Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Predicción de series temporales mediante técnicas de aprendizaje profundo

Time series forecasting using deep learning techniques

José Ramón Morera Campos

La Laguna, 20 de mayo de 2025

D. **Leopoldo Acosta Sánchez**, profesor Catedrático de Universidad adscrito al Departamento de Nombre del Departamento de la Universidad de La Laguna, como tutor

D. **Daniel Acosta Hernández**, profesor Titular de Universidad adscrito al Departamento de Nombre del Departamento de la Universidad de La Laguna, como cotutor

C E R T I F I C A N

Que la presente memoria titulada:

”*Predicción de series temporales mediante técnicas de aprendizaje profundo*”

ha sido realizada bajo su dirección por D. **José Ramón Morera Campos**.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 20 de mayo de 2025

Agradecimientos

xxxxx

Licencia

Resumen

xxxxx

Palabras clave: xxxxx, xxxx, xxxx

Abstract

xxxxx

Keywords: xxxx, xxxx, xxxx

Índice general

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

1.1. Motivación del proyecto

La predicción de series temporales es un campo de estudio que ha cobrado gran relevancia en los últimos años, especialmente en el ámbito del aprendizaje automático y el aprendizaje profundo. La capacidad de anticipar eventos futuros a partir de datos históricos es fundamental en diversas áreas, como la economía, la meteorología, la salud y la ingeniería.

El campo de la meteorología, en particular, ha aumentado su relevancia en los útlimos años debido al aumento de fenómenos climáticos extremos y su impacto en la sociedad. Prever con antelación la evolución de variables meteorológicas no solo permite planificar recursos, sino también contribuir a la prevención de desastres naturales.

En este contexto, el uso de redes neuronales ha demostrado ser una herramienta poderosa para abordar problemas complejos de predicción. Gracias a los avances en capacidad de cómputo y la disponibilidad de grandes volúmenes de datos, arquitecturas como LSTM, GRU y Transformers se han consolidado como soluciones de alto rendimiento. Estas redes no solo capturan patrones temporales de manera eficiente, sino que también pueden adaptarse dinámicamente a cambios en el comportamiento de la serie, mejorando la precisión y la robustez de las predicciones.

Dentro de este ámbito, se ha estudiado extensamente el desarrollo de modelos ajustados para una o varias ubicaciones concretas. Sin embargo, la creación de un modelo generalista, que pueda adaptarse a diferentes ubicaciones y condiciones climáticas, es un tema de investigación poco explorado e interesante. Con este tipo de modelos se puede conseguir, a partir de un sensor básico y sin necesidad de reentrenar, generar predicciones meteorológicas para una ubicación cualquiera, que generalmente serán de calidad mayor a las de los modelos numéricos de predicción climática usados por los servicios meteorológcios debido a las limitaciones de resolución espacial de estos. De esta forma se obtiene una gran flexibilidad y adaptabilidad.

1.2. Planteamineto

En este trabajo se pretende emplear mediciones de múltiples estaciones meteorológicas de la isla de Tenerife con el fin de desarrollar un modelo de predicción climatológica a corto plazo. Se busca que dicho modelo sea capaz de generalizar más allá de las estaciones de entrenamiento. Esto es, que a partir de mediciones meteorológicas de cualquier origen, el modelo sea capaz de generar predicciones a corto plazo (3, 6 o 12

horas) de gran calidad. Se deciden considerar 3 variables meteorológicas: temperatura, humedad y presión atmosférica. La elección de estas variables se basa en su relevancia para la predicción del clima y la existencia de registros en las estaciones meteorológicas de Tenerife.

Se establece un especial énfasis en el tratamiento de los datos, estudiándose exhaustivamente diversas técnicas.

La idea principal pasa por la creación de ventanas de información, que son paquetes de datos con valores de las variables en los instantes anteriores a la predicción.

Como parte del trabajo se contempla el despliegue de una infraestructura tecnológica que permita la captura y el procesamiento de datos públicos, así como la creación y evaluación de diversos modelos predictivos que extraigan información valiosa a partir de estos datos.

Para ello, se emplará el lenguaje de programación Python, que ofrece una amplia gama de bibliotecas y herramientas para el análisis de datos y la implementación de modelos de aprendizaje automático.

Así mismo, se plantea desplegar los modelos en un entorno de producción, permitiendo su uso en aplicaciones prácticas. De esta forma, cualquier usuario podrá suministrar mediciones de la variable que desee durante las últimas horas y obtener un pronóstico.

1.3. Antecedentes y estado del arte

El uso de redes neuronales y técnicas de deep learning (aprendizaje profundo) para la predicción de series temporales tiene sus orígenes en la década de 1940. Aunque el término deep learning ha ganado relevancia en los últimos años, los conceptos fundamentales y algunas de las técnicas más empleadas han existido desde hace mucho tiempo, evolucionando y perfeccionándose con el avance de la tecnología. En 1943, Warren McCulloch y Walter Pitts fueron pioneros en desarrollar un modelo computacional para redes neuronales [1], estableciendo las bases de lo que hoy conocemos como inteligencia artificial basada en redes neuronales. Más adelante, en 1949, Donald Hebb introdujo la famosa Teoría Hebbiana, una hipótesis que proponía un mecanismo de aprendizaje basado en la plasticidad neuronal [2]. Este principio se aplicó a modelos computacionales, impulsando la investigación en la simulación del aprendizaje humano.

En 1958, Frank Rosenblatt avanzó en este campo con la creación del perceptrón [3], un algoritmo pionero de reconocimiento de patrones basado en una red de aprendizaje de computadora de dos capas. Este modelo simple operaba mediante operaciones de adición y sustracción, sentando las bases para el desarrollo de redes neuronales más complejas. Durante las siguientes décadas, la investigación en redes neuronales experimentó un estancamiento debido a limitaciones técnicas y a la falta de métodos eficaces para el entrenamiento de redes con múltiples capas. Sin embargo, en 1982, un gran avance revitalizó este campo: la introducción del algoritmo de propagación hacia atrás (backpropagation) [4], que resolvía el problema del entrenamiento eficiente de redes neuronales profundas, permitiendo la formación de redes multicapa de manera más rápida y eficaz.

La aplicación de redes neuronales profundas a la predicción de series temporales experimentó un impulso decisivo con la introducción de las redes recurrentes (RNN) a principios de los años ochenta. En 1987, Jeffrey Elman describió un modelo de red recurrente simple capaz de aprender dependencias temporales mediante la alimentación de su propia salida en la capa oculta al instante siguiente[5]. Sin embargo, estas primeras

RNN adolecían del problema del desvanecimiento y explosión del gradiente, lo que limitaba su eficacia para secuencias largas.

Para superar estas limitaciones, Hochreiter y Schmidhuber propusieron en 1997 las Long Short-Term Memory (LSTM), una arquitectura RNN que incorpora puertas de entrada, olvido y salida, permitiendo el flujo de gradiente a través de largas secuencias y mejorando sustancialmente la capacidad de memoria de la red [6]. Posteriormente, Cho et al. introdujeron en 2014 las Gated Recurrent Units (GRU), una simplificación de las LSTM con menor número de parámetros, que ha demostrado en muchos casos un rendimiento comparable acelerando el entrenamiento[7]. Más recientemente, la arquitectura Transformer, basada completamente en mecanismos de atención y sin conexiones recurrentes, revolucionó el campo al mejorar la paralelización y capturar dependencias de largo alcance en el tiempo con mayor eficacia[8].

A partir de 2015 surgieron estudios que adaptaron estas arquitecturas al ámbito meteorológico. Shi et al. desarrollaron el ConvLSTM, que añade convoluciones espaciales a la LSTM para modelar simultáneamente la dinámica espacial y temporal en tareas de nowcasting de precipitación[9]. Weyn et al. demostraron en 2019 que redes profundas convolucionales aplicadas a datos globales de variables meteorológicas pueden mejorar la predicción a corto plazo frente a los modelos numéricos tradicionales, especialmente al resolver patrones locales que escapan a la resolución de la rejilla de los sistemas de predicción numérica del tiempo (NWP)[10].

La mayoría de los enfoques anteriores se especializan en una región o estación concreta, entrenando modelos independientes para cada ubicación. Esto requiere un esfuerzo de reentrenamiento y calibración local cada vez que se desea aplicar el sistema en un punto nuevo. Algunos trabajos recientes abordan la generalización geográfica mediante la inclusión de información posicional o condiciones iniciales específicas de cada punto dentro del modelo, aunque todavía con resultados preliminares[11]. El desarrollo de un modelo verdaderamente “generalista” —capaz de recibir como entrada únicamente datos de sensores básicos, sin necesidad de reentrenar, y proporcionar predicciones meteorológicas de calidad comparable o superior a las de los NWP en cualquier ubicación— se conoce como el problema de zero-shot [12], y constituye un reto abierto y de gran interés práctico.

1.4. Objetivos

1. Evaluar y seleccionar una fuente de datos.
2. Evaluar y seleccionar el framework de aprendizaje profundo a usar en el proyecto.
3. Diseño e implementación de la metodología de procesamiento de los datos.
4. Evaluación y selección de arquitecturas pertinentes al problema.
5. Implementación de una arquitectura adecuada.
6. Evaluación del rendimiento de la solución propuesta.
7. Comparación frente a metodologías tradicionales.
8. Despliegue de la solución en un entorno de producción.

Capítulo 2

Adquisición y preprocesado de datos

Se desea trabajar con series temporales sobre mediciones climatológicas. En concreto, se eligen las variables de temperatura del aire, humedad relativa y presión atmosférica en la superficie. Dichas variables son estudiadas con frecuencia horaria, en el intervalo comprendido entre el 1 de marzo de 2023 y el 28 de febrero de 2025.

Es imprescindible disponer de un conjunto de datos que abarque un período de tiempo suficientemente amplio para poder cubrir las diversas condiciones climáticas que pueden presentarse. Además, es relevante el uso de datos en un período múltiplo del año, para asegurar que se capturan las variaciones estacionales y que el conjunto de datos está suficientemente equilibrado. Es decir, si se emplearan un año y 3 meses, se podría sesgar el conjunto introduciendo más días con temperaturas bajas, por ejemplo, si se cubrieran 2 inviernos pero un solo verano.

2.1. Fuentes de los datos

Se han empleado 2 fuentes para recopilar las mediciones:

- **Grafcan**: Cartográfica de Canarias, S.A. es una empresa pública de la Comunidad Autónoma de Canarias. Dispone de una red de estaciones meteorológicas cuyas mediciones son accesibles mediante una API REST de acceso gratuito previa solicitud de una clave[13].

- **Open-Meteo**: API pública de código abierto que proporciona datos de múltiples proveedores de meteorología. Este servicio no dispone de estaciones de medición propias, sino que recopila pronósticos de diferentes modelos de predicción climatológica.

  Se emplea la API de predicciones pasadas[14]. Se seleccionan los modelos ICON Global del servicio meteorológico alemán (DWD) y el modelo ARPEGE Europe de Météo-France. Ambos modelos se actualizan cada 3 horas. Se explora la posibilidad de emplear las predicciones del modelo HAROME de la AEMET, pero no están disponibles de forma pública

Se eligen 4 ubicaciones de la isla de Tenerife con distintas características climáticas para el conjunto de entrenamiento y evaluación:

- **San Cristóbal de La Laguna 1 (A)**: La Cuesta, 35 metros de altitud.

- **San Cristóbal de La Laguna 2 (B)**: La Punta del Hidalgo, 54m.

- **La Orotava (C)**: Camino de Chasna, 812m.
- **Arona (D)**: Punta de Rasca, 25m.

Así mismo, se escogen 2 ubicaciones para el conjunto de test, nunca vistas en el ajuste del modelo:

- **Garachico (E)**: La Montañeta, 922 m.
- **Santa Cruz de Tenerife (F)**: Polígono Costa Sur, 92m.

Las ubicaciones han sido elegidas al contar con estaciones de medición de Grafcan. Sus posiciones se muestran en la Figura 2.1, con la letra indicada en la lista. Se señalan en rojo las estaciones de entrenamiento y en naranja las de test.

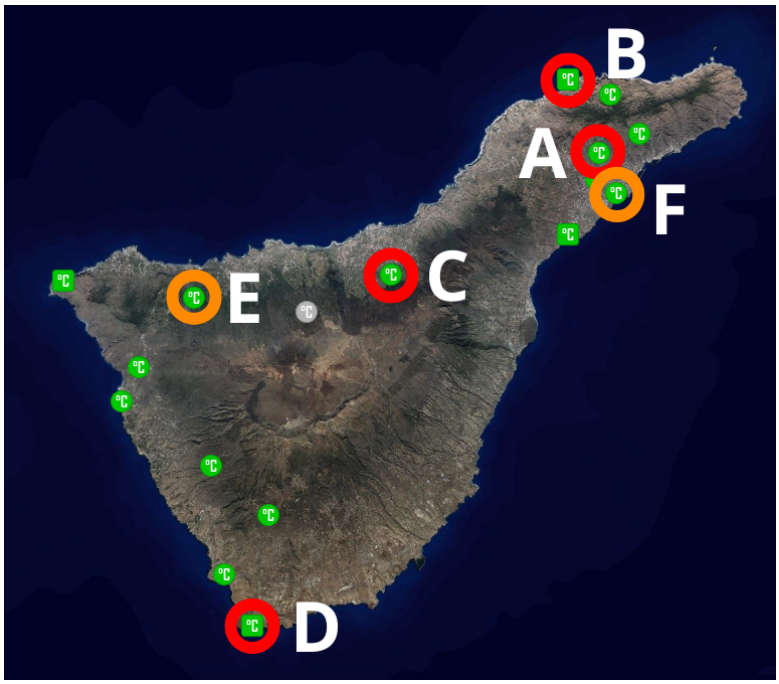

Figura 2.1: Mapa de las estaciones climatológicas Grafcan empleadas.

Inicialmente se valoró emplear las estaciones correspondientes a Los Cristianos, Santiago del Teide o la Punta de Teno, pero fueron descartadas por dos motivos: se detectó que existían períodos prolongados con datos faltantes en las mediciones de Grafcan. Algunas de ellas también exhibían poca correlación entre las mediciones del servicio Grafcan y las de Open-Meteo, lo que podría afectar la calidad de los datos.

2.2. Proceso de adquisición y almacenamiento

Para automatizar la adquisición de datos, se emplea la herramienta de orquestación node-red, que permite crear flujos de información mediante nodos que realizan tareas específicas o ejecutan código de JavaScript. En dicha herramienta se desarrollan dos paneles, uno para cada fuente de datos. Así mismo, dentro de cada panel se desarrollan dos flujos, uno para la adquisición de datos en un intervalo dado, y otro para la adquisición de datos en tiempo real, en particular, se establece la recogida de datos cada 6 horas.

2.2.1. Flujos de adquisición de Grafcan

Debido al funcionamiento de la API de Grafcan, se debe realizar una llamada para obtener la serie temporal de cada variable meteorológica de cada estación. Posteriormente, se unen las series de cada estación en una única serie, que se almacena en una base de datos PostgreSQL. Este flujo está reflejado en la figura 2.2.

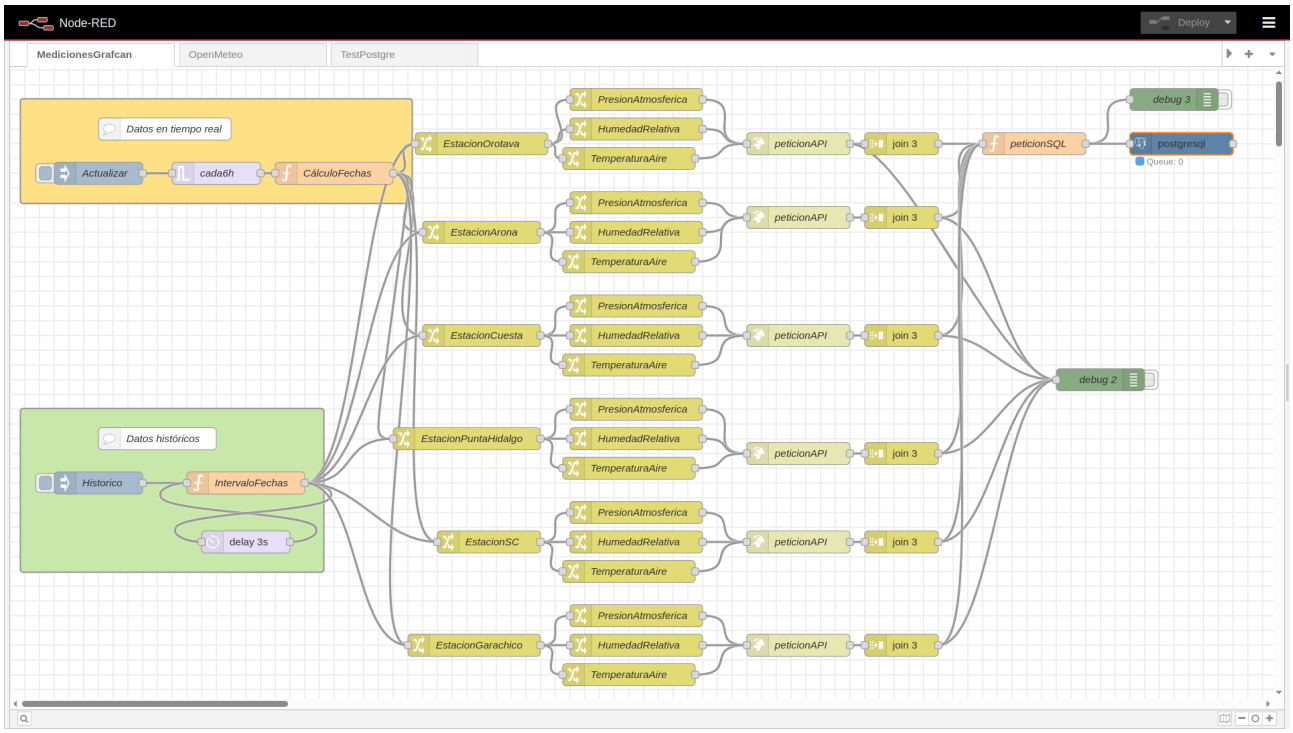

Figura 2.2: Flujo de adquisición de datos de Grafcan en node-red.

Las mediciones de Grafcan se recogen apróximadamente cada 10 minutos, si bien la frecuencia no es consistente y en ocasiones es mayor. Así mismo, los instantes de medición son independientes entre las variables estudiadas. Para manejar esta variabilidad, en este nivel los datos se agregan cada 10 minutos, usando la media de las mediciones del intervalo.

2.2.2. Flujos de adquisición de Open-Meteo

Existe una rama para obtener los datos del modelo ICON y otra para el modelo ARPEGE. Se establecen las coordenadas de cada ubicación como las de la estación de Grafcan seleccionada y se realiza una llamada a la API por cada localización y cada modelo, como se observa en la figura 2.3. Los resultados se almacenan en una base de datos PostgreSQL.

2.2.3. Almacenamiento

Se estudian distintas alternativas para el almacenamiento de los datos, fundamentalmente, servicios de bases de datos como Redis, MongoDB o PostgreSQL. Tras valorar las opciones, se opta por emplear TimescaleDB, una extensión del popular sistema PostgreSQL de bases de datos relacionales, especialmente adaptada para el manejo de series temporales.

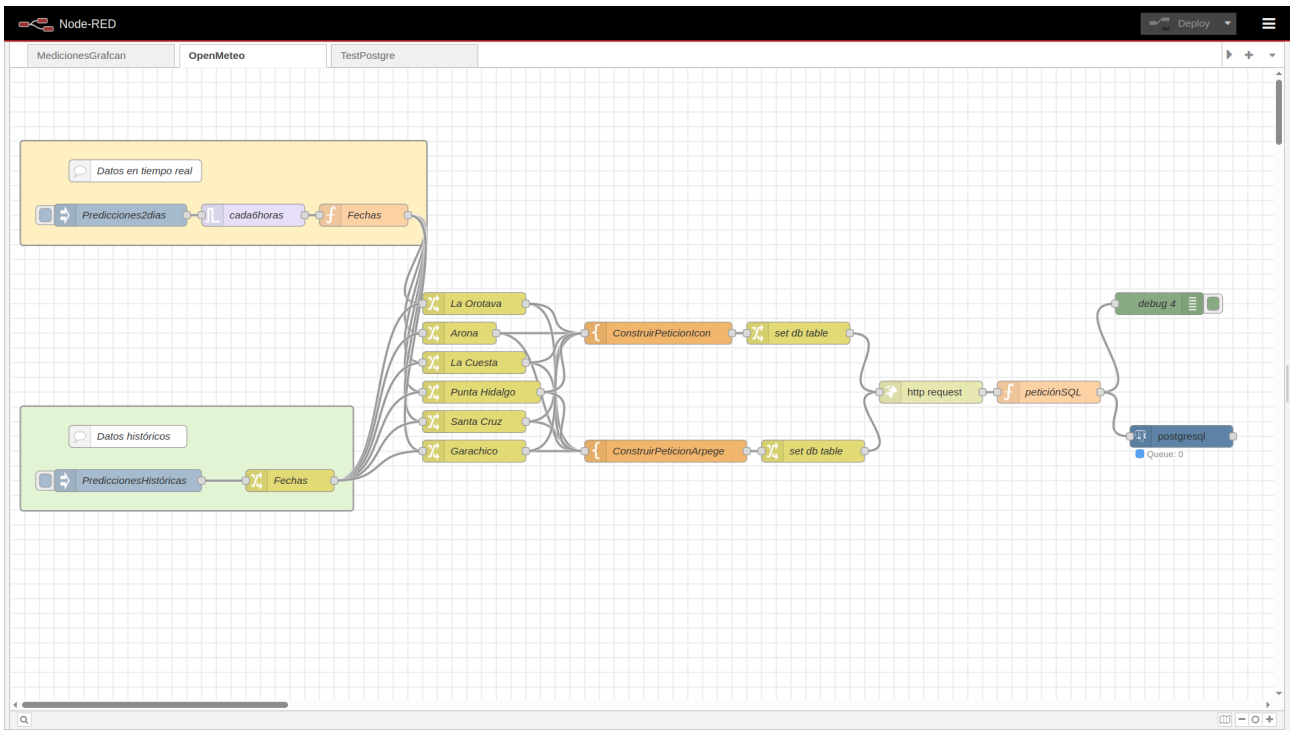

Figura 2.3: Flujo de adquisición de datos de Open-Meteo en node-red.

Se establece un servidor TimescaleDB en un contenedor Docker. Se configura una tabla para cada estación y cada fuente: Grafcan, Open-Meteo ICON y Open-Meteo ARPEGE. Cada tabla emplea como índice y clave primaria la fecha y hora de la medición, así como su zona horaria. Las otras columnas se corresponden a la temperatura media del aire en grados Celsius, la humedad relativa en porcentaje y la presión atmosférica en superficie medida en hPa.

Es importante señalar que las mediciones de Grafcan y Open-Meteo codifican las horas en UTC, en vez de la hora local, puesto que UTC es independiente a los cambios de horario y de esta forma se mantiene la consistencia de los datos.

2.3. Preprocesado

Se desarrolla un cuaderno de Jupyter para realizar el preprocesado de los datos. El proceso descrito en este apartado se aplica de forma separada para cada ubicación.

En primer lugar, se obtienen las series temporales de las 3 fuentes: Grafcan y los dos modelos de Open-Meteo, para el período entre el 1 de marzo de 2023 y el 28 de febrero de 2025. Se agregan los datos con frecuencia horaria mediante la media.

Nota: En la estación de Garachico, usada para el test, el período empleado es del 1 de marzo de 2024 al 28 de febrero de 2025, puesto que el período de 2023 tiene un gran número de datos faltantes.

2.3.1. Visualización

Se visualizan los datos de cada variable para cada año. Podemos ver ejemplos en las figuras 2.4, 2.5 y 2.6.

Se observa claramente que en la presión atmosférica las mediciones de todas las

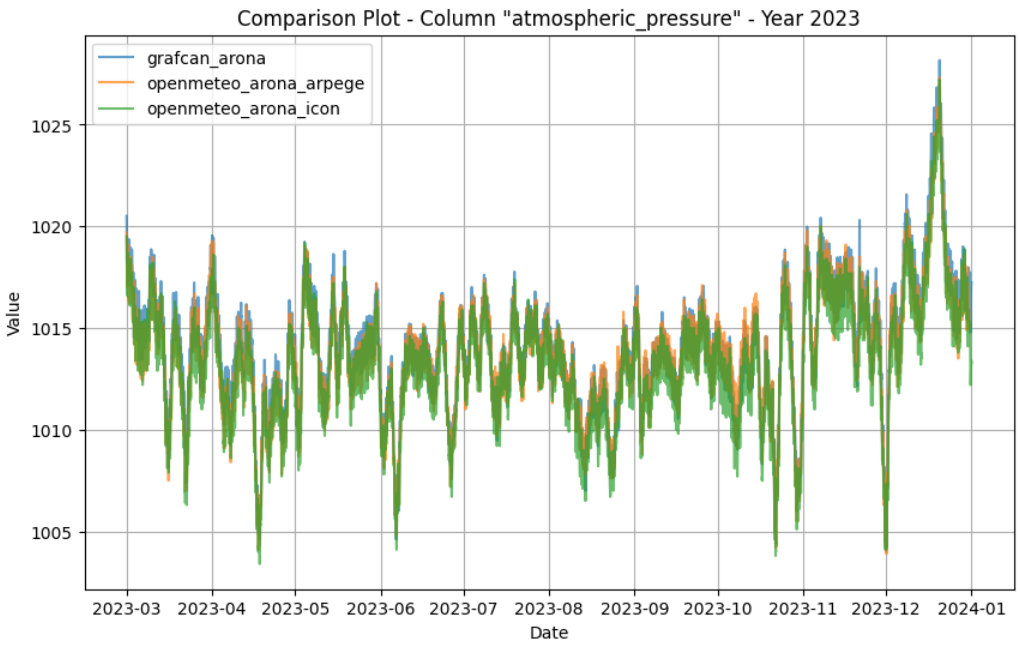

Figura 2.4: Visualización de la presión atmosférica en Arona durante 2023.

fuentes son muy similares. Sin embargo, en la temperatura del aire y la humedad relativa se aprecian diferencias entre las distintas fuentes.

Cabe destacar que en la variable de humedad relativa se observa una gran variabilidad entre los datos de las distintas fuentes, que se constatará más adelante.

2.3.2. Manejo de datos faltantes

Se detectan los datos faltantes para cada fuente. Las estadísticas se muestran en la tabla 2.1. Es reseñable que el modelo ICON no muestra datos faltantes, mientras que el modelo ARPEGE tiene 35 horas faltantes consecutivas, en el período entre el 31 de diciembre de 2023 y el 1 de enero de 2024.

Estación	Grafcan	Open-Meteo ICON	Open-Meteo ARPEGE
La Laguna 1 (La Cuesta)	47	0	35
La Laguna 2(La Punta del Hidalgo)	30	0	35
La Orotava	3	0	35
Arona	17	0	35
Garachico	5	0	0
Santa Cruz de Tenerife	0	0	46

Tabla 2.1: Datos faltantes por estación y fuente de datos (en horas)

Se decide imputar los datos faltantes mediante un método híbrido: si una secuencia de datos faltantes es menor a 5 horas se emplea un método de interpolación cúbica por tramos denominado PCHIP[15], que preserva la forma de los datos y evita oscilaciones indeseadas. Si la secuencia de datos faltantes es mayor a 5 horas, se copian los datos del día anterior en las mismas horas. Este tipo de imputación es común en el ámbito de las series temporales [16].

Los datos sintéticos son etiquetados como tales para poder ser identificados posteriormente.

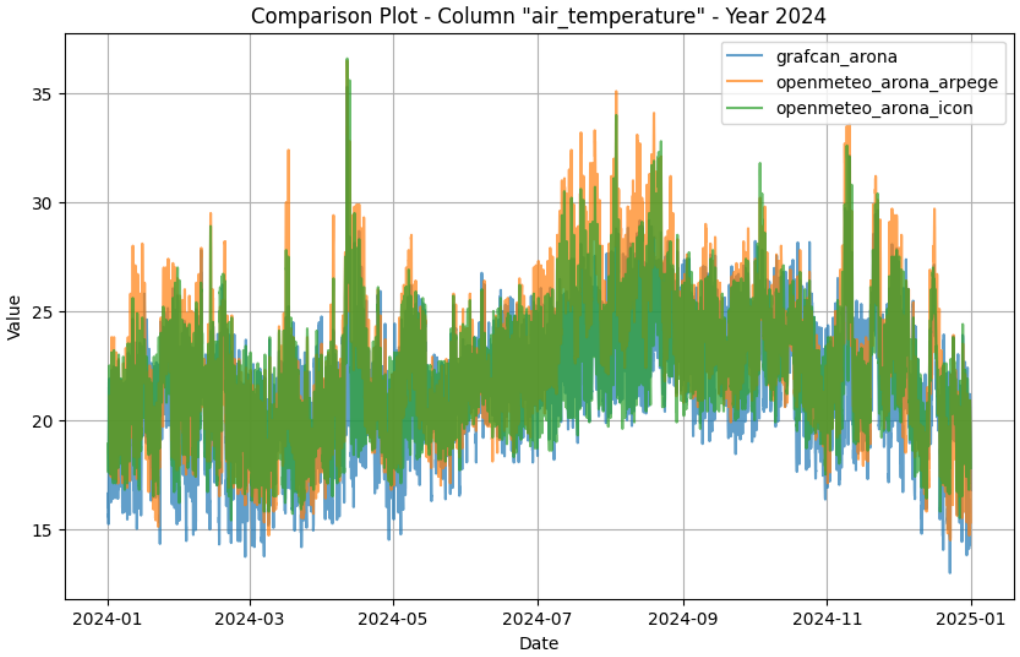

Figura 2.5: Visualización de la temperatura del aire en Arona durante 2024.

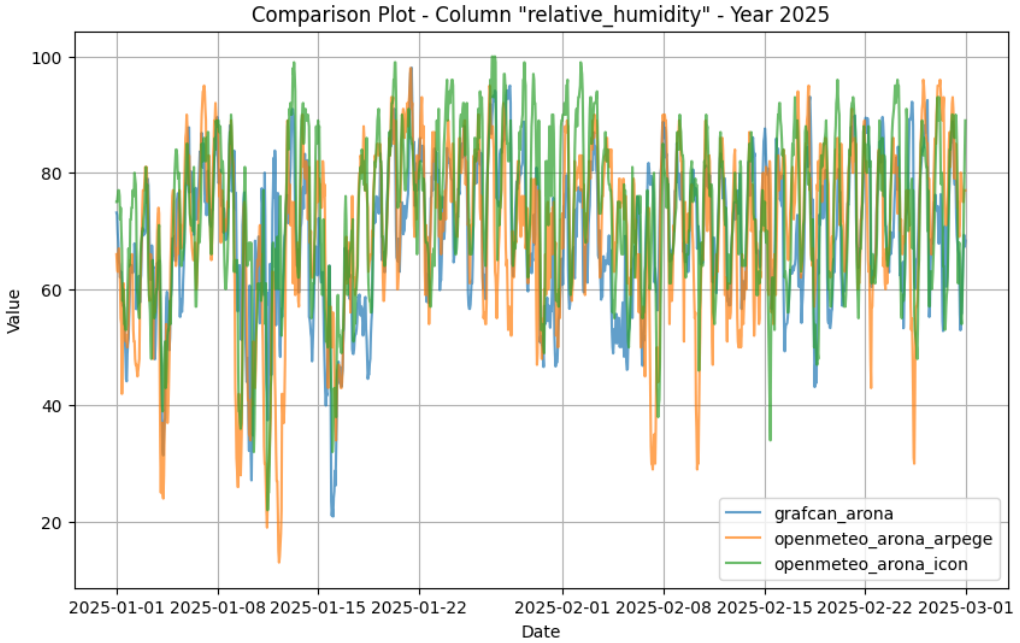

Figura 2.6: Visualización de la humedad relativa en Arona durante 2025.

2.3.3. Selección de modelo de Open-Meteo

Para cada variable se selecciona el modelo de Open-Meteo que mejor se ajusta a los datos de Grafcan. Se emplean diversas métricas: los coeficiente de correlación de Pearson, Spearman y Kendall, así como el error cuadrático medio (MSE) y la distancia euclídea. Se elige el modelo que mejor resultados da en la mayoría de indicadores. Los modelos seleccionados y un ejemplo de las métricas se muestran en las tablas 2.2 y .

Estación	Temperatura del aire	Presión atmosférica	Humedad relativa
La Laguna 1	ICON	ARPEGE	ICON
La Laguna 2	ARPEGE	ARPEGE	ARPEGE
La Orotava	ICON	ARPEGE	ICON
Arona	ICON	ARPEGE	ICON
Garachico	ICON	ICON	ICON
Santa Cruz de Tenerife	ICON	ARPEGE	ICON

Tabla 2.2: Modelo de Open-Meteo seleccionado para cada variable y estación

Tabla 2.3: Métricas de similitud de modelos Open-Meteo con Grafcan para Arona

Métrica	**air_temperature**		**atmospheric_pressure**		**relative_humidity**	
	ICON	**ARPEGE**	**ICON**	**ARPEGE**	**ICON**	**ARPEGE**
Pearson	0.8871	0.8415	0.9890	0.9891	0.6281	0.3838
Spearman	0.9116	0.8519	0.9859	0.9866	0.6026	0.2748
Kendall	0.7489	0.6650	0.9048	0.9070	0.4423	0.1973
MSE	2.8616	5.6179	1.1085	0.5754	208.6453	406.1746
Euclidean Distance	224.0627	313.9447	139.4523	100.4768	1913.2363	2669.4431

Observamos que, en general, el modelo ICON es el que mejor se ajusta a las variables de temperatura del aire y humedad relativa, mientras que el modelo ARPEGE es el que mejor se ajusta a la presión atmosférica.

Resulta reseñable señalar que la diferencia entre los modelos de Open-Meteo y Grafcan es mucho mayor en la variable de humedad relativa que en las otras. Esto puede ser relevante más adelante, de cara al rendimiento de los modelos de predicción.

Tras seleccionar el modelo de Open-Meteo para cada variable, se crea un dataset unificado con las 3 variables. De esta forma se dispone de un dataset de Open-Meteo y otro de Grafcan con las 3 variables para cada estación.

2.3.4. Detección de valores anómalos

Para la detección de valores anómalos, en primer lugar se emplea el método del rango intercuartílico (IQR). Sin embargo, se observa en los histogramas que la distribución de los datos no es puramente gaussiana, existiendo sesgos y colas largas. Por ejemplo, en la figura 2.7 se observa que la humedad presenta un sesgo a la izquierda, mientras que en la figura 2.8 se observa que la temperatura presenta una cola larga a la derecha.

Por esto, se decide emplear como método de detección de anomalías el de los K vecinos más cercanos (KNN), una alternativa robusta que permite detectar anomalías en distribuciones no gaussianas [17].

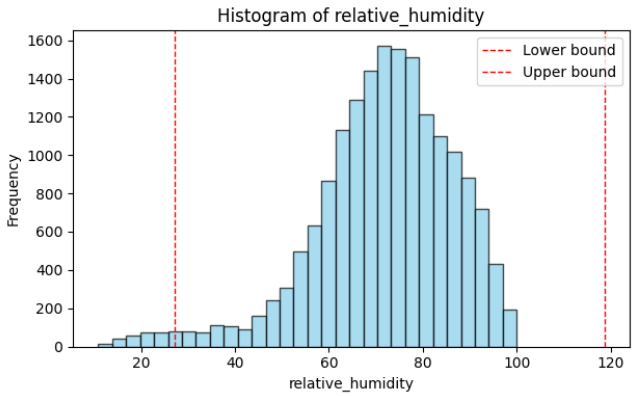

Figura 2.7: Histograma de la humedad relativa en Arona.

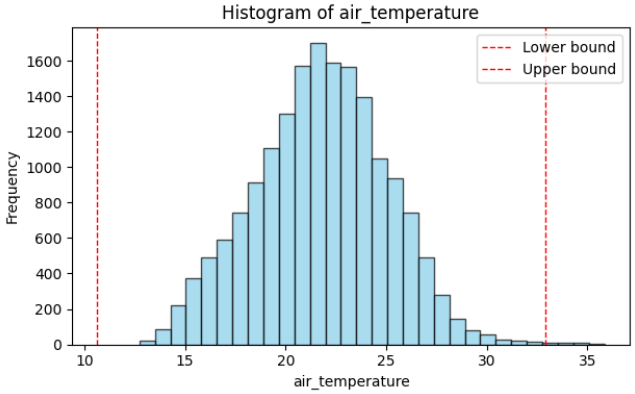

Figura 2.8: Histograma de la temperatura del aire en Arona.

Se aplica el método de KNN a cada variable por separado, con k=10 y límite de distancia 3 veces la desviación típica. Respecto al número de vecinos (K), se observa al graficar la distancia media de los K vecinos más cercanos frente a K que es poco relevante, por lo que se escoge arbitrariamente. Una vez fijado K, para determinar el límite se considera que en una distribución gaussiana el 99,7 % de los datos se encuentran dentro de 3 desviaciones típicas y se comprueba empíricamente este valor, así como otros cercanos, observando los histogramas de los datos detectados como anómalos.

Los valores anómalos son etiquetados como tales para poder ser identificados posteriormente. Se pueden observar ejemplos de las distancias en la figura 2.9. En las figuras 2.10 y 2.11 se muestran ejemplos de los valores anómalos detectados.

Nota: Se muestran los valores anómalos independientemente de la variable respecto a la que se detectaron.

2.3.5. Exploración de frecuencias - Dominio de Fourier

Se realiza un análisis de Fourier para cada variable. De esta forma, se pueden observar las frecuencias dominantes en los datos. Este análisis es relevante para detectar las frecuencias a emplear en la codificación de la información temporal, que se aborda en el siguiente apartado.

Se emplea la transformada rápida de Fourier (FFT), se filtran las frecuencias positivas, se acota a las frecuencias mayores a 10^{-3} (unos 16,66... minutos) y se grafican haciendo uso de escala logarítmica en el eje X. El pseudocódigo se muestra en 2.12. Se pueden

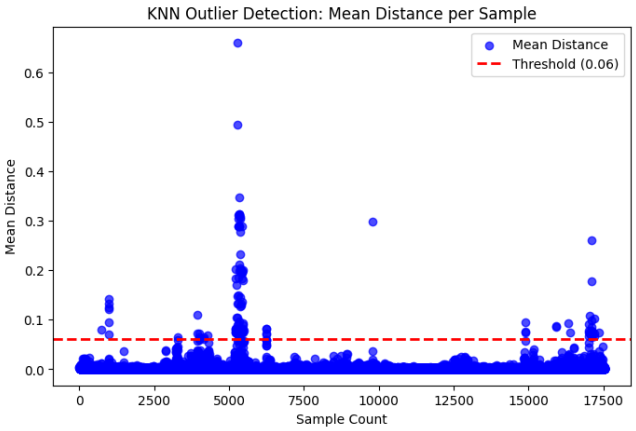

Figura 2.9: Distancias media de los K vecinos más cercanos para la temperatura del aire en Arona.

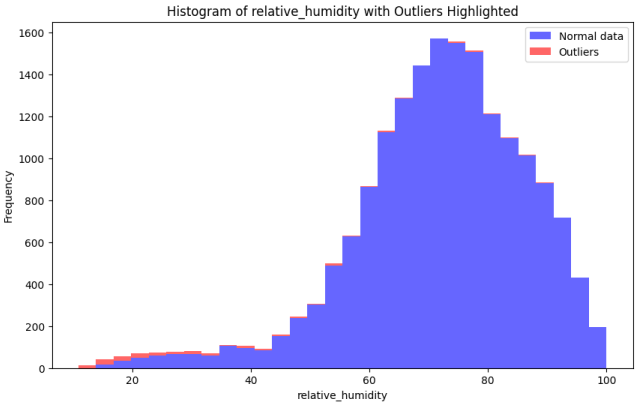

Figura 2.10: Histograma de la humedad relativa en Arona con outliers detectados con knn.

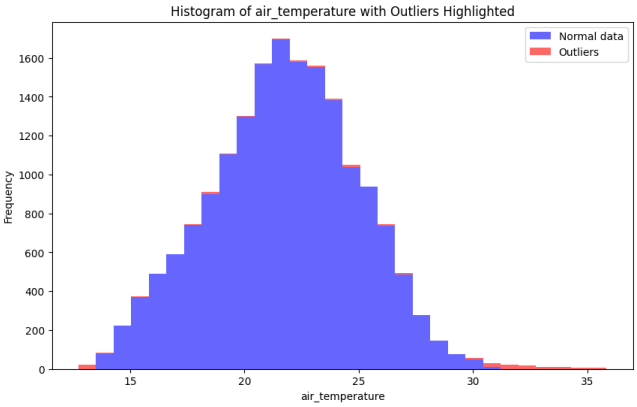

Figura 2.11: Histograma de la temperatura del aire en Arona con outliers detectados con knn.

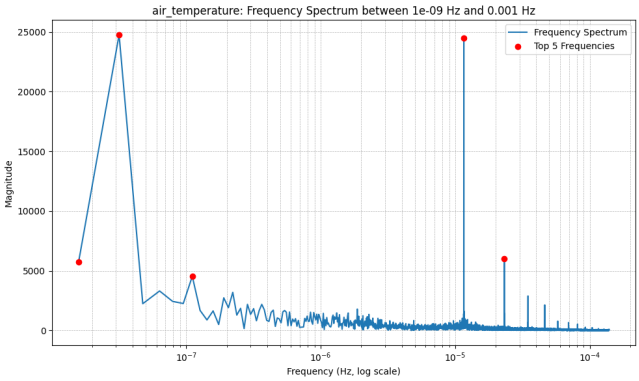

Figura 2.13: Transformada rápida de Fourier de la temperatura del aire en Arona.

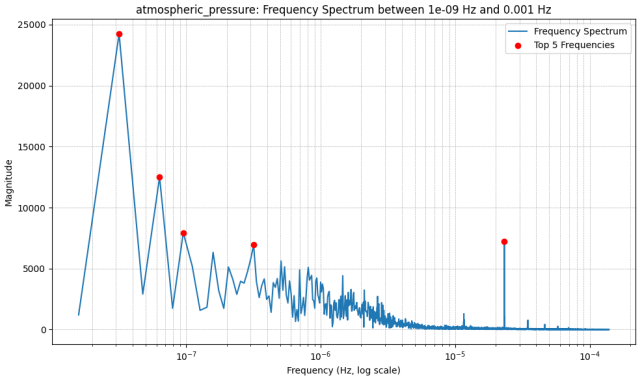

Figura 2.14: Transformada rápida de Fourier de la presión atmosférica en Arona.

observar ejemplos en las figuras 2.13, y 2.14. Se señalan las 5 frecuencias de mayor magnitud con un punto en rojo.

Pseudocódigo Cálculo de FFT Positiva

Función calcular_fft_positiva($valores$, $intervalo\_muestreo$):
    # 1. Computar la transformada rápida de Fourier del vector de entrada
    fft_result = FFT($valores$)
    # 2. Generar los intervalos de frecuencia para cada punto de la FFT
    $frecuencias$ = FFTFREQ(longitud($valores$), $intervalo\_muestreo$)
    # 3. Calcular la mitad de la longitud para aislar frecuencias no negativas
    $mitad$ = piso(longitud($valores$) / 2)
    # 4. Extraer solo la parte de frecuencias positivas
    $resultado\_fft\_positivo$ = fft_result[0:$mitad$]
    $frecuencias\_positivas$ = $frecuencias$[0:$mitad$]
    # 5. Calcular la magnitud de los coeficientes complejos resultantes
    $magnitud$ = valor_absoluto($resultado\_fft\_positivo$)
    # 6. Devolver el eje de frecuencias positivas y su espectro de magnitudes
    Retornar $frecuencias\_positivas$, $magnitud$

Figura 2.12: Pseudocódigo Cálculo de FFT Positiva

En la temperatura y humedad destacan las frecuencias de 24 y 8772 horas, correspondiente esta última a 365,5 días, lo que es razonable considerando que de los 2 años de

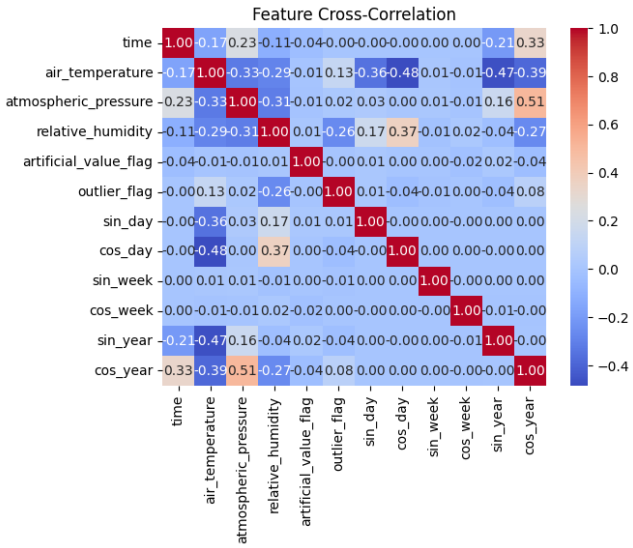

Figura 2.15: Mapa de correlación entre las variables del dataset.

datos, uno es bisiesto. Respecto a la presión atmosférica, la más relevante es la de 12 horas, algo debido a la naturaleza de esta variable, que presenta este ciclo debido a un fenómeno conocido como mareas térmicas [18]. Así mismo, la presión también presenta un pico en la frecuencia anual y la de medio año.

2.3.6. Codificación de la información temporal

Se decide codificar la información temporal mediante el uso de senos y cosenos de las frecuencias dominantes. En base a los resultados del análisis de Fourier, se emplean la frecuencia de 24 horas y del año, teniendo especial cuidado para detectar si el año es bisiesto o no. De esta forma, se añaden 4 variables adicionales al dataset: sin(día), cos(día), sin(año) y cos(año).

Se estudia la frecuencia semanal, pero se descarta al existir poca correlación.

2.3.7. Estudio de correlación

Se estudia la correlación entre las distintas variables que conforman el dataset mediante una matriz de correlación con el coeficiente de Pearson. Se observa que las variables climáticas tienen una correlación negativa de en torno a 0.3, lo que indica que las variables están relacionadas de forma baja/moderada e inversamente proporcional 2.15.

2.4. Creación de ventanas de datos

Para la predicción de series temporales, los modelos neuronales requieren como entrada los valores de las variables en un número de instantes, en nuestro caso horas, previos, que denominaremos P. Así mismo, se define un número de horas a predecir, que denominaremos N. De esta forma, un modelo requiere P horas de datos para predecir N horas futuras. Esto es lo que se conoce como ventana de datos.

En su versión más simple, la univariable, consta de una componente X, los valores de la variable en P pasos, y una componente Y, un vector de tamaño N con los valores a predecir. No obstante, puede ser relevante incluir información adicional exógena sobre

los valores a predecir, como la codificación temporal (o en otros dominios, si el día es festivo, etc).

Optamos por incluir esta información y construir ventanas de datos con 3 componentes, X e Y, ya definidas, así como F, formada por las 4 variables de codificación temporal para cada paso a predecir N. Empíricamente comprobamos que la inclusión de esta información adicional mejora ligeramente el rendimiento del modelo.

En la práctica, estas ventanas se construyen mediante la técnica de ventana deslizante, que consiste en recorrer la serie temporal desplazando la ventana de datos en un número de pasos, que denominaremos S. Establecemos S=6 para que la ventana se desplace cada 6 horas, con el fin de buscar un balance entre evitar la redundancia de datos que existiría si usásemos un paso menor y mantener el máximo de información posible.

Como datos pasados empleamos la variable meteorológica estudiada, así como las restantes como covariables. Se comprueba empíricamente que añadir estas covariables mejora el rendimiento del modelo. También se emplea como datos pasados las variables de codificación temporal.

Es decir, la estructura de una ventana es:

- X: un vector de tamaño P * 7, que contiene los valores de la variable objetivo y las covariables en los pasos previos.
- F: un vector de tamaño N * 4, que contiene las variables de codificación temporal para cada paso a predecir.
- Y: un vector de tamaño N, que contiene los valores a predecir.

2.4.1. Implementación

En primer lugar, se define una función *df_raw_windows* que recibe como parámetros:

- df: un dataframe de pandas, que es una estructura de datos de python similar a una tabla, correspondiente a un dataset de una ubicación,
- P: el número de pasos pasados.
- N: el número de pasos a predecir.
- S: el número de pasos a desplazar la ventana.
- past_features: un array con los nombres de las variables usadas como entradas.
- future_features: un array con los nombres de las variables exógenas usadas como datos futuros.
- target: el nombre de la variable objetivo a predecir.

  La función devuelve un objeto con 3 atributos,*past_variables*, *future_variables* e *y*. Siendo cada uno un arreglo con los datos de cada componente de las ventanas de datos. Es decir, la ventana i ésima está formada por los iésimos elementos de cada uno de los 3 atributos. Esta estructura se debe al comportamiento de las librerías de los modelos de aprendizaje profundo empleados, que procesan los datos de forma más eficiente en este formato, como se verá más adelante.

  El comportamiento de la función principal es el siguiente:

Pseudocódigo Creación de Ventanas de Datos

```
train_data = {}
test_data = {}
Para cada variable objetivo v:
    Para cada ubicación u:
    location_data = []
        Para cada dataset d en la ubicación u:
            ventanas = df_raw_windows()
            location_data.añadir(ventanas)
        windows_count = len(location_data[0][y])
        overlap_windows = ceil((past_n + future_n)/step)
        test_indexes = random_sample(windows_count, test_percent)
        forbidden = set()
        Para cada index en test_indexes:
            # Prohibir el índice y las siguientes ‘overlap_windows - 1‘ ventanas
            Para cada j en rango(overlap_windows):
                forbidden.add(index + j)
        train_indexes = rango(windows_count) - forbidden
        Para cada dataset d en location_data:
            past_variables_windows = location_data[d]["past_variables"]
            future_variables_windows = location_data[d]["future_variables"]
            y_windows = location_data[d][ɀ"]
            # Construcción de conjuntos de entrenamiento y validación
            train_past_variables = [past_variables_windows[i] para i en train_indexes]
            train_future_variables = [future_variables_windows[i] para i en train_indexes]
            train_y = [y_windows[i] para i en train_indexes]
            test_past_variables = [past_variables_windows[i] para i en test_indexes]
            test_future_variables = [future_variables_windows[i] para i en test_indexes]
            test_y = [y_windows[i] para i en test_indexes]
            train_data[v]["past_variables"].extender(train_past_variables)
            train_data[v]["future_variables"].extender(train_future_variables)
            train_data[v]["y"].extender(train_y)
            test_data[v]["past_variables"].extender(test_past_variables)
            test_data[v]["future_variables"].extender(test_future_variables)
            test_data[v]["y"].extender(test_y)
```

Figura 2.16: Pseudocódigo Creación de Ventanas de Datos

La construcción del conjunto de entrenamiento y test se realiza de forma aleatoria, algo poco común en el ámbito de la predicción de series temporales, puesto que habitualmente se emplea como conjunto de validación un período al final de la serie. Pero, de esta forma, creemos que se puede mejorar el modelo, al disponer de un conjunto de validación más diverso y representativo de la meteorología a lo largo del año.

Para cada ubicación se eligen los índices de las ventanas de test aleatoriamente. Estos índices son los mismos para los dos datasets de una localización, puesto que los datos son bastante similares entre ambos y queremos evitar el filtrado de información. Cabe destacar la consideración de ventanas "prohibidas", que

se descartan del conjunto de entrenamienno para evitar filtrar datos. De esta forma, todas las ventanas previas que estén solapadas con una ventana de test son descartadas.

De todas las ventanas creadas originalmente, se escogen un 10 % para el conjunto de test. Como se descartan las ventanas solapadas, según los parámetros empleados, la división resultantes es de alrededor de un 14 % de ventanas de test y un 86 % de ventanas de entrenamiento. Se eligen estos porcentajes puesto que se comprueba empíricamente que a un mayor número de ventanas de entrenamiento el rendimiento del modelo mejora, pero se fija el límite del 10 % para conseguir un conjunto de test representativo y diverso. Esto supone unas 2.300 ventanas de validación y entre 13.900 y 12.500 de entrenamiento según las características de la ventana elegidas.

2.4.2. Normalización

Se utilizan las ventanas del conjunto de entrenamiento para calcular las características de la normalización. Usamos la estandarización o normalización Z-score, que consiste en restar la media y dividir por la desviación típica. De esta forma, los datos quedan centrados en 0 y con una desviación típica de 1.

La normalización se aplica a la variable objetivo y las covariables meteorológicas, tanto en los datos pasados como los datos a predecir (y). No se aplica a las variables de codificación temporal, ya que al tratarse de senos y cosenos, su rango ya es limitado entre -1 y 1.

Debemos almacenar los parámetros de normalización puesto que serán necesarios para tratar cualquier dato que se suministre en el futuro al sistema.

Capítulo 3

Modelos de predicción

Se desarrollan 4 modelos: un modelo ARIMA como base de comparación, un modelo LSTM, un modelo CNN y un híbrido LSTM-CNN.

3.1. ARIMA

Los modelos ARIMA (Autoregressive Integrated Moving Average) son una tipo de modelos estadísticos utilizados históricamente para el análisis y la predicción de series temporales univariable.

Utilizamos como base de comparación los resultados de un ARIMA para cada serie temporal: cada variable en cada ubicación y cada fuente de datos.

Para emplear un modelo ARIMA, es necesario ajustar cada uno de sus parámetros, que son:

- p: el número de términos autorregresivos (AR).
- d: el número de diferencias necesarias para hacer la serie estacionaria (I).
- q: el número de términos de media móvil (MA).

3.1.1. Calibración de términos autorregresivos

Para determinar el número de términos autorregresivos, se utiliza la función `plot_pacf` de la librería `statsmodels`, que permite visualizar la función de autocorrelación parcial (PACF) de la serie temporal. Observamos ejemplos de los resultados en las figuras 3.1, 3.2.

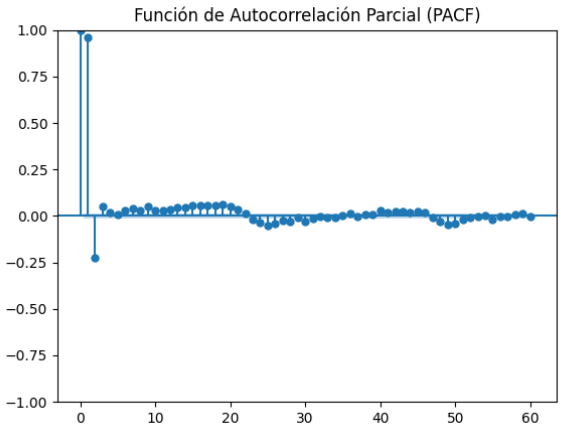

Figura 3.1: Gráfico de PACF para la serie de humedad relativa en La Laguna de Grafcan

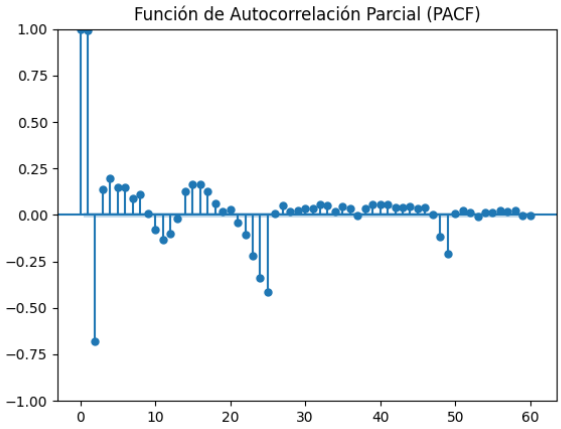

Figura 3.2: Gráfico de PACF para la serie de presión atmosférica en La Laguna de Grafcan

Analíticamente, se emplea el algoritmo descrito en 3.3. Se determina $p = 4$ para las series de temperatura del aire y humedad relativa, mientras que se obtiene $p = 8$ en el caso de la presión atmosférica.

Pseudocódigo Selección de Orden AR vía PACF

```
Función seleccionar_orden_AR(serie, max_lags):
    # 1. Calcular la PACF hasta max_lags retardos
    pacf_valores = PACF(serie, nlags=max_lags)
    # 2. Definir intervalo de confianza al 95 %
    intervalo_confianza = 1.96 / raiz_cuadrada(longitud(serie))
    # 3. Buscar el primer retardo no significativo
    Para lag en 1 ... max_lags:
        Si valor_absoluto(pacf_valores[lag]) < intervalo_confianza:
            imprimir("El mejor orden AR sugerido por la PACF es:", lag − 1)
            Retornar lag − 1
    # 4. Si todos los retardos son significativos
    imprimir("Todos los retardos hasta", max_lags, "son significativos.)
    Retornar max_lags
```

Figura 3.3: Pseudocódigo para Selección de Orden AR usando PACF

3.1.2. Estudio de la estacionariedad

Se emplea la prueba de Dickey-Fuller aumentada (ADF) para determinar la estacionariedad de las series temporales. La prueba ADF es una prueba estadística que evalúa la presencia de una raíz unitaria en una serie temporal, lo que indica si la serie es estacionaria o no. La hipótesis nula de la prueba ADF establece que la serie temporal tiene una raíz unitaria, lo que implica que no es estacionaria. Si el valor p de la prueba es menor que un nivel de significancia predefinido (por ejemplo, 0.05), se rechaza la hipótesis nula y se concluye que la serie es estacionaria.

Empleamos como significancia un nivel de 0.05, y se observa que todas las series temporales son estacionarias, por lo que no es necesario aplicar diferencias, es decir, $d = 0$.

3.1.3. Determinación del orden de la media móvil

Para calcular el número de términos de media móvil, se utiliza la función de Autocorrelación `ACF` de la librería `statsmodels`. Dicha función retorna los coeficientes de autocorrelación para diferentes retardos.

Analíticamente, se emplea el algoritmo descrito en 3.4, cuya idea principal consiste en comprobar para cada retardo si el coeficiente de autocorrelación es significativo, finalizando cuando se encuentra el primer retardo no significativo.

Se calcula el intervalo de confianza como 1.96 por la raíz cuadrada de la longitud de la serie temporal, ya que la distribución de la autocorrelación estimada en cualquier lag se aproxima a una distribución normal con media 0 y desviación estándar $\frac{1}{\sqrt{n}}$, donde n es el número de observaciones. De esta forma se obtiene un intervalo de confianza al 95 % para la autocorrelación. No obstante, se establece un límite inferior de 0.05 para evitar que el nivel de significancia sea demasiado bajo y se detecte el ruido como una señal significativa.

Pseudocódigo Selección de Orden MA vía ACF

```
1: Función seleccionar_orden_MA(serie, max_lags, min_confidence):
2:     # 1. Calcular la ACF hasta max_lags retardos
3:     acf_valores = ACF(serie, nlags=max_lags)
4:     # 2. Definir intervalo de confianza al 95 %
5:     intervalo_confianza = max(1.96 / raiz_cuadrada(longitud(serie)), min_confidence)
6:     # 3. Buscar el primer retardo no significativo
7:     Para lag en 1 ... max_lags:
8:         Si valor_absoluto(acf_valores[lag]) < intervalo_confianza:
9:             imprimir("El mejor orden MA sugerido por la ACF es:", lag − 1)
10:        Retornar lag − 1
11:    Retornar max_lags
```

Figura 3.4: Pseudocódigo para Selección de Orden MA usando ACF

El algoritmo descrito retorna el máximo número de retardos para valores elevados de lags, por lo que se analiza el gráfico de la ACF y se observa que no existe un punto de corte claro, sino que existe un descenso suave y progresivo para todas las variables 3.5 3.6. Por lo tanto, se fija $q = 0$.

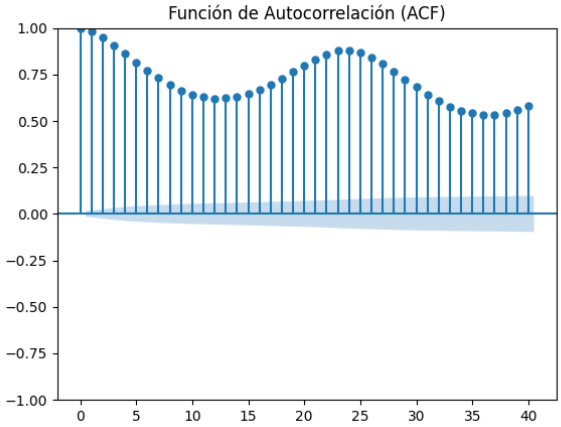

Figura 3.5: Gráfico de ACF para la serie de temperatura del aire en La Laguna de Grafcan

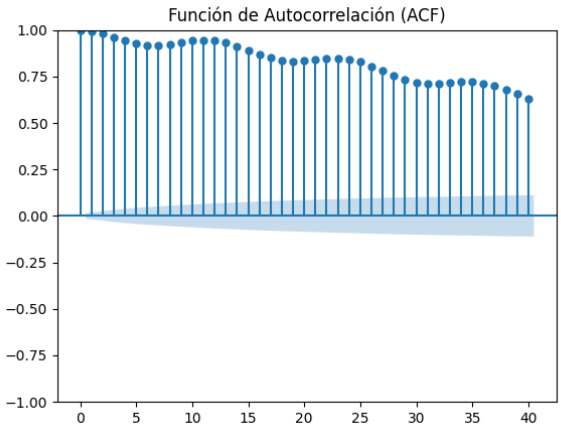

Figura 3.6: Gráfico de ACF para la serie de presión atmosférica en La Laguna de Grafcan

3.1.4. Evaluación

Para evaluar los modelos ARIMA empleamos la validación progresiva (walk-forward validation), que consiste en un bucle en el que se entrena al modelo, se realiza una predicción y se reentrena con los datos más recientes antes de realizar la siguiente predicción. Este proceso se repite hasta que se alcanza el final del periodo de validación 3.7. Emplearemos como métrica la raíz del error cuadrático medio (RMSE), que se calcula como la raíz cuadrada de la media de los errores al cuadrado entre las predicciones y los valores reales. Esta medición proporciona el error en la unidad de la variable.

Pseudocódigo Validación Walk-Forward con ARIMA

1: **Función** validacion_walk_forward($test$, $history$, $pasos\_prediccion$, $orden\_ARIMA$):
2: $predicciones$ = lista_vacía()
3: $valores\_reales$ = lista_vacía()
4: **Para** t **en** rango(0, longitud($test$) $-$ $pasos\_prediccion$ $+$ 1, $pasos\_prediccion$):
5: $modelo$ = ARIMA($history$, orden=$orden\_ARIMA$)
6: $ajuste$ = ajustar($modelo$)
7: $pronostico$ = predecir($ajuste$, pasos=$pasos\_prediccion$)
8: $predicciones$.añadir($pronostico$)
9: $valores\_reales$.añadir($test[t : t + pasos]$)
10: $history$.añadir($test[t : t + pasos]$)
11: $rmse$ = raíz_cuadrada(ECM($valores\_reales$, $predicciones$))
12: **Retornar** $rmse$

Figura 3.7: Pseudocódigo para validación walk-forward usando ARIMA

Se toma como ejemplo la serie de temperatura del aire en Arona de Open-Meteo, con un horizonte de predicción de 3 horas. En la figura 3.8 se observan los residuos (diferencia entre valor real y predicción), mientras que en 3.9 se observa la estimación de la distribución de la densidad de los residuos. Al estar centrada en 0 y tener una forma similar a la campana de Gauss, se puede

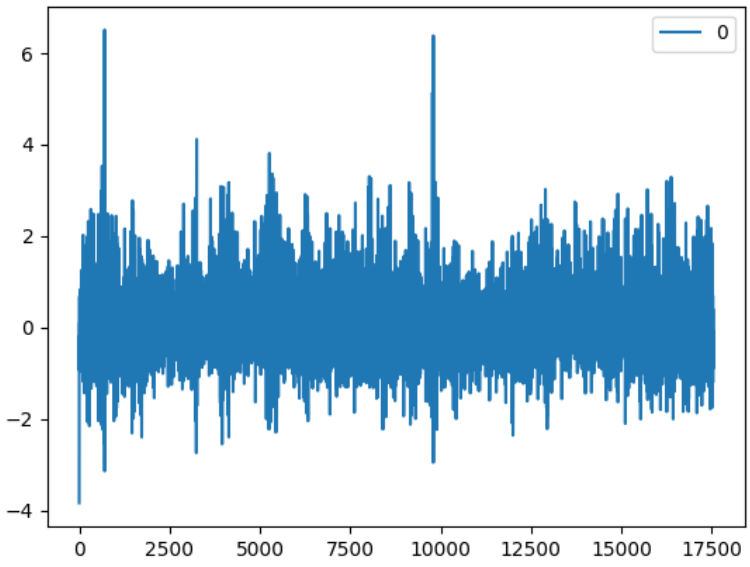

Figura 3.8: Residuos del modelo ARIMA para la serie de temperatura del aire en Arona de Open-Meteo

concluir que el modelo ARIMA se ajusta bien a los datos. Finalmente, en la figura 3.10 se muestra un ejemplo de los resultados obtenidos.

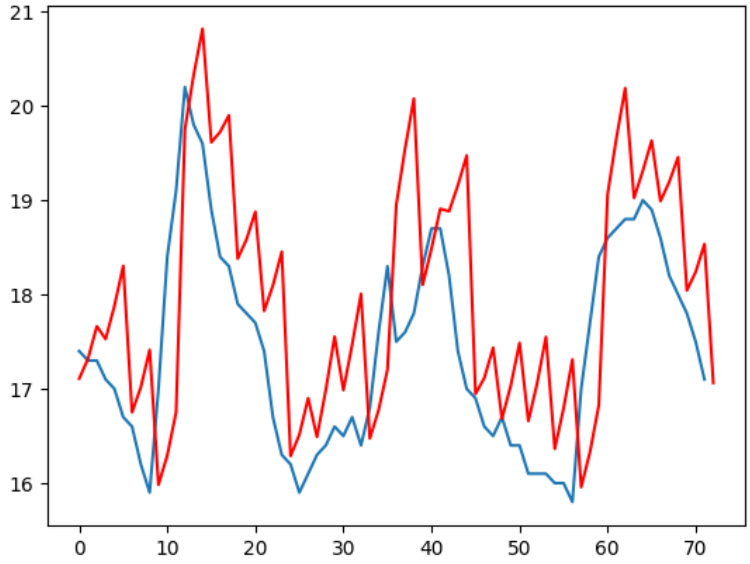

Figura 3.10: Resultados del modelo ARIMA para la serie de temperatura del aire en Arona de Open-Meteo

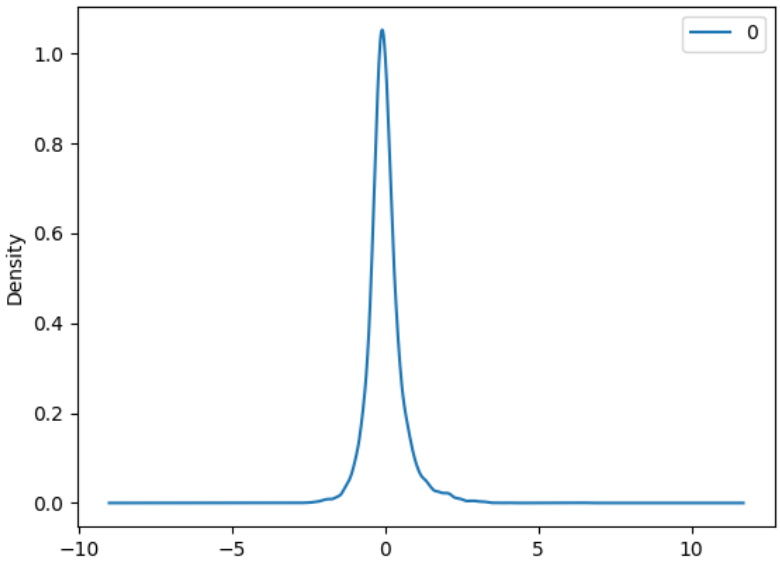

Figura 3.9: Distribución de los residuos del modelo ARIMA para la serie de temperatura del aire en Arona de Open-Meteo

3.2. Modelos de aprendizaje profundo

Existen diferentes frameworks para el desarrollo de modelos de aprendizaje profundo, destacando TensorFlow, PyTorch o JAX/Flax como los más extendidos. Después de evaluar alternativas, se opta por TensorFlow debido a su amplia comunidad de usuarios y su extensa documentación, así como por su integración con Keras, una biblioteca de alto nivel que facilita la creación de modelos de aprendizaje profundo.

3.2.1. Creación del dataset con TensorFlow

Para crear el dataset, se emplea la función de TensorFlow `Dataset.from_tensor_slices()` que permite crear un objeto Dataset a partir de tensores de entrada y salida. Dichos tensores son: la dupla (datos_pasados, datos_futuros) y el arreglo con los datos de validación.

Se crean dos datasets: uno para el entrenamiento y otro para la validación. Ambos se dividen en lotes (batches) de tamaño 64. Este número se elige tras realizar pruebas con diferentes tamaños de lotes en los distintos modelos, y se observa que, pese a ser un poco más inestable, ya que un número pequeño de muestras por lote puede causar que el modelo cambie de forma exagerada en algún lote, con ese valor se consiguen los modelos con menor error.

En el caso del conjunto de entrenamiento, tras crear los lotes se baraja, para evitar el sobreajuste y mejorar la generalización del modelo.

3.2.2. Consideraciones generales

Se emplea como métrica de pérdida el error mse de las ventanas de validación.

Además, para controlar el entrenamiento se emplean dos funciones de retorno (callbacks):

- `EarlyStopping`: detiene el entrenamiento si no se observa mejora en la métrica de validación durante un número determinado de épocas (patience). De esta forma se intenta evitar el sobreajuste. Además, se guarda el modelo con la mejor métrica de validación.
- `ReduceLORonPlateau`: reduce la tasa de aprendizaje por un factor si no se observa mejora en la métrica de validación durante un número determinado de épocas (patience). Se basa en la idea de que cuando el modelo no mejora, es posible que se encuentre cerca de un mínimo local y que una tasa de aprendizaje más baja pueda ayudar a acercarse más.
  Para determinar estas variables, además de otros hiperparámetros como el ratio de aprendizaje, se hace uso de una herramienta de ajuste fino (fine-tuning). En particular, se emplea Keras Tuner, y el método de búsqueda en rejilla (grid search), que consiste en definir un espacio de búsqueda para los hiperparámetros y evaluar todas las combinaciones posibles. Se realizan 5 ejecuciones por prueba y se emplea la media de los resultados.

3.2.3. CNN

Las redes convolucionales (CNN) son un tipo de red neuronal diseñadas originalmente para procesar datos con estructura de cuadrícula (como imágenes), pero también se pueden aplicar a series temporales (1D). Su base es la convolución, una operación en la que uno o varios filtros (o núcleos) se deslizan sobre la entrada, calculando productos escalares para extraer características locales relevantes. Cada filtro tiene un tamaño (kernel size) y avanza con un paso (stride) determinado; además, puede aplicarse padding (relleno) para controlar las dimensiones de la salida. En una capa suelen existir múltiples filtros distintos, de manera que cada uno aprende a detectar un patrón diferente (por ejemplo, tendencias, picos o frecuencias concretas en series temporales).
Tras las capas convolucionales, es común intercalar capas de pooling (sub-muestreo), como max-pooling o average-pooling, para reducir dimensión, extraer características más robustas y controlar el sobreajuste. Durante el entrenamiento de la red, los pesos de los filtros se ajustan para minimizar el error de predicción.

3.2.4. LSTM

Las redes LSTM (Long Short-Term Memory) son un tipo de red neuronal recurrente (RNN) que se utilizan para modelar secuencias temporales. Las LSTM son capaces de aprender patrones a largo plazo en los datos, lo que las hace adecuadas para tareas de predicción de series temporales. Se basan en la idea de utilizar celdas de memoria que pueden almacenar información durante períodos prolongados, lo que les permite recordar información relevante de entradas anteriores. Cada unidad LSTM incluye una celda de memoria y tres puertas que regulan el flujo de información:

- ◇ Puerta de entrada: decide qué información nueva se añade a la celda de memoria.
- ◇ Puerta de olvido: determina qué información de la celda de memoria se olvida.
- ◇ Puerta de salida: controla qué información de la celda de memoria se envía como salida.

  Son relevantes también las capas bidireccionales. Introducidas en 2005 por Graves y Schmidhuber [19], las LSTM bidireccionales permiten que la red procese la secuencia de entrada en ambas direcciones (hacia adelante y hacia atrás), lo que mejora la capacidad de capturar dependencias a largo plazo en los datos, ayudando, por ejemplo, a evitar ambigüedades debido al mayor contexto.

3.2.5. LSTM-CNN

Los modelos híbridos LSTM-CNN combinan las ventajas de las redes LSTM y las redes convolucionales. Intercalan capas convolucionales y LSTM para capturar tanto patrones locales como dependencias a largo plazo en los datos de series temporales. La arquitectura típica de un modelo LSTM-CNN incluye capas convolucionales iniciales para extraer características locales, seguidas de capas LSTM para capturar patrones temporales a largo plazo.

3.3. Comparativa inicial

Para establecer una referencia del rendimiento de los modelos neuronales, se compara un modelo LSTM con un modelo ARIMA, ambos con un horizonte de predicción de 3 horas.
Es difícil realizar una comparativa justa entre modelos ARIMA y de aprendizaje profundo, debido a que tienen naturalezas distintas. Los ARIMA toman una serie temporal unidimensional para su entrenamiento, idealmente con un número elevado de observaciones (normalmente, más de 500), y generan una predicción de su continuación. Mientras que, con los modelos de aprendizaje profundo, buscamos crear un modelo capaz de predecir una serie que no ha usado en su entrenamiento, a partir de una muestra pequeña, de menos de un día.
Debido a estas limitaciones, los ARIMA se suelen evaluar sobre el periodo posterior al entrenamiento, lo que contrata con el enfoque de validación que habíamos implementado en los modelos neuronales, usando un muestreo aleatorio de las ventanas en todo el período contemplado. Para comparar ambos modelos, se opta por emplear el mes de marzo de 2025, siendo ambos modelos entrenados con el intervalo de marzo de 2023 a febrero de 2025.
Los resultados se muestran en la tabla 3.1.
TODO: Conclusiones.
A continuación, se comparan los modelos LSTM, CNN y el híbrido. En este caso, emplearemos el RMSE obtenido en la validación cruzada del

		Temperatura		Presión		Humedad	
Ubicación	Fuente	ARIMA	LSTM	ARIMA	LSTM	ARIMA	LSTM
Arona	Open-Meteo	1,088	0,630	0	0	0	0
	Grafcan	1,263	0,697	0	0	0	0
La Orotava	Open-Meteo	0,917	0,496	0	0	0	0
	Grafcan	1,344	0,832	0	0	0	0
La Laguna 1	Open-Meteo	0,806	0,385	0	0	0	0
	Grafcan	0,902	0,699	0	0	0	0
La Laguna 2	Open-Meteo	1,157	0,497	0	0	0	0
	Grafcan	1,049	0,621	0	0	0	0
Santa Cruz	Open-Meteo	0,806	0,404	0	0	0	0
	Grafcan	1,335	0,832	0	0	0	0
Garachico	Open-Meteo	0,850	0,612	0	0	0	0
	Grafcan	1,434	0,991	0	0	0	0

Tabla 3.1: Comparativa inicial de modelos ARIMA y LSTM con predicciones de 3 horas

entrenamiento.

3.4. Estudios empíricos

Se realizan diferentes experimentos sobre los modelos neuronales de cara a obtener el mejor rendimiento posible.

3.4.1. Características de los modelos

Se prueba el uso de diferentes elementos para evitar el sobreajuste y mejorar la generalización de los modelos, como las capas de abandono (dropout) y la normalización por lotes (batch normalization) o la normalización de capas (layer normalization). También se emplea el abandono recurrente, que consiste en aplicar el abandono a las conexiones recurrentes de la red LSTM, lo que ayuda a prevenir el sobreajuste al forzar a la red a aprender representaciones más robustas y generalizables. Sin embargo, se observa que ninguna de estas técnicas mejora el rendimiento de los modelos, por lo que se opta por no incluirlas en las arquitecturas finales.

3.4.2. Número de estaciones

Se estudia el impacto del número de estaciones en la precisión de los modelos. Se entrena un modelo LSTM con diferente número de estaciones. Los resultados se muestran en la tabla XXX. Para evaluarlo, se mide el desempeño frente a las estaciones de test. Observamos que el rendimiento mejora a medida que se añaden más estaciones.

3.4.3. Tamaño de la ventana

Se estudia exhaustivamente el número de horas previas P que contiene la ventana. Para ello, se construyen datasets desde P=6 hasta P=48 y se evalúan en los modelos desarrollados. Se observan distintos valores óptimos de P según las variables: P=17 para la temperatura del aire, P=24 para la humedad relativa y P=?? para la presión atmosférica.

3.4.4. Uso de ruido

Se plantea la hipótesis de que añadir un cierto ruido a los datos a medida que son consumidos en el entrenamiento puede favorecer la precisión del modelo al evitar el sobreajuste. Esto es, se desea que el modelo en cada época no vea la misma muestra, sino que vea una versión ligeramente distinta.
Para ello, se emplea el método .map del dataset de TensorFlow, que permite mapear otra función para su ejecución perezosa. Se utiliza la función descrita en 3.11 sobre el conjunto de entrenamiento. En dicha función se aplica el ruido mediante una máscara a los datos pasados de la variable y las covariables, sin alterar la codificación temporal. Además, se aplica ruido al valor objetivo y.

Pseudocódigo para agregar ruido

Funciónadd_noise(x, y, $covariates\_indexes$):
    Desempaquetar x en $past$, fut
    # Generar ruido sobre las covariables
    cov_noise = noise_distribution($past$, NOISE_STD)
    # Construir máscara de características afectadas
    $n\_features$ = shape($past$)[-1]
    $mask$ = ceros($(n\_features)$)
    $mask$[$covariates\_indexes$] = 1
    $mask$ = reshape($mask$, $(1, 1, n\_feat)$) # redimensión para aplicar al lote
    # Aplicar ruido sólo en características seleccionadas
    $past\_noise$ = $cov\_noise \times mask$
    # Generar ruido sobre la variable objetivo.
    $target\_noise$ = noise_distribution(y, NOISE_STD)
    # Retornar tupla con datos ruidosos y futuro sin modificar
    Retornar ($(past + past\_noise)$, fut), $y + target\_noise$

Figura 3.11: Pseudocódigo de la función `add_noise`

Se prueban distintos valores de ruido y varias distribuciones: normal y una triangular, con menor desviación, construida mediante dos distribuciones uniformes. Ejemplos de los datos con el ruido aplicado se pueden ver en las figuras 3.12 y 3.13. También se experimenta aplicando coeficientes de ruido distintos a las covariables y a la variable objetivo. Se observa que un ruido menor al de las covariables en la variable objetivo mejora los resultados frente a usar el mismo valor.

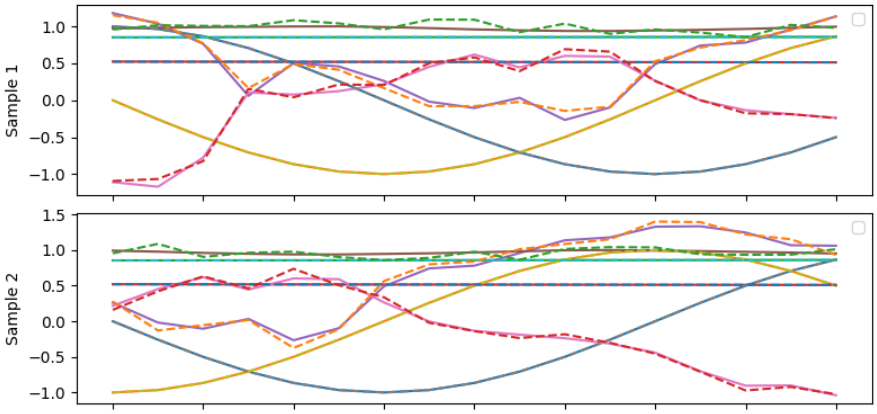

Figura 3.12: Datos pasados de 2 muestras. Datos originales en continua y con el ruido en discontinua.

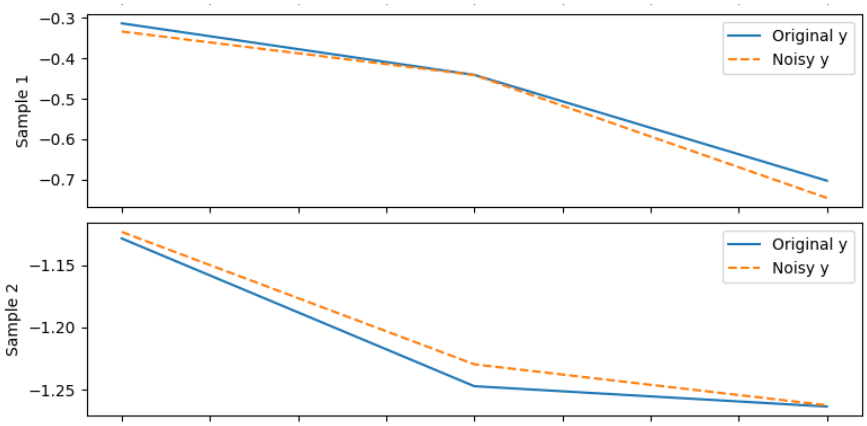

Figura 3.13: Valores objetivo de 2 muestras. Datos originales en continua y con el ruido en discontinua.

Sin embargo, en la práctica se constata que el ruido no mejora los resultados de los modelos. Esto puede deberse a que las series exhiben una tendencia muy fuerte, que la inclusión del ruido no es capaz de modificar, puesto que sería necesario un ruido tan elevado que se perdería el valor de los datos.

3.5. Modelos y Resultados

3.5.1. Temperatura del aire

3.5.2. Humedad relativa

3.5.3. Presión atmosférica

Capítulo 4

Despliegue

Los capítulos intermedios servirán para cubrir los siguientes aspectos: antecedentes, problemática o estado del arte, objetivos, fases y desarrollo del proyecto.

El capítulo 1 se describió bla, bla, bla . . .

Capítulo 5

Conclusiones y líneas futuras

Este capítulo es obligatorio. Toda memoria de Trabajo de Fin de Grado debe incluir unas conclusiones y unas líneas de trabajo futuro
Estudiando el tamaño de la ventana de datos pasados, se oberva que empleando pocas mediciones, menos de un día, se obtienen mejores predicciones que con períodos más extensos.
El conjunto de tratamiento de los datos y selección de estructuras consigue unos modelos que mejoran significativamente los resultados obtenidos por los desarrollos comparables realizados hasta el momento.

Capítulo 6

Summary and Conclusions

This chapter is compulsory. The memory should include an extended summary and conclusions in english.

Capítulo 7

Presupuesto

Este capítulo es obligatorio. Toda memoria de Trabajo de Fin de Grado debe incluir un presupuesto.

7.1. Sección Uno

Tipos	Descripción
AAAA	BBBB
CCCC	DDDD
EEEE	FFFF
GGGG	HHHH

Tabla 7.1: Resumen de tipos

Apéndice A

Título del Apéndice 1

A.1. Algoritmo XXX

```
/*****************************************************************************
*
* Fichero .h
*
*****************************************************************************
*
* AUTORES
*
*
* FECHA
*
*
* DESCRIPCION
*
*
*****************************************************************************
```

A.2. Algoritmo YYY

```
/*****************************************************************************
 *
 * Fichero .h
 *
 *****************************************************************************
 *
 * AUTORES
 *
 * FECHA
 *
 * DESCRIPCION
 *
 *
 *****************************************************************************
```

A.3. Algoritmo ZZZ

```
/*****************************************************************************
 *
 * Fichero .h
 *
 *****************************************************************************
 *
 * AUTORES
 *
 * FECHA
 *
 * DESCRIPCION
 *
 *
 *****************************************************************************
```

Apéndice B

Título del Apéndice 2

B.1. Otro apéndice: Sección 1

texto

B.2. Otro apéndice: Sección 2

texto

Bibliografía

[1] McCulloch, W. S., & Pitts, W. (1943). *A logical calculus of the ideas immanent in nervous activity*. Bulletin of Mathematical Biophysics, 5(4), 115–133. https://doi.org/10.1007/BF02478259

[2] Hebb, D. O. (1949). *The organization of behavior: A neuropsychological theory*. Wiley.

[3] Rosenblatt, F. (1958). *The perceptron: A probabilistic model for information storage and organization in the brain*. Psychological Review, 65(6), 386–408. https://doi.org/10.1037/h0042519

[4] Werbos, P. J. (1982). *Applications of advances in nonlinear sensitivity analysis*. R. F. Drenick & F. Kozin (Eds.), System Modeling and Optimization (Vol. 38, pp. 762–770). Springer-Verlag. https://doi.org/10.1007/BFb0006203

[5] Elman, J. L. (1990). *Finding structure in time*. Cognitive Science

[6] Hochreiter, S.; Schmidhuber, J. (1997). *Long Short-Term Memory*. Neural Computation

[7] Cho, K., Van Merriënboer, B., Gulcehre, C., et al. (2014). *Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation*. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (pp. 1724–1734)

[8] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gómez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). *Attention Is All You Need*. Advances in Neural Information Processing Systems, 30, 5998–6008

[9] Shi, X., Chen, Z., Wang, H., Yeung, D.-Y., Wong, W.-K., & Woo, W.-c. (2015). *Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting*. Advances in Neural Information Processing Systems, 28, 802–810

[10] Weyn, J. A., Durran, D. R., & Caruana, R. (2020). *Improving Data-Driven Global Weather Prediction Using Deep Convolutional Neural Networks on a Cubed Sphere*. Journal of Advances in Modeling Earth Systems, 12(9)

[11] Fu, Y., Wang, F., Shao, Z., Yu, C., Li, Y., Chen, Z., An, Z., & Xu, Y. (2024). *LightWeather: Harnessing Absolute Positional Encoding to Efficient and Scalable Global Weather Forecasting*. arXiv:2408.09695.

[12] Deznabi, I., Kumar, P., & Fiterau, M. (2024). *Zero-shot Microclimate Prediction with Deep Learning*. arXiv:2401.02665

[13] Cartográfica de Canarias, S.A. *Sistema de Observación Meteorológica de Canarias* [Sitio web]. Recuperado el 12 de mayo de 2025, de https://sensores.grafcan.es/

[14] Open-Meteo *Historical Forecast API* [Sitio web]. Recuperado el 12 de mayo de 2025, de https://open-meteo.com/en/docs/historical-forecast-api

[15] Fritsch, F. N., & Carlson, R. E. (1980). *Monotone piecewise cubic interpolation*. SIAM Journal on Numerical Analysis, 17(2), 238–246. https://doi.org/10.1137/0717021

[16] Tawakuli, A., Havers-Zulka, B., Gulisano, V., & Kaiser, D. (2024). *Survey: Time-series data preprocessing: A survey and an empirical analysis*. Journal of Engineering Research. Advance online publication. https://doi.org/10.1016/j.jer.2024.02.018

[17] Gu, X., Akoglu, L., & Rinaldo, A. (2019). *Statistical analysis of nearest neighbor methods for anomaly detection*. Advances in Neural Information Processing Systems, 32 (pp. 10921–10931). Curran Associates, Inc. https://doi.org/10.48550/arXiv.1907.03813

[18] Chapman, S., & Lindzen, R. S. (1970). *Atmospheric Tides*. D. Reidel Publishing Company. https://doi.org/10.1007/978-94-010-3399-2

[19] Graves, A. and Schmidhuber, J. (2005). *Framewise phoneme classification with bidirectional LSTM networks*. Neural Networks, 18(5-6), 602–610. https://doi.org/10.1016/j.neunet.2005.06.042