(som_size, som_size, data.shape[1], sigma=1.0, learning_rate=0.5)
som.train_random(data, 10000)

weights = som.get_weights()

flattened_weights = weights.reshape(-1, weights.shape[-1])

kmeans = KMeans(n_clusters=3, random_state = 42)
clusters = kmeans.fit_predict(flattened_weights)

clusters = clusters.reshape(som_size, som_size)
```

Figura 3-40 SOM y K-Means

Una vez que ya tenemos el clustering hecho, vamos a obtener los perfiles de los grupos que hemos creado con K-Means. Para ello, recorremos y almacenamos los nodos ganadores para cada vector de entrada en los datos mediante el método som.winner(), que nos devuelve la ubicación del nodo en el SOM que es más similar al vector de entrada. Una vez obtenido, recorremos los grupos de K-means para ver que vectores de entrada han sido asignados a cada grupo y los almacenamos en una variable que llamaremos cluster_data. Por último, calculamos del perfil de cada grupo la media de los vectores de entrada en el grupo. Esto resulta en un vector que representa el "centro" del grupo en el espacio de entrada.

```
winners = np.array([som.winner(x) for x in data])

for i in range(3):
    mask = clusters[winners[:,0], winners[:,1]] == i
    cluster_data = data[mask]

    cluster_profile = cluster_data.mean(axis=0)
    print(f"Perfil del cluster {i}: {cluster_profile}")
```

Figura 3-41 Perfiles de los cluster

Ahora vamos a ver los resultados de los perfiles obtenidos:

```
Perfil del cluster 0: [-0.32951088 -0.33702697 -0.34235488 -0.33772137 -0.33585135 -0.34537648
 -0.34389495 -0.34035208 -0.34026896 -0.34009726 -0.33941287 -0.33754122
 -0.3424957  -0.33517084 -0.33399277 -0.33112814 -0.33078586 -0.32667847
 -0.32462216 -0.32593224 -0.33168876 -0.33318316 -0.33399846 -0.33322211
```

Figura 3-42 Perfil del cluster 0

```
Perfil del cluster 1: [0.94256328 0.94981815 0.97072544 0.94242269 0.96021795 0.99621735
 0.94297196 0.91250791 0.906181   0.91232333 0.9188438  0.93928769
 0.9633157  0.90263795 0.90345263 0.89617182 0.89488074 0.88246383
```

Figura 3-43 Perfil del cluster 1

```
Perfil del cluster 2: [2.93108771 3.07412783 3.09121426 3.13047915 2.99006686 3.02798468
 3.27690127 3.35411094 3.3859368  3.34893333 3.29750356 3.14288684
 3.13424906 3.28154458 3.24868628 3.21831944 3.21694329 3.18397815
```

Figura 3-44 Perfil del cluster 2

Esto nos da una idea de cómo se distribuyen los contadores entorno a su consumo, pero vamos a realizar un gráfico para verlo de forma más visual. Para ello, tenemos que obtener los clusters a los que pertenece cada contador mapeando primero que neurona ganadora corresponde a cada vector de entrada y a que grupo pertenece dicha neurona.

Una vez hecho esto, creamos en nuestro dataframe una nueva columna a la que llamaremos cluster, que indicará a que cluster pertenece cada contador. Obtenemos que dentro del cluster 0, están 4379 contadores, en el cluster 1 968 y en el cluster 2 181.

Para crear la gráfica, primero vamos a deshacer la normalización de los datos para ver los valores de consumo reales.

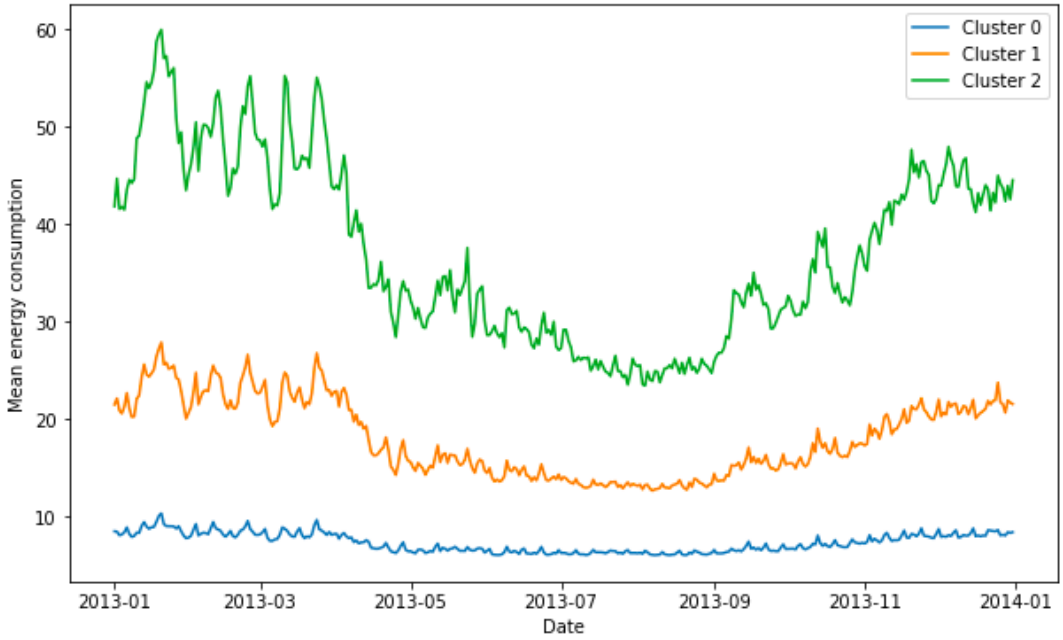


Figura 3-45 Consumo medio de clientes por cluster

En la gráfica se muestra el consumo medio de todos los integrantes del cluster a lo largo del año. Como podemos observar, estos gráficos se corresponden con los valores obtenidos previamente en los perfiles de grupo.

Observamos que los contadores que pertenecen al cluster 2, son los que tienen un mayor consumo, es posible que estos contadores estén instalados en empresas o en establecimientos que supongan un alto gasto energético. El cluster 1, tiene un consumo medianamente alto en relación a la mayoría de contadores, que recordamos que pertenecen al cluster 0. Esto puede deberse a que los contadores estén instalados en pequeños negocios que requieran un consumo mayor que el de un hogar, pero no tan alto como oficinas de empresas o grandes establecimientos. Por último, el cluster 0 es el que más contadores contiene, pero el que menos energía consume a lo largo del año. Esto puede deberse a que los contadores están instalados en contadores de hogares independiente en el que su consumo no es desmesurado.

4. Conclusiones

En el desarrollo de este trabajo de fin de grado, hemos explorado la influencia de diversas variables meteorológicas en el consumo energético de hogares en la ciudad de Londres. Hemos encontrado correlaciones significativas, con una tendencia clara a que el consumo energético aumenta a medida que la temperatura y el índice ultravioleta disminuyen, lo que evidencia una relación inversa. En contraste, la humedad mostró una relación directa con el consumo energético, aumentando este cuando la humedad es más alta. Sin embargo, no se pudo establecer una distinción clara en el consumo energético entre días festivos y no festivos.

Con toda esta información previamente obtenida a través de nuestros datos, hemos logrado implementar 3 modelos predictivos para obtener futuros consumos energéticos de los clientes:

-Un modelo específico de series temporales como Prophet, el cual proporciona una gran información tanto de los componentes de una serie temporal como de las predicciones del propio modelo, obteniendo una tasa de error de un 6.5%

-Un modelo de aprendizaje supervisado como Random Forest basado en árboles de decisión, del que hemos obtenido aparte de un resultado aceptable, con un error de 5.21%, la conclusión de que, en algunos casos, la hiperparametrización de modelos puede conllevar un aumento considerable del coste computacional del proyecto sin ofrecer mejores resultados.

-Una red LSTM que se ha adaptado muy bien a nuestros datos a pesar de no tener múltiples capas, obteniendo un error de 3.71%, y en la que hemos aprendido que, aunque este tipo de redes son más frecuentes en problemas de NLP (Procesamiento de lenguaje natural), pueden resultar de mucha utilidad para problemas de regresión.

De toda esta parte predictiva, concluimos que el modelo que mejor se adapta a nuestros datos es la red neuronal LSTM. A pesar de ser una red que no tiene múltiples capas, hemos conseguido que aprenda y capte bien los componentes de nuestra serie temporal. Esta red podría implementarse con múltiples capas pero debido a que nuestro conjunto de datos no es demasiado extenso, podría ocurrir que la red se sobreentrenase y se volviese más complicado encontrar un equilibrio entre los parámetros para ajustar correctamente nuestra red.

A mayores de toda la parte predictiva, hemos realizado una segmentación de clientes en la que, sin mucho contexto, hemos sido capaces de dividir a nuestros clientes en base a su consumo, obteniendo que nuestros clientes se dividen en 3 grupos, en los que hemos visualizado el consumo medio de cada grupo a lo largo de toda la serie temporal y ver sus diferencias de consumo, permitiendo así tener un mayor conocimiento acerca de nuestros clientes y permitiéndonos realizar ofertas personalizadas a cada grupo en base a su consumo.

Estos hallazgos tienen implicaciones importantes, particularmente en el diseño de políticas de eficiencia energética. La posibilidad de prever las demandas de energía en función de las condiciones meteorológicas puede ayudar a los consumidores a ajustar sus comportamientos y contribuir a un uso más eficiente de la energía.

A pesar de estas conclusiones, debemos ser conscientes de las limitaciones de este estudio. La investigación se centró exclusivamente en la ciudad de Londres y, por lo tanto, es posible que los patrones encontrados no sean aplicables en otras regiones con diferentes condiciones climáticas. Además, se limitó a considerar factores meteorológicos y festivos, sin tener en cuenta otros posibles determinantes del consumo de energía, como los hábitos de los hogares o los precios de la energía.

Mirando hacia el futuro, sería interesante replicar este estudio en otras regiones o introducir nuevas variables en el análisis. Por ejemplo, se podría analizar cómo los patrones de comportamiento de los consumidores, sus horarios laborales y de descanso, influyen en el consumo de energía. Asimismo, sería relevante aplicar y comparar otros modelos de aprendizaje automático en la predicción del consumo de energía.

Para concluir, este trabajo de fin de grado ha ofrecido una valiosa perspectiva sobre el papel de las variables meteorológicas en el consumo energético de los hogares. Aunque queda margen para profundizar y ampliar la investigación, nuestros hallazgos constituyen una base sólida para comprender y predecir el consumo de energía, resaltando la importancia de considerar las condiciones meteorológicas en los esfuerzos por mejorar la eficiencia energética

Bibliografía

1. **D, Jean-Michel. Kaggle. 2022. https://www.kaggle.com/datasets/jeanmidev/smart-meters-in-london. [En línea]**

2. Energia y sociedad. El proceso de liberalización de los sectores energéticos. 2022. [Citado el: 18 de enero de 2023.] https://www.energiaysociedad.es/manual-de-la-energia/1-5-el-proceso-de- liberalizacion-de-los-sectores-energeticos/. [En línea]

3. Investors.Wiki. Series de tiempo. 13 de diceimbre de 2022. [Citado el: 18 de enero de 2023.] https://investors.wiki/es/timeseries. [En línea]

4. Invatati afaceri. ¿Qué es una serie temporal y cómo se utiliza para analizar datos? 10 de diciembre de 2022. [Citado el: 18 de enero de 2023.] https://invatatiafaceri.ro/es/diccionario-financiero/que-es-una-serie-temporal-y- como-se-utiliza-. [En línea]

5. Ricardo, Rodrigo. Exonegocios. ¿Qué es una serie temporal? 2022. [Citado el: 18 de enero de 2023.] https://exonegocios.com/que-es-una-serie- temporal/. [En línea]

6. Banerjee, Probir. Tutorialspoint. Time Series Analysis: Definition and Components. 15 de abril de 2022. [Citado el: 18 de enero de 2022.] https://www.tutorialspoint.com/time-series-analysis-definition-and-components. [En línea]

7. Inversiopedia. Series temporales.2022. [Citado el: 18 de enero de 2023.] https://inversiopedia.com/series-temporales/. [En línea]

8. ***Breiman, Leo. Statistical modeling: The two cultures (with comments and a rejoinder by the author). s.l. : Statistical science 16.3 (2001).***

9. IBM. What is Machine Learning (ML)? Febrero de 2022. [Citado el: 16 de enero de 2023.] https://ischoolonline.berkeley.edu/blog/what-is-machine- learning/. [En línea]

10. Morgunov, Anton. Neptuneai. The Life Cycle of a Machine Learning Project: What Are the Stages? 14 de noviembre de 2022. [Citado el: 16 de enero de 2023.] https://neptune.ai/blog/life-cycle-of-a-machine-learning-project. [En línea]

11. Datacurate. 6 Steps that are Key to Data Preparation in Machine Learning. 2022, 15 de enero de 2023. https://www.datacurate.ai/6-steps-that-are-key- to-data-preparation-in-machine-learning/. [En línea]

12. Datacurate, ref 11. [En línea]. [En línea]

13. Castillo, Dianne. Seldon. How to Build a Machine Learning Model. 11 de septiembre de 2021. https://www.seldon.io/how-to-build-a-machine-learning-model#:~:text=Six%20steps%20to%20build%20a%20machine%20learning%20model,...%206%20Deploy%20th. [En línea]

14. Seldon, ref 13 . [En línea]

15. Seldon, ref 13 . [En línea]

16. AprendeIA. Aprendizaje Supervisado. 2022. [Citado el: 07 de enero de 2023.] https://aprendeia.com/todo-sobre-aprendizaje-supervisado-en-machine- learning/. [En línea]

17. Aprendeia. ¿Qué es el Aprendizaje no Supervisado? 2022. [Citado el: 08 de Enero de 2023.] https://aprendeia.com/aprendizaje-no-supervisado-machine- learning/. [En línea]

18. CEUPE. Aprendizaje por refuerzo: Concepto, características y ejemplo.2022. [Citado el: 09 de Enero de 2023.] https://www.ceupe.com/blog/aprendizaje-por-refuerzo.html. [En línea]

19. Aprende Machine Learning. Principales Algoritmos usados en Machine Learning.4 de Noviembre de 2017. [Citado el: 10 de Enero de 2023.] https://www.aprendemachinelearning.com/principales-algoritmos-usados-en- machine-learning/#:~:text=Los%2. [En línea]

20. Master's in Data Science. What is a Decision Tree? 2022. [Citado el: 14 de enero de 2023.] https://www.mastersindatascience.org/learning/machine- learning-algorithms/decision-tree/. [En línea]

21. *Yaseen, Khan, Muhammad, Shoaib, Siddiqi, Muhammad y Suffian, Nizami, Muhammad. Research gate. A Comprehensive Experiment with Distant Supervision, Machine Learning, and Word Embedding-Based Deep Learning Techniques. Septiembre de 2021. [Cit.*

22. Javatpoint. Support Vector Machine Algorithm. 2022. [Citado el: 14 de enero de 2023.] https://www.javatpoint.com/machine-learning-support-vector- machine-algorithm. [En línea]

23. Tyagi, Neelam. AnalyticsStep. What is K-means Clustering in Machine Learning? 14 de marzo de 2020. [Citado el: 18 de enero de 2023.] https://www.analyticssteps.com/blogs/what-k-means-clustering-machine-learning. [En línea]

24. Deepai. Gradient Boosting. 2022. [Citado el: 16 de enero de 2023.] https://deepai.org/machine-learning-glossary-and-terms/gradient-boosting. [En línea]

25. IBM. Convolutional Nueral Networks. 2022. [Citado el: 12 de enero de 2023.] https://www.ibm.com/topics/convolutional-neural-networks. [En línea]

26. TECHOPEDIA. Long Short-Term Memory. 2022. [Citado el: 11 de enero de 2023.] https://www.techopedia.com/definition/33215/long-short-term- memory-lstm. [En línea]

27. Canziani, Alfredo. ATCOLD. *Arquitectura de las RNNs y modelos LSTM.* [En línea] 3 de marzo de 2020. [Citado el: 3 de julio de 2023.] https://atcold.github.io/pytorch-Deep-Learning/es/week06/06-3/.

28. Dominodatalab. What is Anaconda? 2022. [Citado el: 17 de enero de 2023.] https://www.dominodatalab.com/data-science-dictionary/anaconda. [En línea]

29. Driscoll, Mike. Realpython. Jupyter Notebook: An Introduction. 2022. [Citado el: 17 de enero de 2023.] https://realpython.com/jupyter-notebook-introduction/#:~:text=The%20Jupyter%20Notebook%20is%20an,the%20people%20at%20Project%20Jupyter.. [En línea]

30. Greyrat, Rudeus. BarcelonaGeeks. Python para ciencia de datos. 5 de julio de 2022. [Citado el: 17 de enero de 2023.] https://barcelonageeks.com/python- para-la-ciencia-de-datos. [En línea]

31. Aprende con Alf. La librería Pandas. 2022. [Citado el: 17 de enero de 2023.] https://aprendeconalf.es/docencia/python/manual/pandas/.

32. NumPy. NumPy documentation. 2022. [Citado el: 17 de enero de 2023.] https://numpy.org/doc/stable/.

33. TutorialsTeacher. Python - Random Module. 2022. [Citado el: 17 de enero de 2023.] https://www.tutorialsteacher.com/python/random-module.

34. Programación. Introducción a la librería Matplotlib de Python. 2022. [Citado el: 17 de enero de 2023.] https://programacion.net/articulo/introduccion_a_la_libreria_matplotlib_de_pytho n_1599.

35. Seaborn. Seaborn: statistical data visualization. 2022. [Citado el: 17 de enero de 2023.] https://seaborn.pydata.org/.

36. TutorialsPoint. Scikit Learn Tutorial. 2022. [Citado el: 17 de enero de 2023.] https://www.tutorialspoint.com/scikit_learn/index.htm.

37. Statsmodels. Statsmodels. 2022. [Citado el: 17 de enero de 2023.] https://www.statsmodels.org/stable/index.html.

38. PYpi. PMDARIMA. 2022. [Citado el: 17 de enero de 2023.] https://pypi.org/project/pmdarima/.

39. Ionos. *Keras: biblioteca de código abierto para crear redes neuronales.* [En línea] 8 de octubre de 2022. [Citado el: 01 de julio de 2023.]

40. Application of Long Short-Term Memory (LSTM) Neural Network for Flood Forecasting. Le, Hien, Xuan, y otros. 2019. [En línea]

41. Lucidspark. Cómo hacer y usar árboles de decisiones. 2022. [Citado el: 02 de enero de 2023.] https://lucidspark.com/es/blog/como-hacer-arboles-de- decisiones. [En línea]

42. IBM . What is random forest? 2022. [Citado el: 16 de Enero de 2023.] https://www.ibm.com/topics/random-forest. [En línea]

43. Support Vector Machine Algorithm. 2022. [Citado el: 06 de enero de 2023.] https://www.javatpoint.com/machine-learning-support-vector-machine- algorithm. [En línea]

44. Dehghani, Ali. Baeldung. The K-Means Clustering Algorithm in Java. 26 de noviembre de 2022. [Citado el: 07 de enero de 2022.] https://www.baeldung.com/java-k-means-clustering-algorithm. [En línea]

45. Al Wiki. Gradient Boosting. 2022. [Citado el: 07 de enero de 2023.] https://machine-learning.paperspace.com/wiki/gradient-boosting. [En línea]

46. K, E, Swapna. Developers Breach. Convolution Neural Networks. 2022. [Citado el: 08 de enero de 2023.] https://developersbreach.com/convolution- neural-network-deep-learning/. [En línea]

47. *Aprendeia. ¿Qué es el Aprendizaje no Supervisado? 2022. [Citado el: 08 de Enero de 2023.] https://aprendeia.com/aprendizaje-no-supervisado-machine- learning/.*

48. *Luna, Gonzalez, Javier. Medium. Tipos de aprendizaje automático. 08 de febrero de 2018. [Citado el: 03 de enero de 2023.] https://medium.com/soldai/tipos-de-aprendizaje-automático-6413e3c615e2.*

49. *CEUPE, ref 18.*