GRADO EN INGENIERÍA DEL SOFTWARE

APLICACIÓN DE TÉCNICAS DE MACHINE LEARNING PARA LA DETECCIÓN DE PATRONES DE CONSUMO ENERGÉTICO

MACHINE LEARNING APPLICATION TO DETECT ENERGY CONSUMPTION PATTERNS

Realizado por
Yeray Ruiz Suárez

Tutorizado por
Daniel Garrido Márquez
Cristian Martín Fernández

Departamento
Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA

MÁLAGA, julio de 2022

Resumen

Debido a la entrada en vigor de la nueva ley que aprobó el cambio de tarifa del consumo eléctrico en España, el precio del kWh ha ido en aumento, por lo que urge la necesidad de encontrar soluciones para evitar altos costes en la factura eléctrica. El grupo ERTIS de la Universidad de Málaga acepta el reto iniciando el proyecto K-Project que se basa en la construcción de un prototipo inteligente, llamado Smart Meter, capaz de monitorizar el consumo y proporcionar al usuario una mayor accesibilidad a los datos del consumo eléctrico. Este documento relata la parte del proyecto referente a la detección de patrones de consumo energético y la predicción de valores de consumo mediante la aplicación de técnicas de inteligencia artificial y machine learning. En él se detallan todos los procesos de extracción, análisis y preprocesamiento de los datos además del posterior entrenamiento de series temporales referentes al consumo eléctrico y el análisis de los resultados. Finalmente se lleva a cabo la elaboración de un servicio para poder generar predicciones a los modelos entrenados desde una API web que seguidamente muestra los resultados obtenidos y realiza una comparación respecto a los valores reales y un análisis del error.

Palabras clave:

Machine Learning, Tensorflow, consumo eléctrico, predicción, series temporales.

Abstract

Due to the entry into force of the new law that approved the change of tariff for electricity consumption in Spain, the price of kWh has been increasing, so there is an urgent need to find solutions to avoid high costs in the electricity bill. The ERTIS group of the University of Malaga accepts the challenge by initiating the K-Project, which is based on the construction of an intelligent prototype, called Smart Meter, capable of monitoring consumption and providing the user with greater accessibility to electricity consumption data. This document describes the part of the project relating to the detection of energy consumption patterns and the prediction of consumption values through the application of artificial intelligence and machine learning techniques. It details all the data extraction, analysis, and pre-processing processes, as well as the subsequent training of time series related to electricity consumption and the analysis of the results. Finally, a service is developed to generate predictions to the trained models from a web API that then displays the results obtained and makes a comparison with respect to the real values and an error analysis.

Keywords:

Machine Learning, Tensorflow, forecast, electricity consumption, timeseries.

Índice de contenidos

Índice de figuras

Índice de tablas

1

Introducción

1.1 Motivación

La **inteligencia artificial (IA)** es la "habilidad de una máquina de presentar las mismas capacidades que los seres humanos, como el razonamiento, el aprendizaje, la creatividad y la capacidad de pensar" [1]. La IA intenta predecir los comportamientos a través de acciones anteriores, mediante un entrenamiento basado en algoritmos. **Machine learning** es una "rama de la IA y la informática que se centra en el uso de datos y algoritmos para imitar la forma en la que aprenden los seres humanos, con una mejora gradual de su precisión"[2].

El **consumo eléctrico** es la "cantidad de energía utilizada en un punto de suministro durante un periodo de tiempo determinado"[3]. El consumo eléctrico es medido mediante kilowatios hora (kWh), y es la unidad mediante la cuál las compañías eléctricas realizan el cómputo de la factura de la luz. Normalmente, el consumo se obtiene a través de un medidor global o también conocido como contador de luz, que está capacitado para obtener todo el consumo de una vivienda o instalación.

De acuerdo con Red Eléctrica de España (REE) [3], "el consumo de kWh mensual de un hogar español es de unos 270 kWh, lo que hace un total de 3.272 kWh/año aproximadamente". No obstante, la cantidad de energía consumida en cada vivienda no es la misma para todo el mundo, ya que varía entre las características y dimensiones de la vivienda,

los aparatos eléctricos que estén en funcionamiento, la ubicación del suministro, el número de personas que habitan en ella y sus hábitos de consumo.

El 1 de junio de 2021, entró en vigor la nueva tarifa de luz en España. Por ejemplo, para todos los usuarios con una potencia contratada menor de 15kW y una tensión menor a 1 kV cambia la metodología del cálculo de los peajes de acceso de la electricidad, que son "aquellos importes que se paga en las facturas por tener acceso a la electricidad y que van destinados al mantenimiento de la red de transporte y distribución" [4]. La aplicación de la nueva norma establecía que la tarifa de peaje a utilizar sería la 2.0 TD, con 3 períodos de discriminación horaria que son Punta (más cara), Llano (precio medio) y Valle (precio reducido) (Figura 1.1).

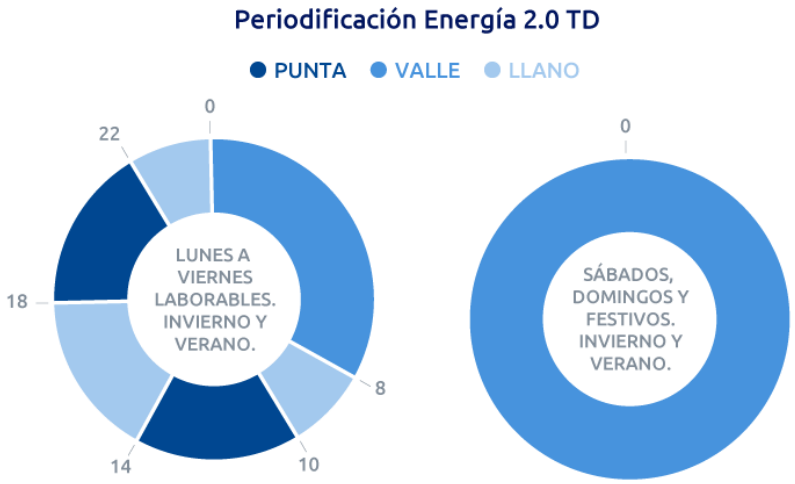


Figura 1.1. Periodos de discriminación de la tarifa eléctrica [5].

Entonces, desde este cambio de tarifa hasta hoy, se produce una inflación del precio de la luz incrementándose con el paso de los meses, batiendo récords históricos, por lo que el nuevo objetivo de muchas personas pasó a ser el de ahorrar en la factura de luz. Para ello, surge la necesidad de predecir valores como el consumo de energía para predecir el futuro comportamiento de un sistema y su facturación. De esta forma, el usuario puede tener un mayor control del consumo energético y prevenir gastos elevados.

La meta es minimizar el problema descrito, por lo que el trabajo se centrará en la aplicación de machine learning para desarrollar un modelo de predicción para inferir los valores del consumo energético de un contador inteligente llamado Smart Meter que actuará

como complemento del proyecto *CONTADORES INTELIGENTES Y ACCESIBLES PARA EL HOGAR* del K-Project del grupo ERTIS de la Universidad de Málaga. "El proyecto *CONTADORES INTELIGENTES Y ACCESIBLES PARA EL HOGAR* es una iniciativa de la UMA que persigue la creación de un prototipo funcional de contador inteligente y accesible (Smart Meter), para la monitorización del consumo energético combinando áreas multidisciplinares como electrónica, desarrollo móvil/web, IA e IoT" [6].

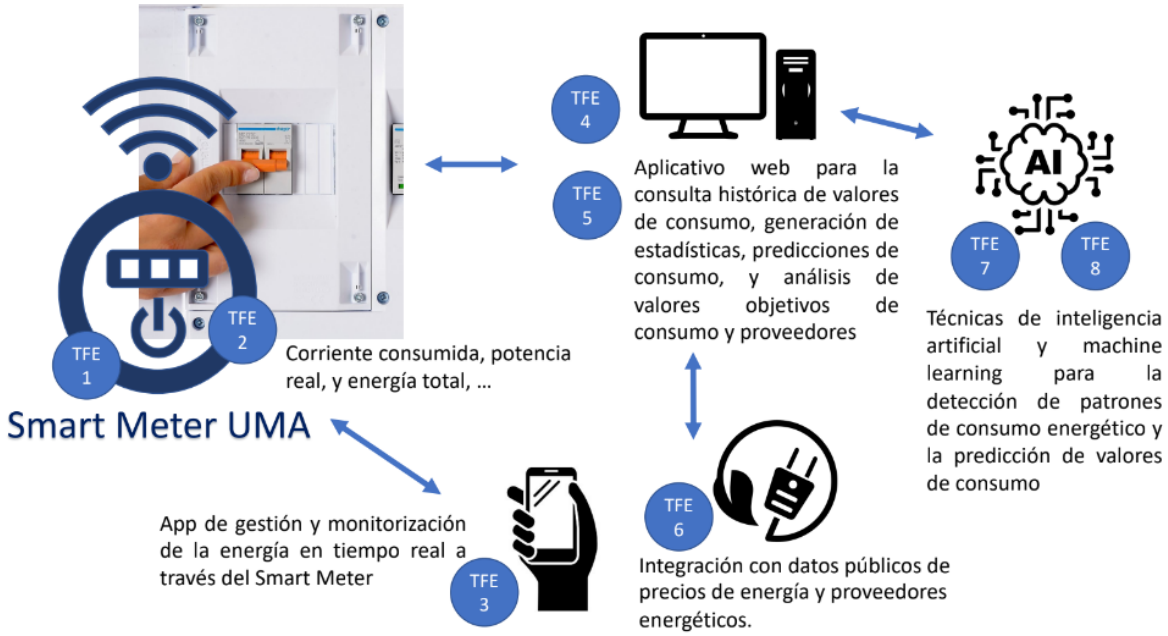


Figura 1.2. Visión general del proyecto.

1.2 Objetivos

El objetivo principal del trabajo será la construcción de un servicio que mediante la aplicación de técnicas de machine learning sea capaz de detectar patrones de consumo energético, para que cuando reciba valores de energía de un contador inteligente pueda devolver una predicción fiable del consumo en un futuro, por ejemplo, en las próximas 24 horas.

Para ello se construirá un sistema entrenado mediante modelos de machine learning que procesen una serie de valores de consumo energéticos reales como entrada y que generen una predicción de consumo de energía para un tiempo futuro determinado. Cuando esté listo y esté optimizado, el servicio podrá ser usado en el proyecto. En primera instancia,

se usarán datasets públicos para el entrenamiento y posteriormente, cuando el Smart Meter del proyecto esté listo, el objetivo será incorporar el servicio como un complemento en el proyecto para ayudar a los usuarios obteniendo predicciones a través de valores reales del Smart Meter.

Para la visualización de los resultados se tendrá acceso a estas predicciones mediante una aplicación web. En este trabajo, a modo de prueba, se podrá predecir desde una fecha concreta los valores energéticos correspondientes a la siguiente hora y a los siguientes 1, 7 o 30 días, es decir, un día, una semana y aproximadamente un mes respectivamente, y luego compararlos con los valores reales para esas fechas.

La aplicación web será desplegada mediante Docker y Kubernetes, y será capaz de realizar peticiones al sistema Tensorflow Serving para obtener las predicciones de los modelos anteriormente construidos. El objetivo será desplegar un sistema con la arquitectura del siguiente diagrama (Figura 1.3).

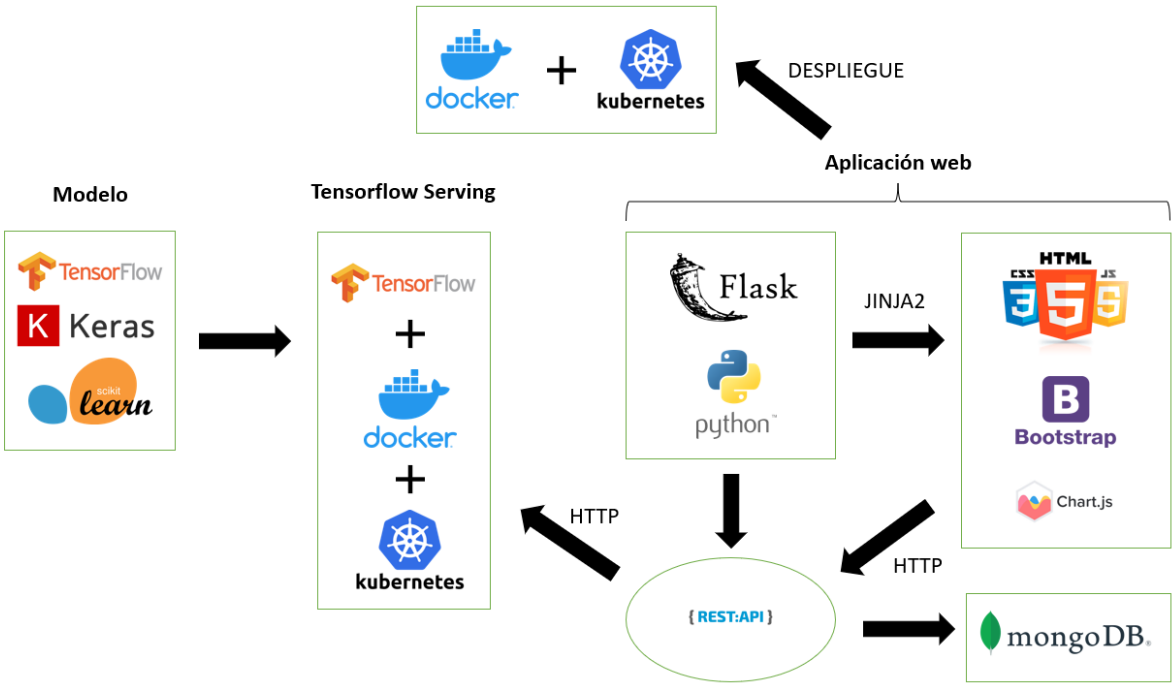


Figura 1.3. Diagrama de la arquitectura del sistema

1.3. Metodología

La metodología es una técnica muy importante en el desarrollo de software, ya que especifica de que forma se va a organizar el trabajo. Seguir una buena metodología es imprescindible para construir software de calidad. Cada proyecto software es un mundo, y hay que pensar muy bien que metodología de trabajo sería adecuada para el buen desarrollo del mismo, según las características del cliente, los entregables, reuniones, entre otros. Actualmente, existen dos tipos de metodologías para el desarrollo de sistemas software: las tradicionales y las ágiles.

Las metodologías tradicionales están centradas en realizar mucha documentación poco flexible respecto a los cambios antes de comenzar con la implementación, por ejemplo, desarrollar una lista de requisitos fija que no cambie en todo el proyecto. Esta metodología está centrada para aquellos proyectos cuya especificación esté muy bien definida o se tenga las ideas muy claras de lo que se tiene que hacer siempre teniendo en cuenta de que se pueda hacer.

Sin embargo, la que se usa en este trabajo es la metodología ágil, que está basada en el desarrollo iterativo e incremental, donde los requisitos y soluciones evolucionan con el tiempo según la necesidad del proyecto. Esta metodología está más enfocada al resultado del producto frente a la documentación. Cada iteración durará entre 2-3 semanas como máximo, coincidiendo con las reuniones del equipo de proyecto que se producirán. El uso de esta metodología proporciona flexibilidad frente a los cambios, que permitirá una buena adaptación a las circunstancias que puedan ocurrir durante el transcurso del trabajo.

1.4. Estructura de la memoria

La estructura de la memoria comienza con el resumen del trabajo y el índice de contenidos, figuras y tablas. El resto de la memoria está organizada en cinco capítulos, incluyendo además las referencias usadas y dos anexos referentes a los pasos de instalación de la aplicación y el manual de usuario. En cuanto a los capítulos, se han dividido de la siguiente forma:

- *Capítulo 1:* Introducción y explicación de la motivación y los objetivos del trabajo, además de la metodología a seguir.
- *Capítulo 2:* Exposición de las tecnologías utilizadas durante el transcurso del trabajo y explicación de los conceptos generales de Machine Learning y su estudio en Tensorflow Keras.
- *Capítulo 3:* Detalles de la especificación y el diseño del sistema tales como requisitos, casos de uso, diagramas de secuencia, modelado IFML o diagrama de despliegue.
- *Capítulo 4:* Desarrollo general de las fases del trabajo pasando desde la extracción, análisis y preprocesamiento de los datos hasta el desarrollo del servicio para la visualización de los resultados del entrenamiento y análisis de modelos.
- *Capítulo 5: Conclusiones obtenidas y líneas futuras a seguir en el trabajo.*

2

Tecnologías y estudio del arte

2.1. Tecnologías utilizadas

2.1.1. Python

Python es un "lenguaje de programación de alto nivel que se utiliza para desarrollar aplicaciones de todo tipo. A diferencia de otros lenguajes como Java o .NET, se trata de un lenguaje interpretado, es decir, que no es necesario compilarlo para ejecutar las aplicaciones escritas en Python, sino que se ejecutan directamente por el ordenador utilizando un programa denominado intérprete, por lo que no es necesario “traducirlo” a lenguaje máquina. Python ha ido ganando adeptos gracias a su sencillez y a sus amplias posibilidades, sobre todo en los últimos años, ya que facilita trabajar con inteligencia artificial, big data, machine learning y data science, entre muchos otros campos en auge" [7] (Figura 2.1).

Figura 2.1. Logo de Python [8].

2.1.2. Tensorflow

TensorFlow es una "biblioteca de código abierto para la computación numérica y Machine Learning a gran escala. TensorFlow reúne una serie de modelos y algoritmos de Machine Learning y Deep Learning y los hace accesibles para poder utilizarlos fácilmente. Utiliza Python para proporcionar una práctica API para crear aplicaciones con el marco de trabajo, a la vez que ejecuta esas aplicaciones en C++ de alto rendimiento" [9].

TensorFlow está capacitado para entrenar y ejecutar redes neuronales profundas para la predicción de series temporales. En este trabajo se utiliza la versión 2 de Tensorflow. Además, de esta misma plataforma se utilizará Tensorflow Serving, para desplegar los modelos en Docker y Kubernetes y luego poder generar desde el backend predicciones (Figura 2.2).

Figura 2.2. Logo de Tensorflow [10].

2.1.3. Keras

Keras es una API de aprendizaje profundo escrita en Python, que se ejecuta sobre la plataforma de aprendizaje automático TensorFlow. "Keras es la API de alto nivel de TensorFlow 2 para resolver problemas de aprendizaje automático que proporciona abstracciones esenciales y bloques de construcción para desarrollar y enviar soluciones de aprendizaje automático con alta velocidad de iteración. Además, se puede exportar los modelos de Keras para ejecutarlos en el navegador o en un dispositivo móvil" [11]. Es muy útil para la construcción de modelos y capas de este trabajo (Figura 2.3).

Figura 2.3. Logo de Keras [12].

2.1.4. Jupyter

Jupyter "es una herramienta de software libre, estándares abiertos y servicios web para la computación interactiva en todos los lenguajes de programación" [13]. En este caso,

se utiliza la extensión de Jupyter Notebook, que es la "aplicación web original para crear y compartir documentos computacionales. Admite más de 40 lenguajes de programación incluidos Python, R, Julia y Scala, son fáciles de compartir, proporcionan salidas interactivas para la visualización de resultados y permiten la integración de grandes datos para big data con pandas, scikit-learn, ggplot2 y TensorFlow" [13].

Los Notebooks de Jupyter se desarrollaran en la infraestructura AdaByron, una plataforma de los grupos del Edificio de Investigación Ada Byron de la UMA, en la cual se ha puesto disponible una GPU habilitada para Keras. En estos notebooks serán donde se desarrollarán toda la parte de machine learning del trabajo: extracción, preprocesado y entrenamiento (Figura 2.4).

Figura 2.4. Logo de Jupyter [14].

2.1.5. Pandas

Pandas es una "librería de Python especializada en el manejo y análisis de estructuras de datos. Las principales características de esta librería son: definir nuevas estructuras de datos basadas en los arrays de la librería Numpy pero con nuevas funcionalidades, leer y escribir fácilmente ficheros en formato CSV, Excel y bases de datos SQL, acceso a los datos mediante índices o nombres para filas y columnas, métodos para reordenar, dividir y combinar conjuntos de datos, trabajos con series temporales y eficiencia" [15] (Figura 2.5).

Figura 2.5. Logo de Pandas [16].

2.1.6. Numpy

NumPy es una "librería de Python especializada en el cálculo numérico y el análisis de datos, especialmente para un gran volumen de datos. Incorpora una nueva clase de objetos llamados arrays que permite representar colecciones de datos de un mismo tipo en varias dimensiones, y funciones muy eficientes para su manipulación" [17]. En este trabajo cumple la función de realizar operaciones y cálculos complejos sobre grandes conjuntos de datos además de construir arrays de forma eficiente (Figura 2.6).

Figura 2.6. Logo de NumPy [18].

2.1.7. Scikit Learn

Scikit Learn es una "librería de Machine Learning en Python que busca ayudar en los principales aspectos a la hora de afrontar un problema de Machine Learning. Más concretamente, Scikit Learn cuenta con funciones para el preprocesamiento de datos, creación de modelos y optimización de hiperparámetros de los modelos" [19]. En este trabajo, se va a utilizar sólo para el preprocesamiento de los datos, realizando la división entre los datos de entrenamiento y validación/test, imputación de valores perdidos y transformación de los datos (Figura 2.7).

Figura 2.7. Logo Scikit-Learn [20].

2.1.8. Matplotlib

Matplotlib es una "librería de Python especializada en la creación de gráficos en dos dimensiones. Permite crear y personalizar los tipos de gráficos más comunes como diagramas de barras, histogramas, diagramas de lineas, diagramas de sectores y muchos más, incluso

realizar combinaciones entre ellos" [21]. Se utilizará para el análisis de datos, visualización de resultados de entrenamiento y comparación de resultados (Figura 2.8).

Figura 2.8. Logo de Matplotlib [22].

2.1.9. Seaborn

Seaborn es una "biblioteca de visualización de datos de Python basada en matplotlib. Proporciona una interfaz de alto nivel para dibujar gráficos estadísticos atractivos e informativos" [23]. En el trabajo se utiliza para hacer más atractivos los gráficos dibujados con Matplotlib (Figura 2.9).

Figura 2.9. Logo de Seaborn [24].

2.1.10. Github

GitHub es una "plataforma de desarrollo colaborativo para alojar proyectos utilizando el sistema de control de versiones Git. Se utiliza principalmente para la creación de código fuente de programas de ordenador" [25]. En este caso, se ha creado una organización de GitHub para alojar el proyecto al completo donde cada componente crea su repositorio específico de su trabajo [26] (Figura 2.10).

Figura 2.10. Logo de GitHub [27].

2.1.11. Docker + Kubernetes

Docker es un "proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software, proporcionando una capa adicional de abstracción y automatización de virtualización de aplicaciones en múltiples sistemas operativos" [28]. Su función en este trabajo es crear un contenedor que agrupe el backend y otro para Tensorflow Serving y disponibilizarlo a través de imágenes (Figura 2.11).

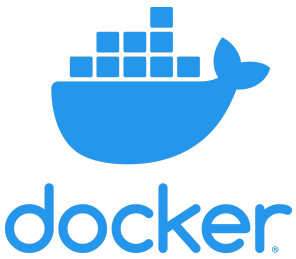


Figura 2.11. Logo de Docker [29].

Kubernetes es una "plataforma portable y extensible de código abierto para administrar cargas de trabajo y servicios. Kubernetes facilita la automatización y la configuración declarativa. Tiene un ecosistema grande y en rápido crecimiento" [30]. En el trabajo se utiliza para desplegar en local las imagenes de Docker que se generan para ejecutarlas y hacerlas accesibles (Figura 2.12).

Figura 2.12. Logo de Kubernetes [31].

2.1.12. Visual Studio Code

Visual Studio Code es un "editor de código fuente independiente que se ejecuta en Windows, macOS y Linux, y la elección principal para desarrolladores web y JavaScript, con extensiones para admitir casi cualquier lenguaje de programación" [32]. En esta herramienta se lleva a cabo el desarrollo web para la visualización y comprobación de los resultados de predicción (Figura 2.13).

Figura 2.13. Logo de Visual Studio Code [33].

2.1.13. Flask y Jinja 2

Flask es un "framework minimalista escrito en Python que permite crear aplicaciones web rápidamente y con un mínimo número de líneas de código. Está basado en la especificación WSGI de Werkzeug y el motor de templates Jinja2 y tiene una licencia BSD" [34].

Jinja2 es un "motor de plantillas rápido, expresivo y extensible. Los marcadores de posición especiales en la plantilla permiten escribir código similar a la sintaxis de Python. Luego, se pasan datos a la plantilla para representar el documento final. Sus características más relevantes, entre otras, son la herencia e inclusión de plantillas y la definición e importación de macros dentro de plantillas" [35]. Las archivos HTML del desarrollo web de este trabajo usa este motor de plantillas específico (Figura 2.14).

Figura 2.14. Logo de Flask [36].

Figura 2.15. Logo de Jinja2 [37].

2.1.14. HTML, CSS, JavaScript, Bootstrap, Ajax y Chart js

"HTML (Lenguaje de Marcas de Hipertexto, del inglés HyperText Markup Language) es el componente más básico de la Web. Define el significado y la estructura del contenido web" [38]. La versión específica del trabajo es HTML5 (Figura 2.16).

"Hojas de Estilo en Cascada (del inglés Cascading Style Sheets) o CSS es el lenguaje de estilos utilizado para describir la presentación de documentos HTML. CSS describe como debe ser renderizado el elemento estructurado en la pantalla" [39]. La versión utilizada es CSS3 (Figura 2.16).

JavaScript (JS) es un "lenguaje de programación ligero, interpretado, o compilado justo-a-tiempo (just-in-time) con funciones de primera clase" [40]. Se usa para el comportamiento dinámico de la aplicación web (Figura 2.16).

Figura 2.16. Logo de HTML5, CSS3 y JavaScript [41].

Bootstrap es una "biblioteca multiplataforma o conjunto de herramientas de código abierto para diseño de sitios y aplicaciones web. Contiene plantillas de diseño con tipografía, formularios, botones, cuadros, menús de navegación y otros elementos de diseño basado en HTML y CSS, así como extensiones de JavaScript adicionales" [42]. A diferencia de muchos frameworks web, solo se ocupa del desarrollo front-end, que será su funcionalidad en este trabajo (Figura 2.17).

Figura 2.17. Logo de Bootstrap [43].

"JavaScript Asíncrono + XML (AJAX) no es una tecnología por sí misma, sino un término que describe un nuevo modo de utilizar conjuntamente varias tecnologías existentes. Hace posible que las aplicaciones web sean capaces de actualizarse continuamente sin tener que volver a cargar la página completa. Esto crea aplicaciones más rápidas y con mejor respuesta a las acciones del usuario. Se utiliza para las peticiones asíncronas en la visualización web de los resultados de las predicciones" [44] (Figura 2.18).

Figura 2.18. Logo de AJAX [45].

Chart.js es una "biblioteca JavaScript gratuita de código abierto para la visualización de datos, que admite 8 tipos de gráficos: barras, linear, área, circular, burbuja, radar, polar y dispersión" [46]. Se utiliza para mostrar los gráficos en el servicio web de visualización de resultados (Figura 2.19).

Figura 2.19. Logo de Chart.js [47].

2.1.15. MagicDraw

MagicDraw es una herramienta de modelado UML que sirve en general para hacer todo tipos de modelos necesarios para especificar una solución software o hardware. En este trabajo se va a utilizar para realizar los diagramas de casos de uso, los diagramas de secuencia y el diagrama de despliegue de la aplicación (Figura 2.20).

Figura 2.20. Logo de MagicDraw [48].

2.1.16. MongoDB

MongoDB es una base de datos de tipo no relacional que alberga colecciones en formato JSON. MongoDB Atlas es la plataforma web para crear colecciones de MongoDB online mediante clusters. Esta herramienta se utilizará para alojar los datos del consumo y se extraerán desde el backend con la librería de python de PyMongo (Figura 2.21).

Figura 2.21. Logo de MongoDB [49].

2.1.17. IFMLEdit

IFMLEdit es una herramienta de modelado web IFML (Lenguaje de modelado de flujo de interacción) que se utiliza para realizar un prototipo básico de la navegación y los componentes de la aplicación web. Se utilizará para realizar un primer diseño de la navegación y los componentes de la aplicación web (Figura 2.22).

Figura 2.22. Logo de IFML [50].

2.2. Estudio del estado del arte

2.2.1. Conceptos básicos de machine learning

Una vez definido qué es el machine learning, es esencial entender los conceptos básicos que lo componen. En su definición se comenta que esta rama aprende de los seres humanos a través de un conjunto de datos (dataset) y algoritmos. En cuanto a los algoritmos, existen tres tipos, el aprendizaje supervisado, el aprendizaje no supervisado y el aprendizaje por refuerzo. Este proyecto se basa en la predicción mediante aprendizaje supervisado sobre redes neuronales.

El aprendizaje supervisado es una técnica que obtiene información a partir de la proporcionada por el conjunto de datos de entrenamiento que muestra la relación entre la entrada y la salida esperada. Existen dos tipos, el de clasificación, que intenta etiquetar a partir de una entrada esperada, por ejemplo, una clasificación de imágenes de gatos y perros, y por otro lado el aprendizaje supervisado por regresión, que trata de predecir valores concretos a partir de un rango de valores posibles.

El entrenamiento se realiza sobre modelos que aplican deep learning o aprendizaje profundo. Deep learning es un "método específico de machine learning que incorpora las redes neuronales en capas sucesivas para aprender de los datos de manera iterativa. Deep learning es especialmente útil cuando trata de aprender patrones de datos no estructurados. Las redes neuronales complejas de deep learning están diseñadas para emular el funcionamiento del cerebro humano, con la finalidad de entrenar a los ordenadores para que puedan afrontar problemas y abstracciones mal definidas" [51].

Una red neuronal es un conjunto de capas divididas en una capa de entrada, una capa de salida y varias capas ocultas entre ellas compuestas por neuronas (Figura 2.23).

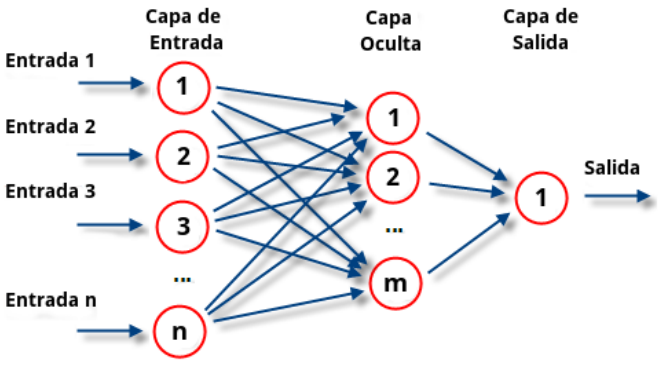


Figura 2.23. Esquema de red neuronal [52].

Un modelo de machine learning es la "salida que se genera cuando se entrena un algoritmo de machine learning con datos. Después del entrenamiento, cuando se proporcione una entrada a un modelo, este dará una salida. Por ejemplo, el entrenamiento de un algoritmo predictivo creará un modelo predictivo. A continuación, cuando se proporcione datos al modelo predictivo, se recibirá una predicción basada en los datos que han entrenado el modelo" [51]. El modelo agrupa los algoritmos y la red neuronal que se ha mencionado antes.

Una parte importante del aprendizaje es el descenso del gradiente. El método del descenso del gradiente es un "algoritmo de optimización que permite converger hacia el valor mínimo de una función mediante un proceso iterativo. En aprendizaje automático básicamente se utiliza para minimizar una función que mide el error de predicción del modelo en el conjunto de datos" [53]. Si se observa la figura 2.24, el objetivo es conseguir llegar a un mínimo global desde el punto más alto. Hay que tener cuidado con ello por que puede suponer un problema, ya que si el descenso se produce mediante pasos muy grandes, puede que el algoritmo no llegue a su objetivo nunca, y si los pasos son muy pequeños puede que tarde demasiado en llegar a un objetivo claro o se quede en un mínimo local. Normalmente esto se regula con el número de lotes de datos de entrada del modelo, mientras el lote sea más grande, mayor serán los pasos.

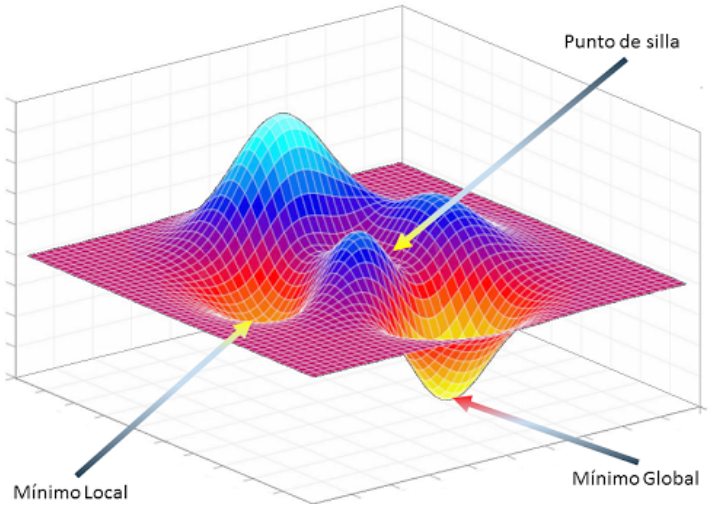


Figura 2.24. Representación gráfica de mínimos para el vector del gradiente [54].

Otros de los problemas más comunes del machine learning son el overfitting y el underfitting. Primero hay que enteder que son el bias y la varianza. Cuando se entrena un modelo, se establece un objetivo que es lo que se espera que el modelo llegue a conseguir, por ejemplo un 95% de fiabilidad en la predicción. El bias es la diferencia entre el error del entrenamiento frente al error objetivo, por ejemplo, el modelo consigue un 75% de fiabilidad, por lo que tiene un 25% de error, entonces el bias es ese 25% menos el 5% del error objetivo. La varianza es la diferencia entre el error de entrenamiento y el error de validación/test, por lo que siguiendo con el mismo ejemplo, si para ese modelo se obtiene un 60% de fiabilidad, la varianza sería el 40% de error de la validación/test menos el 25% de error de entrenamiento.

El overfitting se produce por un sobreajuste del modelo a los datos de entrenamiento y se puede deducir por la varianza. Si en el entrenamiento se produce una varianza muy alta, significa que el modelo se esta adaptando a los datos de entrenamiento y no generaliza bien con buenas predicciones para otros datos diferentes de los de entrenamiento (Figura 2.25).

El underfitting se produce por una insuficiente cantidad o variedad de datos de entrenamiento (Figura 2.25). Por ejemplo, si se entrena un modelo para clasificar imágenes de peces, si sólo se entrena con una especie de pez, no será capaz de predecir imágenes de otros tipos de peces.

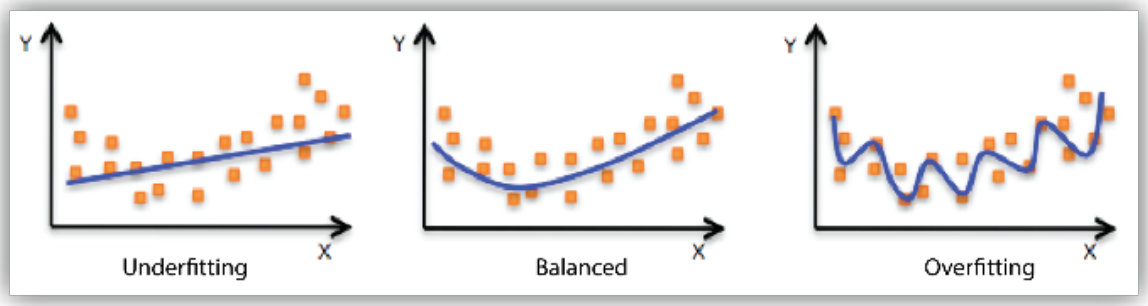


Figura 2.25. Ejemplos gráficos de underfitting, balanceado y overfitting [55].

2.2.2. Machine Learning en Tensorflow Keras.

2.2.2.1. Modelos

Tensorflow Keras proporciona una API para construir modelos de deep learning. Existen 3 tipos, Functional, Model Sub-Classing y Sequential, pero este último es el más popular y el que se usará en todos los ejemplos, y lo que hace es agrupar una pila lineal de capas que se comunican secuencialmente.

2.2.2.2. Capas

Existen muchos tipos de capas, pero aquí sólo se va a mencionar las que se han utilizado en el proyecto para predicción de series temporales. Todas las capas tienen asociada una función de activación para la salida de los datos de la propia capa, entre las que destacan la de tangente, relu, sigmoide y softmax. Los tipos utilizados han sido los siguientes:

- Input: Es la capa de entrada del modelo con una forma definida consistente con los datos de entrada.
- Dense: Es una de las capas primarias en deep learning que se encuentra totalmente conectada con todas las salidas de la capa anterior y lo que hace es realizar una serie de transformaciones a los datos de entrada.
- LSTM: Es una de las capas recurrentes que ofrece keras que tiene memoria a largo y a corto plazo. Tiene un comportamiento complejo, pero a modo de resumen, en cada iteración suele tener referencias de memoria a corto y largo plazo. Es una buena opción para deducir las relaciones de una series temporal y obtener el patrón de consumo.

- GRU: GRU es otra capa recurrente que está diseñada para capturar mejor las dependencias a largo plazo. A diferencia de LSTM, suele tener problemas en el corto plazo.
- Conv1D: Realiza una convolución de 1 dimensión a los datos de entrada.
- ConvLSTM1D: Combina la capa de convolución de 1 dimensión con LSTM.
- ConvLSTM2D: Combina la capa de convolución de 2 dimensiones con LSTM.
- Bidirectional: Es una capa que recorre la red neuronal tanto hacia delante como hacia atrás.
- TimeDistributed: Es una capa envoltorio de otras capas que aplica la capa que envuelve de forma iterativa una vez por cada paso de tiempo establecido.
- MaxPooling: Realiza una reducción agrupando la muestra que recibe.
- Flatten: Aplana el array que recibe para quitar dimensiones.
- RepeatVector: Devuelve un vector de interpretación de la entrada que recibe.

2.2.2.3. Compilado

La compilación del modelo es un paso anterior al entrenamiento con el propósito de detectar cualquier fallo en la arquitectura de la red y definir los argumentos principales de optimización, pérdida y métricas para el siguiente paso.

- Optimizers (Optimizadores): Prácticamente todos los optimizadores de Tensorflow Keras están basados en el gradiente, pero de ellos, Adam será el que se va a usar debido a las ventajas computacionales y de eficiencia en memoria que proporciona. Además es sensible a los hiperparámetros lo que lo hace un optimizador adecuado para el aprendizaje profundo [56].
- Loss (Pérdida): Es la función que evalua cuál es el error que está teniedo el modelo respecto a las predicciones y toma decisiones según sea este valor. En general, en los problemas de regresión, lo más comun es usar el MSE (Error Cuadrático Medio), que es la diferencia entre el valor predecido y el valor real elevado al cuadrado, o el MAE (Error Absoluto Medio), que es el mismo error pero en valor absoluto sin elevar al cuadrado, aunque es preferible usar el primero porque mide la dispersión del error y suele ser más alto.

- Metrics (Métricas): Las métricas, aunque no afectan al entrenamiento, se utilizan para que uno mismo pueda evaluar el rendimiento del modelo durante el mismo. En este caso, como es un problema de regresión, se utilizará el MAE, MSE y SMAPE. Este último es una variedad del MAPE (Porcentaje de error absoluto medio), con la diferencia de que el SMAPE es simétrico y está comprendido entre el rango 0-200%, y está adaptado para usar valores pequeños como los del dataset utilizado y en cambio el MAPE falla cuando trata valores muy pequeños.

2.2.2.4. Entrenamiento

El entrenamiento del modelo es el paso que realiza iteraciones sobre los conjuntos de entrada y salida. En TensorflowKeras, cuando se va a proceder a entrenar un modelo hay que especificar una serie de parámetros en la función. Existen muchos parámetros que se pueden pasar pero en este proyecto se van a utilizar los siguientes:

- X : Representa la característica de entrada que se le pasa al modelo cuando tiene que predecir un cierto valor. Se suele recomendar que se escale la entrada para un mejor entrenamiento, que consiste es establecer los valores en un rango determinado, por ejemplo entre el 0 y el 1.
- Y : Representa los valores que tiene que predecir el modelo para las anteriores entradas.
- batch_size: Es el tamaño del lote de entrada de datos al modelo. Este parámetro influye en el descenso del gradiente ya que representa el número de pasos que anteriormente se comentaba.
- epochs: Simboliza el número de iteraciones del entrenamiento del modelo. El número de épocas puede derivarse en un problema de overfitting por lo que generalmente se recomienda empezar con un número de épocas bajo e ir incrementando hasta encontrar un número bueno de iteraciones.
- validation_data: Es una tupla con datos de entrada y datos de salida y representa los datos de validación del modelo. En cada época se comprueba cuál es el error para esos datos, para comprobar si el modelo está generalizando bien o si se está sobreajustando a los datos de entrenamiento.

- callbacks: Son funciones que se aplican en cada época de entrenamiento pero no afectan directamente al modelo, como por ejemplo, para guardar el modelo en una época concreta o para terminar el entrenamiento. EarlyStopping es el callback que se va a utilizar para evitar el problema de las épocas descrito anteriormente. Este método lo que hace es que después de las épocas que se le especifica en el parámetro 'patience', que significa ten una paciencia de un número de épocas específico, puede parar y terminar el entrenamiento si ve que no está mejorando según los objetivos que se le marque. Por ejemplo, se le podría especificar que busque reducir al mínimo el error de validación, y cuando observa que el modelo no puede o en vez de reducirlo lo sube, termina automáticamente el entrenamiento.

2.2.2.5. Optimización de modelos

Una vez que los modelos han pasado un primer proceso de entrenamiento, el siguiente objetivo es mejorar los resultados que ofrecen. Uno de los principales problemas que se da en la mayoría de problemas de machine learning es el overfitting o tener un bias demasiado alto. Normalmente para reducir el bias se suele ampliar el número de neuronas o de capas de la red, pero puede provocar un mayor overfitting al ajustarse más el modelo a los datos de entrenamiento. Para evitarlo, se pueden hacer dos cosas, añadir más datos de entrenamiento, o en el caso que no se disponga, añadir regularización. Esto último puede provocar un incremento del bias. Existen varias técnicas de regularización, pero en este trabajo sólo se van a aplicar dos ténicas, dropout y regularización L1 y L2.

Dropout es una técnica de regularización que consiste en el abandono de neuronas de la red neuronal. Esto significa que aleatoriamente se desactiva un porcentaje de neuronas en cada época de entrenamiento, con el objetivo de que las neuronas no memoricen parte de la entrada, provocando un sobreajuste en el entrenamiento.

La regularización L2 tiene como objetivo reducir el valor de los parámetros elevándolos al cuadrado y multiplicándolos por una constante que debe ser pequeña porque si no provocará que los parámetros se aproximen mucho al 0 y puede ser un problema. La regularización L1 es igual sólo que en vez de elevarlos al cuadrado realiza el valor absoluto, pero el objetivo es el mismo.

Este tipo de regularizaciones en Keras se pueden aplicar añadiendo una capa de estas características al modelo o especificándolo en los parámetros de las demás capas.

3

Especificación y diseño

3.1. Especificación del sistema

3.1.1. Requisitos funcionales

RF1. El usuario podrá acceder y navegar por las distintas secciones de la aplicación.

RF2. El usuario podrá solicitar varios tipos de predicciones de valores de energía eléctrica a partir de una fecha determinada. **NOTA:** Concretamente, se podrá predecir la siguiente hora, día, semana o mes.

RF3. El usuario podrá visualizar los resultados de las predicciones y compararlos con los valores reales. **NOTA:** Los resultados mostrarán una gráfica lineal comparando la predicción con los valores reales y el error numérico de la predicción.

RF3.1. El usuario podrá comparar valores uno a uno en la gráfica.

3.1.2. Requisitos no funcionales

RNF1. Los datos se almacenarán en una base de datos no relacional (MongoDB) mediante archivos BSON.

RNF2. El backend delL sistema se realizará con el framework Flask.

RNF3. El frontend del sistema se realizará usando HTML, CSS, JavaScript y Bootstrap usando el motor de plantillas Jinja2.

RNF4. El sistema será una aplicación disponible para los navegadores web.

RNF5. La interfaz seguirá una estructura consistente en toda la plataforma para asegurar la estética general con una paleta de colores predefinida.

RFN6. El sistema debe ser intuitivo y accesible para cualquier tipo de usuario como para poderla manejar con soltura en un tiempo de aprendizaje de entre 5 y 15 minutos.

RNF7. El sistema deberá optimizar los tiempos de carga y descarga de archivos.

RNF8. El sistema será desplegado mediante Docker y Kubernetes.

RNF9. El sistema realizará la consulta de datos mediante una API REST con formato JSON.

3.2. Casos de uso

3.2.1. Diagrama de casos de uso

El trabajo contará con dos casos de uso que agruparan los requisitos principales de la aplicación (Figura 3.1).

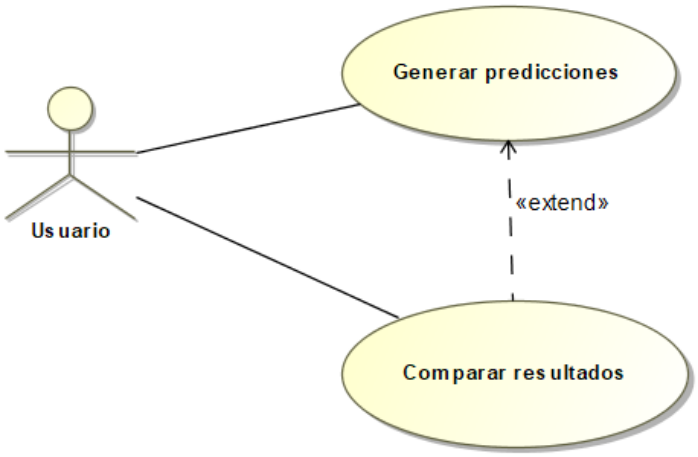


Figura 3.1. Diagrama de casos de uso de la aplicación

3.2.2. Descripción de casos de uso

En las tablas 3.1 y 3.2 se muestra una descripción del caso de uso detallado con su escenario principal y los posibles escenarios alternativos.

ID	RF1, RF-2
Título	Generar predicciones
Descripción	El usuario podrá generar predicciones del tipo que seleccione a partir de la fecha que elija.
Pre-condición	
Post-condición	
Prioridad	Alta
Escenario principal	
1. El usuario accede al servicio web. 2. El sistema muestra la interfaz web y un formulario para generar predicciones. 3. El usuario selecciona el tipo y la fecha y pulsa el botón "predecir". 4. El usuario pulsa el botón "predecir". 5. El sistema comprueba que el tipo y la fecha seleccionada es válida. 6. El sistema realiza una petición POST a Tensorflow Serving y obtiene las predicciones y obtiene de la base de datos los valores reales. 7. El sistema muestra los resultados.	
Escenario alternativo	
3.b. El usuario no rellena el tipo o la fecha y pulsa el botón "predecir". 4.b. El sistema devuelve un error pidiendo que se completen los campos. 4.c. El sistema comprueba que la fecha no es válida porque no existen suficientes valores anteriores para generar la predicción o no existen valores reales para realizar la comprobación. 5.c. El sistema muestra un error pidiendo que se seleccione una fecha posterior.	

Tabla 3.1. Descripción del caso de uso de "Generar predicciones".

ID	RF-3, RF-3.1
Título	Comparar resultados
Descripción	El usuario podrá visualizar la comparación entre la predicción y los valores reales junto a su error de la predicción que ha generado.
Pre-condición	El usuario ha accedido al servicio web. El usuario ha seleccionado el tipo de predicción y una fecha válida y ha generado las predicciones.
Post-condición	
Prioridad	Alta
Escenario principal	
1. El sistema muestra un gráfico de la predicción frente a los valores reales y el error. 2. El usuario coloca el ratón encima de cualquier punto de la gráfica. 3. El sistema muestra el valor real y la predicción para ese punto.	
Escenario alternativo	

Tabla 3.2. Descripción del caso de uso de "Comparar resultados".

3.2.3. Diagramas de secuencia

- Caso de uso "Generar predicciones":
    - Escenario de éxito:

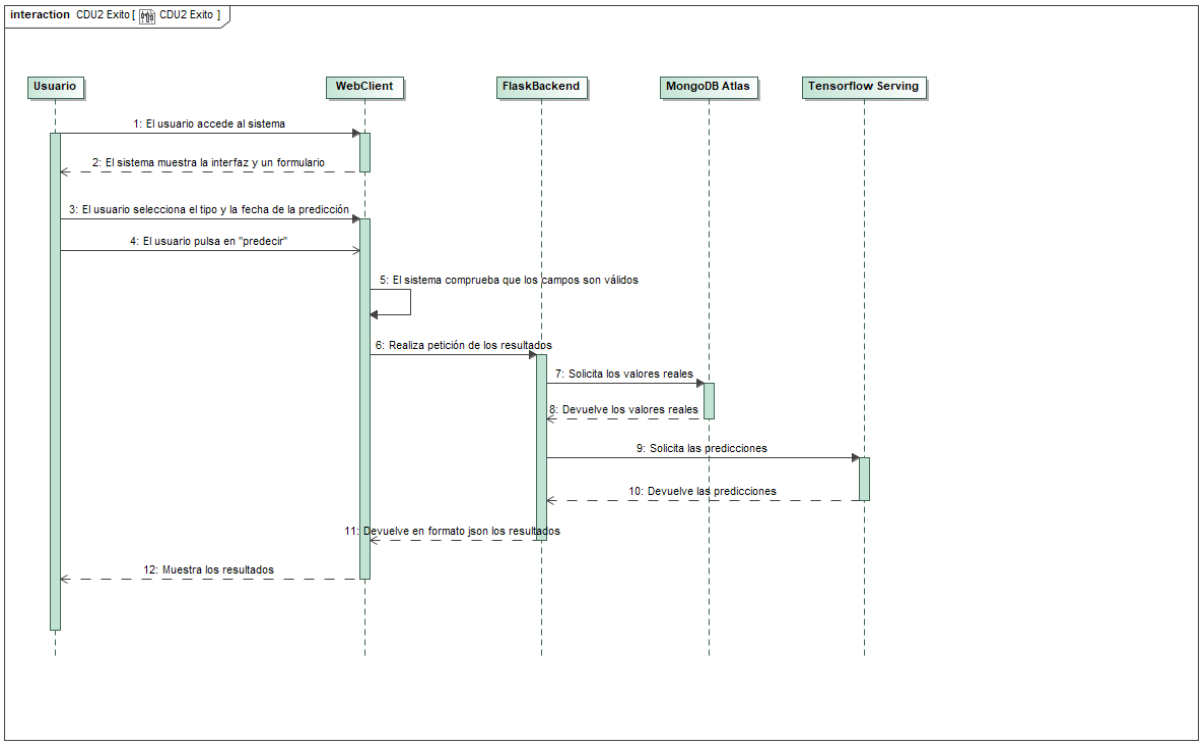


Figura 3.2. Escenario principal caso de uso "Generar predicciones".

- Escenario alternativo 1:

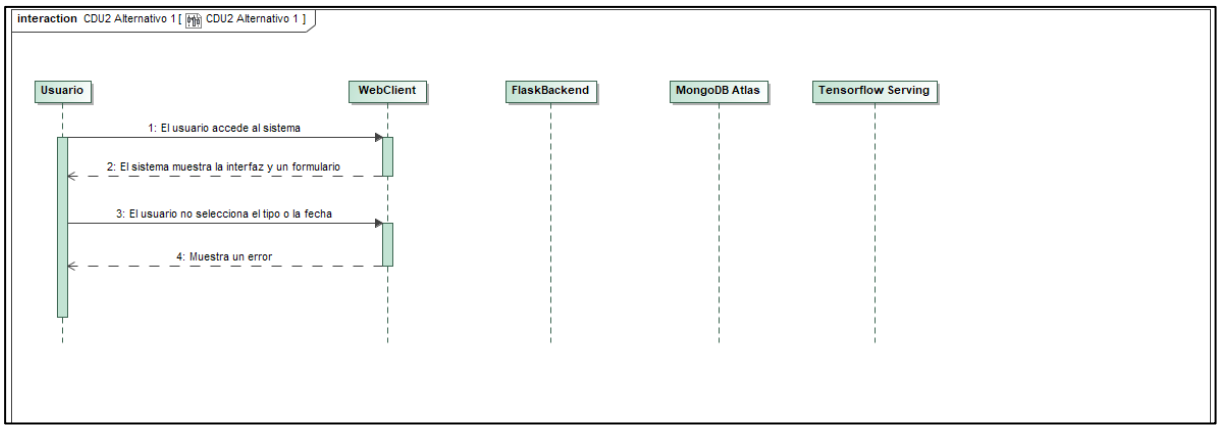


Figura 3.3. Escenario alternativo 1 caso de uso "Generar predicciones".

- Escenario alternativo 2:

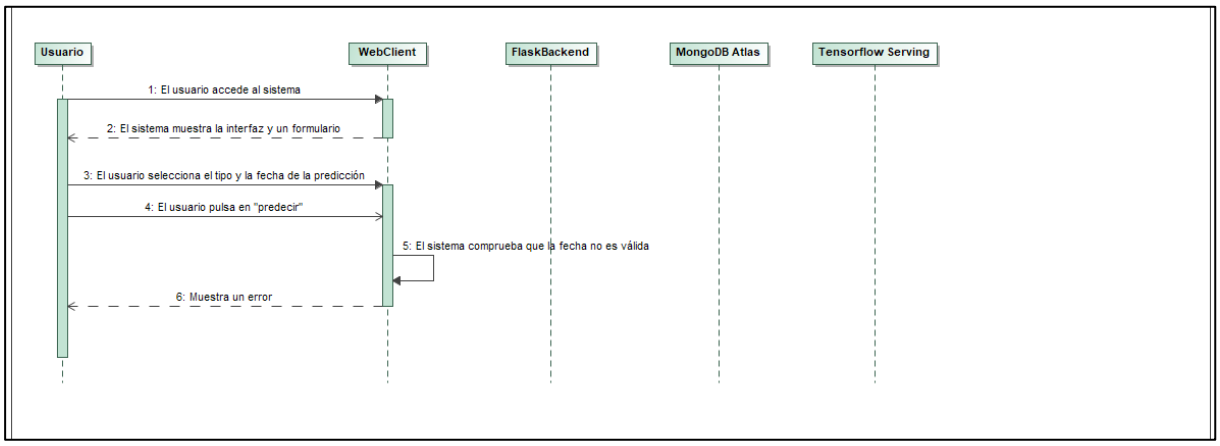


Figura 3.4. Escenario alternativo 2 caso de uso "Generar predicciones".

- Caso de uso "Comparar resultados"
  - Escenario de éxito:

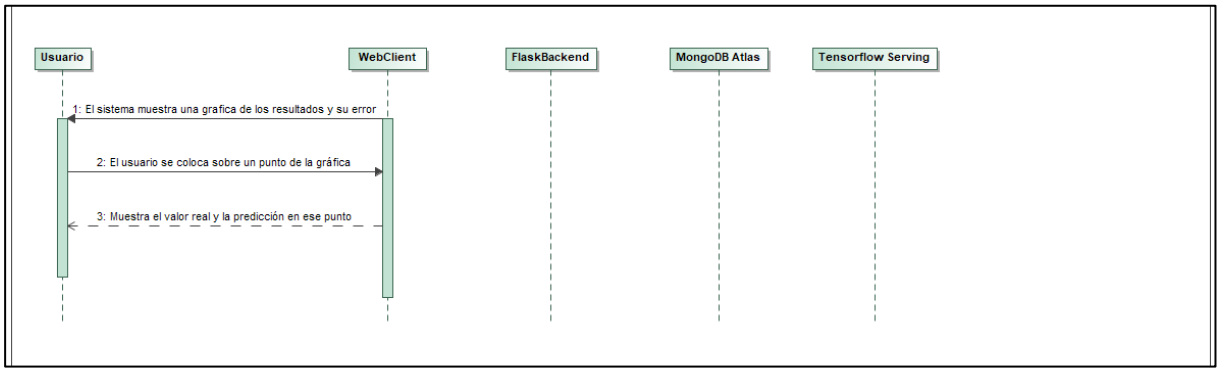


Figura 3.5. Escenario principal caso de uso "Comparar resultados".

3.3. Modelo IFML

La navegación del frontend web consistirá en una sóla ventana (Figura 3.6) con un formulario donde se especificará el tipo de predicción (hora, día, semana, mes) y la fecha desde la que comenzarían las predicciones (Figura 3.7). Una vez se envíe el formulario, siempre y cuando se rellenen bien los campos, se generarán las predicciones y se mostrarán en la parte de abajo de la ventana mediante un gráfico que compare las predicciones generadas frente a los valores reales que debería predecir, y también un análisis del error describiendo el MAE, MSE, RMSE y SMAPE.

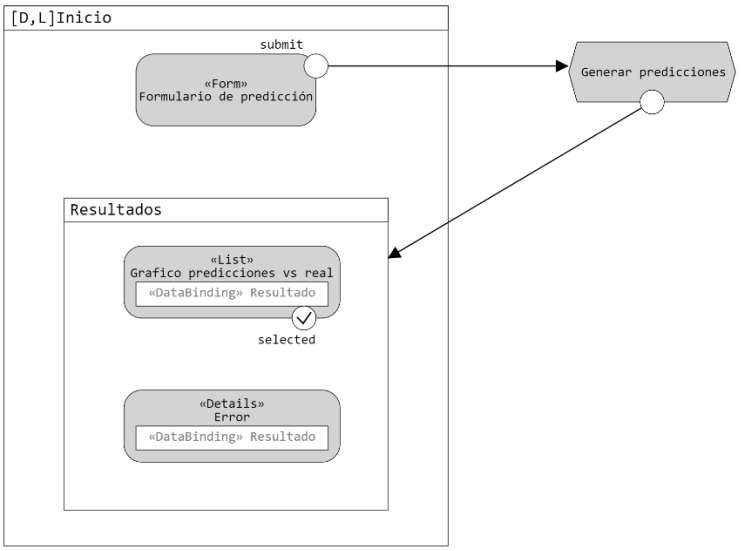


Figura 3.6. Modelo de navegación web (IFML).

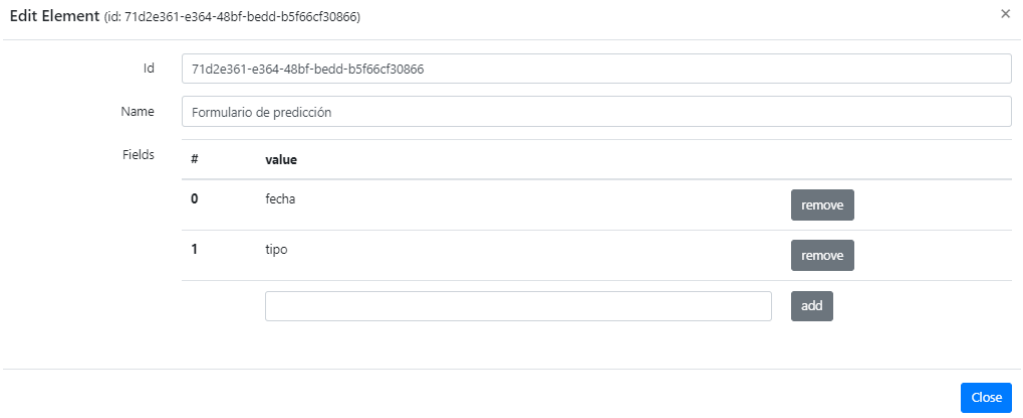


Figura 3.7. Descripción de los elementos que contendrá el formulario

3.4. Diagrama de despliegue

La aplicación web constará de un de un backend en Python y el framework Flask junto a un frontend HTML que realice peticiones mediante AJAX al backend. Esta aplicación se contendrá en un contenedor de Docker que luego será desplegado en Kubernetes. Además, para que el funcionamiento sea similar al que tendría en el proyecto de Smart Meter, se han alojado los datos que se van a usar en MongoDB Atlas para poder acceder a ellos desde el backend y realizar consultas mediante la librería PyMongo. Finalmente, para generar las predicciones, hay que utilizar Tensorflow Serving y Docker para crear una imagen de los modelos que se han creado y luego desplegarlas en Kubernetes, y mediante una API REST proporcionada por Tensorflow Serving, el servicio de la aplicación web podrá conectarse a este servicio generado y solicitar predicciones de los modelos que se han desplegado. El despliegue de la aplicación resultaría en el siguiente diagrama (Figura 3.8):

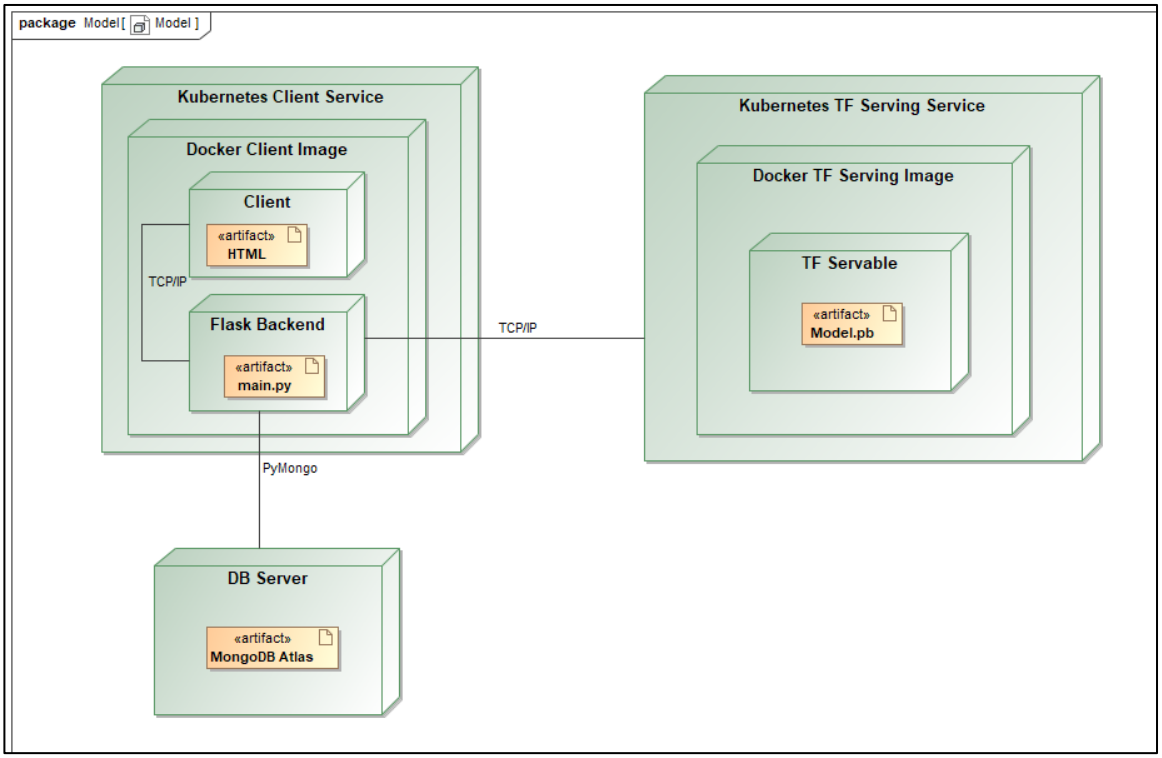


Figura 3.8. Diagrama de despliegue de la aplicación.

4

Fases de desarrollo

4.1 Formación

El machine learning es un mundo complejo de entender para aquellos que no hayan tenido un contacto previo con él, por lo que se requiere una formación inicial en todos los conceptos importantes cómo pueden ser modelos, capas, redes neuronales, overfitting, underfitting, preprocesamiento, etc. Además, se requiere conocimiento previo de la librería utilizada, Tensorflow, por lo que se necesita también de una formación.

Por parte de los tutores se han proporcionado dos libros, *Machine Learning Yearning de Adrew NG*, que ofrece una serie de recomendaciones y soluciones a la hora de implementar modelos de machine learning [57], y *Machine Learning with Python*, donde se ha recomendado la lectura del capítulo 6 *Deep Learning for text and sequences*, que explica conceptos teóricos y prácticos sobre la predicción de series temporales en Python [58].

En cuanto a Tensorflow, se ha dispuesto de una serie de guías e instructivos en la página web original de la biblioteca para entender su funcionalidad básica [59], [60]. Además, se ha realizado el curso *Complete Tensorflow 2 and Keras Deep Learning* de la plataforma Udemy [61], para profundizar en los tipos de modelos de predicción de Tensorflow.

4.2. Búsqueda de datasets de variables y preprocesamiento

4.2.1. Búsqueda y extracción de datos

El primer paso hacia la predicción del consumo eléctrico es la búsqueda de datasets para el entrenamiento. En este caso, el dispositivo Smart Meter no se encontraba disponible para la recogida de datos, por lo que inicialmente el modelo sería entrenado con datasets reales obtenidos en repositorios web.

Una vez analizados los distintos datasets disponibles, se ha escogido el proporcionado por el repositorio Figshare que registra el consumo eléctrico de hogares residenciales en Uruguay. En él se encuentran disponibles 3 tipos de datos diferenciados por el prefijo del fichero según las necesidades del modelo que son el consumo total de la casa (THC), el consumo del calentador eléctrico de agua (EWH) y el consumo específico de los electrodomésticos (DEC). Además se proporcionan otros ficheros correspondientes a la descripción de los clientes o electrodomésticos [62], [63].

Para este modelo se han seleccionado los ficheros de THC que registran el consumo total en kWh cada 15 min desde enero de 2019 hasta noviembre de 2020 de hogares residenciales en Uruguay. En concreto, se han usado los ficheros comprimidos del repositorio con el nombre *THC-consumption_data_YYYYMM.csv.tar.gz*, donde *YYYY* representa el año y *MM* el mes de los registros. Si se analiza el conjunto de datos seleccionado, se observan 3 campos, el campo 'datetime' que representa la fecha del registro en formato timestamp, 'id' que representa a que cliente pertenece la medición, y por último el campo 'value' que muestra los kWh consumidos por dicho cliente en el datetime correspondiente. Además, se puede deducir que si aparecen mediciones de distintos clientes, es muy probable que para una misma fecha haya varios registros por lo que no se podría crear una serie temporal correctamente.

Para solucionar lo anterior, se procede a extraer los datos de un sólo cliente mediante un notebook de Jupyter llamado "extract-data.ipynb", que recorre cada uno de los ficheros comprimidos del repositorio, en total 12 de 2019 y 11 de 2020, y extrae los datos de, por ejemplo, el id 17, y los guarda en un csv llamado "THC-id-17-original.csv" pasando las fechas al formato "YYYY-MM-DD hh:mm:ss" para facilitar su análisis. La extracción de los datos de id

17 tiene como resultado 64456 registros cada 15 min, desde el 01-01-2019 a las 03:00:00 hasta el 03-11-2020 a las 23:00:00 (Figura 4.2).

Otro dato que habría que tener en cuenta es que los datos provienen de Uruguay, y aunque tenga un clima parecido al de España, es cierto que allí es invierno cuando en España es verano, por lo que el consumo varía según la fecha. No obstante, el objetivo principal es encontrar un modelo que sea capaz de adaptarse a los picos de consumo eléctrico, y en un futuro probablemente no se usarán estos datos o incluso podrían adaptarse perfectamente simplemente invirtiendo la fecha, por ejemplo, si la fecha de España es de enero, la entrada al modelo podría ser como si fuera de julio.

4.2.2. Análisis y preprocesamiento de datos

El primer paso para el preprocesamiento del dataset es cargar el fichero 'THC-id-17.csv' generado en un objeto DataFrame de Pandas, donde se pueden almacenar y preprocesar los datos. Una vez realizada la extracción de los datos, el siguiente paso es el análisis de ellos. Para ello, el objeto Dataframe ofrece una función .describe() para su análisis estadístico (Figura 4.2). Lo más importante que podemos sacar de esta función es que las mediciones oscilan entre 0 y 0.676, la media es 0.056 aproximadamente, la mediana es 0.03 y el 75% de los datos se encuentra entre 0 y 0.052. Esto demuestra que existen valores atípicos en los datos que pueden representar un elevado consumo de electridad en un momento específico.

	value
count	64456.000000
mean	0.056218
std	0.083212
min	0.000000
25%	0.016000
50%	0.031000
75%	0.052000
95%	0.241000
max	0.676000

Figura 4.1. Estadísticas del dataset.

Además, con el método isna() se puede comprobar que no existe ningún valor a nulo en los datos por lo que no se requiere un preprocesamiento previo para este tipo de valores. Sin embargo, se comprueba que en algunos periodos no existen mediciones (por ejemplo el

día 11-12-2019 desde las 9:15 hasta las 11:15), y que quizás en ese período hubo un corte eléctrico o un fallo y el medidor no funcionó. Para solucionar esto, se interpolan las mediciones que faltan. Primero se pasa las fechas a timestamp y luego desde el primer timestamp, se va iterando el dataframe y comprobando si en cada paso de 900 segundos (15 min) existe una medición con ese timestamp y si no existe se inserta un valor NaN, y luego mediante la función de la Pandas interpolate(), se rellenan los campos NaN con la interpolación lineal respecto al valor anterior y al siguiente (Figura 4.2). Este nuevo dataset imputado se guarda en el archivo "THC-id-17.csv".

	datetime	id	value
0	2019-01-01 03:00:00	17.0	0.087
1	2019-01-01 03:15:00	17.0	0.100
2	2019-01-01 03:30:00	17.0	0.084
3	2019-01-01 03:45:00	17.0	0.057
4	2019-01-01 04:00:00	17.0	0.167
...	...	...	...
64451	2020-11-03 22:00:00	17.0	0.031
64452	2020-11-03 22:15:00	17.0	0.055
64453	2020-11-03 22:30:00	17.0	0.019
64454	2020-11-03 22:45:00	17.0	0.021
64455	2020-11-03 23:00:00	17.0	0.052

64456 rows × 3 columns

	datetime	id	value
0	2019-01-01 03:00:00	17.0	0.087
1	2019-01-01 03:15:00	17.0	0.100
2	2019-01-01 03:30:00	17.0	0.084
3	2019-01-01 03:45:00	17.0	0.057
4	2019-01-01 04:00:00	17.0	0.167
...	...	...	...
64588	2020-11-03 22:00:00	17.0	0.031
64589	2020-11-03 22:15:00	17.0	0.055
64590	2020-11-03 22:30:00	17.0	0.019
64591	2020-11-03 22:45:00	17.0	0.021
64592	2020-11-03 23:00:00	17.0	0.052

64593 rows × 3 columns

Figura 4.2. Comparación y visualización del dataset extraído (izquierda) con el dataset imputado (derecha).

En cuanto a las columnas, la columna 'id' no sirve para el modelo con el mismo valor repitiéndose en todas las filas, por lo que se prescinde de ella. La columna 'datetime' tampoco es de utilidad en ese formato, ni tampoco en formato timestamp, ya que corresponden a valores linealmente crecientes y el modelo no podría obtener ninguna pista de la predicción a partir de dicho dato. Sin embargo, puede ser de utilidad si se realiza una serie de transformaciones según los objetivos de predicción en base a ondas longitudinales con el seno y el coseno. El primer análisis sería concretar qué características se pueden sacar de la fecha que estén relacionados con el consumo y comprobar estas suposiciones visualmente utilizando la librería Matplotlib:

- Momento del día: El consumo por la madrugada puede ser muy inferior al consumo del medio día cuando habitualmente es la hora de la comida.

- Día de la semana: Cabe destacar que puede haber una gran diferencia en el consumo entre los días laborables y los fines de semana.
- Día del mes: Habría que analizar si es importante tener en cuenta si la medición se produce a principios de mes o final de mes pero a priori no debe tener mucha importancia.
- Mes del año: Normalmente en los meses de verano e invierno suele haber un consumo más elevado que en primavera y otoño por el uso de aires acondicionados y calefactores por lo que sería de gran ayuda tenerlo en cuenta.

En primer lugar, se muestra en una gráfica la comparación de las mediciones del primer y segundo día del dataset (Figura 4.3). Como se puede comprobar, en las primeras horas del día se cumple que el consumo de electricidad es bajo y suelen ocurrir picos sobre todo en las horas del medio día y la noche, que suelen coincidir con las horas de la comida, por lo que se considera que es importante saberlo.

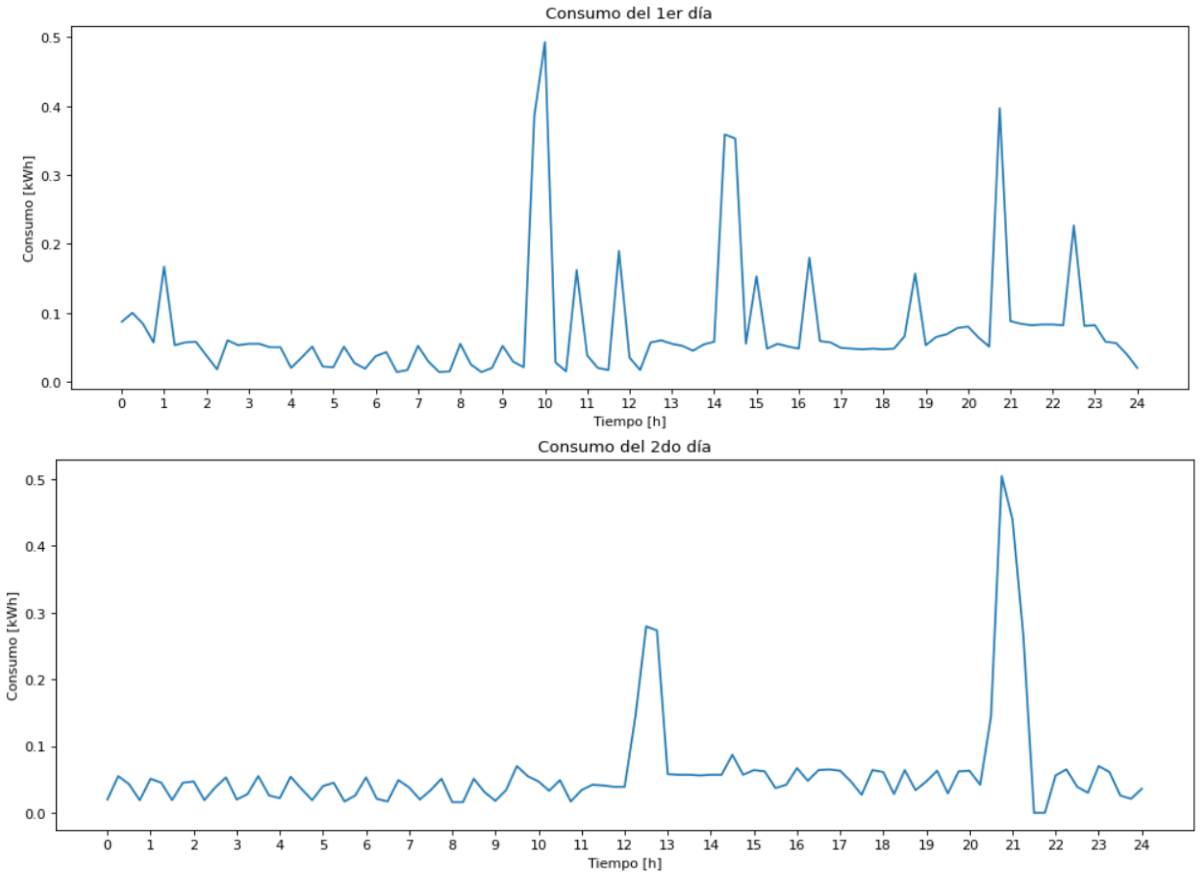


Figura 4.3. Comparativa del primer y segundo día del dataset.

En cuanto al día de la semana, comparando las gráficas de la semana 1 y la semana 2 se cumple que el lunes, martes, viernes y domingo suele haber un consumo más elevado que el resto de días de la semana por lo que será importante saberlo también (Figura 4.4).

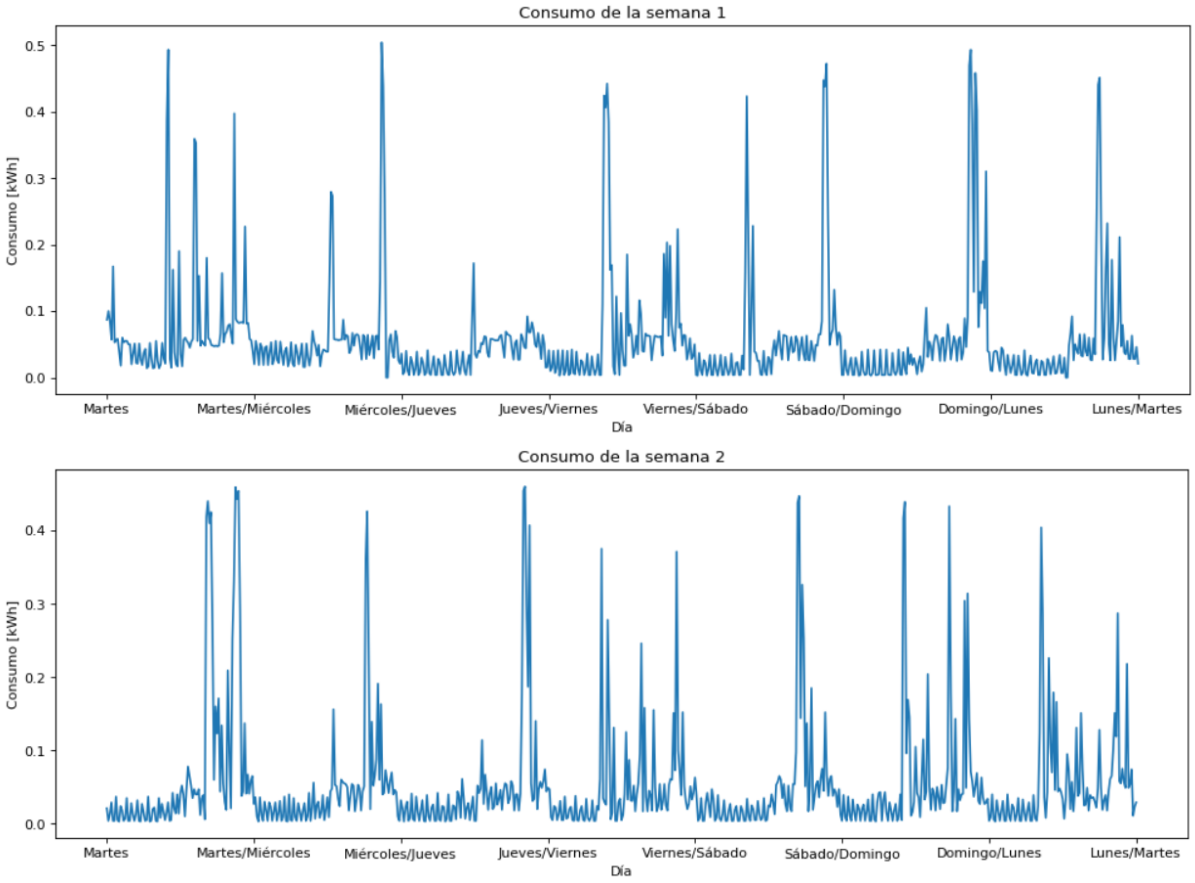


Figura 4.4. Comparativa de la primera y segunda semana del dataset.

En relación al momento del mes, comparando los tres primeros meses se puede deducir que no existe una gran relación entre ellos en cuanto a si se encuentra en el principio del mes o en el final del mes, suele variar según el mes, por lo que no se considera una característica importante para entrenar al modelo y podría provocar confusión (Figura 4.5).

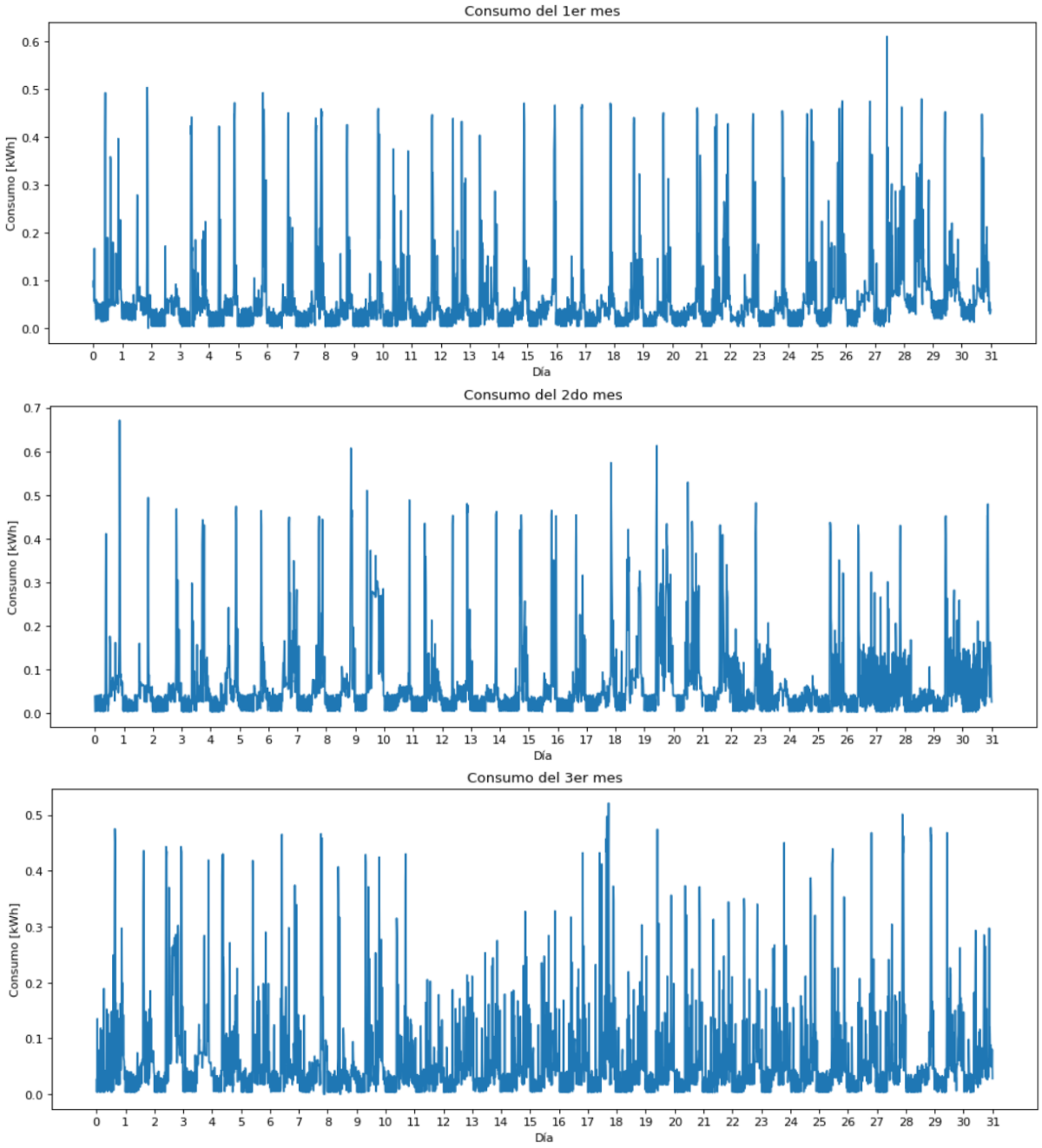


Figura 4.5. Comparativa de los tres primeros meses del dataset

Por último, respecto al mes del año, se cumple la suposición de que en los meses de invierno y verano suele haber un consumo mayor por lo anteriormente comentado, así que también sería una característica importante a tener en cuenta (Figura 4.6).

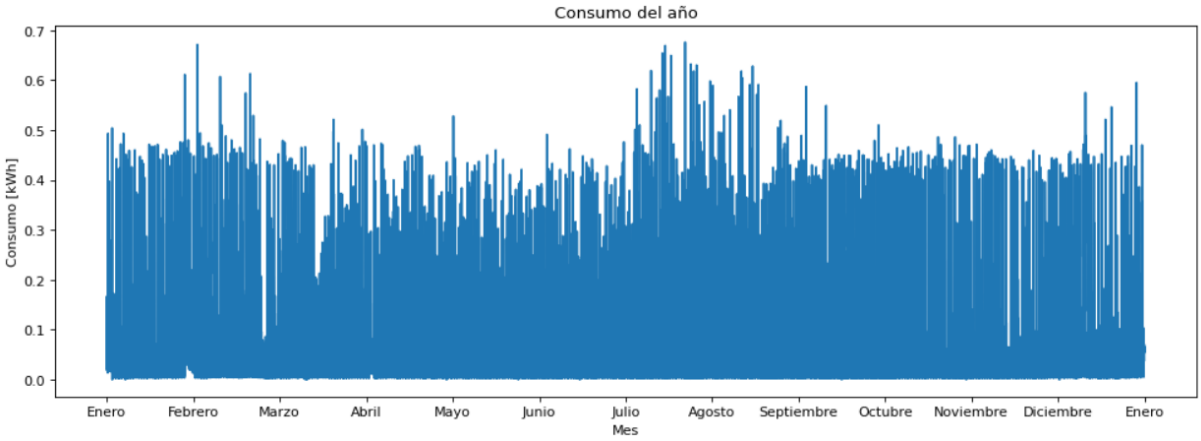


Figura 4.6. Visualización del consumo en el dataset en el primer año.

En definitiva, se usará la hora del día, el día de la semana y el mes del año para la entrada de los datos del modelo. Para obtener estas características, por ejemplo, para la hora del día, se convierte la fecha a timestamp y luego se divide entre los segundos que tiene un día (24*60*60), y después se pasa a radianes multiplicando el valor obtenido por 2 * π y posteriormente calcular su seno y coseno. Lo mismo se hace con la semana y el año, sólo que en este caso se sustituye los segundos del día por los segundos de una semana o los segundos de un año respectivamente. Una vez calculado el seno y el coseno de todos los valores se puede comprobar que se crea una onda longitudinal imprimiendolos en una gráfica, por ejemplo, de los primeros dos días, y como se puede apreciar, el valor del seno y coseno coincide para la misma hora sea del día que sea (Figura 4.7).

Figura 4.7. Representación de la onda longitudinal referente a la hora del día.

Una vez transformado el dataset, el resultado que quedaría sería el de la figura 4.8. Este nuevo conjunto de datos transformado se guarda en un nuevo fichero csv con nombre 'THC-id-17-preprocessed.csv' para usarlo en todos los modelos de entrenamiento.

	day_sin	day_cos	week_sin	week_cos	year_sin	year_cos	value
0	0.707107	0.707107	-0.993712	-0.111964	0.004172	0.999991	0.087
1	0.751840	0.659346	-0.994716	-0.102669	0.004351	0.999991	0.100
2	0.793353	0.608761	-0.995632	-0.093364	0.004530	0.999990	0.084
3	0.831470	0.555570	-0.996461	-0.084051	0.004709	0.999989	0.057
4	0.866025	0.500000	-0.997204	-0.074730	0.004888	0.999988	0.167
...	...	...	...	...	...	...	...
64451	-0.500000	0.866025	-0.826239	0.563320	-0.835100	0.550099	0.031
64452	-0.442289	0.896873	-0.820936	0.571021	-0.835001	0.550248	0.055
64453	-0.382683	0.923880	-0.815561	0.578671	-0.834902	0.550398	0.019
64454	-0.321439	0.946930	-0.810115	0.586271	-0.834804	0.550548	0.021
64455	-0.258819	0.965926	-0.804598	0.593820	-0.834705	0.550697	0.052

64593 rows × 7 columns

Figura 4.8. Resultado de la transformación del dataframe.

4.3. Desarrollo de modelos de machine learning para la predicción de valores

En primer lugar, al comenzar el desarrollo de cualquier modelo, lo primero que se hará será importar las librerías necesarias y extraer el dataset de los csv disponibles que se han generado en pasos anteriores en un objeto DataFrame de Pandas.

Los modelos seguirán un aprendizaje supervisado, que consiste en pasarle al modelo una tupla de valores de entrada y el resultado esperado. Sin embargo, hay que especificar de que forma van a salir y entrar los datos del modelo. Existen varias opciones:

- **Método sólo fecha:** En esta opción, la entrada del dato será de una serie de valores que corresponden a los senos y cosenos relacionados con la fecha calculados en el paso anterior y la salida será de un único valor de consumo que se produciría en esa fecha en concreto. Si se requiere predecir por ejemplo, la hora siguiente, se requeriría pasar desde la última medición todos los valores de la fecha correspondientes a cada paso de 15 min hasta llegar a la hora.
- **Método de la ventana de un sólo paso:** De esta forma el modelo recibe una entrada de X filas con el valor en kWh y la fecha transformada de la medición y devuelve el siguiente valor que correspondería a esa entrada. Otra forma sería hacerlo sólo con

los valores sin fecha, es decir, sólo con X valores y el modelo intenta predecir el siguiente valor. La forma de predecir sería de forma recursiva usando las mismas predicciones que se van generando en cada paso, más adelante se explica más claro.

- **Método de la ventana de varios pasos:** Esta forma es una forma de predicción simplificada de la anterior a la hora de predecir, ya que la entrada sería de X filas y la salida en vez de ser de un sólo valor, podría ser de varios valores. La forma de predecir para la siguiente hora sería simplemente coger los últimos X valores y que devuelva los 4 siguientes valores correspondientes a la siguiente hora.

El modelo, además de los datos de entrada y objetivo, requiere una serie de parámetros para el entrenamiento. A nivel general, se va a utilizar un número máximo de 200 épocas, que es el número de veces que el modelo aprende de los datos de entrada proporcionados, un batch size de 128, que refiere al tamaño del lote de entrada del modelo, para acelerar el entrenamiento, y verbose a 1 para que muestre información relativa al error del modelo en cada época. Asimismo, utiliza unos datos de validación con la misma forma de los de entrenamiento con el objetivo de comprobar en cada época que cantidad de error tiene el modelo. Por otra parte, para evitar el overfitting, se establece el callback EarlyStopping monitorizando el 'val_loss' orientado a buscar el mínimo valor de pérdida de la validación del modelo, y con una paciencia de 100 épocas para los modelos de la hora y el día, y de 50 para los modelos con conjuntos de datos muy grandes como la semana y el mes, que una vez pasadas, es cuando empieza a buscar el mínimo y cuando el error de validación tiende a crecer se finaliza el entrenamiento aunque no haya llegado al número máximo de épocas establecido. Esta paciencia es un poco arriesgada, porque con se ha observado que con un número mayor de épocas el modelo se adapta mejor a los picos, pero también aumenta el overfitting.

Al compilar el modelo, se utiliza como medida de la pérdida o también llamado 'loss', el error cuadrático medio (MSE), como optimizador se utiliza Adam y como métricas el error cuadrático medio (MSE), el error medio absoluto (MAE) y el porcentaje del error medio absoluto simétrico (SMAPE).

Para todos los modelos se va a realizar una repartición similar del dataset excepto para los modelos del mes con el método de ventana de varios pasos, 70% para el entrenamiento, que son los que utilizará el modelo para entrenar cada época, 15% para validación, que comprueba como de bueno es el modelo en cada época y 15% para test, que son los que se utilizan para comprobar el error cuando el modelo ha terminado su etapa de entrenamiento. En los modelos del mes con el método de ventana de varios pasos la repartición sera 60% para entrenamiento, 20% para test y 20% para validación, ya que al tener que construir una ventana más grande, con la repartición anterior no habrían datos suficientes para construirla con la repartición anterior en los datos de test y de validación.

Excepto para el método de sólo fecha, en todos los métodos se escala la entrada del modelo usando la transformación de MaxAbsScaler de Scikit-Learn. Existen otras alternativas para escalar los datos, como por ejemplo MinMaxScaler, pero el problema de este es que en nuestros datos existen picos de consumo y MinMaxScaler tiende a reducir la dispersión de los datos, es decir, estandarizarlos, y elimina estos picos de consumo que son importantes conocer. Sin embargo, MaxAbsScaler es indicado para este tipo de series temporales ya que se basa en el máximo absoluto de todos los datos y por lo tanto, no elimina estas características del consumo. En la figura 4.9, se puede observar que la mayoría de los datos se centran entre el 0 y 0.2, y existen muchos datos dispersos que son importantes conocer para que el modelo aprenda de ellos.

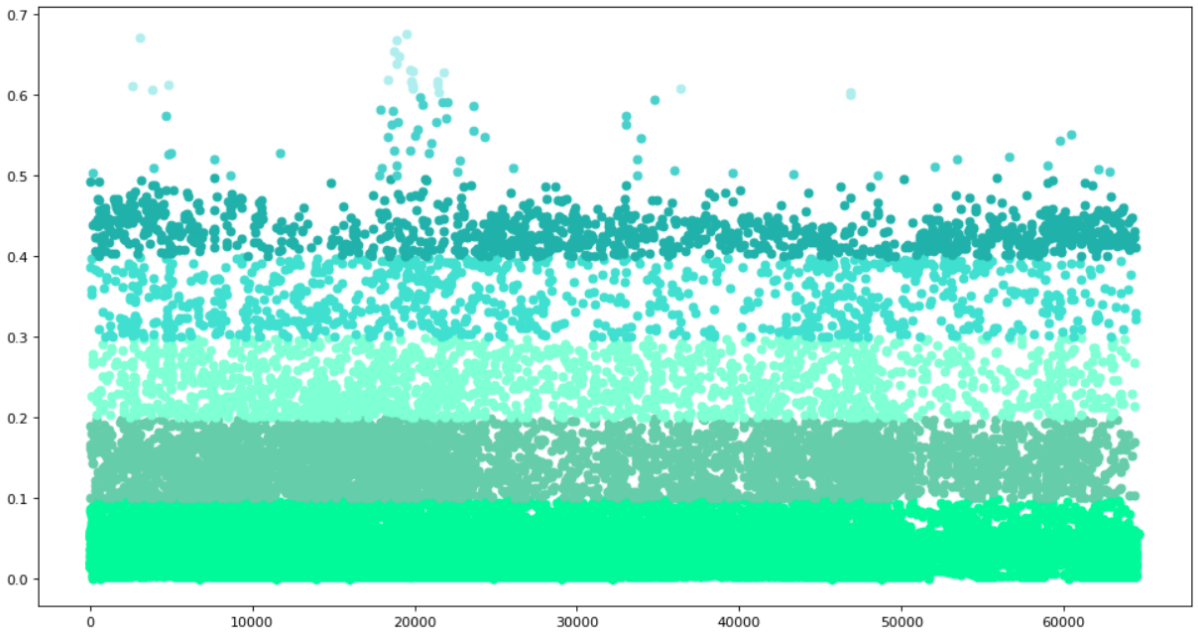


Figura 4.9. Representación de la dispersión de los datos.

4.3.1. Modelos utilizados

En este apartado se van a describir los modelos que se han usado para el entrenamiento. Estos modelos tendrán una serie de variables en común entre sí, que son *n_features* que corresponde al número de características de entrada de los datos, por ejemplo, si se utiliza sólo los campos correspondientes a la fecha como datos de entrada, *n_features* será igual a 6 (day_sin, day_cos, week_sin, week_cos, year_sin, year_cos). Otro de los parámetros es *n_predicciones*, que representa el número de predicciones que se quiere obtener a partir de la entrada, por ejemplo, en el método de la ventana de un paso, se quiere predecir el siguiente valor, por lo que *n_predicciones* será 1, sin embargo, para el método de varios pasos, por ejemplo para predecir la hora, se requiere que *n_predicciones* sea igual a 4 (número de mediciones en una hora). El último y por ello no menos importante es *n_entrada*, que especifica cuantos arrays de entrada con *n_features* características se requieren para predecir *n_predicciones.*

Otro de los aspectos comunes en los modelos suele ser el número de neuronas. Este número no viene dado por ningún cálculo ni método que pueda deducir cuál es el mejor, sino que se va probando hasta obtener un número óptimo. Lo mismo pasa con el número de capas. Normalmente, cuando se necesita reducir el bias, suele aumentarse el número de neuronas o de capas, pero esto puede provocar un riesgo de overfitting, por lo que lo mejor es ir probando hasta encontrar la forma adecuada.

En las posteriores explicaciones, cuando aparezca *lenX* hará referencia a la longitud total de los datos de entrada al modelo.

4.3.1.1. Modelo denso

El modelo denso básico sólo va a ser utilizado en el método de entrada de sólo fecha ya que la entrada por defecto es de 2 dimensiones y en posteriores métodos los datos tendrán 3 dimensiones y no se adapta muy bien en la entrada devolviendo un error. Se compone de varias capas densas apiladas y con la función de activación 'relu'. La entrada de los datos será de la forma (lenX, n_features) y la salida de la forma (lenX, n_predicciones).

```python
def build_model0():
    model = tf.keras.Sequential([
        tf.keras.layers.Input(shape=n_features, name='input'),
        tf.keras.layers.Dense(256, activation='relu'),
        tf.keras.layers.Dense(256, activation='relu'),
        tf.keras.layers.Dense(128, activation='relu'),
        tf.keras.layers.Dense(64, activation='relu'),
        tf.keras.layers.Dense(n_predicciones)
    ])

    optimizer = tf.keras.optimizers.Adam()

    model.compile(loss='mse', optimizer=optimizer, metrics=['mae', 'mse', smape])

    return model
```

Figura 4.10. Modelo denso.

4.3.1.2. Modelo LSTM de una capa

El modelo LSTM de una capa, como su nombre dice, está formado por una capa LSTM con 128 neuronas. El número de neuronas se ha establecido a 128 porque aunque aumente un poco más el overfitting, se comprueba que con un mayor número de neuronas el modelo se adapta mejor a los picos del consumo, aunque a partir de 128 tampoco hay mucha mejora, por lo que se fija en ese número para no aumentar demasiado el overfitting y el tiempo de entrenamiento. En la capa LSTM se utiliza la función de activación de la tangente para poder usar la GPU y que el modelo entrene más rápido. La entrada de los datos será de la forma (lenX, n_entrada, n_features) y la salida de la forma (lenX, n_predicciones).

```python
def build_model1():
    model = tf.keras.Sequential([
        tf.keras.layers.Input(shape=(n_entrada, n_features)),
        tf.keras.layers.LSTM(128, activation='tanh'),
        tf.keras.layers.Dense(n_predicciones)
    ])

    optimizer = tf.keras.optimizers.Adam()

    model.compile(loss='mse', optimizer=optimizer, metrics=['mae', 'mse', smape])

    return model
```

Figura 4.11. Modelo LSTM de una capa.

4.3.1.3. Modelo LSTM de dos capas

El modelo LSTM de dos capas, es igual que el anterior, incluso en la entrada y en la salida, sólo que esta vez son dos capas LSTM apiladas. Sin embargo, es importante que en las capas anteriores a la última capa LSTM tengan el parámetro return_sequences a True, para que la entrada a la siguiente capa LSTM siga siendo de 3 dimensiones.

```
def build_model2():
    model = tf.keras.Sequential([
        tf.keras.layers.Input(shape=(n_entrada, n_features)),
        tf.keras.layers.LSTM(128, activation='tanh', return_sequences = True),
        tf.keras.layers.LSTM(128, activation='tanh'),
        tf.keras.layers.Dense(n_predicciones)
    ])

    optimizer = tf.keras.optimizers.Adam()

    model.compile(loss='mse', optimizer=optimizer, metrics=['mae', 'mse', smape])

    return model
```

Figura 4.12. Modelo LSTM de dos capas.

4.3.1.4. Modelo LSTM de tres capas

El modelo LSTM de tres capas es igual al de dos capas sólo que ahora son tres capas. Cabe destacar que el objetivo de estos dos últimos modelos es comprobar cuál es el comportamiento de estos cuando se les aumenta el número de capas, para probar cuál es el número óptimo como se ha mencionado antes.

```
def build_model3():
    model = tf.keras.Sequential([
        tf.keras.layers.Input(shape=(n_entrada, n_features)),
        tf.keras.layers.LSTM(128, activation='tanh', return_sequences=True),
        tf.keras.layers.LSTM(128, activation='tanh', return_sequences=True),
        tf.keras.layers.LSTM(64, activation='tanh'),
        tf.keras.layers.Dense(n_predicciones)
    ])

    optimizer = tf.keras.optimizers.Adam()

    model.compile(loss='mse', optimizer=optimizer, metrics=['mae', 'mse', smape])

    return model
```

Figura 4.13. Modelo LSTM de tres capas.

4.3.1.5. Modelo LSTM Bidireccional

El modelo LSTM Bidireccional pretende que la capa LSTM aprenda la serie tanto hacia delante como hacia atrás, por lo que es otra alternativa que se puede probar a ver qué tal funciona.

```
def build_model4():
    model = tf.keras.Sequential([
        tf.keras.layers.Input(shape=(n_entrada, n_features)),
        tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(128, activation='tanh')),
        tf.keras.layers.Dense(n_predicciones)
    ])

    optimizer = tf.keras.optimizers.Adam()

    model.compile(loss='mse', optimizer=optimizer, metrics=['mae', 'mse', smape])

    return model
```

Figura 4.14. Modelo LSTM Bidireccional.

4.3.1.6. Modelo Convolucional 1D

El modelo Convolucional 1D, desde sus inicios, fue preparado para la predicción de imágenes pero posteriormente, se comprobó que también era bueno para predecir patrones de números como las series temporales, por lo que no es una opción a descartar en el entrenamiento. La capa convolucional de 1D especifica el filtro de 64, que igual que las neuronas, se obtiene a modo de prueba y el kernel_size que es algo similar a un batch size a 1. Luego se aplica una capa MaxPooling1D para reducir el resultado de la capa anterior con un pool_size de 2, que representa el tamaño de agrupación de la ventana. Por último, se aplana el resultado con la capa Flatten y se devuelve las predicciones. La entrada de este modelo es diferente a las anteriores, ya que, necesita un array de 4 dimensiones de entrada y devuelve un array de 3 dimensiones de salida. Para ello, se configurará la entrada del modo (lenX, 1, n_entrada, n_features), donde el 1 representa en cuantas partes se quiere dividir los datos, pero en este caso no se quiere dividir, y la salida del modo (lenX, 1, n_predicciones).

```
def build_model5():
    model = tf.keras.Sequential([
        tf.keras.layers.TimeDistributed(tf.keras.layers.Conv1D(filters=64, kernel_size=1, activation='relu'),
                                        input_shape=(None, n_entrada, n_features)),
        tf.keras.layers.TimeDistributed(tf.keras.layers.MaxPooling1D(pool_size=2)),
        tf.keras.layers.TimeDistributed(tf.keras.layers.Flatten()),
        tf.keras.layers.Dense(n_predicciones)
    ])

    optimizer = tf.keras.optimizers.Adam()

    model.compile(loss='mse', optimizer=optimizer, metrics=['mae', 'mse', smape])

    return model
```

Figura 4.15. Modelo Convolucional 1D.

4.3.1.7. Modelo Convolucional 1D con LSTM

El modelo Convolucional 1D con LSTM es igual que el anterior sólo que se añade una capa LSTM con return_sequences a True, ya que la salida del modelo convolucional debería ser de 3 dimensiones. En esta ocasión, se intenta mezclar las dos capas principales que se ha visto para ver cómo funcionan juntas.

```
def build_model6():
    model = tf.keras.Sequential([
        tf.keras.layers.Input(shape=(None, n_entrada, n_features)),
        tf.keras.layers.TimeDistributed(tf.keras.layers.Conv1D(filters=64, kernel_size=1, activation='relu')),
        tf.keras.layers.TimeDistributed(tf.keras.layers.MaxPooling1D(pool_size=2)),
        tf.keras.layers.TimeDistributed(tf.keras.layers.Flatten()),
        tf.keras.layers.LSTM(128, activation='tanh', return_sequences=True),
        tf.keras.layers.Dense(n_predicciones)
    ])

    optimizer = tf.keras.optimizers.Adam()

    model.compile(loss='mse', optimizer=optimizer, metrics=['mae', 'mse', smape])

    return model
```

Figura 4.16. Modelo Convolucional 1D con LSTM.

4.3.1.8. Modelo ConvLSTM1D

El modelo ConvLSTM1D es otra variante de capa que mezcla la capa convolucional y la capa LSTM en una capa ConvLSTM, y tiene una construcción similar a la de la convolucional. Igualmente, la entrada debe ser de 4 dimensiones como se ha visto antes para el modelo convolucional, pero la salida será de 2 dimensiones de la forma (lenX, n_predicciones).

```
def build_model7():
    model = tf.keras.Sequential([
        tf.keras.layers.Input(shape=(None, n_entrada, n_features)),
        tf.keras.layers.ConvLSTM1D(filters=64, kernel_size=1, activation='tanh'),
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(n_predicciones),
    ])

    optimizer = tf.keras.optimizers.Adam()

    model.compile(loss='mse', optimizer=optimizer, metrics=['mae', 'mse', smape])

    return model
```

Figura 4.17. Modelo ConvLSTM1D.

4.3.1.9. Modelo ConvLSTM2D

El modelo ConvLSTM2D es igual que el anterior sólo que en este caso la entrada será de una dimensión más, es decir, de 5 dimensiones. Para ello, se transforman los datos de entrada a la forma (lenX, 1, 1, n_entrada, n_features), donde los 1 representan también la división de los datos. En este caso, al ser de una convolucional de 2 dimensiones, el kernel size tiene que definirse mediante una tupla. La salida igualmente será de 2 dimensiones, igual que el anterior.

```
def build_model8():
    model = tf.keras.Sequential([
        tf.keras.layers.Input(shape=(1, 1, n_entrada, n_features)),
        tf.keras.layers.ConvLSTM2D(filters=64, kernel_size=(1,1), activation='relu'),
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(n_predicciones),
    ])

    optimizer = tf.keras.optimizers.Adam()

    model.compile(loss='mse', optimizer=optimizer, metrics=['mae', 'mse', smape])

    return model
```

Figura 4.18. Modelo ConvLSTM2D.

4.3.1.10. Modelo LSTM Encoder Decoder

El modelo LSTM Encoder Decoder es un modelo especializado para la predicción de series temporales de varios pasos. El objetivo es codificar la entrada en la primera capa LSTM y luego pasarla por la capa RepeatVector con n_predicciones pasos para que interprete la

salida de la primera capa como un vector, una vez por cada paso, y luego se decodifique por otra capa LSTM posterior. Luego, se pueden añadir más capas como se ve en la Figura 4.19, con otra capa densa con 100 neuronas y una capa de salida, en la cuál esta vez el valor de salida se especifica a 1. La entrada a este modelo es de 3 dimensiones, como en los primeros LSTM que se han explicado, pero la salida es un array de la forma (lenX, n_predicciones, 1).

```
def build_model9():
    model = tf.keras.Sequential([
        tf.keras.layers.Input(shape=(n_entrada, n_features)),
        tf.keras.layers.LSTM(128, activation='tanh'),
        tf.keras.layers.RepeatVector(n_predicciones),
        tf.keras.layers.LSTM(128, activation='tanh', return_sequences=True),
        tf.keras.layers.TimeDistributed(tf.keras.layers.Dense(100)),
        tf.keras.layers.TimeDistributed(tf.keras.layers.Dense(1))
    ])

    optimizer = tf.keras.optimizers.Adam()

    model.compile(loss='mse', optimizer=optimizer, metrics=['mae', 'mse', smape])

    return model
```

Figura 4.19. Modelo LSTM Encoder Decoder.

4.3.1.11. GRU una capa

El modelo GRU una capa es otra variante de las capas recurrentes existentes en Keras. Esta capa está especializada en series temporales largas, pero tiene un defecto que LSTM corrige que es en la memoria a corto plazo. Sin embargo, no pasará desapercibido en este entrenamiento para ver su funcionamiento. La entrada y salida es igual que en LSTM. En esta primera prueba se probará con una sola capa con 128 neuronas.

```
def build_model10():
    model = tf.keras.Sequential([
        tf.keras.layers.Input(shape=(n_entrada, n_features)),
        tf.keras.layers.GRU(128, activation='tanh'),
        tf.keras.layers.Dense(n_predicciones)
    ])

    optimizer = tf.keras.optimizers.Adam()

    model.compile(loss='mse', optimizer=optimizer, metrics=['mae', 'mse', smape])

    return model
```

Figura 4.20. Modelo GRU una capa.

4.3.1.12. GRU dos capas

El modelo GRU dos capas es igual que LSTM de dos capas pero con la capa GRU en vez de LSTM.

```python
def build_model11():
    model = tf.keras.Sequential([
        tf.keras.layers.Input(shape=(n_entrada, n_features)),
        tf.keras.layers.GRU(128, activation='tanh', return_sequences=True),
        tf.keras.layers.GRU(128, activation='tanh'),
        tf.keras.layers.Dense(n_predicciones)
    ])

    optimizer = tf.keras.optimizers.Adam()

    model.compile(loss='mse', optimizer=optimizer, metrics=['mae', 'mse', smape])

    return model
```

Figura 4.21. Modelo GRU dos capas.

4.3.1.13. LSTM unrestricted de una capa

El modelo LSTM unrestricted de una capa es otra variante más de LSTM sólo que esta vez, aplicando el return_sequences a True en la última capa LSTM lo que provoca es que esta no esté restringida y muestre salidas ocultas que en el otro caso no ocurre. Después de esta capa se aplica una capa Flatten para aplanar el resultado para devolverla en un formato de 2 dimensiones. La entrada y la salida es igual que para LSTM normal.

```python
def build_model12():
    model = tf.keras.Sequential([
        tf.keras.layers.Input(shape=(n_entrada, n_features)),
        tf.keras.layers.LSTM(128, activation='tanh', return_sequences=True),
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(n_predicciones)
    ])

    optimizer = tf.keras.optimizers.Adam()

    model.compile(loss='mse', optimizer=optimizer, metrics=['mae', 'mse', smape])

    return model
```

Figura 4.22. Modelo LSTM unrestricted de una capa.

4.3.1.14. LSTM unrestricted de dos capas

El modelo LSTM unrestricted de dos capas es igual que la anterior, sólo que con dos capas.

```python
def build_model13():
    model = tf.keras.Sequential([
        tf.keras.layers.Input(shape=(n_entrada, n_features)),
        tf.keras.layers.LSTM(128, activation='tanh', return_sequences=True),
        tf.keras.layers.LSTM(128, activation='tanh', return_sequences=True),
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(n_predicciones)
    ])

    optimizer = tf.keras.optimizers.Adam()

    model.compile(loss='mse', optimizer=optimizer, metrics=['mae', 'mse', smape])

    return model
```

Figura 4.23. Modelo LSTM unrestricted de una capa.

4.3.2. Método de entrenamiento de sólo fecha

Al utilizar el método de entrada de sólo fecha, para preparar los datos de entrada del modelo se separa la columna objetivo de las demás características del dataset, lo que resulta en un array de entrada de varias columnas referentes a los senos y cosenos calculados de la fecha y un array de resultado objetivo con los valores a predecir.

Una vez separadas las características de los objetivos, se procede a dividir el dataset para obtener los datos de entrenamiento, validación y test. Normalmente, para un mejor comportamiento del modelo, se suele escalar la entrada mediante algún método de estandarización o normalización, pero en este caso no se va a realizar ningún escalado ya que los datos de entradas son senos y cosenos con valores entre el rango -1 y 1, por lo que a priori no haría falta ya que al aplicar la transformación de la fecha anterior se estarían escalando.

Uno de los principales inconvenientes de este método es que al recibir sólo la entrada de la fecha, el modelo no tendría más información sobre los valores anteriores y por ejemplo, el 1 de enero de 2017 tendría en la práctica la misma entrada que el 1 de enero de 2019, por lo que el modelo podría recordar el valor que ha usado para el entrenamiento y producirse overfitting o provocar que el modelo en vez de aprender simplemente recuerde y use los valores que se le han introducido anteriormente.

Una vez que los datos están preparados, se comienza con el entrenamiento de modelos. Hay tener en cuenta que el objetivo es que los modelos reciban 6 características (seno y coseno del día, seno y coseno de la semana, seno y coseno del año) y devuelvan una predicción de un sólo valor en kWh del consumo.

Para este método no se ha hecho una prueba muy grande de modelos, en este caso sólo 4, ya que como se podrá observar en los resultados, no es un método muy prometedor al ver que todos los modelos suelen generar las mismas predicciones, ya que cae en el error de usar los valores anteriores con características similares de la fecha (Figura 4.24, Figura 4.25).

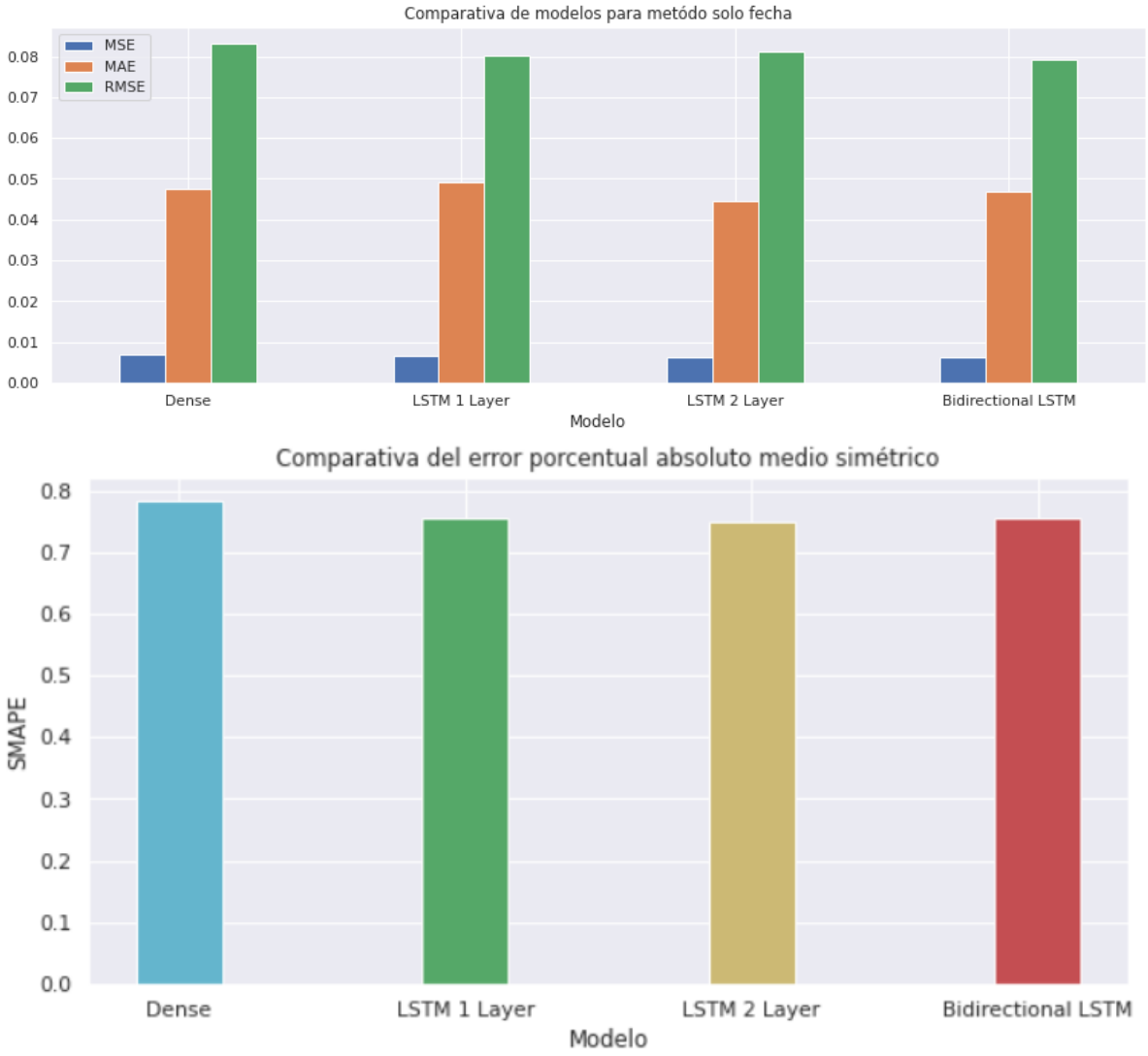


Figura 4.24. Resultados del entrenamiento del método sólo fecha.

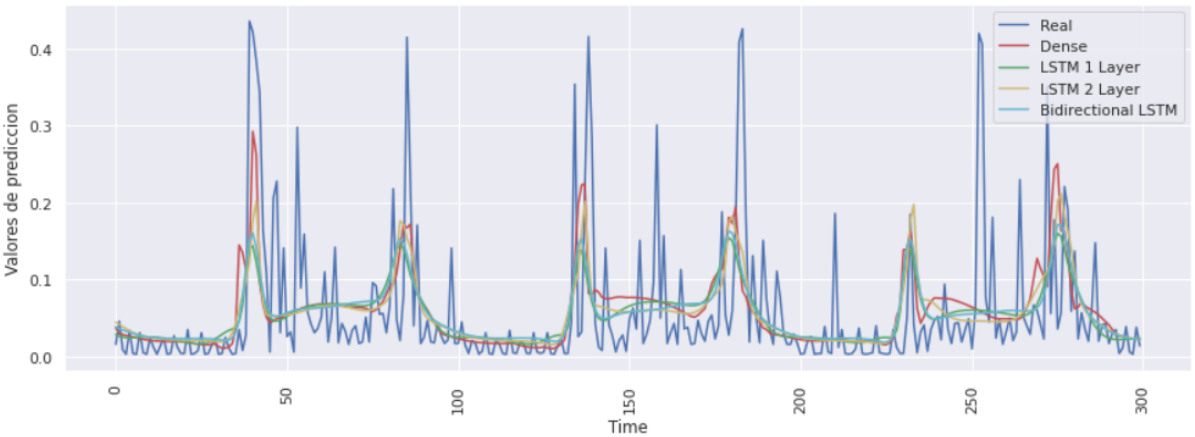


Figura 4.25. Gráfica de comparación de predicciones vs valores reales.

4.3.3. Método de la ventana de un sólo paso

El método de la ventana es el más utilizado para la predicción de series temporales. El objetivo es que a partir de una serie de valores anteriores, se haga la predicción del siguiente. Existen dos tipos de ventanas, de un paso, que es la que se explicará en este apartado, y el de varios pasos, que corresponde al siguiente apartado. Para el método de la ventana de un sólo paso, la entrada al modelo sería algo como lo siguiente, donde Xi representa cada valor del array de características que se utiliza para entrenar al modelo e Y1 el valor de salida.

Entrada: X1, X2, X3, X4, X5, X6 Salida: Y1

Sin embargo, como su propio nombre dice, este método es de un sólo paso, es decir, que sólo predice un valor, que representaría los siguientes 15 minutos, y el objetivo de este trabajo es predecir varios, como por ejemplo, los 4 siguientes para la hora. Para ello, si se quiere usar este método se podría plantear algo como lo siguiente:

Entrada: X1, X2, X3, X4, X5, X6 Salida: Y1

Entrada: X2, X3, X4, X5, X6, Y1 Salida: Y2

Entrada: X3, X4, X5, X6, Y1, Y2 Salida: Y3

Entrada: X4, X5, X6, Y1, Y2, Y3 Salida: Y4

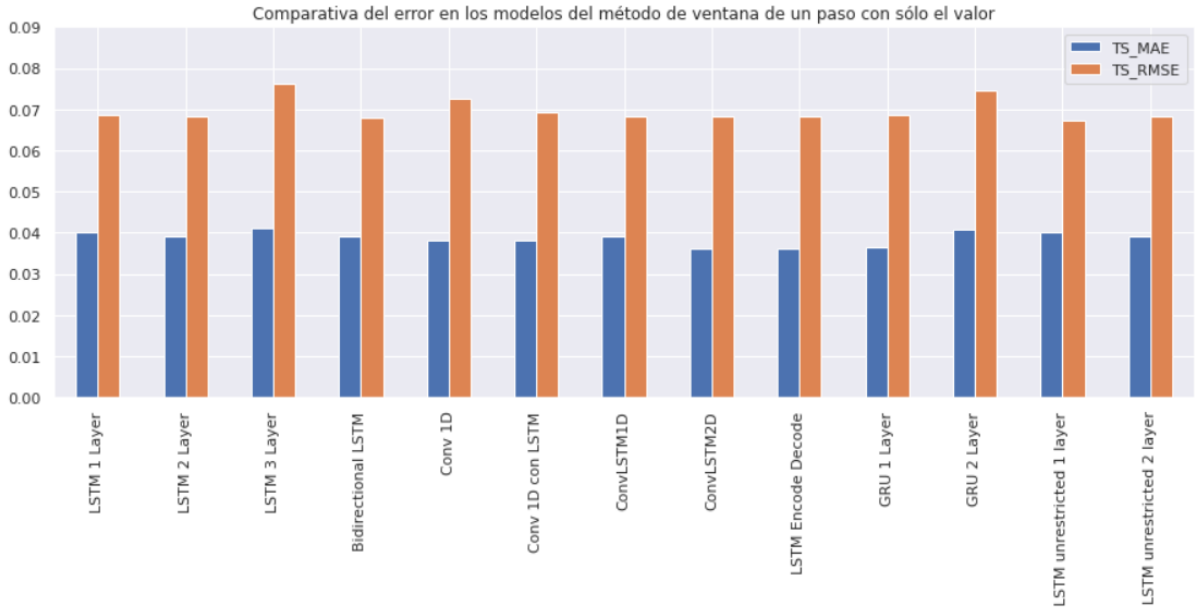
Esta sería la forma de predecir para este tipo, utilizando las salidas de las predicciones anteriores. Por ejemplo, si se requiere predecir la siguiente hora, se predicen primero los primeros 15 minutos, y estos se utilizan para predecir los siguientes 15 minutos del anterior calculado, así sucesivamente hasta llegar al objetivo. Pero este método en este caso tiene una serie de riesgos y problemas que finalmente lleven a que se descarte. El primer problema es que habría que analizar cómo de bueno es predecir a partir de predicciones, ya que, si las predicciones son malas, utilizarlas conllevaría un grave inconveniente. Otro problema sería el coste computacional que requiere esta operación, dado que sería necesario realizar una petición POST desde el cliente por cada valor a predecir. Por ejemplo, para predecir los siguientes 30 días, que es el objetivo máximo de predicciones, serían necesarias 2880 peticiones, por lo que se requeriría de un gran coste computacional tanto de tiempo como de recursos.

Por consiguiente, para analizar el primer problema, se comprueba como de buenas son las predicciones para este tipo de entrada. Esto se puede hacer de dos formas, la primera es que la característica de entrada corresponda sólo a los valores de las mediciones, es decir, usando sólo los anteriores valores de la columna 'value'. La otra forma es usando las columnas que corresponden a la fecha además del valor de las mediciones, básicamente, todas las columnas del dataset preprocesado. Los resultados del entrenamiento son los siguientes (Tabla 4.1, Figura 4.26):

Tabla de comparación de todos los modelos - Método de ventana sólo valor 6 - 1

	TR_MAE	TR_MSE	TR_SMAPE	TR_RMSE	V_MAE	V_MSE	V_SMAPE	V_RMSE	TS_MAE	TS_MSE	TS_SMAPE	TS_RMSE
LSTM 1 Layer	0.038253	0.004301	0.713337	0.065579	0.033723	0.003896	0.675466	0.062417	0.040178	0.004689	0.721866	0.068477
LSTM 2 Layer	0.036406	0.004142	0.697554	0.064361	0.033420	0.003933	0.681679	0.062711	0.039030	0.004666	0.717159	0.068311
LSTM 3 Layer	0.028682	0.002448	0.646914	0.049473	0.036258	0.004914	0.701833	0.070100	0.041217	0.005798	0.720810	0.076148
Bidirectional LSTM	0.037412	0.004355	0.705066	0.065991	0.033448	0.003934	0.679252	0.062719	0.039086	0.004613	0.715317	0.067922
Conv 1D	0.038924	0.005217	0.716826	0.072226	0.033091	0.004905	0.641433	0.070034	0.038033	0.005286	0.678598	0.072707
Conv 1D con LSTM	0.038995	0.004777	0.709953	0.069117	0.032254	0.004234	0.635076	0.065070	0.038046	0.004801	0.678709	0.069290
ConvLSTM1D	0.038575	0.004589	0.720980	0.067745	0.033928	0.004039	0.682556	0.063550	0.039138	0.004651	0.711770	0.068201
ConvLSTM2D	0.036930	0.004601	0.698821	0.067833	0.030734	0.004083	0.619553	0.063901	0.036049	0.004678	0.653086	0.068395
LSTM Encode Decode	0.034989	0.004150	0.654656	0.064422	0.029663	0.003888	0.589979	0.062350	0.035990	0.004665	0.636794	0.068298
GRU 1 Layer	0.035695	0.004253	0.673588	0.065216	0.030546	0.003978	0.611468	0.063071	0.036477	0.004715	0.649853	0.068666
GRU 2 Layer	0.031368	0.003173	0.655625	0.056331	0.034676	0.004658	0.688439	0.068249	0.040821	0.005582	0.726117	0.074715
LSTM unrestricted 1 layer	0.039104	0.004402	0.729111	0.066346	0.034155	0.003800	0.696240	0.061648	0.040240	0.004536	0.735788	0.067347
LSTM unrestricted 2 layer	0.036795	0.004210	0.702741	0.064883	0.033447	0.003964	0.678095	0.062957	0.039063	0.004650	0.713503	0.068191

Tabla 4.1. Tabla de resultados del entrenamiento de los modelos para método de ventana de un sólo paso usando sólo el valor.



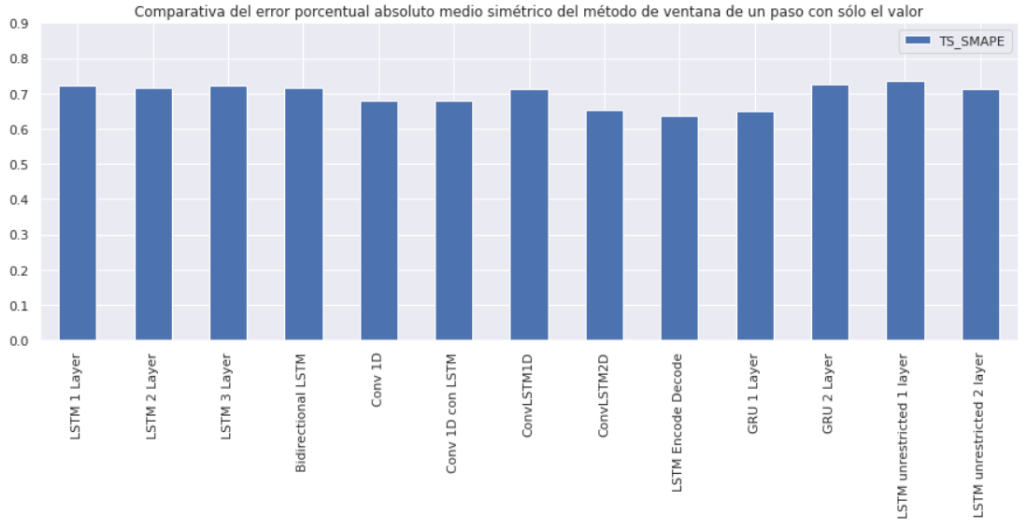


Figura 4.26. Gráficas comparativas de los resultados del entrenamiento para método de ventana de un sólo paso usando sólo el valor.

Para analizar los resultados del entrenamiento, primero hay que comprender qué se consideraría un valor alto o bajo de error. Por ejemplo, si se obtiene un RMSE de 0.07, para interpretarlo, lo primero que hay que saber es qué características tienen los valores que se están entrenando. Como se ha visto antes, el valor mínimo es 0 y el valor máximo era 0.676 (Figura 4.1). Sin embargo, no hay que fijarse sólo en eso, ya que, como se puede observar, la media es 0.056, y el percentil 75 es 0.054, por lo que significa que la mayoría de datos está muy por debajo del valor máximo. Además, con el valor 0.083 de la desviación estándar, que mide la dispersión de los datos respecto la media, se puede concluir que la mayoría de los datos están entre 0 y 0.139 aproximadamente, por lo que tener una media de error de 0.07, que representa aproximadamente la mitad del valor máximo de la mayoría, significa existe un error alto. Entonces, viendo que existe un error alto, no sería buena idea predecir a partir de predicciones ya que un error alto en una de las predicciones puede influir mucho en las siguientes.

4.3.4. Método de la ventana de varios pasos

El método de la ventana de varios pasos es similar al anterior sólo que en vez de devolver un sólo paso devuelve varios, por lo que tiene un comportamiento más eficiente ya que no hay que iterar sobre las predicciones para sacar los siguientes valores. Cabe decir que el error obtenido puede ser mayor que en el anterior método, pero se debe a que es más difícil predecir, por ejemplo, 4 valores que 1, pero para que la comparación fuera justa se

debería comparar con el error de todas las predicciones obtenidas de manera recursiva. La estructura de los datos de entrada y salida para predecir una mismamente una hora sería la siguiente, donde X representa la entrada e Y la salida:

X: X1, X2, X3, X4, X5, X6, X7, X8, X9, X10, X11, X12 Y: Y1, Y2, Y3, Y4

Como se puede observar, en este método hay que incrementar el número de valores de entrada ya que el número de predicciones es mayor. En este trabajo se seguirá un patrón en el que la entrada será 3 veces el tamaño del número de predicciones a obtener. Por otra parte, para este método, se comprobarán dos formas de entrada de los datos en cuanto a las características, la primera pasando un array compuesto sólo por los valores de las predicciones anteriores y la segunda con un array formado por los campos relacionados con la fecha además de los valores, para comprobar de que forma se ajustan mejor los datos. Una vez que se separan los datos en la forma de entrada y salida especificada se procede con el entrenamiento.

4.3.4.1. Análisis de resultados para la hora

Como las mediciones tienen una periodicidad de 15 minutos, en total en una hora se tomarán 4 mediciones, y como se ha comentado antes la entrada deberá ser 3 veces más grande por lo tanto será de 12 valores. En este caso, para el entrenamiento se va a utilizar un batch size de 128, ya que los datos tampoco son tan grandes.

Los siguientes resultados del entrenamiento corresponden a la entrada de sólo valores (Tabla 4.2, Figura 4.27):

Tabla de comparación de todos los modelos - Método de ventana sólo valor 12 - 4

	TR_MAE	TR_MSE	TR_SMAPE	TR_RMSE	V_MAE	V_MSE	V_SMAPE	V_RMSE	TS_MAE	TS_MSE	TS_SMAPE	TS_RMSE
LSTM 1 Layer	0.042712	0.004984	0.777289	0.070598	0.045937	0.006235	0.813459	0.078963	0.050052	0.006604	0.832758	0.081266
LSTM 2 Layer	0.026366	0.001953	0.696648	0.044189	0.047017	0.007809	0.843733	0.088370	0.050807	0.008168	0.860081	0.090379
LSTM 3 Layer	0.024339	0.001593	0.643558	0.039914	0.046391	0.007559	0.799654	0.086945	0.050856	0.008125	0.829802	0.090138
Bidirectional LSTM	0.041587	0.004981	0.767284	0.070579	0.042403	0.006008	0.772889	0.077514	0.046661	0.006357	0.795871	0.079734
Conv 1D	0.046715	0.005998	0.818791	0.077447	0.040288	0.006055	0.733244	0.077816	0.044540	0.006168	0.763747	0.078539
Conv 1D con LSTM	0.043336	0.005742	0.768011	0.075776	0.036568	0.005879	0.671132	0.076676	0.041890	0.006207	0.710687	0.078782
ConvLSTM1D	0.044867	0.005819	0.798691	0.076279	0.042067	0.005817	0.781959	0.076272	0.045807	0.005980	0.803610	0.077330
ConvLSTM2D	0.043305	0.005785	0.779046	0.076061	0.038873	0.005860	0.729254	0.076554	0.042697	0.005990	0.754501	0.077396
LSTM Encode Decode	0.030887	0.002782	0.689927	0.052747	0.045182	0.007080	0.801130	0.084145	0.049414	0.007564	0.821169	0.086972
GRU 1 Layer	0.036775	0.003672	0.760527	0.060597	0.046959	0.007177	0.831508	0.084714	0.050762	0.007493	0.846767	0.086559
GRU 2 Layer	0.030674	0.002505	0.744953	0.050046	0.046149	0.007227	0.840795	0.085010	0.050276	0.007803	0.858957	0.088335
LSTM unrestricted 1 layer	0.042083	0.005560	0.753438	0.074567	0.040184	0.005715	0.748732	0.075595	0.044588	0.006026	0.773648	0.077624
LSTM unrestricted 2 layer	0.037245	0.004314	0.735883	0.065678	0.040754	0.006265	0.754671	0.079152	0.046216	0.006869	0.795169	0.082881

Tabla 4.2. Tabla de resultados del entrenamiento de los modelos de la hora con el método de ventana de varios pasos usando sólo el valor.

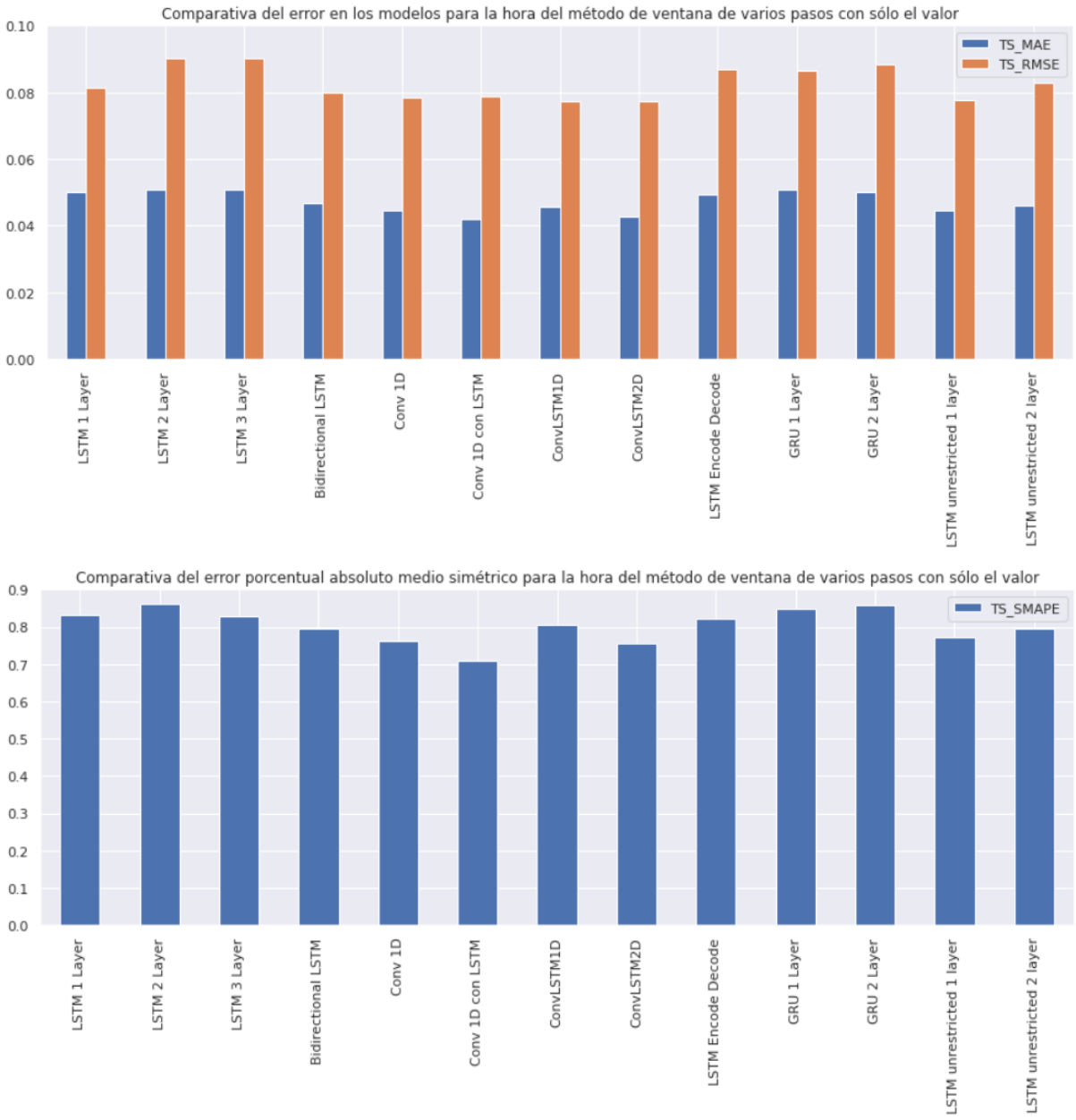


Figura 4.27. Gráficas comparativas de los resultados del entrenamiento para la hora con el método de ventana de varios pasos usando sólo el valor.

Observando sólo el error se puede deducir que uno de los mejores modelos es el ConvLSTM1D, pero no hay que fiarse sólo en estas métricas. Si se analiza el comportamiento de este modelo mediante la visualización gráfica de las predicciones con los datos de test para aproximadamente las primeras 120 y 500 ventanas de predicciones (Figura 4.28), representando mediante el gráfico lineal azul los valores reales y los puntos rojos los valores que se han generado como predicción en cada una de las ventanas, se puede observar que este modelo no se adapta bien a los picos de consumo que existen, y el error bajo se debe a que se ajusta a los valores mayoritarios en el consumo entre 0 y 0.1. Hay que tener en cuenta

que esta gráfica no se puede representar como un gráfico lineal porque los datos se dividen de la siguiente forma, por lo que al predecir se obtendrán varias predicciones para la misma fecha, por lo que es mejor representarlo con una nube de puntos:

1. X1 X2 X3 X4 X5 X6 X7 X8 X9 X10 X11 X12 -> X13 X14 X15 X16

2. X2 X3 X4 X5 X6 X7 X8 X9 X10 X11 X12 X13 -> X14 X15 X16 X17

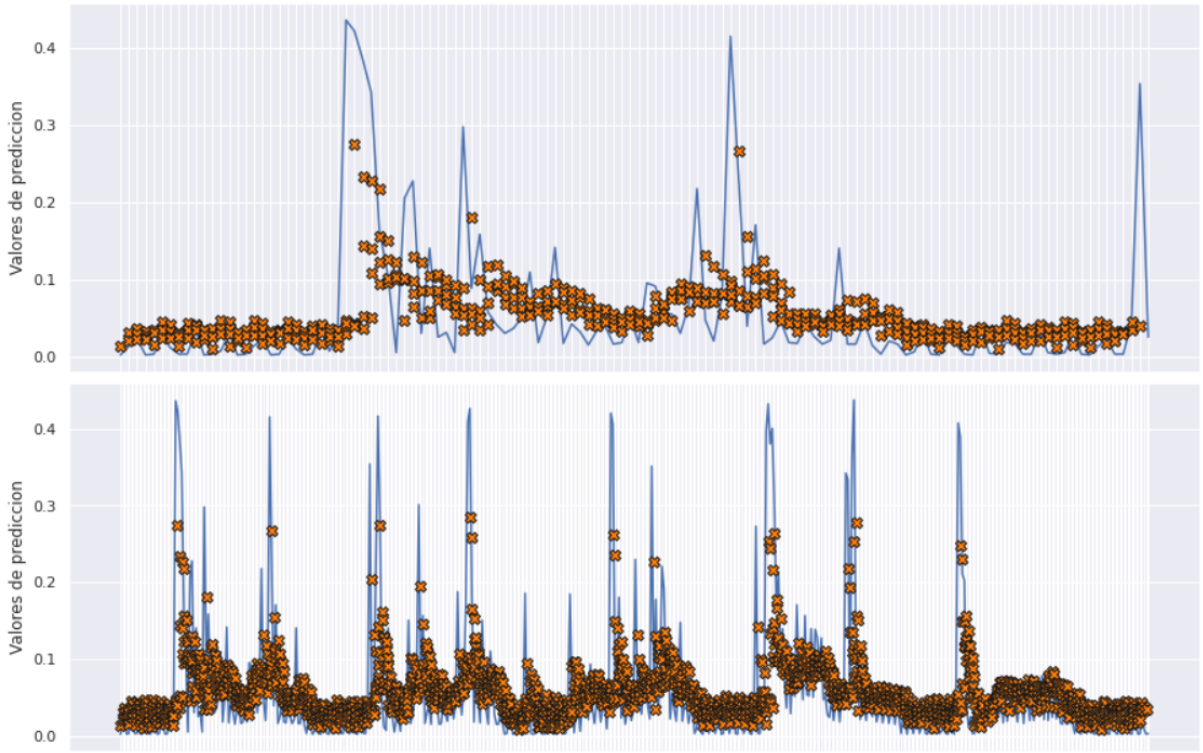


Figura 4.28. Muestra de aproximadamente 120 (arriba) y 500 (abajo) predicciones del modelo CONVLSTM1D para la hora con el método de entrada de la ventana de varios pasos con sólo el valor.

Sin embargo, analizando el comportamiento de los demás modelos, se puede observar que el modelo GRU de 2 capas tiene un comportamiento mejor para estos picos que se comentan (Figura 4.29), y por eso posee el error más alto, ya que se desvía más de los valores típicos que los otros modelos. Este último modelo será el elegido para este tipo de entrada.

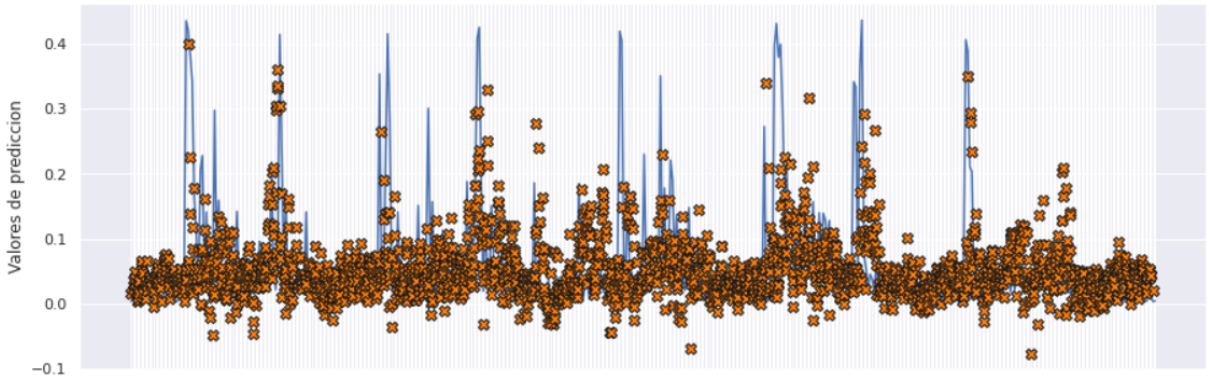


Figura 4.29. Muestra de aproximadamente 120 (arriba) y 500 (abajo) predicciones del modelo GRU de 2 capas para la hora con el método de entrada de la ventana de varios pasos con sólo el valor.

Igualmente, si ahora se comprueban los resultados para la entrada con los campos de la fecha y el valor, se obtienen los siguientes resultados:

Tabla de comparación de todos los modelos - Método de ventana fecha y valor 12 - 4

	TR_MAE	TR_MSE	TR_SMAPE	TR_RMSE	V_MAE	V_MSE	V_SMAPE	V_RMSE	TS_MAE	TS_MSE	TS_SMAPE	TS_RMSE
LSTM 1 Layer	0.028097	0.002235	0.649124	0.047279	0.053982	0.010051	0.884097	0.100254	0.053692	0.008574	0.867815	0.092596
LSTM 2 Layer	0.018853	0.000862	0.561423	0.029361	0.053438	0.009550	0.897113	0.097724	0.056985	0.009309	0.910084	0.096485
LSTM 3 Layer	0.013552	0.000429	0.474585	0.020724	0.049342	0.008773	0.856427	0.093662	0.051583	0.008111	0.854479	0.090059
Bidirectional LSTM	0.027925	0.001946	0.665205	0.044112	0.052171	0.008979	0.871940	0.094760	0.053371	0.008219	0.860490	0.090657
Conv 1D	0.039710	0.004783	0.741040	0.069156	0.045010	0.006358	0.789538	0.079740	0.046048	0.006120	0.778845	0.078234
Conv 1D con LSTM	0.032745	0.003062	0.684594	0.055338	0.054632	0.009909	0.880072	0.099542	0.051682	0.008221	0.831791	0.090672
ConvLSTM1D	0.040869	0.004715	0.735339	0.068666	0.044845	0.006172	0.789327	0.078563	0.047706	0.006065	0.796006	0.077876
ConvLSTM2D	0.037488	0.004479	0.717141	0.066924	0.045188	0.006611	0.858902	0.081308	0.046904	0.006292	0.830365	0.079323
LSTM Encode Decode	0.023551	0.001495	0.595364	0.038668	0.053117	0.010500	0.815615	0.102468	0.052238	0.008867	0.801247	0.094163
GRU 1 Layer	0.028709	0.002151	0.682574	0.046377	0.054209	0.009667	0.922015	0.098322	0.055473	0.008983	0.884485	0.094780
GRU 2 Layer	0.016552	0.000590	0.552420	0.024287	0.050712	0.008899	0.870367	0.094336	0.053675	0.008429	0.894318	0.091808
LSTM unrestricted 1 layer	0.036688	0.003685	0.718178	0.060706	0.051711	0.008643	0.859435	0.092970	0.053147	0.007920	0.836711	0.088994
LSTM unrestricted 2 layer	0.020746	0.001111	0.589869	0.033324	0.055145	0.010584	0.869345	0.102877	0.054695	0.008940	0.863721	0.094550

Tabla 4.3. Tabla de resultados del entrenamiento de los modelos de la hora con el método de ventana de varios pasos usando la fecha y el valor.

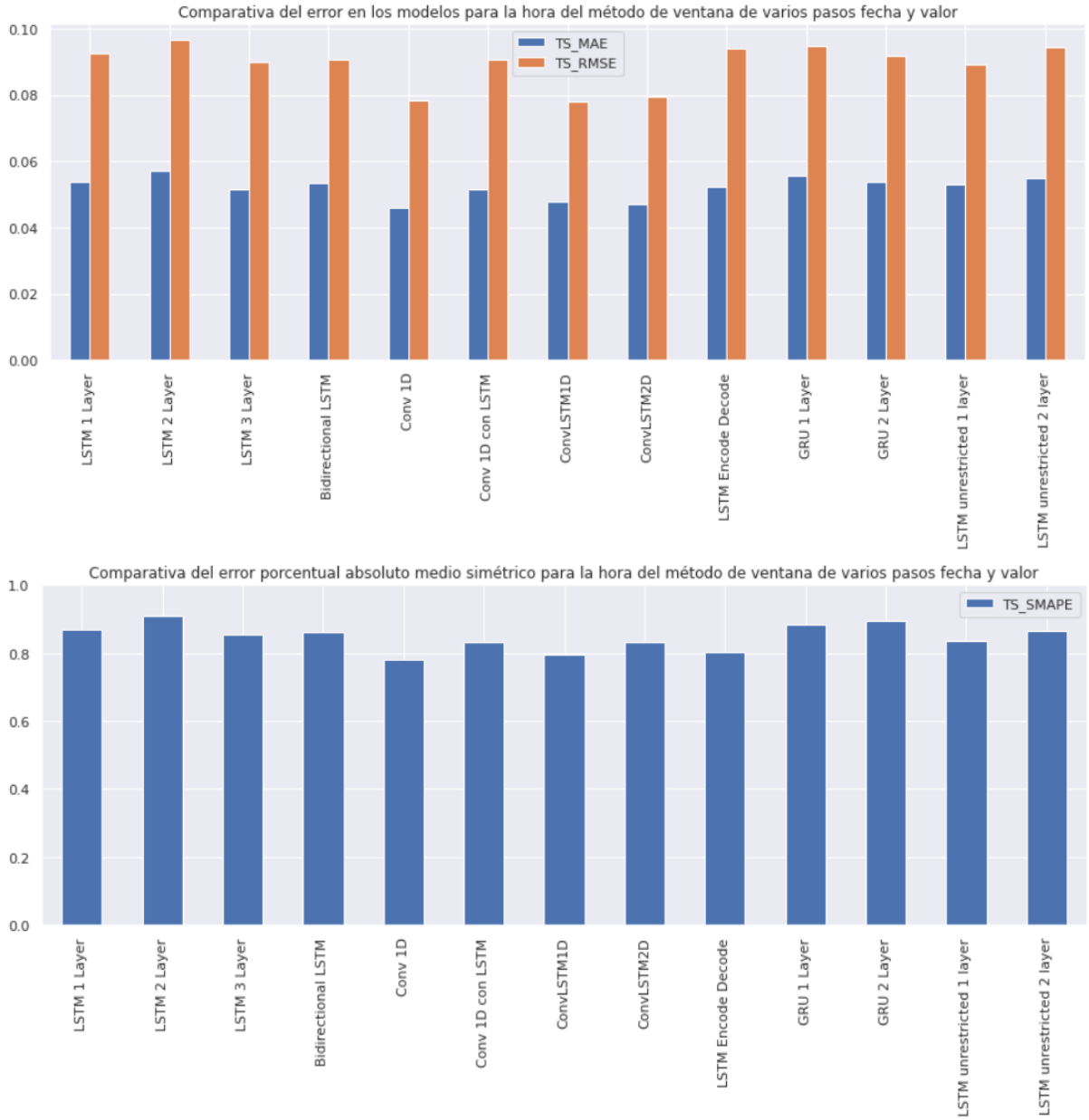


Figura 4.30. Gráficas comparativas de los resultados del entrenamiento para la hora con el método de ventana de varios pasos usando la fecha y el valor.

De nuevo, una de los modelos con el error más bajo es el ConvLSTM1D, por lo que se comprueba gráficamente como antes como de bien se adapta al consumo (Figura 4.31):

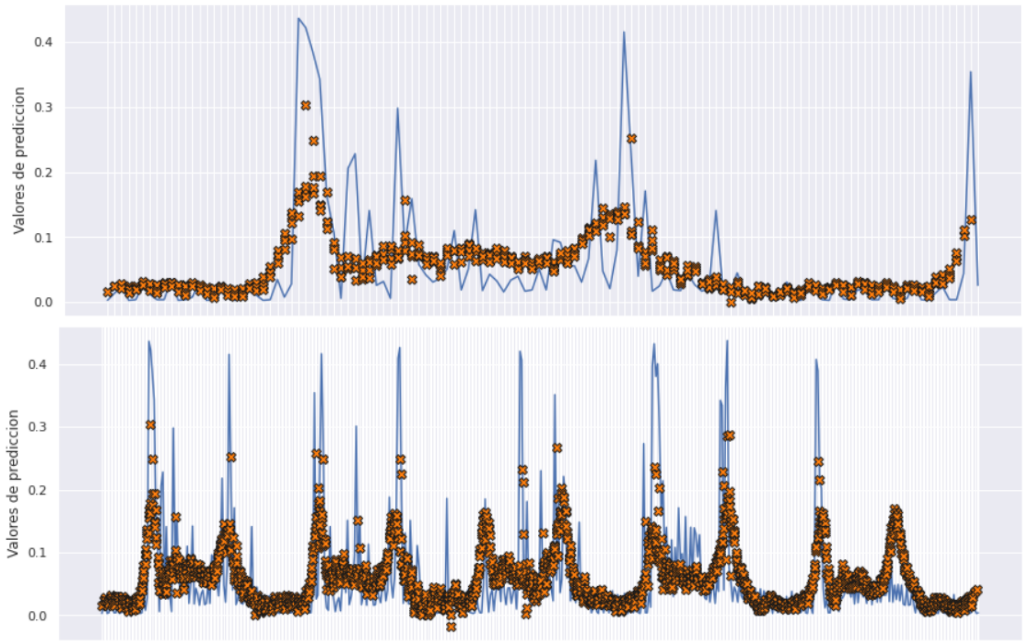


Figura 4.31. Muestra de aproximadamente 120 (arriba) y 500 (abajo) predicciones del modelo ConvLSTM1D para la hora con el método de entrada de la ventana de varios pasos con la fecha y el valor.

Como pasó anteriormente, tener el error más bajo no significa tener un mejor ajuste, y más que comprobando los demás modelos, en este caso LSTM de 2 capas es el modelo que mejor se adapta a los picos (Figura 4.32) aunque tenga un RMSE de los más altos, por lo que será este el elegido. También se puede notar que existe una mejoría respecto a la entrada de sólo valor, por lo que será algo importante a tener en cuenta.

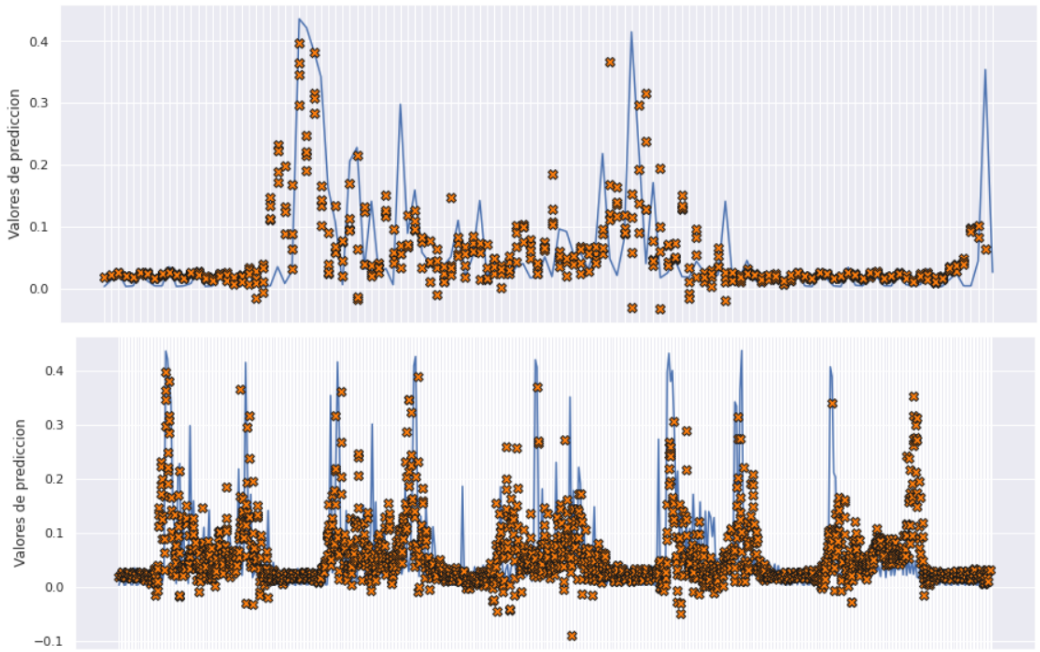


Figura 4.32. Muestra de aproximadamente 120 (arriba) y 500 (abajo) predicciones del modelo LSTM de 2 capas para la hora con el método de entrada de la ventana de varios pasos con la fecha y el valor.

4.3.4.2. Análisis de resultados para el día

Respecto a la predicción del día, la entrada ha sido de 288 valores, correspondientes a los 3 días anteriores y la salida de 96 valores para un día completo. Para la predicción de los días mediante la entrada de sólo los valores se obtienen los siguientes resultados (Tabla 4.4, Figura 4.33):

Tabla de comparación de todos los modelos - Método de ventana sólo valor 288 - 96

	TR_MAE	TR_MSE	TR_SMAPE	TR_RMSE	V_MAE	V_MSE	V_SMAPE	V_RMSE	TS_MAE	TS_MSE	TS_SMAPE	TS_RMSE
LSTM 1 Layer	0.033896	0.003065	0.734225	0.055359	0.047379	0.008356	0.820066	0.091409	0.047902	0.007712	0.817422	0.087817
LSTM 2 Layer	0.027777	0.001828	0.678928	0.042757	0.044293	0.007981	0.764885	0.089337	0.046457	0.007536	0.782035	0.086808
LSTM 3 Layer	0.027499	0.001771	0.680533	0.042084	0.045645	0.008019	0.781881	0.089551	0.049305	0.008070	0.811237	0.089835
Bidirectional LSTM	0.032435	0.002594	0.731026	0.050934	0.046645	0.007699	0.826987	0.087746	0.048129	0.007335	0.822940	0.085643
Conv 1D	0.043862	0.005332	0.798541	0.073018	0.044440	0.006356	0.800184	0.079722	0.046746	0.006148	0.812746	0.078410
Conv 1D con LSTM	0.038977	0.004026	0.781880	0.063448	0.053306	0.008358	0.954356	0.091423	0.050812	0.007161	0.902259	0.084622
ConvLSTM1D	0.043255	0.005261	0.795332	0.072534	0.044318	0.006384	0.794958	0.079897	0.046462	0.006169	0.802714	0.078543
ConvLSTM2D	0.042807	0.005082	0.806269	0.071285	0.044421	0.006426	0.811450	0.080163	0.046938	0.006297	0.825686	0.079353
LSTM Encode Decode	0.030513	0.002651	0.666455	0.051483	0.045832	0.008384	0.742376	0.091564	0.047073	0.007694	0.762502	0.087715
GRU 1 Layer	0.036665	0.003680	0.752455	0.060661	0.046023	0.007893	0.800694	0.088842	0.048603	0.007795	0.822848	0.088291
GRU 2 Layer	0.029294	0.002068	0.696164	0.045478	0.044985	0.007967	0.775634	0.089256	0.046824	0.007458	0.793172	0.086359
LSTM unrestricted 1 layer	0.117047	0.025971	1.340808	0.161155	0.120915	0.028076	1.380112	0.167558	0.120423	0.027114	1.366780	0.164664
LSTM unrestricted 2 layer	0.005620	0.000054	0.330443	0.007349	0.061058	0.009354	1.102024	0.096716	0.062559	0.009201	1.093998	0.095924

Tabla 4.4. Tabla de resultados del entrenamiento de los modelos del día con el método de ventana de varios pasos usando sólo el valor.

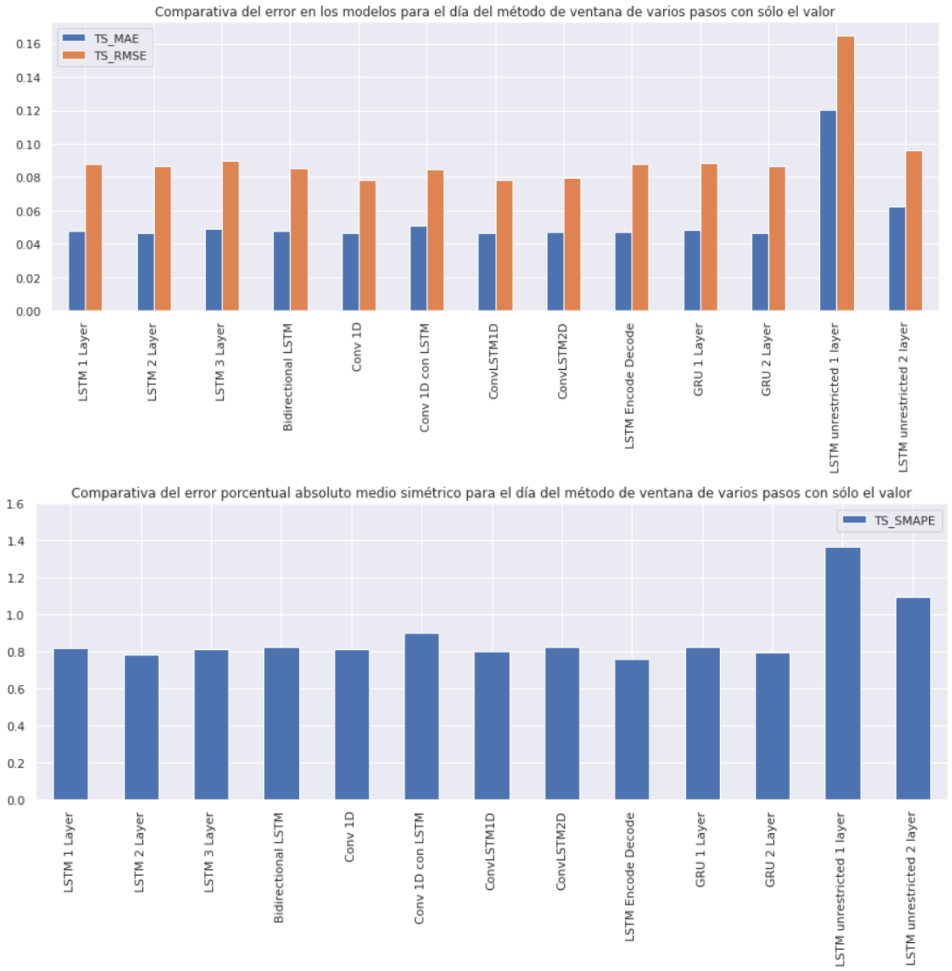


Figura 4.33. Gráficas comparativas de los resultados del entrenamiento para el día con el método de ventana de varios pasos usando sólo el valor.

En esta ocasión, había varios modelos que se adaptaban bien a las características de consumo del día, pero el seleccionado ha sido el GRU de una capa, y no se caracteriza por tener el error más alto, pero se comporta de la siguiente manera adaptándose muy bien a los valores reales (Figura 4.34):

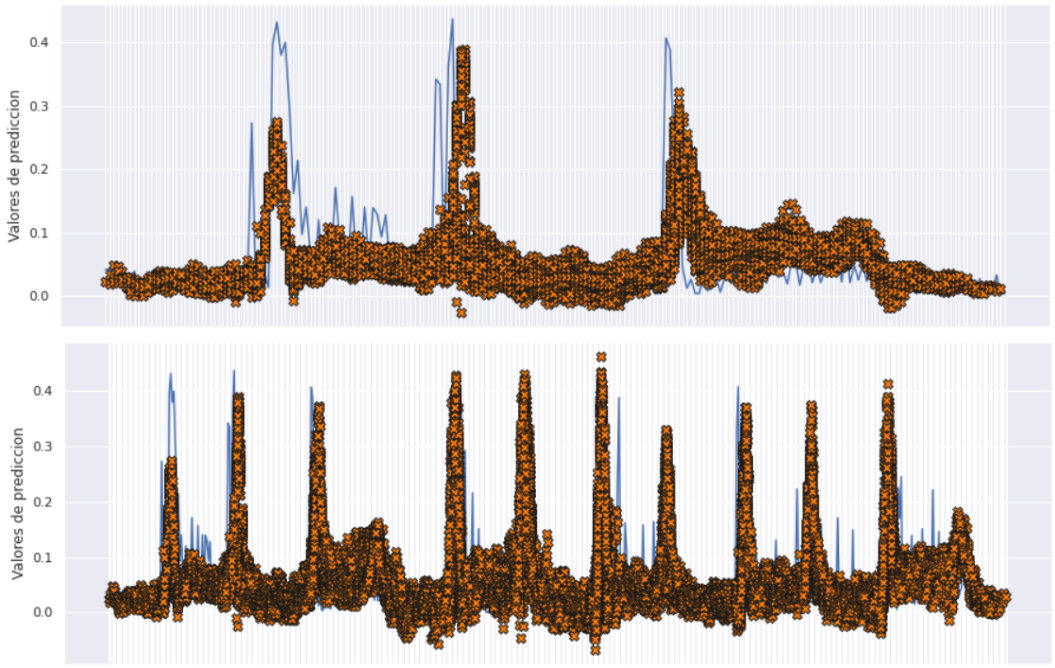


Figura 4.34. Muestra de aproximadamente 120 (arriba) y 500 (abajo) predicciones del modelo GRU de una capa para el día con el método de entrada de la ventana de varios pasos sólo con el valor.

En cuanto al entrenamiento con la entrada de la fecha y el valor, se obtienen los siguientes resultados (Tabla 4.5, Figura 4.35):

Tabla de comparación de todos los modelos - Método de ventana fecha y valor 288 - 96

	TR_MAE	TR_MSE	TR_SMAPE	TR_RMSE	V_MAE	V_MSE	V_SMAPE	V_RMSE	TS_MAE	TS_MSE	TS_SMAPE	TS_RMSE
LSTM 1 Layer	0.032098	0.002598	0.717653	0.050970	0.047730	0.008442	0.852143	0.091881	0.049294	0.007532	0.831647	0.086790
LSTM 2 Layer	0.025493	0.001467	0.659625	0.038302	0.050006	0.009351	0.870154	0.096698	0.052123	0.008299	0.887368	0.091097
LSTM 3 Layer	0.022246	0.001096	0.624587	0.033110	0.052163	0.009556	0.911267	0.097754	0.052901	0.008086	0.926417	0.089921
Bidirectional LSTM	0.030160	0.002220	0.710397	0.047113	0.049836	0.008849	0.878539	0.094071	0.049658	0.007571	0.846647	0.087013
Conv 1D	0.040407	0.003672	0.886493	0.060598	0.059936	0.008754	1.144560	0.093563	0.056191	0.007427	1.070987	0.086179
Conv 1D con LSTM	0.028810	0.001884	0.719854	0.043410	0.065204	0.011022	1.105981	0.104985	0.061945	0.009105	1.048523	0.095421
ConvLSTM1D	0.036549	0.002660	0.882804	0.051573	0.076538	0.011718	1.254549	0.108250	0.070054	0.009766	1.173037	0.098825
ConvLSTM2D	0.036310	0.002635	0.877007	0.051329	0.074289	0.011337	1.248686	0.106477	0.063600	0.008773	1.168804	0.093663
LSTM Encode Decode	0.023290	0.001389	0.608997	0.037267	0.051358	0.010537	0.816113	0.102652	0.051400	0.009046	0.797016	0.095113
GRU 1 Layer	0.033970	0.002977	0.750240	0.054558	0.048617	0.008658	0.886895	0.093049	0.049934	0.007712	0.852758	0.087819
GRU 2 Layer	0.025622	0.001486	0.678550	0.038548	0.050379	0.008960	0.919760	0.094658	0.051719	0.008101	0.915161	0.090007
LSTM unrestricted 1 layer	0.022566	0.000919	0.717643	0.030308	0.075689	0.012726	1.234899	0.112809	0.076854	0.012083	1.223194	0.109924
LSTM unrestricted 2 layer	0.004145	0.000029	0.257462	0.005400	0.068290	0.010726	1.188506	0.103566	0.067427	0.009764	1.156759	0.098811

Tabla 4.5. Tabla de resultados del entrenamiento de los modelos del día con el método de ventana de varios pasos usando la fecha y el valor.

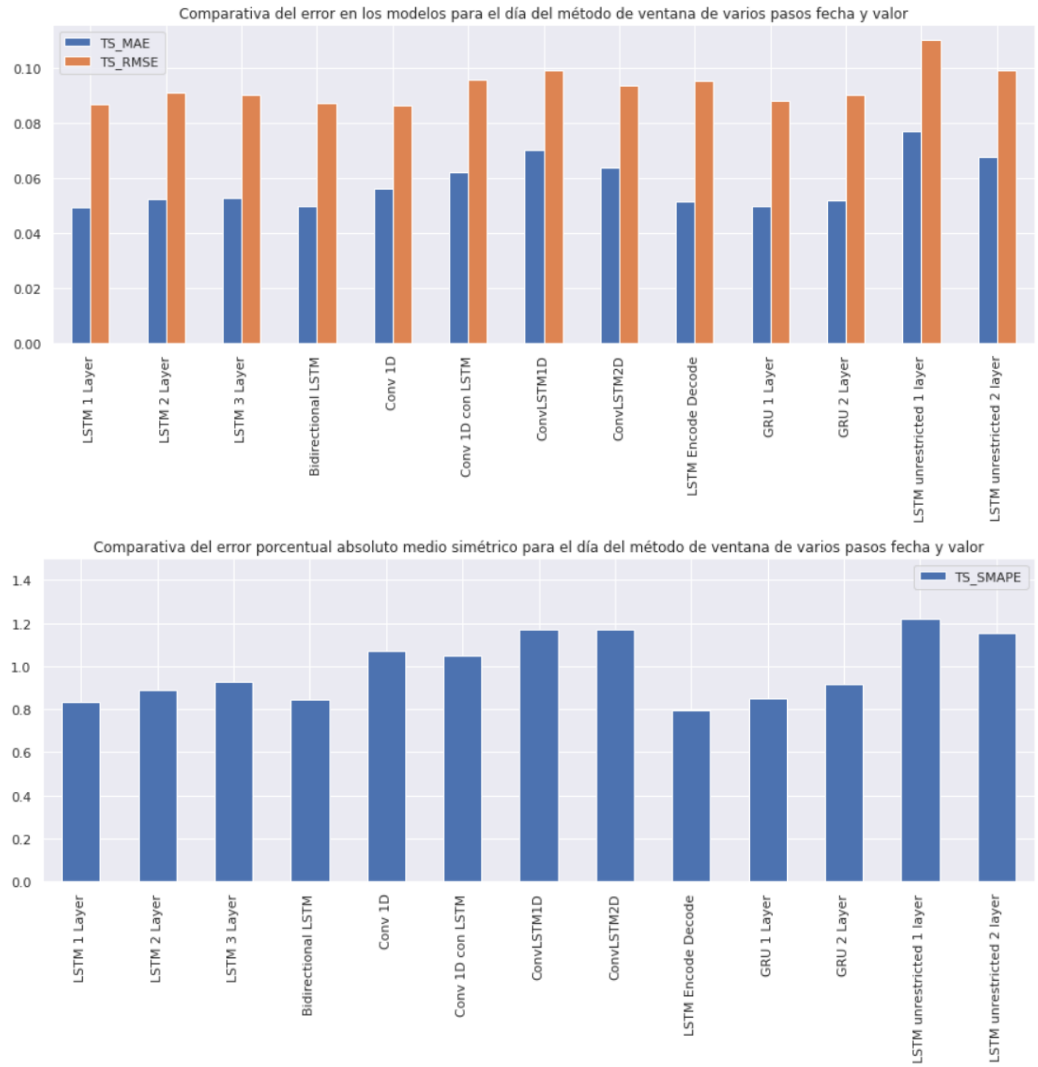


Figura 4.35. Gráficas comparativas de los resultados del entrenamiento para el día con el método de ventana de varios pasos usando la fecha y el valor.

Una vez analizado el comportamiento de cada modelo en la gráfica comparativa, en esta ocasión se deduce que LSTM de dos capas es el que mejor se adapta al consumo tanto en los valores bajos como en los picos para este tipo de entrada de fecha y valor (Figura 4.36). De nuevo, se vuelve a comprobar que el método de la fecha y valor produce resultados ligeramente más precisos que pasándo sólo el valor.

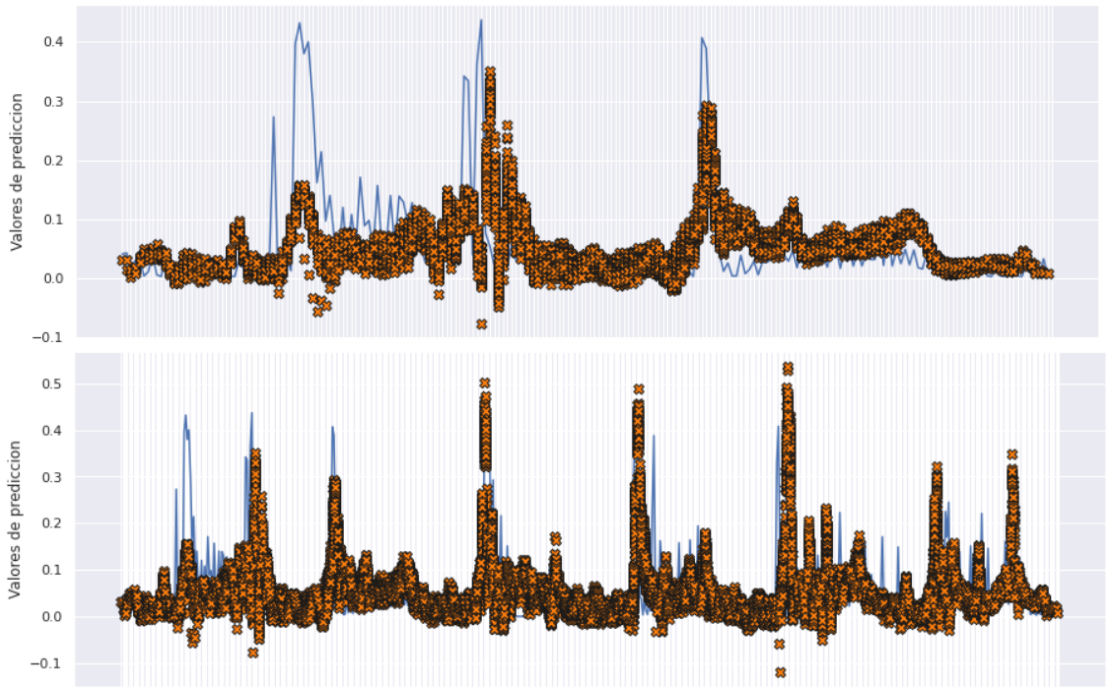


Figura 4.36. Muestra de aproximadamente 120 (arriba) y 500 (abajo) predicciones del modelo LSTM de 2 capas para el día con el método de entrada de la ventana de varios pasos con la fecha y el valor.

4.3.4.3. Análisis de resultados para la semana

Para predecir la semana, la entrada que se ha aplicado ha sido de 2016 valores, correspondiente a las 3 semanas anteriores y 672 para la salida de la semana. Esta vez, la partida de modelos a probar no ha sido tan extensa como en las anteriores debido al gran tamaño de la ventana que provocaba un gran aumento del tiempo de entrenamiento. Por ello, se ha optado por utilizar los modelos más prometedores de las comprobaciones anteriores para evitar así el uso excesivo de recursos computacionales que provocaba en ocasiones que el kernel se quedara sin memoria y tuviera que reiniciarse el entrenamiento. Los tres modelos elegidos han sido LSTM de 2 capas, GRU de 1 capa y GRU de 2 capas.

Una vez realizado el entrenamiento con el método de entrada de sólo el valor, los resultados obtenidos han sido los siguientes (Tabla 4.6, Figura 4.37):

Tabla de comparación de todos los modelos - Método de ventana sólo valor 2016 - 672

	TR_MAE	TR_MSE	TR_SMAPE	TR_RMSE	V_MAE	V_MSE	V_SMAPE	V_RMSE	TS_MAE	TS_MSE	TS_SMAPE	TS_RMSE
LSTM 2 Layer	0.040939	0.004864	0.749622	0.069741	0.046834	0.007117	0.816675	0.084365	0.046730	0.006586	0.774191	0.081154
GRU 1 Layer	0.047718	0.006322	0.838204	0.079513	0.049572	0.007044	0.875857	0.083926	0.048327	0.006410	0.824069	0.080065
GRU 2 Layer	0.041572	0.005073	0.753697	0.071222	0.046413	0.006880	0.821196	0.082946	0.045546	0.006254	0.761041	0.079084

Tabla 4.6. Tabla de resultados del entrenamiento de los modelos de la semana con el método de ventana de varios pasos usando sólo el valor.

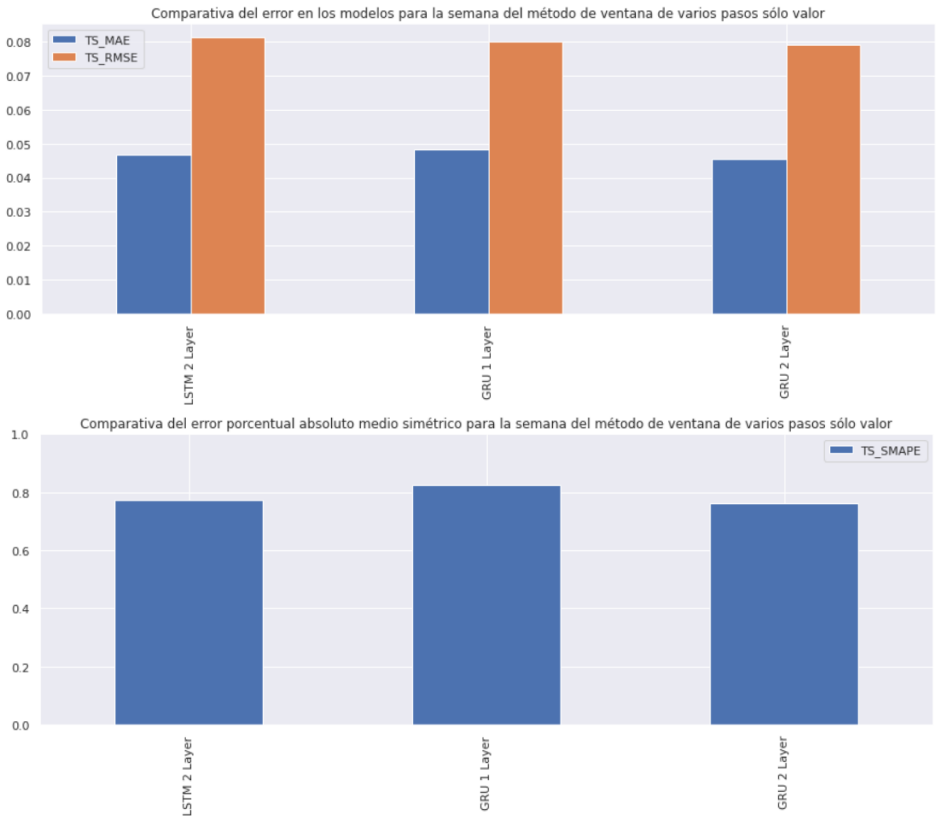


Figura 4.37. Gráficas comparativas de los resultados del entrenamiento para la semana con el método de ventana de varios pasos usando sólo el valor.

Como se puede observar en la Figura 4.38, parece que para entradas y salidas muy largas, estos modelos no actúan bien, ya que se puede observar una periodicidad en las predicciones y no se adaptan bien a los picos de consumo como lo hicieron las veces anteriores. Habría que analizar bien por qué se produce esta periodicidad en las predicciones, que puede ser debido a que la cantidad de datos de entrenamiento para este tipo de entrada sea insuficiente o el modelo no se adapte bien a ventanas de datos tan grandes. Una vez que se obtengan los datos reales del Smart Meter, habría que probar cómo se comporta el modelo e incluso se podría valorar una reducción de la ventana agrupando los datos en meses o días en el peor de los casos, pero esto significaría una pérdida de la información respecto al comportamiento exacto del consumo, ya que si en una hora se produce un valor muy alto y a la vez otro muy bajo, al final el resultado será un punto intermedio y no representará el pico que ha ocurrido. En este caso, claramente, es el modelo GRU de 2 capas el que mejor se adapta.

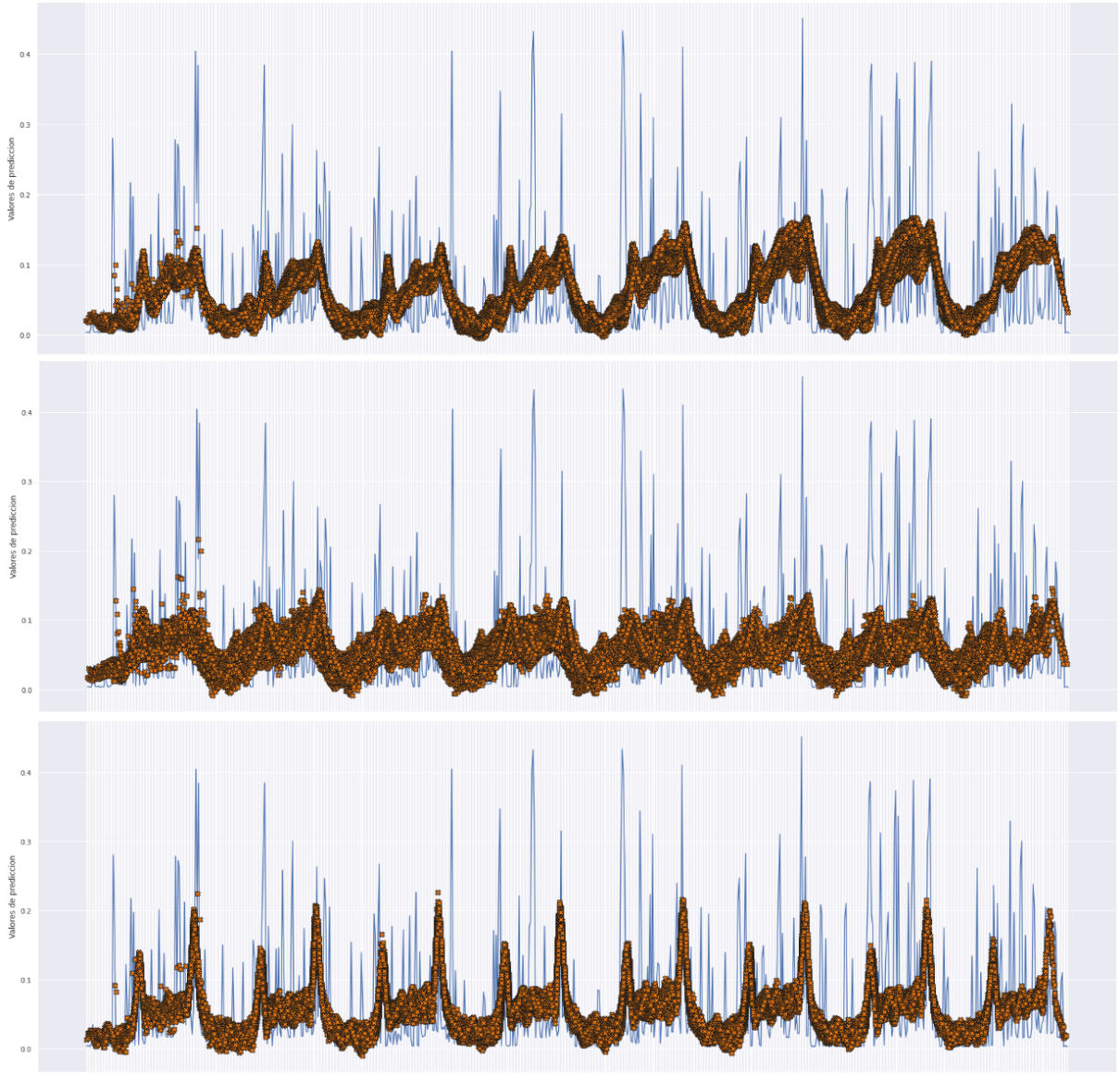


Figura 4.38. Resultados de predicciones para la semana del método de entrada sólo valor de los modelos LSTM de dos capas (arriba), GRU 1 capa (en medio) y GRU 2 capas (abajo).

En cuanto al método de entrada con la fecha y el valor se han obtenido los siguientes resultados (Tabla 4.7, Figura 4.39):

Tabla de comparación de todos los modelos - Método de ventana fecha y valor 2016 - 672

	TR_MAE	TR_MSE	TR_SMAPE	TR_RMSE	V_MAE	V_MSE	V_SMAPE	V_RMSE	TS_MAE	TS_MSE	TS_SMAPE	TS_RMSE
LSTM 2 Layer	0.039301	0.004068	0.786912	0.063782	0.051573	0.008178	0.952437	0.090435	0.049109	0.006894	0.848378	0.083027
GRU 1 Layer	0.041086	0.004943	0.752634	0.070304	0.047698	0.007633	0.834222	0.087369	0.046529	0.006394	0.774888	0.079961
GRU 2 Layer	0.040369	0.004603	0.763414	0.067843	0.048533	0.007529	0.879319	0.086767	0.047450	0.006685	0.796321	0.081760

Tabla 4.7. Tabla de resultados del entrenamiento de los modelos de la semana con el método de ventana de varios pasos usando la fecha y el valor.

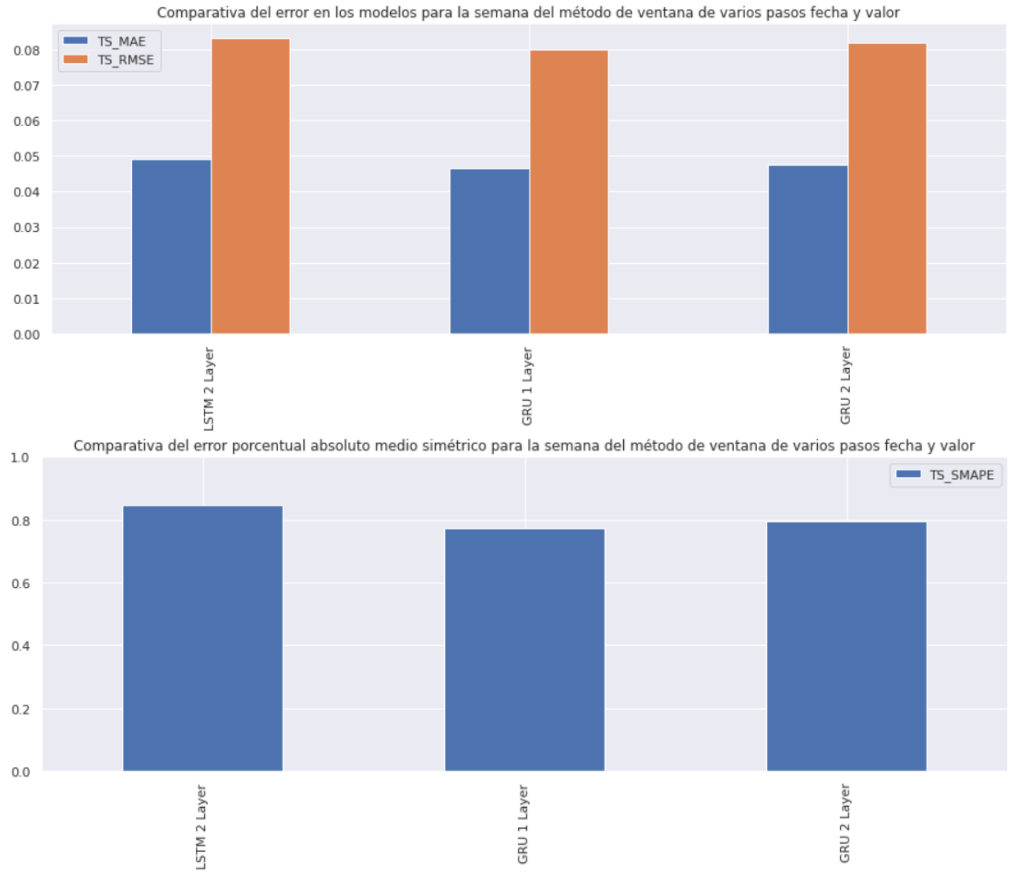


Figura 4.39. Gráficas comparativas de los resultados del entrenamiento para la semana con el método de ventana de varios pasos usando la fecha y el valor.

Usando la fecha y el valor se nota un poco de mejoría respecto a una entrada con sólo el valor (Figura 4.39), aunque excepto en LSTM que parece no estar tan remarcada, se sigue mostrando una periodicidad en los picos de consumo de las predicciones. Aquí habría que descartar que por el uso de la fecha se esté produciendo la periodicidad ya que también pasa cuando se usa sólo el valor, por lo que el error no viene de ahí, sino de una insuficiencia de datos o de un tamaño excesivo en la ventana. De todas formas, LSTM de 2 capas es el que mejor se adapta y no presenta una periodicidad totalmente clara como en los demás modelos, por eso será el elegido.

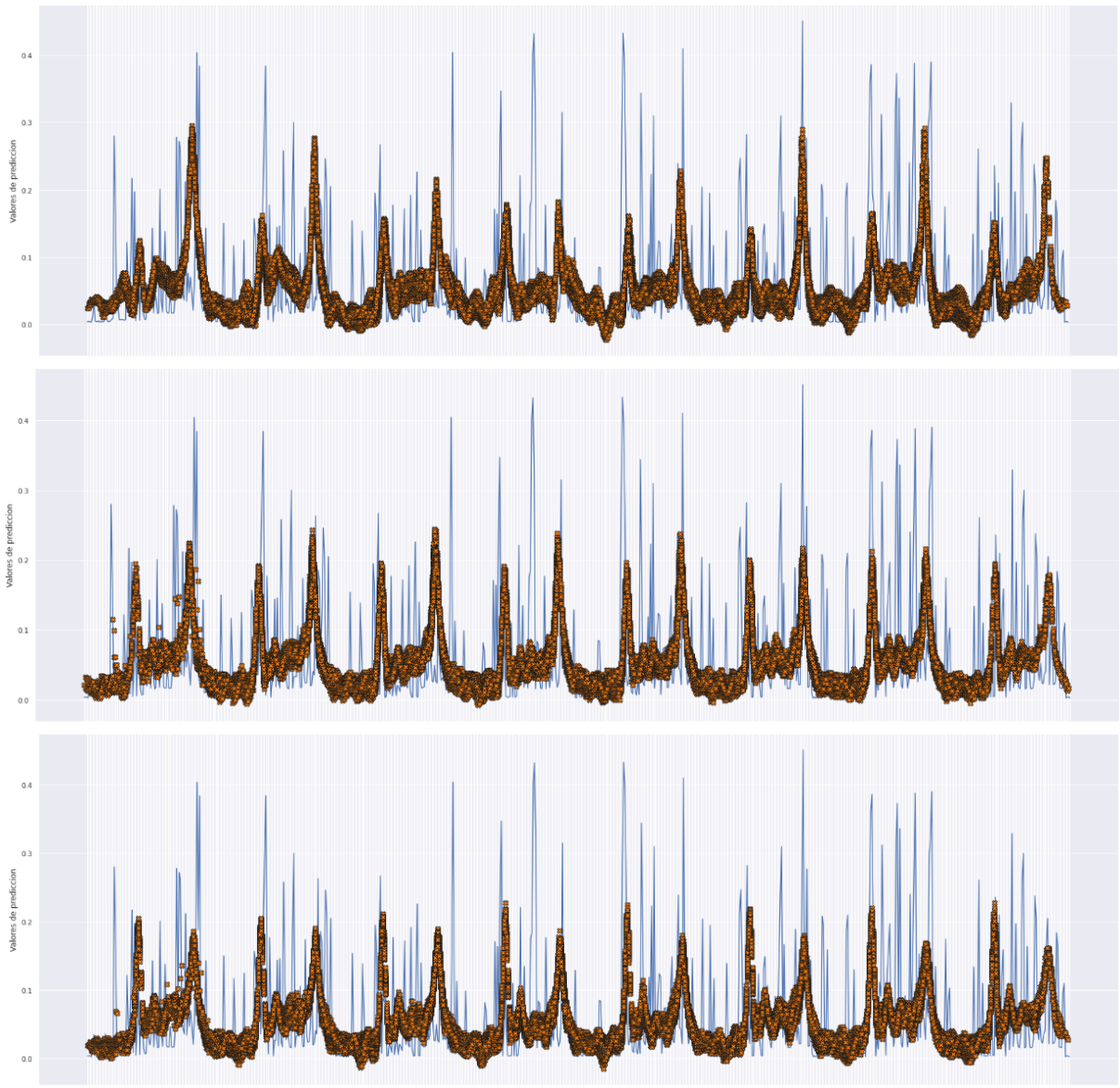


Figura 4.40. Resultados de predicciones para la semana del método de entrada con la fecha y el valor de los modelos LSTM de dos capas (arriba), GRU 1 capa (en medio) y GRU 2 capas (abajo).

4.3.4.4. Análisis de resultados para el mes

Para predecir el mes, la entrada que se ha aplicado ha sido de 8640 valores, correspondiente a los 3 meses anteriores y 2880 para la salida del mes. En este análisis se va a hacer lo mismo que en el apartado anterior, usando sólo los 3 modelos anteriores, ya que la ventana es más grande todavía.

Una vez realizado el entrenamiento con el método de entrada de sólo el valor, los resultados obtenidos han sido los siguientes (Tabla 4.8, Figura 4.41):

Tabla de comparación de todos los modelos - Método de ventana sólo valor 8640 - 2880

	TR_MAE	TR_MSE	TR_SMAPE	TR_RMSE	V_MAE	V_MSE	V_SMAPE	V_RMSE	TS_MAE	TS_MSE	TS_SMAPE	TS_RMSE
LSTM 2 Layer	0.041590	0.005111	0.784326	0.071494	0.050271	0.006680	1.006028	0.081733	0.049132	0.005652	0.984324	0.075181
GRU 1 Layer	0.049412	0.006729	0.893427	0.082031	0.050907	0.007092	1.005355	0.084216	0.051746	0.005963	1.041685	0.077219
GRU 2 Layer	0.052069	0.007288	0.921884	0.085369	0.051288	0.007205	1.009615	0.084880	0.052712	0.006202	1.046003	0.078750

Tabla 4.8. Tabla de resultados del entrenamiento de los modelos del mes con el método de ventana de varios pasos usando sólo el valor.

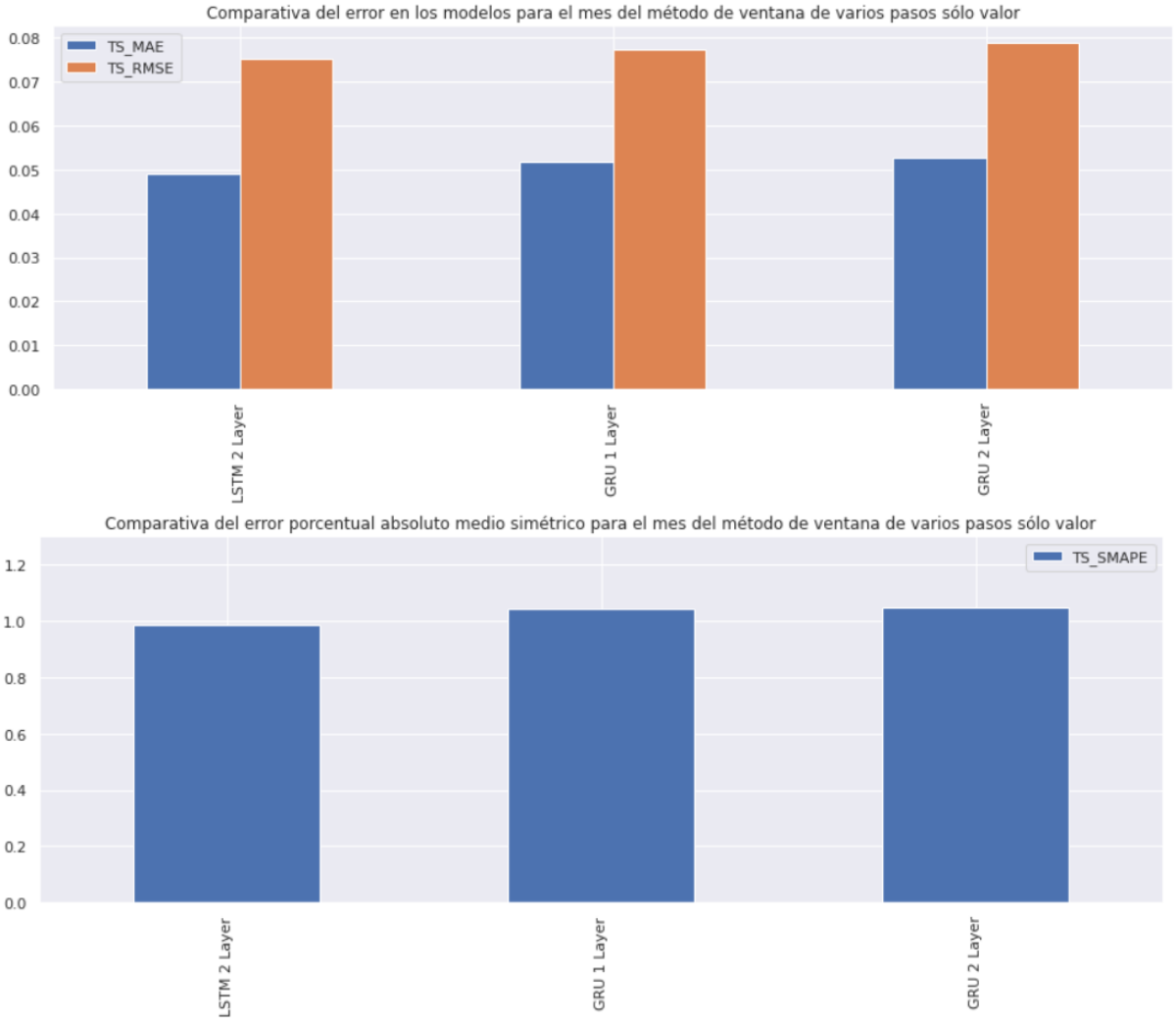


Figura 4.41. Gráficas comparativas de los resultados del entrenamiento para el mes con el método de ventana de varios pasos usando sólo el valor.

De nuevo, observando la Figura 4.42, se puede observar que vuelve a ocurrir la periodicidad en las predicciones ya que la ventana es incluso más grande todavía. Para este método, se elige el modelo GRU de 2 capas aunque no ofrezca un buen resultado, pero servirá como punto de partida para la mejora con los datos extraídos del Smart Meter en el futuro.

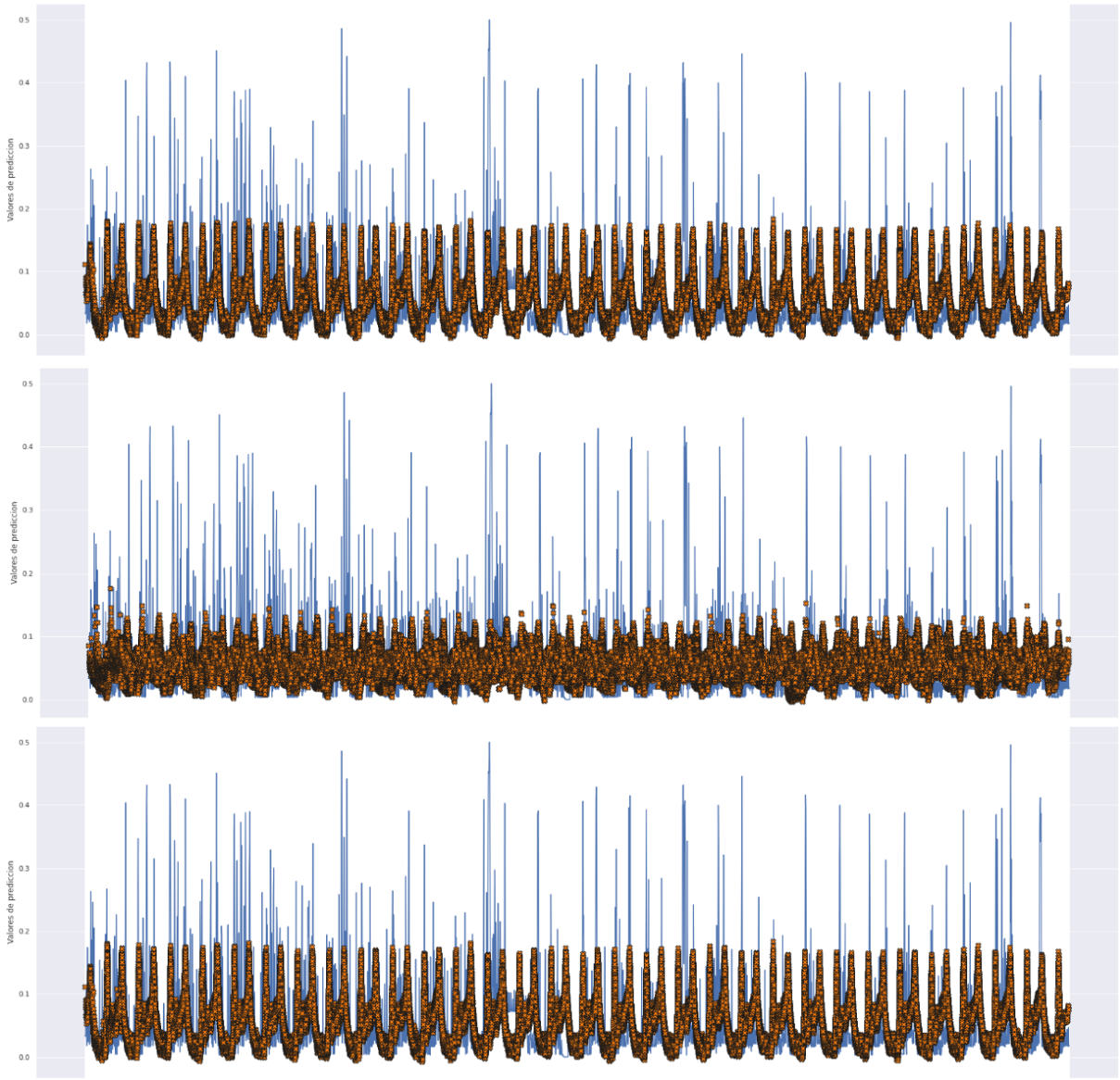


Figura 4.42. Resultados de predicciones para el del método de entrada sólo valor de los modelos LSTM de dos capas (arriba), GRU 1 capa (en medio) y GRU 2 capas (abajo).

En cuanto al método de entrada con la fecha y el valor se han obtenido los siguientes resultados (Tabla 4.9, Figura 4.43):

Tabla de comparación de todos los modelos - Método de ventana fecha y valor 8640 - 2880

	TR_MAE	TR_MSE	TR_SMAPE	TR_RMSE	V_MAE	V_MSE	V_SMAPE	V_RMSE	TS_MAE	TS_MSE	TS_SMAPE	TS_RMSE
LSTM 2 Layer	0.040805	0.004705	0.810241	0.068594	0.049618	0.007874	0.976291	0.088734	0.049178	0.006003	0.962941	0.077479
GRU 1 Layer	0.041054	0.005040	0.779466	0.070991	0.046457	0.007431	0.899662	0.086204	0.048304	0.005874	0.952318	0.076643
GRU 2 Layer	0.040775	0.004893	0.784297	0.069953	0.047713	0.007570	0.926859	0.087005	0.048699	0.005969	0.956804	0.077262

Tabla 4.9. Tabla de resultados del entrenamiento de los modelos del mes con el método de ventana de varios pasos usando la fecha y el valor.

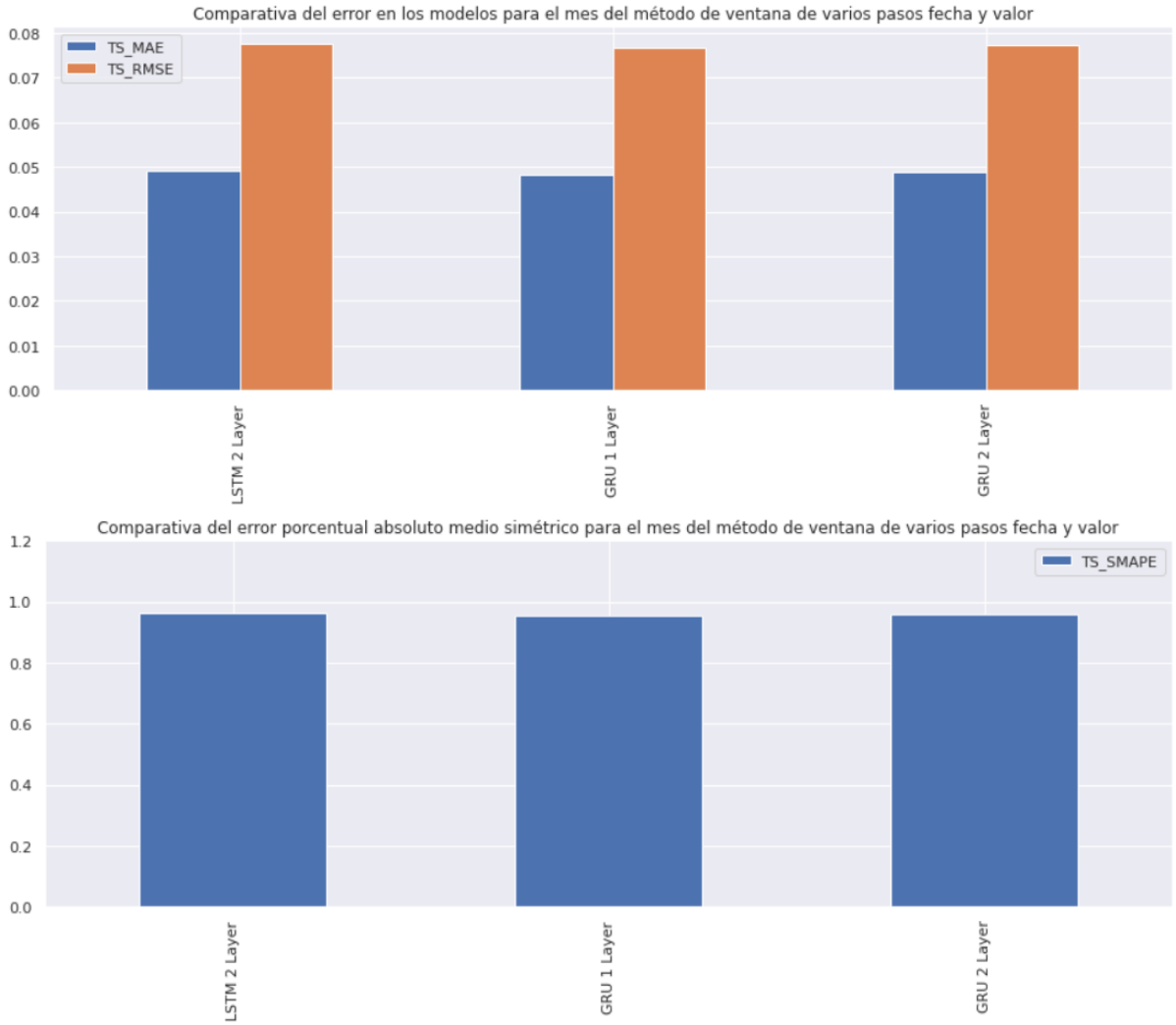


Figura 4.43. Gráficas comparativas de los resultados del entrenamiento para la semana con el método de ventana de varios pasos usando la fecha y el valor.

En este método de entrada con la fecha y el valor (Figura 4.44), de nuevo se nota mejoría respecto a la entrada de sólo valor, pero vuelve a repetirse la periodicidad del apartado anterior. En este caso, se elige LSTM de 2 capas porque al igual que antes, no muestra una periodicidad tan clara como los demás modelos.

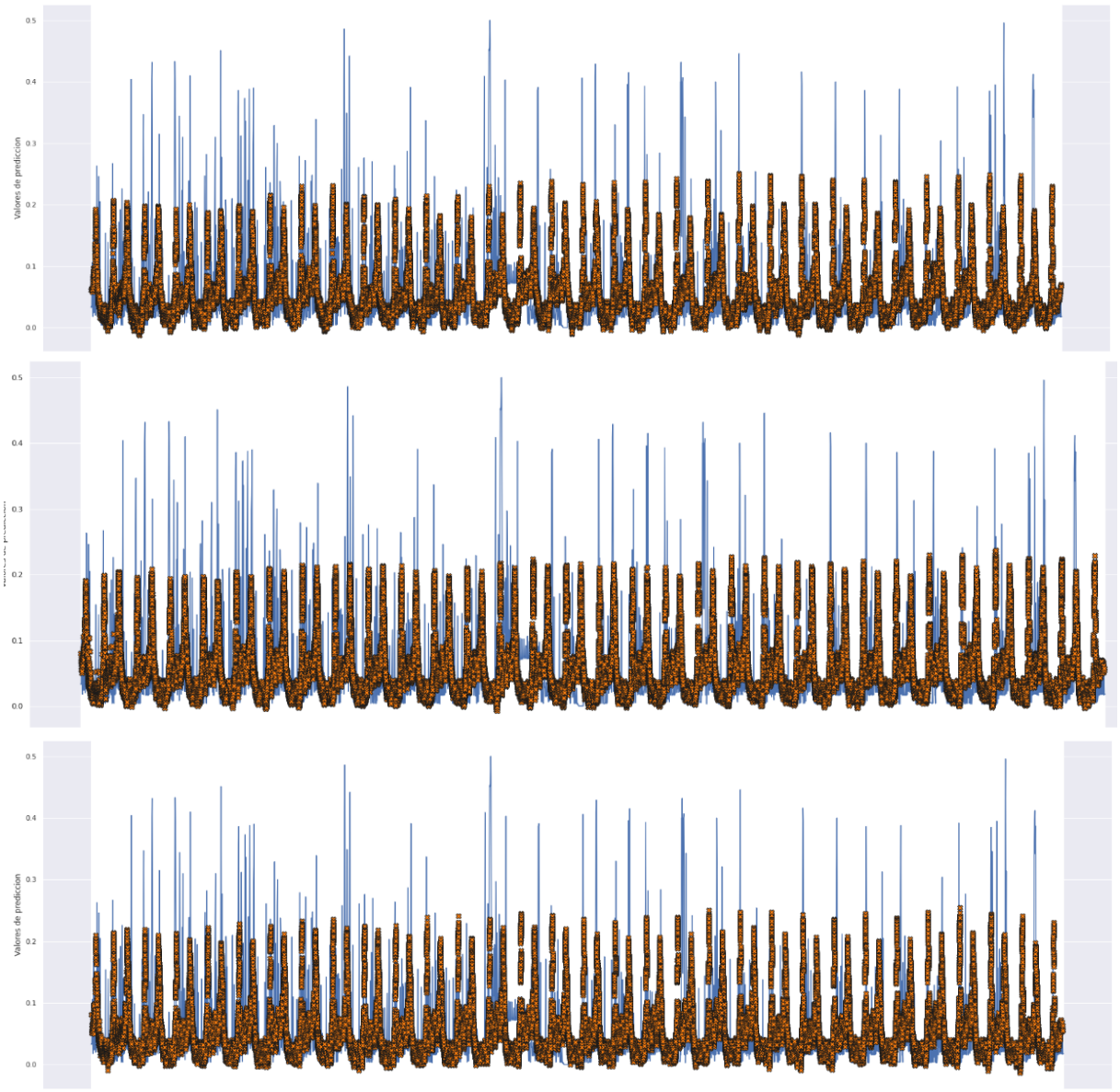


Figura 4.44. Resultados de predicciones para la semana del método de entrada con la fecha y el valor de los modelos LSTM de dos capas (arriba), GRU 1 capa (en medio) y GRU 2 capas (abajo).

4.4. Optimización de los parámetros

Si se analiza la tabla 4.3, se puede observar que para el modelo LSTM de 2 capas, que es el que ha sido elegido para la predicción de la hora con el método de entrada de fecha y valor, existe un valor alto de varianza entre el error de entrenamiento y el error de validación. Para intentar reducir la varianza, se pueden aplicar más datos, pero en este caso no se disponen más datos para ese mismo cliente por lo que no sería posible. Otra de las opciones es aplicar dropout o regularización L1L2. Sin embargo, se ha probado a crear un modelo con dropout a 0.1 en la primera capa LSTM (Figura 4.44) y el resultado obtenido reduce un poco

la varianza, pero aumenta el bias, y si se observa el comportamiento en cuanto a las predicciones (Figura 4.45), se ve que se pierde un poco de lo que antes se buscaba que era obtener un modelo que se adaptara bien a los picos. Si se utiliza un valor más alto de dropout, el ruido es mayor y peor se comporta el modelo. El resultado no varía mucho, pero cabe decir que los valores ahora están un poco más cercanos a la media y no es lo que se busca, por lo que para predecir el consumo se va a descartar el uso de dropout.

```python
def build_model1():
    model = tf.keras.Sequential([
        tf.keras.layers.Input(shape=(n_entrada, n_features)),
        tf.keras.layers.LSTM(128, activation='tanh', return_sequences = True, dropout=0.1),
        tf.keras.layers.LSTM(128, activation='tanh'),
        tf.keras.layers.Dense(n_predicciones)
    ])

    optimizer = tf.keras.optimizers.Adam()

    model.compile(loss='mse', optimizer=optimizer, metrics=['mae', 'mse', smape])

    return model
```

Figura 4.44. Modelo Encoder Decoder con dropout a 0.1 en la primera capa.

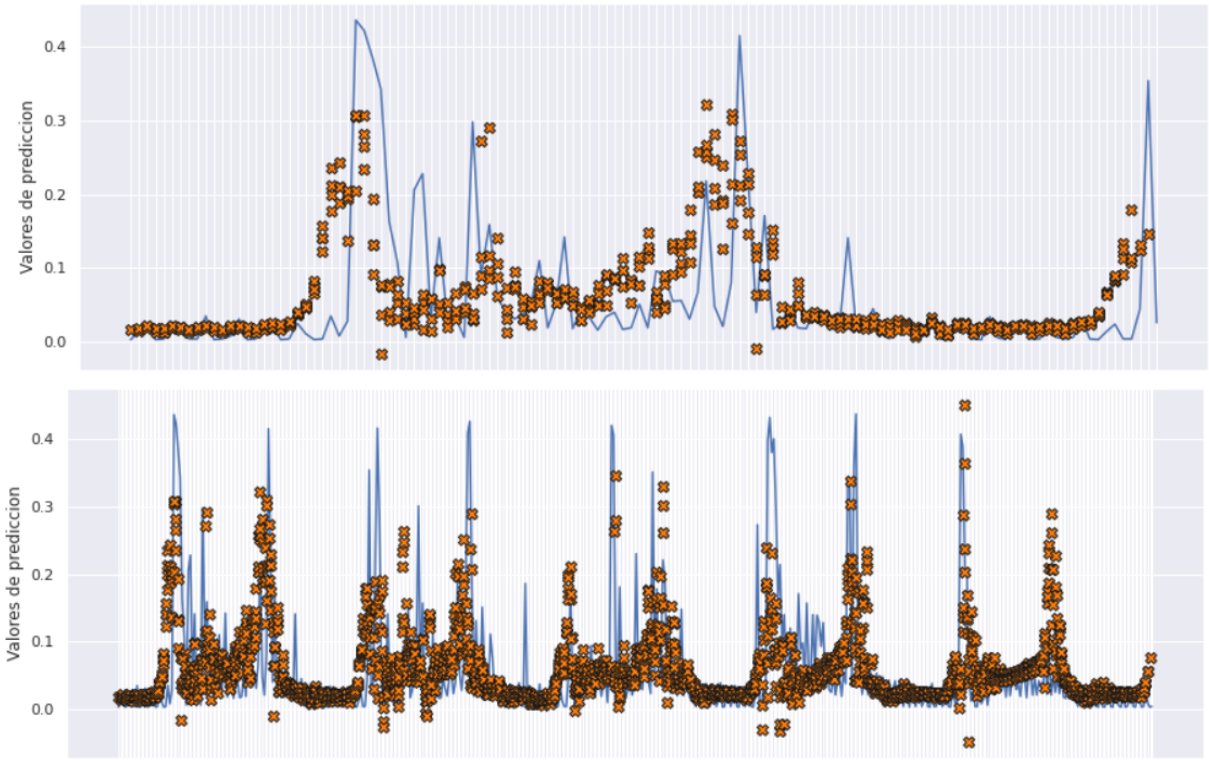


Figura 4.45. Muestra de aproximadamente 120 (arriba) y 500 (abajo) predicciones del modelo LSTM de 2 capas con dropout para la hora con el método de entrada de la ventana de varios pasos con la fecha y el valor.

En cuanto a la regularización L1L2, se puede aplicar de varias formas, en concreto aplicando regularización L1 sólo, L2 sólo o L1 y L2 a la vez. Además, en la capa de LSTM se puede aplicar regularización a la matriz de pesos del kernel, a la matriz de pesos del kernel

recurrente, al vector de bias o a la salida de la red neuronal (Figura 4.46). Se ha probado cada caso y tampoco mejora mucho al modelo normal, por que si se usa un número de regularización pequeño, no cambia mucho el comportamiento del modelo ni baja mucho la varianza (Figura 4.47), incluso en ocasiones funcionando mal, perdiendo el comportamiento de los picos acercando los valores a la media general (Figura 4.48). Si se usara un número más grande el ruido que se metería al modelo sería mayor y por tanto peor se comportaría.

```
def build_model2():
    model = tf.keras.Sequential([
        tf.keras.layers.Input(shape=(n_entrada, n_features)),
        tf.keras.layers.LSTM(128, activation='tanh', return_sequences = True,
                             kernel_regularizer=tf.keras.regularizers.L1L2(0.01, 0)),
        tf.keras.layers.LSTM(128, activation='tanh'),
        tf.keras.layers.Dense(n_predicciones)
    ])

    optimizer = tf.keras.optimizers.Adam()

    model.compile(loss='mse', optimizer=optimizer, metrics=['mae', 'mse', smape])

    return model
```

Figura 4.46. Modelo LSTM Encoder Decoder con regularización L1 a 0.01 en la matriz de pesos del kernel

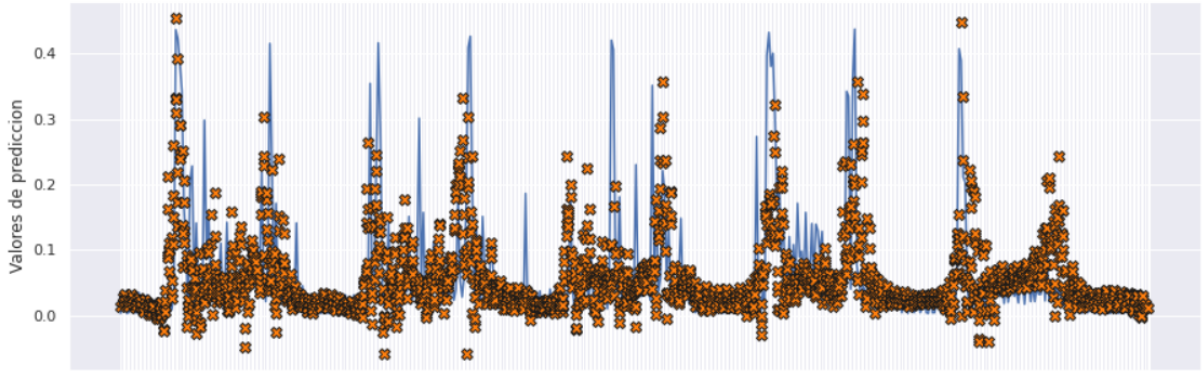


Figura 4.47. Muestra de 500 prediciones aproximadamente del Modelo LSTM de 2 capas con regularización L1 del kernel recurrente a 0.01.

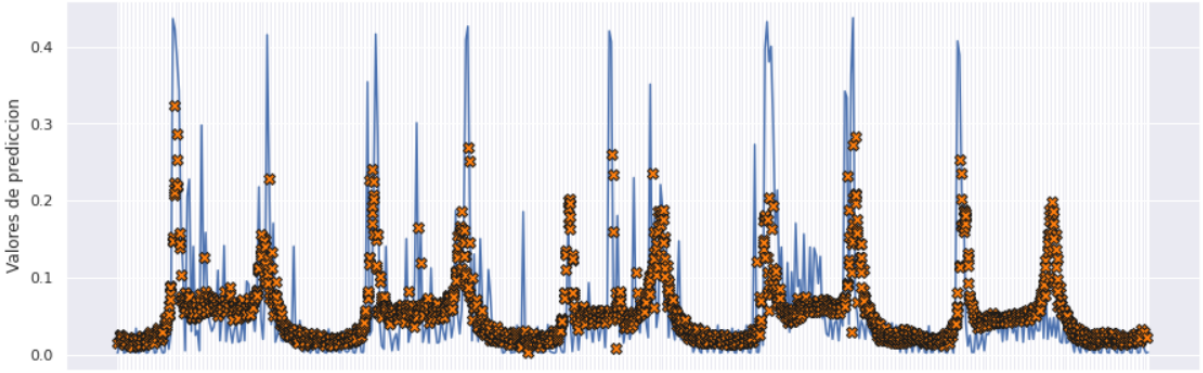


Figura 4.48. Muestra de 500 prediciones aproximadamente del Modelo LSTM de 2 capas con regularización L2 de 0.01 en la salida de la red neuronal.

Observando los resultados de las distintas formas de optimización, la conclusión es que, en este caso que hay muchos picos de consumo, no mejoran mucho el comportamiento del modelo normal por lo que no se entrará en mucha profundidad sobre estos tipos de optimización y se seguirá trabajando con el modelo normal. Esto sirve para una primera impresión del uso de dropout y regularización, quizás más adelante, con un poco más de conocimiento se use para mejorar un modelo definitivo con los datos del Smart Meter.

4.5. Creación de servicio para predicciones

Tensorflow Serving ofrece una forma fácil de hacer disponibles los modelos mediante una API REST para poder generar predicciones mediante peticiones HTTP. En primer lugar, se deben guardar los modelos en formato con la función *save()* que ofrece Tensorflow con una ruta similar a *'models/nombreModelo/1/'*, cuyo 1 final representa la versión que se guarda. En este caso la rutas serán de la forma '../../models/only_value/nombreModelo/1/', para aquellos modelos que sólo reciben por entrada los valores de la columna del consumo, y *'models/datetime_and_value/nombreModelo/1/'* para los que reciben los campos relacionados con la fecha y el valor, y así podrán diferenciarse los distintos métodos. Los nombres de los modelos corresponderán al tipo de predicción: *'hourModel'* para la predicción de la hora, *'dayModel'* para el día, *'weekModel'* para la semana y por último *'monthModel'* para el mes.

Una vez guardados los modelos, el siguiente paso es crear una imagen en Docker de Tensorflow Serving y subirla a la cuenta de Docker Hub, para ello se ejecutará el siguiente comando:

```
docker run -p 8500:8500 -p 8501:8501 --name NOMBRE_IMAGEN -t
tensorflow/serving
```

y por cada modelo se ejecuta el siguiente comando,

```
docker cp RUTA/NOMBRE_MODELO
NOMBRE_IMAGEN:/models/NOMBRE_MODELO
```

donde *NOMBRE_IMAGEN* será el nombre que se le quiera dar a la imagen, *RUTA* es el path donde se aloja el modelo, *NOMBRE_MODELO* será el nombre que se le ha asignado al modelo al guardarlo junto a la *VERSIÓN*. Además, se debe especificar un fichero '.config' que tenga la siguiente estructura:

```
model_config_list: {
    config: {
        name: NOMBRE_MODELO,
        base_path: "/models/NOMBRE_MODELO",
        model_platform: "tensorflow"
    },
}
```

Cabe destacar que se puede especificar más de un modelo simultáneamente tanto en el comando como en el fichero '.config', por si se quieren hacer disponibles más modelos en la imagen.

Por último se define otro fichero *'.sh' de la siguiente forma:*

```
#!/bin/bash
tensorflow_model_server --port=8500 --rest_api_port=8501 --
model_config_file=/models/models.config
```

Y por último se copia estos dos archivos en la imagen de Docker con los comandos:

```
docker cp RUTA\models.config NOMBRE_IMAGEN:/models/models.config
docker cp RUTA\tfserving_entrypoint.sh
NOMBRE_IMAGEN:/usr/bin/tfserving_entrypoint.sh
```

Con la imagen ya creada, lo siguiente por hacer es subir la imagen al repositorio de Docker Hub para poder acceder a ella desde Kubernetes ejecutando los siguientes comandos:

```
docker commit yerayrs/consumo_serving:v1
docker push yerayrs/consumo_serving:v1
```

Una vez disponibilizada la imagen en el repositorio, sólo queda desplegarla en Kubernetes mediante dos ficheros '.yaml', uno para el despliegue y otro para la creación del servicio para poder acceder al contenedor. El primer fichero se estructura de la siguiente forma:

```
apiVersion: apps/v1
kind: Deployment
metadata:
    name: consumoserving
spec:
    replicas: 1
    selector:
        matchLabels:
            app: consumoserving
    template:
        metadata:
            labels:
                app: consumoserving
        spec:
            containers:
            - name: consumoserving
                image: yerayrs/consumo_serving:v1
                ports:
                - containerPort: 8501
```

El otro fichero '.yaml' a realizar para desplegar el servicio tendría el siguiente contenido:

```
apiVersion: v1
kind: Service
metadata:
    name: consumoserving
spec:
    ports:
    - port: 8501
        nodePort: 30111
    selector:
        app: consumoserving
    type: NodePort
```

Con esto, el servicio ya estaría desplegado y accesible para acceder desde otro contenedor Docker como el backend a través de una API REST. El proceso detallado se verá en el siguiente paso.

4.6. Web de visualización para validación

El resultado final del trabajo es la realización de una aplicación web que haga peticiones para generar predicciones y las muestre por pantalla. En el proyecto K-Project del Smart Meter el objetivo sería que, a partir de las últimas mediciones, se generasen las predicciones de los valores siguientes. Sin embargo, para este trabajo se ha realizado una web de validación de los resultados comparando las predicciones con los valores reales disponibles en la base de datos que deberían salir como resultado. En el Apéndice B del manual de usuario se puede observar con más detalle las funcionalidades y la interfaz gráfica de la página web.

El primer paso es renderizar la página de inicio de la aplicación y que se muestre un formulario para solicitar los datos para generar las predicciones.

Luego, una vez generada la página de inicio, el usuario podrá elegir el tipo de entrada en la predicción, eligiendo entre una entrada de sólo valores o de fecha y valores, además, qué tipo de predicción quiere realizar, ya sea la siguiente hora, el siguiente día, o los siguientes

7 días o 30 días a partir de la fecha que se seleccione. Es importante saber que no se puede elegir cualquier fecha, ya que no hay datos disponibles de todas las fechas, y por ejemplo, si se quiere predecir la siguiente hora, se requiere que la entrada sea de las 3 horas anteriores, por lo que tampoco se podrá coger la primera fecha que hay en la base de datos porque no habría entrada para el modelo y daría un error. Esto está totalmente controlado mostrando un error e indicando que rango de fechas está disponible para seleccionar en cada tipo. En el momento que el funcionamiento sea con los datos reales en el marco del proyecto K-Project, esto no supondría ningún problema ya que se actuaría siempre con las últimas fechas registradas.

Una vez que se completen todos los campos del formulario correctamente, cuando se pulsa el botón predecir, con AJAX se solicita al backend que le devuelva las predicciones para esos campos. El backend obtiene los parámetros de la petición y construye una URL que corresponde al servicio de Tensorflow Serving que se ha construido en el apartado anterior. Para solicitar las predicciones, se debe hacer una petición POST, pasándole por el cuerpo un JSON del tipo *"{ 'instances': datos_entrada }"*, en el cuál *datos_entrada* representa los datos que van a entrar al modelo para generar las predicciones. Hay que tener en cuenta que dependiendo del método o del tipo de modelo, la entrada hay que pasarla de una forma u otra, ya sea dimensión, tamaño, etc. Además, como para entrenar los modelos se escala la entrada, hay que utilizar la misma escala, guardándola de los modelos que se han entrenado y luego importándola en el backend con la librería joblib de Python. En este caso, es distinta la escala cuando el tipo de entrada es de sólo valores o fecha y valores, por lo que hay que importar las dos. Además, la fecha, si se utiliza para la entrada, hay que transformarla de la misma forma en senos y cosenos.

Una vez se ha generado la predicción, el servicio devuelve una respuesta JSON con la forma *"{'predictions' : array_predicciones}",* donde *array_predicciones* hace referencia a la salida del modelo a partir de la entrada que se le ha pasado. El backend devuelve la respuesta a la petición AJAX mediante un JSON de la forma:

```
{ 'input_array': entrada escalada que se le pasa al modelo,
    'outputs_timestamp': timestamps de los datos objetivo,
    'outputs_values': valores de los datos objetivo,
    'outputs_sum' : suma total de los valores objetivo,
    'predictions': predicciones,
    'predictions_sum' : suma total de predicciones,
    'results': {
            'mae': mae (salida_values, predicciones),
            'mse': mse(salida_values, predicciones),
            'rmse': rmse(salida_values, predicciones)),
            'smape': smape(salida_values, predicciones)
        }
}
```

El frontend recibe la respuesta y la procesa generando con Chart.js un gráfico lineal comparando las predicciones con los valores objetivos, un gráfico de barras comparando el consumo total objetivo con el de la predicción y mostrando los valores de los errores que se han utilizado para los entrenamientos de los modelos (Figura 4.49). Además, si el usuario situa el ratón en cualquier punto de la gráfica, le mostrará el valor para ese punto exacto además de la fecha a la que le corresponde. Como se ha comentado, si se desea ver cómo han quedado la interfaz y las funcionalidades, puede verlas en el apéndice B, donde se detallan paso por paso.

Figura 4.49. Ejemplo de visualización de resultados en el servicio web

Una vez que la aplicación funciona correctamente, lo siguiente que hay que hacer es crear una imagen de Docker y subirla al repositorio Docker Hub. Para ello se crea un archivo Dockerfile de la siguiente forma:

FROM python:3.9

WORKDIR /usr/src/app

RUN pip install --upgrade pip

COPY ./requirements.txt /usr/src/app/requirements.txt

RUN pip install -r requirements.txt

COPY . /usr/src/app/

EXPOSE 5000

RUN chmod +x ./start.sh

CMD ["./start.sh"]

Y para subirlo al repositorio se ejecutan los siguientes comandos en la ruta del Dockerfile:

docker build -t NOMBRE_IMAGEN .

docker tag NOMBRE_IMAGEN yerayrs/consumo_backend:v1

docker push yerayrs/consumo_backend:v1

Una vez alojadada la imagen en la nube, el despliegue en Kubernetes es igual que el del apartado anterior, sólo que habría cambiar los puertos para que fueran distintos y no hubiera conflictos, el nombre por ejemplo, en vez de consumoserving se establecería backendserving y por último especificar la imagen de Docker Hub, en este caso yerayrs/consumo_backend:v1.

5

Conclusiones y líneas futuras

Como resultado de este trabajo se han conseguido varios modelos de predicción para predecir el consumo eléctrico de la siguiente hora, día, semana o mes. En cuanto a los resultados obtenidos, se puede deducir que es muy difícil predecir el consumo eléctrico de un hogar, ya que forma series temporales irregulares que varían mucho según el día y la hora y no tienen un comportamiento fijo a lo largo del tiempo al depender de muchos factores que puede deberse simplemente a que el hogar no tenga unos horarios establecidos para el uso de los electrodomésticos grandes, que son los que más elevan los picos de consumo, si no que un día un usuario puede, por ejemplo, utilizar la lavadora de noche y al otro día usarla por la mañana, cambiando radicalmente la regularidad de dicha serie temporal. Sin embargo, a pesar de esta dificultad, se han obtenido resultados aceptables para predecir la siguiente hora y el siguiente día, viendo en cada caso que las predicciones varíaban poco respecto a los valores reales. Sin embargo, los resultados obtenidos para la semana y el mes tienen mucho que mejorar.

La realización de este trabajo ha servido mucho para un primer acercamiento al paradigma de Machine Learning y para ver el funcionamiento interior de un proyecto del grupo Ertis de la Universidad de Málaga, y aunque los resultados que se han obtenido no

hayan sido los más deseados, estos tienen mucho margen de mejora. Además, hay que tener en cuenta que esto no ha sido una tarea sencilla, ya que empresas grandes dedicadas al sector eléctrico, a día de hoy, siguen investigando como encontrar un modelo que se adapte bien al consumo eléctrico.

En cuanto a las líneas futuras, el primer objetivo que se plantea será adaptar estos modelos a las mediciones de consumo eléctrico proporcionadas por el Smart Meter del K-Project, y ver cómo se comportan respecto a los nuevos datos. Además, una vez que estén los modelos construidos, si los resultados para predecir la semana y el mes siguen siendo malos, podría optarse por usar una ventana más pequeña agrupando los datos o incluso usando otros tipos de modelos que mejoren el comportamiento. Una vez que se obtengan unos buenos resultados de entrenamiento, el siguiente objetivo será acoplar esta parte de predicción del consumo eléctrico a la web del K Project del Smart Meter, para poder mostrarle esta información al usuario.

Referencias

[1] Parlamento Europeo, “¿Qué es la inteligencia artificial y cómo se usa? | Noticias | Parlamento Europeo,” 2020. https://www.europarl.europa.eu/news/es/headlines/society/20200827STO85804/que-es-la-inteligencia-artificial-y-como-se-usa (accessed Jun. 22, 2022).

[2] IBM Cloud Computing, “¿Qué es machine learning? - España | IBM,” 2020. https://www.ibm.com/es-es/cloud/learn/machine-learning (accessed Jun. 22, 2022).

[3] Tarifaluzhora by Selectra, “¿Cuántos kWh consume una casa? Calcula tu consumo eléctrico.” https://tarifaluzhora.es/info/calcular-consumo-electrico-casa (accessed Jun. 22, 2022).

[4] Tarifaluzhora by Selectra, “Nueva Tarifa de la Luz 2022 ▷ ¿Cómo va a afectar? | Cambios.” https://preciogas.com/faq/nueva-tarifa-luz (accessed Jun. 24, 2022).

[5] Tarifaluzhora by Selectra, “Periodificación de energía eléctrica.” https://preciogas.com/sites/preciogas.com/files/periodificacion-energia-750.png (accessed Jun. 24, 2022).

[6] Grupo Ertis UMA, “Contadores inteligentes y accesibles para el hogar,” 2022. http://ertis.uma.es/k-project/ (accessed Jun. 24, 2022).

[7] Santander Universidades, “¿Qué es Python? | Blog Becas Santander,” 2021. https://www.becas-santander.com/es/blog/python-que-es.html (accessed Jun. 24, 2022).

[8] 1000MARCAS, “Logo de Python.” https://1000marcas.net/wp-content/uploads/2020/11/Python-logo.png (accessed Jun. 24, 2022).

[9] AprendeIA, “¿Qué es TensorFlow? ¿Cómo funciona? - Aprende IA.” https://aprendeia.com/que-es-tensorflow-como-funciona/ (accessed Jun. 24, 2022).

[10] “Logo de Tensorflow.” https://camo.githubusercontent.com/aeb4f612bd9b40d81c62fcbebd6db44a5d4344b8b962be0138817e18c9c06963/68747470733a2f2f7777772e74656e736f72666c6f772e6f72672f696d616765732f74665f6c6f676f5f686f72697a6f6e74616c2e706e67 (accessed Jun. 24, 2022).

[11] Keras, “About Keras.” https://keras.io/about/ (accessed Jun. 24, 2022).

[12] Keras, “Logo de Keras.” https://keras.io/img/logo.png (accessed Jun. 24, 2022).

[13] Jupyter, “Project Jupyter | Home.” https://jupyter.org/ (accessed Jun. 24, 2022).

[14] Wikimedia Commons, “Logo de Jupyter.” https://upload.wikimedia.org/wikipedia/commons/thumb/3/38/Jupyter_logo.svg/1200px-Jupyter_logo.svg.png (accessed Jun. 24, 2022).

[15] Alfredo Sánchez Alberca, “La librería Pandas | Aprende con Alf.” https://aprendeconalf.es/docencia/python/manual/pandas/ (accessed Jun. 24, 2022).

[16] Wikimedia Commons, “Logo de Pandas.” https://upload.wikimedia.org/wikipedia/commons/thumb/e/ed/Pandas_logo.svg/512px-Pandas_logo.svg.png (accessed Jun. 24, 2022).
[17] Alfredo Sánchez Alberca, “La librería Numpy | Aprende con Alf.” https://aprendeconalf.es/docencia/python/manual/numpy/ (accessed Jun. 24, 2022).
[18] Alfredo Sánchez Alberca, “Logo de Numpy.” https://aprendeconalf.es/docencia/python/manual/img/numpy-logo.png (accessed Jun. 24, 2022).
[19] Ander Fernández Jauregui, “Tutorial Sklearn Python - Ander Fernández.” https://anderfernandez.com/blog/tutorial-sklearn-machine-learning-python/ (accessed Jun. 24, 2022).
[20] Wikimedia Commons, “Logo de Scikit Learn.” https://upload.wikimedia.org/wikipedia/commons/thumb/0/05/Scikit_learn_logo_small.svg/1280px-Scikit_learn_logo_small.svg.png (accessed Jun. 24, 2022).
[21] Alfredo Sánchez Alberca, “La librería Matplotlib | Aprende con Alf.” https://aprendeconalf.es/docencia/python/manual/matplotlib/ (accessed Jun. 24, 2022).
[22] Matplotlib, “Logo de Matplotlib.” https://matplotlib.org/stable/_images/sphx_glr_logos2_003.png (accessed Jun. 24, 2022).
[23] Seaborn, “seaborn: statistical data visualization — seaborn 0.11.2 documentation.” https://seaborn.pydata.org/ (accessed Jun. 24, 2022).
[24] Seaborn, “Logo de Seaborn.” https://seaborn.pydata.org/_static/logo-wide-lightbg.svg (accessed Jun. 24, 2022).
[25] Wikipedia, “GitHub - Wikipedia, la enciclopedia libre.” https://es.wikipedia.org/wiki/GitHub (accessed Jun. 24, 2022).
[26] Yeray Ruiz Suárez, “Smart-Meter-UMA/Machine_Learning_Consumption: Electricity consumption forecasting.” https://github.com/Smart-Meter-UMA/Machine_Learning_Consumption (accessed Jun. 24, 2022).
[27] LogosWorld, “Logo de GitHub.” https://logos-world.net/wp-content/uploads/2020/11/GitHub-Logo.png (accessed Jun. 24, 2022).
[28] Wikipedia, “Docker (software) - Wikipedia, la enciclopedia libre.” https://es.wikipedia.org/wiki/Docker_(software) (accessed Jun. 24, 2022).
[29] Docker, “Logo de Docker.” https://www.docker.com/wp-content/uploads/2022/03/vertical-logo-monochromatic.png (accessed Jun. 24, 2022).
[30] Kubernetes, “¿Qué es Kubernetes? | Kubernetes.” https://kubernetes.io/es/docs/concepts/overview/what-is-kubernetes/ (accessed Jun. 24, 2022).
[31] Wikimedia Commons, “Logo de Kubernetes.” https://upload.wikimedia.org/wikipedia/commons/thumb/6/67/Kubernetes_logo.svg/2560px-Kubernetes_logo.svg.png (accessed Jun. 24, 2022).
[32] Microsoft, “Visual Studio: IDE y Editor de código para desarrolladores de software y Teams.” https://visualstudio.microsoft.com/es/ (accessed Jun. 24, 2022).
[33] Microsoft, “Logo de Visual Studio Code.” https://visualstudio.microsoft.com/wp-content/uploads/2019/09/vs-code-responsive-01-1.png (accessed Jun. 24, 2022).
[34] Wikipedia, “Flask - Wikipedia, la enciclopedia libre.” https://es.wikipedia.org/wiki/Flask (accessed Jun. 24, 2022).

[35] Jinja, “Jinja — Jinja Documentation (3.1.x).” https://jinja.palletsprojects.com/en/3.1.x/ (accessed Jun. 24, 2022).
[36] Flask, “Logo de Flask.” https://flask.palletsprojects.com/en/2.1.x/_images/flask-logo.png (accessed Jun. 24, 2022).
[37] Jinja, “Logo de Jinja.” https://jinja.palletsprojects.com/en/3.1.x/_images/jinja-logo.png (accessed Jun. 24, 2022).
[38] MDN Web Docs, “HTML: Lenguaje de etiquetas de hipertexto | MDN.” https://developer.mozilla.org/es/docs/Web/HTML (accessed Jun. 24, 2022).
[39] MDN Web Docs, “CSS | MDN.” https://developer.mozilla.org/es/docs/Web/CSS (accessed Jun. 24, 2022).
[40] MDN Web Docs, “JavaScript | MDN.” https://developer.mozilla.org/es/docs/Web/JavaScript (accessed Jun. 24, 2022).
[41] CursosGIS, “Logo de HTML, CSS3 y JavaScript.” https://www.cursosgis.com/wp-content/uploads/2017/06/lenguajes_1.png (accessed Jun. 24, 2022).
[42] Wikipedia, “Bootstrap (framework) - Wikipedia, la enciclopedia libre.” https://es.wikipedia.org/wiki/Bootstrap_(framework) (accessed Jun. 24, 2022).
[43] Wikimedia Commons, “Logo de Bootstrap.” https://upload.wikimedia.org/wikipedia/commons/thumb/b/b2/Bootstrap_logo.svg/375px-Bootstrap_logo.svg.png (accessed Jun. 24, 2022).
[44] MDN Web Docs, “AJAX - Guía de Desarrollo Web | MDN.” https://developer.mozilla.org/es/docs/Web/Guide/AJAX (accessed Jun. 24, 2022).
[45] Wikimedia Commons, “Logo de AJAX.” https://upload.wikimedia.org/wikipedia/commons/thumb/a/a1/AJAX_logo_by_gengns.svg/398px-AJAX_logo_by_gengns.svg.png (accessed Jun. 24, 2022).
[46] Wikipedia, “Chart.js - Wikipedia, la enciclopedia libre.” https://es.wikipedia.org/wiki/Chart.js (accessed Jun. 24, 2022).
[47] Chart JS, “Logo de Chart JS.” https://www.chartjs.org/img/chartjs-logo.svg (accessed Jun. 24, 2022).
[48] Wikimedia Commons, “Logo de MagicDraw.” https://images.g2crowd.com/uploads/product/image/social_landscape/social_landscape_46aad15b478f27aee5de195e2079aebd/magicdraw.png (accessed Jun. 24, 2022).
[49] Wikimedia Commons, “Logo de MongoDB.” https://upload.wikimedia.org/wikipedia/commons/thumb/9/93/MongoDB_Logo.svg/2560px-MongoDB_Logo.svg.png (accessed Jun. 24, 2022).
[50] Wikimedia Commons, “Logo de IFML.” https://upload.wikimedia.org/wikipedia/commons/3/3c/IFML-Interaction-Flow-Modeling-Language.png (accessed Jun. 24, 2022).
[51] IBM, “IBM Machine Learning.” https://www.ibm.com/es-es/analytics/machine-learning (accessed Jun. 24, 2022).
[52] Atria Innovation, “Esquema de redes neuronales.” https://www.atriainnovation.com/wp-content/uploads/2019/10/Redes_neuronales_esquema.png (accessed Jun. 25, 2022).
[53] Analytics Lane, “Implementación del método descenso del gradiente en Python - Analytics Lane.” https://www.analyticslane.com/2018/12/21/implementacion-del-metodo-descenso-del-gradiente-en-python/ (accessed Jun. 24, 2022).

[54] Numerentur, “Gráfico del descenso del gradiente.” https://numerentur.org/wp-content/uploads/2018/08/gradiente-descendente-grafico60a.png (accessed Jun. 24, 2022).
[55] EpicalSoft, “Underfitting y overfitting.” https://app-epicalsoftsite-prod.azurewebsites.net/wp-content/uploads/2019/02/underfitting-overfitting.png (accessed Jun. 24, 2022).
[56] Chitta Ranjan, *Understanding Deep Learning. Application in Rare Event Prediction.* 2020.
[57] Andrew NG, *Machine Learning Yearning*, Draft Version. 2018.
[58] François Chollet, *Deep Learning with Python*. Shelter Island, 2018.
[59] Tensorflow, “TensorFlow Core.” https://www.tensorflow.org/tutorials?hl=es-419 (accessed Jun. 25, 2022).
[60] Tensorflow, “TensorFlow Core.” https://www.tensorflow.org/guide?hl=es-419 (accessed Jun. 25, 2022).
[61] Udemy, “Complete Tensorflow 2 and Keras Deep Learning Bootcamp | Udemy.” https://www.udemy.com/course/complete-tensorflow-2-and-keras-deep-learning-bootcamp/ (accessed Apr. 25, 2022).
[62] J. Chavat, S. Nesmachnow, J. Graneri, and G. Alvez, “ECD-UY, detailed household electricity consumption dataset of Uruguay,” *Scientific Data*, vol. 9, no. 1, Dec. 2022, doi: 10.1038/S41597-022-01122-X.
[63] Figshare, “ECD-UY: Detailed household electricity consumption dataset of Uruguay.” https://figshare.com/collections/ECD-UY_Detailed_household_electricity_consumption_dataset_of_Uruguay/5428608 (accessed Jun. 24, 2022).

Apéndice A

Manual de instalación

Este manual describirá paso a paso los pasos necesarios para poner la aplicación en marcha de una forma clara y sencilla.

El proceso de instalación de la aplicación es muy sencillo, sólo requiere tener instalado Docker con Kubernetes (se recomienda el uso de Docker Desktop y activar en los ajustes Kubernetes), para poder desplegar la aplicación en local. El despliegue y la creación del servicio se hará sólo con cuatro comandos sobre los cuatro ficheros '.yaml' que se preparan en la entrega del código para el backend y Tensorflow Serving. Para el backend, corresponderán los ficheros con el nombre *'backend_deploy.yaml'* y *'backend_service.yaml'*, y para el servicio de los modelos los nombres serán *'tfserving_deploy.yaml'* y *'tfserving_service.yaml'*.

El orden de ejecución de los comandos no es relevante, da igual si se despliega primero el backend o el servingpor que antes de crear el servicio hay que desplegar el contenedor, sino lanzará un error. El orden recomendado de ejecución y los comandos a aplicar son los siguientes:

kubectl apply -f tfserving_deploy.yaml

kubectl apply -f tfserving_service.yaml

kubectl apply -f backend_deploy.yaml

kubectl apply -f backend_service.yaml

Una vez ejecutados dichos comandos, con el comando *'kubect get all'* se podrán ver los recursos que están en ejecución (Figura A.1). Como se puede comprobar, los contenedores se han desplegado correctamente en Kubernetes.

```
D:\yeray\Documents\TFG\tfserving>kubectl get all
NAME                                    READY   STATUS    RESTARTS   AGE
pod/backendconsumo-66fbf9b786-69spv     1/1     Running   0          2m58s
pod/consumoserving-c7ccf8cd9-qmnmz      1/1     Running   0          13s

NAME                      TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
service/backendconsumo    LoadBalancer   10.97.174.14    localhost     5000:32185/TCP   2m47s
service/consumoserving    NodePort       10.108.110.12   <none>        8501:30111/TCP   8s
service/kubernetes        ClusterIP      10.96.0.1       <none>        443/TCP          14m

NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/backendconsumo      1/1     1            1           2m58s
deployment.apps/consumoserving      1/1     1            1           13s

NAME                                           DESIRED   CURRENT   READY   AGE
replicaset.apps/backendconsumo-66fbf9b786      1         1         1       2m58s
replicaset.apps/consumoserving-c7ccf8cd9       1         1         1       13s
```

Figura A.1. Comprobación de funcionamiento del despliegue de los servicios.

Si se quiere abrir la aplicación web, sólo tendrá que situarse en el navegador web (se recomienda el uso de Google Chrome), y ir a la ruta 'localhost:5000" y podrá ver la aplicación en funcionamiento (Figura A.2).

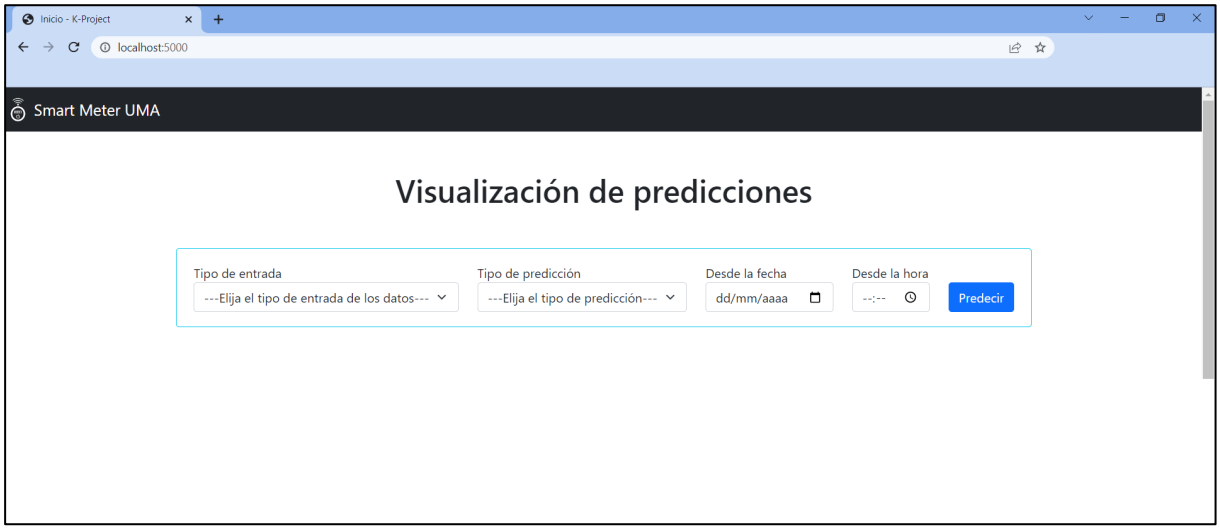


Figura A.2. Inicio de la aplicación desplegado en Kubernetes

Apéndice B

Manual de usuario

El manual de usuario servirá como guía para que los usuarios que deseen utilizar la aplicación web puedan realizar predicciones y comprobar su funcionamiento. Este manual explicará cada detalle paso a paso con claridad para su total entendimiento.

Una vez que el servicio está instalado, el primer paso es abrir la aplicación en un navegador web mediante la ruta 'localhost:5000".

Lo primero que aparecerá en la pantalla será la página de inicio (Figura B.1).

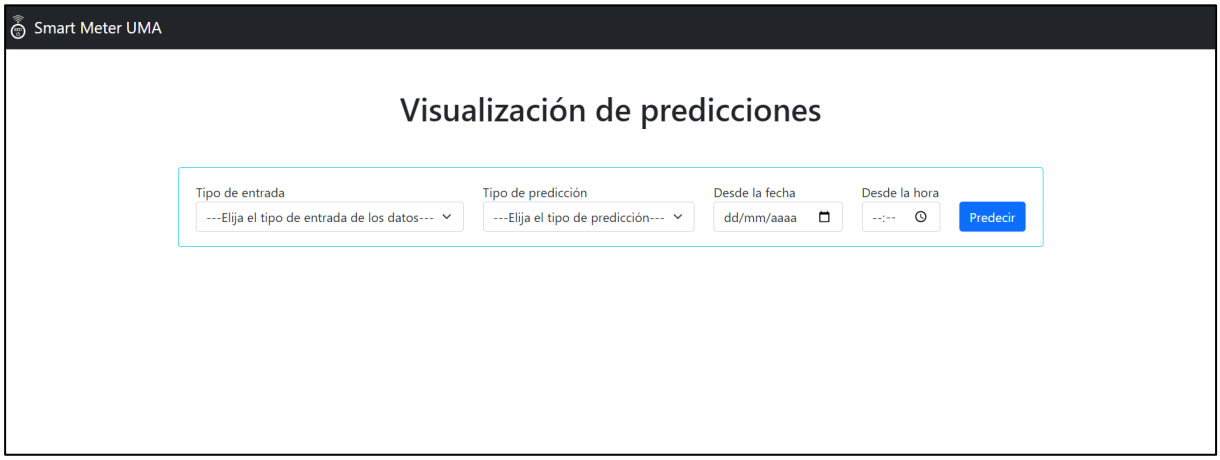


Figura B.1. Pantalla de inicio de la aplicación web.

En esta página se puede observar un formulario para escoger que tipo de entrada se usará para la predicción (Figura B.2), si sólo los valores o los campos de la fecha y los valores, y además se podrá elegir entre varios tipos de predicción (Figura B.3). Finalmente, se escogerá desde que fecha se quiere hacer la predicción, por ejemplo, si se elige la fecha 16/12/2019 a las 00h:00m, el primer valor de predicción será el correspondiente a esa fecha y a la hora 00:15, ya que las mediciones tienen una periodicidad de 15 minutos.

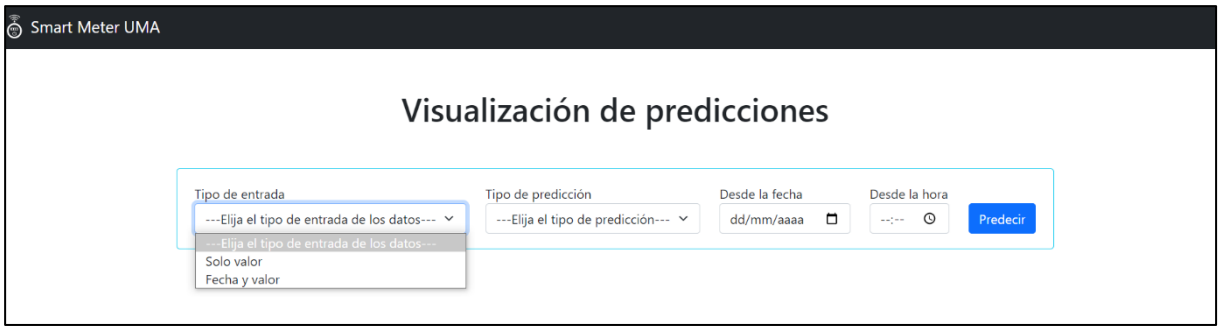


Figura B.2. Selección de tipos de entrada.

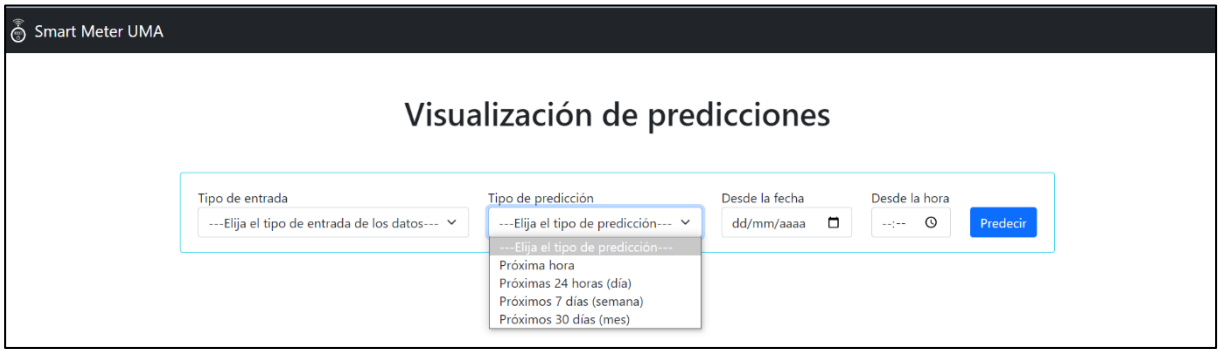


Figura B.3. Selección de tipos de predicción.

Además, todo tipo de error que pueda tener el usuario estará controlado. Por ejemplo, si el usuario se deja campos sin rellenar, la interfaz web le avisará que deberá completar los campos (Figura B.4). Otro error puede deberse a que para la fecha seleccionada, no existan suficientes datos de entrada o datos reales para usar en la comparación, por lo que el servicio avisará con una alerta al usuario indicando que rango de valores podrá tomar para ese tipo de predicción (Figura B.5). Si el usuario intenta generar las predicciones obviando la alerta anterior, se mostrará un error avisándole de ello para que la cambie (Figura B.6).

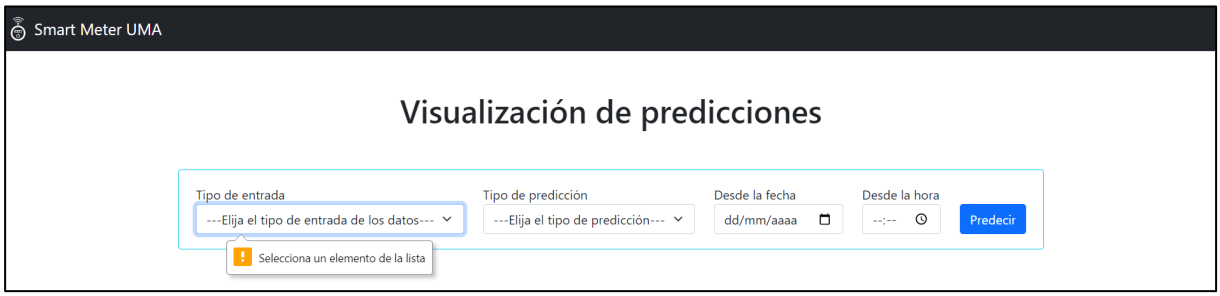


Figura B.4. Error de campos incompletos.

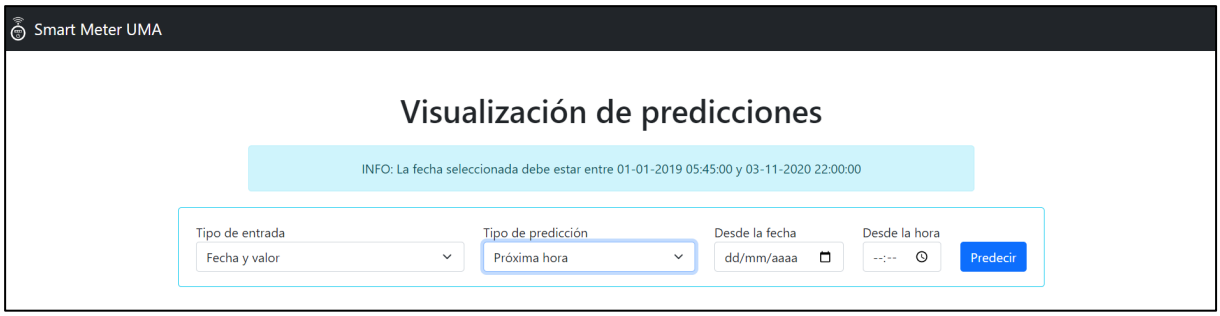


Figura B.5. Información de rango de fechas.

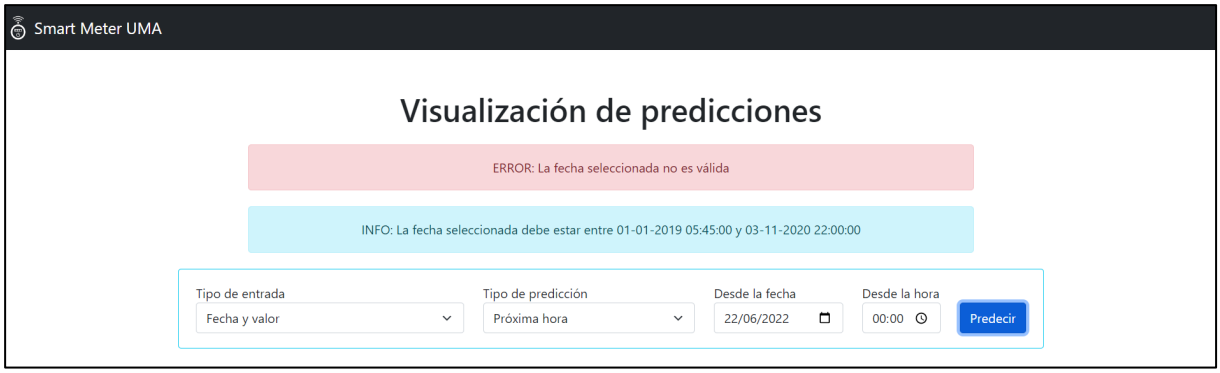


Figura B.6. Error para fechas fuera del rángo.

Si el usuario rellena todos los campos e introduce una fecha correcta, el sistema generará una gráfica lineal de comparación de los valores reales frente a las predicciones y además mostrará una gráfica de comparación del consumo total y los valores obtenidos por el cálculo del error del MSE, MAE, RMSE y SMAPE (Figura B.7). Otra de las funcionalidades es que, si el usuario desea ver los valores exactos en cada punto de la gráfica líneal, sólo tiene que colocar el ratón encima del punto y le aparecerá la información relacionada a dicho punto (Figura B.8). Igualmente, si desea ver el valor exacto en el gráfico de barras sólo tendrá que colocar el ratón encima de la barra cuyo valor desea obtener (Figura B.9).

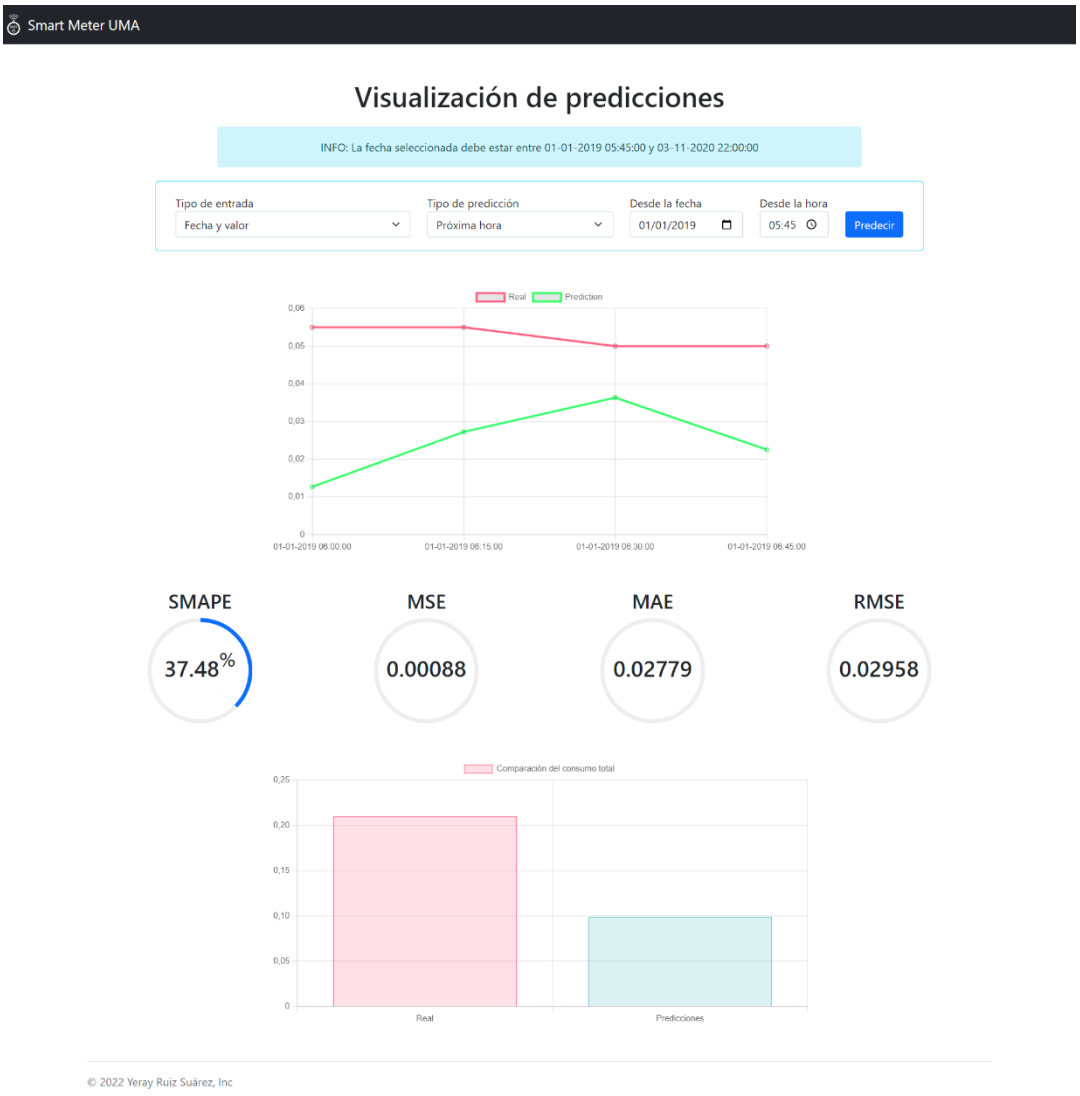


Figura B.7. Visualización de los resultados de las predicciones.

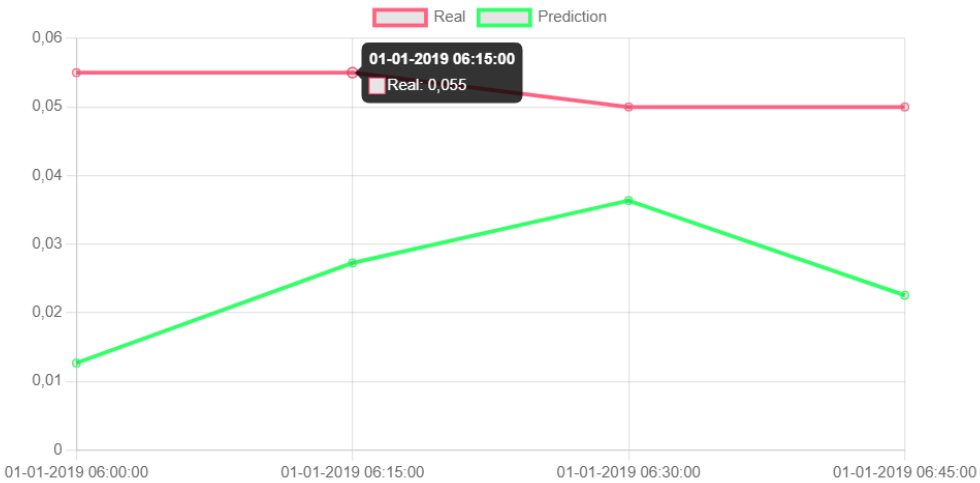


Figura B.8. Visualización de valores exactos de un punto de la gráfica lineal.

Figura B.9. Visualización del valor exacto del consumo total en el gráfico de barras

.

UNIVERSIDAD DE MÁLAGA | uma.es

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga

E.T.S. DE INGENIERÍA INFORMÁTICA