Proyecto Fin de Máster
Máster en Ingeniería Industrial

Previsión de la Demanda Mediante Deep Learning

Autor: Carlos De la Cruz Bailén

Tutor: Jose Manuel Framiñán Torres

Dpto. de Organización Industrial y Gestión de Empresas I
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2020

Proyecto Fin de Máster
Máster en Ingeniería Industrial

Previsión de la Demanda Mediante Deep Learning

Autor:
Carlos De la Cruz Bailén

Tutor:
Jose Manuel Framiñán Torres
Profesor Titular

Dpto. de Organización Industrial y Gestión de Empresas I
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2020

Proyecto Fin de Máster: Previsión de la Demanda Mediante Deep Learning

Autor: Carlos De la Cruz Bailén
Tutor: Jose Manuel Framiñán Torres

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

A mi familia por apoyarme siempre.

A mi tutor D.José Manuel Framiñán Torres por su ayuda y atención en todo momento.

Sevilla, 2020

Resumen

En el presente trabajo fin de máster, se considerará los problemas de previsión de la demanda tan relevantes en la estrategia y planificación de las empresas. Para ello, se propondrá diferentes modelos de inteligencia artificial mediante Deep Learning para aplicarlos a un caso práctico del sector automovilístico que busca predecir el número de ventas de un modelo de producto. Para el análisis de su aplicabilidad y desempeño, los experimentos computacionales se generarán en lenguaje Python para posteriormente analizar los resultados obtenidos y las diferentes comparativas mediante procesos de mejora al modelo.

Abstract

In this master's thesis, the problems of demand forecasting that are so relevant in the strategy and planning of companies will be considered. Different artificial intelligence models will be proposed using Deep Learning to apply them to a practical case of the automotive sector that seeks to predict the number of sales of a product model. For the analysis of its applicability and performance, the computational experiments will be generated in Python language to later analyze the results obtained and the different comparisons by means of improvement processes to the model.

Índice

Resumen III
Abstract V

Índice de Figuras IX
Índice de Tablas XI

1 Introducción 1
1.1 Introducción y objeto del proyecto 1
1.2 Sumario sobre la estructura del proyecto 2

2 Antecedentes 3
2.1 Introducción a la previsión de la demanda 3
2.1.1 Introducción 3
2.1.2 Métodos clásicos más populares 4
2.2 Formalización del problema 5
2.2.1 Descripción del problema 5
2.2.2 Objetivo del problema 7
2.3 Descripción del caso práctico 7

3 Metodología 11
3.1 Introducción al Deep Learning 11
3.2 Arquitecturas 14
3.2.1 Redes Neuronales Recurrentes 14
Redes LSTM 14
3.2.2 Redes Neuronales Convolucionales 15
3.3 Introducción a la Previsión de la Demanda en lenguaje Python 17
3.3.1 Introducción a Keras 17
3.4 Resumen y orientación de la metodología al proyecto 18

4 Modelado y aplicación al caso práctico 19
4.1 Tratamiento de Datos 19
4.1.1 Pasos y técnicas empleados 19
Extracción tipo de producto 19
Unificación 19
Interpolación periodos festivos 19
Limpieza de datos 20

Normalización 20
Problema de Datos Supervisados 20
K-Folding Cross Validation 21
4.2 Modelado computacional 23
4.3 Modelo 1. LSTM Univariable 23
4.3.1 Descripción del Modelo 23
4.3.2 Modelado Computacional 24
4.4 Modelo 2. CNN Univariable 25
4.4.1 Descripción del Modelo 25
4.4.2 Modelado Computacional 26
4.5 Modelo 3: LSTM Multivariable 27
4.5.1 Descripción del Modelo 27
Introducción Multivariable 27
Variables del Modelo 28
Procesos y descripción del modelo 29
4.5.2 Modelado Computacional 31
Bibliotecas externas implementadas 32
Lectura y limpieza de datos 32
Modelado de la función Normalizar 33
Definición datos de entrenamiento y validación 33
Modelado de la función Data_supervised 34
Construcción Modelo LSTM 35

5 Resultados 37
5.1 Resultados computacionales 37
5.2 Análisis experimental (Fitting) 41
5.2.1 Tratamiento de Overfitting 43
Técnica Dropout 44
5.2.2 Mejora multivariable al modelo y eficiencia frente a métodos clásicos 46
Eficiencia frente a métodos clásicos 49

6 Conclusiones 51

Apéndice A Codifiación en Python 53
A.1 Tratamiento de Datos 53
A.2 Modelo 1. LSTM Univariable 55
A.3 Modelo 2. CNN Univariable 60
A.4 Modelo 3. LSTM Multivariable 63
A.5 Holt-Winters 69
Bibliografía 71

Índice de Figuras

2.1 Modelo multivariable en un sistema de previsión [Fuente: Elaboración propia] 3
2.2 Tendencia, estacionalidad y ruido de una serie temporal arbitraria [Fuente: Elaboración propia] 4
2.3 Evolución temporal de variables correlacionadas y predicción a cuatro periodos [Fuente: Elaboración propia] 6
2.4 Rangos de entrenamiento y validación/testeo [Fuente: Elaboración propia] 6
2.5 Ventas del producto en el tiempo [Fuente: Elaboración propia] 7
2.6 Rangos de entrenamiento y validación/testeo para el producto a estudiar [Fuente: Elaboración propia] 8

3.1 Esquema Machine Learning e IA Simbólica [Fuente: Elaboración propia] 12
3.2 Red Neuronal de dos capas [Fuente: Elaboración propia] 12
3.3 Esquema Deep Learning [Fuente: Elaboración propia] 13
3.4 Esquema Red Neuronal Recurrente [Fuente:[28]] 14
3.5 Esquema LSTM [Fuente: [10]] 15
3.6 Esquema Redes Convolucionales [Fuente: Elaboración propia] 16

4.1 Instrucciones para el tratamiento de datos [Fuente: Elaboración propia] 20
4.2 Serie Temporal Producto [Fuente: Elaboración propia] 21
4.3 Esquema Problema Supervisado [Fuente: Elaboración propia] 21
4.4 Esquema K-Folding Cross Validation [Fuente: Elaboración propia] 22
4.5 Esquema adaptación K-Folding Cross Validation [Fuente: Elaboración propia] 22
4.6 Diagrama Modelo LSTM Univariable [Fuente: Elaboración propia] 23
4.7 Esquema CNN [Fuente: Elaboración propia] 25
4.8 Esquema de Multivariedad en los Datos del Modelo [Fuente: Elaboración propia] 27
4.9 Análisis de variables dependientes [Fuente: Elaboración propia] 28
4.10 Evolución temporal IBEX 35 en el rango de estudio [Fuente: Elaboración propia] 29
4.11 Evolución temporal Gasolina95 en el rango de estudio[Fuente: Elaboración propia] 29
4.12 Diagrama modelo LSTM Multivariable [Fuente: Elaboración propia] 30
4.13 Esquema Modelo LSTM Multivariable [Fuente: Elaboración propia] 31

5.1 Gráficos resultados computacionales por modelo: evolución del error por cada Epoch [Fuente: Elaboración propia] 40
5.2 Evolución LSTM Multivariable para 100 epochs [Fuente: Elaboración propia] 41

5.3 Evolución temporal datos pronosticados y reales [Fuente: Elaboración propia] 42
5.4 Representación temporal de los fenómenos Underfitting y Overfitting [Fuente: Elaboración propia] 44
5.5 Representación temporal del error de validación con/sin Dropout [Fuente: Elaboración propia] 45
5.6 Representación de picos de demanda en el año 2019 [Fuente: Elaboración propia] 46
5.7 Resultados computacionales del modelo [Fuente: Elaboración propia] 48
5.8 Evolución temporal datos pronosticados y reales [Fuente: Elaboración propia] 49
5.9 Evolución temporal datos pronosticados y reales (Holt-Winters) [Fuente: Elaboración propia] 50

Índice de Tablas

2.1 Métodos clásicos recomendados para cada tipo de serie temporal [Fuente: Elaboración propia] 5

3.1 Flujo Redes Neuronales Convulacionales [Fuente: Elaboración propia] 17

4.1 Parámetros seleccionados modelo LSTM Univariable [Fuente: Elaboración propia] 24
4.2 Parámetros seleccionados modelo CNN Univariable [Fuente: Elaboración propia] 26
4.3 Parámetros seleccionados modelo LSTM Multivariable [Fuente: Elaboración propia] 31
4.4 Datos de entrenamiento y validación de la variable ventas del producto [Fuente: Elaboración propia] 34

5.1 Resultados computacionales para cada modelo [Fuente: Elaboración propia] 39
5.2 Parámetros seleccionado del modelo LSTM Multivariable mejorado[Fuente: Elaboración propia] 48

1 Introducción

1.1 Introducción y objeto del proyecto

El proyecto que se presenta se enfoca en los problemas de previsión de la demanda tan importantes dentro de la planificación y estrategia de la empresa, donde se resolverá un problema real de estimación en el número de ventas de un producto del sector automovilístico. Se tomará como ejemplo este sector debido a la facilidad de búsqueda y análisis de otras variables que puedan ser utilizadas como entradas adicionales al problema para que mejoren el pronóstico a analizar y ejecutar mediante las correlaciones que presenten entre ellas.

Una previsión adecuada de la demanda ofrece a las empresas una información anticipada que las permita conocer su situación y potencial en diferentes mercados, para poder, de esta forma, anticiparse mediante toma de decisiones y estrategias que redirijan a la empresa a la adaptación y crecimiento dentro de los mercados. Por otro lado, en los últimos años se ha visto mayor uso de las metodologías de Inteligencia Artificial (IA) para la resolución de estos problemas debido a la gran cantidad de datos que pueden procesar mediante el uso de ecuaciones no lineales.

Teniendo esto en cuenta, el presente proyecto busca realizar una previsión de la demanda con métodos de IA que permitan una minimización del error en dicha previsión para maximizar la eficiencia de la resolución y la metodología. Para ello, se propone y desarrolla diferentes modelos Deep Learning que se resolverán y analizarán en profundidad para poder comparar su desempeño y resultados con las metodologías clásicas tan empleadas en los últimos años. Se implementarán tres modelos Deep Learning diferentes a lo largo del proyecto. En primer lugar, una estructura LSTM (Long Short Term Memory) y una estructura CNN (Convolutional Neural Network), ambos con enfoque univariable, esto es, se realizará una estimación mediante análisis de la única variable que se quiere predecir. El último modelo, busca, además de obtener unas correlaciones con el entrenamiento de observaciones pasadas de la propia variable ventas que se estudia, mejorar el pronóstico de forma que se incorporen variables que añadan patrones adicionales mediante una estructura Long Short Term Memory.

Con todo ello, el objetivo del proyecto es hacer uso de estos modelos de IA para reducir en mayor medida el error que se genera entre los valores reales de demanda y los valores pronosticados que aplican a nuestro caso práctico, realizando también un análisis de los resultados experimentales obtenidos en cada uno de ellos y poder así verificar si se produce un aumento de la eficiencia de los modelos Deep Learning frente a métodos clásicos. El modelado computacional se realizará mediante lenguaje Python, siendo este lenguaje de gran utilidad para el modelado y experimentación en problemas de inteligencia artificial, y, más concretamente, aquellos referidos a la previsión de

series temporales.

1.2 Sumario sobre la estructura del proyecto

El documento se desarrollará en seis capítulos y un apéndice que contiene los distintos códigos en Python. A continuación, se procederá a describir cada uno de los puntos que lo componen para facilitar la identificación de los mismos.

En primer lugar, el capítulo 1 "Introducción" presenta el objetivo general de proyecto. Posteriormente, se describen y resumen los distintos capítulos que componen el documento a modo de sumario.

El capítulo 2 "Antecedentes" expone una introducción a la predicción de la demanda junto a las características principales que las definen además de los métodos clásicos más empleados para su resolución. La descripción de la problemática identificada, así como el propósito que se busca para su mejora se describen también en este punto. Por último, se introduce la descripción del caso práctico en el sector automovilístico con el que trabajaremos para la aplicación de los métodos y modelos empleados a lo largo del documento.

En el capítulo 3 "Metodología" introduce las bases teóricas que se van a aplicar a nuestro caso práctico. Entre ellas, una introducción al Deep Learning debido a su papel principal en el pronóstico de la demanda en el proyecto, las arquitecturas que se emplearán para la aplicación de nuestros modelos de inteligencia artificial que describiremos y una introducción al lenguaje Python, donde se detallará la librería Keras para la modelización computacional. Por último se realizará un resumen con los puntos metodológicos más importantes y su aplicabilidad al desarrollo del proyecto.

En el capítulo 4 "Modelado y aplicación al caso práctico" se estudiarán y aplicarán los diferentes modelos introducidos asociado a la problemática descrita en el punto 2.2. Seguidamente, se trasladará el modelo aplicado de nuestro caso práctico al ámbito computacional, explicando los diferentes comandos auxiliares necesarios para el correcto modelado en lenguaje Python.

El capítulo 5 "Resultados" expondrán los diferentes resultados experimentales obtenidos junto con una evaluación de los mismos, para, posteriormente, analizando las características principales en los comportamientos de los modelos Deep Learning, proponer mejoras al modelo y comparaciones respecto a la eficiencia frente a otros modelos clásicos populares.

El capítulo 6 "Conclusiones" recoge las distintas conclusiones, valoraciones, y cuestiones abiertas que se han ido generando a lo largo del desarrollo del proyecto.

2 Antecedentes

A lo largo de este capítulo se establecerán las bases de la previsión de la demanda, así como los métodos clásicos más populares que se han utilizado en la empresa. Posteriormente, el punto 2.2 formalizará en detalle la descripción y objetivo de nuestro problema para, finalmente, describir el caso práctico con el que trabajaremos a lo largo del documento en el punto 2.3.

2.1 Introducción a la previsión de la demanda

2.1.1 Introducción

La previsión o forecast, definido según [11], es la predicción de un evento o eventos futuros. En un mercado global cada vez más competitivo, el empleo de técnicas de previsión de la demanda más precisas cobran un papel indispensable a la hora de planificar las actividades productivas dentro de la empresa, y, por tanto, los recursos que se requieren para satisfacer dicha demanda.

Según el análisis a emplear, se puede diferenciar dos formas de previsión: previsión subjetiva, métodos intuitivos que pueden no estar relacionados con los datos pasados y previsión cuantitativa, la cual supone que las características de tendencias de los datos del pasado continuarán en el futuro [7]. A lo largo del documento, trabajaremos con esta última mediante un enfoque de modelo de regresión. Un análisis de regresión estudia las relaciones entre diferentes variables, cuantificando la respuesta de una variable ante la variación de otras [7]. De este modo, con el objetivo de buscar una mayor precisión en la estimación de la demanda, el proyecto estudiará este enfoque de regresión mediante modelos multivariables, estos son, modelos que empleen múltiples variables de entrada analizadas a lo largo del tiempo para obtener como salida el valor de predicción. La figura 2.1 recoge el esquema multivariable en el sistema de previsión.

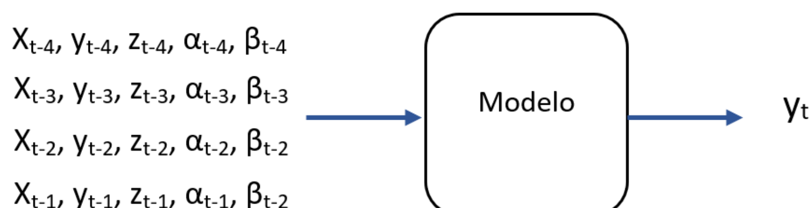

Figura 2.1 Modelo multivariable en un sistema de previsión [Fuente: Elaboración propia].

De igual forma, se estudiarán y analizarán el desempeño de los modelos univariables, aquellos

que emplean una única variable de entrada medida en el tiempo para obtener un valor de predicción futuro.

El horizonte de previsión se define como el número de periodos futuros a determinar analizando los datos pasados conocidos. Cuando el número de periodos es mayor que uno, hablamos de un modelo de múltiples pasos [14].

Un proceso relevante a la hora de seleccionar un método de previsión es el análisis de los componentes de la serie temporal [14].

1. Nivel: El valor medio de la serie si tuviese que ser representada mediante una recta.
2. Tendencia: Comportamiento lineal creciente o decreciente que sigue la serie temporal.
3. Estacionalidad: Patrones repetitivos o ciclos de comportamiento a lo largo del tiempo.
4. Ruido: Valores que no pueden ser explicados por el modelo de regresión.

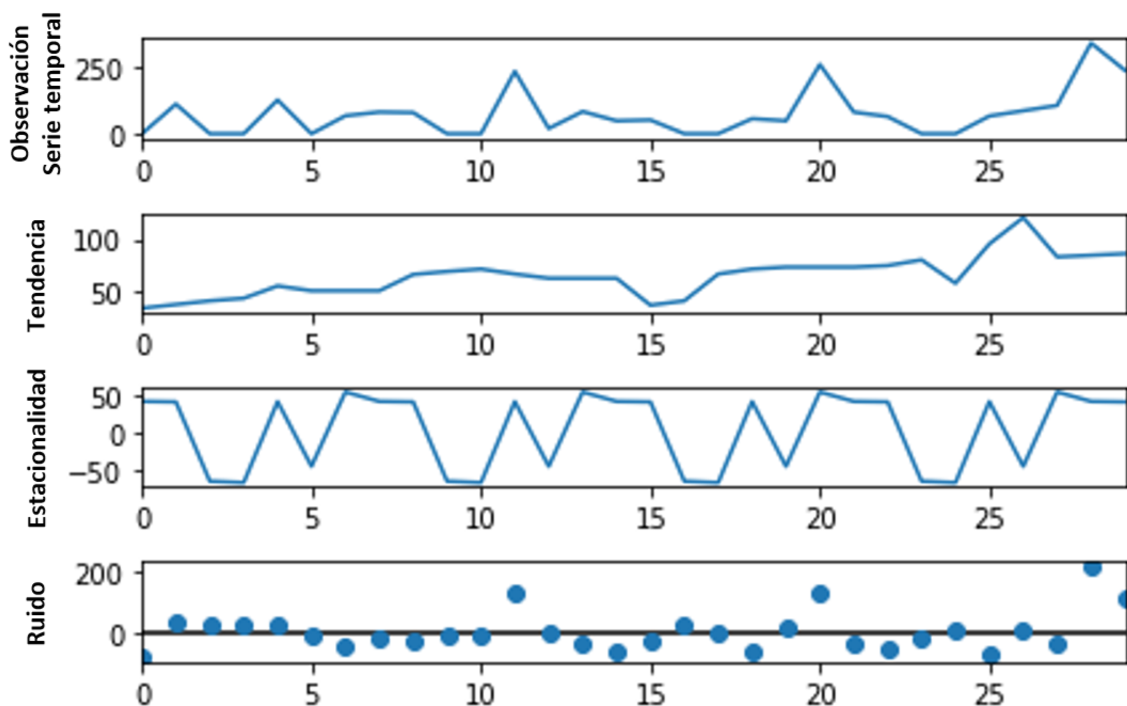

Figura 2.2 Tendencia, estacionalidad y ruido de una serie temporal arbitraria [Fuente: Elaboración propia].

Por otro lado, en términos de Inteligencia Artificial, la estimación se puede aproximar como el proceso de calibración del modelo (entrenamiento) mediante el uso de datos de entrada. Puesto que el proyecto dará una solución mediante metodología Deep Learning, la problemática será abordada bajo una ventana temporal de datos de largo plazo, definida por [7] como un estudio de valores pasados superior a un año y poder dar como resultado una previsión de los valores futuros.

2.1.2 Métodos clásicos más populares

Un método de previsión de la demanda es un algoritmo que proporciona una predicción puntual: uno o varios valores que se han calculado como una predicción para uno o varios periodos temporales

futuros (múltiples pasos) [21]. Por otro lado, un modelo estadístico trabaja con datos estadísticos para generar una predicción futura mediante distribuciones de probabilidad.

A la hora de pronosticar series temporales, existen numerosos métodos clásicos de previsión muy populares. En el desarrollo del documento, haremos referencia a los métodos clásicos como aquellos métodos cuantitativos ampliamente utilizados hasta la fecha que no atienden a una metodología de Inteligencia Artificial.

La elección de un método concreto para un determinado tipo de serie temporal puede suponer el éxito del pronóstico. Se pueden identificar dos tipos de series temporales según las correlaciones existententes entre los datos temporales:

1. Datos con tendencia y/o estacionalidad: bien con tendencia creciente o decreciente, por un lado, y estacionalidad aditiva o multiplicativa por otro.
2. Datos estacionarios: serie temporal donde los datos no siguen una correlación en el tiempo, no existiendo por tanto tendencia ni estacionalidad.

El éxito del método empleado será mayor para aquellos que se adapten mejor a las series temporales y a su naturaleza. La tabla 2.1 muestra los métodos clásicos más recomendados en función del tipo de serie temporal.

Tabla 2.1 Métodos clásicos recomendados para cada tipo de serie temporal [Fuente: Elaboración propia].

Tendencia sin estacionalidad	Estacionalidad sin tendencia	Tendencia y estacionalidad	Serie estacionaria
Doble suavizado exponencial	Estacionalidad aditiva	Holt-Winters' Aditivo	ARMA
ARIMA	Estacionalidad multiplicativa	Holt-Winters Multiplicativo	Suavizado de Media Móvil
-	-	ARIMA Estacional	Suavizado Exponencial Simple

Para mayor más información de cada uno de los modelos: [21], [19].

2.2 Formalización del problema

2.2.1 Descripción del problema

Para la formalización del problema que se aborda en nuestro proyecto, se considera un problema de previsión de la demanda con una ventana temporal conocida.

El problema abarcará variables adicionales de entrada correlacionadas entre sí además de aquella a la que se le realizará el pronóstico, partiendo de la base de que la naturaleza de la serie temporal de cada una de ellas puede ser arbitraria (con tendencia y/o estacionalidad, estacionaria, etc). A continuación, se muestra gráficamente la evolución temporal de cuatro variables arbitrarias de diferente naturaleza y la predicción de cuatro periodos para una de ellas:

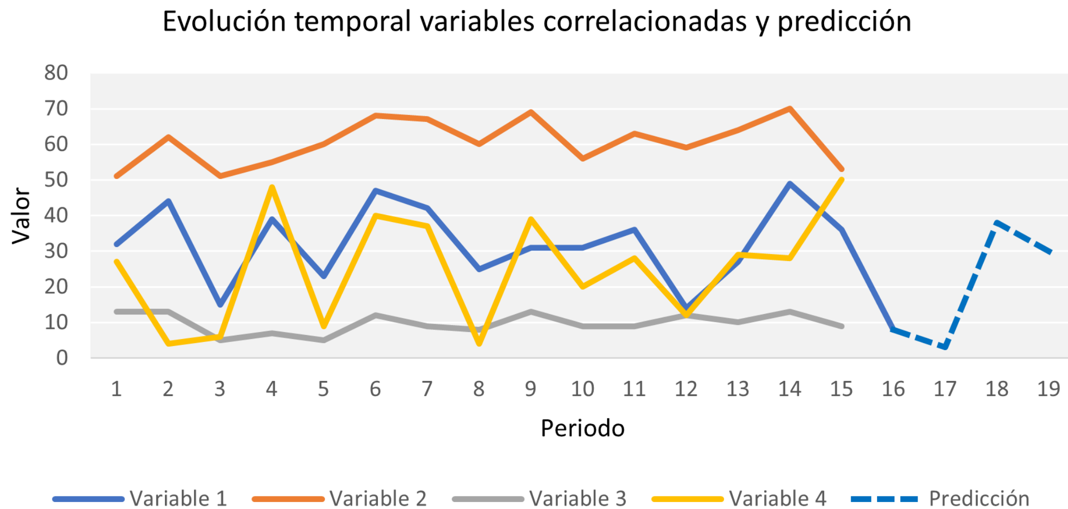

Figura 2.3 Evolución temporal de variables correlacionadas y predicción a cuatro periodos [Fuente: Elaboración propia].

A lo largo de la ventana de tiempo, se diferenciarán dos franjas temporales: rango de entrenamiento y rango de validación/testeo. El rango de entrenamiento servirá como análisis de cada una de las variables para la búsqueda de correlaciones internas, mientras que el rango de validación/testeo comprobará que las asunciones hechas a lo largo del entrenamiento son correctas, mostrando además las desviaciones correspondientes que aplican mediante el error generado.

Volviendo con el problema abstracto, se muestra a continuación dos franjas en periodos cualesquiera:

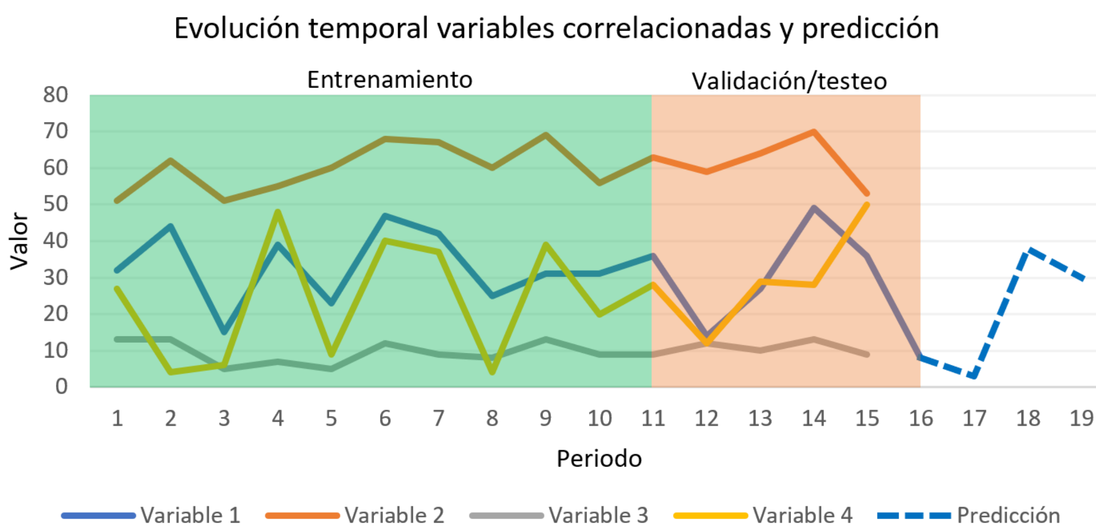

Figura 2.4 Rangos de entrenamiento y validación/testeo [Fuente: Elaboración propia].

Para mayor facilidad de modelado, se asumirán los siguientes puntos para nuestro problema:

1. El modelo trabajará bajo una ventana de planificación de tamaño T dividido en periodos de igual longitud correspondiendo un periodo por día.
2. La previsión de la demanda será aplicada a una sola variable, siendo redundante para el resto.

3. El testeo para la comprobación del desempeño a la hora de pronosticar la demanda se realizará en la fase de validación.
4. Al tratar con diferentes tipos de datos de entrada, se trabajarán con variables naturales y reales.
5. El modelo trabajará con aquellas variables que tengan periodicidad diaria para mayor desempeño del mismo.
6. Se normalizarán aquellas variables no binarias para mayor desempeño del modelo.
7. El modelo contempla un rango diferente de valores para cada una de las variables de entrada.
8. Se asumirá en todo momento un rango positivo de valores [0,infinito] para cada una de las variables.

2.2.2 Objetivo del problema

El propósito de nuestro problema atiende al siguiente objetivo:

1. Minimizar el error de validación: error generado entre los valores pronosticados y los reales en el rango de validación.

Esta función de minimización supone un gran impacto frente a métodos clásicos, ya que estos últimos pueden tener más dificultad a la hora de recoger patrones y comportamientos internos entre variables internas.

2.3 Descripción del caso práctico

Una vez introducido y formalizado el problema, procederemos a la descripción del caso práctico en el que se basa el presente trabajo.

En primer lugar, la variable que se estudiará para el pronóstico/previsión de la demanda se basa en número de ventas diarias de un modelo de producto del sector automovilístico cuyos datos se han obtenido de información pública proporcionado en la web mediante el número de nuevas matriculaciones diarias en España. A continuación, se puede ver gráficamente la evolución de las ventas del modelo de producto.

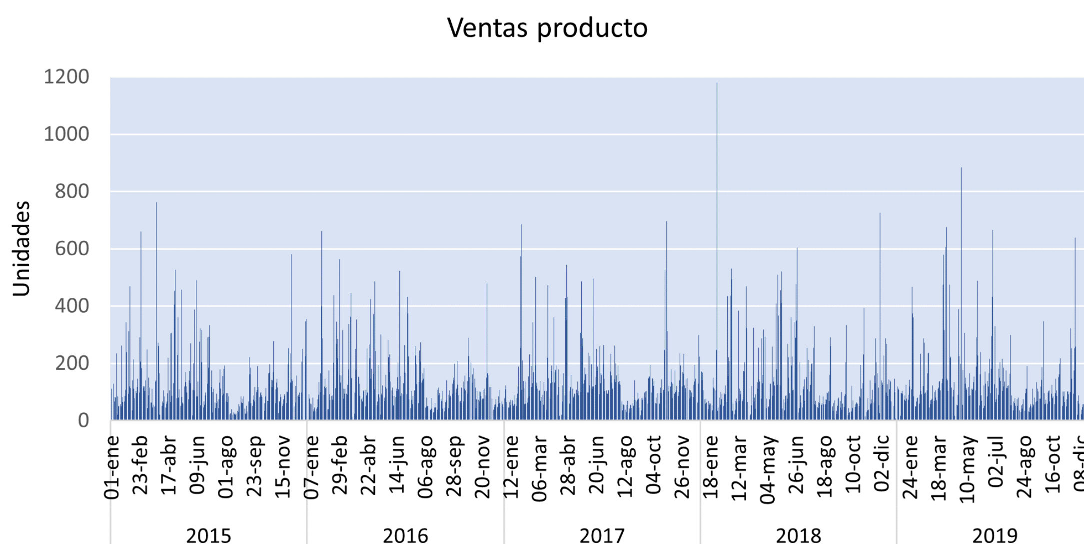

Figura 2.5 Ventas del producto en el tiempo [Fuente: Elaboración propia].

Observando la figura 2.5, se pueden apreciar los siguientes puntos:

1. Los datos se han recogido para una ventana temporal de cinco años comenzando desde el año 2015 hasta el año 2019 inclusive.
2. Los días festivos no computan ningún valor puesto que no se registran matriculaciones en la base de datos. Este punto se tendrá en cuenta para el desarrollo del proyecto.
3. Cabe destacar la existencia de picos de demanda para cada uno de los años cuya causa, a priori, se desconocen, pero se deberán tener gran consideración para el adecuado pronóstico de los valores.
4. Como primer análisis, no se aprecia ninguna tendencia en la serie temporal, pero si una estacionalidad semanal.

En segundo lugar, se realizará una búsqueda de variables adicionales que muestren correlaciones con los datos de ventas del modelo de producto, permitiendo así realizar un mejor pronóstico en términos de minimización de error. En esta búsqueda, entrarán aquellas variables cuya periodicidad temporal sea mayor o igual a la del producto que se analiza, en este caso, mayor o igual a días. Este se debe a que una periodicidad semanal, mensual, trimestral o anual implicaría un descuadre de número de variables de entrada al modelo, reduciendo así el desempeño del mismo ya que se deberían tomar hipótesis agresivas tales como interpolaciones.

A lo largo de la ventana temporal, para cada una de las variables de estudio que se implementen en el problema, se tendrá como rango de entrenamiento los periodos comprendidos desde el año 2015 hasta el año 2019 inclusive, mientras que el periodo 2019 se atribuirá al rango de validación/testeo. El éxito de nuestro problema dependerá en gran medida de la cantidad de datos que se tengan para entrenar el modelo, a mayor número de datos de entrenamiento, mayores correlaciones se podrán encontrar, aumentando así la exactitud de la previsión de la demanda.

A continuación, se muestra las franjas de periodos considerados en nuestro caso práctico:

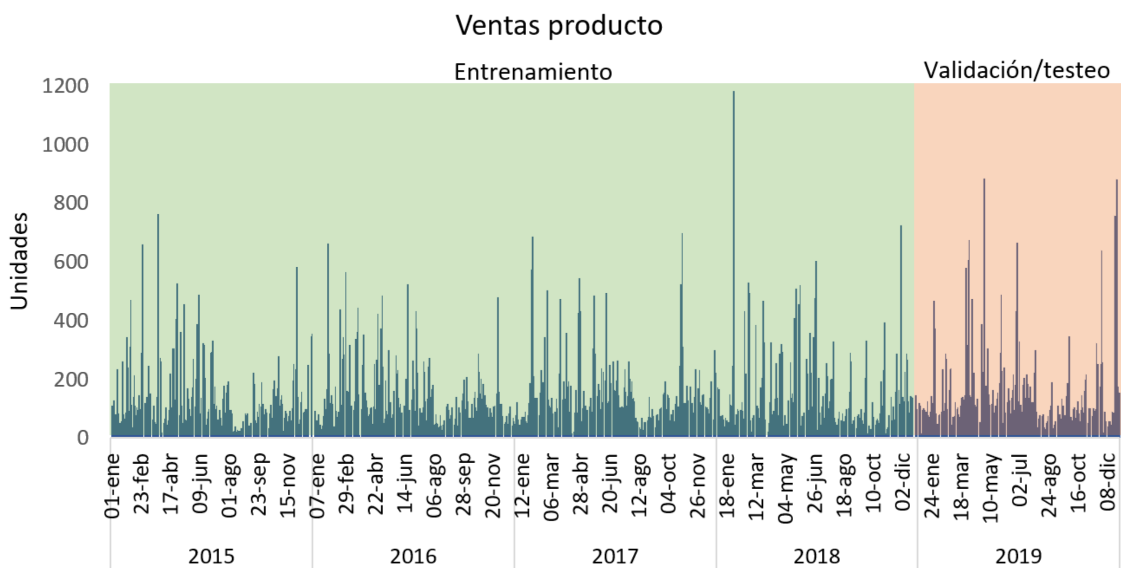

Figura 2.6 Rangos de entrenamiento y validación/testeo para el producto a estudiar [Fuente: Elaboración propia].

Puesto a que pueden aparecer problemas computacionales debido a la posible existencia de diferentes rangos de valores en las variables del modelo, se ha decidido trabajar con valores normalizados cuya media y desviación típica se calcularán en función de los datos que reflejen cada una de las variables.

3 Metodología

Una vez introducidas las series temporales orientadas a la previsión de la demanda, sus métodos clásicos más populares y, ante todo, descrita la problemática y descripción de nuestro caso práctico, se procederá con la metodología mediante un carácter teórico requerida para resolver el problema descrito en el apartado 2.2. Debido a que nuestro caso será resuelto mediante métodos Deep Learning, haremos una breve introducción a los mismos en el apartado 3.1. Posteriormente, definiremos las bases teóricas de los métodos de resolución que vamos a emplear para realizar nuestra previsión en el apartado 3.2. En el punto 3.3, trasladaremos estos modelos a un modelo computacional en lenguaje Python mediante la librería Keras. Por último, se hará un resumen de la metodología introducida y se especificará su aplicabilidad al desarrollo del proyecto en el punto 3.3.

3.1 Introducción al Deep Learning

La Inteligencia Artificial (IA) es la ciencia e ingeniería que permite hacer las máquinas inteligentes [20] con el objetivo de imitar la inteligencia humana caracterizada por comportamientos como la capacidad cognitiva, la memoria, el aprendizaje y la toma de decisiones [24]. Esta ciencia, ha sido considerada hasta los años 80 como una IA simbólica, permitiendo establecer reglas mediante código duro que manipulan el conocimiento de la máquina.

A pesar de que la IA simbólica ha permitido resolver varios problemas lógicos, no ha sido suficiente para hacer frente a problemas más complejos, tales como reconocimiento de voz, visión artificial o traducción del lenguaje [20]. De esta forma, surge el Machine Learning. El Machine Learning (ML) es un campo de estudio que proporciona la capacidad de un sistema para adquirir e integrar conocimientos a través de observaciones a gran escala con el fin de aprender en lugar de programar esos conocimientos [6]. De esta forma, permite obtener como salida unas reglas fruto de la entrada de datos y respuestas del modelo, mientras que la IA simbólica parte de unos datos y unas reglas para obtener unas respuestas.

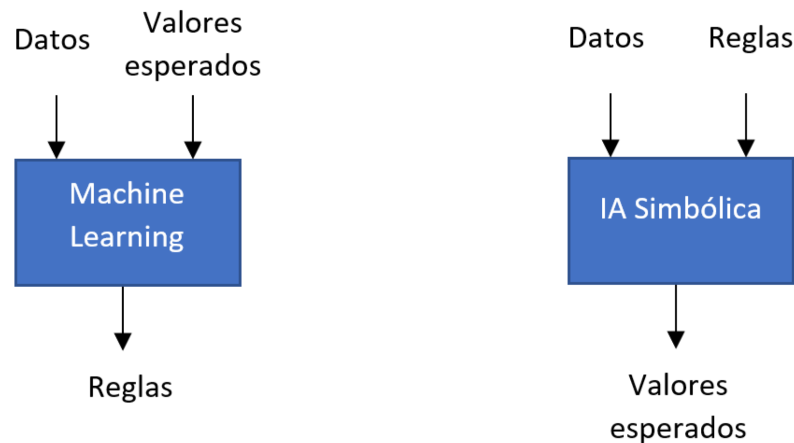

Figura 3.1 Esquema Machine Learning e IA Simbólica [Fuente: Elaboración propia].

La Inteligencia Profunda (Deep Learning) se define como un sub-campo del Machine Learning, el cual utiliza múltiples capas para extraer progresivamente características de alto nivel de la entrada a partir de datos en crudo mediante reglas/capas [16], atribuyéndole el nombre Deep para resolver problemas más complejos de forma no lineal.

La arquitectura más empleada en Deep Learning son las redes neuronales. Una red neuronal es una red interconectada de elementos simples procesadores, llamados nodos o unidades, cuyo funcionamiento simula a una neurona animal. La capacidad de procesamiento de la red se almacena en las conexiones entre unidades, o hiperparámetros llamados pesos, obtenidos mediante un proceso de adaptación o aprendizaje de un conjunto de patrones de entrenamiento [13].

La figura 3.2 muestra una red neuronal simple de dos capas con dos nodos por capa.

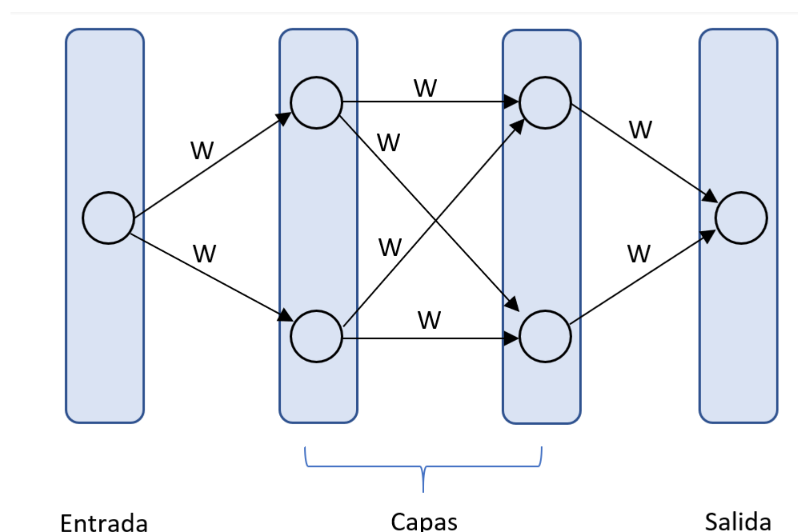

Figura 3.2 Red Neuronal de dos capas [Fuente: Elaboración propia].

Por otro lado, a modo introductorio, el diagrama de la figura 3.3 refleja el procedimiento de

aprendizaje de una red neuronal. Una vez la entrada a pasado por las diferentes capas que procesan la información, se obtiene una salida Y inicial y se compara con el valor real esperado. El modelo busca minimizar una función error para posteriormente realizar el proceso de Back Propagation, donde los pesos serán actualizados de forma que minimice ese error en la siguiente iteración.

A continuación, se definirán una serie de hiperparámetros característicos de las redes neuronales además de los comentados y su aplicabilidad durante el aprendizaje de la red neuronal:

1. Muestra: Serie de datos que compone por un lado la entrada a la red y la salida real Y empleada para comparar el valor real esperado. La entrada X puede estar compuesta por una o más muestras.
2. Batch: Número de muestras que va a recibir la red neuronal como entrada antes de realizar la actualización de los parámetros o pesos.
3. Epoch/iteración: Se produce al completar la actualización de los pesos de todas las muestras de la entrada, y por tanto, de la completitud de los datos de entrenamiento de la red neuronal.
4. Ratio de aprendizaje (Lr): Parámetro que define la velocidad de aprendizaje de la red.
5. Optimizador: Algoritmo encargado en la minimización del error por cada epoch que recibe como argumento una serie de hiperparámetros, entre ellos, los pesos y el ratio de aprendizaje.
6. Función de activación: Función que permite acotar las salidas de cada uno de los nodos además de permitir funciones complejas entre variables de la red neuronal. Debido al uso a lo largo del proyecto por la propiedad que otorga al modelo en cuanto a facilidad de entrenamiento y buenos resultados, cabe destacar la función ReLU, la cual devuelve el valor de entrada si es positivo y cero en cualquier otro caso.

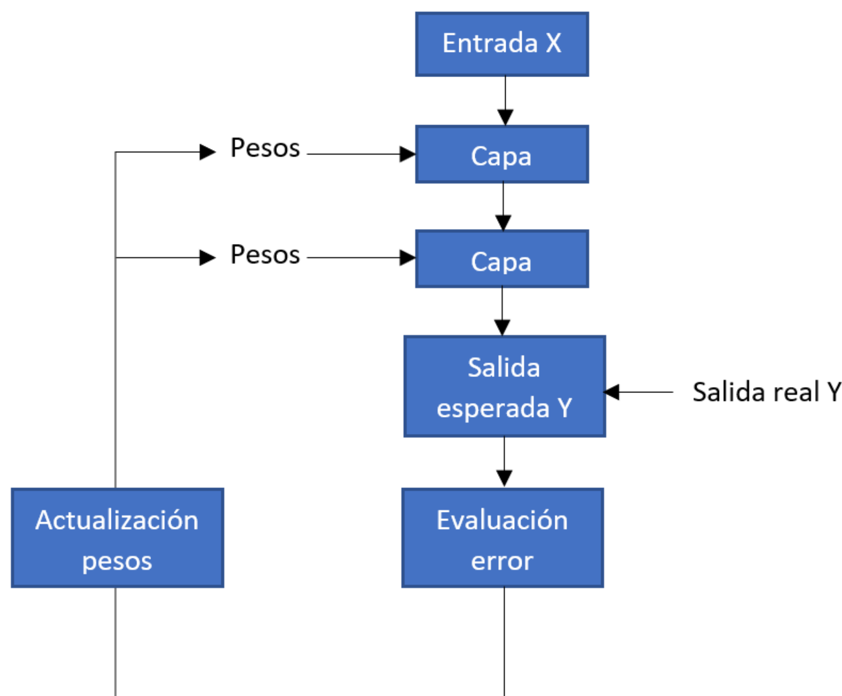

Figura 3.3 Esquema Deep Learning [Fuente: Elaboración propia].

3.2 Arquitecturas

Una arquitectura dentro del contexto Deep Learning se puede definir como un sistema de red neuronal que presenta una morfología y propiedades concretas con el fin de obtener unos resultados más eficientes para la naturaleza del problema que se presenta.

3.2.1 Redes Neuronales Recurrentes

Una Red Neuronal Recurrente (RNN) pertenece a una clase de redes neuronales artificiales donde las conexiones entre unidades forman un ciclo dirigido [25]. De este modo, las células pertenecientes a la red son retroalimentadas formando ciclos, permitiendo así una memoria interna [15]. Este tipo de redes, permiten ser ideales para problemas donde el tiempo es un factor indispensable a la hora de su resolución. La figura 3.4 refleja la arquitectura de una red recurrente en forma bucle y desarrollada en el tiempo.

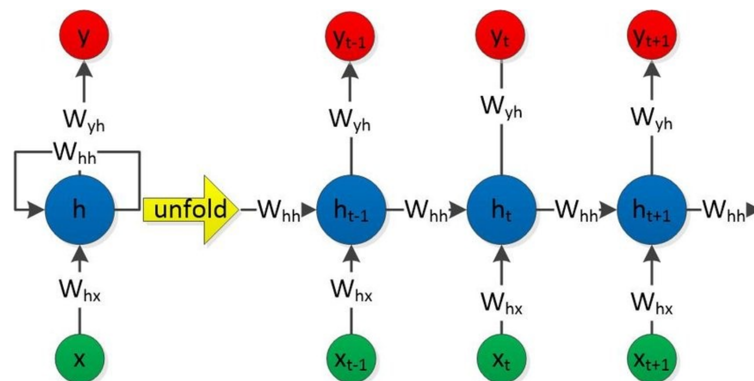

Figura 3.4 Esquema Red Neuronal Recurrente [Fuente:[28]].

Como se puede apreciar en la figura anterior, la red neuronal recibe dos argumentos como entradas con sus correspondientes matrices de pesos W_{hx} y W_{hh}:

1. Variable propia de entrada a la red neuronal.
2. Variable salida generada en la iteración anterior.

De este modo, la siguiente iteración recibirá una matriz de pesos que se comportará como una memoria generada en las pasadas iteraciones y poder realizar así una previsión basada en datos actuales y datos pasados. Esta arquitectura establece en una red neuronal una memoria a lo largo del tiempo, sin embargo, presentan una problemática conocida como desvanecimiento del gradiente desarrollado por [27] y cuyo aprendizaje basado en iteraciones anteriores queda limitado debido a que la red neuronal recurrente no es capaz de retener en memoria grandes cantidades de datos generados en iteraciones pasadas, limitando así su aprendizaje en el tiempo. Con el fin de mitigar este problema, aparecen la arquitectura Long Short Term Memory, también conocida como LSTM, introducidas por [22]. Puesto que se van a emplear estas redes a lo largo del proyecto, conviene definir debidamente su estructura interna para poder comprender la filosofía de los métodos a emplear.

Para mayor información y conocimiento: [5].

Redes LSTM

Una red LSTM es una arquitectura de red neuronal recurrente utilizada en el campo del aprendizaje profundo [22] donde en cada celda se realizan una serie de operaciones con el objetivo de mitigar el problema de aprendizaje a largo plazo que presentan las Redes Neuronales Recurrentes:

1. Estado de la celda: Variable que almacena en memoria los datos más relevantes en el tiempo.

2. Puerta de olvido: Operación que decide qué información es relevante y cuál no respecto a la información de entrada, en este caso, datos de entrada del periodo concreto y los datos de salida del periodo anterior.

3. Puerta de entrada: Operación cuya misión es la de actualizar valores de relevancia para añadirlos al estado de la celda.

4. Actualización estado de la celda: El vector estado de la celda se multiplica por el vector de olvido y el vector de entrada para finalmente pasar por la puerta de salida y tomar los datos relevantes de la celda actual.

5. Puerta de salida: La puerta de salida, al igual que las redes recurrentes, proporciona como salida el output de la celda o como se denomina normalmente estado oculto, que será entrada de la celda siguiente.

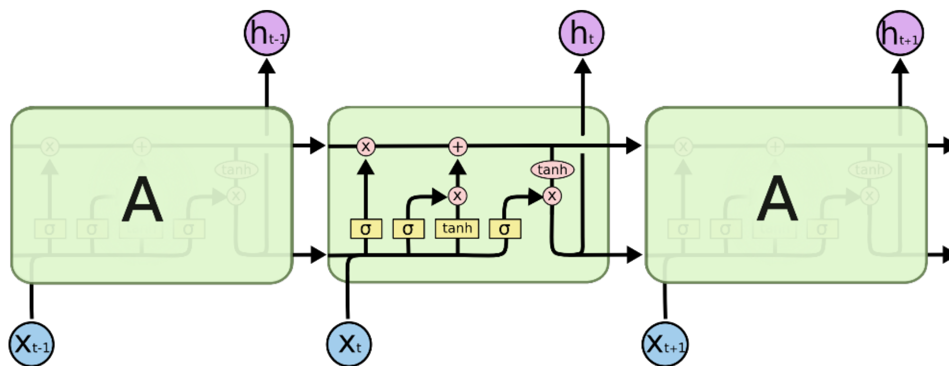

Figura 3.5 Esquema LSTM [Fuente: [10]].

Cada una de las puertas de la celda contienen una red neuronal con una función de activación sigmoidal.

3.2.2 Redes Neuronales Convolucionales

Las redes neuronales convolucionales, conocidas también como CNN, son un tipo de redes neuronales que se emplean principalmente para la detección de patrones en conjuntos de datos de entrada. La principal diferencia de las redes neuronales comunes radica en el uso de filtros en lugar de neuronas que son convolucionados con la entrada de cada capa [23]. Este tipo de redes se emplearán a lo largo del proyecto para la detección de patrones y correlaciones en datos pasados, siendo conveniente, por tanto, realizar una introducción más precisa del flujo y los procesos que conforman.

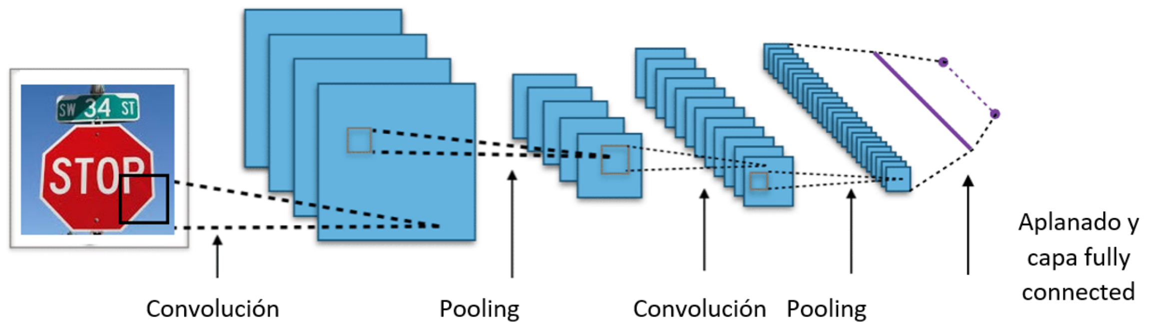

Figura 3.6 Esquema Redes Convolucionales [Fuente: Elaboración propia].

En primer lugar, vamos a definir una serie de términos:

1. Filtro/Kernel: Matriz compuesta por una serie de números enteros que actúan como pesos. Estos pesos se irán actualizando conforme el modelo vaya aprendiendo. La inicialización de los mismos se puede asignar de forma arbitraria.

2. Convolución: Proceso por el cual una entrada es filtrada. En términos matemáticos, la matriz filtro recorre la entrada mediante multiplicación para dar una salida.

3. Stride: Número de desplazamientos que realiza el filtro durante el proceso de convolución por cada tiempo.

4. Padding: Proceso que aplica píxeles de valor cero en cada convolución con motivo de obtener matrices de mismo tamaño o para recoger información relevante en los datos situados en las esquinas de las matrices.

5. Pooling: Para evitar exceso de información irrelevante a lo largo del flujo, el proceso pooling permite recoger los patrones más importantes.

6. Aplanado: Matriz 1D que recoge todos y cada uno de los diferentes patrones que se obtienen mediante los diferentes filtros aplicados

Una vez definidos los términos, vamos a introducir el flujo en una red convolucional básica con una sola capa convolucional mediante la tabla 3.1.

Tabla 3.1 Flujo Redes Neuronales Convulacionales [Fuente: Elaboración propia].

Proceso	Descripción
Preparación de datos	Normalización y preparación de los datos de entrada para que puedan ser leídos y tratados de forma más uniforme en la red.
Definición de filtro	Definimos dimensionalidad de la matriz, número de filtros n a aplicar e inicialización de los valores.
Convolución	Proceso de convolución obteniendo un número n de patrones con misma dimensionalidad que la entrada de datos.
Pooling	Aplicación del proceso pooling a cada uno de los patrones.
Aplanado	Se obtiene una matriz monodimensional de todos los patrones obtenidos.
Capa fully connected	Implementación de una capa fully connected al igual que se aplica en redes neuronales comunes para identificar las salidas relevantes.
Salida	Una vez se aplica una función de activación obtenemos la salida esperada para compararla con la real (gradient descendent method). Primera iteración de la red completada.

Para mayor información, [23] y [26] detallan de manera matemática cada uno de los procesos.

3.3 Introducción a la Previsión de la Demanda en lenguaje Python

Debido a los múltiples beneficios que proporciona Python tales como facilidad de uso, legibilidad de código, naturalidad de escritura y más importante: el potencial en Deep Learning gracias a distintas bibliotecas como Keras o TensorFlow, optaremos por dicho lenguaje de programación para el modelado de nuestro problema y su resolución.

3.3.1 Introducción a Keras

Para el modelado y resolución de los diferentes modelos Deep Learning que se emplearán a lo largo del proyecto, haremos uso de Keras. Keras es una Interfaz de Programación de Aplicaciones empleada para el modelado y entrenamiento de modelos Deep Learning que se ejecuta mediante la plataforma TensorFlow [2], una biblioteca de código abierto orientada al aprendizaje automático desarrollada por Google.

Keras incluye comandos para la definición de los diferentes bloques Deep Learning: número de capas neuronales, número de neuronas, función objetivo, función de activación y optimizadores entre otros. Esta biblioteca tiene un gran potencial en términos de versatilidad y configuración interna para todo tipo de problemas, implementándose en competiciones Deep Learning de gran prestigio como M Competitions.

A continuación, se describen las características que hacen de esta biblioteca, una de las mejores herramientas para problemas Deep Learning:

1. Permite un fácil uso mediante bloques previamente definidos, incentivando una rápida modelación y velocidad de entrenamiento del modelo.

2. Keras soporta arquitecturas arbitrarias tales como modelos de múltiples pasos, problemas multivariables y conexiones especificas entre capas neuronales entre otras, permitiendo una versatilidad ante un gran número de problemas de diferente clase [12].
3. Conversión de modelos a otras plataformas: Keras dispone de una mayor implementación y variedad de otras plataformas que otras API's de fuente abierta [2].
4. Capacidad de personalización: El modelo de código abierto de Keras, proporciona una alta personalización a la hora de modelar las arquitecturas Deep Learning frente a otras, pudiendo definir diferentes tipos de funciones objetivos, optimizadores o número de capas entre otros.

3.4 Resumen y orientación de la metodología al proyecto

A modo de resumen, el presente capítulo ha establecido en primer lugar las bases Deep Learning, donde su principal cualidad radica en el empleo de múltiples capas en una red neuronal para extraer características de alto nivel. En segundo lugar, se ha introducido la filosofía y conceptos de las Redes Neuronales Recurrentes, una arquitectura de red neuronal que incorpora memoria interna de datos pasados, y, posteriormente, se ha detallado la arquitectura LSTM, un tipo de red recurrente que solventa la problemática del desvanecimiento del gradiente generada en las primeras. En tercer lugar, se ha introducido el flujo y enumerado los procesos que incorporan las Redes Neuronales Convolucionales, cuya naturaleza se basa en la detección de patrones mediante filtros y convoluciones sucesivas para ser posteriormente procesados en capas neuronales. Finalmente, se han definido los beneficios y características más importantes que hacen del lenguaje Python y la biblioteca Keras una herramienta de gran potencial para el modelado e implementación de problemas Deep Learning.

Respecto a la aplicabilidad de la metodología al desarrollo del proyecto, se harán uso de las arquitecturas LSTM y CNN para aplicar una serie de modelos basados en las mismas que solventen el problema aplicado al caso práctico descrito en el apartado 2.2 mediante lenguaje Python y la implementación de la biblioteca Keras.

4 Modelado y aplicación al caso práctico

Comprendidos los conceptos que rodean nuestra problemática, este capítulo aborda la aplicación de la metodología al caso práctico. En primera instancia, se realiza un tratamiento de datos de entrada a cada modelo en el apartado 4.1, para posteriormente aplicar cada uno de los modelos a nuestro problema junto con su correspondiente modelado computacional en Python.

4.1 Tratamiento de Datos

Previo al desarrollo de cada uno de los modelos Deep Learning, se requiere un tratamiento en los datos que permita normalizar, simplificar y adaptar los datos de entrada para que la red neuronal pueda trabajar con los mismos.

A continuación, se detallarán los pasos y técnicas empleados.

4.1.1 Pasos y técnicas empleados

Extracción tipo de producto

En primer lugar, se parte de unos archivos *.txt* llamados *Export_mensual_matXXXXXX.txt* que contienen un número considerable de datos, donde aparecen todas las matriculaciones de turismos desde el año 2015 hasta el año 2020. Por tanto, se recogerán los datos únicamente que pertenezcan al vehículo marca y modelo que vamos a realizar nuestro análisis, para ello, para cada mes se generará un archivo *.txt* que contenga estas matriculaciones.

Cabe destacar que los datos de matriculaciones extraídos no contemplan los fines de semana ni los días festivos, por lo que deberá tenerse en cuenta en los próximos pasos de nuestro modelo.

Unificación

Una vez obtenidos las matriculaciones para el producto a analizar, unificamos todos los meses en un archivo *.txt* llamado *Marca_Modelo.txt*.

Interpolación periodos festivos

Debido a que los modelos de predicción de la demanda mediante Deep Learning normalmente no trabajan adecuadamente con valores nulos, se propone realizar una interpolación concreta para 3 casos diferentes para ver si es necesario aplicar el proceso de interpolación analizando los resultados:

Caso 1: Valor nulo en i y valor positivo en $i+1$ (día festivo corriente).

Para este caso se obtendrá el cociente de la división del valor $i+1$ positivo entre 2 y se le asignará al día i para eliminar la nulidad, para posteriormente asignar al día $i+1$ la diferencia entre el valor inicial de ese mismo día y el resultado del cociente.

Caso 2: Valores nulos en i e $i+1$ y valor positivo en $i+2$ (fines de semana y festivos dobles).
Se obtendrá el cociente de la división del valor en $i+2$ entre 3 para atribuírselo al día i. Una variable que se llamará n almacenará la diferencia entre ese resultado del cociente y el valor de $i+2$. El valor de $i+1$ se calculará como el valor del cociente entre n y 2 y finalmente $i+2$ será la diferencia los nuevos valores $i+2$ e i.

Caso 3: Valor nulo en i y valor unitario en $i+1$.
No se realizan operaciones de interpolación.

La figura 4.1 muestra a modo de esquema los pasos e instrucciones realizados en el tratamiento de datos para mayor facilidad de entendimiento.

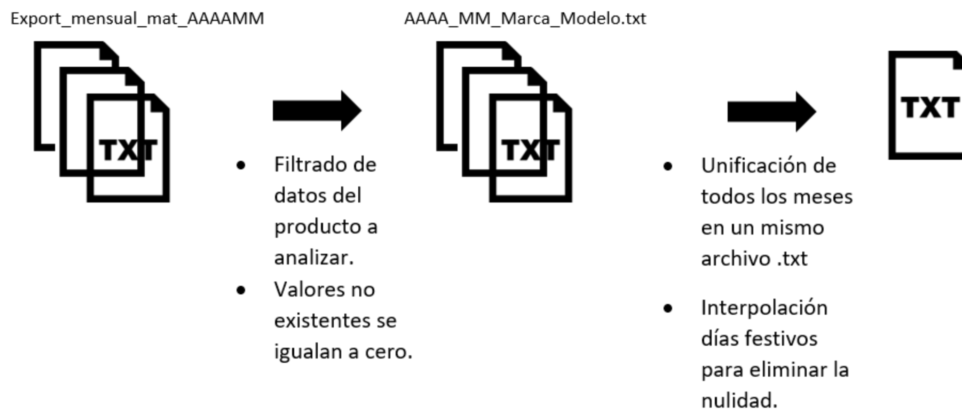

Figura 4.1 Instrucciones para el tratamiento de datos [Fuente: Elaboración propia].

Limpieza de datos

Con motivo de facilitar el procesamiento de los datos de entrada en modelos Deep Learning, se requiere hacer una limpieza previa que incluya los siguientes procesos: comprobar que no existen valores NaN (valores no existentes) en la entrada, asegurar que el intervalo de periodos es correcto y convertir todos los datos en tipo Float32.

Normalización

Al tratar con valores de diferentes rangos, un modelo de inteligencia artificial debería lidiar con estos de modo que sería costoso tanto en tiempo de tratamiento como en posibles errores de escalación matemática. Por ello, se va a realizar una simple normalización: para cada valor de una variable, se le resta la media obtenida de todos los valores de esa variable y se divide por la desviación estándar.

Problema de Datos Supervisados

La estimación, es decir, la calibración del modelo para su entrenamiento, se puede realizar mediante un problema de datos supervisados, esto es, entrenar el modelo mediante datos de entrenamiento de entrada y compararlos con los de salida.

La figura 4.2 muestra una ventana temporal ejemplo donde se diferencian los datos de entrenamiento para el problema de datos supervisados de forma visual.

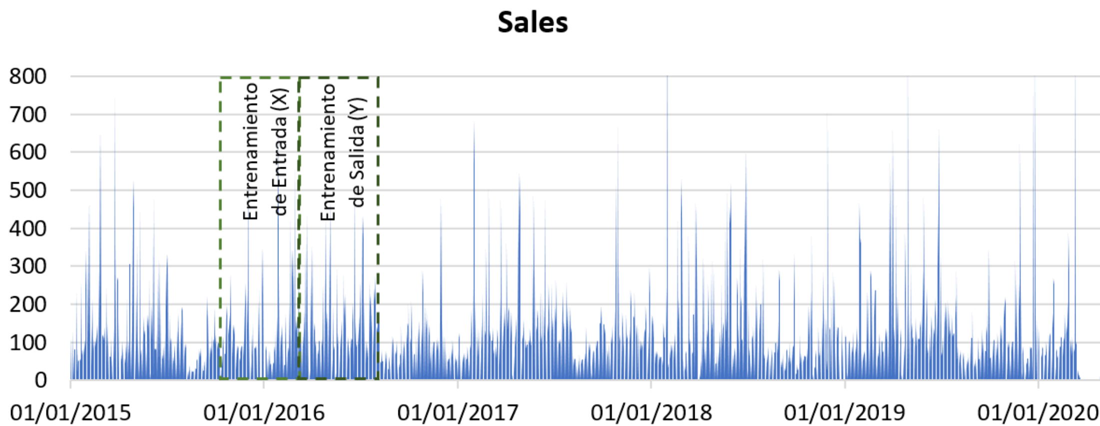

Figura 4.2 Serie Temporal Producto [Fuente: Elaboración propia].

A modo de ejemplo, supongamos que tenemos una secuencia de valores de demanda. Estos datos se pueden reestructurar en un problema de datos supervisados usando valores como entrada aquellos en periodos de tiempo previos y los valores posteriores como variables de salida. Supongamos que tenemos una serie temporal de 7 valores, realizaremos la transformación mediante el método de la ventana deslizante: predecir el siguiente valor mediante los periodos de tiempo previos.

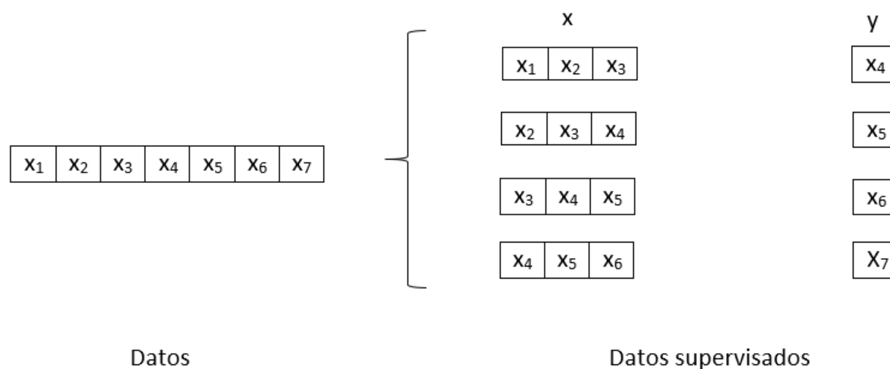

Figura 4.3 Esquema Problema Supervisado [Fuente: Elaboración propia].

En este caso, se han empleado 4 periodos de tiempo previos para predecir el periodo temporal posterior. Donde antes se tenía un vector de siete componentes, la aplicación del problema de datos supervisados a generado un total de cuatro muestras donde cada una de ellas se compone por una entrada x de cuatro valores y un vector de salida y de un único valor (la definición de muestra queda definida en el apartado 3.1).

K-Folding Cross Validation

K-Folding Cross Validation es una técnica que consiste en dividir los datos en K particiones [12] para un total de K iteraciones o también denominado folds. Por cada iteración, los datos se dividirán en K-1 particiones de entrenamiento y una de validación. En cada iteración se evaluará un modelo Deep Learning con misma arquitectura obteniendo así unos resultados para, finalmente, realizar la media de todos ellos.

La figura 4.4 describe gráficamente el comportamiento de la técnica.

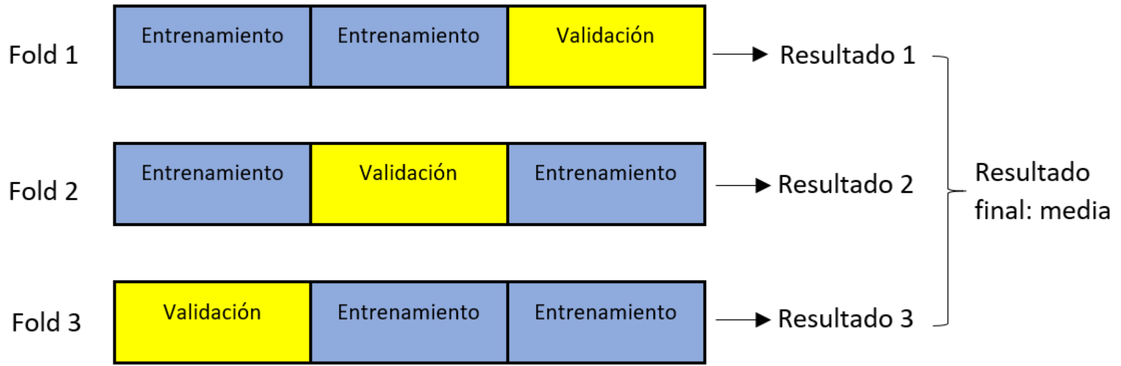

Figura 4.4 Esquema K-Folding Cross Validation [Fuente: Elaboración propia].

Esta técnica resulta de interés en el Deep Learning cuando la selección de datos de entrenamiento y validación pueden impactar en gran medida en los resultados a buscar, ya sea por cantidad de datos o grandes diferencias de valores entre los mismos. Sin embargo, una cualidad indispensable de esta técnica para poder usarse es que los datos no deben requerir un orden para poder ser analizados, algo estrictamente necesario para el modelado y resolución de nuestro caso práctico.

Por tanto, se va a contemplar una variante de esta técnica que tiene en cuenta este último criterio: una división ordenada de los datos de entrada siguiendo la filosofía K-Fold Crossing Validation. Adaptándolo a la problemática, se tiene:

- Fold 1: Los datos que usará el modelo como entrenamiento contemplarán los dos primeros años de ventas (2015 y 2016). Paralelamente, el modelo comparará los resultados de las estimaciones con los datos de validación del año 2017.
- Fold 2: Los datos que usará el modelo como entrenamiento contemplarán los tres primeros años de ventas (2015, 2016 y 2017). Paralelamente, el modelo comparará los resultados de las estimaciones con los datos de validación del año 2018.
- Fold 3: Por último, los datos de entrenamiento se compondrán de una ventana temporal que abarca desde el año 2015 hasta el año 2018 incluido, siendo los datos de validación el periodo restante.

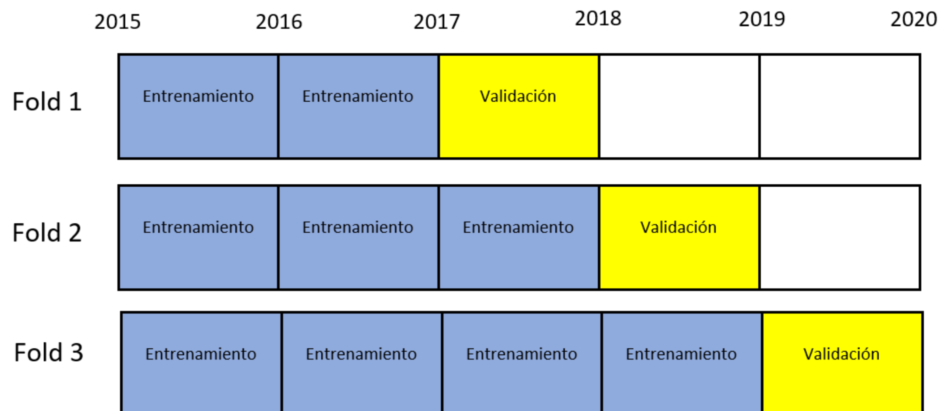

Figura 4.5 Esquema adaptación K-Folding Cross Validation [Fuente: Elaboración propia].

4.2 Modelado computacional

Parte del modelado computacional en lenguaje Python para el tratamiento de datos se encuentra en el apéndice A.1, el resto, al ser específico para cada modelo, quedará reflejado en el modelado de cada uno de ellos: apéndice A.2, A.3 y A.4.

4.3 Modelo 1. LSTM Univariable

4.3.1 Descripción del Modelo

Como primera aproximación y con el fin de analizar una primera respuesta, emplearemos la arquitectura LSTM debido a los buenos resultados que muestra en series temporales para aplicar un modelo univariable en la resolución del caso práctico, esto es, tal como se introdujo en el apartado 2.1.1 y aplicado a la previsión de la demanda, estimar valores futuros obteniendo correlaciones y comportamientos en valores pasados sin incorporar variables adicionales al modelo.

A continuación, se muestra un diagrama donde se representan los pasos que se han llevado a cabo para la obtención de los resultados.

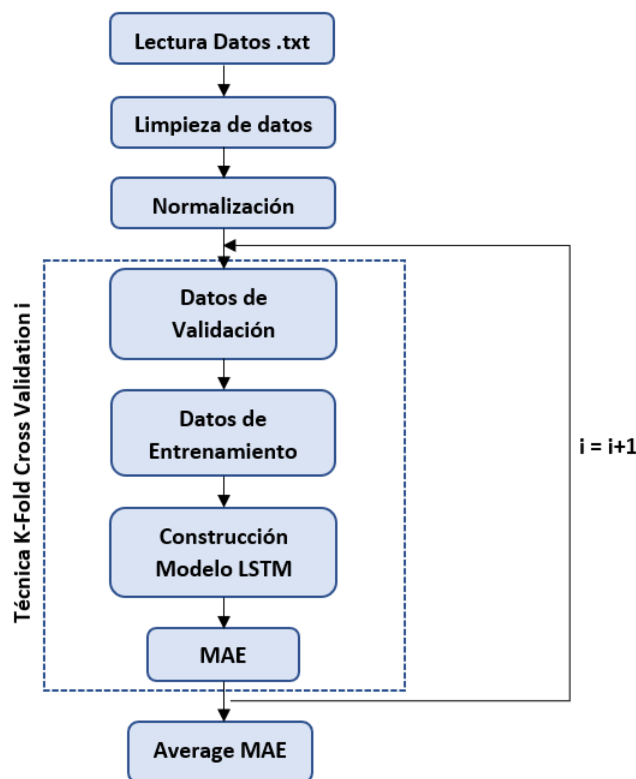

Figura 4.6 Diagrama Modelo LSTM Univariable [Fuente: Elaboración propia].

En primer lugar, se realiza la extracción del número de ventas desde el periodo 01-01-2015 hasta el periodo 31-12-2019 inclusive y se añaden a una estructura de datos, siendo el índice para los periodos y una columna para el valor de las ventas. Posteriormente, se procede con los procesos que conforman la limpieza previa de datos definidos en el apartado 4.1.1 y los procesos necesarios para la normalización, descritos en el apartado 4.1.1.

Para el evaluar el desempeño completo del modelo, y debido a la escasez de datos de entrada, se empleará la técnica K-Folding Cross Validation, donde para cada iteración se definirán los siguientes subprocesos:

1. Definición de los datos de entrenamiento.
2. Definición de los datos de validación.
3. Conversión de los datos de entrenamiento en un problema supervisado tal como se vio en el apartado 4.1.1. Para nuestro problema, se utilizarán 7 valores temporales previos para predecir el siguiente valor temporal. Como salida a la transformación de los datos de entrenamiento al problema de aprendizaje supervisado, se tendrán dos vectores: train_x y train_y que tendrán la siguiente forma respectivamente: (724, 7, 1) y (724, 1) para la primera iteración, (1089, 7, 1) y (1089, 1) para la segunda iteración y, por último, (1454, 7, 1) y (1454, 1) para la tercera iteración. Por tanto, el modelo recibirá datos como entrada después de realizar la transformación de la siguiente forma: (muestras, periodos, características). Puesto que estamos ante un modelo univariable, la dimensión de "características" será igual a uno.
4. Construcción del modelo LSTM. Cada modelo tendrá las mismas características y parámetros (número de capas, morfología, número de nodos, función de activación, etc.) para cada una de las iteraciones que se procederán a describir en la tabla 4.1.
5. Se van a aplicar un total de 200 epochs y un batch compuesto por 14 muestras, número suficiente para analizar la respuesta del sistema y obtener un mínimo error de validación, esto es, que se tomará en consideración para el posterior fitting (ajuste de parámetros para mejorar la precisión del modelo) en el apartado 5.2.
6. Obtención del error MAE (error absoluto medio) para analizar los resultados de forma absoluta.

Tabla 4.1 Parámetros seleccionados modelo LSTM Univariable [Fuente: Elaboración propia].

Parámetros	Valor
Capas	2
Nodos por capa	32
Función de activación	Unidad Lineal Rectificada (Relu)
Ratio de aprendizaje (Lr)	0.001
Función error	Error de media absoluta (MAE)
Optimizador	RMSprop

Finalmente, se calculará el MAE global, obtenido mediante la media de los calculados por cada iteración.

4.3.2 Modelado Computacional

El modelado computacional en Python se encuentra en el apéndice A.2.

4.4 Modelo 2. CNN Univariable

4.4.1 Descripción del Modelo

De la misma forma se ha descrito el modelo LSTM Univariable, se hará una segunda aproximación con un modelo convolucional para ver la respuesta del mismo al problema, su alcance y los resultados que proporciona. Para la descripción de este modelo, se empleará de nuevo la metodología K-folding descrita en el apartado 4.1.1 debido a la escasez de datos de entrada.

Para mayor facilidad de comprensión, a continuación, se presenta el esquema del modelo interno (entradas y salidas) para desarrollarlo posteriormente.

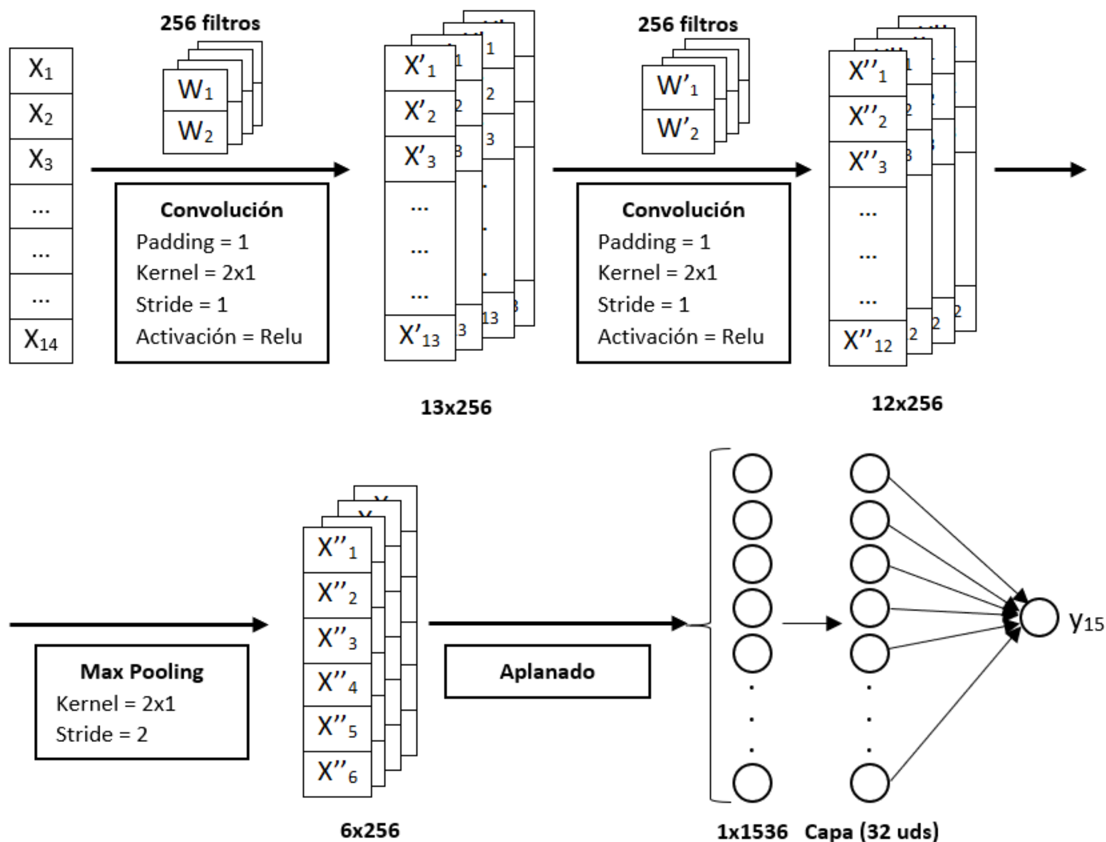

Figura 4.7 Esquema CNN [Fuente: Elaboración propia].

El esquema anterior refleja la secuencia para una sola muestra. Se ha escogido un batch compuesto por 14 muestras. La entrada al modelo contendrá una cadena que contiene 14 valores (2 semanas). Posteriormente, se aplica la primera convolución con un Kernel 2x1, 256 filtros y un padding y stride unitarios además de una función de activación ReLU (conceptos introducidos en el apartado 3.2.2). Como salida, se obtiene una matriz 13x256 fruto de la convolución. A continuación, se repite otra convolución para posteriormente aplicar un Max Pooling, esto es, aplicar el proceso Pooling introducido en 3.2.2 obteniendo el máximo de los valores, un Kernel 2x1 y un Stride igual a 2, obteniéndose así una matriz 6x256. Por último, se aplanan los datos en la red neuronal de una capa intermedia (32 nodos) y una capa de un nodo como salida del modelo.

Una vez que el modelo ha tomado todos los datos de entrenamiento, se alcanza el primer ciclo (epoch). El modelo repetirá el proceso tantas veces como ciclos (epochs) se consideren como parámetro.

Por cada iteración, al igual se ha reflejado en el apartado 4.3.1 para el modelo LSTM Univariable, se realizarán los mismos subprocesos con las siguientes variaciones:

1. El modelo CNN Univariable utilizará 14 valores temporales previos para predecir el siguiente valor temporal. Como salida a la transformación de los datos de entrenamiento al problema de aprendizaje supervisado, se tendrán dos vectores: train_x y train_y que tendrán la siguiente forma respectivamente: (717, 14, 1) y (717, 1) para la primera iteración, (1082, 14, 1) y (1082, 1) para la segunda iteración y, por último, (1447, 14, 1) y (1447, 1) para la tercera iteración.
2. El modelo recibirá las características y parámetros que se presentan en la tabla 4.2.
3. Se van a aplicar un total de 100 epochs y un batch compuesto por 14 muestras, número suficiente para analizar la respuesta del sistema y obtener un mínimo error de validación, esto es, que se tomará en consideración para el posterior fitting en el apartado 5.2.

Tabla 4.2 Parámetros seleccionados modelo CNN Univariable [Fuente: Elaboración propia].

Parámetros	Valor
Capas	1
Nodos por capa	32
Función de activación	Unidad Lineal Rectificada (Relu)
Ratio de aprendizaje (Lr)	0.001
Función error	Error de media absoluta (MAE)
Optimizador	Adam

Por último, se obtendrán el error MAE global para posteriormente analizarlos en el punto 5.2.

4.4.2 Modelado Computacional

El modelado computacional en Python se encuentra en el apéndice A.3.

4.5 Modelo 3: LSTM Multivariable

4.5.1 Descripción del Modelo

Introducción Multivariable

Un modelo multivariable puede proporcionar una solución más precisa al disponer de variables adicionales que hagan que aumente el número de correlaciones y patrones a analizar durante la fase de entrenamiento. Antes de explicar las variables adicionales a considerar, conviene describir cómo se estructuran los datos en el modelo.

Como se ha visto en el apartado 4.3, la entrada que recibe el modelo tiene la forma: (muestras, periodos, características), siendo el índice "características" igual a uno. Sin embargo, para el problema multivariable, será mayor que uno, y el modelo recibirá "características" veces más cantidad de datos de entrada.

A modo de ejemplo, como entrada, supongamos que tenemos 4 muestras de 7 periodos con un desplazamiento temporal de un periodo por muestra y 3 características. La salida se considera de un solo paso, es decir, para una muestra se predice un solo periodo. La matriz tridimensional de entrada y el vector de salida tendrán la siguiente forma que se muestra a continuación.

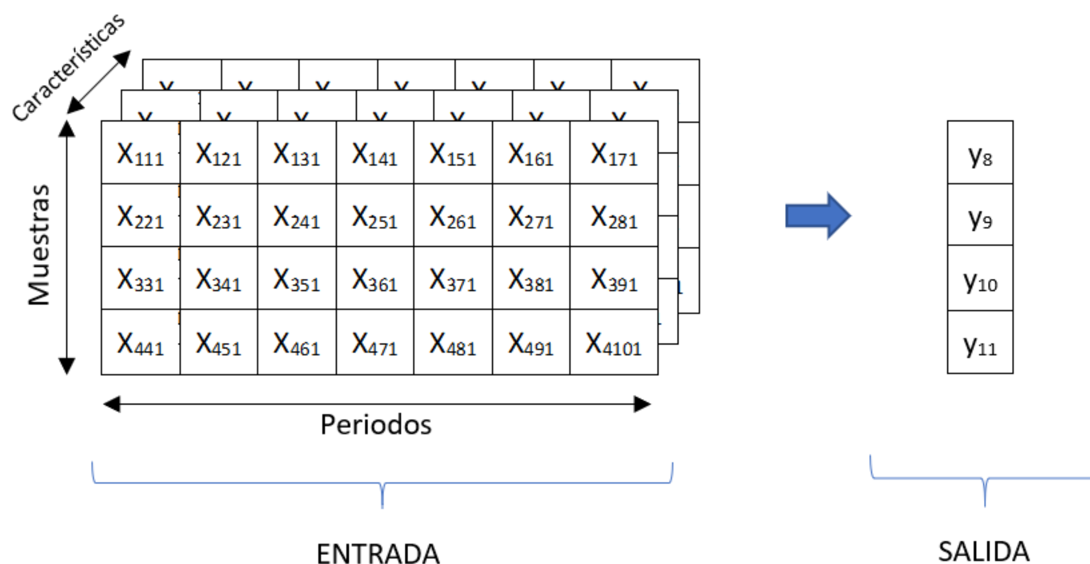

Figura 4.8 Esquema de Multivariedad en los Datos del Modelo [Fuente: Elaboración propia].

Siendo i, j, k las muestras, los periodos y las características correspondientemente.

Variables del Modelo

Para obtener una mejora en comparación con el problema univariable, las variables a seleccionar en este modelo deben tener una correlación con los datos de ventas de nuestro producto, de forma que se pueda entrenar la red neuronal en su completitud. Para ello, se ha elaborado un diagrama árbol que se muestra a continuación donde se analizan las dependencias de nuestro producto frente a las distintas variables.

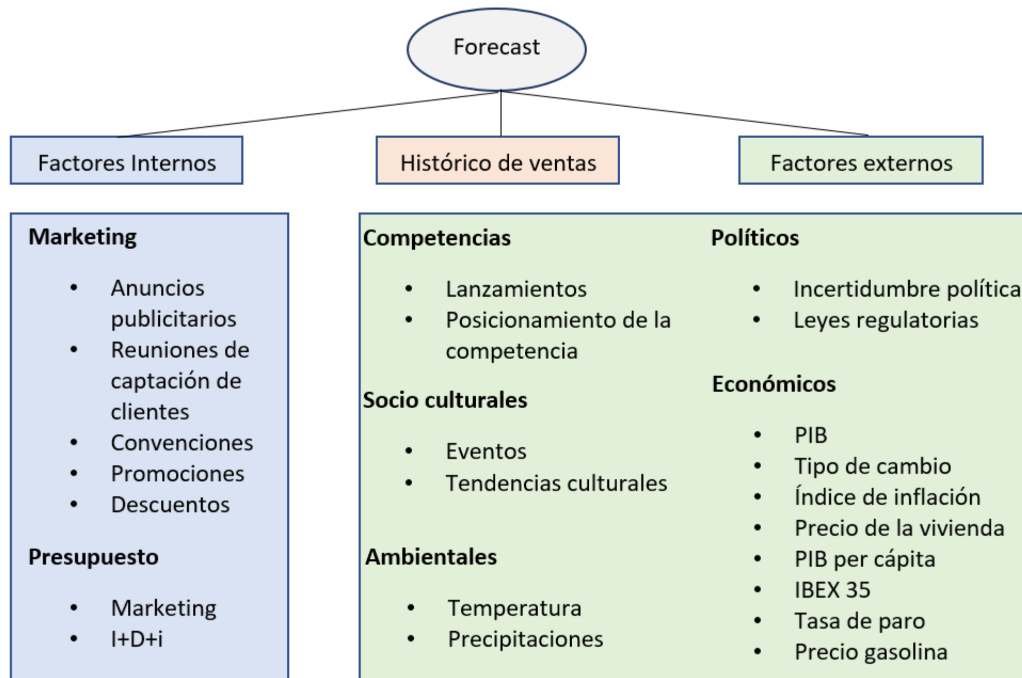

Figura 4.9 Análisis de variables dependientes [Fuente: Elaboración propia].

En nuestro problema, se han escogido las siguientes variables adicionales por su dependencia directa y por su facilidad de obtener datos diarios, ya que la obtención de variables que presentan datos mensuales o trimestrales (frecuencia mayor a los valores del ventas del producto) disminuirían el rendimiento del modelo al tener que realizar interpolaciones considerables:

- IBEX 35.
- Gasolina 95.

Una vez implementadas en el modelo, corroboraremos la correlación existente con la variable ventas, ya que un entrenamiento conjunto con estas variables permite mejorar la eficiencia del modelo mediante una reducción del error en los resultados.

La figura 4.10 muestra la evolución que ha tenido el índice IBEX 35 en el intervalo de estudio (2015-2020).

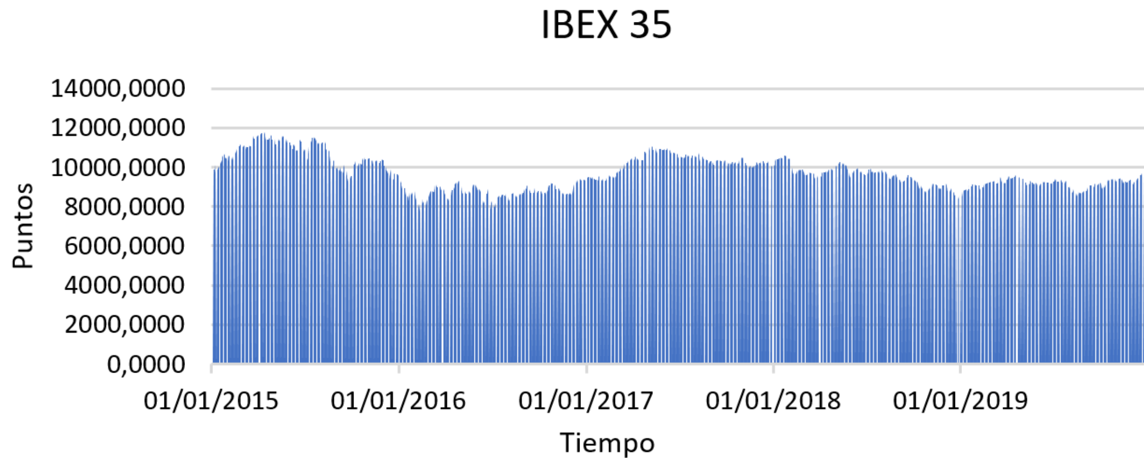

Figura 4.10 Evolución temporal IBEX 35 en el rango de estudio [Fuente: Elaboración propia].

La figura 4.11 muestra la evolución que ha tenido la variable Gasolina95 en el intervalo de estudio (2015-2020).

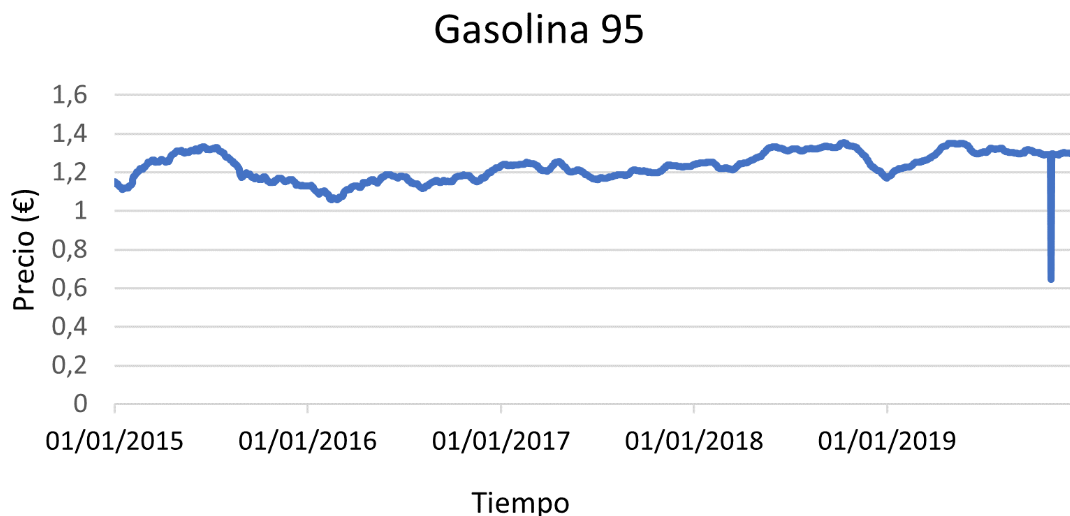

Figura 4.11 Evolución temporal Gasolina95 en el rango de estudio[Fuente: Elaboración propia].

Procesos y descripción del modelo

Para el modelado de nuestro problema multivariable, no se va a aplicar la técnica K-Folding Cross Validation por los siguientes motivos: primero, el número de parámetros de entrada es mayor y segundo, en caso de ser aplicado, el modelo entrenado y, por tanto, el error resultante no es fiable para diferentes periodos de testeo.

Por consiguiente, tal como se ha realizado en el Modelo 1 y 2, los datos de entrenamiento para cada una de las variables (ventas del producto, IBEX35 y Gasolina95) se compondrán desde el periodo 01-01-2015 hasta el periodo 31-12-2018 inclusive (3 años de entrenamiento).

A continuación, se muestra un diagrama donde se representan los pasos que se han llevado a cabo para la aplicación del modelo al caso práctico.

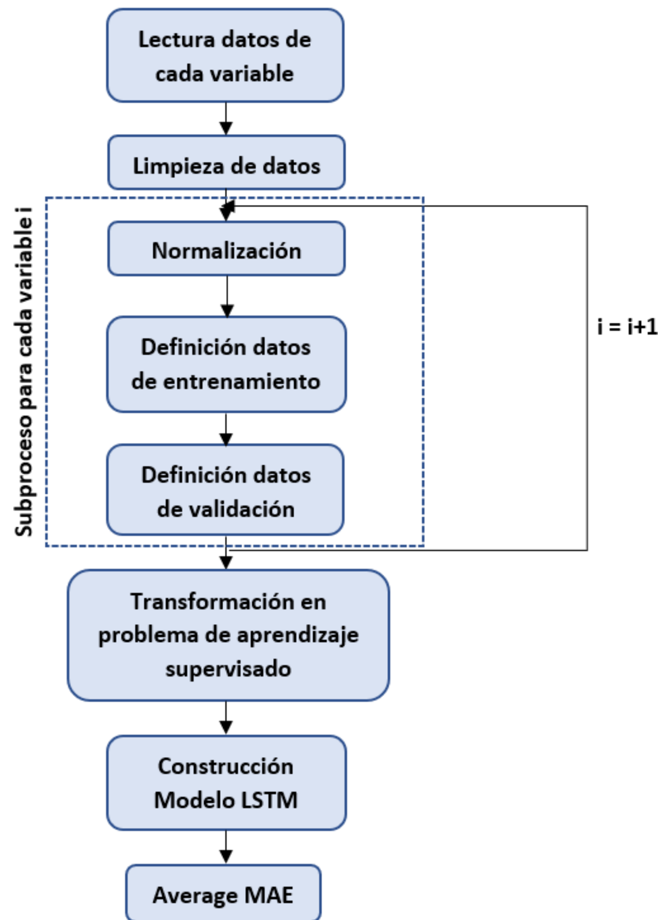

Figura 4.12 Diagrama modelo LSTM Multivariable [Fuente: Elaboración propia].

En primer lugar, para cada una de las variables, se extraen los datos de los ficheros *TXT* desde el periodo 01-01-2015 hasta el periodo 31-12-2019 inclusive y se añaden en una estructura de datos de dos columnas, una para los periodos y otra para los valores de las variables. Posteriormente se procede con la limpieza de datos vista en el apartado 4.1.1, la normalización según el procedimiento del apartado 4.1.1 y se realizan los siguientes subprocesos previo a la construcción del modelo LSTM para cada una de las variables:

1. Definición de los datos de entrenamiento para la variable *i*.
2. Definición de los datos de validación para la variable *i*.

Posteriormente, se procede con la conversión en un problema multivariable supervisado tal como se vio en el apartado 4.1.1. Al igual se aplicó en el modelo 1, para el modelo 3 se utilizarán 7 valores temporales previos para predecir el siguiente valor temporal. Como salida a la transformación de los datos de entrenamiento al problema de aprendizaje supervisado, se tendrán dos vectores: train_x y train_y que tendrán la siguiente forma respectivamente: (1454, 7, 3) y (1454, 1). Como era de esperar, al ser un problema multivariable, cada muestra de entrada al modelo es una matriz 7x3 donde 3 es el número de características (variables).

Una vez tenemos los datos de entrada tratados, se procede a la construcción del modelo LSTM Multivariable. La figura 4.13 muestra el esquema del mismo para mayor facilidad de comprensión.

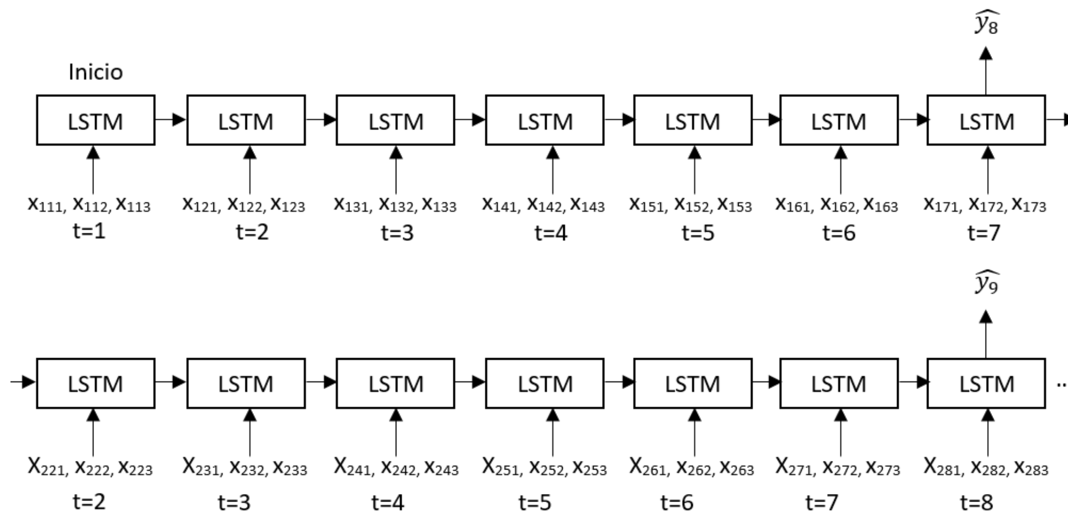

Figura 4.13 Esquema Modelo LSTM Multivariable [Fuente: Elaboración propia].

Donde, como se indicó anteriormente, i, j y k representan: muestra, periodo y característica respectivamente. Cabe recordar, que, en la figura anteriormente representada, cada caja LSTM no es diferente una de otra, sino que todas comparten las mismas características y parámetros, entrenándose el modelo en bucle.

Internamente, debido a las altas prestaciones que se han obtenido mediante ensayo, se ha construido el modelo LSTM con la siguiente estructura y características.

Tabla 4.3 Parámetros seleccionados modelo LSTM Multivariable [Fuente: Elaboración propia].

Parámetros	Valor
Capas	2
Nodos por capa	128
Función de activación	Unidad Lineal Rectificada (Relu)
Ratio de aprendizaje (Lr)	0.001
Función error	Error de media absoluta (MAE)
Optimizador	Adam

Finalmente, a la hora de periodicidad de actualización de los pesos del modelo, se ha escogido un Batch compuesto por 14 muestras y un total de 23 Epochs.

4.5.2 Modelado Computacional

Una vez tenemos descrito el modelo, procederemos a describir el modelado computacional y los comandos utilizados para mayor facilidad de comprensión de los resultados. Se seguirá el mismo

orden descrito en el diagrama de la figura 4.12.

Bibliotecas externas implementadas

Una biblioteca es un conjunto de implementaciones funcionales codificadas en un lenguaje de programación que ofrece una interfaz bien definida para la funcionalidad que se invoca [1]. A la hora de la codificación en Python, además de las bibliotecas propias de Keras, haremos uso de la biblioteca Pandas. Pandas es una biblioteca de manipulación y análisis de datos de código abierto rápida, potente, flexible y fácil de usar [4]. A causa de la gran cantidad de datos con la que nuestro modelo trabaja, Pandas es una herramienta muy útil para la manipulación de archivos de texto, así como para transformaciones de datos. Otra biblioteca con la que trabajaremos en nuestra codificación será Matplotlib. Matplotlib es una biblioteca dedicada a la creación de visualizaciones estáticas, dinámicas e interactivas en Python [3], que será imprescindible para la visualización de nuestros datos y resultados. Por último, otras bibliotecas secundarias nos ayudarán a facilitar nuestra codificación: Math, Numpy, Sklearn y Calendar. Las tres primeras dedicadas a codificación matemática en Python mientras que la última se usará para operaciones de calendario. A continuación, se muestran las bibliotecas necesarias en nuestra codificación en Python:

```
import pandas as pd
import matplotlib.pyplot as plt

from keras import layers
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dense
from keras.optimizers import RMSprop, Adam, SGD, Adadelta

import numpy as np
from math import sqrt
from sklearn.metrics import mean_squared_error
import calendar
```

A lo largo de la sección, se describirán el uso de cada una de las bibliotecas.

Lectura y limpieza de datos

Para la lectura de datos en Python, haremos uso de la biblioteca Pandas introducida anteriormente. En primer lugar, para cada una de las variables, se crea un Dataframe (estructura de datos de 2 dimensiones) con dos columnas (“Fecha” y “Unidades”) para la variable datos de ventas, para la variable IBEX35 (“Fecha” y “Cierre”) y también (“Fecha y “Precio”) para la variable Gasolina95. Estas columnas se definirán mediante la variable my_cols. Posteriormente, establecemos la primera columna como índice del Dataframe mediante la propiedad set_index y aplicamos el formato fecha que queramos, en nuestro caso: Año, mes, día.

Al leer un archivo de texto que comprende datos desde el periodo 01-01-2015, hasta otro periodo distinto del 31-12-2019, simplemente eliminamos los datos que no entran en nuestro análisis. Por último, convertimos todos los valores en formato float32. A continuación, se muestra la codificación en Python.

```
''' Lectura de los datos de ventas del producto '''
my_cols = [str(i) for i in range(2)]
datos = pd.read_csv('SEAT_LEON.txt', sep="\t", names=my_cols, engine='python')
datos.columns = ['Fecha','Unidades']
datos.set_index('Fecha', inplace=True)
```

```
datos.index = pd.to_datetime(datos.index, format='%Y-%m-%d')

'''Lectura de datos de la variable Ibex35'''
#DATOS IBEX
datos_ibex = pd.read_csv('IBEX.txt', sep=",", header=0)
datos_ibex.set_index('Fecha', inplace=True)
datos_ibex = datos_ibex['Cierre']
datos_ibex = datos_ibex.astype('float32')

'''Lectura de datos de la variable Gasolina95'''
#DATOS GASOLINA 95
datos_gasolina95 = pd.read_csv('PreciosGasolina.txt', sep=",", header=0)
datos_gasolina95.set_index('Fecha', inplace=True)
datos_gasolina95 = datos_gasolina95[:-59]
datos_gasolina95 = datos_gasolina95['Precio']
datos_gasolina95 = datos_gasolina95.astype('float32')
```

Modelado de la función Normalizar

La función normalizar recibirá como único argumento una variable Dataframe. Dentro de la función, se crearán tres variables que a su vez, serán las salidas de la función: mean_train (que será la media de todos los valores de la variable de entrada), std_train (desviación de todos los valores de la variable de entrada) y por último la variable train, que será el valor normalizado para cada uno de los periodos del Dataframe. La codificación en Python para la definición de la función Normalizar quedaría de la siguiente manera:

```
def normalizar(train):
    mean_train = train.mean(axis=0)
    train -= mean_train
    std_train = train.std(axis=0)
    train /= std_train
    return train, mean_train, std_train
```

Posteriormente, se normalizan cada una de las variables de entrada al modelo llamando a la función Normalizar mediante la siguiente codificación:

```
datos_norm, mean_train_datos, std_train_datos = normalizar(datos)
ibex35_norm, mean_train_ibex35, std_train_ibex35 = normalizar(datos_ibex)
gasolina95_norm, mean_train_gasolina95, std_train_gasolina95 = normalizar(
    datos_gasolina95)
```

Definición datos de entrenamiento y validación

Una vez tenemos los datos normalizados, se procede a la definición de los datos de entrenamiento y validación para cada una de las variables. Al seguir trabajando con Dataframes, simplemente se divide cada variable en variables de validación y entrenamiento: val_datos, val_gasolina95 y val_ibex por un lado, y train_datos, train_gasolina95 y train_ibex por otro.

Tabla 4.4 Datos de entrenamiento y validación de la variable ventas del producto [Fuente: Elaboración propia].

```
Fecha
2015-01-01   -0.812319
2015-01-02    0.140567
2015-01-03   -0.812319
2015-01-04   -0.812319
2015-01-05    0.276694
                ...
2018-12-27    0.429836
2018-12-28    0.038472
2018-12-29   -0.752764
2018-12-30   -0.812319
2018-12-31   -0.812319
Name: Unidades, Length: 1461, dtype: float32
```

```
Fecha
2019-01-01   -0.812319
2019-01-02    0.208630
2019-01-03    0.140567
2019-01-04    0.098027
2019-01-05   -0.812319
                ...
2019-12-27    0.693581
2019-12-28   -0.812319
2019-12-29   -0.812319
2019-12-30    0.506407
2019-12-31   -0.812319
Name: Unidades, Length: 365, dtype: float32
```

La codificación en Python para la definición de los datos de entrenamiento y validación quedaría de la siguiente forma:

```
train_datos = datos_norm[:365*4+1]
train_gasolina95 = gasolina95_norm[:365*4+1]
train_ibex = ibex35_norm[:365*4+1]

val_datos = datos_norm[365*4+1:]
val_gasolina95 = gasolina95_norm[365*4+1:]
val_ibex = ibex35_norm[365*4+1:]
```

Modelado de la función Data_supervised

La función Data_supervised recibirá los siguientes argumentos:

- Valores pasados: Periodos previos del problema de aprendizaje supervisado.
- Valores futuros: Periodos futuros a pronosticar del problema de aprendizaje supervisado (múltiples pasos).
- Datos de variable ventas del producto.
- Datos de variable IBEX35.
- Datos de variable Gasolina95.

Como salida de la función, devolverá dos arrays, un array de dimensión=3 (muestras, periodos y características) y otro de dimensión=2 donde se almacenarán los valores futuros del problema supervisado (muestras, valores futuros).

En primer lugar, creamos los arrays entrada y salida que almacenarán los valores previos y futuros del problema. Posteriormente, por cada array de entrada y salida, tendremos dos subarrays (x e y) que almacenarán los periodos temporales previos y futuros. Por cada subarray, la variable p recorrerá los valores pasados del array x para añadir las distintas características. Por último, insertamos los valores futuros de la variable ventas del producto en la variable y, y se añade junto a x a los arrays de entrada y salida.

El código para la definición de la función Data_supervised quedaría de la siguiente forma:

```
''' Definición de la función data_supervised '''
''' Definimos los argumentos de entrada '''
def data_supervision(past_values, future_values, datos1, datos2, datos3, datos4):
```

```
    ''' Creamos dos arrays que almacenarán los valores previos y posteriores \
    respectivamente '''
    entra, sale = list(), list()
    '''Por cada array de entrada y salida, tendremos "len(datos1)-\
    past_values-future_values+1" subarrays que almacenarán los timesteps '''
    for i in range(len(datos1)-past_values-future_values+1):
        x,y = list(), list()
        ''' Por cada subarray, se incorporan las características en un nuevo subarray '''
        for j in range(past_values):
            p = list()
            p.append(datos1[j+i])
            p.append(datos2[j+i])
            p.append(datos3[j+i])
            x.append(p)
        for j in range(future_values):
            y.append(datos1[past_values+j+i])
        entra.append(x)
        sale.append(y)
    return np.array(entra), np.array(sale)
```

Cabe destacar del código anterior la necesidad de la biblioteca Numpy para la creación de los arrays del modelo.

Con la función modelada en Python, la aplicamos a los datos de entrenamiento y supervisión respectivamente.

```
train_x, train_y = data_supervision(past_values, future_values, train_datos,
    train_gasolina95, train_ibex,)

val_x, val_y = data_supervision(past_values, future_values, val_datos, val_gasolina95,
    val_ibex)
```

Construcción Modelo LSTM

Una vez codificado nuestros datos de entrada, procedemos con la construcción del modelo multivariable LSTM.

En primer lugar, creamos un objeto model, y llamamos a la función Sequential. Esta función, definirá una interfaz básica de un modelo Deep Learning donde se podrán añadir capas secuenciales, sirviendo para la mayoría de los problemas de aprendizaje profundo.

```
''' Construcción del modelo multivariable LSTM'''
''' Llamamos al API Sequential '''
model = Sequential()
```

A continuación, pasamos a la construcción de las capas para la estructura LSTM mediante la función add(LSTM(...)). Para cada capa, se indicará el número de nodos que queremos tener (128), la función de activación para cada salida de cada nodo (ReLU) mediante el argumento activation y la forma de entrada que tendrá cada capa mediante el argumento input_shape (que, como se ha comentado, recibirá por cada muestra la forma (past_values, n_features)).

Puesto que se van a añadir varías capas consecuentivas con las mismas características internas, es

necesario llamar al argumento return_sequences = True para que la salida de datos de una capa sea la entrada de la siguiente.

```
''' Construimos sobre la interfaz 2 capas con 128 cada una'''
model.add(LSTM(128, activation='relu', input_shape=(past_values, n_features),
      return_sequences=True))
model.add(LSTM(128, activation='relu', input_shape=(past_values, n_features)))
```

Por último, construimos una capa de salida que será el valor pronosticado de nuestro modelo, para ello, la función Dense crea una capa independiente que recibirá como argumento el número de nodos que la compone (future_values).

```
''' Construimos una capa de salida para la predicción del valor futuro '''
model.add(Dense(future_values))
```

Para terminar con la estructura del modelo, queda definir caracterísitcas internas para la compilación. Para definirlas, llamamos a la función compile (compilación), donde los argumentos optimizer, loss y metrics asignarán el optimizador junto a su ratio de aprendizaje, la función de error y las métricas a analizar.

Finalmente, quedaría entrenar el modelo. La función fit se encargará de ello recibiendo como argumentos los datos de entrenamiento que serán comparados con los datos de validación conforme va entrenando y los argumentos: epochs, batch y el argumento verbose, que, asignándolo a 1, muestra por pantalla las métricas por cada batch.

```
''' Llamamos a la función compile para definir el optimizador, el ratio de aprendizaje ,\
 función objetivo y las métricas a analizar '''
    model.compile(optimizer=Adam(lr = 0.001), loss='mae', metrics=['acc'])

    ''' Almacenamos los resultados de entramiento y error del modelo en la variable \
    history , junto con la definición de los epochs y tamaño del batch'''
    history = model.fit(train_x, train_y,
                        validation_data=(val_x, val_y),
epochs=num_epochs, batch_size=batch, verbose=1)
```

5 Resultados

En el presente capítulo, se procederá en primer lugar al entrenamiento y evaluación de los resultados de los modelos Deep Learning en el punto 5.1, para posteriormente, en el punto 5.2, realizar un análisis experimental mediante fitting que permita mejorar el desempeño de los resultados de nuestro caso práctico. Por último, en el apartado **??** se realizará un análisis de la eficiencia de los modelos empleados frente a otros métodos clásicos.

5.1 Resultados computacionales

Para evaluar la efectividad y, por tanto, la exactitud de los modelos de nuestro caso práctico, procedemos al entrenamiento de los mismos mediante la función fit de Keras con objeto de obtener la predicción de ventas mediante Deep Learning.

Para el entrenamiento y muestreo de los resultados computacionales, los algoritmos implicados, al igual que para el modelado de nuestro problema, se han codificado en lenguaje Python 3.8 y depurados con las siguientes características: 2.7GHZ i5-7200U CPU, Windows 10 PC.

Una vez definidas mediante experimentación las características más prometedoras para el entrenamiento de cada modelo: tamaño de batch, nodos, número de capas, función de activación, ratio de aprendizaje, función error y el optimizador, se va a realizar un muestreo por pantalla de la evolución del error del modelo tomando las variables de validación como testeo por cada epoch de entrenamiento. Para ello, tendremos una gráfica donde el eje de abscisas definirá el número de epochs y el eje de ordenadas el valor del error. El objetivo de este muestreo es analizar tres puntos imprescindibles en el modelado de una red neuronal:

1. Obtener el número de epochs óptimo que presenta un mínimo error en los resultados del modelo.
2. Comparar la evolución del error para las variables de entrenamiento y las variables nunca antes vistas (validación/testeo).
3. Observar si existen problemas de overfitting o underfitting.

En primer lugar, el número óptimo de epochs se va a elegir mediante el error obtenido por el testeo de las variables de validación. Como es de esperar, por cada epoch, el modelo irá aprendiendo progresivamente las diferentes correlaciones que permitan aumentar la exactitud de las predicciones, sin embargo, llegará un momento en el que las correlaciones aprendidas para los datos de entrenamiento no sirvan para datos no vistos nunca. Es aquí cuando decimos que el modelo está sobreentrenado. Por tanto, si siguiésemos entrenando el modelo con más epochs, el error para los

datos de entrenamiento seguirá disminuyendo mientras que para los datos de validación el error irá aumentando progresivamente.

En segundo lugar, es de vital importancia analizar el comportamiento del modelo durante el entrenamiento puesto que nos puede dar información de si se está seleccionando una correcta entrada de datos, si se ha realizado una correcta normalización, o si existen valores que perturben el entrenamiento del modelo.

Por último, tenemos el problema del overfitting y underfitting. Cuando un modelo Deep Learning tiene una gran cantidad de parámetros que aprender se requiere una gran cantidad de datos de entrenamiento que consigan obtener el resultado esperado con suficiente efectividad. Sin embargo, si los datos de entrenamiento son demasiado pequeños comparado con el número de pesos/parámetros de aprendizaje, el modelo sufrirá overfitting [17]. Por otro lado, el problema de underfitting se debe al motivo contrario, no existe una cantidad de datos de entrenamiento suficiente para capturar las correlaciones entre las entradas al modelo y las variables objetivo.

Para el muestreo de los resultados, la función fit se encargará de ello recibiendo como argumentos los datos de entrenamiento que serán comparados con los datos de validación conforme va entrenando y los argumentos para el número de epochs, el tamaño del batch y el argumento verbose.

```
history  = model. fit ( train_x ,  train_y ,
                        validation_data =(val_x ,  val_y ),
epochs=num_epochs, batch_size=batch,  verbose=1,  shuffle =False)
```

Con el modelo entrenado, vamos a almacenar en un diccionario llamado history_dic las dos métricas más relevantes:

- Evolución del error por cada epoch con variables de entrenamiento (loss_values).
- Evolución del error por cada epoch con variables de validación (val_loss_values).

Cabe destacar que otras métricas no resultan de interés para un problema de previsión. A modo de ejemplo, la métrica accuracy mide la exactitud de un valor real con el que predice el modelo. Al no ser un problema de clasificación (aquel cuya salida solo esta comprendida entre un número finito de categorías), el valor de accuracy será siempre muy bajo ya que es muy difícil obtener un valor idéntico con variables reales.

Una vez tenemos almacenadas las métricas de nuestro modelo entrenado, se representarán los resultados en la gráfica y posteriormente se identificará el error mínimo de validación. A continuación, se presenta a modo de resumen los resultados obtenidos de nuestro problema práctico para cada uno de los modelos construidos para posterior análisis de estos además de la representación gráfica descrita anteriormente.

Tabla 5.1 Resultados computacionales para cada modelo [Fuente: Elaboración propia].

Modelo	K-Folding	Epochs	Min MAE Validación	MAE Entrenamiento
Modelo 1: LSTM Univariable	No	200	0,3941	0,3567
Modelo 1: LSTM Univariable	Sí	200	0,4091	0,3531
Modelo 2: CNN Univariable	No	100	0,4201	0,3785
Modelo 2: CNN Univariable	Sí	100	0,4394	0,3767
Modelo 3: LSTM Multivariable	No	23	0,399	0,3667

Para el muestreo del error de los modelos que añaden la técnica K-Folding Cross Validation, se ha obtenido la media de los batch para cada uno de los epochs, quedando la codificación en Python de la siguiente manera:

```
all_mae_histories_val = []
all_mae_histories_loss = []

mae_history_val = history.history['val_loss']
all_mae_histories_val.append(mae_history_val)
mae_history_loss = history.history['loss']
all_mae_histories_loss.append(mae_history_loss)

average_mae_history_val = [
np.mean([x[i] for x in all_mae_histories_val]) for i in range(num_epochs)]
average_mae_history_loss = [
np.mean([x[i] for x in all_mae_histories_loss]) for i in range(num_epochs)]
```

Observando los resultados de nuestro caso práctico, podemos ver como la técnica K-Folding Cross Validation obtiene peores resultados computacionales debido a que el modelo trabaja con un número menor de variables para realizar un entrenamiento maduro.

Por otro lado, queda demostrado experimentalmente que el modelo 2: CNN Puro Univariable tiene un menor rendimiento frente al resto, quedando por tanto fuera de estudio para el posterior análisis experimental y mejora del modelo (fitting).

Cabe destacar los modelos 1 y 3: LSTM Puro Univariable y LSTM Multivariable por su menor error de validación. Es importante puntualizar que el error puede variar por cada experimento ligeramente ya que los pesos de las redes neuronales son actualizados de manera arbitraria para minimizar la función objetivo final, por tanto, puede haber ocasiones donde un modelo obtenga una pequeña variación por experimento.

Por último, podemos verificar que efectivamente, los variables adicionales tomadas como entrada al modelo (Gasolina95 e IBEX35) han mejorado el desempeño analizar los resultados del mínimo MAE en el rango de validación, sin embargo, no ha habido una mejora notable. Esto se debe a que las variables escogidas no tienen una fuerte dependencia/correlación con la variable ventas.

El objetivo del punto 5.2.2, será comprobar y corroborar la importancia de una correcta variable de entrada para una mayor mejora de los resultados mediante la realización de una simulación. A continuación, se muestra la representación gráfica del desempeño de los modelos propuestos para nuestro problema:

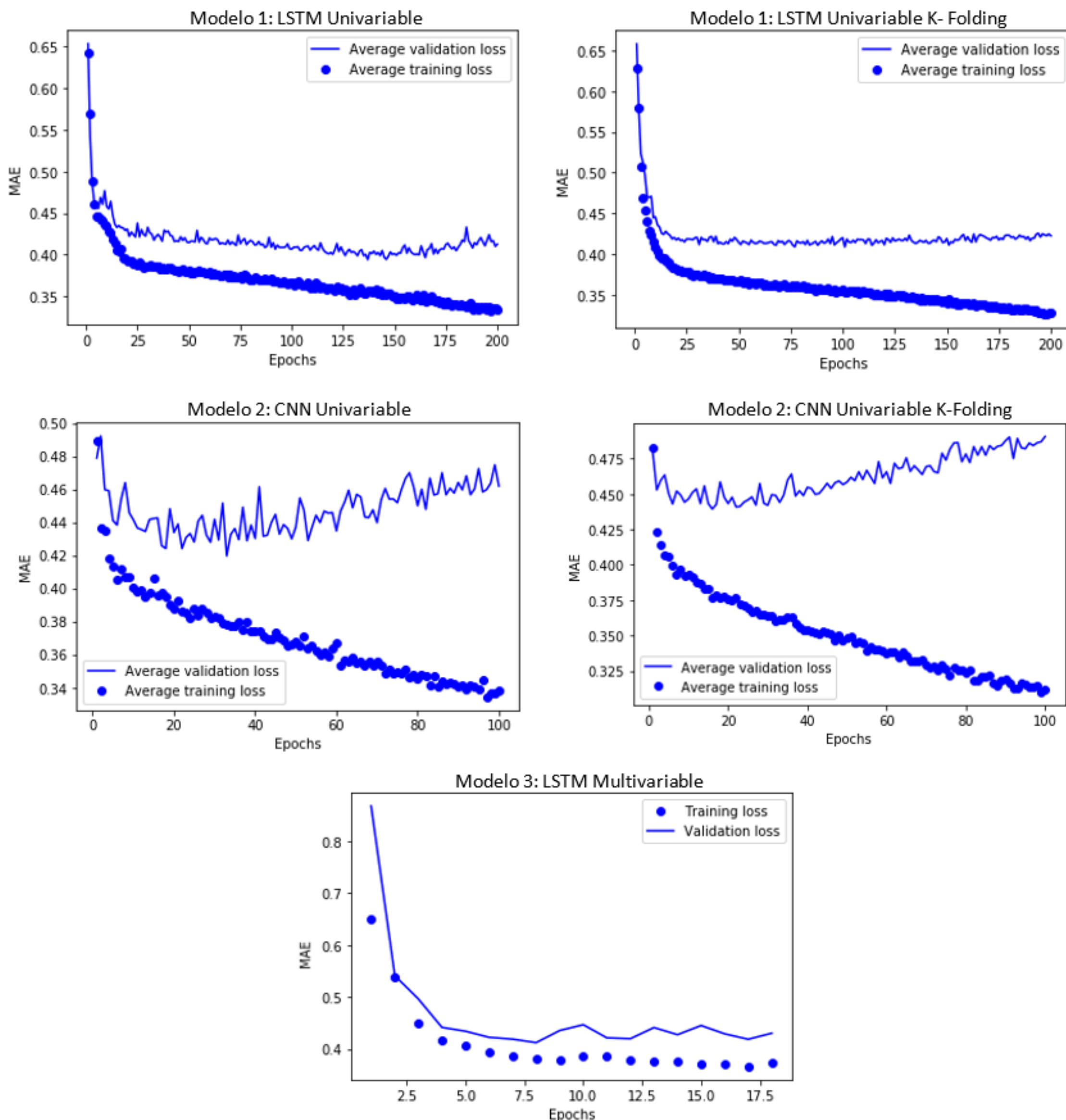

Figura 5.1 Gráficos resultados computacionales por modelo: evolución del error por cada Epoch [Fuente: Elaboración propia].

Analizando los gráficos de la figura que se muestra anteriormente, era de esperar la evolución que tendría los resultados del error de entrenamiento frente a los de validación por cada epoch. A medida que el modelo aprende, va obteniendo correlaciones que terminen en un mayor desempeño para los años que se han indicado como entrenamiento, sin embargo, para variables no vistas anteriormente, es predecible que el error será mayor.

Por otro lado, se puede apreciar un claro comportamiento de las redes neuronales durante el entrenamiento: overfitting. A medida que se van actualizando los pesos del modelo, ambos errores disminuyen considerablemente hasta que el error de validación llega a un mínimo para un epoch

dado. En caso de seguir entrenando el modelo más epochs, el error de validación aumentará mientras que el de entrenamiento seguirá disminuyendo debido a que las correlaciones nuevas que se crean en el modelo no se aplican a las variables nunca antes vistas, sino que sobreentrena variables ya conocidas, perdiendo así el correcto rendimiento de predicción.

Para mayor facilidad de comprensión, se muestra a continuación el desempeño del modelo multivariable para un entrenamiento de 50 epochs:

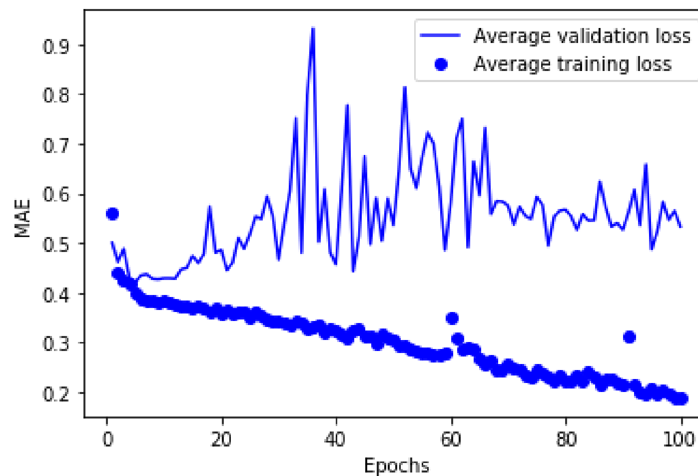

Figura 5.2 Evolución LSTM Multivariable para 100 epochs [Fuente: Elaboración propia].

En el apartado 5.2.1 daremos solución al problema de Overfitting mediante la introducción una serie de técnicas y se verá la respuesta del modelo.

Debido a esto, además se ha comprobado como las variables del problema multivariable no han proporcionado una minimización del error considerable, por ello, puesto que el modelo 1 no tiene mayor potencial de mejora, en el apartado 5.2, haremos un análisis experimental del modelo multivariable e introduciremos unas mejoras al mismo mediante fitting para dar como resultado un menor error en el forecasting de nuestro caso práctico.

5.2 Análisis experimental (Fitting)

Este subapartado se dedicará a la mejora del modelo LSTM Multivariable (fitting). En el apartado 5.2.1 resolveremos el problema de Overfitting mientras que finalmente, en el apartado 5.2.2 se realizará una mejora multivariable mediante una simulación fuera del caso práctico.

Como parámetro de medición de mejora, se evaluará una medida de error adicional: error de la raíz de la media cuadrática (RMSE), con el objetivo de poder analizar la sensibilidad de las grandes desviaciones entre los valores pronosticados y reales.

En primer lugar, una vez, el modelo ha sido entrenado, vamos a proceder a representar mediante un gráfico la respuesta del mismo haciendo una previsión de la demanda en el rango de validación. Para ello, se comparará los datos del rango de validación reales con la predicción del modelo en el tiempo. Para la previsión y muestreo de nuestro caso práctico, se han llevado a cabo los siguientes pasos:

1. Previsión de los valores de validación: la función `model.predict()` toma como argumento un

array de la forma (muestras, periodos, características) que para nuestro caso práctico serán los datos de validación.

2. Se crea un array que contenga dichos valores pronosticados almacenados.
3. De-normalizamos los valores pronosticados.
4. Igualamos a cero los valores cercanos a cero (negativos y positivos) forzando el conjunto de positividad y nulidad cuando el modelo se ha aproximado en gran medida.
5. De-normalizamos los valores de validación reales.
6. Calculamos los errores a analizar teniendo los valores de validación reales y los pronosticados.
7. Representamos de forma gráfica la evolución de ambas series de valores en el tiempo.

A continuación, se muestra la representación gráfica de la evolución temporal de los valores pronosticados y los valores reales para el rango de validación, es decir, el año 2019 de nuestro caso práctico, donde se obtiene un error RMS igual a 108.

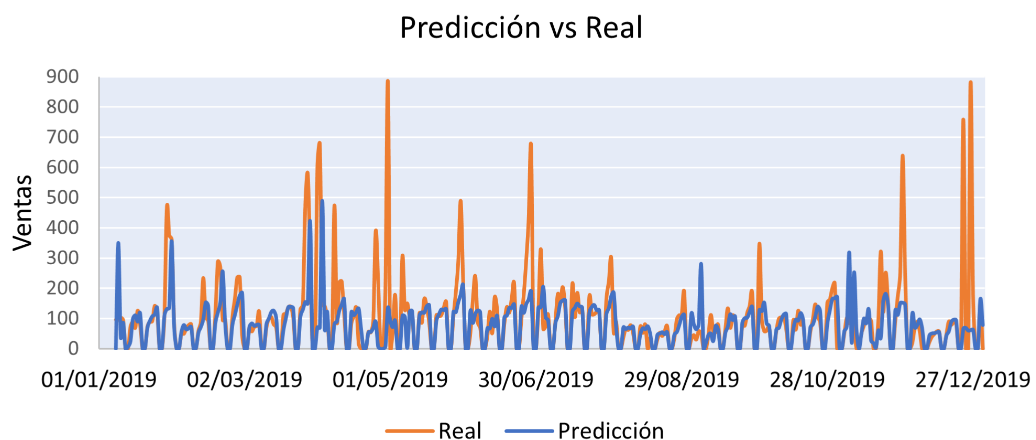

Figura 5.3 Evolución temporal datos pronosticados y reales [Fuente: Elaboración propia].

Cabe destacar de la gráfica anterior, que los valores pronosticados comienzan desde el día 08/01/2019. Esto se debe a que el modelo toma como entrada un array de forma (periodos, características), siendo los periodos igual a 7 días para cada muestra.

Por otro lado, se puede apreciar el buen desempeño del modelo respecto a estacionalidad (ha aprendido a establecer valores iguales a 0 de forma semanal) y respecto a los valores de ventas que no tienen grandes incrementos puntuales. Estos incrementos puntuales en los valores de ventas mayores de 400 unidades pueden deberse a diversos factores que no se han tenido en cuenta en el presente modelo como variables adicionales: campañas publicitarias, ferias de automóvil, promociones, etc. En el aparatdo 5.2.2 se evaluará la efectividad del modelo considerando esto último.

La codificación para la representación gráfica y el modelado del error RMS queda de la siguiente manera:

```
''' Predecimos los valores de validación'''
y_predict_nor = model.predict(val_x)
''' Creamos un array que contendrá dichos valores '''
y_predict= []
```

```
''' Añadimos los resultados en el array '''
for i in range(len(y_predict_nor)):
    y_predict.append(y_predict_nor[i][0])
''' De-normalizamos los resultados '''
for i in range(len(y_predict)):
    y_predict[i] = y_predict[i]* std_train_datos +mean_train_datos

''' Igualamos a cero los valores cercanos '''
for i in range(len(y_predict)):
    if y_predict[i] < 1:
        y_predict[i] = 0

''' Array con los valores de validación reales '''
datos_real = datos_norm[365*4+1+7:]
datos_real = datos_real.values

''' De-normalizamos los valores de validación reales '''
for i in range(len(datos_real)):
    datos_real[i] = datos_real[i]* std_train_datos +mean_train_datos


''' Calculamos el RMSE de la predicción del modelo y los valores reales '''
rms = sqrt(mean_squared_error(datos2[365*4+1+7:], y_predict))
print(rms)

''' Imprimimos por pantalla la comparativa resultados del modelo-valores reales '''
prediccion = y_predict
real = datos_real
plt.figure(figsize=(30,10))
plt.plot(prediccion, 'b', label = 'Predicción')
plt.plot(datos2[365*4+1+7:].values,'g', label = 'Real')
plt.title('Predicción vs Real')
plt.xlabel('Día')
plt.ylabel('Unidades')
plt.legend()

plt.show()
```

5.2.1 Tratamiento de Overfitting

Como se ha comprobado para nuestro caso práctico, a partir de un determinado número de epochs se produce el fenómeno de Overfitting. Como hemos introducido en el apartado 5.1 introducido en el apartado 5.1. La figura 5.4 muestra las etapas: Underfitting y Overfitting para nuestro problema.

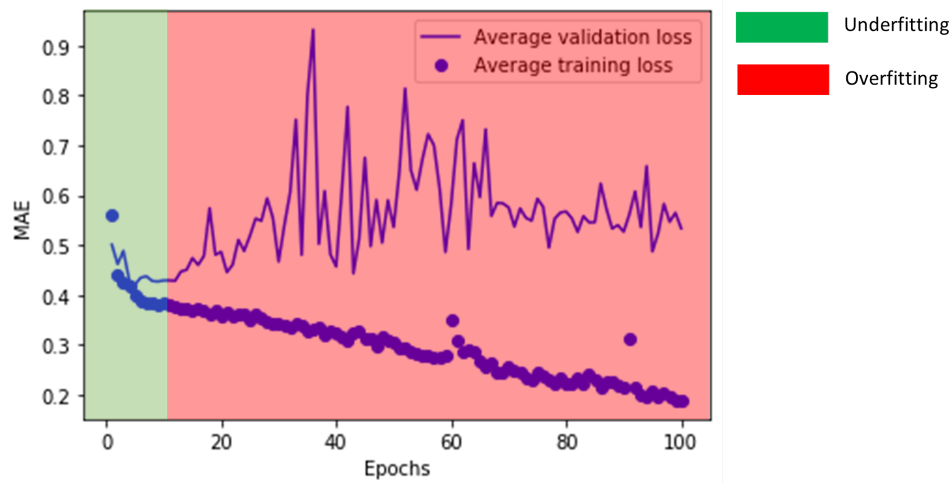

Figura 5.4 Representación temporal de los fenómenos Underfitting y Overfitting [Fuente: Elaboración propia].

Para evitar que un modelo Deep Learning sobre aprenda determinados patrones, la solución óptima es conseguir más datos de entrenamiento [12]. Sin embargo, tal como ocurre en nuestro problema, si carecemos de más datos, podemos aplicar una serie de restricciones en nuestro modelo que limita la cantidad de información que se almacena. El proceso que combate el Overfitting se denomina regularización. La base matemática de la regularización dentro de un modelo Deep Learning es reducir los grados de libertad mediante restricciones en los parámetros (pesos) del modelo [18]. Entre las técnicas más usadas y aplicadas en la práctica para la regularización, podemos encontrar:

1. Reducción del tamaño de la red: Esto se debe a que un mayor tamaño de la red trae consigo una mayor capacidad de aprendizaje del modelo, y, por tanto, puede provocar un aprendizaje de patrones no deseados.
2. Regularización en los pesos.
3. Dropout.

Para nuestro caso práctico emplearemos la técnica Dropout por ser una de las más efectivas y usadas. Procederemos primero a su definición, para posteriormente aplicarla y realizar un análisis de la respuesta computacional.

Técnica Dropout

La técnica Dropout consiste en ignorar (drop out) algunos de los valores de salida de las capas de la red del modelo durante la fase de entrenamiento. Esto tiene el efecto de provocar que la capa se asemeje o se comporte como una capa que contiene un número diferente de nodos que se conectan a la capa previa [8], provocando así la limitación del aprendizaje de la red a correlaciones más generales en lugar de aquellos sobre entrenamientos que mejoren el rendimiento en los datos de entrenamiento.

La implementación del Dropout en una red se realiza por capas. Supongamos que tenemos un modelo Deep Learning donde una capa devuelve el siguiente vector: [0.8, 0.23, 0.56, 0.94, 0.38, 0.11] durante la fase de entrenamiento. Un parámetro, denominado factor de Dropout se encargará de dar una salida igual a 0 en base a la probabilidad del factor. Por tanto, dado un factor 0.5, se tendrá una asignación arbitraria de la mitad de los elementos, para nuestro ejemplo: [0. 0.23, 0.56, 0, 0, 0.11].

Cabe destacar que el Dropout aplica a cualquier capa de la red excepto la de salida. Además, se emplea únicamente durante la fase de entrenamiento, por ello, para que no haya descuadre entre el número de unidades que están activas en la fase de entrenamiento y en la fase de testeo, se debe des-escalar por un factor igual al empleado en el Dropout.

A continuación, vamos a aplicar la técnica a nuestro problema haciendo un estudio de 50 epochs para ver la evolución de la respuesta temporal comparándolo con el modelo original. Para ello, a la salida de cada capa oculta de nuestra red, añadiremos Dropout con un factor de 0.5, que suele ser el más empleado para la mayoría de los modelos. En nuestro modelado computacional, se hará uso en Keras del comando layers.Dropout que recibirá como argumento el factor de Dropout. La codificación en Python queda de la siguiente manera.

```
''' Construcción del modelo multivariable LSTM con Dropout'''
''' Llamamos al API Sequential '''
model = Sequential ()
''' Construimos sobre la interfaz 4 capas con 124 cada una'''
model.add(LSTM(128, activation='relu', input_shape=(past_values, n_features),
return_sequences=True))
model.add(layers.Dropout(0.5))
model.add(LSTM(128, activation='relu', input_shape=(past_values, n_features)))
model.add(layers.Dropout(0.5))
''' Construimos una capa de salida para la predicción del valor futuro '''
model.add(Dense(future_values))
#model.compile(optimizer='adam', loss='mse',metrics=['acc'])
''' Llamamos a la función compile para definir el optimizador, el ratio
de aprendizaje, la función objetivo y las métricas a analizar '''
model.compile(optimizer=Adam(lr = 0.009), loss='mae', metrics=['acc'])
```

Una vez entrenado el modelo añadiendo las restricciones de Dropout, la figura 5.5 muestra una clara mejora respecto al modelo original.

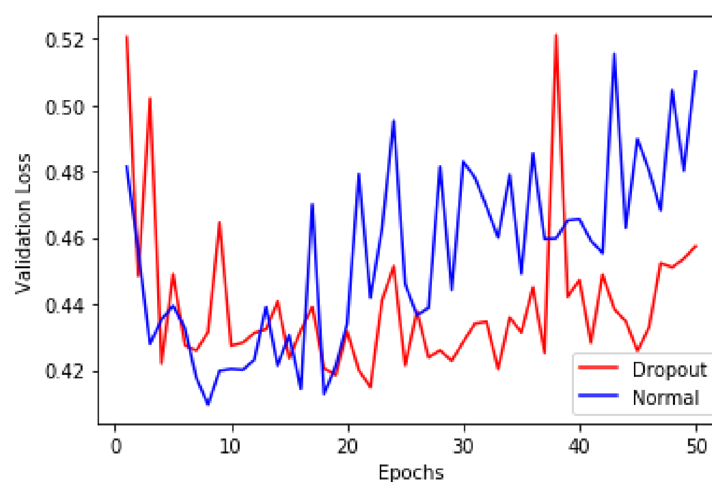

Figura 5.5 Representación temporal del error de validación con/sin Dropout [Fuente: Elaboración propia].

5.2.2 Mejora multivariable al modelo y eficiencia frente a métodos clásicos

Como se ha introducido en el apartado 5.2, la representación gráfica de la evolución temporal del número de ventas pronosticadas y reales muestra claramente unos picos de demanda que el modelo Deep Learning no es capaz de correlacionar debido a que la selección de las variables de entrada no identifica las correlaciones para pronosticar dichos picos de demanda. La gráfica que se muestra a continuación muestra los valores de venta por encima de 400 unidades durante la ventana temporal de validación.

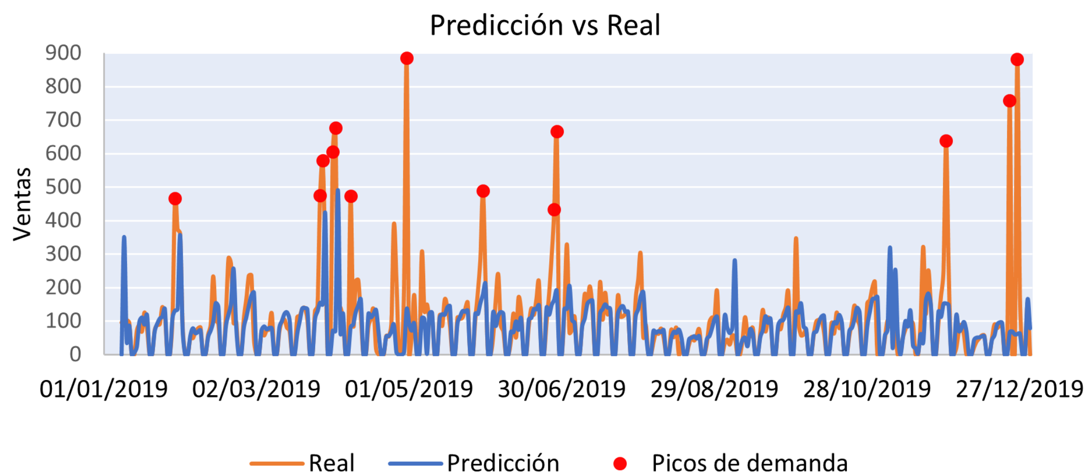

Figura 5.6 Representación de picos de demanda en el año 2019 [Fuente: Elaboración propia].

Estos picos de demanda afectan negativamente al error RMS. Suponiendo que nuestro modelo pronosticase exactamente el valor de estos picos, el porcentaje de reducción del error RMS sería de un 47 %. Por tanto, dar solución o mejorar en este campo permite que la previsión de la demanda de cualquier problema sea más exacta. La solución para dicho problema reside en identificar variables que sean correlativas a los picos de demanda, y, por tanto, el modelo Deep Learning pueda aprender de ellas y aplicarlas durante la fase de previsión. Puesto que carecemos de dichas variables mencionadas, y debido a la importancia de este estudio para el trabajo que se desarrolla, se va a proceder a realizar una simulación con las variables ya introducidas en nuestro problema además de una variable ficticia que queda fuera del caso práctico.

La nueva variable introducida se comportará como una variable de naturaleza: campañas publicitarias/ferias de automóvil/promociones entre otras de forma binaria [0,1] con un desfase temporal de menos un periodo para cada pico de demanda. Más concretamente, supongamos que tenemos para nuestra variable ventas del producto los siguientes datos para tres periodos: [56, 156, 452]. Esto implicaría que en el periodo tres hay un pico de demanda, por tanto, la nueva variable introducida: variable picos de demanda almacenará los siguientes datos para los mismos periodos: [0, 1, 0] (suponiendo que el periodo número cuatro no es un pico de demanda). Esta simulación es totalmente válida en el día a día, ya que una empresa puede ser consciente de los días donde puede haber picos de demanda debido a acciones internas de la empresa que disparen las ventas.

Una vez tenemos identificada la naturaleza y comportamiento de la variable picos de demanda, se procede a su modelado en Python:

```
picos = datos.copy()
for col in picos.columns:
```

```
    picos[col].values[:] = 0

for i in range(len(datos.values)):
    if datos.values[i]>300:
        picos['Unidades'][i-1] = 1

picos = picos['Unidades']

n_features = 4

picos_norm = picos
train_picos = picos_norm[:365*4+1]


train_x, train_y = data_supervision(past_values, future_values, train_datos,\
train_gasolina95, train_ibex, train_picos)

val_picos = picos_norm[365*4+1:]

val_x, val_y = data_supervision(past_values, future_values, val_datos, val_gasolina95,\
val_ibex, val_picos)
```

Al haber incorporado una variable más a nuestro modelo LSTM Multivariable, se deben realizar algunas modificaciones computacionales, las cuales se indican a continuación:

1. El número de características de nuestro modelo es igual a 4: aplica a variable n_features.

2. La variable picos de demanda se dividirá en datos de entrenamiento y validación al igual se realizó con las demás.

3. La función data_supervisión recibirá un argumento adicional para esta nueva variable, además de la incorporación de ésta en el subarray que almacena las características. Los datos de entrenamiento de entrada al modelo pasarán a tener la forma: (muestras, periodos = 7, características = 4).

Cabe destacar, que la variable picos de demanda no requiere de normalización puesto que es una variable binaria, siendo por tanto una forma ya normalizada con la que nuestra red puede trabajar fácilmente.

De nuevo, se llama a la función fit para comenzar con el entrenamiento de la red neuronal que se compone de la siguiente estructura y características internas.

Tabla 5.2 Parámetros seleccionado del modelo LSTM Multivariable mejorado[Fuente: Elaboración propia].

Parámetros	Valor
Capas	2
Nodos por capa	128
Función de activación	Unidad Lineal Rectificada (Relu)
Ratio de aprendizaje (Lr)	0.001
Función error	Error de media absoluta (MAE)
Optimizador	RMSprop
Epochs	25

A continuación, se presentan a modo de resumen los resultados obtenidos para la simulación que se está estudiando junto con los gráficos de desempeño del modelo por cada epoch.

Error	Valor
Min MAE Validación	0,3072
MAE Entrenamiento	0,2661
RMS	71,73

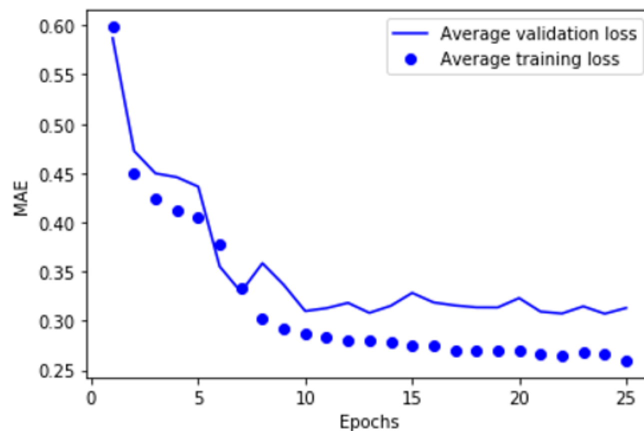

Figura 5.7 Resultados computacionales del modelo [Fuente: Elaboración propia].

Observando la figura 5.7, podemos concluir que ha habido una mejora significativa en el pronóstico del modelo, en términos de error RMS, ha pasado de un valor de 108 a un valor de 71,72, es decir, una reducción del error de un 34 %. También se puede apreciar la disminución en el valor del MAE de entrenamiento tal y como era de esperar puesto que una disminución del MAE de validación trae consigo una disminución en los datos de entrenamiento, pero no viceversa. A continuación, se muestra un gráfico comparativo entre los valores pronosticados y los reales durante el año 2019 (periodo de validación).

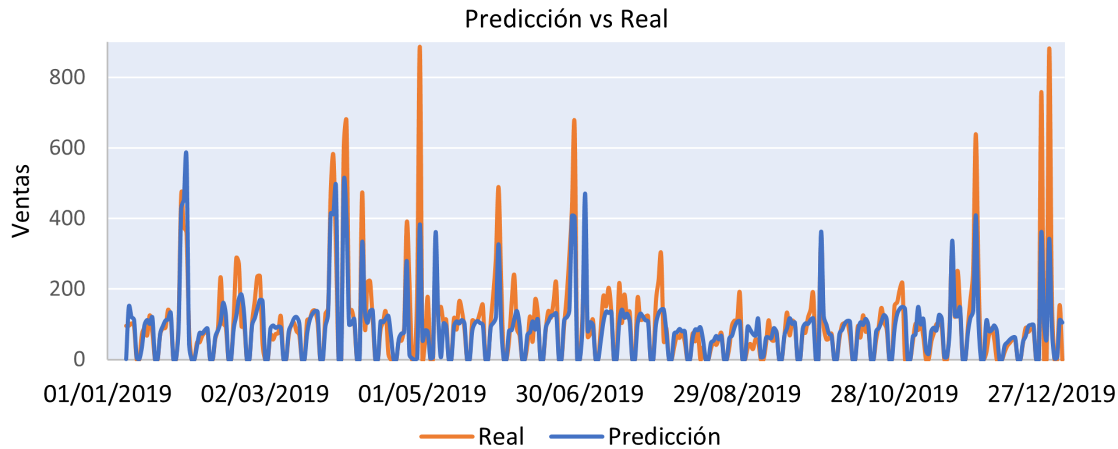

Figura 5.8 Evolución temporal datos pronosticados y reales [Fuente: Elaboración propia].

Por último, analizando la figura 5.8, el pronóstico realizado por el modelo es más cercano a los valores reales de los picos de demanda ya que se han generado nuevas correlaciones durante la fase de entrenamiento entre la variable picos de demanda y los datos de venta del producto.

Eficiencia frente a métodos clásicos

Para determinar la eficiencia de los modelos Deep Learning aplicados a nuestro caso práctico frente a otros métodos clásicos, se ha realizado un problema de forecasting al caso que se nos presenta mediante el método clásico Holt-Winters debido a los siguientes motivos:

1. Existe una estacionalidad semanal en el número de ventas.

2. El método Holt-Winters es uno de los métodos más empleados debido a su adecuado desempeño y eficiencia en los resultados de predicción.

El modelo, el cual trabajará de forma univariable, recogerá como entrenamiento, al igual que se ha realizado con los modelos Deep Learning, los primeros cuatro años de ventas y se hará un pronóstico para el siguiente año completo (2019). Es importante destacar que, puesto que realizar una sola o pocas predicciones mediante este método clásico puede llevar a resultados que están fuera de la realidad debido a los picos que se presentan de demanda, se va a realizar una simulación con aprendizaje progresivo, esto es, conforme se vaya realizando un pronóstico para un número de días (siete días), esos pronósticos se incorporarán al entrenamiento del modelo para la próxima iteración. A continuación, se presentan a modo de resumen el resultado obtenido mediante la aplicación del método Holt-Winter a nuestro caso práctico junto con el gráfico comparativo de los valores pronosticados y los valores reales en el periodo de estudio.

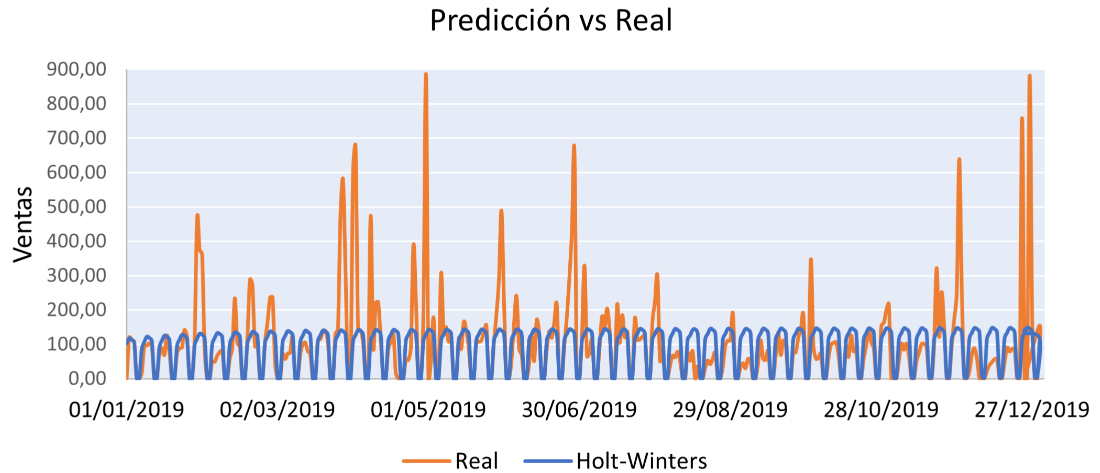

Figura 5.9 Evolución temporal datos pronosticados y reales (Holt-Winters) [Fuente: Elaboración propia].

El modelo ha obtenido un error RMS de 113,53, algo esperado debido a los picos de demanda que presenta los datos de ventas, los cuales generan un mayor error en el RMS. Se puede apreciar, como para nuestro caso práctico, el método clásico ha mostrado unos peores resultados que el último método Deep Learning mejorado introducido anteriormente el cual se obtuvo un error RMS de 71,73, obteniendo este último una mayor eficiencia.

El modelado computacional en Python queda recogido en el apéndice A.5.

6 Conclusiones

Como conclusión, se van a describir los puntos más importantes que componen el proyecto y finalizar con el análisis y valoraciones finales de los resultados obtenidos, entre ellos, el cumplimiento de los objetivos establecidos en el objeto del proyecto.

- En primer lugar, se ha planteado el problema de predicción de la demanda, para, posteriormente, realizar un estudio de los métodos más populares, así como de las características y comportamientos fundamentales que presentan las series temporales. Posteriormente, se ha realizado una introducción a la Inteligencia Artificial, proponiendo diferentes modelos de resolución para estos tipos de problemas mediante Deep Learning que minimicen el error de predicción: LSTM Univariable, CNN Univariable y LSTM Multivariable.

- Una vez estudiadas las metodologías de resolución con IA, se han aplicado dichos modelos a un caso práctico de predicción del número de ventas en el sector automovilístico con objeto de reducir en mayor medida el error que se genera entre los valores reales de demanda y los valores pronosticados. Los modelos Deep Learning aplicados al problema real se han modelado computacionalmente en lenguaje Python debido al potencial de la librería Keras para el modelado y resolución de estos tipos de problemas. Se ha conseguido trasladar series de datos en matrices que se han comportado como entrada de las redes neuronales.

- Respecto a los modelos Deep Learning aplicados a nuestro caso práctico, se ha podido analizar su buena respuesta y el correcto desempeño, especialmente del modelo LSTM Multivariable, obteniendo una minimización del error en el pronóstico de las ventas del producto. Por otro lado, se ha podido corroborar la importancia y la influencia de las variables de entrada al modelo de los problemas multivariables en los resultados del pronóstico para que el modelo sea capaz de encontrar las diferentes correlaciones no lineales y obtener una mayor minimización del error.

- Debido a la naturaleza del Deep Learning, se ha analizado y puesto en valor la relevancia del número de datos de entrada para el correcto entrenamiento, y por tanto, correcto desempeño del modelo, habiéndose podido obtener una mayor eficiencia de predicción en caso de disponer de más cantidad de datos/variables.

- Aunque se haya reflejado una mayor eficiencia en los modelos Deep Learning empleados para nuestro caso práctico frente a otros métodos clásicos, no podemos generalizar afirmando que los métodos Deep Learning presentan un mejor desempeño obteniendo un menor error de predicción en los problemas de predicción de la demanda, ya que el tipo de serie temporal presenta un papel crucial en el desempeño del modelo empleado. Sin embargo, podemos afirmar y queda demostrado en el presento proyecto, el potencial y la eficiencia de los modelos Deep Learning, los cuales, son cada

vez más empleados en los problemas forecasting ya que son capaces de trabajar una gran cantidad de patrones mediante formulación no lineal y de esta forma, mediante un número considerable de variables de entrenamiento, poder pronosticar de manera muy efectiva.

Apéndice A
Codificación en Python

A.1 Tratamiento de Datos

```
from pandas import read_csv
from sklearn.metrics import mean_squared_error
from matplotlib import pyplot
import calendar
import pandas as pd

a_0 = 2015
m_0 = 1
marca_dato = 'SEAT'
modelo_dato ='LEON'

p = 0
q = 0
n = 0
while p < 6:
    while q < 12:
        a = str(a_0+p)
        m = str(m_0+q)
        if len(m) == 1:
            m = '0' + m
        if a == '2020':
            if m == '05':
                break
        f = open('export_mensual_mat_' + a + m + '.txt', "r", errors = 'ignore')
        out_file = open('export_mensual_mat_'+ a + m + '_MODIFICADO.txt', "w")
        f1 = f.readlines()

        for x in f1[1:]:
          x= x[:8] + ',' +x[8:9] + ',' + x[9:17] + ',' + x[17:47] + ',' + x[47:69] + '
              ,' + x[69:]
          out_file.write(x)

        my_cols = [str(i) for i in range(60)]
```

```
#datos = read_csv('export_mensual_mat_201412_MODIFICADO.txt', sep=",",
    names=my_cols, engine='python', usecols = ['0','3','4'],
    infer_datetime_format =True, parse_dates ={'Fecha ':[0]}, index_col =['Fecha
    '])
datos = read_csv('export_mensual_mat_'+ a + m + '_MODIFICADO.txt', sep=",",
    names=my_cols, engine='python', usecols = ['0','3','4'])
datos.columns = ['Fecha','Marca', 'Modelo']
datos['Marca']= datos['Marca'].str.strip()
datos['Modelo']= datos['Modelo'].str.strip()
#sep="\s +|,|:"
datos = datos.sort_values(by = 'Fecha')
datos = datos.reset_index(drop=True)
datos['Fecha'] = datos['Fecha'].astype(str)


modificar = datos['Fecha']

for i in range(len(modificar)):
    if len(modificar[i]) != 8:
        modificar[i] = '0' + modificar[i]

datos['Fecha'] = modificar
datos.set_index('Fecha', inplace=True)
datos.index = pd.to_datetime(datos.index, format ='%d%m%Y')

año = datos.index.year[0].astype(str)
mes = datos.index.month[0].astype(str)

def filtrar_modelo (marca, modelo):
    #datos_modelo = datos[datos.Marca.isin([marca]) & datos.Modelo.isin([
        modelo])]
    datos_modelo = datos[datos.Marca.isin([marca]) & datos['Modelo'].str.
        contains(modelo, case = False)]
    return datos_modelo
#------------------------------------------------
#Unidades por modelo
datos = filtrar_modelo (marca_dato, modelo_dato)
datos['Unidades'] = 1
datos = datos.groupby(datos.index).agg({'Marca': ' first ',
                                        'Modelo': ' first ',
                                        'Unidades':sum})
datos = datos[['Unidades']]
#------------------------------------------------
#Limpiar fechas no correspondientes
datos = datos.loc[a + '-' + m]
#------------------------------------------------

def missing_dates(mes,año,datos):
    dias_totales = calendar.monthrange(int(año), int(mes))
    dias_totales = str( dias_totales [1])
```

```
            idx = pd.date_range(año + '-' + mes + '-' + '01', año + '-' + mes + '-' +
                dias_totales )
            datos.index = pd.DatetimeIndex(datos.index)
            datos = datos.reindex(idx,  fill_value =0)
            return datos

        datos = missing_dates(m,a,datos)
        datos.to_csv(a + m + '_' + marca_dato + '_' + modelo_dato + '.txt', header=
            False, index=True, sep='\t', mode='a')
        q = q+1
        print(65-n)
        n = n+1
    q = 0
    p = p+1

filenames = []

a_0 = 2015
m_0 = 1
p=0
q=0
while p < 6:
    while q < 12:
        a = str(a_0+p)
        m = str(m_0+q)
        if len(m) == 1:
            m = '0' + m
        if a == '2020':
            if m == '05':
                break
        filenames.append(a + m + '_SEAT_LEON.txt')
        q = q+1
    q=0
    p = p+1

with open('SEAT_LEON.txt', 'w') as outfile :
    for fname in filenames:
        with open(fname) as infile :
            for line in infile :
                outfile . write( line )
```

A.2 Modelo 1. LSTM Univariable

```
from pandas import read_csv
from sklearn.metrics import mean_squared_error
from matplotlib import pyplot
import calendar
import pandas as pd
import numpy as np
```

```
from numpy import array

from keras import layers

import matplotlib.pyplot as plt

from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dense
from keras.optimizers import RMSprop, Adam

from sklearn.metrics import mean_squared_error, mean_absolute_error
from math import sqrt

my_cols = [str(i) for i in range(2)]
datos = read_csv('SEAT_LEON.txt', sep="\t", names=my_cols, engine='python')
datos.columns = ['Fecha','Unidades']
datos.set_index('Fecha', inplace=True)
datos.index = pd.to_datetime(datos.index, format ='%Y-%m-%d')

def interpolacion_festivos (data):
    for i in range(len(data)-1):
        if data[i] == 0:
            if data[i+1] == 1:
                data[i] = 0
            else:
                if data[i+1] == 0:
                    data[i] = data[i+2]//3
                    n = data[i+2] - data[i]
                    data[i+1] = n//2
                    data[i+2] = data[i+2] - data[i] -data[i+1]
                else:
                    data[i] = data[i+1]//2
                    data[i+1] = data[i+1] - data[i]
    return data

datos = datos['Unidades']
#datos = interpolacion_festivos (datos)
datos = datos[:-4]
datos = datos[:-5*15]
datos = datos[:-7]
datos = datos[:-35]
#datos = datos[262:522]

train = datos

def normalizar(train):
    mean_train = train.mean(axis=0)
    train -= mean_train
    std_train = train.std(axis=0)
```

```
    train /= std_train
    return train , mean_train, std_train

train , mean_train, std_train = normalizar( train )

n_features = 1

 train_inicio = train

def data_supervision ( past_values , future_values , datos1 ):
    ''' Creamos dos arrays que almacenarán los valores previos y posteriores
        respectivamente '''
    entra , sale = list () , list ()
    ''' 'Por cada array de entrada y salida , tendremos "len(datos1 )−past_values−
        future_values +1" subarrays que almacenarán los timesteps '''
    for i in range(len(datos1 )−past_values−future_values +1):
        x,y = list () , list ()
        ''' Por cada subarray , se incorporan las caracter í sticas en un nuevo subarray
            '''
        for j in range( past_values ):
            p = list ()
            p.append(datos1 [ j+i ])
            x.append(p)
        for j in range( future_values ):
            y.append(datos1 [ past_values +j+i ])
        entra .append(x)
        sale .append(y)
    return np. array ( entra ) , np. array ( sale )

num_epochs = 200
 all_mae_histories_val = []
 all_mae_histories_loss = []
k = 4
dias_semana = 7
pasos = 1
for i in range (1,k):
    print (' processing␣fold␣#', i )
    if i == 1:
        val = train_inicio [365∗2+1:365∗3+1]
        val_x, val_y = data_supervision (dias_semana,pasos, val )
        val_x = val_x . reshape (( val_x . shape [0], val_x . shape [1], n_features ))
        train = train_inicio [:365∗2+1]
        train_x , train_y = data_supervision (dias_semana,pasos, train )
        train_x = train_x . reshape (( train_x . shape [0], train_x . shape [1], n_features ))

    if i == 2:
        val = train_inicio [365∗3+1:365∗4+1]
        val_x, val_y = data_supervision (dias_semana,pasos, val )
        val_x = val_x . reshape (( val_x . shape [0], val_x . shape [1], n_features ))
        train = train_inicio [:365∗3+1]
```

```
        train_x , train_y = data_supervision (dias_semana,pasos, train )
        train_x = train_x . reshape (( train_x . shape [0], train_x . shape [1], n_features ))

    if i == 3:
        val = train_inicio [365*4+1:]
        val_x, val_y = data_supervision (dias_semana,pasos, val)
        val_x = val_x . reshape (( val_x . shape [0], val_x . shape [1], n_features ))
        train = train_inicio [:365*4+1]
        train_x , train_y = data_supervision (dias_semana,pasos, train )
        train_x = train_x . reshape (( train_x . shape [0], train_x . shape [1], n_features ))

    model = Sequential ()
    model.add(LSTM(32, activation=' relu ' , input_shape=(dias_semana, n_features ) ,
        return_sequences =True))
    model.add(LSTM(32, activation=' relu ' ))
    model.add(Dense(pasos))
    #model.compile(optimizer='adam', loss='mse', metrics =[' acc '])
    model.compile(optimizer=Adam(lr = 0.001) , loss='mae', metrics=[' acc' ])
    history = model. fit ( train_x , train_y ,
                            validation_data =(val_x, val_y) ,
epochs=num_epochs, batch_size=14, verbose=1)
    mae_history_val = history . history [ ' val_loss ' ]
    all_mae_histories_val .append(mae_history_val)
    mae_history_loss = history . history [ ' loss ' ]
    all_mae_histories_loss .append(mae_history_loss)

average_mae_history_val = [
np.mean([x[i] for x in all_mae_histories_val ]) for i in range(num_epochs)]
average_mae_history_loss = [
np.mean([x[i] for x in all_mae_histories_loss ]) for i in range(num_epochs)]
plt . plot (range (1, len ( average_mae_history_val ) + 1), average_mae_history_val , 'b' , label
    = 'Average␣validation␣loss ' )
plt . plot (range (1, len ( average_mae_history_val ) + 1), average_mae_history_loss , 'bo',
    label = 'Average␣training␣loss ' )
plt . xlabel ( 'Epochs')
plt . ylabel ( 'MAE')
plt . legend ()
plt . show()
#print (np.where(average_mae_history_val == min(average_mae_history_val)))

print ( 'mae␣average:' , min(average_mae_history_val))
print ( 'mean_train: ' ,mean_train)
print ( ' std_train : ' , std_train )
print ( 'Delta_x: ' , min(average_mae_history_val)* std_train )

''' test = residual [-31:]
test_x , test_y = data_supervision (7, test )
test_x = test_x [0]
print ( test_x . shape)
test_x = test_x . reshape ((1, test_x . shape [0], n_features ))
```

```
y_predict = model.predict(test_x)
print(len(y_predict))
print(y_predict)
y_real = test_y[0]
print(y_real)'''

#Gráficas Val_x VS Predicción

'''Predecimos los valores de validación'''
y_predict_nor = model.predict(val_x)
'''Creamos un array que contendrá dichos valores'''
y_predict= []
'''Añadimos los resultados en el array'''
for i in range(len(y_predict_nor)):
    y_predict.append(y_predict_nor[i][0])

'''De-normalizamos los resultados'''
for i in range(len(y_predict)):
    y_predict[i] = y_predict[i]* std_train +mean_train

'''Igualamos a cero los valores cercanos'''
for i in range(len(y_predict)):
    if y_predict[i] < 1:
        y_predict[i] = 0

'''Array con los valores de validación reales'''
y_real = val[7:]
y_real = y_real.values

'''De-normalizamos los valores de validación reales'''
for i in range(len(y_real)):
    y_real[i] = y_real[i]* std_train +mean_train

'''Calculamos el MSE de la predicción del modelo y los valores reales'''
rms = sqrt(mean_squared_error(y_real, y_predict))
mae_result = mean_absolute_error(y_real, y_predict)
print(rms)
print(mae_result)

'''Imprimimos por pantalla la comparativa resultados del modelo-valores reales'''
plt.figure(figsize=(30,10))
plt.plot(y_predict, 'b', y_real,'g')
plt.title('Training and validation loss')
plt.xlabel('Día')
plt.ylabel('Unidades')
plt.legend()

plt.show()
```

A.3 Modelo 2. CNN Univariable

```
from pandas import read_csv
from sklearn.metrics import mean_squared_error
from matplotlib import pyplot
import calendar
import pandas as pd
import numpy as np
from numpy import array
from statsmodels.graphics.tsaplots import plot_pacf, plot_acf


from keras import layers

import matplotlib.pyplot as plt

from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dense
from keras.optimizers import RMSprop, Adam

from keras.layers import Flatten
from keras.layers.convolutional import Conv1D
from keras.layers.convolutional import MaxPooling1D

my_cols = [str(i) for i in range(2)]
datos = read_csv('SEAT_LEON.txt', sep="\t", names=my_cols, engine='python')
datos.columns = ['Fecha','Unidades']
datos.set_index('Fecha', inplace=True)
datos.index = pd.to_datetime(datos.index, format ='%Y-%m-%d')

#---------------ELIMINAR FINES DE SEMANA
#datos = datos[datos.index.dayofweek < 5]

def interpolacion_festivos (data):
    for i in range(len(data)-1):
        if data[i] == 0:
            if data[i+1] == 1:
                data[i] = 0
            else:
                if data[i+1] == 0:
                    data[i] = data[i+2]//3
                    n = data[i+2] - data[i]
                    data[i+1] = n//2
                    data[i+2] = data[i+2] - data[i] -data[i+1]
                else:
                    data[i] = data[i+1]//2
                    data[i+1] = data[i+1] - data[i]
    return data
```

```
datos = datos['Unidades']

datos = datos[:-4]
datos = datos[:-5*15]
datos = datos[:-7]
datos = datos[:-35]
datos = datos.astype('float32')

train = datos

def normalizar(train):
    mean_train = train.mean(axis=0)
    train -= mean_train
    std_train = train.std(axis=0)
    train /= std_train
    return train, mean_train, std_train

train, mean_train, std_train = normalizar(train)

n_features = 1

train_inicio = train


def data_supervision(past_values, future_values, datos1):
    '''Creamos dos arrays que almacenarán los valores previos y posteriores
        respectivamente'''
    entra, sale = list(), list()
    '''’Por cada array de entrada y salida, tendremos "len(datos1)-past_values-
        future_values+1" subarrays que almacenarán los timesteps'''
    for i in range(len(datos1)-past_values-future_values+1):
        x,y = list(), list()
        '''Por cada subarray, se incorporan las características en un nuevo subarray
            '''
        for j in range(past_values):
            p = list()
            p.append(datos1[j+i])
            x.append(p)
        for j in range(future_values):
            y.append(datos1[past_values+j+i])
        entra.append(x)
        sale.append(y)
    return np.array(entra), np.array(sale)

num_epochs = 100
batch = 14
all_mae_histories_val = []
```

```
all_mae_histories_loss = []
k = 4
past_values = 14
future_values = 1

for i in range(1,k):
    print('processing fold #', i)
    if i == 1:
        val = train_inicio[365*2+1:365*3+1]
        val_x, val_y = data_supervision(past_values, future_values, val)
        val_x = val_x.reshape((val_x.shape[0], val_x.shape[1], n_features))
        train = train_inicio[:365*2+1]
        train_x, train_y = data_supervision(past_values, future_values, train)
        train_x = train_x.reshape((train_x.shape[0], train_x.shape[1], n_features))

    if i == 2:
        val = train_inicio[365*3+1:365*4+1]
        val_x, val_y = data_supervision(past_values, future_values, val)
        val_x = val_x.reshape((val_x.shape[0], val_x.shape[1], n_features))
        train = train_inicio[:365*3+1]
        train_x, train_y = data_supervision(past_values, future_values, train)
        train_x = train_x.reshape((train_x.shape[0], train_x.shape[1], n_features))

    if i == 3:
        val = train_inicio[365*4+1:]
        val_x, val_y = data_supervision(past_values, future_values, val)
        val_x = val_x.reshape((val_x.shape[0], val_x.shape[1], n_features))
        train = train_inicio[:365*4+1]
        train_x, train_y = data_supervision(past_values, future_values, train)
        train_x = train_x.reshape((train_x.shape[0], train_x.shape[1], n_features))

    n_filters = 256
    n_kernel = 2
    model = Sequential()
    model.add(Conv1D(filters= n_filters, kernel_size=n_kernel, activation='relu',
    input_shape=(14, 1)))
    model.add(Conv1D(filters= n_filters, kernel_size=n_kernel, activation='relu'))
    model.add(MaxPooling1D(pool_size=2))
    model.add(Flatten())
    model.add(Dense(32))
    model.add(Dense(1))
    #model.compile(optimizer='adam', loss='mse',metrics=['acc'])
    model.compile(optimizer=Adam(lr = 0.001), loss='mae', metrics=['acc'])
    history = model.fit(train_x, train_y,
                        validation_data=(val_x, val_y),
epochs=num_epochs, batch_size=15, verbose=1)
    mae_history_val = history.history['val_loss']
    all_mae_histories_val.append(mae_history_val)
    mae_history_loss = history.history['loss']
    all_mae_histories_loss.append(mae_history_loss)
```

```
average_mae_history_val = [
np.mean([x[i] for x in all_mae_histories_val ]) for i in range(num_epochs)]
average_mae_history_loss = [
np.mean([x[i] for x in all_mae_histories_loss ]) for i in range(num_epochs)]
plt.plot(range(1, len(average_mae_history_val) + 1), average_mae_history_val,'b', label
    = 'Average validation loss')
plt.plot(range(1, len(average_mae_history_val) + 1), average_mae_history_loss,'bo',
    label = 'Average training loss')
plt.xlabel('Epochs')
plt.ylabel('MAE')
plt.legend()
plt.show()

print('mae average:',min(average_mae_history_val))
print('mean_train:',mean_train)
print(' std_train :', std_train )
print('Delta_x:', min(average_mae_history_val)* std_train )

print( average_mae_history_loss [15])
print(np.where(average_mae_history_val == min(average_mae_history_val)))



print(model.summary())
```

A.4 Modelo 3. LSTM Multivariable

```
import pandas as pd
import matplotlib.pyplot as plt

from keras import layers
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dense
from keras.optimizers import RMSprop, Adam, SGD, Adadelta

import numpy as np
from math import sqrt
from sklearn.metrics import mean_squared_error, mean_absolute_error
import calendar
import random

''' Lectura de los datos de ventas del producto '''
my_cols = [ str(i) for i in range(2)]
datos = pd.read_csv('SEAT_LEON.txt', sep="\t", names=my_cols, engine='python')
datos.columns = ['Fecha','Unidades']
datos.set_index('Fecha', inplace=True)
datos.index = pd.to_datetime(datos.index, format ='%Y-%m-%d')
```

```
#DATOS VENTAS

#DATOS PUNTUALES PICOS
picos = datos.copy()
for col in picos.columns:
    picos[col].values[:] = 0

picos = picos[:-4]
picos = picos[:-5*15]
picos = picos[:-7]
picos = picos[:-35]

''' Limpieza de datos de ventas del producto '''
datos = datos[:-4]
datos = datos[:-5*15]
datos = datos[:-7]
datos = datos[:-35]
datos = datos['Unidades']
datos = datos.astype('float32')
print(datos)
print(datos[:-30])
datos2 = datos.copy()

#DATOS PUNTUALES PICOS
for i in range(len(datos.values)):
    if datos.values[i]>300:
        picos['Unidades'][i-1] = 1
#random.randint(1,3)
picos = picos['Unidades']
print(picos[:-30])

''' Lectura de datos de la variable Ibex35 '''
#DATOS IBEX
datos_ibex = pd.read_csv('IBEX.txt', sep=",", header=0)
datos_ibex.set_index('Fecha', inplace=True)
''' Comprobar si hay valores NaN'''
#print(np.where(datos_ibex['Cierre'].isna())[0])

datos_ibex = datos_ibex['Ibex35']
datos_ibex = datos_ibex.astype('float32')

''' Lectura de datos de la variable Gasolina95 '''
#DATOS GASOLINA 95
datos_gasolina95 = pd.read_csv('PreciosGasolina.txt', sep=",", header=0)
datos_gasolina95.set_index('Fecha', inplace=True)
datos_gasolina95 = datos_gasolina95[:-59]
''' Comprobar si hay valores NaN'''
#print(np.where(datos_gasolina95['Precio'].isna())[0])
datos_gasolina95 = datos_gasolina95['Gasolina95']
```

```
datos_gasolina95 = datos_gasolina95 . astype ( ' float32 ' )

''' Modelado función Normalizar '''
def normalizar ( train ) :
    mean_train = train . mean(axis=0)
    train -= mean_train
    std_train = train . std ( axis =0)
    train /= std_train
    return train , mean_train , std_train

datos_norm, mean_train_datos , std_train_datos = normalizar ( datos )
ibex35_norm, mean_train_ibex35 , std_train_ibex35 = normalizar ( datos_ibex )
gasolina95_norm, mean_train_gasolina95 , std_train_gasolina95 = normalizar (
    datos_gasolina95 )
picos_norm = picos

''' Definici ón de la funci ón data_supervised '''
''' Definimos los argumentos de entrada '''
def data_supervision ( past_values , future_values , datos1 , datos2 , datos3 , datos4 ) :
    ''' Creamos dos arrays que almacenarán los valores previos y posteriores
        respectivamente '''
    entra , sale = list () , list ()
    ''' 'Por cada array de entrada y salida , tendremos "len ( datos1 )−past_values−
        future_values +1" subarrays que almacenarán los timesteps '''
    for i in range( len ( datos1 )−past_values−future_values +1):
        x,y = list () , list ()
        ''' Por cada subarray , se incorporan las caracter í sticas en un nuevo subarray
            '''
        for j in range( past_values ) :
            p = list ()
            p.append(datos1 [ j+i ])
            p.append(datos2 [ j+i ])
            p.append(datos3 [ j+i ])
            p.append(datos4 [ j+i ])
            x.append(p)
        for j in range( future_values ) :
            y.append(datos1 [ past_values +j+i ])
        entra . append(x)
        sale . append(y)
    return np. array ( entra ) , np. array ( sale )

num_epochs = 23
batch = 14
all_mae_histories_val = []
all_mae_histories_loss = []
k = 4
past_values = 7
future_values = 1
n_features = 4
```

```
for i in range(3,k):
    print('processing fold #', i)
    if i == 1:
        train_datos = datos_norm[:365*2+1]
        train_gasolina95 = gasolina95_norm[:365*2+1]
        train_ibex = ibex35_norm[:365*2+1]

        train_x, train_y = data_supervision(past_values, future_values, train_datos,
            train_gasolina95, train_ibex)

        val_datos = datos_norm[365*2+1:365*3+1]
        val_gasolina95 = gasolina95_norm[365*2+1:365*3+1]
        val_ibex = ibex35_norm[365*2+1:365*3+1]

        val_x, val_y = data_supervision(past_values, future_values, val_datos,
            val_gasolina95, val_ibex)

        #val_x = val_x.reshape((val_x.shape[0], val_x.shape[1], n_features))
        #train_x = train_x.reshape((train_x.shape[0], train_x.shape[1], n_features))
    if i == 2:
        train_datos = datos_norm[:365*3+1]
        train_gasolina95 = gasolina95_norm[:365*3+1]
        train_ibex = ibex35_norm[:365*3+1]

        train_x, train_y = data_supervision(past_values, future_values, train_datos,
            train_gasolina95, train_ibex)

        val_datos = datos_norm[365*3+1:365*4+1]
        val_gasolina95 = gasolina95_norm[365*3+1:365*4+1]
        val_ibex = ibex35_norm[365*3+1:365*4+1]

        val_x, val_y = data_supervision(past_values, future_values, val_datos,
            val_gasolina95, val_ibex)

    if i == 3:

        train_datos = datos_norm[:365*4+1]
        train_gasolina95 = gasolina95_norm[:365*4+1]
        train_ibex = ibex35_norm[:365*4+1]
        train_picos = picos_norm[:365*4+1]


        train_x, train_y = data_supervision(past_values, future_values, train_datos,
            train_gasolina95, train_ibex, train_picos)

        val_datos = datos_norm[365*4+1:]
        val_gasolina95 = gasolina95_norm[365*4+1:]
        val_ibex = ibex35_norm[365*4+1:]
        val_picos = picos_norm[365*4+1:]
```

```
        val_x, val_y = data_supervision ( past_values , future_values , val_datos ,
            val_gasolina95 , val_ibex , val_picos )
        #train_x = train_x . reshape (( train_x . shape [0], train_x . shape [1], n_features ))
    ''' Construcción del modelo multivariable LSTM con Dropout''
    ''' Llamamos al API Sequential '''
    model = Sequential ()
    ''' Construimos sobre la interfaz 4 capas con 124 cada una'''
    model.add(LSTM(128, activation='relu', input_shape=( past_values , n_features ),
        return_sequences=True))
    #model.add( layers .Dropout(0.5))
    model.add(LSTM(128, activation='relu', input_shape=( past_values , n_features )))
    #model.add( layers .Dropout(0.5))
    ''' Construimos una capa de salida para la predicción del valor futuro '''
    model.add(Dense( future_values ))
    #model.compile( optimizer='adam', loss='mse',metrics =['acc'])
    ''' Llamamos a la función compile para definir el optimizador, el ratio de
        aprendizaje , la función objetivo y las métricas a analizar '''
    model.compile( optimizer=Adam(lr = 0.001), loss='mae', metrics=['acc'])
    #Optimizers: RMSprop, Adam(32,lr 0.001, batch 14, past=14 o 7, future=1): MAE
        0,4241, SGD, Adadelta
    #Loss: mae for multioutput
    ''' Almacenamos los resultados de entramiento y error del modelo en la variable
        history , junto con la definición de los epochs y tamaño del batch '''
    history = model. fit ( train_x , train_y ,
                           validation_data =(val_x, val_y),
epochs=num_epochs, batch_size=batch, verbose=1)
    mae_history_val = history . history ['val_loss']
    all_mae_histories_val .append(mae_history_val)
    mae_history_loss = history . history ['loss']
    all_mae_histories_loss .append(mae_history_loss)

average_mae_history_val = [
np.mean([x[i] for x in all_mae_histories_val ]) for i in range(num_epochs)]
average_mae_history_loss = [
np.mean([x[i] for x in all_mae_histories_loss ]) for i in range(num_epochs)]
plt . plot (range (1, len( average_mae_history_val ) + 1), average_mae_history_val ,'b', label
    = 'Average␣validation␣loss')
plt . plot (range (1, len( average_mae_history_val ) + 1), average_mae_history_loss ,'bo',
    label = 'Average␣training␣loss')
plt . xlabel ('Epochs')
plt . ylabel ('MAE')
plt .legend ()
plt .show()

print ('mae␣average:',min(average_mae_history_val))
print ('mean_train:',mean_train_datos)
print (' std_train :', std_train_datos )
print ('Delta_x:', min(average_mae_history_val)* std_train_datos )
```

```
#
  ----------------------------------------------------------------------------


#Gráficas Val_x VS Predicción

'''Predecimos los valores de validación'''
y_predict_nor = model.predict(val_x)
'''Creamos un array que contendrá dichos valores'''
y_predict= []
'''Añadimos los resultados en el array'''
for i in range(len(y_predict_nor)):
    y_predict.append(y_predict_nor[i][0])
'''De-normalizamos los resultados'''
for i in range(len(y_predict)):
    y_predict[i] = y_predict[i]* std_train_datos +mean_train_datos

'''Igualamos a cero los valores cercanos'''
for i in range(len(y_predict)):
    if y_predict[i] < 1:
        y_predict[i] = 0

'''Array con los valores de validación reales'''
datos_real = datos_norm[365*4+1+7:]
datos_real = datos_real.values

'''De-normalizamos los valores de validación reales'''
for i in range(len(datos_real)):
    datos_real[i] = datos_real[i]* std_train_datos +mean_train_datos

'''Calculamos el MSE de la predicción del modelo y los valores reales'''
rms = sqrt(mean_squared_error(datos2[365*4+1+7:], y_predict))
print(rms)

'''Imprimimos por pantalla la comparativa resultados del modelo-valores reales'''
prediccion = y_predict
real = datos_real
plt.figure(figsize =(30,10))
plt.plot(prediccion, 'b', label = 'Predicción')
plt.plot(datos2[365*4+1+7:].values,'g', label = 'Real')
plt.title('Predicción vs Real')
plt.xlabel('Día')
plt.ylabel('Unidades')
plt.legend()
plt.show()

plt.plot(range(1, len(val_loss_drop) + 1), val_loss_drop,'r', label = 'Dropout')
plt.plot(range(1, len(average_mae_history_val) + 1), val_loss,'b', label = 'Normal')
plt.xlabel('Epochs')
plt.ylabel('Validation Loss')
```

```
plt.legend()
plt.show()
```

A.5 Holt-Winters

```
from pandas import read_csv
from sklearn.metrics import mean_squared_error
from matplotlib import pyplot, figure
import calendar
import pandas as pd
import numpy as np
from numpy import array
from math import sqrt

import matplotlib.pyplot as plt
from statsmodels.tsa.api import ExponentialSmoothing, SimpleExpSmoothing, Holt
from statsmodels.graphics.tsaplots import plot_pacf
from statsmodels.graphics.tsaplots import plot_acf
from pandas import Series

my_cols = [str(i) for i in range(2)]
datos = read_csv('SEAT_LEON.txt', sep="\t", names=my_cols, engine='python')
datos.columns = ['Fecha','Unidades']
datos.set_index('Fecha', inplace=True)
datos.index = pd.to_datetime(datos.index, format ='%Y-%m-%d')

datos = datos[:-4]
datos = datos[:-5*15]
datos = datos[:-7]
datos = datos[:-35]
datos = datos.astype('float32')
print(datos)

from statsmodels.tsa.stattools import adfuller

datos_orig = datos.copy()

train_datos = datos[:365*4+1]
val_datos = datos[365*4+1:365*4+1+7]

y_hat_avg = pd.DataFrame()
print( train_datos )
train_datos = datos[:365*4+1]
for i in range(52):
    print("Predicción Semana ", i)
    train_datos = pd.concat([ train_datos , y_hat_avg], axis=0)
    val_datos = datos[365*4+1+(7*i):365*4+1+7+(7*i)]

    y_hat_avg = val_datos.copy()
```

```
        fit1  = ExponentialSmoothing(np.asarray( train_datos ['Unidades']) , seasonal_periods
            =7 , trend='add', seasonal='add'). fit ()
        y_hat_avg['Holt-Winter'] = fit1 . forecast (len( val_datos ))
        ''' Igualamos a cero los valores cercanos '''
        for i in range(len(y_hat_avg)):
            if y_hat_avg['Holt-Winter'][i] < 1:
                y_hat_avg['Holt-Winter'][i] = 0
        print (y_hat_avg)

        del(y_hat_avg['Unidades'])
        y_hat_avg = y_hat_avg.rename(columns={'Holt-Winter':'Unidades'})

        rms = sqrt(mean_squared_error(val_datos['Unidades'], y_hat_avg["Unidades"]))
        plt . figure ( figsize =(16,8))
        plt . plot ( val_datos ['Unidades'], label='Real')
        plt . plot (y_hat_avg['Unidades'], 'r', label='Holt_Winter')
        plt .legend(loc='best')
        plt .show()

    train_datos  = pd.concat([ train_datos , y_hat_avg], axis=0)
    plt . figure ( figsize =(16,8))
    plt . plot ( train_datos ['Unidades'][365*4+1:], label='Holt-Winter')
    plt . plot (datos['Unidades'][365*4+1:], label='Real')
    plt .legend(loc='best')
    plt .show()

    rms = sqrt(mean_squared_error(datos['Unidades'][365*4+1:-1], train_datos ['Unidades'
        ][365*4+1:]))

    print (rms)
```

Bibliografía

[1] *Biblioteca informática*, October 2020, https://es.wikipedia.org/w/index.php?title=Biblioteca_-(inform%C3%A1tica)&oldid=130286670 [Último acceso: Noviembre 2020].

[2] *Keras: the Python deep learning API*, November 2020, https://keras.io/ [Último acceso: Noviembre 2020].

[3] *Matplotlib: Python plotting Matplotlib 3.3.3 documentation*, November 2020, https://matplotlib.org/ [Último acceso: Noviembre 2020].

[4] *pandas - Python Data Analysis Library*, November 2020, https://pandas.pydata.org/ [Último acceso: Noviembre 2020].

[5] Barbara Hammer, *Learning with Recurrent Neural Networks*, Springer, October 2007 (en), Google-Books-ID: H3_1BwAAQBAJ.

[6] Beverly Park Woolf, *Building Intelligent Interactive Tutors: Student-centered Strategies for Revolutionizing E-learning*, Morgan Kaufmann, July 2010 (en), Google-Books-ID: MnrUj3J_-VuEC.

[7] Bovas Abraham and Johannes Ledolter, *Statistical methods for forecasting*, Wiley series in probability and statistics, Wiley-Interscience, Hoboken, N.J, 2005 (en).

[8] Jason Brownlee, *Better deep learning: Train faster, reduce overfitting, and make better predictions*, Machine Learning Mastery, December 2018 (en).

[9] Christopher Chatfield, *Time-series forecasting*, Chapman & Hall/CRC, Boca Raton, 2001 (en).

[10] Christopher Olah, *Understanding lstm networks*, November 2020, https://colah.github.io/posts/2015-08-Understanding-LSTMs/ [Último acceso: Noviembre 2020].

[11] Douglas C. Montgomery, Cheryl L. Jennings, and Murat Kulahci, *Introduction to time series analysis and forecasting*, Wiley series in probability and statistics, Wiley-Interscience, Hoboken, N.J, 2008 (en), OCLC: ocn125406072.

[12] François Chollet, *Deep learning with Python*, Manning Publications Co, Shelter Island, New York, 2018 (en), OCLC: ocn982650571.

[13] Kevin Gurney, *An Introduction to Neural Networks*, CRC Press, December 2003 (en), Google-Books-ID: sn6oBHq8qQQC.

[14] Jason Brownlee, *Introduction to Time Series Forecasting With Python: How to Prepare Data and Develop Models to Predict the Future*, Machine Learning Mastery, February 2017 (en).

[15] Konuralp Ilbay, *A New Application of Recurrent Neural Networks for EMG-based Diagnosis of Carpal Tunnel Syndrome*, 2011 (en), OCLC: 1154273058.

[16] Li Deng and Dong Yu, *Deep Learning: Methods and Applications*, (2014) (en), Google-Books-ID: 46qNoAEACAAJ.

[17] M. Arif Wani, Farooq Ahmad Bhat, Saduf Afzal, and Asif Iqbal Khan, *Advances in Deep Learning*, Studies in Big Data, Springer Singapore, 2020 (en).

[18] Nan Zheng and Pinaki Mazumder, *Learning in Energy-Efficient Neuromorphic Computing: Algorithm and Architecture Co-Design*, John Wiley & Sons, December 2019 (en), Google-Books-ID: IvC0DwAAQBAJ.

[19] Peter J. Brockwell and Richard A. Davis, *Introduction to time series and forecasting*, 2nd ed ed., Springer texts in statistics, Springer, New York, 2002 (en).

[20] Ridha B.C. Gharbi and G. Ali Mansoori, *An introduction to artificial intelligence applications in petroleum exploration and production*, Journal of Petroleum Science and Engineering **49** (2005), no. 3-4, 93–96 (en).

[21] Robin Hyndman, Anne B. Koehler, J. Keith Ord, and Ralph D. Snyder, *Forecasting with Exponential Smoothing: The State Space Approach*, Springer Series in Statistics, Springer-Verlag, Berlin Heidelberg, 2008 (en).

[22] S. Hochreiter and J. Schmidhuber, *Long short-term memory*, Neural Computation **9** (1997), no. 8, 1735–1780 (eng).

[23] Salman Khan, Hossein Rahmani, Syed Afaq Ali Shah, and Mohammed Bennamoun, *A Guide to Convolutional Neural Networks for Computer Vision*, vol. 8, February 2018 (en).

[24] Shoba Ranganathan, Kenta Nakai, and Christian Schonbach, *Encyclopedia of Bioinformatics and Computational Biology: ABC of Bioinformatics*, August 2018, Google-Books-ID: rs51DwAAQBAJ.

[25] Tatyana I. Poznyak, Isaac Chairez Oria, and Alexander S. Poznyak, *Chapter3 - Background on dynamic neural networks*, Ozonation and Biodegradation in Environmental Engineering (Tatyana I. Poznyak, Isaac Chairez Oria, and Alexander S. Poznyak, eds.), Elsevier, January 2019, pp. 57–74 (en).

[26] Umberto Michelucci, *Advanced Applied Deep Learning: Convolutional Neural Networks and Object Detection*, Apress, Berkeley, CA, 2019 (en).

[27] Y. Bengio, P. Simard, and P. Frasconi, *Learning long-term dependencies with gradient descent is difficult*, IEEE Transactions on Neural Networks **5** (1994), no. 2, 157–166, Conference Name: IEEE Transactions on Neural Networks.

[28] Jian Zheng, Cencen Xu, Ziang Zhang, and Xiaohua Li, *Electric load forecasting in smart grids using Long-Short-Term-Memory based Recurrent Neural Network*, 2017 51st Annual Conference on Information Sciences and Systems (CISS) (Baltimore, MD, USA), IEEE, March 2017, 2017, pp. 1–6 (en).