Universidad de **Valladolid**

Facultad de Ciencias

TRABAJO FIN DE GRADO

Grado en Estadística

Comparativa de modelos en Python para la predicción de una serie temporal diaria de temperatura

Autor: Pablo Marcos Parra
Tutora: María del Pilar Rodríguez del Tío
Año 2024

A todas las personas que han creído en mí y me han apoyado, en especial a mis padres.

"Todos nosotros somos capaces de los cambios más increíbles. Podemos evolucionar mientras permanezcamos fieles a quienes somos. Podemos honrar a quien hemos sido y escoger lo que queremos ser."

- Doctor Who

Agradecimientos

A todos los que me han acompañado todos estos años, en especial a mi madre que siempre me cuida. Y a mi padre que donde quiera que esté también lo estará haciendo.

Muchas gracias a mi tutora Pilar por toda la paciencia y las revisiones del trabajo que parecía que no se iba a acabar nunca.

A mis amigos, compañeros de carrera y compañeros de la oficina, porque sin ellos no habría llegado hasta aquí.

Resumen

En este trabajo fin de grado se estudiará el funcionamiento de varios modelos implementados en el lenguaje de programación Python para la predicción de una serie temporal diaria con único periodo.

Se han planteado un total de 7 modelos para la comparación, de los cuales 2 son modelos estadísticos clásicos (modelos metodología Box-Jenkins y modelos de suavizado exponencial) y los otros 5 modelos se basan en redes neuronales recurrentes y algunas variantes.

Palabras clave: Series Temporales, Python, LSTM, modelos Box-Jenkins, Redes Neuronales Recurrentes, GRU, modelos de suavizado exponencial.

Abstract

In this final degree thesis we will study the performance of several models implemented in the Python programming language for the prediction of a daily time series with a single period.

A total of 7 models have been proposed for comparison, of which 2 are classical statistical models (Box-Jenkins methodology models and exponential smoothing models) and the other 5 models are based on recurrent neural networks and some variants.

Keywords: Time Series, Python, LSTM, Transformer Architecture, Box-Jenkins models, Recurrent Neural Network, GRU, Exponential Smoothing Models.

Índice general

Índice de cuadros

Índice de figuras

Capítulo 1

Introducción

En la actualidad, la predicción de series temporales, el análisis y modelado de datos secuenciales a lo largo del tiempo, es una técnica esencial en numerosos campos, como las finanzas, la meteorología, la salud pública y la gestión de inventarios, entre otros. A medida que la cantidad y complejidad de los datos han aumentado, también lo han hecho los métodos y modelos utilizados para analizar estos datos y hacer mejores predicciones.

Los modelos estadísticos ARIMA (*Autoregressive Integrated Moving Average*) y su variante estacional SARIMA (*Stational Autoregressive Integrated Moving Average*) han sido los modelos más utilizados para la predicción de series temporales. Sus implementaciones tradicionalmente han sido realizadas en lenguajes de programación con orientación a datos como R y SAS.

Sin embargo, en los últimos años la capacidad computacional ha ido aumentando, lo que ha permitido dar un empuje a los métodos de suavizado exponencial e introducir nuevos modelos como los de redes neuronales y de aprendizaje profundo que han mostrado ser técnicas efectivas para esta tarea. El lenguaje de programación Python, ha emergido al mismo tiempo como uno de los lenguajes de programación más populares y versátiles, facilitando enormemente el desarrollo e implementación de estos modelos gracias a la disponibilidad de una extensa colección de librerías y herramientas específicas para el análisis de datos y el aprendizaje automático.

Este trabajo tiene como objetivo realizar una comparación de varios modelos usando Python para la predicción de series temporales univariantes. Los modelos considerados en este estudio incluyen el modelo SARIMA, el modelo de suavizado de Holt-Winters, redes neuronales recurrentes simples (RNN), redes LSTM (*Long Short-Term Memory*) tanto básicas como bidireccionales, redes GRU (*Gated Recurrent Unit*) y una red con una combinación de capas convolucionales con capas LSTM simples. Para ello, se evaluará la precisión y eficiencia de estos modelos para obtener predicciones trabajando con un conjunto de datos diarios de temperatura en la ciudad de Valladolid entre los años 1974 y 2024.

Capítulo 2

Contexto

2.1 Metodología Box-Jenkins

La metodología Box-Jenkins es un enfoque probabilístico para poder identificar, estimar y validar modelos de series temporales con el fin de realizar predicciones sobre los datos. Este método fue desarrollado por George Box y Gwylim Jenkins y se detalló en su libro "Time Series Analysis: Forecasting and Control"[1].

En este trabajo se usarán modelos que provienen de la metodología citada, en concreto modelos estacionales o SARIMA, casos particulares de los modelos ARIMA que modelizan series estacionales.

2.1.1 Modelo SARIMA *(p,d,q)(P,D,Q)[s]*

Los modelos SARIMA (*Seasonal AutoRegressive Integrated Moving Average*) son modelos ARIMA de órdenes altos con restricciones los coeficientes que permiten modelizar la estacionalidad. Su denotación es SARIMA *(p,d,q)(P,D,Q)[s]* donde *p, d, q* representan los órdenes de los polinomios asociados a los retardos de la parte regular; *P, D, Q* los valores de la parte estacional; y *s* es el periodo de la estacionalidad. Estos valores se describirán a continuación.

Los tres componentes de SARIMA son:

- Componente Autorregresivo (AR): refleja que el valor de una serie dado un momento Y_t está influenciado por los valores pasados. En SARIMA la componente se divide en dos:
  - Componente AR No Estacional (p): este componente tiene en cuenta los valores pasados de la serie a intervalos regulares. Su ecuación es [2]:

$$X_t = \phi_1 X_{t-1} + \phi_2 X_{t-2} + ... + \phi_p X_{t-p} + a_t \tag{2.1}$$

    Donde $\phi_1, \phi_2, ..., \phi_P$ son los coeficientes AR.

- Componente AR Estacional (P): este componente incorpora los valores pasados de la serie con un desfase de *s* unidades de tiempo (por ejemplo, doce meses en series con datos mensuales). La ecuación se expresa como [3]:

$$X_t = \Phi_1 X_{t-s} + \Phi_2 X_{t-2s} + ... + \phi_P X_{t-P_s} + a_t \tag{2.2}$$

  Donde $\Phi_1, \Phi_2, ..., \Phi_P$ son los coeficientes estacionales AR.

- Componente Integrado (I): se refiere a la diferenciación de la serie temporal necesaria para conseguir que la serie sea estacionaria.

  - Diferenciación Regular (d): es el número de veces que se diferencia la serie para eliminar las tendencias y hacerla estacionaria. [2]

$$X'_t = X_t - X_{t-1} \tag{2.3}$$

  - Diferenciación Estacional (D): es el número de veces que se diferencia la serie para eliminar los patrones estacionales y alcanzar la estacionaridad por la parte estacional [3]

$$X"_t = X_t - X_{t-s} \tag{2.4}$$

- Componente de Media Móvil (MA):modela el error como una combinación lineal de errores pasados.

  - Componente MA No Estacional (q): se expresa como [2]:

$$a_t = \theta_1 a_{t-1} + \theta_2 a_{t-2} + ... + \theta_q a_{t-q} \tag{2.5}$$

    Donde $\theta_1, \theta_2, ..., \theta_q$ son los coeficientes MA.

  - Componente MA Estacional (Q): incorpora los errores pasados con un desfase de *s* periodos siguiendo la fórmula [3]:

$$a_t = \Theta_1 a_{t-s} + \Theta_2 a_{t-2s} + ... + \Theta_Q a_{t-Q_s} \tag{2.6}$$

    Donde $\Theta_1, \Theta_2, ..., \Theta_Q$ son los coeficientes MA Estacionales.

La fórmula general del modelo SARIMA es [4]:

$$\begin{aligned}(1 - \phi_1 B - \phi_2 B^2 - \ldots - \phi_p B^p)(1 - \Phi_1 B^s - \Phi_2 B^{2s} - \ldots - \Phi_P B^{Ps})(1 - B)^d (1 - B^s)^D X_t \\ = (1 - \theta_1 B - \theta_2 B^2 - \ldots - \theta_q B^q)(1 - \Theta_1 B^s - \Theta_2 B^{2s} - \ldots - \Theta_Q B^{Qs}) a_t\end{aligned} \tag{2.7}$$

donde B es el operador de retardo y θ, ϕ, Θ y Φ han sido descritos anteriormente.

2.1.2 Pasos de la metodología Box-Jenkins

1. **Identificación del modelo:** se usan los datos y cualquier información sobre cómo se han generado para proponer una serie de modelos que podrían ser adecuados, identificando posibles valores para *p, d, q, P, D* y *Q*. Se utilizan los siguientes métodos:

   - Gráficos de Autocorrelación (ACF) y Autocorrelación Parcial (PACF): son gráficos que se usan para identificar patrones en los datos y determinar los órdenes de *p, q* y de *P,Q*. El gráfico de autocorrelación muestra cómo los datos en una serie están correlacionados con sus propios valores para diferentes retardos, mientras que el gráfico de autocorrelación parcial muestra estas correlaciones eliminando la influencia de observaciones en retardos intermedios. [2]
   - Pruebas de estacionaridad: se aplican pruebas como la prueba de Dickey-Fuller aumentada (ADF) para evaluar si la serie es estacionaria o si por otra parte, requiere diferenciación (valores de *d* y *D*). La observación de esquemas en la ACF y PACF de las series y sus diferenciadas también ayuda a elegir dichos valores de *d* y *D*

2. **Estimación de parámetros:** una vez identificado el modelo, se utilizan técnicas de estimación para obtener los valores de los parámetros del modelo.

   En el caso de los algoritmos aplicados y usados, descritos en 3.2.2, se ha utilizado una estimación numérica basada en el algoritmo BFGS (*Broyden-Fletcher-Goldfarb-Shanno*) que es uno de los métodos cuasi-Newton más utilizados para la optimización de funciones no lineales, como es el caso de la verosimilitud del modelo. Se emplea en problemas de optimización sin restricciones y destaca por la rapidez en la convergencia hacia un óptimo local. A nivel de memoria del ordenador, este algoritmo requiere una cantidad ingente, por lo que en el caso de este trabajo se ha utilizado la variante L-BFGS (*Low-memory BFGS*)[5][6].

3. **Validación:** en este paso se revisan los residuos del modelo ajustado para asegurar que son aleatorios y comprobar si su distribución puede suponerse normal. Esto se hace mediante pruebas de independencia, homocedasticidad en los residuos y contrastes de bondad de ajuste tipo Shapiro-Wilk. Si los residuos no cumplen estos requisitos, no se consideran aleatorios y el modelo debe ser ajustado de nuevo [4].

4. **Predicción:** una vez validado el modelo, se pueden realizar predicciones de pasos futuros. Para estos pronósticos se considera tanto la parte regular como la parte estacional.

   Utilizando el modelo, se generan las predicciones definidas como esperanzas condicionadas y se calculan los intervalos probabilísticos para las observaciones futuras a partir de ellas. La fórmula general para generar las predicciones se obtiene tomando esperanzas condicionadas por las observaciones conocidas en la expresión (2.7) [3]:

   $$\phi_p(B)\Phi_P(B^s)(1-B)^d(1-B^s)^D X_n(h) = 0 \tag{2.8}$$

   Donde *n* es el número de observaciones utilizadas para el ajuste y *h* es el horizonte de la predicción. Se hace notar que los parámetros de la parte de la media móvil no influyen a medio y largo plazo en las predicciones. A largo plazo las predicciones cumplirán la ecuación.

2.2 Modelos de suavizado

Los modelos de suavizado exponencial son métodos ampliamente utilizados para el análisis y previsión de series temporales debido a su simplicidad y eficacia. Estos métodos aplican un promedio ponderado de las observaciones pasadas donde los pesos decrecen exponencialmente al alejarse las observaciones en el tiempo [7][8]. Algunos ejemplos son:

- **Suavizado Exponencial Simple**: es un modelo adecuado para series sin tendencia ni estacionalidad.

$$\hat{y}_{t+1} = \alpha y_t + (1-\alpha)\hat{y}_t \tag{2.9}$$

  Donde $\hat{y}_{t+1}$ es la predicción del siguiente periodo, y_t es la observación actual, $\hat{y}_t$ es la previsión del periodo actual y α es el parámetro de suavizado (0<α<1).

- **Suavizado Exponencial Doble**: introduce un componente de tendencia, adecuado para series con tendencia pero sin estacionalidad. Utiliza las ecuaciones de nivel y tendencia siguientes [8]:

$$l_t = \alpha y_t + (1-\alpha)(l_{t-1} + b_{t-1}) \tag{2.10}$$

$$b_t = \beta(l_t - l_{t-1}) + (1-\beta)b_{t-1} \tag{2.11}$$

  Con l_t siendo el nivel, b_t es la tendencia y α y β son los parámetros de suavizado

- **Suavizado Exponencial de Winter estacional**: también conocido como modelo Holt-Winters, es un modelo adecuado para series con tendencia y estacionalidad.

2.2.1 Modelo de Winters Estacional

El modelo Winters Estacional, es un modelo que extiende el suavizado exponencial doble incorporando la componente estacional. Es muy útil con series con variaciones estacionales regulares y puede ser aplicada en su forma aditiva o multiplicativa [7].

- **Forma aditiva**: utilizada cuando la estacionalidad es constante a lo largo del tiempo.

$$l_t = \alpha(X_t - s_{t-m}) + (1-\alpha)(l_{t-1} + b_{t-1}) \tag{2.12}$$

$$b_t = \beta(l_t - l_{t-1}) + (1-\beta)b_{t-1} \tag{2.13}$$

$$s_t = \gamma(X_t - l_t) + (1-\gamma)s_{t-m} \tag{2.14}$$

$$\hat{X}_{t+h} = l_t + hb_t + s_{t+h-m(k+1)} \tag{2.15}$$

- **Forma multiplicativa**: usada cuando la estacionalidad varía proporcionalmente con el nivel de la serie. Las ecuaciones son similares a las de la forma aditiva, pero el componente estacional se modela multiplicativamente.

2.2.2 Damping

El damping es una técnica introducida para modificar la tendencia a largo plazo en los modelos de suavizado exponencial. Fue propuesta por Gardner y McKenzie [9] y reduce la influencia de la tendencia conforme el horizonte de previsión aumenta, lo que produce previsiones optimistas o pesimistas.

Para el modelo Winters Aditivo, se introduce un factor de damping ϕ (0<ϕ<1) en las ecuaciones de nivel y tendencia para amortiguar la tendencia, asegurando que en las predicciones a largo plazo la suposición de pendiente constante no lleve a errores significativos.

$$l_t = \alpha(X_t - s_{t-m}) + (1-\alpha)(l_{t-1} + \phi b_{t-1}) \tag{2.16}$$

$$b_t = \beta(l_t - l_{t-1}) + (1-\beta)\phi b_{t-1} \tag{2.17}$$

$$s_t = \gamma(X_t - l_t) + (1-\gamma)s_{t-m} \tag{2.18}$$

$$\hat{X}_{t+h} = l_t + \phi^h b_t + s_{t+h-m(k+1)} \tag{2.19}$$

2.3 Redes neuronales recurrentes

Las redes neuronales recurrentes (RNN, en inglés) son una clase de redes neuronales diseñadas específicamente para el procesamiento de secuencias de datos, lo que hace que sean ideales para el procesamiento y predicción de series temporales. Las RNN tienen conexiones recurrentes que permiten mantener en memoria interna los estados anteriores.

2.3.1 Arquitectura Redes Neuronales Recurrentes (simples)

La arquitectura básica de una RNN incluye una capa de entrada, una o más capas ocultas con conexiones recurrentes y una capa de salida. En cada paso temporal, la red recibe una secuencia de entrada y actualiza su estado interno (también conocido como estado oculto).

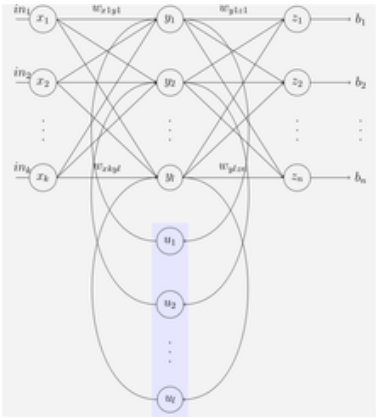

Figura 2.1: Arquitectura Elman (*Simple Recurrent Network*) [10]

La red Elman, también conocida como *"Simple Recurrent Networks"* (SRN) es una red de tres capas (ordenadas horizontalmente como x, y, z como en la figura) con el añadido de un set de

unidades de contexto. La capa central está conectada a esas unidades de contexto fijadas con un peso de 1 [11].

A cada paso que da la red, se introduce la entrada y la regla de aprendizaje aplicada, las conexiones guardan una copia del valor previo de las unidades ocultas en las unidades de contexto. Así la red mantiene una especie de estado, permitiendo la tarea de predicción de secuencias. Las fórmulas son[12]:

$$h_t = \sigma_h(W_h x_t + U_h h_{t-1} + b_h) \tag{2.20}$$

$$y_t = \sigma_y(W_y h_t + b_y) \tag{2.21}$$

Existen varios tipos de RNN que se clasifican según la entrada y salida que se obtienen [13]:

- One-to-many: arquitectura que recibe un único dato de entrada y como salida da una secuencia de datos. Un ejemplo de uso de este tipo es la descripción de imágenes.
- Many-to-one: arquitectura que recibe una secuencia como dato de entrada y como salida un único valor.
- Many-to-many: arquitectura que recibe una secuencia como dato de entrada y como salida otra secuencia de datos.

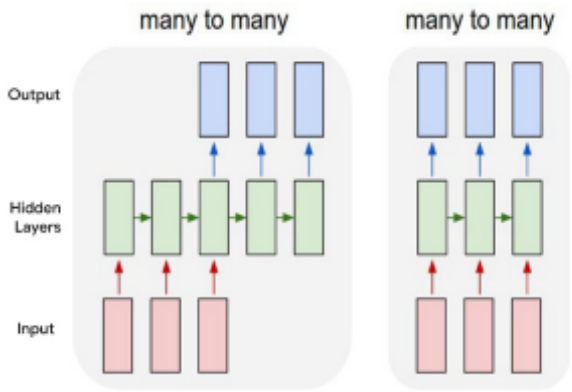

Figura 2.2: Tipo many-to-many [14]

En el caso de las series temporales, se utilizan los tipos many-to-many (predicción de varios pasos futuros) y many-to-one (predicción del siguiente paso).

2.3.2 Arquitectura LSTM (Long-Short Term Memory) básica

Las redes LSTM (*Long Short Term Memory*, en inglés) son una variación de la arquitectura anteriormente descrita y que arregla uno de los mayores problemas que tienen las SRN, que es el manejo de dependencias en los datos a largo plazo en la secuencia. Este problema es comúnmente conocido como problema del desvanecimiento del gradiente [15] y su solución fue mostrada en el paper "Long Short-term Memory"[16] de Sepp Hochreiter y Jürgen Schmidhuber en el año 1997.

La solución consistía en añadir a la arquitectura RNN una unidad de memoria llamada "celda LSTM"que es la parte de la lógica que se encarga de seleccionar qué datos recordar o qué datos olvidar en base a los datos que se reciben en la entrada. La celda se compone de tres compuertas:

- Input gate: puerta que decide qué información agregar a la celda de memoria.
- Forget gate: puerta que decide la información que se va a olvidar y no va a acceder a la celda.
- Output gate: preserva el estado oculto actual y envía dicho estado al siguiente estado oculto.

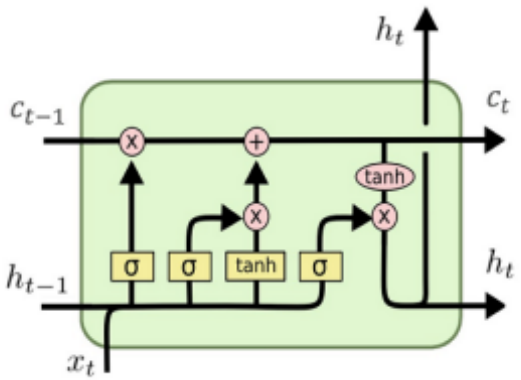

Figura 2.3: Arquitectura LSTM [17]

2.3.3 Arquitectura LSTM-Bidireccional

Para mejorar el modelo LSTM básico, en el año 1997 Schuster y Paliwal [18] introdujeron el concepto de las redes bidireccionales, unas redes que permiten incrementar la cantidad de información disponible en el entrenamiento de la red, al conectar dos capas ocultas con datos en direcciones opuestas. Por ejemplo, la secuencia [1,2,3] se pasaba dos veces, la primera como [1,2,3] y la segunda en dirección contraria. Esta arquitectura es especialmente útil cuando el contexto de la entrada es necesario.

Esta conexión bidireccional permite que la información pasada y futura de la ventana de tiempo actual pueda usarse, a diferencia de las redes básicas que requieren la información futura.

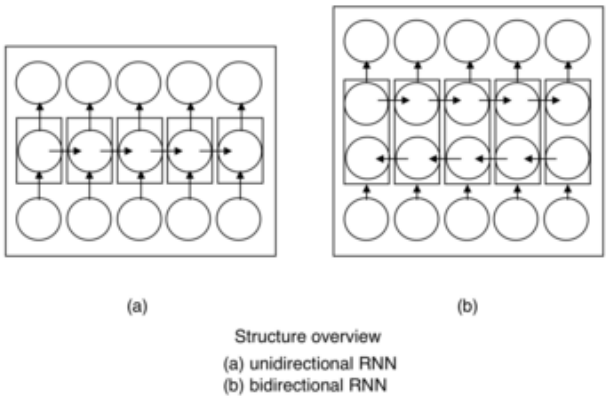

Figura 2.4: Estructura de RNN vs Bidirectional RNN [19]

2.3.4 Arquitectura GRU

Las redes GRU (*Gated Recurrent Units*) son una arquitectura que se introdujo en el año 2014 por Kyunghyun Cho et al. [20]. Las redes GRU son parecidas a las redes LSTM, con la capacidad de recordar u olvidar ciertas características de la serie.

La gran diferencia de esta red con respecta a la red LSTM, es la ausencia de una puerta del olvido o *forget gate*, lo que hace que el modelo tenga menos parámetros y sea más pequeño y eficiente.

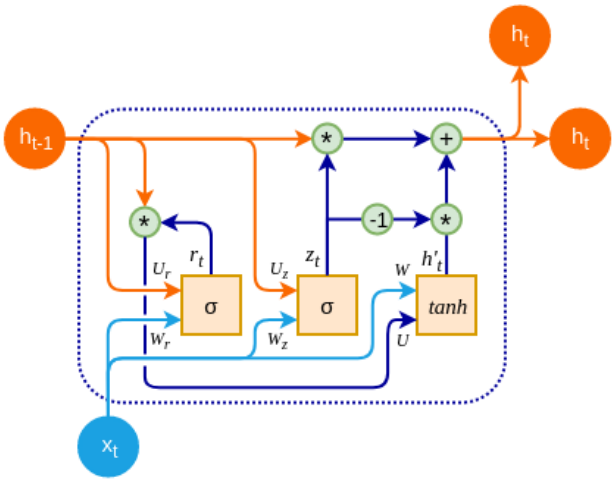

Figura 2.5: Arquitectura GRU [21]

2.3.5 Arquitectura CNN (Convolutional Neural Network) con LSTM

2.3.5.1. Arquitectura CNN

Las Redes Neuronales Convolucionales (CNN, por sus siglas en inglés) son un tipo de red neuronal diseñada para procesar datos en una estructura de rejilla (como en el caso de imágenes) [22].

Las CNN se basan en la operación matemática de la convolución que permite a la red extraer características locales de los datos de entrada.

Las capas que se han utilizado en el trabajo han sido:

- Capa convolucional: la capa se compone de filtros o kernels, que son pequeñas matrices de pesos que se van deslizando por la entrada para calcular un mapa de características. Cada filtro aprende una característica específica, como por ejemplo patrones locales de los datos. Se aplica el filtro a la entrada en lo que se conoce como convolución.
- Capa de pooling: reduce la dimensionalidad del mapa de características resumiendo las regiones de datos adyacentes en una sola. En el caso de este trabajo se ha utilizado el max pooling, en el cual se toma el valor máximo de una región de los datos.

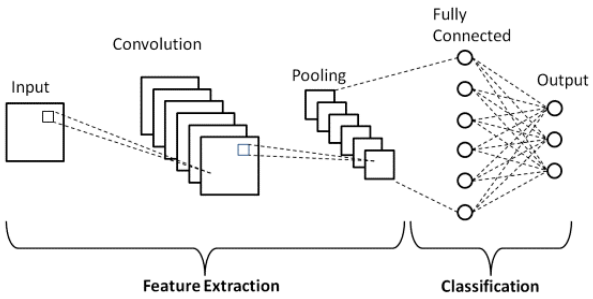

Figura 2.6: Arquitectura CNN completa [23]

2.3.5.2. CNN+LSTM para series temporales

Las redes CNN+LSTM son redes que combinan los dos tipos de redes neuronales que se han presentado anteriormente, CNN y LSTM, el uso de esta combinación está muy extendido para las tareas relacionadas con series temporales [24].

Las redes CNN son eficaces en la extracción de características locales y patrones espaciales de los datos, lo que capta las dependencias a corto plazo en las series temporales. Las redes LSTM en cambio se encargan de capturar patrones y tendencias a largo plazo. Esta combinación permite crear modelo más robustos y precisos.

La arquitectura típica es una capa de entrada (secuencia de datos de la serie temporal), las capas CNN, seguidas de las capas LSTM y finalmente la capa de salida con los resultados de la predicción de la serie temporal.

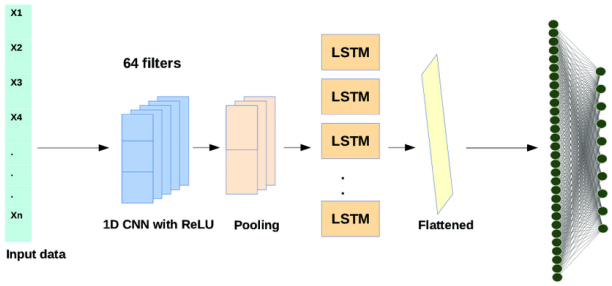

Figura 2.7: Arquitectura CNN+LSTM básica [25]

Capítulo 3

Datos y librerías usadas

3.1 Datos

Para este Trabajo de Fin de Grado, se ha utilizado un conjunto de datos que incluye registros diarios de temperatura media en Valladolid desde el 1 de enero de 1974 al 31 de marzo de 2024. Estos datos han sido obtenidos mediante *scraping web* de la web MeteoClima [26], según se especifica en el apéndice D

A continuación se detalla la estructura del conjunto de datos:

- Número de registros: 18,353
- Número de registros vacíos: 0
- Número de columnas: 2

Columnas:

- *mean_temp*: temperatura media.
- *date*: fecha (formato año-mes-día).

	date	mean_temp
0	1974-01-01	3.20
1	1974-01-02	4.10
2	1974-01-03	4.45
3	1974-01-04	4.50
4	1974-01-05	6.45

Cuadro 3.1: Valores de ejemplo del conjunto de datos

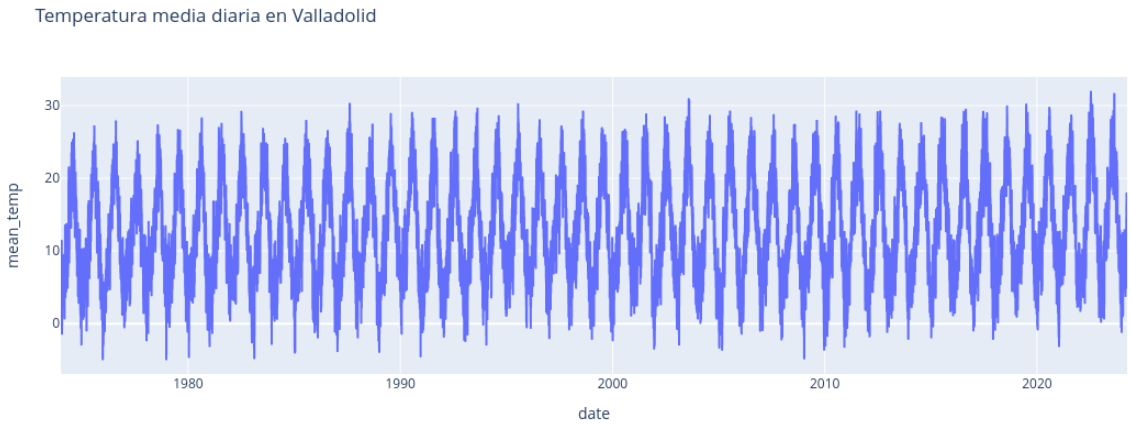

Figura 3.1: Serie temporal de la temperatura media de Valladolid (1974 a 2024)

3.2 Librerías en Python

A continuación, se detallarán algunas herramientas y librerías.

3.2.1 Pandas

Para el manejo del conjunto de datos, incluyendo su carga y manipulación, se ha usado la librería Pandas [27].

Pandas es una librería de código abierto creado por Wes McKinney para Python para poder trabajar de forma sencilla con estructuras de datos variadas [28]. Entre las operaciones implementadas en la librería Pandas se han utilizado en este trabajo la carga de datos desde un archivo en formato CSV (Comma-Separated Values), su filtrado y selección, así como el preprocesamiento y limpieza de los datos.

3.2.2 Statsmodels

Para la implementación del modelo SARIMA y el modelo de suavizado exponencial, se ha utilizado la librería Statsmodels [29] que es una librería de Python de código abierto que contiene varias utilidades y funciones para trabajar y estimar con diferentes modelos estadísticos, así como realizar tests estadísticos y exploración de datos. Está pensado para desarrolladores y usuarios de todas las disciplinas con experiencia previa en Python.

Para este trabajo, se ha implementado la utilidad `tsa.statespace.sarimax.SARIMAX` para el modelo SARIMA y la utilidad `tsa.holtwinters.ExponentialSmoothing` para el modelo Holt-Winters. Ambas se describirán en mayor profundidad en el capítulo 4.

Versión usada (1 de julio de 2024): `statsmodels=0.14.2`

3.2.3 Keras

Keras [30] es una librería de Python de código abierto que proporciona a los usuarios una interfaz de alto nivel para la creación y entrenamiento de redes neuronales artificiales. Su principal beneficio es la sencillez y facilidad de uso. [31]

Keras contiene numerosas implementaciones de utilidades y funciones usadas comunmente en la construcción y validación de redes neuronales, incluyendo tipos de capas, funciones de activación y optimizadores.

Inicialmente una librería independiente, actualmente puede ser integrada con otras librerías nativas como JAX, TensorFlow y PyTorch. Para este trabajo, se ha utilizado como backend, para la implementación a bajo nivel de las redes neuronales y sus respectivos cálculos, TensorFlow [32].

Versión utilizada (1 de julio de 2024): `Keras=3.4.1`, `Tensorflow=2.16.2`

Capítulo 4

Implementación

4.1 Modelo SARIMA

En primer lugar, se describe la serie con una descomposición clásica para tener en cuenta el tipo de tendencia y estacionalidad que presenta.

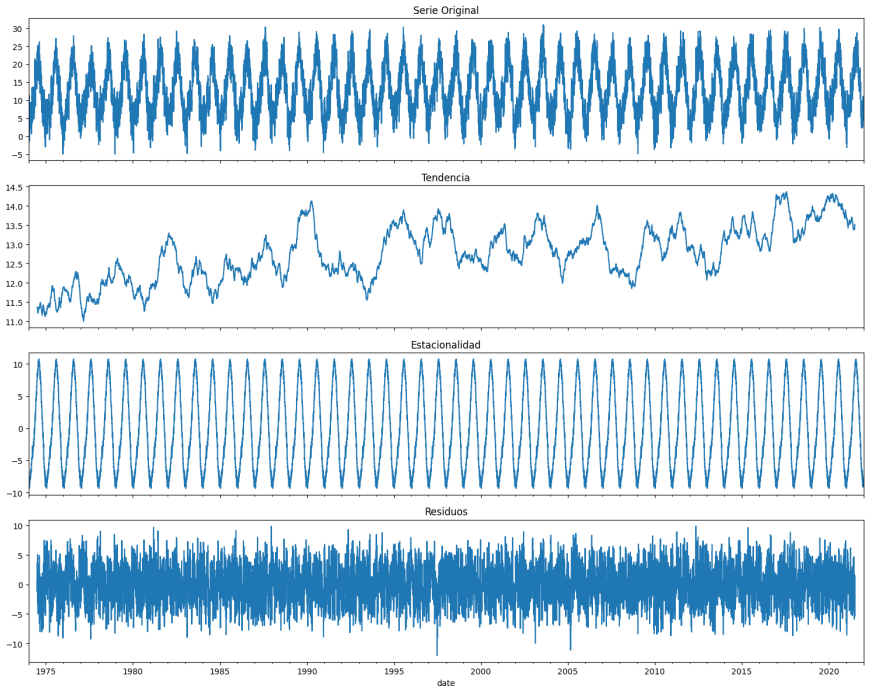

Figura 4.1: Descomposición de la serie temporal de la temperatura media de Valladolid (1974 a 2022)

La tendencia no se ve a simple vista, eso se estudiará a través del p-valor de la constante al ajustar el modelo a la serie diferenciada, pero se puede confirmar que el periodo estacional de la serie es $s = 365$.

Para determinar los órdenes del modelo *(p,d,q)(P,D,Q)* usamos los gráfico ACF y PACF. Lo primero que se puede determinar son los valores de *d* o *D*, es decir, si se necesita diferenciar por la parte regular, por la parte estacional o por ambas.

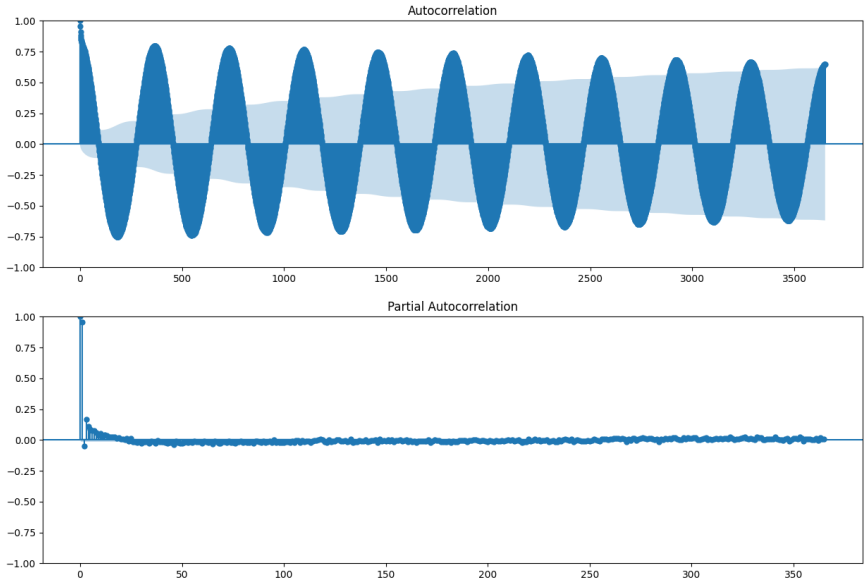

Figura 4.2: ACF y PACF de la serie original

En la parte regular, no parece necesitarse diferenciación, sin embargo, por la parte estacional no es estacionaria ya que en las autocorrelaciones en los múltiplos de 365 se ve decrecimiento lento, no exponencial. Es por ello que se realiza una diferenciación estacional, es decir:

$$\nabla_{365} x_t = x_t - x_{t-365} \tag{4.1}$$

Tras la diferenciación se aplica el test de Dickey-Fuller, que nos da un p-valor significativo por lo que la serie diferenciada es estacionaria por la parte regular y por la parte estacional. Esto implica que se tomará $d = 0$ y $D = 1$.

Se vuelven a representar los gráficos ACF y PACF para poder decidir los valores de p, q y P, Q que vamos a elegir como órdenes de los polinomios *backward* de los modelos SARIMA que se van a probar.

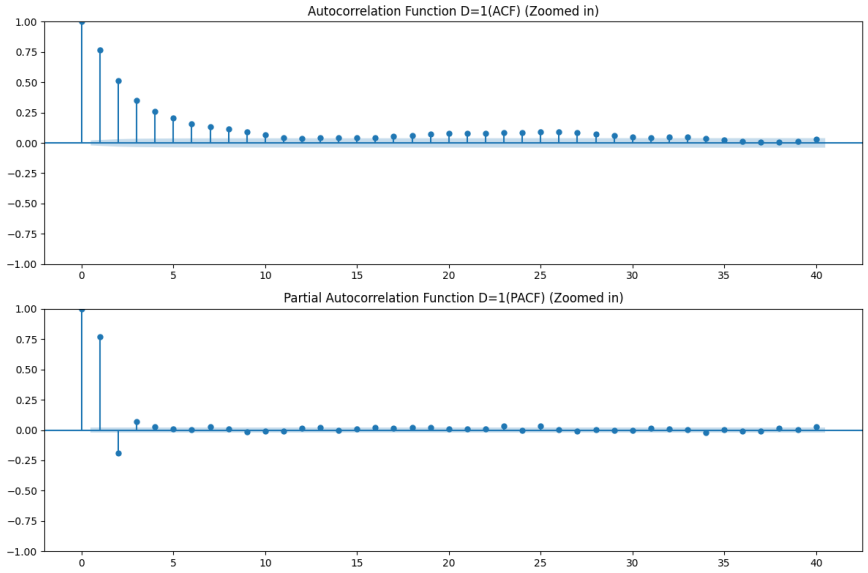

Figura 4.3: ACF y PACF de la serie diferenciada una vez por la parte estacional

A la vista de la figura 4.3, se decide ajustar un modelo SARIMA $(3,0,0)(0,1,1)_{365}$ cuya fórmula en términos generales sería:

$$(1-\phi_1 B-\phi_2 B^2-\phi_3 B^3)(1-B^{365})X_t = c+(1-\Theta_1 B^{365})a_t \tag{4.2}$$

A continuación, se procede a la implementación en Python, tal y como se describe en el apéndice C. Tras el ajuste del modelo, la estimación obtenida ha sido:

```
                                SARIMAX Results
==========================================================================================
Dep. Variable:                           mean_temp   No. Observations:                17532
Model:            SARIMAX(3, 0, 0)x(0, 1, [1], 365)   Log Likelihood             -36616.221
Date:                              Thu, 04 Jul 2024   AIC                          73244.442
Time:                                      11:30:30   BIC                          73290.946
Sample:                                           0   HQIC                         73259.770
                                              - 17532
Covariance Type:                              approx
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
intercept      0.0090      0.001      7.781      0.000       0.007       0.011
ar.L1          0.8991      0.008    118.070      0.000       0.884       0.914
ar.L2         -0.2120      0.010    -20.948      0.000      -0.232      -0.192
ar.L3          0.0669      0.008      8.783      0.000       0.052       0.082
ma.S.L365     -0.9747      0.007   -139.623      0.000      -0.988      -0.961
sigma2         3.9203      0.046     85.728      0.000       3.831       4.010
===================================================================================
Ljung-Box (L1) (Q):                   0.07   Jarque-Bera (JB):               128.63
Prob(Q):                              0.79   Prob(JB):                         0.00
Heteroskedasticity (H):               0.96   Skew:                            -0.14
Prob(H) (two-sided):                  0.16   Kurtosis:                         3.31
===================================================================================

Warnings:
[1] Covariance matrix calculated using numerical (complex-step) differentiation.
```

Figura 4.4: Resultados SARIMA en Python (usando L-BFGS para maximizar la verosimilitud)

Los p-valores de todos los estimadores son significativos, la serie es invertible (en base al

coeficiente del parámetro MA) y estacionaria. Al sustituir los estimadores en la fórmula y despejar obtenemos que la fórmula de nuestro SARIMA $(3,0,0)(0,1,1)_{365}$ es:

$$\begin{aligned}\phi_1 &= 0{,}8991\\ \phi_2 &= -0{,}2120\\ \phi_3 &= 0{,}0669\\ \Theta_1 &= 0{,}9747\\ c &= 0{,}0090\\ a_t &= 3{,}9203\end{aligned}$$

$$\begin{aligned}X_t = {} & 0{,}0090 + 0{,}8991X_{t-1} + 0{,}2120X_{t-2} - 0{,}0669X_{t-3} + X_{t-365} - 0{,}8991X_{t-366}\\ & - 0{,}2120X_{t-367} + 0{,}0669X_{t-368} + a_t + 0{,}9747a_{t-365}\end{aligned} \tag{4.3}$$

Cabe señalar que se consideraron otros modelos como, por ejemplo, el modelo SARIMA $(3,0,1)(0,1,0)_{365}$ sin constante, sin embargo las medidas habituales de comparación de modelos hicieron que se decidiera seguir trabajando con el modelo SARIMA $(3,0,0)(0,1,1)_{365}$

A continuación, se muestra el análisis de los residuos del modelo elegido.

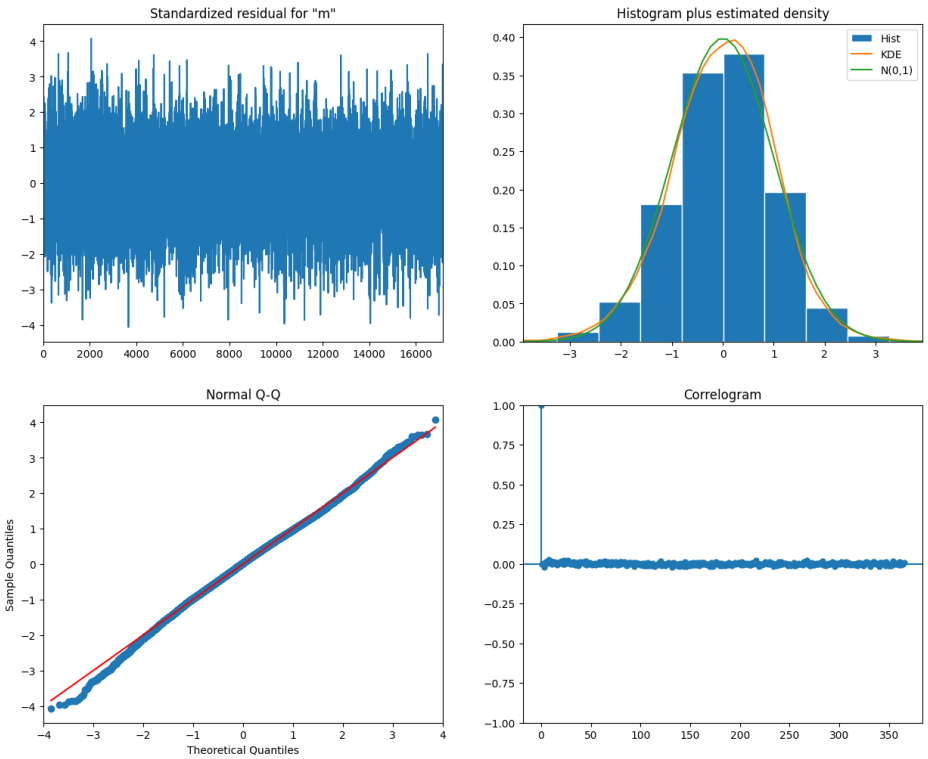

Figura 4.5: Gráficos de análisis de residuos. De arriba a abajo y de izquierda a derecha: Residuos Estandarizados, Histograma y Densidad Estimada, Gráfico Q-Q, Correlograma)

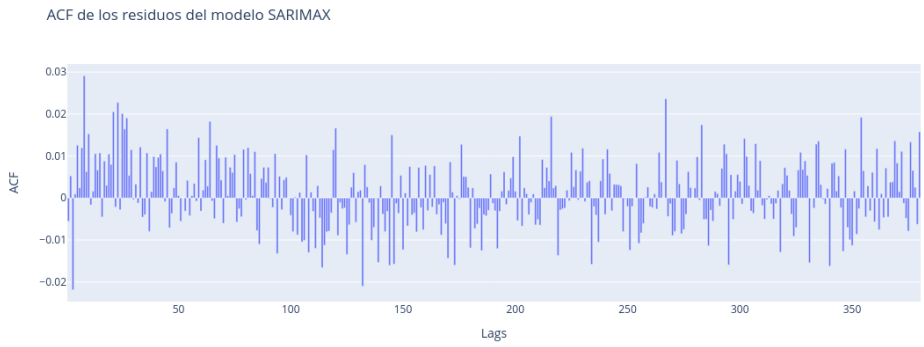

Figura 4.6: ACF de los residuos del modelo

En la figura 4.5, en el gráfico de Residuos Estandarizados, vemos que los residuos parecen oscilar alrededor de cero y no muestran patrones visibles, lo que sugiere que tienen media cero y una varianza constante. El histograma se asemeja a una distribución normal, ya que la línea de densidad estimada (KDE) está muy cerca de la distribución normal estándar. En el plot Q-Q, se puede ver que los puntos se alinean bastante bien con la línea, con alguna desviación en el extremo inferior. En el correlograma (excluyendo el primer valor que siempre es 1), prácticamente todas las barras dentro de las bandas de confianza lo que sugiere que las autocorrelaciones no son significativamente distintas de cero, esto se aprecia mejor en el gráfico 4.6.

Una vez validado el modelo, se obtendrán las predicciones, que tomando esperanzas condicionadas (Fórmula 2.8), puede probarse que a partir de un horizonte de 369 se relacionan como se muestra en 4.5. Para horizontes más cercanos intervendrá también la parte de media móvil. El gráfico con las predicciones y sus bandas se muestra en la figura 5.1.

$$(1 - 0{,}8991B + 0{,}2120B^2 - 0{,}0669B^3)(1 - B^{365})X_n(h) = 0{,}0090 \tag{4.4}$$

$$\begin{aligned} X_n(h) = {} & 0{,}0090 + 0{,}8991X_n(h-1) - 0{,}2120X_n(h-2) + 0{,}0669X_n(h-3) + X_n(h-365) \\ & - 0{,}8991X_n(h-366) + 0{,}2120X_n(h-367) - 0{,}0669X_n(h-368) \quad \forall h > 368 \end{aligned} \tag{4.5}$$

Del modelo SARIMA $(3,0,0)(0,1,1)_{365}$ cabe destacar, que al tener una diferenciación estacional e incluir constante significativa en el modelo, eso supone que la temperatura tendría una parte de tendencia determinista $a + bt$ donde la pendiente b (incremento diario) sería:

$$b = \frac{\text{Media de la serie diferenciada}}{365} = \frac{\text{Constante del modelo}}{(1 - \phi_1 - \phi_2 - \phi_3) \cdot 365} = 1{,}002 \cdot 10^{-4} \tag{4.6}$$

Si bien la pendiente es muy pequeña, es positiva, lo que indica que la temperatura podría sufrir un aumento lento en los próximos años, siendo el **incremento anual de 0.037 grados**. Si hemos elegido el modelo adecuado, el ligero incremento que supone la constante significativa del modelo encajaría con las afirmaciones científicas sobre el calentamiento global[33][34].

4.2 Modelo de suavizado Holt-Winters

Usando la función ExponentialSmoothing Holt-Winters de la librería Statsmodels [35], se ha entrenado el modelo con los valores desde 1974 a 2022. A la vista de los datos y de su descomposición, tal y como se vio en el apartado 4.1, se decide entrenar el modelo aditivo por la parte estacional y la tendencia, además de incluir *damping*.

El código completo está descrito en el apéndice B.

El resultado de las constantes de suavizado, valores iniciales y medidas de ajustes para los datos de entrenamiento se muestran en la figura 4.5.

ExponentialSmoothing Model Results

Dep. Variable:	mean_temp	No. Observations:	17532
Model:	ExponentialSmoothing	SSE	76968.687
Optimized:	True	AIC	26676.331
Trend:	Additive	BIC	29551.890
Seasonal:	Additive	AICC	26692.503
Seasonal Periods:	365	Date:	Mon, 08 Jul 2024
Box-Cox:	False	Time:	23:19:08
Box-Cox Coeff.:	None		

	coeff	code	optimized
smoothing_level	0.9909708	alpha	True
smoothing_trend	0.0068145	beta	True
smoothing_seasonal	0.0014209	gamma	True
initial_level	9.9582772	l.0	True
initial_trend	1.2253513	b.0	True
damping_trend	0.8277667	phi	True

Figura 4.7: Resultados Holt-Winters aditivo con damping

A continuación, se presentan las ecuaciones de actualización del modelo:

- **Nivel (l_t)**:

$$l_t = 0{,}9909708(X_t - s_{t-365}) + 0{,}0090292(l_{t-1} + 0{,}8277667 b_{t-1}) \tag{4.7}$$

- **Tendencia (b_t)**:

$$b_t = 0{,}0068145(l_t - l_{t-1}) + 0{,}822126 b_{t-1} \tag{4.8}$$

- **Estacional (s_t)**:

$$s_t = 0{,}0014209(X_t - l_t) + 0{,}9985791 s_{t-365} \tag{4.9}$$

- **Predicción ($\hat{X}_{t+h}$)**:

$$\hat{X}_{t+h} = l_t + 0{,}8277667^h \cdot b_t + s_{t+h-365(k+1)} \tag{4.10}$$

  donde $k = \left\lfloor \frac{h-1}{365} \right\rfloor$.

Cabe notar que los valores de las constantes de suavizado obtenidas al ajustar el modelo están cercanas a 0 ó 1, lo que indica que un modelo SARIMA será más adecuado que este modelo de suavizado exponencial por la necesidad de la diferenciación para mejorar la modelización.

4.3 Redes neuronales

Como se definió en la sección 3.2.3, para la implementación de los modelos de redes neuronales se ha utilizado la herramienta Keras [30] para la creación de las diferentes arquitecturas, su entrenamiento y evaluación de resultados.

Para la construcción de todos los modelos se han usado dos capas comunes:

- **Dropout**: es una técnica de regularización utilizada para prevenir el sobreajuste del modelo. Esta capa "apaga" aleatoriamente un porcentaje de neuronas durante el entrenamiento, lo que hace que a cada paso del entrenamiento, algunas neuronas se desactivan de forma temporal y aleatoria con probabilidad *p*, lo que obliga a la red a aprender representaciones más robustas de los datos al no depender de la presencia de neuronas específicas [36].
- **Dense**: es la capa de salida. Conocida comúnmente como capa totalmente conectada o capa *feedforward*, esta capa recibe entradas de todas las neuronas de la capa anterior y envia su salida a las neuronas de la siguiente capa [37].

En el caso de las funciones de activación de las neuronas, se han usado en todos los modelos la función de activación ReLU (*rectified linear unit*, que se define como [38]:

$$f(x) = x^+ = max(0, x) = \frac{x + |x|}{2} = \begin{cases} x & \text{if } x > 0, \\ 0 & \text{otherwise.} \end{cases} \tag{4.11}$$

La razón de escoger esta función frente a otras funciones de activación como la función de activación tangencial hiperbólica es su eficiencia computacional y menor cantidad de problemas de desvanecimiento de gradiente.

A continuación, se va a definir la estructura general utilizada para todos los modelos, así como los diferentes pasos que se han seguido hasta obtener los resultados. En el apéndice A se detalla el código.

1. **Carga y preprocesamiento de datos**: usando la herramienta Pandas[27], se ha cargado el conjunto de datos con las fechas y temperaturas. Posteriormente, se ha comprobado la estructura e integridad de los datos cargados.
2. **Normalización**: la normalización es un paso muy importante a la hora de mejorar el funcionamiento de los modelos de redes neuronales, ya que ayuda a acelerar el entrenamiento y mejora su capacidad de generalización [39].
3. **Creación de secuencias**: para que los modelos pudieran admitir la serie temporal, se ha implementado una función para la creación de subsecuencias o ventanas (*windows*). Estas ventanas son vectores de datos consecutivos de longitud definida.

Por ejemplo, suponiendo que tenemos una serie temporal $Y = [Y_1, Y_2, Y_3, ..., Y_n]$, seleccionando una ventana $w = 3$, como *input* de la red neuronal se introducirían vectores de datos tal que $[Y_1, Y_2, Y_3]$ para predecir Y_4, $[Y_2, Y_3, Y_4]$ para predecir Y_5 y así sucesivamente. Se puede manipular la estructura para que prediga los N pasos siguientes en base a la ventana seleccionada.

En el caso de esta implementación se ha decidido probar con varias ventanas de predicción, siendo las ventanas de 1, 2, 3, 4, 5, 6, 7, 10, 14, 28, 60, 90, 180 y 365 días. Las ventanas de predicción más grandes conllevan un mayor coste computacional por lo que se ha decidido limitar las pruebas a las ventanas anteriormente mencionadas. Es decir, se ha testeado tanto con la entrada como con la salida, combinando los valores de la ventana de entrada con los N siguientes pasos (como se verá en el siguiente apartado, 1, 5, 7, 14, 28, 60,180 y 365 pasos).

4. **Separar datos para entrenamiento y validación**: una vez creadas las secuencias, se separa el conjunto de datos en dos conjuntos. El conjunto de entrenamiento, con el que se entrena la red neuronal, con los datos desde 1974 al 31 de diciembre de 2021 (48 años); el conjunto de validación, con el que se evalua posteriormente la eficacia del modelo con el resto de datos.

5. **Definición de la arquitectura de la red neuronal**: el siguiente paso ha sido definir la estructura del modelo, las diferentes capas y los diferentes parámetros de dichas capas. La arquitectura se define en las siguientes subsecciones para cada uno de los casos.

6. **Entrenamiento del modelo**: el proceso mediante el cual la red aprende la regresión de la serie temporal, ajustando los parámetros (pesos y sesgos) usando los datos de entrenamiento. También se han incluído pequeñas modificaciones para mejorar la eficiencia del entrenamiento, como en este caso el *early stop* en el cual el entrenamiento se detiene si no mejora sustancialmente en el número de etapas (paciencia) determinado, ahorrando mucho tiempo.

7. **Evaluación y predicción de los siguientes N pasos**: finalmente, una vez entrenado el modelo, para poder realizar la comparación de los modelos se ha procedido a usar los modelos entrenados para las diferentes ventanas para predecir los N pasos siguientes considerando los pasos de 1, 5, 7, 14, 28, 60, 180, 365días, también los pasos desde el principio del conjunto de validación hasta el final (821 días).

4.3.1 Implementación redes neuronales recurrentes simple

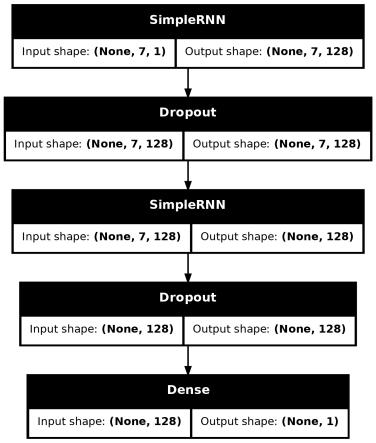

Figura 4.8: Ejemplo de arquitectura del modelo RNN simple

La arquitectura descrita en la figura, tiene dos capas Simple RNN (ventana de 7 días, 128 unidades), dos capas Dropout para evitar sobreajuste y una capa Dense de salida para predecir el paso siguiente, se puede modificar el número de pasos que devuelve el modelo.

4.3.2 Implementación LSTM básicas

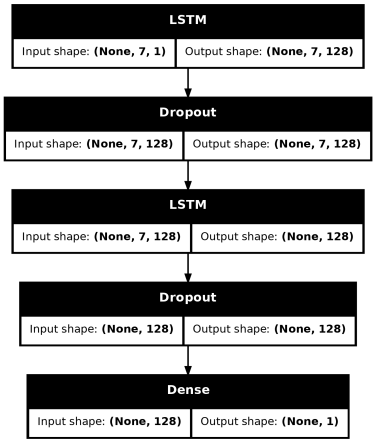

Figura 4.9: Ejemplo de arquitectura del modelo LSTM básico

La arquitectura descrita en la figura, tiene dos capas LSTM (ventana de 7 días, 128 unidades), dos capas Dropout para evitar sobreajuste y una capa Dense de salida para predecir el paso siguiente,se puede modificar el número de pasos que devuelve el modelo.

4.3.3 Implementación LSTM bidireccional

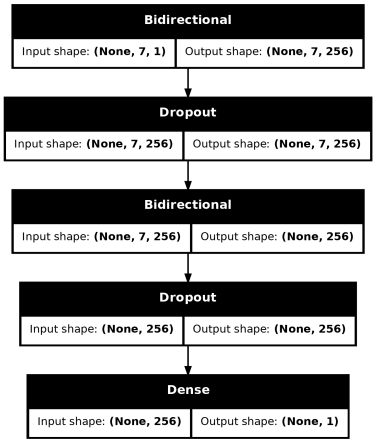

Figura 4.10: Ejemplo de arquitectura del modelo LSTM Bidireccional

La arquitectura descrita en la figura, tiene dos capas Bidireccionales LSTM (ventana de 7 días, 256 unidades), dos capas Dropout para evitar sobreajuste y una capa Dense de salida para predecir el paso siguiente, se puede modificar el número de pasos que devuelve el modelo.

4.3.4 Implementación redes GRU

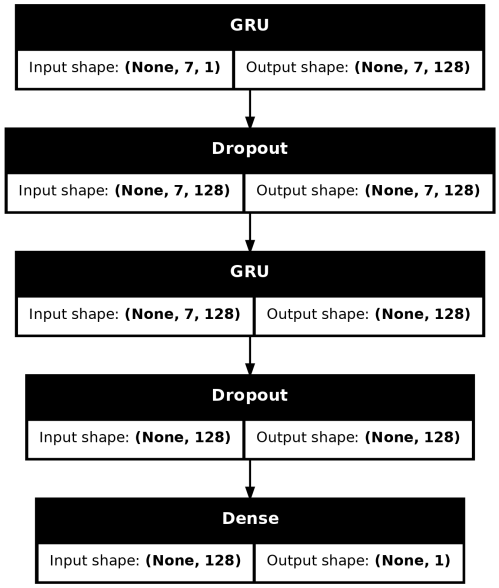

Figura 4.11: Ejemplo de arquitectura del modelo GRU

La arquitectura descrita en la figura, tiene dos capas GRU (ventana de 7 días, 128 unidades), dos capas Dropout para evitar sobreajuste y una capa Dense de salida para predecir el paso siguiente, se puede modificar el número de pasos que devuelve el modelo.

4.3.5 Implementación redes convolucionales con LSTM

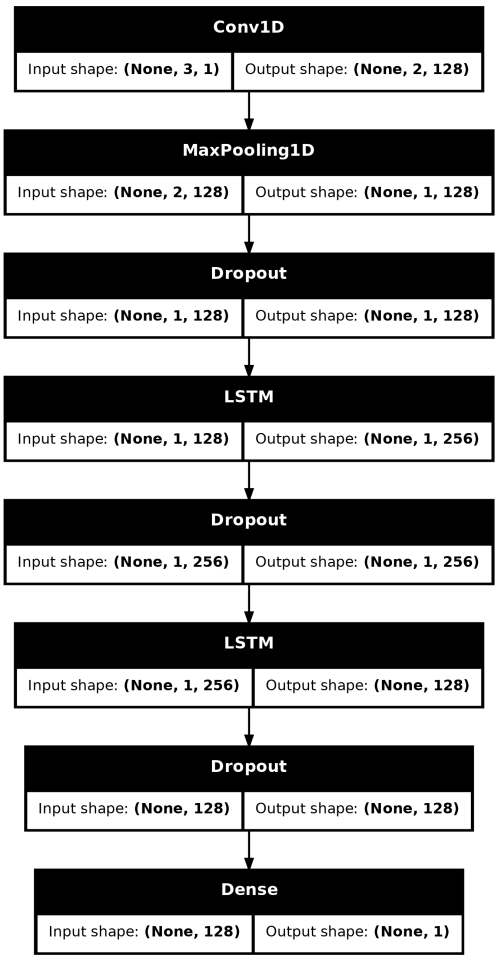

Figura 4.12: Ejemplo de arquitectura del modelo Convolucional+LSTM

La arquitectura descrita en la figura, tiene una capa Convolucional (ventana de 3 días, 128 filtros, 2 kernels), una capa MaxPooling (2 kernels, 128 filtros), dos capas LSTM (128 unidades), tres capas Dropout para evitar sobreajuste y una capa Dense de salida para predecir el paso siguiente, se puede modificar el número de pasos que devuelve el modelo.

Capítulo 5

Resultados y conclusiones

Para comparar la precisión en la predicción de los modelos se ha decidido usar la métrica del RMSE (*Root Mean Squared Error*, que es una métrica muy utilizada para la comparación de modelos bajo un mismo conjunto de datos [40]. Esta métrica calcula la raíz cuadrada de la media de los errores al cuadrado entre los valores predichos por el modelo y los valores observados. Es una medida de la magnitud de los errores y da una idea de cuánto difieren, en promedio, las predicciones de los valores reales. Su fórmula es:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{5.1}$$

Esta métrica se ha utilizado para calcular el error en la predicción de los siguientes N pasos (días) siendo este N: 1 día, 7 días, 14 días, 60 días, 365 días, 821 días (longitud conjunto de prueba).

Para realizar las pruebas para los modelos descritos, se ha utilizado el siguiente *hardware* a fin de igualar las condiciones de ejecución para poder comparar la eficiencia de cada algoritmo.

- Procesador: Ryzen 7 7840HS
- Memoria RAM: 16GB
- Tarjeta Gráfica: RTX 4050

5.1 Comparación de resultados

5.1.1 Modelo SARIMA $(3,0,0)(0,1,1)_{365}$

Tiempo de ejecución: 18 horas, 43 minutos, 20 segundos

Steps	**1**	**7**	**14**	**60**	**365**	**821**
RMSE	2.5884	2.5037	2.5713	2.038	3.4762	3.4311

Cuadro 5.1: Tabla con el RMSE del modelo SARIMA para los horizontes de tiempo mostrados

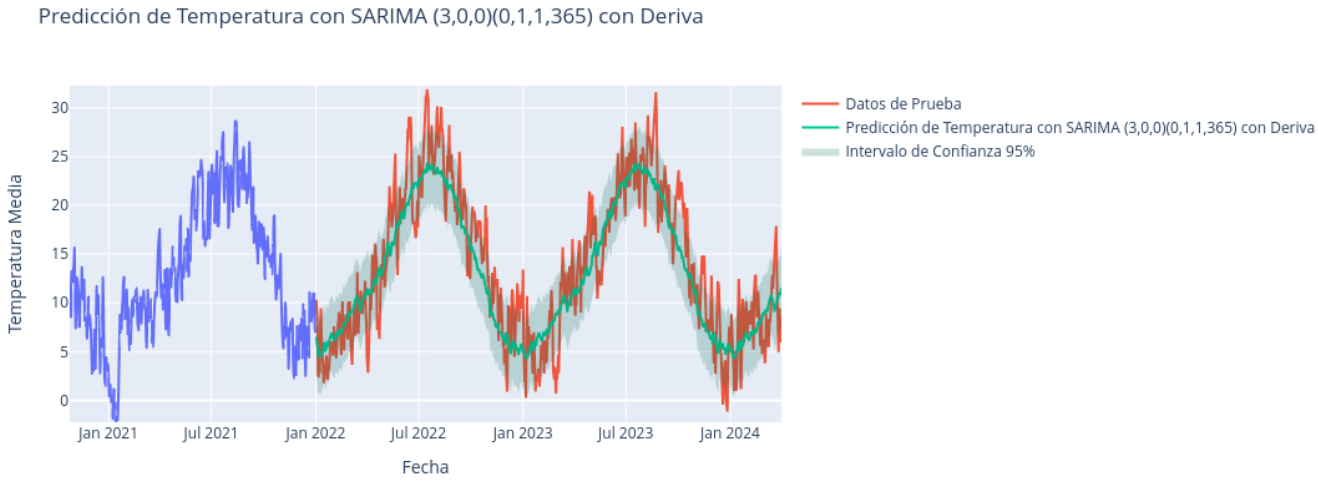

Figura 5.1: Resultado predicción SARIMA $(3,0,0)(0,1,1)_{365}$ con constante (2022-2024)

5.1.2 Modelo Holt-Winters con damping

Tiempo de ejecución: 24 segundos

Steps	**1**	**7**	**14**	**60**	**365**	**821**
RMSE	1.9112	2.7763	3.2050	4.4624	9.8471	3.9210

Cuadro 5.2: Tabla con el RMSE del modelo de suavizado exponencial para los horizontes de tiempo mostrados

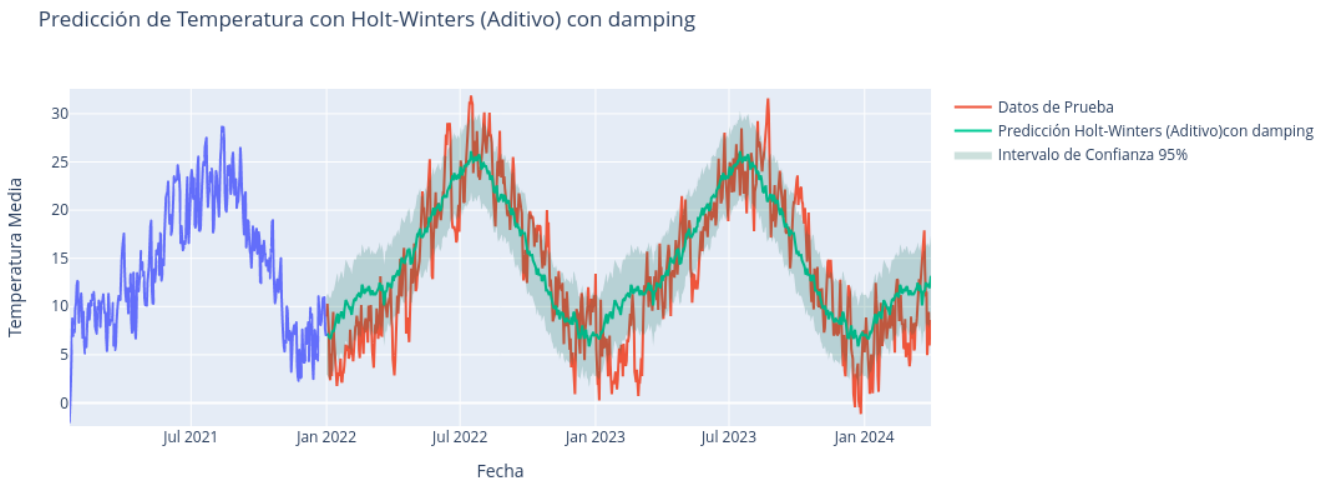

Figura 5.2: Resultado predicción Holt-Winters aditivo con damping (2022-2024)

5.1.3 Modelo RNN Simple

Tiempo de ejecución promedio (se ha entrenado un modelo para cada una de las ventanas de tiempo): 1 minuto, 5 segundos

window	steps=1	steps=7	steps=14	steps=28	steps=60	steps=180	steps=365
1	0.015	2.172	5.425	4.734	4.344	5.858	11.972
2	0.165	2.172	6.082	3.749	3.225	13.408	13.400
3	0.145	2.413	5.550	7.328	12.315	14.594	12.974
4	0.058	2.188	5.635	3.356	9.994	6.668	12.975
5	0.239	2.161	5.466	3.662	3.631	13.458	8.502
6	0.189	2.502	5.512	3.893	3.093	7.088	8.162
7	0.376	2.386	5.206	4.626	3.380	5.288	11.214
10	0.392	2.136	5.042	4.183	3.316	11.038	8.460
14	0.603	3.386	4.872	3.242	3.421	6.306	7.389
28	0.487	3.611	4.837	4.887	3.944	10.323	8.097
60	0.716	4.062	4.925	4.426	6.837	11.724	7.848
90	0.697	2.723	5.566	4.891	5.474	8.740	4.945
180	0.449	3.139	5.384	5.233	8.732	11.997	4.468
365	0.410	2.155	4.756	4.229	4.147	11.933	10.677

Cuadro 5.3: Tabla de resultados del RMSE para RNN simple de cada ventana de tiempo frente a las predicciones con diferentes horizontes de tiempo. Marcado en azul el menor RMSE.

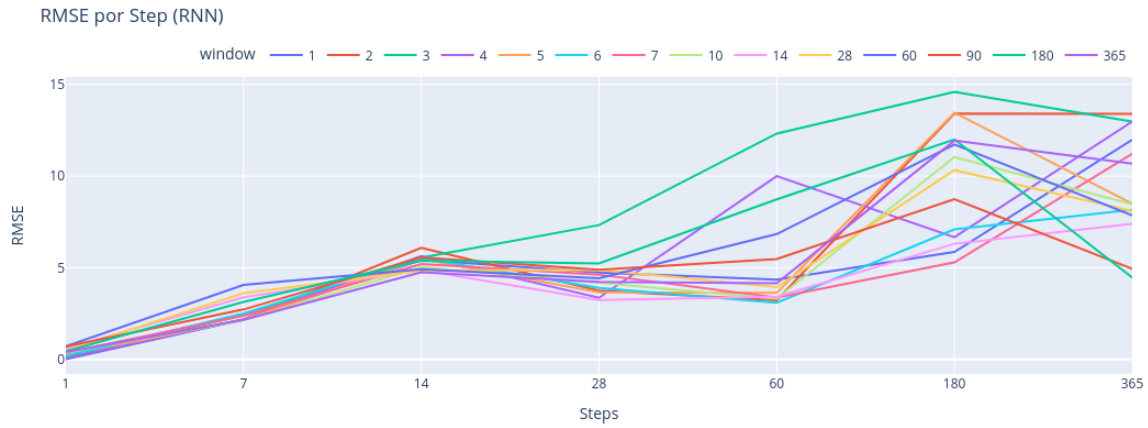

Figura 5.3: RMSE por paso en en el horizonte de tiempo (Simple RNN)

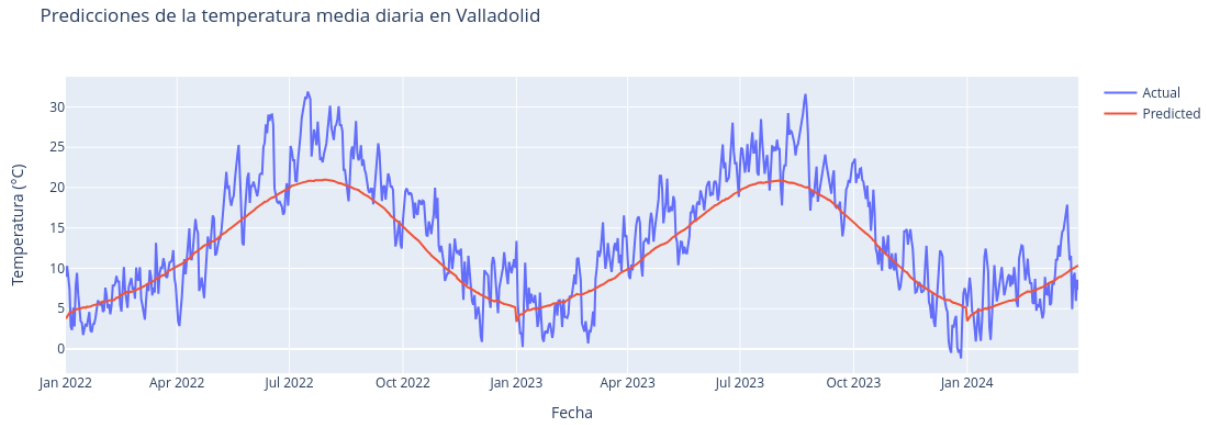

Figura 5.4: Ejemplo de predicción RNN Simple a largo plazo sobre los datos de prueba (821 días) (Modelo window=180, n_steps=365)

Como se puede apreciar tanto en la tabla 5.3 como visualmente en la figura 5.3, inicialmente todas las ventanas de tiempo predicen a corto plazo con un RMSE similar, sin embargo a medio y largo plazo, los modelos comienzan a diferir. Siendo el mejor modelo, el modelo con ventana de 180 días con salida de predicción de los siguientes 365 días. Le sigue muy de cerca el modelo de ventana de 90 días. Cabe señalar el modelo con ventana de 365 días, ya que parece obtener un RMSE mucho mayor que otros modelos, esto podría deberse probablemente al desvanecimiento de gradiente.

En la figura 5.4 podemos ver con el horizonte de tiempo de 821 días que el modelo capta decentemente la estacionalidad de la serie, sin embargo, parece quedarse un poco corto frente a los modelos clásicos. El RMSE obtenido para los 821 dias de previsión es 5.6829, superior a los modelos clásicos.

5.1.4 Modelo LSTM Básico

Tiempo de ejecución promedio (se ha entrenado un modelo para cada una de las ventanas de tiempo): 1 minuto, 33 segundos

window	steps=1	steps=7	steps=14	steps=28	steps=60	steps=180	steps=365
1	0.206	2.280	5.569	6.179	5.190	5.217	12.001
2	0.148	2.312	5.290	5.297	7.587	5.638	11.658
3	0.124	2.439	5.466	4.543	3.917	5.648	12.513
4	0.241	2.283	5.489	3.600	4.023	5.422	12.992
5	0.072	2.267	5.390	4.109	4.727	8.112	12.555
6	0.603	2.228	5.539	3.395	8.945	11.294	11.795
7	0.209	2.067	5.223	4.316	3.301	12.049	11.670
10	0.445	2.627	4.637	3.356	3.488	11.472	9.541
14	0.696	3.770	4.119	3.540	4.173	14.718	8.959
28	0.832	3.665	4.295	4.761	4.221	11.637	10.010
60	0.546	2.945	5.059	5.928	6.342	10.969	5.148
90	0.467	3.942	4.825	5.978	6.849	11.147	20.398
180	1.264	4.106	4.636	4.713	5.064	12.188	4.925
365	1.177	3.117	4.688	3.844	6.152	12.964	3.511

Cuadro 5.4: Tabla de resultados del RMSE para LSTM básicas de cada ventana de tiempo frente a las predicciones con diferentes horizontes de tiempo. Marcado en azul el menor RMSE.

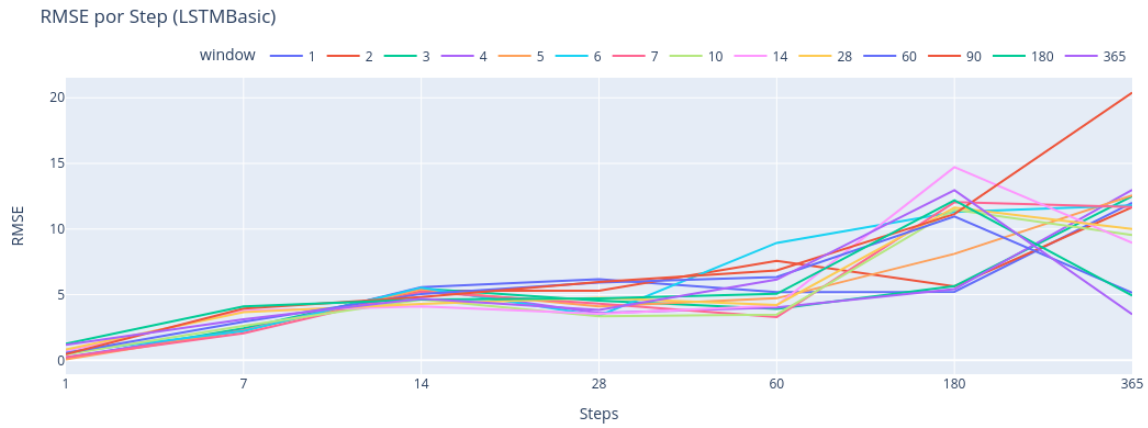

Figura 5.5: RMSE por paso en el horizonte de tiempo (LSTM Básica)

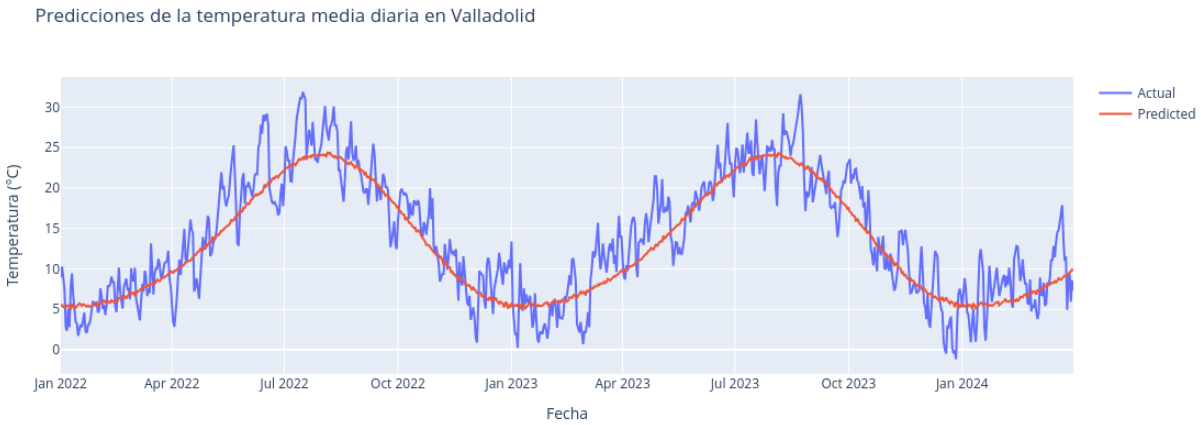

Figura 5.6: Ejemplo de predicción LSTM Básica a largo plazo sobre los datos de prueba (821 días) (Modelo window=365, n_steps=365)

Tanto en la tabla 5.4 como visualmente en la figura 5.5, inicialmente todas las ventanas de tiempo predicen a corto plazo con un RMSE similar. A partir de los 60 días es cuando los modelos comienzan a diverger. Siendo el mejor modelo, el modelo con ventana de 365 días con salida de predicción de los siguientes 365 días, seguido por el modelo de ventana 180 días. Como se puede ver, en este caso al haber solucionado el problema del desvanecimiento del gradiente con las unidades LSTM, los modelos con mayor ventana son capaces de captar mejor los patrones más alejados en el tiempo.

En la figura 5.6 podemos ver con el horizonte de tiempo de 821 días que el modelo capta mejor que el modelo anterior el patrón estacional, asemejándose un poco más a los modelos clásicos. El RMSE de la predicción de los 821 días es 3.5324, ligeramente peor que el modelo SARIMA pero mejor que el modelo de suavizado.

5.1.5 Modelo LSTM Bidireccional

Tiempo de ejecución promedio (se ha entrenado un modelo para cada una de las ventanas de tiempo): 1 minuto, 49 segundos

window	steps=1	steps=7	steps=14	steps=28	steps=60	steps=180	steps=365
1	0.064	2.332	6.389	5.915	5.129	5.542	11.996
2	0.269	2.080	5.761	5.468	5.142	5.414	9.913
3	0.122	2.226	5.632	4.335	3.375	13.593	10.674
4	0.004	2.133	5.307	4.546	4.437	6.037	12.577
5	0.155	2.163	5.066	3.611	3.113	14.312	11.970
6	0.243	2.176	5.351	5.013	3.800	5.838	11.710
7	0.190	2.105	5.217	3.406	11.496	12.481	11.984
10	0.767	2.265	5.031	4.639	7.958	6.236	7.719
14	1.157	2.913	4.478	3.601	8.993	16.041	10.186
28	0.666	3.832	4.711	3.697	3.444	11.282	9.061
60	0.628	3.451	6.561	3.725	5.815	11.756	4.803
90	1.027	3.323	4.726	3.976	8.376	11.380	4.387
180	0.711	2.807	4.872	3.781	6.677	11.962	3.960
365	1.012	3.893	4.962	4.467	6.331	12.331	3.175

Cuadro 5.5: Tabla de resultados del RMSE para LSTM Bidireccionales de cada ventana de tiempo frente a las predicciones con diferentes horizontes de tiempo. Marcado en azul el menor RMSE.

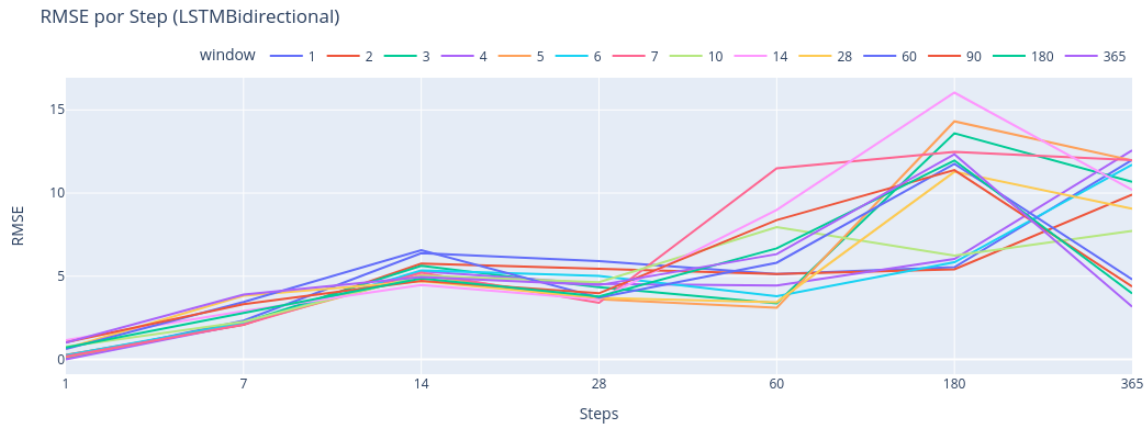

Figura 5.7: RMSE por paso en el horizonte de tiempo (LSTM Bidireccional)

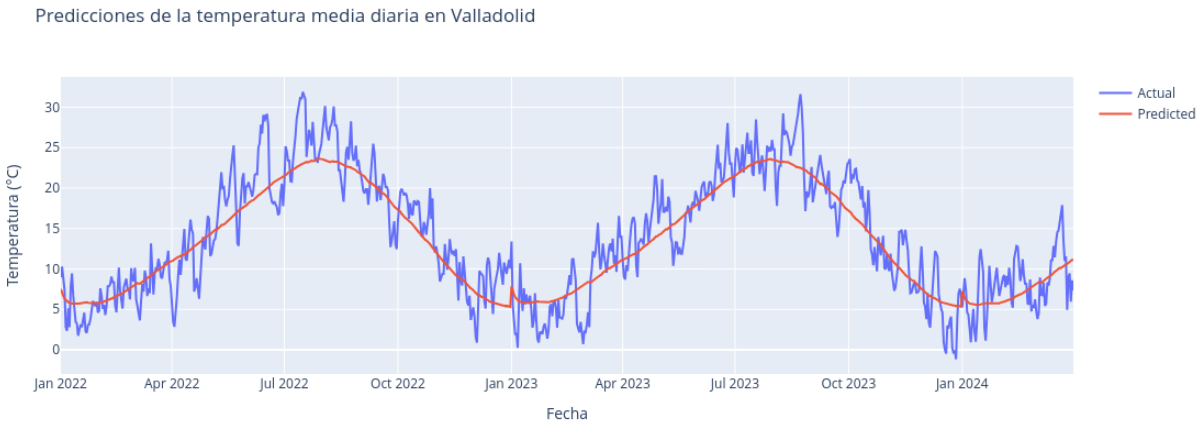

Figura 5.8: Ejemplo de predicción LSTM Bidireccional a largo plazo sobre los datos de prueba (821 días) (Modelo window=365, n_steps=365)

Tanto en la tabla 5.5 como visualmente en la figura 5.7, inicialmente todas las ventanas de tiempo predicen a corto plazo con un RMSE similar. En la barrera de los 60 días suben los valores de todos los modelos bastante y de forma más abrupta que en la arquitectura LSTM Básica. El mejor modelo es el modelo con ventana de 365 días con salida de predicción de los siguientes 365 días, lo que es congruente con la arquitectura LSTM Básica ya que ambas arquitecturas se basan en el mismo principio. Cabe destacar, que el RMSE del modelo es ligeramente menor a largo plazo que en su contraparte básica.

En la figura 5.8 podemos ver con el horizonte de tiempo de 821 días que el modelo captura de forma similar al modelo anterior la estacionalidad de los datos. El RMSE de la predicción de los 821 días es 3.4682, mejor que el modelo anterior, pero ligeramente peor que el modelo SARIMA.

5.1.6 Modelo GRU

Tiempo de ejecución promedio (se ha entrenado un modelo para cada una de las ventanas de tiempo): 1 minuto, 26 segundos

window	steps=1	steps=7	steps=14	steps=28	steps=60	steps=180	steps=365
1	0.239	2.166	5.710	4.641	4.995	6.333	7.667
2	1.157	1.958	5.479	4.958	10.298	7.091	7.491
3	0.233	2.717	5.748	4.769	4.408	6.159	7.550
4	0.001	2.183	5.922	4.438	3.492	6.337	7.631
5	0.411	2.049	5.733	4.339	3.476	6.640	7.490
6	0.355	2.336	5.440	4.648	4.135	5.997	10.922
7	0.875	2.045	5.619	4.089	3.251	6.135	10.599
10	1.772	2.037	4.942	3.693	3.368	6.623	12.368
14	1.726	2.625	4.301	3.611	3.750	6.775	8.715
28	1.722	2.421	4.241	3.587	5.136	7.970	11.625
60	1.437	4.337	4.579	4.382	6.146	6.451	4.499
90	1.582	4.643	5.088	5.462	7.850	6.733	7.451
180	1.100	5.399	4.355	3.746	5.956	11.342	7.485
365	1.503	2.100	4.550	3.819	4.398	6.292	8.725

Cuadro 5.6: Tabla de resultados del RMSE para GRU de cada ventana de tiempo frente a las predicciones con diferentes horizontes de tiempo. Marcado en azul el menor RMSE.

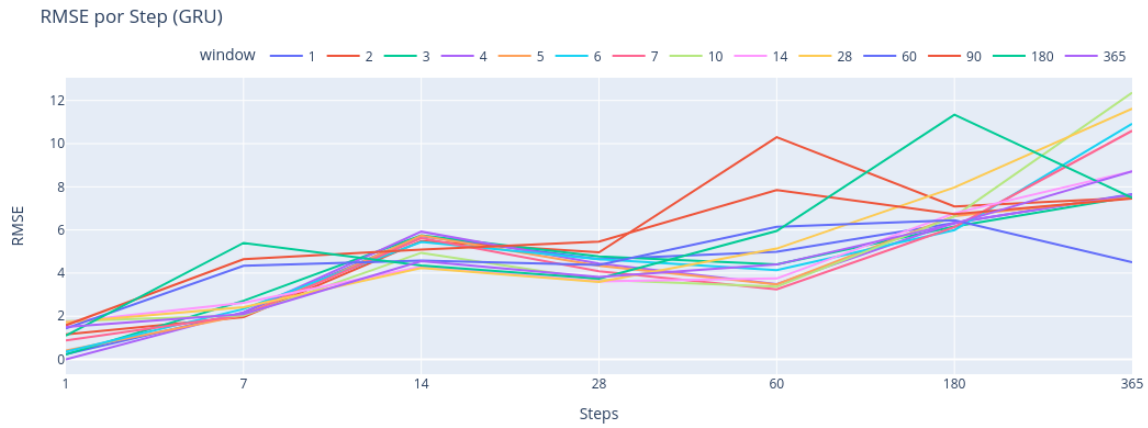

Figura 5.9: RMSE por paso en en el horizonte de tiempo (GRU)

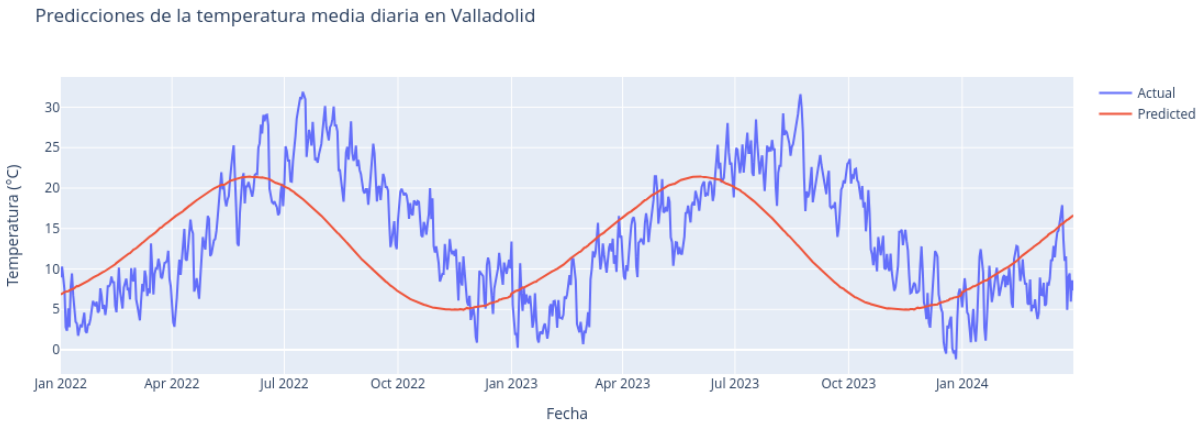

Figura 5.10: Ejemplo de predicción GRU a largo plazo sobre los datos de prueba (821 días) (Modelo window=60, n_steps=365)

Tanto en la tabla 5.6 como más visualmente en la figura 5.9, inicialmente todas las ventanas de tiempo predicen a corto plazo con un RMSE similar, bastante más elevado que las arquitecturas anteriores. El mejor modelo es el modelo con ventana de 60 días con salida de predicción de los siguientes 365 días. Esto tiene sentido teniendo en cuenta la ausencia de la puerta de olvido, lo que hace que el modelo sea más sencillo y eficiente, pero a costa de una precisión en las predicciones menor.

En la figura 5.10 podemos ver con el horizonte de tiempo de 821 días que el modelo captura una estacionalidad, pero no una estacionalidad que se parezca a los datos reales, lo que hace que el error sea muy elevado; el RMSE de este modelo para los datos de predicción es 6.2302, el peor modelo de todos los de redes neuronales planteado.

5.1.7 Modelo Convolucional+LSTM

Tiempo de ejecución promedio (se ha entrenado un modelo para cada una de las ventanas de tiempo): 2 minutos, 3 segundos

window	steps=1	steps=5	steps=7	steps=14	steps=28	steps=60	steps=180	steps=365
4	1.109	1.845	2.002	5.411	5.888	4.509	5.795	12.031
5	0.713	1.126	2.109	5.881	5.915	4.864	5.185	10.552
6	0.772	1.335	2.111	5.445	6.549	4.901	12.513	9.016
7	0.369	1.421	2.001	5.141	4.386	4.663	12.215	9.514
10	0.559	1.678	3.171	4.664	4.909	4.771	13.278	8.470
14	1.061	3.361	3.706	3.884	3.535	4.064	12.242	8.369
28	0.945	3.154	4.309	4.375	5.405	3.223	12.706	8.106
60	1.448	3.342	3.368	5.369	3.587	6.904	11.628	5.268
90	1.879	3.387	3.632	4.518	4.728	5.471	11.157	5.737
180	1.941	2.922	4.196	4.401	4.505	4.814	11.053	4.279
365	1.952	3.195	3.154	4.422	4.769	5.503	11.901	3.708

Cuadro 5.7: Tabla de resultados del RMSE para Convolucionales + LSTM de cada ventana de tiempo frente a las predicciones con diferentes horizontes de tiempo. Marcado en azul el menor RMSE.

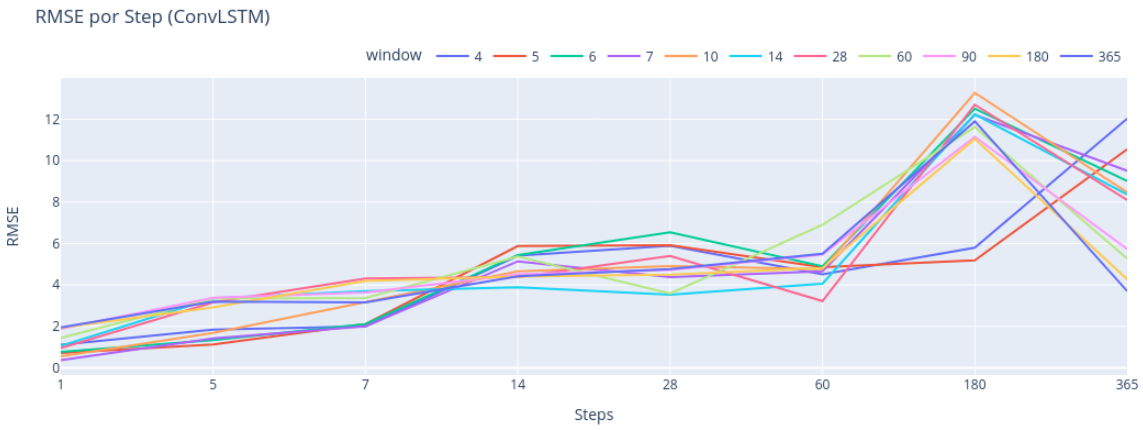

Figura 5.11: RMSE por paso en en el horizonte de tiempo (Convolucionales+LSTM)

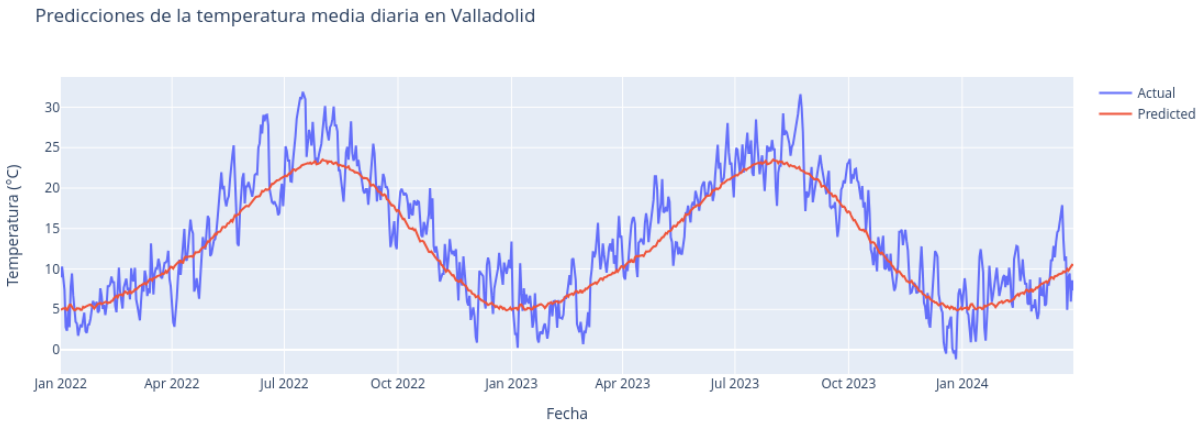

Figura 5.12: Ejemplo de predicción Convolucionales+LSTM a largo plazo sobre los datos de prueba (821 días) (Modelo window=365, n_steps=365)

Tanto en la tabla 5.7 como en la figura 5.11, se aprecia en este caso que el error es el más consistente de todos los modelos para todas las ventanas de predicción y todos los pasos, a excepción quizás de las predicciones para el horizonte de tiempo de 180 días, donde la mayoría de modelos empeoran bastante el error. Si bien es cierto que el error es mayor que en las redes LSTM básicas y LSTM bidireccionales a corto, medio y largo plazo, este modelo tiene un error mucho más estable.

En la figura 5.12 podemos ver con el horizonte de tiempo de 821 días que el modelo captura bastante bien la estacionalidad, e incluso algún pico de temperatura. Su RMSE es de 3.6993, lo que lo sitúa por detras del modelo SARIMA y ambas redes LSTM (básicas y bidireccionales) pero por delante del modelo de suavizado, RNN Simple y GRU. Este resultado podría mejorarse, como se mencionará en el apartado 5.3, buscando mejores parámetros.

5.1.8 Comparación final

	Modelo SARIMA	Modelo LSTM Bidirec.	Modelo LSTM Básico	Modelo Conv. +LSTM	Modelo de suavizado exp.	Modelo RNN Simple	Modelo GRU
RMSE 821 días	3.4311	3.4682	3.5324	3.6993	3.9210	5.6829	6.2302

Cuadro 5.8: Comparativa de RMSE de todos los modelos (de menor a mayor)

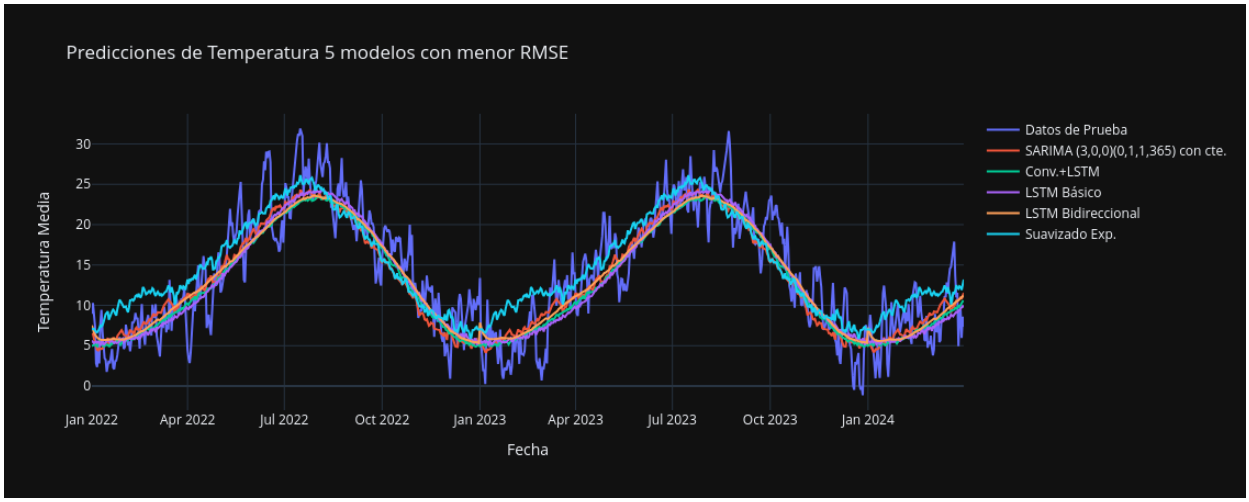

Figura 5.13: Gráfico de los 4 modelos con menor RMSE frente a los datos de prueba reales

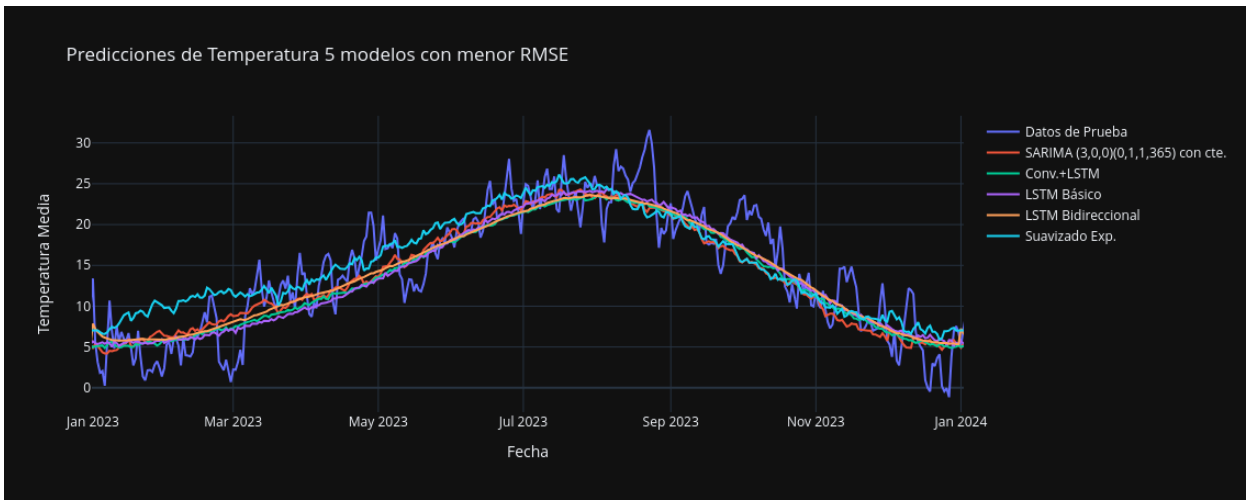

Figura 5.14: Gráfico de los 4 modelos con menor RMSE frente a los datos de prueba reales (Ampliado a vista un año)

Finalmente, tenemos la comparativa visual de los 5 modelos con menor RMSE (SARIMA, LSTM Bidireccional, LSTM básico, Convolucional+LSTM y modelo de suavizado exponencial)

en la figura 5.13. Se han elegido estos 5 modelos ya que son los modelos con valores de RMSE más próximos. Como se puede apreciar en la imagen y de forma más ampliada en la figura 5.14, todos los modelos hacen predicciones muy similares. Destaca un poco más el modelo de suavizado exponencial ya que es el que mayor RMSE tenía.

5.2 Conclusiones

Los 7 modelos planteados en Python tienen sus fortalezas y debilidades, cabe destacar las siguientes:

- **Precisión en la predicción:** los modelos clásicos (SARIMA y Holt-Winters) han dado resultados parecidos o mejores a largo plazo que algunos modelos de redes neuronales, ya que la serie estudiada tiene una estacionalidad muy clara y quizás una ligera tendencia lineal. Los modelos de redes neuronales captan bien los patrones locales más ocultos pero no captan los patrones estacionales tan bien como los modelos clásicos, a excepción de los que usan una ventana grande (180 días o 365 días). A corto e incluso medio plazo, predicen con mayor precisión que los modelos clásicos. Podrían mejorarse las precisiones a largo plazo con ventanas más amplias, pero aumentaría mucho la complejidad de las redes.

- **Complejidad computacional y temporal:** en Python, el modelo más eficiente y menos costoso computacionalmente hablando ha sido el modelo Holt-Winters. Las librerías en Python que trabajan con SARIMA son muy ineficientes y costosas, llegando a consumir casi todos los recursos de la máquina. Los modelos de redes neuronales son costosos computacionalmente, especialmente para ventanas de tiempo altas, pero sus tiempos no son extremadamente elevados.

- **Interpretabilidad:** los modelos clásicos son más interpretables ya que se pueden ver los parámetros y estimadores que utilizan para generar las predicciones. Los modelos de redes neuronales son más vistos como "caja negra", lo que no permite ver realmente cómo se calculan esas predicciones. Además cabe destacar que en el caso de los modelos clásicos se dispone de las bandas de predicción de base, que si bien es cierto que se podrían implementar en los modelos de redes neurolales, incrementarían mucho la complejidad.

En conclusión, cada modelo tiene sus desventajas y ventajas y su uso dependerá del conjunto de datos utilizado, así como del uso que se quiera dar (predicción a corto, medio o largo plazo).

5.3 Limitaciones y líneas futuras

- **Mejora de resultados de los modelos de redes neuronales propuestos mediante búsqueda de hiperparámetros**: la búsqueda de hiperparámetros es una técnica muy utilizada a la hora de entrenar modelos, ya que optimiza los resultados obtenidos y que consiste en probar diferentes valores para los parámetros de la red neuronal y sus capas. Algunos de los parámetros más utilizados suelen ser el ratio de aprendizaje, el número de neuronas por capa o la

función de activación de la capa. En el caso de las redes convolucionales también se podrían haber probado diferentes tamaños de kernel o de filtro.

Debido a su gran coste computacional, la técnica se ha aplicado en menor medida en el trabajo, al estar probando diferentes valores de ventanas de tiempo y diferentes tamaños de salida.

- **Pruebas con otros modelos**: se pueden probar muchos modelos, que incluyen la mezcla de varios modelos diferentes u otros modelos que han tomado popularidad en los últimos años:
  - Redes Transformer: son la base de los modelos de lenguaje grandes como GPT, LLaMa o Gemini, y que han probado ser muy eficientes con grandes volúmenes de datos gracias a sus mecanismos de atención [41], lo que podría ser muy interesante incluso para series temporales subdiarias.
  - Modelos de Lenguaje: los propios modelos de lenguaje (*GPT, LLama* o *Gemini*) que procesan bien secuencias, han llamado también la atención de acádemicos para su uso en la predicción de series temporales, sin embargo su uso ha sido recientemente puesto en duda en el artículo *Are Language Models Actually Useful for Time Series Forecasting?*[42].
  - Redes *Temporal Convolutional Network* (TCN): basadas en las redes convolucionales, este modelo está pensado para capturar patrones temporales de alto nivel [43] y es más rápido de entrenar que las redes LSTM. Aunque pensadas inicialmente en el procesamiento de series de tiempo en imágenes como vídeos de vigilancia o la visión de robots, podría ser utilizada en otras tareas.
- **Creación de una librería más eficiente para el uso de modelos SARIMA**: la librería *Statsmodels* ha demostrado ser muy lenta e ineficiente para un conjunto de datos diarios de 48 años, sin embargo, esto parece ser un problema común en las herramientas de Python que usan series temporales, como por ejemplo también la librería *pmdarima* [44] basada en la librería *autoarima* de R. Cabe destacar las grandes diferencias de tiempos entre las librerías de Python probadas y el método de cálculo ARIMA (PROC ARIMA[45]) en el lenguaje SAS, ya que este último trabaja en orden de magnitud de segundos con los cálculos mientras que en Python serían horas.

  Con la librería utilizada, Statsmodels, no se pueden implementar modelos SARIMA con varios periodos, cosa que se sí se puede hacer en SAS, como podría ser el caso, por ejemplo, de una serie diaria de concentración de gases en la atmósfera donde influyen tanto la actividad semanal como las estaciones del año.

  Se podría crear una librería alternativa en Python que permita entrenar modelos con menor coste computacional y de tiempo y que incluyera la funcionalidad anteriormente nombrada.

Bibliografía

[1] George EP Box y Gwilym M Jenkins. *Time series analysis: forecasting and control*. Holden-Day, 1976.

[2] George EP Box, Gwilym M Jenkins, Gregory C Reinsel y Greta M Ljung. *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

[3] Daniel Peña. *Análisis de series temporales*. Alianza, 2005.

[4] Peter J Brockwell y Richard A Davis. *Introduction to time series and forecasting*. Springer, 2002.

[5] Jorge Nocedal. «Updating quasi-Newton matrices with limited storage». En: *Mathematics of computation* 35.151 (1980), págs. 773-782.

[6] Roger Fletcher. «A new approach to variable metric algorithms». En: *The computer journal* 13.3 (1970), págs. 317-322.

[7] Peter R Winters. «Forecasting sales by exponentially weighted moving averages». En: *Management science* 6.3 (1960), págs. 324-342.

[8] Charles C Holt. «Forecasting seasonals and trends by exponentially weighted moving averages». En: *International journal of forecasting* 20.1 (2004), págs. 5-10.

[9] Everette S Gardner Jr y ED McKenzie. «Forecasting trends in time series». En: *Management science* 31.10 (1985), págs. 1237-1246.

[10] Fyedernoggersnodden. *Arquitectura Elman*. Última consulta: 09/07/24. url: `https://commons.wikimedia.org/wiki/File:Elman_srnn.png`.

[11] Holk Cruse. *Neural networks as cybernetic systems*. Thieme Stuttgart, 1996.

[12] Jeffrey L Elman. «Finding structure in time». En: *Cognitive science* 14.2 (1990), págs. 179-211.

[13] TokioSchool. *RNN Types*. Última consulta: 06/07/24. url: `https://www.tokioschool.com/noticias/redes-neuronales-recurrentes/`.

[14] Andrej Karpathy. *Many-to-many RNN*. Última consulta: 09/07/24. url: `https://karpathy.github.io/2015/05/21/rnn-effectiveness/`.

[15] Sepp Hochreiter. «Untersuchungen zu dynamischen neuronalen Netzen». En: *Diploma, Technische Universität München* 91.1 (1991), pág. 31.

[16] Sepp Hochreiter y Jürgen Schmidhuber. «Long Short-term Memory». En: *Neural computation* 9 (dic. de 1997), págs. 1735-80. doi: `10.1162/neco.1997.9.8.1735`.

[17] Rahuljha. *Many-to-many RNN*. Última consulta: 09/07/24. url: `https://towardsdatascience.com/lstm-gradients-b3996e6a0296`.

[18] Mike Schuster y Kuldip K Paliwal. «Bidirectional recurrent neural networks». En: *IEEE transactions on Signal Processing* 45.11 (1997), págs. 2673-2681.

[19] Incfk8. *RNN vs RNN Bidireccional*. Última consulta: 09/07/24. url: https://commons.wikimedia.org/wiki/File:Structural_diagrams_of_unidirectional_and_bidirectional_recurrent_neural_networks.png.

[20] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho y Yoshua Bengio. «Empirical evaluation of gated recurrent neural networks on sequence modeling». En: *arXiv preprint arXiv:1412.3555* (2014).

[21] Ivan Vasilev. *GRU Architecture*. Última consulta: 09/07/24. url: https://www.oreilly.com/library/view/advanced-deep-learning/9781789956177/8ad9dc41-3237-483e-8f6b-7e5f653dc693.xhtml.

[22] Keiron O'shea y Ryan Nash. «An introduction to convolutional neural networks». En: *arXiv preprint arXiv:1511.08458* (2015).

[23] Samvardhan Singh. *Types of neural networks: Convolutional Neural Networks*. Última consulta: 09/07/24. url: https://medium.com/@shekhawatsamvardhan/types-of-neural-networks-convolutional-neural-networks-bd973e4fe78c.

[24] Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong y Wang-chun Woo. «Convolutional LSTM network: A machine learning approach for precipitation nowcasting». En: *Advances in neural information processing systems* 28 (2015).

[25] Rebeen Ali Hamad, Longzhi Yang, Wai Lok Woo y Bo Wei. «Joint learning of temporal models to handle imbalanced data for human activity recognition». En: *Applied Sciences* 10.15 (2020), pág. 5293.

[26] Skipper Seabold y Josef Perktold. *Web Statsmodels*. Última consulta: 04/07/24. url: https://datosclima.es/Aemethistorico/Meteosingleday.php.

[27] Wes McKinney. *Web Pandas*. Última consulta: 05/07/24. url: https://pandas.pydata.org/.

[28] Wes McKinney et al. «pandas: a foundational Python library for data analysis and statistics». En: *Python for high performance and scientific computing* 14.9 (2011), págs. 1-9.

[29] Skipper Seabold y Josef Perktold. «statsmodels: Econometric and statistical modeling with python». En: *9th Python in Science Conference*. 2010.

[30] François Chollet. *Web Keras*. Última consulta: 04/07/24. url: https://keras.io/.

[31] Francois Chollet. *Deep learning with Python*. Simon y Schuster, 2021.

[32] Google. *Web Tensorflow*. Última consulta: 04/07/24. url: https://www.tensorflow.org/?hl=es-419.

[33] National Geographic. *Qué es el calentamiento global*. Última consulta: 08/07/24. url: https://nationalgeographic.es/medio-ambiente/que-es-el-calentamiento-global.

[34] NASA. *Evolución temperatura global*. Última consulta: 10/07/24. url: https://climate.nasa.gov/en-espanol/signos-vitales/temperatura-global/?intent=111.

[35] Skipper Seabold y Josef Perktold. *Statsmodels Holt-Winters*. Última consulta: 04/07/24. url: `https://www.statsmodels.org/stable/generated/statsmodels.tsa.holtwinters.ExponentialSmoothing.html`.

[36] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever y Ruslan Salakhutdinov. «Dropout: a simple way to prevent neural networks from overfitting». En: *The journal of machine learning research* 15.1 (2014), págs. 1929-1958.

[37] Tensorflow. *Dense Layer*. Última consulta: 06/07/24. url: `https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dense`.

[38] Jason Brownlee. *A Gentle Introduction to the Rectified Linear Unit (ReLU)*. Última consulta: 08/07/24. url: `https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/`.

[39] Lei Huang, Jie Qin, Yi Zhou, Fan Zhu, Li Liu y Ling Shao. «Normalization techniques in training dnns: Methodology, analysis and application». En: *IEEE transactions on pattern analysis and machine intelligence* 45.8 (2023), págs. 10173-10196.

[40] Rob J Hyndman y Anne B Koehler. «Another look at measures of forecast accuracy». En: *International journal of forecasting* 22.4 (2006), págs. 679-688.

[41] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser e Illia Polosukhin. «Attention is all you need». En: *Advances in neural information processing systems* 30 (2017).

[42] Mingtian Tan, Mike A Merrill, Vinayak Gupta, Tim Althoff y Thomas Hartvigsen. «Are Language Models Actually Useful for Time Series Forecasting?» En: *arXiv preprint arXiv:2406.16964* (2024).

[43] Colin Lea, Michael D Flynn, Rene Vidal, Austin Reiter y Gregory D Hager. «Temporal convolutional networks for action segmentation and detection». En: *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, págs. 156-165.

[44] Taylor G Smith. *Librería PMDARIMA*. Última consulta: 05/07/24. url: `https://alkaline-ml.com/pmdarima/`.

[45] SAS. *PROC ARIMA*. Última consulta: 05/07/24. url: `https://documentation.sas.com/doc/en/etsug/15.2/etsug_arima_syntax02.htm`.

Apéndices

Apéndice A

Código Python para redes neuronales

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from keras.callbacks import EarlyStopping, ReduceLROnPlateau
from keras.models import Sequential
from keras.layers import Dense, Dropout, Input #Add capa o capas de cada
    arquitectura
from sklearn.metrics import mean_squared_error
import json


# Crear secuencias de entrenamiento y etiquetas
def create_sequences(data, seq_length, n_steps):
    sequences = []
    labels = []
    for i in range(len(data) - seq_length - n_steps + 1):
        sequences.append(data[i:i + seq_length])
        labels.append(data[i + seq_length:i + seq_length + n_steps])
    return np.array(sequences), np.array(labels)

#Funcion para predecir los siguientes N pasos.
def predict_future(model, initial_sequence, n_steps, seq_length):
    sequence = initial_sequence.copy()
    predictions = []
    for _ in range(n_steps):
        input_seq = sequence[-seq_length:].reshape((1, seq_length, 1))
        next_value = model.predict(input_seq, verbose=0)[0, 0]
        predictions.append(next_value)
        sequence = np.append(sequence, next_value)
    return predictions

if __name__ == "__main__":
    # Cargar los datos
    data = pd.read_csv('temp_data_valladolid.csv')
    data['date'] = pd.to_datetime(data['date'])
    data.set_index('date', inplace=True)
    temperature_data = data['mean_temp'].iloc[-365:].values.reshape(-1, 1)
```

```
37
38      # Normalizar los datos de temperatura
39      scaler = MinMaxScaler(feature_range=(0, 1))
40      data['mean_temp'] = scaler.fit_transform(data['mean_temp'].values.reshape
    (-1, 1))
41
42      data_mse = {}
43      for seq_length in [1, 2, 3, 4, 5, 6, 7, 10, 14, 28, 60, 90, 180, 365]:
44          best_mse = 10000
45          n_steps_mse={}
46          for n_steps in [1,5,7,14,28,60,180,365]:
47              sequences, labels = create_sequences(data[['mean_temp']].values,
    seq_length, n_steps)
48
49              # Dividir los datos en entrenamiento y prueba
50              split = data.index.get_loc('2022-01-01')  # 2022-01-01 es la
    fecha de inicio de los datos de prueba
51              X_train, y_train = sequences[:split], labels[:split]
52              X_test, y_test = sequences[split:], labels[split:]
53
54              # Crear el modelo
55              ################################################
56              #
57              #
58              # Aqui va la arquitectura del modelo
59              #
60              #
61              ################################################
62
63              model.compile(optimizer='adam', loss='mse')
64
65              # Reshape de los datos para que sean aceptados
66              X_train_reshaped = X_train.reshape((X_train.shape[0], X_train.
    shape[1], 1))
67              X_test_reshaped = X_test.reshape((X_test.shape[0], X_test.shape
    [1], 1))
68
69              # Configurar early stopping
70              early_stopping = EarlyStopping(monitor='val_loss', patience=10,
    restore_best_weights=True)
71              reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2,
    patience=5, min_lr=0.0001)
72
73              # Entrenar el modelo
74              history = model.fit(X_train_reshaped, y_train, epochs=100,
    batch_size=64, validation_data=(X_test_reshaped, y_test), callbacks=[
    early_stopping, reduce_lr], verbose=2)
75
76              # Evaluar el modelo
77              loss = model.evaluate(X_test_reshaped, y_test)
78
79              # Realizar predicciones
80              predictions = model.predict(X_test_reshaped)
81
82              # Desnormalizar los datos para obtener las temperaturas
```

```
originales
83              predictions = scaler.inverse_transform(predictions)
84              actual = scaler.inverse_transform(y_test.reshape(-1, n_steps))
85              mse = mean_squared_error(actual, predictions)
86              print(f"MSE (step {n_steps}): {mse}", file=f)
87              if mse < best_mse:
88                  best_mse = mse
89                  best_seq = seq_length
90                  best_n_steps = n_steps
91                  print(f"Best MSE: {best_mse} with {best_seq} days, {
    best_n_steps} n_steps", file=f)
92
93              # Predicciones futuras
94              last_sequence = data['mean_temp'].values[-seq_length:]
95              predicted = predict_future(model, last_sequence, n_steps,
    seq_length)
96              predicted_temp = scaler.inverse_transform(np.array(predicted).
    reshape(-1, 1))
97              y_true = temperature_data[-n_steps:].flatten()
98              y_pred = predicted_temp.flatten()
99              n_steps_mse[n_steps]=mean_squared_error(y_true, y_pred)
100
101             #Guardar modelo
102             model.save(f"RNN/model_rnn_{seq_length}_{n_steps}.keras")
103
104         data_mse[seq_length]={"model_mse":best_mse, "best_n_steps":
    best_n_steps, "n_steps_mse":n_steps_mse}
105
106         #Guardar resultados
107         with open("RNN/mse_rnn.json", "w") as file:
108             json.dump(data_mse, file)
109         file.close()
```

Listing A.1: Código general del entrenamiento las redes neuronales

Apéndice B

Código Python para modelo Holt-Winters aditivo con damping

```
import pandas as pd
import numpy as np
from statsmodels.tsa.holtwinters import ExponentialSmoothing

#Cargar datos
data=pd.read_csv('temp_data_valladolid.csv')
data['date']=pd.to_datetime(data['date'])
data.set_index('date', inplace=True)
data = data.asfreq('D')

train_data= data[data.index.year <2022]
test_data= data[data.index.year >=2022]

#Entrenar modelo
hw_model = ExponentialSmoothing(train_data['mean_temp'], seasonal='add',
    trend='add',seasonal_periods=365, damped_trend=True).fit()
hw_model.summary()
```

Listing B.1: Código Python Holt-Winters

Apéndice C

Código Python para modelos Box-Jenkins

```
import pandas as pd
import numpy as np
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.statespace.sarimax import SARIMAX

# Descomponer la serie temporal
decomposition = seasonal_decompose(data['mean_temp'], model='additive',
    period=365)

# Graficar los componentes
fig, (ax1, ax2, ax3, ax4) = plt.subplots(4, 1, figsize=(15, 12), sharex=True)
decomposition.observed.plot(ax=ax1, title='Serie Original')
decomposition.trend.plot(ax=ax2, title='Tendencia')
decomposition.seasonal.plot(ax=ax3, title='Estacionalidad')
decomposition.resid.plot(ax=ax4, title='Residuos')

plt.tight_layout()
plt.show()

#Cargar datos
data=pd.read_csv('temp_data_valladolid.csv')
data['date']=pd.to_datetime(data['date'])
data.set_index('date', inplace=True)
data = data.asfreq('D')
data = data[data.index.year<2022]

start_time = time.time()
print("Fitting SARIMA(3,0,0)(0,1,1,365) with drift")
model = SARIMAX(data['mean_temp'], order=(3, 0, 0), seasonal_order=(0, 1, 1,
    365), enforce_stationarity=True, enforce_invertibility=True)
results = model.fit(low_memory=True)

print("Time taken: ", time.time() - start_time)
print(f'AIC: {results.aic}')
print(results.summary())
print("\n\n")

```

```
36 #Guardar modelo
37 with open(f'Modelos/sarimax_model(3,0,0)(0,1,,,365)_w_drift.pkl', 'wb') as
       pkl_file:
38     pickle.dump(results, pkl_file)
```

Listing C.1: Código Python SARIMA

Apéndice D

Código Python para la extracción de datos meteorológicos

```
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.support.ui import Select
from selenium.webdriver.common.by import By
import time
import datetime
import pandas as pd
from bs4 import BeautifulSoup

# Opciones para el navegador (opcional)
options = webdriver.ChromeOptions()
options.add_argument('--headless')  # Para correr Chrome en modo headless (sin interfaz gráfica)

# Inicializa el WebDriver
driver = webdriver.Chrome(options=options)

# Abre la página web
driver.get('https://datosclima.es/Aemethistorico/Meteosingleday.php')
df=pd.DataFrame(columns=["Fecha","Temperatura Máxima","Temperatura Mínima"])
time.sleep(1)

Select(driver.find_element(By.NAME,"Provincia")).select_by_value("VALLADOLID")
Select(driver.find_element(By.NAME,"id_hija")).select_by_value("2422")
time.sleep(1)

fecha_inicio=datetime.date(1974,1,1)
fecha_fin=datetime.date(2024,4,1)
fecha=fecha_inicio
i=0

while fecha<=fecha_fin:
    try:
        driver.find_element(By.NAME, "Iday").clear()
```

```
34          driver.find_element(By.NAME, "Iday").send_keys(str(fecha.day))
35          driver.find_element(By.NAME, "Imonth").clear()
36          driver.find_element(By.NAME, "Imonth").send_keys(str(fecha.month))
37          driver.find_element(By.NAME, "Iyear").clear()
38          driver.find_element(By.NAME, "Iyear").send_keys(str(fecha.year))
39          driver.find_element(By.XPATH, '//*[@id="col2"]/div[1]/div/form/input
        [5]').click()
40          time.sleep(1)
41          page_source = driver.page_source
42          soup = BeautifulSoup(page_source, 'html.parser')
43
44          #Temperatura -> ºC
45          datos_dia={"Fecha":fecha,"Temperatura Máxima":"","Temperatura Mínima"
        :""}
46
47          tabla=soup.find("table", id="Resultados")
48          columna_actual=""
49          for row in tabla.find_all("tr"):
50              for cells in row.find_all("td"):
51                  texto=cells.text.replace(":","").strip()
52                  if columna_actual !="":
53                      try:
54                          datos_dia[columna_actual]=float(texto.split()[0])
55                          columna_actual=""
56                      except:
57                          datos_dia[columna_actual]=pd.NA
58                          columna_actual=""
59                  else:
60                      if texto in datos_dia.keys():
61                          columna_actual=texto
62
63          dato = pd.DataFrame([datos_dia])
64          df = pd.concat([df,dato], ignore_index=True)
65          fecha= fecha + datetime.timedelta(days=1)
66
67          i+=1
68          time.sleep(0.5)
69      except:
70          print(f"Fallo {fecha}")
71          driver = webdriver.Chrome(options=options)
72          driver.get('https://datosclima.es/Aemethistorico/Meteosingleday.php')
73          time.sleep(1)
74          #Recargar página
75          Select(driver.find_element(By.NAME,"Provincia")).select_by_value("
        VALLADOLID")
76          Select(driver.find_element(By.NAME,"id_hija")).select_by_value("2422"
        )
77          time.sleep(3)
78
79  df.to_csv("datos_meteo_valladolid.csv")
80
81  time.sleep(5)
82  # Cierra el navegador
83  driver.quit()
```

Listing D.1: Código Python web scraping