Universidad de Valladolid

Escuela de Ingeniería Informática de Segovia

Trabajo Fin de Grado

Grado en Ingeniería Informática
de Servicios y Aplicaciones

Construcción de algoritmos de aprendizaje automático para predicción de consumos energéticos en edificios inteligentes

---

Alumno: David Montalvo García

Tutores: Aníbal Bregón Bregón

Carlos J. Alonso González

Construcción de algoritmos de aprendizaje automático para predicción de consumos energéticos en edificios inteligentes

David Montalvo García

"Podría parecer que hemos llegado a los límites
alcanzables por la tecnología informática,
aunque uno debe ser prudente con estas afirmaciones,
pues tienden a sonar bastante tontas en cinco años".

John Von Neumann, 1949

Agradecimientos

Los caminos ya de por sí duros lo son aún más si cabe cuando se recorren en solitario. Este proyecto ha podido salir adelante gracias a todas las personas que me han apoyado y han confiado en mí. Gracias en primer lugar a mis tutores, D. Aníbal Bregón y D. Carlos Javier Alonso González, y al profesor D. José Belarmino Pulido Junquera, por haber confiado en mí y en el proyecto y haberme dado la oportunidad de formar parte de su grupo de investigación.

En lo personal, dar las gracias a mi familia, amigos y pareja, por haberme apoyado durante esta etapa de formación, tratando siempre de sacar lo mejor de mí y apoyándome en todas y cada una de las decisiones tomadas.

Resumen

Dada la notable tendencia creciente en la demanda de energía a nivel global, se hace necesario disponer de herramientas que faciliten la toma de decisiones en aspectos relativos al ahorro de energía, la eficiencia energética o la sostenibilidad. En el ámbito de la industria 4.0, en especial en los Edificios 4.0, el ahorro y la eficiencia energética son dos conceptos básicos sobre los que se asienta su fundamentación.

El presente proyecto persigue el estudio y la construcción de diversos modelos de predicción de consumos, dada su importancia como herramientas de pronóstico de tendencias a corto y largo plazo. Para ello el proyecto se centra en las Redes Neuronales Recurrentes, en especial en la redes LSTM (*Long Short-Term Memory*), enmarcadas ambas dentro del área del aprendizaje automático y seleccionadas para este proyecto dada su gran capacidad de detección de patrones en los datos a corto y largo plazo. El estudio se complementa con modelos estadísticos clásicos de predicción de series temporales, a fin de disponer de un punto de partida sobre el que poder contrastar los resultados obtenidos con los modelos de aprendizaje automático. Tras su construcción, todos los modelos finales son aplicados sobre datos reales de consumo de energía eléctrica obtenidos de dos edificios de oficinas del ayuntamiento de Dublín.

Palabras claves: Edificios 4.0, Predicción de Consumos de Energía, Series Temporales, Modelos Estadísticos, Aprendizaje Automático, Redes LSTM.

Abstract

Given the significant growing trend in global energy demand, there is a need for tools to provide decision-making on issues of energy saving, energy efficiency or sustainability. In the field of Industry 4.0, in particular in Buildings 4.0, energy saving and efficiency are two basic concepts on which its foundation is based.

This project aims to study and build different consumption forecasting models, given their importance as tools for forecasting short- and long-term trends. To this end, the project focuses on Recurring Neural Networks, especially the LSTM (Long Short-Term Memory) networks, both framed within the area of machine learning and selected for this project due to their great ability to detect patterns in short- and long-term data. The study is complemented by classical statistical time series prediction models, in order to have a starting point on which to compare the results obtained with machine learning models. After construction, all final models are applied on actual electricity consumption data obtained from two Dublin City Council office buildings.

Keywords: Buildings 4.0, Energy Load Forecasting, Time Series, Statistical Models, Machine Learning, LSTM Networks.

Índice general

Índice de figuras

Índice de tablas

Índice de códigos Python

Acrónimos

ACF Auto-Correlation Function. 82

AIC Akaike Information Criterion. 42

ARIMA Autoregressive Integrated Moving Average. 38

BIC Bayesian Information Criterion. 43

BPTT Backpropagation Through Time. 59

L-BFGS Limited-memory Broyden–Fletcher–Goldfarb–Shanno. 42

LSTM Long Short-Term Memory. 62

MAE Mean Absolute Error. 120

MLP Multilayer Perceptron. 60

MSE Mean Squared Error. 102

PACF Partial Auto-Correlation Function. 82

RMSE Root Mean Squared Error. 119

RNN Recurrent Neural Network. 51

SARIMA Seasonal ARIMA. 44

SCORE SCRUM for Research. 5

SGD Stochastic Gradient Descent. 102, 103

TBATS Trigonometric seasonality, Box-Cox transformation, ARMA errors, Trend and Seasonal components. 48

Capítulo 1

Introducción

El sector de la energía en España supone aproximadamente un 3 % del PIB del país[1]. Sin embargo, el mayor valor que aporta la energía va más allá de su participación en la producción económica. El sector energético constituye un pilar fundamental del que dependen el resto de actividades económicas. A día de hoy es impensable, ni tan siquiera imaginable, cómo sería la sociedad de la que formamos parte con sólo un ápice de la energía que utilizamos diariamente.

El rápido crecimiento del uso de la energía, no sólo a nivel nacional, ha suscitado en todo el mundo una gran preocupación debido a las dificultades de suministro, al agotamiento de los recursos energéticos y al impacto medioambiental derivado de su explotación. En España existe una fuerte dependencia energética de países extranjeros. Según el INE (Instituto Nacional de Estadística), cerca del 75 % de la energía que utiliza nuestro país no se genera en él[2]. La escasez de recursos energéticos y el aumento incesante de la demanda han propiciado en los últimos años la aparición y el posterior desarrollo de fuentes de energía alternativas. Sin embargo sigue sin ser suficiente, y día a día y año tras año el aumento de la demanda de consumo se hace más notable.

Contextualizada por la ya inmersa Cuarta Revolución Industrial, bautizada también con el nombre de Industria 4.0, la necesidad de mejorar la eficiencia energética y garantizar la sostenibilidad en el día a día de los edificios ha dado lugar al desarrollo de un nuevo concepto de edificación dedicado a la integración en sí mismo de los sistemas tecnológicos necesarios para la consecución de estos objetivos. Estos edificios son conocidos coloquialmente como "edificios inteligentes" (*smart buildings*) o edificios 4.0, y entre sus principios básicos destaca la reducción del consumo energético. En esta línea entra en juego la necesidad de disponer de herramientas

[1] https://www.ree.es/es/datos/publicaciones/analisis-informes-demanda-electrica (último acceso 23/09/2020)

[2] https://www.ine.es/prodyser/espa_cifras/2019/ (último acceso 23/09/2020)

y modelos dedicados expresamente al estudio y el análisis de las variables que constituyen el edificio, a fin de poder construir sistemas autónomos de gestión energética.

Entre estas herramientas, los algoritmos de aprendizaje automático (*machine learning* en inglés) has demostrado su eficacia (Aviek Naug, Ahmed y Biswas, 2019) debido a su capacidad de manejar y procesar grandes volúmenes de información para generalizar comportamientos y reconocer patrones. Esto les hace idóneos para procesar la ingente cantidad de datos que pueden ser extraídos de los diferentes sensores y medidores instalados en estos edificios.

1.1. Motivación

El fuerte crecimiento y repercusión de los modelos de aprendizaje automático en la sociedad actual, junto con la necesidad anteriormente expresada de construir modelos de predicción de consumos de energía que aporten una poderosa herramienta de análisis de consumos, hacen de la idea de este proyecto un punto de partido idóneo sobre el que investigar.

La sensibilidad por el mundo que nos rodea y la necesidad de buscar medidas drásticas ante la tendencia creciente en la demanda de energía constituyen dos de los motivos principales por los que he optado por desarrollar el presente proyecto. Además no son muchos los estudios realizados sobre la construcción de modelos de predicción de energía eléctrica, sin embargo sí son trascendentes sus aplicaciones en una sociedad cada día más dependiente de la energía.

Por otro lado, los conocimientos adquiridos tanto en el Grado en Ingeniería Informática de Servicios y Aplicaciones como en el Grado en Matemáticas, me permiten la profundización en el lado más abstracto de los modelos. Esto me ha permitido optar por un proyecto ambicioso que a su vez trate de abstraer y profundizar más en los modelos a construir, tratando de aportar valor en los resultados con el fin de que estos modelos puedan servir para mejorar la sostenibilidad y reducir el consumo energético, ayudando a predecir con precisión el consumo de energía eléctrica y el rendimiento de las fuentes renovables.

1.2. Objetivos del trabajo

Tal y como ya se ha mencionado, el presente proyecto tratará sobre la construcción de modelos de predicción de consumos de energía. En esta línea de trabajo se identifican los siguientes objetivos:

- **OBJ-1** Estudio y comparación de diferentes modelos dedicados a la predicción de consumos. Ligado a este objetivo se llevarán a cabo tareas de investigación con el fin de encontrar

aquellos modelos que mejor se adapten al problema de consumos planteado. Entre los modelos se deberán incluir modelos estadísticos clásicos de predicción de series temporales y modelos de aprendizaje automático, con el fin de contrastar ambos enfoques y poder analizar posteriormente las diferencias, en cuanto a aproximación se refiere, obtenidas con ellos.

- **OBJ-2** Búsqueda y análisis de conjuntos de datos de consumos energéticos de edificios inteligentes sobre los que aplicar los modelos de predicción, a fin de poder analizar el grado de aproximación de los mismos sobre datos reales.
- **OBJ-3** Construcción de los modelos de predicción utilizando el lenguaje *Python*. Para ello se deberán buscar las librerías que mejor aproximen los modelos buscados, y realizar posteriormente las implementaciones necesarias para llevar a cabo la construcción de los modelos y la posterior realización de las predicciones de consumos.
- **OBJ-4** Realización de predicciones, búsqueda de los modelos óptimos y evaluación de resultados. Tras disponer de todos los modelos, se deberán escoger aquellos modelos que mejor aproximen los datos a estudio y realizar con ellos predicciones tanto a corto como a largo plazo, analizando en cada caso qué modelo aporta las mejores predicciones. Para este análisis deberán estudiarse las métricas que mejor se adaptan al problema a estudio, y analizar con ellas el grado de aproximación de las predicciones.

1.3. Organización de la memoria

El presente documento se encuentra dividido en nueve capítulos sobre los cuales se tratan los diversos objetivos planteados en el proyecto, junto con algunas consideraciones ligadas al proceso de desarrollo y gestión del proyecto.

El actual capítulo 1 trata de introducir al lector en los conceptos que fundamentan la realización de este proyecto. Los siguientes dos capítulos constituyen la parte de desarrollo y gestión del proyecto. El capítulo 2 incluye la metodología utilizada durante el desarrollo del proyecto y las tecnologías y herramientas utilizadas en él, mientras que el capítulo 3 ahonda en el proceso de planificación. Este último incluye, además de la planificación temporal del proyecto, una estimación de esfuerzos y costes junto con un balance final del proyecto.

El capítulo 4 incluye una contextualización del proyecto desde el punto de vista de la Industria 4.0, los edificios inteligentes y el ahorro energético. En él se incluye también un análisis de los estudios previos relativos a modelos de predicción de consumos.

Uno de los capítulos más importantes de esta memoria es el capítulo 5. En él se profundiza en los modelos de predicción desde un enfoque constructivo. Se detallan los parámetros que

definen el modelo, así como los algoritmos y métodos de entrenamiento y ajuste de parámetros. La primera mitad del capítulo muestra los modelos clásicos, si bien la segunda parte profundiza en las redes neuronales recurrentes como modelos de aprendizaje automático dedicados a series temporales. El capítulo 6 muestra el modelo de datos seleccionado para este proyecto. En el capítulo se incluye además el preprocesado llevado a cabo sobre estos registros, así como un análisis del modelo final.

El otro capítulo más relevante es el capítulo 7, dado que en él se lleva a cabo la construcción en *Python* de todos los modelos de predicción estudiados en el capítulo 5. Además se lleva a cabo la construcción del modelo óptimo para cada uno de los modelo y se realizan las predicciones con distintos horizontes temporales. El capítulo siguiente, el capítulo 8, analiza las predicciones realizadas y compara los resultados haciendo uso de diversas métricas de evaluación.

Por último, el capítulo 9 analiza las conclusiones del proyecto y estudia el grado de consecución de los objetivos. Se termina el capítulo con un análisis de posibles trabajos futuros.

Capítulo 2

Metodología de trabajo

A lo largo de este capítulo se detallan cuestiones relacionadas con la metodología utilizada para el desarrollo del proyecto, así como las herramientas y tecnologías empleadas, terminando el capítulo con las siglas y abreviaturas de uso extendido en la memoria.

2.1. Metodología SCORE

El presente trabajo se enmarca dentro de un proyecto de investigación del Grupo de Sistemas Inteligentes (GSI) de la Universidad de Valladolid. Dado que el proyecto planteado presenta ciertas incógnitas de cara al resultado futuro y a las posibles decisiones que deberán tomarse a medida que este avance, y dado que se encuentra englobado dentro del marco universitario, el trabajo se abordará bajo una metodología ágil orientada a proyectos de investigación en el entorno académico. Brota así la idea de consolidar como metodología de desarrollo del proyecto la metodología **SCORE** (*SCRUM for Research*), derivada de la metodología SCRUM.

La metodología SCORE fue desarrollada por Michael Hicks y Jeffrey S. Foster (2010a), profesores del Departamento de Ciencias de la Computación de la Universidad de Maryland. SCORE surge de la necesidad de agilizar los procesos de planificación y seguimiento de los proyectos de investigación universitarios con el fin de mantener un ritmo de trabajo constante y poder, desde el punto de vista del profesorado, llevar con éxito el seguimiento de múltiples proyectos con diversos equipos constituidos por diferentes alumnos. Si bien la metodología surge bajo el contexto de los proyectos de doctorado, es extensible a todo proyecto de investigación en el marco universitario.

El principal motivo que promueve la creación de esta nueva metodología de investigación se fundamenta en el notable crecimiento del número de alumnos tutorados por cada profesor. En su origen, los profesores Hicks y Foster contaban con un máximo de dos o tres alumnos por curso académico, llegando a ascender esta cifra hasta los siete u ocho alumnos, lo que dificultaba

su seguimiento y la atención individualizada. La nueva metodología tratará de mantener los objetivos principales en el desarrollo de los proyectos de investigación:

1. mantener la calidad de las investigaciones, y

2. motivar a los estudiantes para incrementar su autonomía en la resolución de problemas y fomentar el trabajo en ambientes de colaboración e investigación universitaria.

Para comprender correctamente la fundamentación de esta nueva metodología es necesario antes que nada conocer de primera mano la metodología **SCRUM** sobre la que asienta sus bases (véase Schwaber, 1997, para ampliar la información). La Figura 2.1 ilustra el proceso básico de desarrollo de software siguiendo el modelo de trabajo SCRUM. Bajo esta metodología, las tareas se dividen entre los distintos equipos de trabajo, cada equipo formado por no más de siete miembros, uno de los cuales desempeña el papel de ***scrum master***, quedando este encargado de la organización del equipo y de asegurar la consecución de los objetivos y el cumplimiento de los procedimientos marcados por la metodología. El equipo implementa las diversas características del producto en una serie de ***sprints*** de una duración de 1 a 4 semanas. Cada *sprint* termina con una nueva versión funcional del producto y con dos reuniones de control, el ***sprint review*** (para revisar con las partes interesadas en el desarrollo (*stakeholders*) la nueva versión para acercar posturas sobre las funcionalidades que no están tal y como se esperaba y tratar los motivos) y el ***sprint retrospective*** (para evaluar el trabajo realizado en el *sprint*).

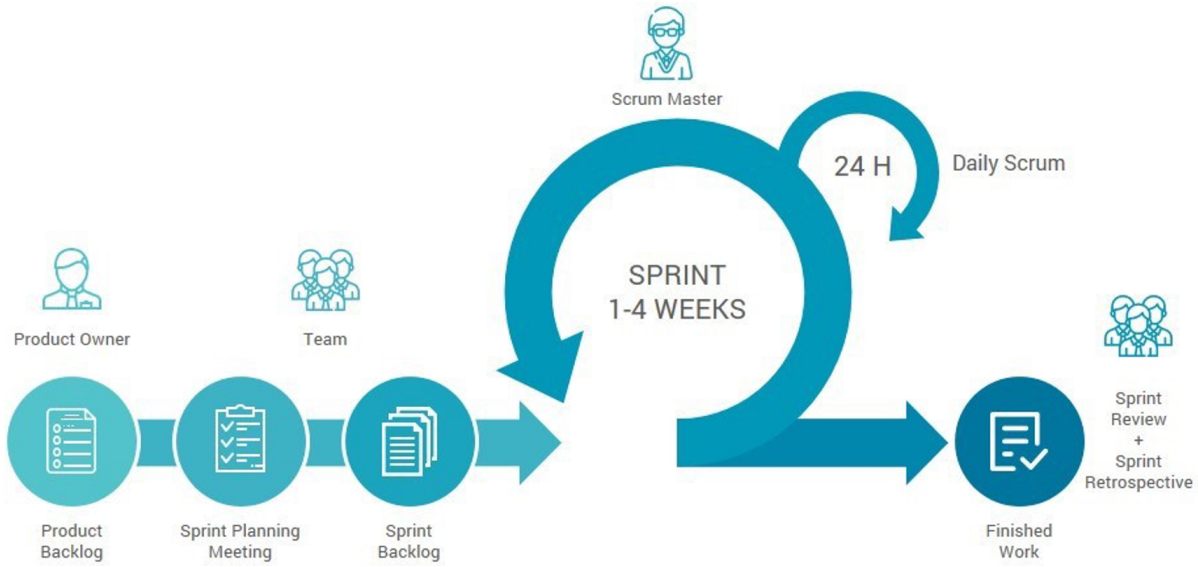


FIGURA 2.1: FLUJO DE TRABAJO CON LA METODOLOGÍA SCRUM

Antes de comenzar cada *sprint*, el equipo de desarrollo junto con el ***product owner*** (nexo de unión entre los *stakeholders* y los equipos de desarrollo) celebran una reunión de planificación, conocida como ***sprint planning meeting***, para decidir qué tareas deben llevarse a cabo durante el siguiente *sprint*. Este conjunto de tareas se denomina ***sprint backlog***, y se extrae de entre el total de características, funcionalidades, requerimientos y demás necesidades o modificaciones, recogidas todas ellas y ordenadas por prioridad en el ***product backlog***.

Cada día, los equipos celebran una reunión informal, llamada ***daily scrum***, y entre las normas que rigen la reunión se encuentran (1) la hora de inicio deberá ser siempre la misma, (2) deberá realizarse de pie para evitar que se alargue más de lo necesario, y (3) la duración no deberá exceder los 15 minutos. En esta reunión, cada miembro del equipo deberá responder a tres cuestiones:

1) ¿Qué has realizado desde el último *scrum*?

2) ¿Tienes algún obstáculo que te impide continuar?

3) ¿Qué vas a realizar antes del próximo *scrum*?

Para cualquier tema que no pueda ser resuelto de inmediato, o si algún miembro del equipo parece tener problemas para avanzar, el *scrum master* deberá organizar una reunión separada o bien tomar cualquier otra medida alternativa que permita resolver el problema. Estas reuniones alternativas poseen un marcado carácter técnico, y se fundamentan en la resolución de problemas específicos.

Hicks y Fooster vieron en esta metodología un camino para solucionar muchos de los problemas que se encontraban a diario en el tutelado concurrente de diversos proyectos de investigación en el marco universitario. Los dos elementos clave de la metodología SCORE, heredados ambos de SCRUM, son las ***status meeting***, análogas a las reuniones *scrum*, si bien en lugar de ser reuniones diarias se realizan dos o tres días por semana; y las ***on-demand meetings*** entre el alumno y los tutores previa solicitud por parte del alumno, orientadas a solucionar problemas concretos surgidos durante la investigación y detectados en la mayoría de casos durante las reuniones periódicas.

2.1.1. Status meetings

El elemento principal y que aporta un mayor beneficio a la metodología SCORE son las *status meetings* o reuniones de estado. Estas reuniones deben tener lugar entre dos y tres días fijos por semana (por ejemplo lunes, miércoles y jueves), los cuales deberán a su vez mantenerse fijos, salvo causas de fuerza mayor, a lo largo de todo el curso académico.

Durante la reunión, todos los estudiantes tutelados por el profesor, pertenezcan o no al mismo grupo de investigación, deben describir las tareas que han realizado desde la última reunión, los resultados logrados y los obstáculos encontrados, así como los planes que tienen previsto llevar a cabo antes de la próxima reunión. Los profesores deben describir también el trabajo realizado para fomentar las inquietudes y las ganas de trabajar de los alumnos, ya que en ocasiones el trabajo del profesor puede levantar nuevas curiosidades y aportar ideas para incluir en los proyectos actuales, o incluso de cara a proyectos futuros.

Los progresos en la investigación pueden venir dados en multitud de formas, por este motivo los estudiantes pueden tratar temas diversos como la implementación de código, la realización de experimentos, la lectura de *papers*, la elaboración de demostraciones matemáticas, la redacción de resultados, o incluso la preparación de la exposición final. No es de notable importancia que los estudiantes se centren en objetivos a corto plazo, sino más bien que cada día demuestren algún tipo de avance en su investigación. Con estas reuniones se incentiva la exposición en grupo de los resultados, y no de forma individual del alumno al profesor, lo cual proporciona riqueza y variedad de opiniones entre investigaciones distintas. Por parte del profesorado se debe mantener una actitud de comprensión, dado que en ocasiones puede ocurrir que el alumno no haya podido avanzar, bien sea por el trabajo ligado a otras asignaturas de la universidad, bien por motivos personales.

Al igual que las reuniones *daily scrum*, las reuniones de estado deben durar aproximadamente 15 minutos. En la explicación se debe evitar ahondar en detalles técnicos, dejando estos para las reuniones *on-demand*. Para mantener además mejor la concentración, estas reuniones se deben realizar de pie, lo que a su vez ayuda a prevenir que se alarguen.

2.1.2. On-demand meetings

Las *status meeting*, por cómo han sido concebidas, no deben entrar en mucha profundidad técnica. Para la discusión de cuestiones sobre la investigación, resultados, métodos, retos tecnológicos, etc. se deben llevar a cabo las reuniones bajo demanda, también llamadas *on-demand meetings* por su término anglosajón, las cuales tienen lugar previa petición del alumno. Como es evidente tras lo expuesto, estas reuniones no se programan de forma regular ni se encuentran previamente establecidas, sino que acontecen únicamente cuando existe su necesidad. Este tipo de reuniones son análogas a las reuniones de la metodología SCRUM que surgen a partir de la detección en las reuniones diarias de la necesidad de abordar con más detalle alguna característica técnica no tratada en dicha reunión para poder proseguir con el desarrollo. En el mejor de los casos han de ser los propios estudiantes quienes soliciten la reunión, no obstante los tutores pueden tomar la iniciativa y concretar una reunión si detectan que puede ser bueno para el avance del alumno en su proyecto.

Mientras en la metodología SCRUM la planificación se lleva a cabo durante las *spring planning meetings* al inicio de cada sprint, en SCORE la planificación de las actividades de investigación se lleva a cabo durante las reuniones bajo demanda. En ellas, por ejemplo, se pueden discutir los siguientes pasos a dar una vez presentado un documento que los respalde, planificar actividades antes de una fecha límite, o bien sentar las bases de una línea de investigación tras el descubrimiento de una idea prometedora. Esta característica permite agilizar la toma de decisiones, y avanzar en el proyecto según los resultados previos obtenidos, evitando "dar palos de ciego" en la investigación.

La mayor ventaja que aportan las reuniones *on-demand* es la capacidad de adaptación a las necesidades del alumno, permitiendo regular su uso en función del avance del proyecto y de la capacidad del estudiante de abordar los problemas técnicos encontrados durante su desarrollo. Estas reuniones poseen un propósito claro y su desarrollo se centra en él, lo que permite ahorrar tiempo. Las reuniones bajo demanda sólo se desarrollan cuando el alumno previamente ha trabajado en profundidad el tema, evitando al tutor perder tiempo explicando facetas que el alumno todavía no ha trabajado y experimentado en profundidad.

2.1.3. Otros elementos de SCORE

Las reuniones de estado y las reuniones bajo demanda son los elementos principales de la metodología SCORE, ambos heredados de la metodología SCRUM. No obstante, existen otros elementos que permiten agilizar el seguimiento y la puesta en marcha de los proyectos de investigación. En primer lugar, en la medida que sea posible y con el fin de favorecer la comunicación entre ambas partes, el tutor debe facilitar al alumno el desarrollo del trabajo en el mismo lugar donde él mismo trabaje o, si se da el caso, en el espacio de trabajo asignado al equipo de investigación del que forme parte el tutor. De esta forma la mayor proximidad entre alumno y profesor tiende a fomentar mayores niveles de interacción.

Por otro lado, SCORE aconseja encuentros sociales distendidos de forma regular, como comidas o cafés en grupo, con el fin de buscar una mayor complicidad en la relación tutor-alumno. Por último, SCORE recomienda al tutor crear grupos de lectura opcionales o algún tipo de actividad alternativa entre alumnos, en los que tratar temas relacionados con el área en el que se centran sus investigaciones y poder sacar temas de debate con el fin de favorecer la integración de los estudiantes y que a su vez sean ellos mismos quienes puedan extraer nuevas ideas para sus proyectos.

2.1.4. Adaptación e implantación

Dado que el contexto bajo el que se crea la metodología SCORE coincide con el ámbito del presente proyecto, salvando las diferencias derivadas del gran abanico de proyectos de investigación posibles, su implantación en él presenta importantes ventajas de cara incluso a la planificación y la organización del mismo. Sin embargo, la dificultad que añade el hecho de trabajar entre dos provincias por cómo se encuentra estructurado el Programa de Estudios Conjunto en el que se enmarca, como son Segovia y Valladolid, dificulta la puesta en práctica de las reuniones de estado con el conjunto total de estudiantes que enmarcan su Trabajo de Fin de Grado en el contexto de la investigación universitaria. Por ello, el planteamiento de las reuniones de estado debe reducirse a reuniones breves únicamente entre tutor y alumno, perdiendo así la oportunidad de cruzar opiniones con otros compañeros de investigación. Por este motivo, las reuniones de estado en ocasiones serán sustituidas por breves correos electrónicos de control que permitan dar una idea general de los avances realizados, evitando así que el proyecto pueda quedar paralizado o postergado. Estos correos de control se denominarán en adelante ***status emails*** o correos electrónicos de estado, y se crean como alternativa a las *status meetings*.

Derivado del hecho de transformar las reuniones de estado en los emails de control, la fechas de las mismas no se mantendrá necesariamente constante, y dependerá de la situación del tutor y del alumno el mantener las reuniones de estado en fechas prefijadas o controlar el avance mediante correos electrónicos.

En cuanto a las reuniones bajo demanda, se mantendrá la estructura y planteamiento propuesta en la metodología SCORE, destacando como mayor ventaja la flexibilidad temporal que aportan y el uso de las mismas para organizar el trabajo futuro y establecer entregas. Sin embargo, tras la entrada en vigor del Real Decreto 463/2020 del 14 de marzo por el que se declara el estado de alarma para la gestión de la situación de crisis sanitaria ocasionada por la COVID-19 y que obliga al cierre de los centros universitarios, las reuniones bajo demanda pasan a celebrarse por videoconferencia, manteniendo no obstante su misma filosofía y finalidad. De igual modo todas las reuniones de estado pasan a ser correos electrónicos de estado, eliminando todo contacto directo entre los miembros del equipo de investigación.

De cara a agilizar la organización del trabajo y la planificación de tareas al inicio del proyecto, se establecerá un mecanismo adicional de gestión del trabajo basado en el concepto de **bloque de trabajo**. Los bloques de trabajo, análogos a los *sprints* de la metodología SCRUM, contarán con una duración aproximada de 20 días lectivos (4 semanas), y estarán formados por aquellas tareas que deban ser desarrolladas en dicho periodo. Este nuevo concepto se incorpora a la metodología de trabajo con el fin de llevar a cabo una planificación efectiva con anterioridad a la fecha de comienzo del proyecto. Para su gestión y control será necesario un nuevo tipo de reunión que permita controlar los progresos y el cumplimiento, en la medida que sea posible,

de los plazos de realización de tareas. Surgen así las ***management meetings*** o reuniones de gestión, las cuales engloban en una única reunión las funciones del *sprint planning*, *sprint review* y del *sprint retrospective* de la metodología SCRUM. El motor de estas reuniones es la gestión del progreso, permitiendo la incorporación de nuevas tareas y modificaciones en el rumbo del proyecto, manteniendo en todo momento un ritmo de trabajo constante que evite sobrecargas.

Aunque conceptualmente la metodología SCORE resulta simple, los beneficios de su aplicación son significativos. Por un lado, destaca la facilidad de mantenerse al día en los progresos del proyecto gracias a las reuniones y correos de estado. Además, cuando existe una dificultad para avanzar, la sustitución de las reuniones periódicas por reuniones bajo demanda permite agilizar los trámites para abordar el problema. Esta metodología permite una gestión más eficaz del tiempo desde el punto de vista de los tutores, dado que las reuniones bajo demanda evitan reuniones periódicas sin apenas contenido para tratar. Las reuniones *on-demand* tienen un propósito claro y por lo tanto son mucho más productivas. La gestión a mayores de los bloques de trabajo junto con las reuniones de gestión permite llevar a cabo una planificación previa al comienzo del proyecto, pero manteniendo siempre la capacidad de cambio y adaptación ante nuevas líneas de trabajo.

2.2. Herramientas y tecnologías utilizadas

En esta sección se detallan tanto las herramientas utilizadas para la consecución de los objetivos generales del proyecto (tales como IDEs y editores de texto), como las tecnologías utilizadas para su desarrollo (lenguajes de programación, *frameworks* y librerías).

Las herramientas que se han utilizado son:

- **JupyterNotebook**: aplicación web de código abierto que permite crear y compartir documentos que integran código, resultados de ejecución y texto (en formato Markdown). Entre sus usos más extendidos destaca el procesado de datos, la simulación numérica, la creación de modelos estadísticos o de aprendizaje automático y la visualización de datos. Soporta más de 40 lenguajes de programación, incluidos Python, R, Julia y Scala. JupyterNotebook ha sido utilizado durante el proyecto con Python para el procesado de datos y la construcción de los modelos de predicción.

- **FileZilla**: cliente FTP multiplataforma y de código abierto para gestionar y acceder a archivos almacenados en un servidor FTP. El cliente FileZilla soporta los protocolos FTP, FTP sobre SSL/TLS (FTPS) y SFTP (SSH File Transfer Protocol). Este último protocolo es el utilizado en el presente proyecto para el intercambio de archivos entre la máquina local y el servidor.

- **Texmaker**: editor LaTeX multiplataforma de código abierto con visor integrado de documentos PDF y soporte Unicode. Este editor ha sido utilizado para la elaboración del presente documento.
- **Docker**: plataforma de código abierto dedicada a la automatización del despliegue de aplicaciones mediante el uso de contenedores de software, proporcionando una capa adicional de abstracción y automatización de aplicaciones en múltiples sistemas operativos.

Por otro lado, las tecnologías que se han empleado para el desarrollo del proyecto son:

- **Python**: Lenguaje de programación interpretado de tipado dinámico, de alto nivel y de propósito general, cuya filosofía hace hincapié en una sintaxis que favorezca el código legible. Se trata de un lenguaje de código abierto, multiplataforma y multiparadigma, soportando orientación a objetos, programación imperativa y programación funcional. Fue creado a final de los noventa por Guido van Rossum en los Países Bajos. Las numerosas librerías creadas en Python por la gran comunidad que respalda el lenguaje hacen de él un gran punto de entrada para el análisis y visualización de datos, así como la construcción de modelos de aprendizaje y modelos estadísticos de predicción.
- **Pandas**: librería de código abierto de Python que provee estructuras de datos rápidas y flexibles diseñadas para trabajar con datos estructurados y series temporales de forma sencilla e intuitiva. Entre las principales características que aportan mayor valor al actual proyecto destacan los tipos de datos *DataFrame* y *Series* con indexación integrada, la incorporación de herramientas de escritura y lectura de datos en memoria para diversos formatos de archivos, así como el manejo de series temporales, mediante la generación de rangos de fechas y conversión de frecuencias y la manipulación de ventanas temporales.
- **Numpy**: librería de código abierto de Python que añade soporte para matrices y *arrays* multidimensionales (tensores), incorporando además una gran colección de funciones matemáticas de alto nivel para operar sobre dichas estructuras. La principal ventaja de la implantación de la librería sobre el proyecto es el uso de la estructura de datos multidimensional (*ndarray*) para la construcción y manipulación de los tensores empleados en la definición, entrenamiento y la posterior predicción con redes neuronales a bajo nivel.
- **Matplotlib**: librería de visualización de código abierto para el lenguaje Python. Permite la creación de visualizaciones en 2D estáticas, dinámicas e interactivas, tales como gráficos, histogramas, gráficos de barras, gráficos de error, diagramas de dispersión, etc. Su uso en el proyecto se centra en la representación gráfica de series temporales.
- **Joblib y Pickle**: librerías de código abierto de Python que implementan protocolos binarios de serialización de objetos Python. Joblib provee a su vez herramientas de serialización

eficientes orientadas a grandes estructuras de datos, por lo que su uso se restringirá a la serialización de los modelos de predicción ya entrenados.

- **Statsmodels**: librería de código abierto de Python que provee herramientas de cálculo estadístico, incluidas herramientas de estimación e inferencia sobre modelos estadísticos. Entre las características destacadas para su aplicación en el proyecto destacan los modelos de análisis de series temporales, incluyendo los modelos ARIMA y SARIMA para el análisis de series univariantes, así como el uso de métricas de evaluación de errores de predicción.

- **Pmdarima**: librería estadística de código abierto diseñada para cubrir las necesidades de análisis de series temporales de Python. En el presente proyecto se utiliza como complemento a la librería estadística *statsmodels*, destacando las herramientas de búsqueda de modelos ARIMA y SARIMA óptimos para el problema concreto que se aborda, así como el uso de la transformación de variables exógenas tales como las transformadas de Fourier y de Box-Cox, y las herramientas de test estadísticos y de descomposiciones estacionales de series temporales.

- **Tbats**: librería Python de código abierto que provee implementaciones de los modelos de predicción BATS y TBATS de series temporales.

- **TensorFlow**: biblioteca de código abierto orientada al aprendizaje automático, cuyas principales funciones son la construcción y entrenamiento de redes neuronales con el fin de detectar y descifrar patrones y correlaciones, de forma análoga al aprendizaje y razonamiento humano. Tensorflow provee además una API estable sobre Python.

- **Keras**: librería de alto nivel desarrollada en Python para el aprendizaje automático, capaz de correr sobre TensorFlow, Microsoft Cognitive Toolkit o Theano. Diseñada para trabajar de forma rápida con redes neuronales profundas, centrada en la facilidad de uso y la modularidad. Además de las redes neuronales estándar, Keras soporta redes neuronales convolucionales y recurrentes. Esta última característica fundamenta su aplicación en el proyecto, dado que permitirá la construcción de redes neuronales recurrentes con distintas arquitecturas, enmarcando su uso dentro de las necesidades del problema a estudio.

A rasgos generales, el motivo por el que se ha seleccionado Python como lenguaje para la implementación de los modelos de predicción frente al lenguaje estadístico R, es la posibilidad de poder integrar con aplicaciones web los modelos obtenidos, de cara a su uso posterior aplicado en la gestión de consumos por parte de personas ajenas al proyecto. Además, la rápida curva de aprendizaje de Python y su versatilidad frente a R motiva a mayores su uso, dado que el objetivo del proyecto radica en resolver el problema a estudio, y el aprendizaje de las herramientas para resolverlo no debe convertirse en el núcleo del problema en sí.

Capítulo 3

Planificación

El actual proyecto se enmarca dentro de una Beca de Colaboración del Ministerio de Educación y Formación Profesional para la realización de un proyecto de investigación de un mínimo de 420 horas. En este capítulo se incluye una estimación de esfuerzos dirigida a estructurar la planificación temporal previa al inicio del proyecto. Para ello se seguirán las pautas establecidas en la sección 2.1.4 del capítulo previo, utilizando la estructura de bloques de trabajo para la agrupación de las tareas. Se presentará además como complemento a la planificación un presupuesto con los posibles gastos derivados del proyecto. El capítulo se concluye con la valoración final, tras haber finalizado el proyecto, de la distribución final de tareas y el coste final del proyecto, analizando las posibles desviaciones con respecto a la estimación inicial.

3.1. Estimación del esfuerzo

Previo a la planificación temporal, es necesario llevar a cabo un análisis de las distintas tareas que componen el proyecto junto con una estimación del tiempo que será necesario destinar a cada una de ellas. Para ello, siguiendo la adaptación que la metodología SCORE hace de SCRUM, se hará uso de tareas descriptivas análogas a las historias de usuario de SCRUM, complementadas con el uso de los **puntos de historia** y el ***planning poker*** para la estimación.

El *planning poker* (Grenning, 2002) es una técnica de estimación según la cual el equipo de trabajo se reúne y llega a un acuerdo sobre el esfuerzo aproximado necesario para desarrollar cada tarea. Para ello, cada miembro del equipo dispone de un set de cartas compuesto por once cartas con los valores de la pseudo-secuencia de Fibonacci 0, 1/2, 1, 2, 3, 5, 8, 13, 20, 40 y 100, y dos cartas extra con los símbolos de infinito (∞) e interrogación (?). Estas dos últimas se utilizan para mostrar imposibilidad y desconocimiento, respectivamente (véase la Figura 3.1). Existen diversas alternativas acerca de las cartas a utilizar, aunque todas son similares a la presentada.

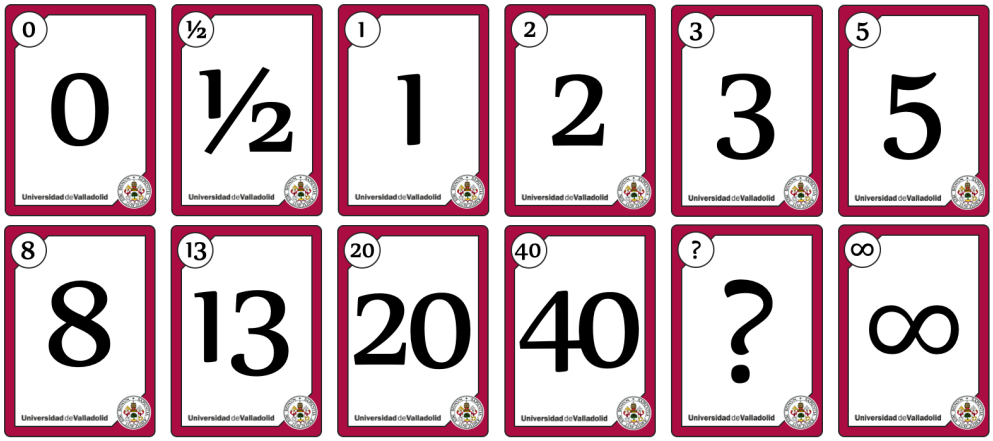


FIGURA 3.1: CARTAS DE ESTIMACIÓN DE PLANNING POKER

Los valores de las cartas pueden indicar días reales, días ideales o puntos de historia. Los Puntos de Historia, denotados de forma abreviada por *SP* (como acrónimo de su término inglés *Story Points*) indican el esfuerzo relativo para la realización de un ítem frente a otro, y se utilizan como medida de estimación. Aportan una idea del tamaño y el esfuerzo que se necesita para que las tareas sean desarrolladas.

Una a una se presentan todas las tareas que componen el proyecto, y para cada una de ellas todos los miembros del equipo han de enseñar una carta en función del grado de esfuerzo que cada uno considera que le corresponde a la tarea concreta. Si tras conocer todas las estimaciones no se llega a un consenso entre el equipo, se deberá debatir brevemente los motivos y repetir la estimación con las cartas, hasta un máximo de tres veces. Si finalmente no hay acuerdo, se tomará bien el mayor valor (lo que suele ser la opción preferible), bien la media de las estimaciones.

Para la gestión del actual proyecto, en primer lugar se realizará la definición de las tareas y subtareas que componen inicialmente el proyecto. Para la puesta en práctica de la técnica de estimación de *planning poker*, tutor y alumno deberán asignar un valor de puntos de historia a cada una de las subtareas del proyecto, con el fin de realizar en el siguiente paso la planificación temporal. A continuación se listan las tareas y subtareas que componen el proyecto, junto con las asignación de los puntos de historia correspondientes:

- **T-1 Estudio del dominio del proyecto**.
  - **T-1.1** Estudio del uso eficiente de la energía y de modelos de ahorro energético (*1 SP*).
  - **T-1.2** Estudio sobre industria 4.0 (*1 SP*).
  - **T-1.3** Estudio de las *smart cities* y la edificación 4.0 (*2 SP*).

  - **T-1.4** Estudio de los sistemas HVAC (*1/2 SP*).
  - **T-1.5** Estudio de casos previos (*2 SP*).

- **T-2 Estudio de los modelos de predicción**.
  - **T-2.1** Estudio de la caracterización de series temporales (*2 SP*).
  - **T-2.2** Estudio de modelos ARIMA (*3 SP*).
  - **T-2.3** Estudio de modelos ARIMA con estacionalidad (modelos SARIMA) (*2 SP*).
  - **T-2.4** Estudio de modelo TBATS (*1 SP*).
  - **T-2.5** Estudio de Redes Neuronales Recurrentes - RNN (*3 SP*).
  - **T-2.6** Estudio de redes LSTM (*5 SP*).

- **T-3 Carga y preprocesado de datos**.
  - **T-3.1** Carga y estudio preliminar de los datos (*1 SP*).
  - **T-3.2** Preprocesamiento de datos (*2 SP*).
  - **T-3.3** Análisis y exportación del conjunto de datos final (*1 SP*).

- **T-4 Implementación y selección de modelos**.
  - **T-4.1** Implementación y construcción de los modelos ARIMA (*3 SP*).
  - **T-4.2** Selección y configuración del modelo ARIMA óptimo (*2 SP*).
  - **T-4.3** Implementación y construcción de los modelos SARIMA (*1 SP*).
  - **T-4.4** Selección del modelo SARIMA óptimo y realización de predicciones (*1 SP*).
  - **T-4.5** Implementación y construcción de los modelos TBATS (*1/2 SP*).
  - **T-4.6** Selección y configuración del modelo TBATS óptimo (*1 SP*).
  - **T-4.7** Implementación y construcción de las LSTMs (*8 SP*).
  - **T-4.8** Selección y configuración de la red LSTM óptima (*5 SP*).

- **T-5 Análisis de predicciones**.
  - **T-5.1** Estudio del modelo de ventanas móviles para predicción (*1 SP*).
  - **T-5.2** Puesta en práctica de los modelos óptimos y realización de predicciones (*2 SP*).

- **T-6 Evaluación y análisis de resultados**.
  - **T-6.1** Estudio y elección de métricas (*1 SP*).
  - **T-6.2** Aplicación de métricas en las predicciones y análisis de resultados (*2 SP*).
- **T-7 Documentación del proyecto**.
  - **T-7.1** Contextualización del proyecto (*1 SP*).
  - **T-7.2** Documentación de la metodología (*2 SP*).
  - **T-7.3** Documentación de la estimación, planificación y presupuestación (*1 SP*).
  - **T-7.4** Documentación del estado del arte (*2 SP*).
  - **T-7.5** Documentación de los modelos de predicción (*3 SP*).
  - **T-7.6** Documentación del modelo de datos (*1 SP*).
  - **T-7.7** Documentación del proceso de construcción de modelos (*3 SP*).
  - **T-7.8** Documentación del análisis de resultados (*1 SP*).
  - **T-7.9** Documentación de las conclusiones y trabajos futuros (*1 SP*).
  - **T-7.10** Revisión final del proyecto (*2 SP*).

Tras dividir el proyecto en tareas y estimar el esfuerzo de cada una de ellas mediante puntos de función, se ha obtenido un total de 71 puntos de historia.

3.2. Planificación temporal

Tal y como se ha indicado en la sección de adaptación de la metodología SCORE al proyecto actual (véase 2.1.4), las tareas se agruparán en bloques de trabajo, análogos a los *sprints* de SCRUM, de una duración aproximada de 20 días lectivos. La fecha de comienzo del proyecto es el 28 de octubre de 2019, y la fecha prevista de finalización el 20 de julio de 2020. Dada estas perspectivas de extensión del proyecto, serán necesarios un total de 8 bloques de trabajo.

Las tareas se deben distribuir de forma que el total de puntos de historia a trabajar en cada bloque de trabajo sea similar en todos los bloques que componen el proyecto. En este caso los puntos de historia por bloque deberán estar entre 8 y 10, tratando así de equilibrar el trabajo y evitar sobrecargas en la recta final. En la Tabla 3.1 se puede observar el reparto de tareas en bloques, utilizando para ello los identificadores previamente asignados. En ella se muestra a

mayores las fechas de inicio y finalización de los bloques. El día fijado como fecha de finalización del bloque tendrá lugar una reunión de gestión (*management meeting*) para revisar el trabajo realizado y llevar a cabo, si es necesario, modificaciones en la planificación de cara a adaptar las tareas a las nuevas necesidades o vías de trabajo que puedan haber surgido durante el transcurso del bloque.

Bloque	**Fecha de inicio**	**Fecha de finalización**	**Tareas**	**Puntos de historia**	**Progreso**
#1	28/10/2019	02/12/2019	T-1.1 / T-1.2 / T-2.1 / T-2.2 / T-7.1	8	11,3 %
#2	03/12/2019	20/01/2020	T-1.3 / T-1.4 / T-2.3 / T-3.1 / T-7.2 / T-7.3	8,5	23,2 %
#3	21/01/2020	18/02/2020	T-1.5 / T-2.4 / T-3.2 / T-3.3 / T-7.4 / T-7.6	9	35,9 %
#4	19/02/2020	19/03/2020	T-4.1 / T-4.2 / T-4.3 / T-4.4 / T-4.5 / T-5.1	8,5	47,9 %
#5	20/03/2020	21/04/2020	T-2.5 / T-2.6 / T-6.1	9	60,6 %
#6	22/04/2020	25/05/2020	T-4.6 / T-4.7	9	73,2 %
#7	26/05/2020	22/06/2020	T-4.8 / T-5.2 / T-7.5	10	87,3 %
#8	23/06/2020	20/07/2020	T-6.2 / T-7.7 / T-7.8 / T-7.9 / T-7.10	9	100 %

Tabla 3.1: Distribución de tareas del proyecto en bloques de trabajo

Para la estimación de horas de trabajo, necesario para la elaboración del presupuesto, se estima que se dedicarán 3 horas al día durante todos los días lectivos que abarque el proyecto, excluyendo por tanto para la estimación los fines de semana y los días festivos. De este modo, dado que el conjunto de bloques de trabajo engloba un total de *162 días*, se obtiene un total de *486 horas* de trabajo.

El presente proyecto abarca tanto el Trabajo de Fin de Grado relativo al Grado en Ingeniería Informática de Servicios y Aplicaciones, como la Beca de Colaboración del Ministerio del Educación y Formación Profesional. Por un lado para el Proyecto de Fin de Grado se requiere

cubrir la carga lectiva de de 12 ETCS[1], lo que equivale a un total de 300 horas de trabajo. Por su parte, para cubrir la beca de investigación se necesitan un total de 420 horas. Se observa así que la estimación de 486 horas de trabajo supera de forma adecuada la dedicación necesaria para cubrir ambos programas.

En la Figura 3.2 se puede observar el diagrama de Gantt realizado para planificar las tareas del proyecto. En él se puede observar además las fechas de inicio y finalización estimadas para cada una de las tareas que componen el proyecto, así como la agrupación de tareas en bloques de trabajo. Dado que el proyecto se desarrolla con un único miembro, no es posible paralelizar las tareas, y la planificación sigue una estructura lineal y de dependencia entre todas las tareas.

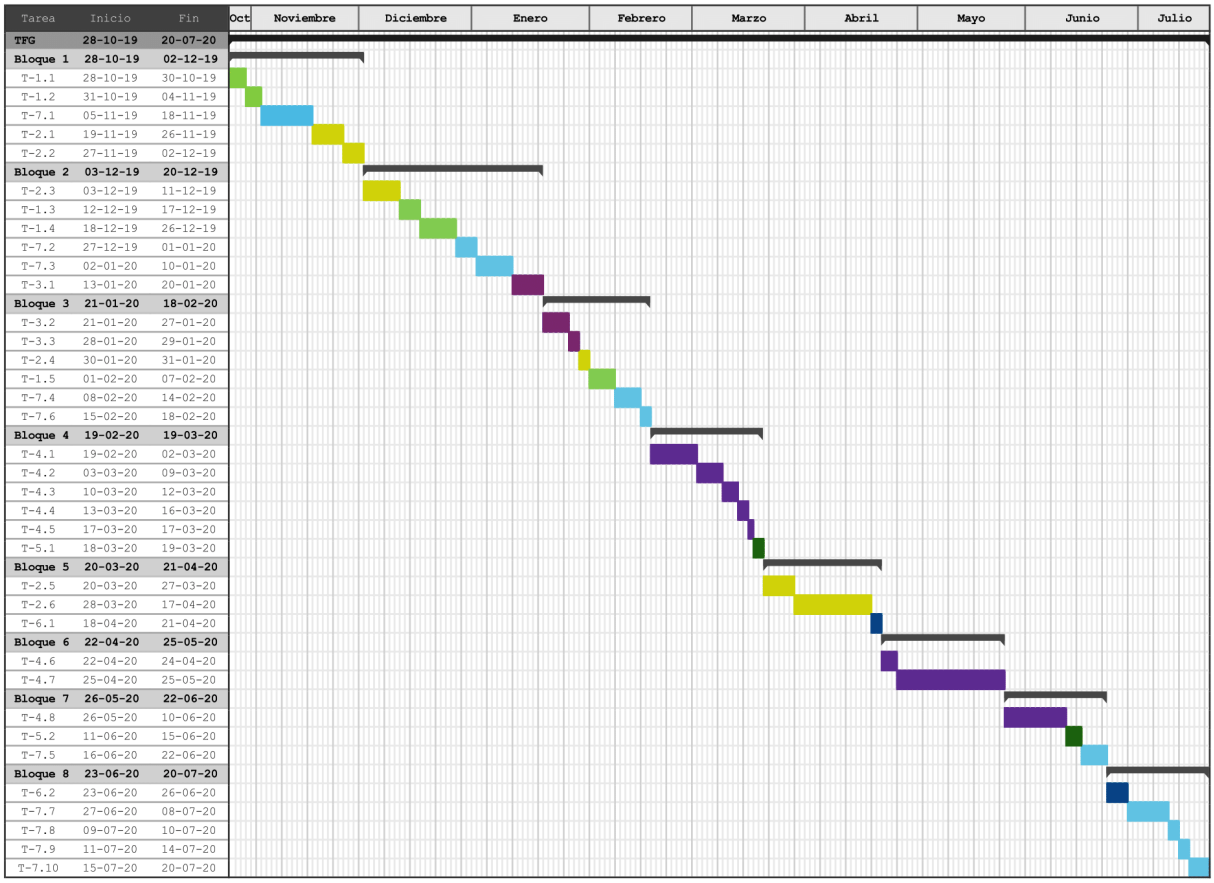


FIGURA 3.2: DIAGRAMA DE GANTT DE PLANIFICACIÓN

[1] *European Credit Transfer and Accumulation System*

3.3. Presupuesto económico

Antes del comienzo es necesario estimar los costes de desarrollo del proyecto. Para la estimación del presupuesto se tendrán en cuenta las herramientas hardware y software utilizadas, atendiendo a los factores de impacto correspondientes a la duración del proyecto, junto con los gastos derivados del personal, atendiendo para ello a la estimación de horas llevada a cabo en la planificación y al tipo de rol correspondiente a cada tarea.

La clasificación se desglosará en cuatro categorías: hardware (Tabla 3.2), servicios (Tabla 3.3), licencias de software (Tabla 3.4) y costes de personal (Tabla 3.5).

	Coste (€)	**Uso (%)**	**Total (€)**
Ordenador portátil	1400,00	8	112,00

Tabla 3.2: Costes de hardware

Para el desarrollo del proyecto se hará uso de una máquina virtual del Departamento de Informática de 16GB de RAM y 20GB de almacenamiento HDD. Para estimar su coste se utilizarán los precios de uso de una máquina virtual de Microsoft Azure con unas especificaciones similares[2].

	Coste	**Uso (%)**	**Duración**	**Total (€)**
Conexión a Internet	60,00 €/mes	10	9 meses	54,00
Servidor de pruebas	0,14 €/hora	100	300 horas	42,00

Tabla 3.3: Costes de servicios

	Licencia	**Coste (€)**	**Uso (%)**	**Total (€)**
Jupyter Notebook	BSD modificada	0	100	0
FileZilla	GPLv2	0	100	0
Texmaker	GPLv2	0	100	0

Tabla 3.4: Costes de licencias software

[2]Véase https://azure.microsoft.com/es-es/pricing/calculator/

Para la estimación del coste de personal se deberá desglosar el coste en función del rol de la tarea. Por ello se establece un salario bruto de 25.000 €para el rol de analista de datos junior y 20.000 €para el rol de desarrollador Python junior, considerando un total de 1735 horas de trabajo anuales (véase el desglose de la nómina en la Tabla 3.5)

	Coste (€/hora)	**Duración (horas)**	**Total (€)**
Analista de datos junior	14,40	291	4.190,40
Desarrollador Python junior	11,53	195	2.248,35

TABLA 3.5: COSTES ESTIMADOS DE PERSONAL

A partir del estudio previo de costes estimados se deduce que el presupuesto final del proyecto es de *6.646,75 €*, de los cuales el $96,87\,\%$ se destina al pago de nóminas (véase la Tabla 3.6).

	Total (€)	**% del total**
Hardware	112,00	1,69
Servicios	96,00	1,44
Licencias software	0,00	0,00
Nóminas	6.438,75	96,87
	6.646,75	100

TABLA 3.6: DESGLOSE TOTAL DE COSTES ESTIMADOS

3.4. Balance final del proyecto

A continuación se lleva a cabo un balance final sobre cómo ha transcurrido el proyecto una vez se ha finalizado, incluyendo un análisis temporal y un balance de costes.

En primer lugar, la Figura 3.3 muestra el calendario del proyecto, con las fechas de inicio y finalización, los días festivos y las reuniones realizadas entre tutor y alumno. Para las reuniones se ha mantenido la estructura fijada en la Sección 2.1.4, manteniendo emails de control, reuniones bajo demanda y reuniones de gestión del proyecto.

A causa de los exámenes finales de junio y la necesidad de dedicar tiempo al Trabajo de Fin de Grado del Grado en Matemáticas, el proyecto se ha visto parado desde el 25/05/2020 al 24/07/2020. Esto ha provocado un retraso en la fecha de entrega del presente proyecto, pasando a ser la nueva fecha de finalización el *21/09/2020*.

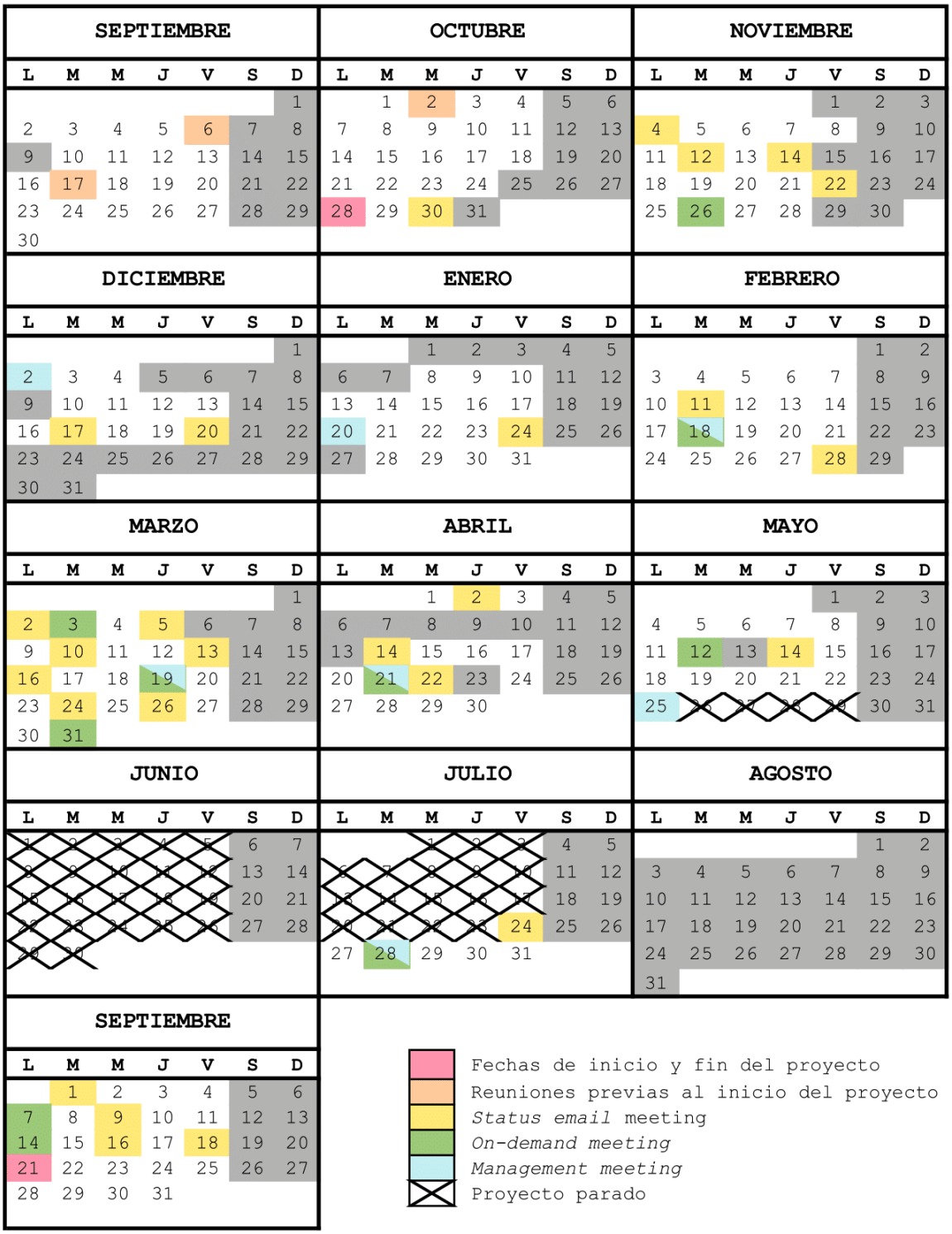


Figura 3.3: Calendario de reuniones del proyecto

Durante el transcurso de la tarea T-2.4 (Estudio de los modelos TBATS) los días 30 y 31 de enero de 2020, se descubrió la posibilidad de poder añadir términos de Fourier al modelo ARIMA para modelar múltiples estacionalidades, lo que derivó en la aparición de 3 nuevas tareas:

- **T-2.7** Estudio de modelos ARIMA con series de Fourier (*1 SP*).
- **T-4.9** Implementación y construcción de los modelos ARIMA con series de Fourier (*1 SP*).
- **T-4.10** Selección y configuración del modelo ARIMA con series de Fourier óptimo (*1 SP*).

Sin embargo, el esfuerzo que se estimó necesario en la reunión de gestión del 18/02/2020 para el desarrollo de estas tareas fue bajo. Por este motivo se decidió incorporar estas nuevas tareas a bloques de trabajo existentes, evitando añadir nuevos bloques de trabajo para no poner en riesgo la fecha de finalización del proyecto. En la Tabla 3.7 se puede observar el cambio en la distribución de tareas con el fin de incorporar las nuevas tareas en los bloques de trabajo restantes. Se incorpora además la modificación del calendario debida al parón de junio y julio.

Bloque	**Fecha de inicio**	**Fecha de finalización**	**Tareas**	**Puntos de historia**	**Progreso**
#1	28/10/2019	02/12/2019	T-1.1 / T-1.2 / T-2.1 / T-2.2 / T-7.1	8	11.3 %
#2	03/12/2019	20/01/2020	T-1.3 / T-1.4 / T-2.3 / T-3.1 / T-7.2 / T-7.3	8.5	23.2 %
#3	21/01/2020	18/02/2020	T-1.5 / T-2.4 / T-3.2 / T-3.3 / T-7.4 / T-7.6	9	35.9 %
#4	19/02/2020	19/03/2020	T-4.1 / T-4.2 / T-4.3 / T-4.4 / T-4.5 / T-5.1 / T-2.7	9.5	47.9 %
#5	20/03/2020	21/04/2020	T-2.5 / T-2.6 / T-6.1 / T-4.9	10	60.6 %
#6	22/04/2020	25/05/2020	T-4.6 / T-4.7 / T-4.10	10	73.2 %
#7	28/07/2020	24/08/2020	T-4.8 / T-5.2 / T-7.5	10	87.3 %
#8	25/08/2020	21/09/2020	T-6.2 / T-7.7 / T-7.8 / T-7.9 / T-7.10	9	100 %

TABLA 3.7: DISTRIBUCIÓN FINAL DE LAS TAREAS DEL PROYECTO EN BLOQUES DE TRABAJO

Por otro lado, la Figura 3.4 muestra el diagrama de Gantt real tras haber desarrollado el proyecto. Los cambios más notables son la incorporación de las nuevas tareas y el parón debido a los exámenes y el proyecto final del grado en matemáticas.

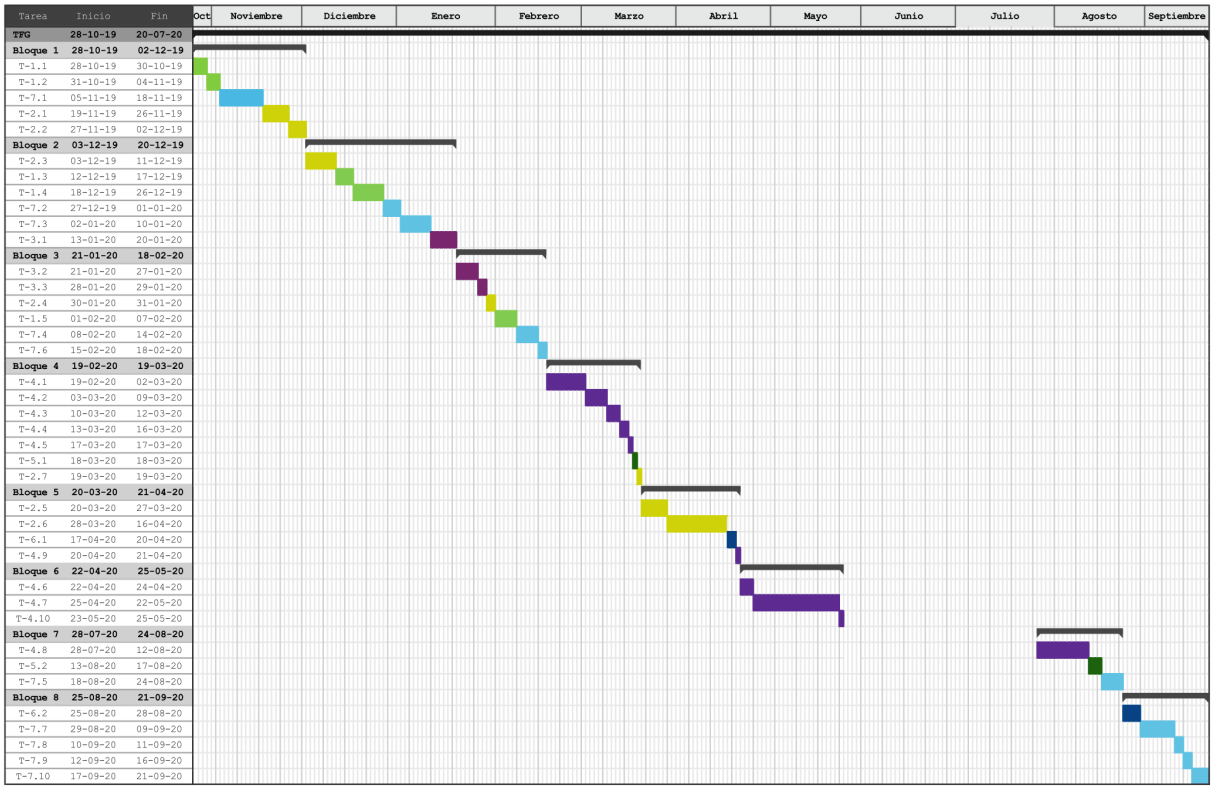


Figura 3.4: Diagrama de Gantt tras la finalización del proyecto

En lo relativo a los costes del proyecto, derivado de las nuevas tareas que han surgido durante el desarrollo del proyecto, el coste final se ha visto incrementado con respecto a los costes presupuestados. En cuanto a los servicios, han sido necesarias 43 horas más de funcionamiento del servidor de pruebas, lo que ha supuesto un incremento en el coste de 6,02 €.

	Coste	**Uso (%)**	**Duración**	**Total (€)**
Conexión a Internet	60,00 €/mes	10	9 meses	54,00
Servidor de pruebas	0,14 €/hora	100	343 horas	48,02

Tabla 3.8: Gastos finales de servicios

Para poder cumplir con los plazos de realización de los bloques de trabajo, la adición de nuevas tareas ha supuesto un incremento del número de horas de trabajo durante los bloques 4,

5 y 6. En total fueron necesarias 23 horas de trabajo a mayores de las estimadas, 14 en tareas de análisis y 9 en tareas de desarrollo, lo que ha derivado en un incremento de 305,37 € en el coste final.

	Coste (€/hora)	**Duración (horas)**	**Total (€)**
Analista de datos junior	14,40	305	4.392,00
Desarrollador Python junior	11,53	204	2.352,12

TABLA 3.9: GASTOS FINALES DE PERSONAL

En el balance final los costes del proyecto ascienden a 6.958,14 (ver Tabla 3.10), con un total de 509 horas de trabajo distribuidas a lo largo del curso en 7 bloques de trabajo con una carga aproximada de 3 horas de trabajo diarias.

	Total (€)	**% del total**	**Incremento (€)**
Hardware	112,00	1,61	0
Servicios	102,02	1,47	6,02
Licencias software	0,00	0,00	0,00
Nóminas	6.744,12	96,92	305,37
	6.958,14	100	311,39

TABLA 3.10: DESGLOSE FINAL DE GASTOS

Capítulo 4

Estado del arte

4.1. Industria 4.0

Industria 4.0 es el nombre con el que se ha apodado la ya establecida Cuarta Revolución Industrial, consistente en la introducción de las tecnologías digitales actuales en la industria. Hablamos por tanto de un cambio hacia una industria más inteligente.

Para comprender las consecuencias de una revolución industrial se debe retornar a la segunda mitad del siglo XVIII en Reino Unido, donde tuvo sus orígenes el proceso de transformación económica, social y tecnológica más importante de la historia, la Primera Revolución Industrial. Este cambio marcó un punto de inflexión en la historia, comenzando una transición que acabará con siglos de mano de obra manual y el uso de la tracción animal, siendo estos sustituidos por maquinaria para la fabricación industrial y el transporte de mercancías y pasajeros. El hito histórico con mayor trascendencia y que supuso el paso definitivo en el éxito de la transformación fue la introducción de la máquina de vapor de James Watt en la industria. Este hecho supuso el desplazamiento de los pequeños talleres por grandes centros fabriles, produciéndose un aumento muy notable en la producción, destacando el sector textil. Como consecuencia del desarrollo industrial se creó una nueva clase social, el proletariado, constituida por los trabajadores de las fábricas que vivían en las ciudades.

Desde mediados del siglo XIX hasta el estallido de la Primera Guerra Mundial en 1914, tuvo lugar la segunda transformación industrial relevante de la historia, conocida como La Segunda Revolución Industrial, marcada por cambios socioeconómicos a causa de los cambios en la vida de la sociedad por la aceleración industrial del momento. Las innovaciones técnicas y científicas, junto con el descubrimiento de nuevas fuentes de energía, como la electricidad y el petroleo, impulsaron el desarrollo de esta nueva transformación industrial, expandiéndose hasta alcanzar el resto de Europa Occidental, Estados Unidos y Japón. Entre los cambios acontecidos en este periodo cabe destacar los cambios organizativos de las empresas: se promueve la producción

sistemática o en serie, aumentando la productividad de las personas y reduciendo costes de producción. Esto produjo un aumento de la competitividad de las grandes empresas, y surgen en este contexto las primeras multinacionales. Tras este modelo nació el moderno capitalismo financiero.

La Tercera Revolución Industrial, también llamada Revolución Científico-Tecnológica o Revolución de la Inteligencia, data sus inicios a mediados del siglo XX, y se asienta sobre las nuevas tecnologías de la información y la comunicación y en el desarrollo de las energías renovables. Se produce un cambio en la producción, automatizándose procesos de fabricación e incorporando los avances en tecnología en las cadenas de producción.

Se ha visto visto por tanto cómo, a lo largo de la historia, la visión sobre el proceso productivo ha ido cambiando, pero con dos fines comunes claros: mejorar la productividad y reducir costes de producción. Como no podría ser de otra forma, la Industria 4.0 también busca estos objetivos. Tal y como ya se adelantaba, la Industria 4.0, también llamada industria inteligente, es el nombre con el que se ha bautizado la Cuarta Revolución Industrial, la cual basa el cambio de paradigma en la adopción de las nuevas tecnologías para la automatización del proceso productivo. Se trata de tecnologías innovadoras cuya aplicación a la industria se irá desarrollando día tras día. Cabe destacar entre estas tecnologías el BigData y el análisis de datos, el Cloud Computing, la ciberseguridad, la robótica colaborativa, los sistemas ciberfísicos, la realidad aumentada, el internet de las cosas (IoT, *Internet of Things*), la fabricación aditiva, la inteligencia operacional... entre otras muchas.

Tal y como puede leerse en Roblek, Meško y Krapež (2016), la transformación de la tecnología todavía está en progreso, y los nuevos avances en Inteligencia Artificial, en Big Data o en conectividad así lo demuestran. Esta transformación digital tendrá una importante influencia en la transformación completa de esta nueva Industria, dado que representará un claro avance en tres puntos estratégicos:

- Digitalización de los sistemas de información aplicados a la gestión y planificación de tareas.
- Creación de sistemas de automatización para la adquisición de datos.
- Vinculación de lugares gracias a un amplio y continuo intercambio automático de datos.

Ya se han comentado tres beneficios que se busca obtener con un cambio en los procesos de producción: la automatización del proceso, la reducción de costes y la mejora de la productividad y la eficiencia. Pero estos no son los únicos. Las empresas que digitalizan sus procesos y hacen uso de plataformas conectadas pueden beneficiarse de una mayor personalización y un control remoto de sus procesos, una mayor adaptabilidad al mercado, un servicio al cliente más especializado, y una disminución de los tiempo de desarrollo, producción y venta, entre otros.

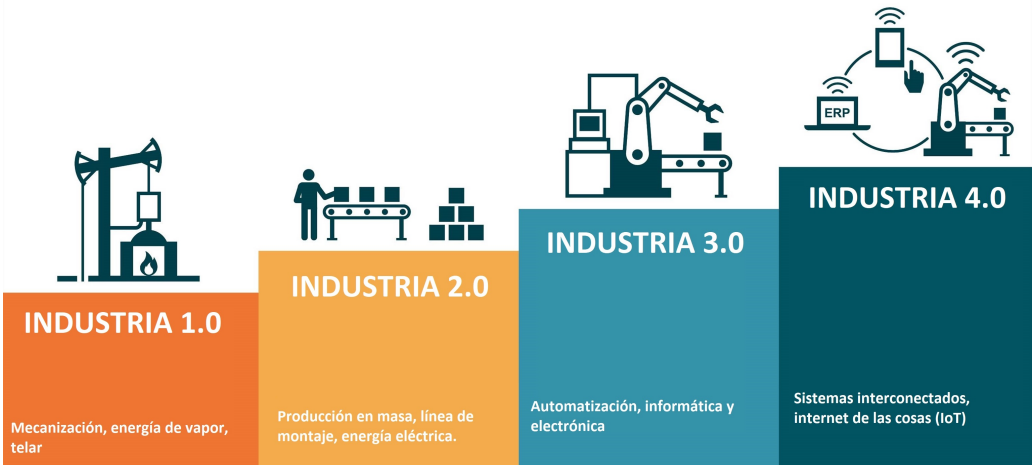


Figura 4.1: Transformaciones industriales a lo largo de la historia

La Industria 4.0 se puede resumir (Lu, 2017) como un proceso de fabricación integrado, adaptado, optimizado, orientado al servicio e interoperable que está correlacionado con algoritmos, datos y nuevas tecnologías.

Demostrando un profundo conocimiento en el área específica, el término "inteligente" se utiliza para hacer referencia a las aplicaciones de la Industria 4.0. Entre estas aplicaciones destacan los edificios inteligentes (*Smart Buildings*), la fabricación y manufacturación inteligente (*Smart Factory and Manufacturing*) y las ciudades inteligentes (*Smart Cities*), entre otras.

4.2. Edificación 4.0 y *Smart Cities*

Dos ejemplos notable de Industria 4.0, que ya hoy en día comienzan a ser una realidad cada vez más extendida, los encontramos en el control remoto y la automatización de los de edificios y ciudades inteligentes.

Los edificios inteligentes, conocidos también como edificios 4.0, son construcciones que utilizan la tecnología para hacer un uso eficiente de sus recursos, además de permitir su control y monitorización de manera remota. El principal objetivo que persiguen estas construcciones es reducir el consumo energético sin sacrificar el confort. Este objetivo es al mismo tiempo el mayor reto en investigación en torno a la edificación inteligente que encontramos en la actualidad.

Dentro del concepto de edificio inteligente se engloban construcciones de todo tipo, tales como naves industriales, locales comerciales, viviendas, espacios deportivos, docentes, sanitarios, culturales, de ocio, o administraciones públicas y privadas. Sin embargo los objetivos que persiguen

estos edificios son independientes a su tipología, y de hecho pueden agruparse en cuatro grandes bloques:

- Nivel arquitectónico. Los edificios deben cumplir las necesidades funcionales ligadas a su uso y a los usuarios finales, dado que serán estos quienes exploten sus características. Entre las garacterísticas arquitectónicas a considerar destacan el *confort*, seguridad y durabilidad futura, además de un diseño adaptativo y modular orientado a facilitar cambios y transformaciones futuros.
- Nivel tecnológico. El edificio debe contar con sistemas integrados dedicados a la motorización y automatización de las instalaciones.
- Nivel ambiental. Estos edificios deben cumplir una serie de compromisos de sostenibilidad y eficiencia energética, que pasan por el tipo de materiales utilizados en su construcción, la explotación de los elementos de iluminación y ventilación natural, o la previsión de reducción de residuos y vertidos contaminantes en su funcionamiento.
- Nivel económico. La Edificación 4.0 debe perseguir la reducción de costes de funcionamiento y mantenimiento, con el fin de alargar su vida útil y generar si cabe un mayor interés de adquisición u ocupación.

Por otro lado, las Ciudades Inteligentes (*Smart Cities*), basan su política de desarrollo en seis grandes pilares: economía, movilidad, medio ambiente, personas, vida y gobernanza inteligente. Las *Samrt Cities* tienen como objetivo garantizar la sostenibilidad de las ciudades, mejorar la calidad de vida y la seguridad de los ciudadanos, y proporcionar eficiencia energética.

4.3. Eficiencia energética

Tal y como se acaba de ver, entre los pilares que sustentan tanto los edificios como las ciudades inteligentes se encuentra la búsqueda de la eficiencia energética. El uso eficiente de la energía se puede definir como la optimización del consumo energética bajo unas determinadas condiciones de confort y niveles de servicio adaptativos. Su aplicación práctica pasa por tanto por la implementación de medidas que minimizan las pérdidas de energía, tratando de conseguir ahorros y ajustes en el consumo energético según las necesidades específicas del cliente. La eficiencia energéticas se encuentra por tanto en las primeras líneas de las agendas políticas y económicas de los países desarrollados.

Gran parte del consumo de los edificios inteligentes (en función del texto se habla de entre un 50 y un 70 % de la demanda) se destina a los sistemas de ventilación, aire acondicionado y

calefacción del edificio. Un sistema HVAC (*Heating, Ventilation and Air Conditioning*) es un sistema de climatización y ventilación. Su finalidad es la de proporcionar al usuario un ambiente interior cuya temperatura, tasa de humedad relativa y purificación del aire sean confortables. Su importancia en estas edificaciones hace que muchos estudios traten de centrar únicamente sus esfuerzos en reducir, mediante la monitorización y la gestión de los datos del edificio, los consumos de estos sistemas.

4.4. Trabajos previos

Como punto de partida para la realización del presente trabajo se han considerados estudios previos realizados por diversos grupos de la Universidad de Vanderbilt, Nashville (Estados Unidos), véase así Aviek Naug y col., 2019, 2018. En estos proyectos se trata de dar una visión general sobre la importancia de gestionar y automatizar los procesos de predicción de consumos de energía, centrándose en el modelado de los sistemas HVAC. Se proponen diversas alternativas basadas principalmente en el uso de los datos disponibles del edificio. Entre los algoritmos propuestos destacan las Redes LSTM, diversos algoritmos de *clustering* jerárquico o Máquinas de Vectores de Soporte.

Por otro lado, aunque en la misma línea, el artículo de aborda el probelma de predicción mediante diversas estructuras y configuraciones de las redes neuronales recurrentes, entre las que destaca la aplicación de una arquitectura de codificación y decodificación de secuencias de consumos construida sobre redes LSTM, que recibe el nombre de *Encoder-Decoder*. Los resultados tan prometedores de este estudio hacen de este modelo un potencial candidato a ser utilizando en el presente proyecto.

Ya por último, algunos autores como Ozturk y Ozturk (2018), tratan el modelado de consumos mediante modelos estadísticos clásicos, tales como modelos autorregresivos o de alisado exponencial. Estos modelos serán utilizados como punto de partida para comenzar el estudio de las predicciones y como base sobre la que comparar los nuevos enfoques.

Capítulo 5

Modelos de predicción

El riesgo y la incertidumbre son problemas reales en el contexto de la toma de decisiones con carácter futuro. Es por ello que el disponer en el presente de un conocimiento futuro, aun siendo este aproximado, facilita la toma de decisiones. En el contexto de la edificación 4.0, en concreto el caso a estudio de la construcción de modelos de predicción de consumo, este conocimiento futuro permitirá determinar patrones y tendencias, los cuales, contextualizados en función del fin del edificio, mostrarán hábitos y rutinas ligados a su uso. En el caso de disponer de información sobre el consumo estimado en un futuro próximo basando su predicción en variables del sistema permitirá detectar a mayores posibles anomalías que puedan derivarse del deterioro de ciertas componentes de los sistemas implicados.

En esta búsqueda de la aproximación del conocimiento futuro toman especial relevancia los modelos de predicción, y en el caso a estudio que nos concierne se centra el estudio en los modelos de predicción basados en series temporales, dada la naturaleza de los datos disponibles. Un modelo predictivo o modelo de predicción es un mecanismo utilizado para aproximar el conocimiento futuro a partir de los datos conocidos en el momento de su aplicación. Por su parte, los modelos de predicción basados en series temporales utilizan un histórico de datos como base para la estimación de las observaciones futuras.

En esta sección se exponen las dos clases principales de modelos de predicción con series temporales: los modelos ARIMA (Box y Jenkins, 1970), los modelos TBATS (De Livera, Hyndman y Snyder, 2011) y las Redes Neuronales Recurrentes o RNN (de sus siglas en ingles Recurrent Neural Network) (Rumelhart, Hinton y Williams, 1986). El estudio se centra en la fundamentación de estas clases, extendiendo su uso con diversas variantes derivadas del modelo clásico ARIMA debidas a la naturaleza estacional de los datos, junto con el estudio de un modelo específico de RNN, la red neuronal LSTM (acrónimo del ingles Long Short-Term Memory) cuyo uso, tal y como se deducirá de sus propiedades, se adapta a los objetivos de predicción en el contexto del análisis del consumo de energía.

Si bien antes de comenzar con el estudio de los modelos de predicción sobre series temporales se expone una breve introducción a los aspectos más relevantes de las series temporales, dada su importancia en el modelado del problema.

5.1. Introducción a las series temporales

Para el desarrollo de los modelos de predicción que permitirán conocer de forma aproximada los consumos futuros de energía es clave, tal y como se adelantaba previamente, el concepto de serie temporal. Una **serie temporal** se define como la sucesión de observaciones correspondientes a una variable a lo largo de intervalos regulares de tiempo. De esta forma, en función de los periodos de tiempo en los que vengan recogidos los datos que componen la serie temporal, se habla de series temporales con periodicidad diaria, mensual, trimestral, anual, etc. El objetivo del análisis de las series temporales es el conocimiento de un patrón o secuencia de comportamiento que permita prever la evolución futura de la variable a estudio, suponiendo que las condiciones del entorno futuras no cambiarán con respecto a las condiciones que enmarcan las observaciones pasadas.

Desde un punto de vista teórico y bajo el objetivo de comprender la evolución temporal de una determinada variable, el enfoque clásico de series temporales considera que el comportamiento de una variable en el tiempo es el resultado de la integración de cuatro componentes: tendencia, ciclo, componente estacional y componente irregular o residual:

- Se considera **tendencia** al movimiento suave y regular de la serie a largo plazo, bien sea de crecimiento, decrecimiento o estabilización. La tendencia puede visualizarse como una pendiente, no necesariamente lineal, a través de las observaciones en largos periodos de tiempo.

- Los **ciclos** se definen como las fluctuaciones repetidas no periódicas. La duración de las fluctuaciones dependerá de la naturaleza de la serie. El concepto de ciclo toma especial relevancia en el contexto económico, en el cual estos movimientos oscilatorios se presentan como consecuencia de la propia dinámica del sistema. En el estudio general de las descomposiciones de series temporales la componente cíclica se incluye en conjunto con la tendencia de la serie.

- La **estacionalidad** se define como los movimientos regulares de la serie que se repiten bajo una periodicidad fija (cada semana, cada día, cada hora, etc.). La estacionalidad se observa como patrones cíclicos repetidos periódicamente e influidos por factores estacionales ligados al contexto de la serie.

- La **componente irregular o residual** describe las influencias irregulares y aleatorias en la serie temporal. Esta componente incluye las variaciones de la serie que no pueden ser explicadas dado que no siguen un comportamiento sistemático o regular, y en consecuencia no es posible su predicción.

Si denotamos por y_t el valor que toma la variable a estudio en el instante t, tal y como se ha visto, este valor puede ser explicado por un factor de tendencia (T_t) que recoge las oscilaciones a largo plazo y las oscilaciones no periódicas (ciclos), un factor estacional (E_t) caracterizado por variaciones regulares, y un factor residual (R_t) formado por las fluctuaciones que no pueden ser explicadas por las componentes previas. De esta forma es posible la descomposición $y_t = f(T_t, E_t, R_t)$, donde f denota una función, más o menos compleja, que relaciona las componentes para dar lugar al valor de la variable. En la práctica destacan por su simplicidad dos descomposiciones, la descomposición aditiva y la descomposición multiplicativa:

- Si asumimos una **descomposición aditiva**, el modelo se describe a partir de sus componente en la forma $y_t = T_t + E_t + R_t$, de tal manera que el valor de la serie para cada periodo de tiempo t resulta de la suma del valor de cada una de las componentes. La descomposición aditiva es la más apropiada si la magnitud de las fluctuaciones estacionales y de las oscilaciones en torno a la tendencia no varía con el tiempo.

- Alternativamente, si consideramos una **descomposición multiplicativa**, el modelo puede escribirse en la forma $y_t = T_t \times E_t \times R_t$ para cada periodo t de tiempo. En este caso la descomposición multiplicativa es más adecuada si la variabilidad de los patrones estacionales o la variabilidad en torno a la tendencia no se mantiene constante a lo largo del tiempo, viéndose esta incrementada a medida que aumenta el tiempo.

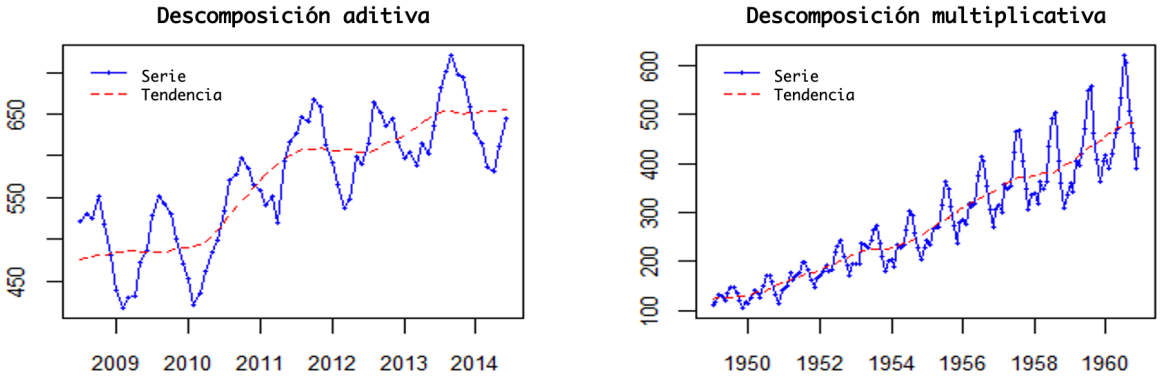


Figura 5.1: Ejemplos de aplicación de los esquemas de descomposición aditiva y multiplicativa

Antes de dar paso al estudio teórico de los modelos de predicción, un último concepto clave en el análisis de series temporales es el concepto de **estacionariedad**. Una serie estacionaria es aquella cuyas propiedades no dependen del instante en el cual la serie es observada. Para una definición más formal de estacionariedad, es necesario recurrir a diversos conceptos matemáticos enmarcados en el área de la teoría de probabilidad. Veamos a continuación de una forma liviana, sin entrar en excesivos detalles y demostraciones, los conceptos que dan forma a la estacionariedad de la serie.

Un enfoque común en el análisis de series temporales estriba en considerar las observaciones de la serie temporal como parte de la realización de un proceso estocástico. Para la definición de proceso estocástico, se requiere en primer lugar el establecimiento de los conceptos de espacio probabilístico y variable aleatoria:

- Un espacio probabilístico es una tripleta $(\Omega, \mathcal{A}, P)$ donde
  1. Ω es un conjunto no vacío, llamado espacio muestral.
  2. F es una σ-álgebra de subconjuntos de Ω, esto es, una familia cerrada con respecto a la unión numerable y el complementario de subconjuntos de Ω.
  3. P es una medida de probabilidad definida sobre los elementos de $\mathcal{A}$.

- Una variable aleatoria real, o variable estocástica real, definida sobre el espacio probabilístico $(\Omega, \mathcal{A}, P)$ es una aplicación $X : \Omega \to \mathbb{R}$ medible, esto es, la contraimagen de todo intervalo $(-\infty, a] \subset \mathbb{R}$ pertenece a $\mathcal{A}$ $\big[$es decir, $X^{-1}\big((-\infty, a]\big) \in \mathcal{A}$ para todo $a \in \mathbb{R}\big]$.

Se define así un **proceso estocástico** real como una familia $X = \{X_i(\omega),\, i \in T\}$ de variables aleatorias reales indexadas por un conjunto T, todas ellas definidas sobre el mismo espacio de probabilidad $(\Omega, \mathcal{A}, P)$. El conjunto T se denomina conjunto de índices del proceso. Si $T \subset \mathbb{Z}$, el proceso se denomina proceso estocástico discreto, y en cambio si $T \subset \mathbb{R}$, el proceso estocástico se denomina continuo. Dado un conjunto finito de índices $T = \{t_1, ..., t_n\}$, se define la función de distribución conjunta del proceso estocástico $X = \{X_i(\omega),\, i \in T\}$ como

$$F_{t_1,...,t_n}(x_{t_1}, ..., x_{t_n}) = P\big(X_{t_1}(\omega) \leq x_{t_1}, ..., X_{t_n}(\omega) \leq x_{t_n}\big)$$

Si no hay confusión denotaremos $F_X(x_{t_1}, ..., x_{t_n}) = F_{t_1,...,t_n}(x_{t_1}, ..., x_{t_n})$ para hacer referencia a la distribución conjunta de las variables con índices $t_1, ..., t_n$ del proceso estocástico X.

A partir de la definición de proceso estocástico se define serie temporal como una realización de un proceso estocástico en tiempo discreto, donde los elementos del conjunto de índices T están ordenados y corresponden a instantes equiespaciados de tiempo.

Teniendo una definición básica de proceso estocástico desde la que partir, es posible ahora introducir el concepto de estacionariedad. La estacionariedad es una propiedad asociada a los procesos estocásticos, y no a las realizaciones de dichos procesos (es decir, la propiedad de estacionariedad no es nativa de las series temporales, sino de los procesos estocásticos que desde el punto de vista probabilístico las definen).

Un proceso estocástico es fuertemente estacionario si la distribución conjunta de toda secuencia finita de variables aleatorias del proceso permanece invariante ante traslaciones de los índices. Formalmente, el proceso estocástico $X = \{X_i(\omega),\, i \in T\}$ es fuertemente estacionario si

$$F_X(x_{t_{1+s}}, ..., x_{t_{n+s}}) = F_X(x_{t_1}, ..., x_{t_n})$$

para cada $n \in \mathbb{N}$ y $s \in \mathbb{Z}$. Esta definición, aun no siendo la única, es la más extendida, y comúnmente se hace referencia a ella simplemente como estacionariedad. Cuando el conjunto T de índices se interpreta como el tiempo, tal y como ocurre con el caso que nos atañe, un proceso estocástico es estacionario si permanece invariante a lo largo del tiempo.

A partir de la definición de estacionariedad se deduce que, desde el punto de vista de las series temporales, una serie temporal con tendencia y/o estacionalidad no es estacionaria, ya que tanto la tendencia como la estacionalidad influyen directamente sobre los valores de la serie en función del tiempo. Por el contrario, una serie temporal de ruido blanco[1] sí forma una serie estacionaria. Tal y como se observa en la Figura 5.2, las series temporales con media, varianza y autocovarianza no constantes suponen ejemplos de series temporales no estacionarias, puesto que esta variabilidad es contraria a la igualdad de las respectivas distribuciones conjuntas del proceso estocástico asociado.

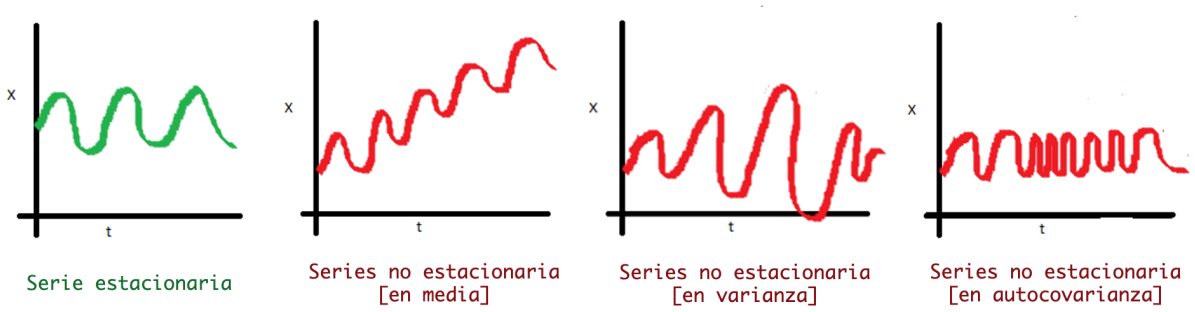


Figura 5.2: Ejemplos de series temporales no estacionarias

[1]Ruido blanco: proceso estocástico incorrelado, de media 0 y varianza finita y constante

5.2. Modelos ARIMA

En el contexto del análisis de series temporales, los **modelos Autorregresivos Integrados de Media Móvil** o **modelos ARIMA** (acrónimo del inglés *Autoregressive Integrated Moving Average*) constituyen una clase de modelos predictivos utilizados para aproximar valores futuros de la serie temporal a partir de los valores pasados de la serie histórica. Su uso fue popularizado por los estadísticos británicos George E. P. Box y Gwilym M. Jenkins (1970), quienes desarrollaron una metodología, llamada metodología de Box-Jenkins en su honor, aplicable sobre estos modelos y dirigida a encontrar el modelo que mejor se ajusta a la serie temporal.

Los modelos ARIMA son **modelos de predicción de series temporales univariantes**, dado que sólo es posible analizar conjuntos de observaciones sobre una única variable. No obstante, existen generalizaciones de modelos de predicción ARIMA al caso de series temporales multivariantes, como los modelos VARMA (*Vectorial Autoregressive Moving Average*), para trabajar con series temporales constituidas como conjuntos de observaciones de varios procesos estocásticos (puede consultarse Reinsel, 2003, para ampliar información).

Si nos situamos en un instante t de la serie temporal, el objetivo buscado por un modelo ARIMA radica en estimar el valor de la observación x_t haciendo uso para ello únicamente de las observaciones pasadas. Los modelos ARIMA obtienen el valor futuro por un lado como combinación lineal de las propias observaciones pasadas de la serie, y por otro como combinación lineal de los errores de predicción pasados. Un modelo ARIMA consta así de tres partes:

- Una suma ponderada de valores retardados de la serie (parte Autoregresiva)
- Una suma ponderada de errores de predicción retardados (parte de Media Móvil)
- La serie de diferencias de los valores de la serie temporal original (parte Integrada)

La tercera y última parte que define el modelo ARIMA tiene su fundamentación en el concepto de **diferenciación** de la serie, o **serie de diferencias**. Para eliminar la variabilidad en media de una serie temporal, es preciso tomar en lugar de la serie original, la serie de diferencias, obtenida a partir de las variaciones entre observaciones consecutivas:

$$x'_t = x_t - x_{t-1}$$

Si la serie original posee un total de N observaciones, la serie de diferencias cuenta, como es evidente, con $N - 1$ observaciones, dado que no es posible calcular la diferencia x'_1 para la primera observación. Aunque no es habitual, puede ocurrir que sea necesario tomar más de una diferencia en aquellos casos en los que no se haya obtenido una serie estacionaria tras calcular su serie de diferencias. De este modo se define de forma recursiva la serie de diferencias de segundo orden como

$$\begin{aligned} x''_t &= x'_t - x'_{t-1} \\ &= (x_t - x_{t-1}) - (x_{t-1} - x_{t-2}) \\ &= x_t - 2x_{t-1} + x_{t-2} \end{aligned}$$

De forma análoga a lo ocurrido con la serie de diferencias, la serie de diferencias de segundo orden cuenta con un total de $N-2$ términos, dada la imposibilidad de calcular los dos primeros valores x''_1 y x''_2.

Volviendo a los modelos ARIMA, la necesidad de transformar la serie a estudio para obtener una serie estacionaria a partir de la cual trabajar con las componentes Autorregresiva y de Media Móvil se asienta sobre la condición necesaria de estacionariedad como requisito básico para la acpliacción de dichos modelos.

De forma general un modelo ARIMA se define a partir de tres parámetros (p, d y q), cada uno de los cuales se corresponde con una de las tres partes en las que se ha dividido el modelo:

- El parámetro $\boldsymbol{p}$ denota el número de observaciones previas o retardos utilizados para estimar el siguiente valor de la serie.
- El parámetro $\boldsymbol{q}$ denota el número de errores de predicción pasados utilizados para estimar el siguiente valor de la serie.
- El parámetro $\boldsymbol{d}$ denota el orden de diferenciación de la serie, de forma que la serie de diferencias de orden d constituya una serie estacionaria.

A partir de estos parámetros un modelo $\boldsymbol{ARIMA(p,d,q)}$ se representa en la forma

$$x_t^{(d)} = c + \phi_1 x_{t-1}^{(d)} + \cdots + \phi_p x_{t-p}^{(d)} + \epsilon_t - \theta_1 \epsilon_{t-1} - \cdots - \theta_q \epsilon_{t-q}$$

donde $x_t^{(d)}$ representa la observación en el instante t de la serie de diferencias de orden d, y los términos de error ϵ_t se asume que son muestras de variables independientes e igualmente distribuidas con distribución normal de media cero y varianza finita σ_ϵ^2.

Para simplificar la escritura de los modelos ARIMA es de utilidad la aplicación de varios operadores que actúan sobre elementos de la serie temporal dando como resultado nuevos elementos de la serie. Estos operadores clásicos se conocen con los nombres de operador de retardos y operador de diferencias y, dada una serie $\{x_1, x_2, ...\}$, verifican:

- El **operador de retardos**, B, toma como entrada un elemento de la serie temporal y devuelve el elemento previo en la serie, esto es, $B(x_t) = x_{t-1}$.

- El **operador de diferencias**, ∇, toma como entrada un elemento de la serie temporal y retorna la diferencia entre dicho elemento y su elemento previo, esto es, $\nabla(x_t) = x_t - x_{t-1}$.

Nótese que se verifica $\nabla = 1 - B$, dado que $\nabla(x_t) = x_t - x_{t-1} = x_t - B(x_t) = (1 - B)(x_t)$. Además, ambos operadores pueden aplicarse de forma recursiva, obteniendo de este modo dos nuevos operadores: el operador de retardo de orden k, $B^k(x_t) = x_{t-k}$, y el operador de diferencias acumuladas u operador de diferencia de orden l, $\nabla^l(x_t) = \nabla\big(\nabla^{l-1}(x_t)\big)$. Haciendo uso de estos operadores es posible rescribir el modelo ARIMA, obteniendo así:

$$(1 - \phi_1 B - \cdots - \phi_p B^p)\,\nabla^d x_t = c + (1 - \theta_1 B - \cdots - \theta_q B^q)\,\epsilon_t \tag{5.1}$$

Volviendo al punto de vista de la teoría de la probabilidad, la ecuación 5.1 anterior representa el modelado de un nuevo proceso estocástico, conocido como proceso ARIMA(p,d,q), en el cual interviene un segundo proceso estocástico estacionario de errores, $\{\epsilon_t\}_{t\in T}$ compuesto por variables aleatorias reales cuyo único conocimiento del que se dispone es su distribución (supuesta normal de media cero y varianza σ_ϵ^2 finita). Conocida una realización concreta del proceso estocástico de errores en el instante t ($\{\epsilon_t\}$) y conocidos los valores de las p observaciones previas de la variable a estudio ($\{x_{t-1}, ..., x_{t-p}\}$) y de los q errores previos ($\{\epsilon_{t-1}, ..., \epsilon_{t-q}\}$), es posible obtener una realización del proceso ARIMA, logrando así un nuevo valor de la serie temporal de observaciones, de la cual conocemos de forma explícita el proceso estocástico de generación.

Sin embargo, en el contexto de los modelos de predicción, es conocida la serie temporal de observaciones, pero se desconoce el proceso estocástico que las ha generado. Por este motivo, la construcción de un modelo de predicción ARIMA consiste en la búsqueda del proceso estocástico ARIMA(p,d,q) que ha podido generar la serie temporal objeto de estudio. Una vez conocido el proceso estocástico de generación será posible llevar a cabo la generación de nuevas muestras, utilizando para ello el histórico de observaciones de la serie (véase Figura 5.3).

Tras conocer la estructura de un modelo ARIMA, los siguientes dos pasos necesarios para completar la construcción (inferencia) del modelo son la determinación del orden del modelo (esto es, determinar los valores de los parámetros p, d y q), y la estimación de los coeficientes $c, \phi_1, ...\phi_p, \theta_1, ..., \theta_q$.

Estimación de coeficientes

Supuesto conocido en primer lugar el orden del modelo ARIMA (los valores p, d y q), se debe fijar un criterio a seguir para el cálculo de los coeficientes. Dos de los métodos de estimación más extendidos con aplicación directa en los modelos ARIMA son la estimación por mínimos cuadrados condicionados y la estimación por máxima log-verosimilitud condicionada. En ambos

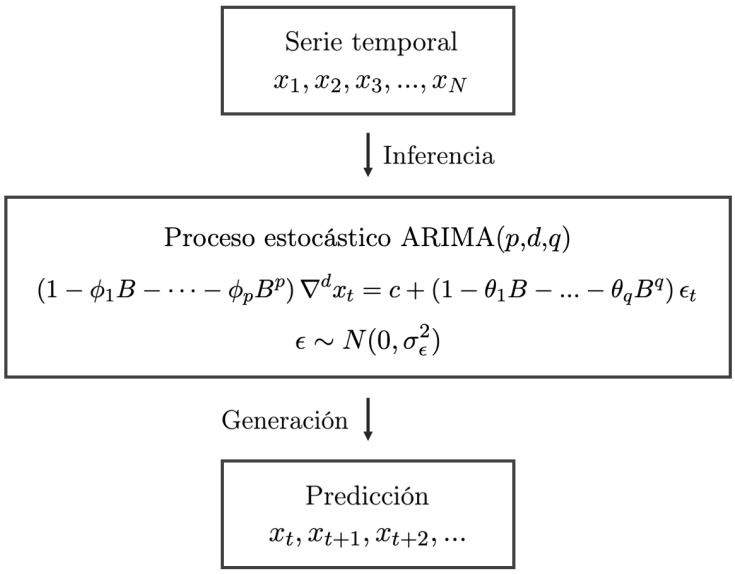


Figura 5.3: Mecanismo de inferencia y generación sobre un proceso estocástico ARIMA(p,d,q)

casos la estimación se basa en el estudio de los residuos del modelo definidos como la diferencia entre el valor estimado por el modelo y el valor real de la serie, esto es, dados los coeficientes $(\tilde{c}, \tilde{\phi_1}, ..., \tilde{\phi_p}, \tilde{\theta_1}, ..., \tilde{\theta_q})$, se define el residuo en el instante de tiempo t como

$$e_t = y_t - \tilde{c} - \tilde{\phi}_1\, y_{t-1} - \cdots - \tilde{\phi}_p\, y_{t-p} + \tilde{\theta}_1\, e_{t-1} + \cdots + \tilde{\theta}_q\, e_{t-q}$$

donde, para simplificar, se han definido las observaciones de la serie de diferencias $y_t = \nabla^d x_t$ para cada $t \in T$. Para la aplicación de los métodos de estimación de coeficientes se requiere a mayores el conocimiento de todos los valores de la serie temporal, así como de los primero q residuos $e_1, e_2, ..., e_q$ (dado que no podrán ser calculados a partir del modelo), que se supondrán iguales a 0. A partir de estos supuestos, los métodos de estimación de parámetros se definen del siguiente modo:

- Estimación por **mínimos cuadrados condicionados**. Dados $(\tilde{c}, \tilde{\phi_1}, ..., \tilde{\phi_p}, \tilde{\theta_1}, ..., \tilde{\theta_q})$, y considerados los residuos $\hat{\epsilon}_t$, la estimación de los parámetros $c, \phi_1, ...\phi_p, \theta_1, ..., \theta_q$ por medio del método de mínimos cuadrados se obtiene a través de los valores $\tilde{c}, \tilde{\phi_1}, ..., \tilde{\phi_p}, \tilde{\theta_1}, ..., \tilde{\theta_q}$ que minimizan la suma de los cuadrados de los residuos, es decir, los valores que minimizan la función

  $$S(\tilde{c}, \tilde{\phi_1}, ..., \tilde{\phi_p}, \tilde{\theta_1}, ..., \tilde{\theta_q}) = \sum_{t=1}^{N} e_t^2,$$

  condicionada a los valores de los residuos iniciales $e_1, ..., e_q$.

- Estimación por **máxima log-verosimilitud condicionada**. Dado el vector de coeficientes $(\tilde{c}, \tilde{\phi_1}, ..., \tilde{\phi_p}, \tilde{\theta_1}, ..., \tilde{\theta_q})$, tomando para simplificar la serie de diferencias $y_t = \nabla^d x_t$ y llamando $r = max(p, q)$, la estimación de los parámetros $c, \phi_1, ...\phi_p, \theta_1, ..., \theta_q$ por medio del método de máxima log-verosimilitud condicionado se obtiene a través de los valores $\tilde{c}, \tilde{\phi_1}, ..., \tilde{\phi_p}, \tilde{\theta_1}, ..., \tilde{\theta_q}$ que maximizan la función de log-verosimilitud

$$\mathscr{L}_c\left(\tilde{c}, \tilde{\phi_1}, ..., \tilde{\phi_p}, \tilde{\theta_1}, ..., \tilde{\theta_q}, \tilde{\sigma}_\epsilon^2\right) = \frac{-(T-r)}{2} log(\tilde{\sigma}_\epsilon^2) - \sum_{t=1}^{N} e_t^2$$

  condicionada a los valores de los residuos iniciales $e_1, ..., e_q$.

Nótese que el procedimiento de estimación por mínimos cuadrados no tiene en cuanta el valor de la varianza σ_ϵ^2 del proceso estocástico de errores de cara a la obtención de los parámetros óptimos. Por ello de forma general se prescinde de este procedimiento, limitando su uso al caso de modelos ARIMA en los cuales $q = 0$. Para ampliar la información sobre las técnicas de estimación de coeficientes, consúltese Peña (2005).

Para el cómputo de los coeficientes, existen varios algoritmos numéricos que implementan el proceso de optimización. Entre ello, el más extendido y que además se utiliza por defecto en las implementaciones de los modelos ARIMA en las librerías *statsmodels* y *pmdarima* de Python, es el algoritmo **L-BFGS** (de sus siglas en ingles, *Limited-memory Broyden–Fletcher–Goldfarb–Shanno*). El algoritmo L-BFGS es un método iterativo para la resolución de problemas de optimización no lineales, variante del algoritmo BFGS con memoria limitada, adecuado para problemas con un gran número de variables. El algoritmo lleva el nombre de los matemáticos especializados en problemas de optimización Charles George Broyden, Roger Fletcher, Donald Goldfarb y David Shanno.

Criterios de selección de modelos

Tras conocer el procedimiento para calcular los coeficientes del modelo, el último paso necesario se basa en la elección de los parámetros p, d y q de forma que el modelo ARIMA(p, d, q) obtenido sea el que mejor aproxime la serie temporal. Para ello se deberá seguir un criterio de evaluación que analice el grado de calidad del modelo, basándose para ello en el ajuste y en la complejidad del mismo. El criterio de uso más extendido con los modelos ARIMA es el denominado **Criterio de Información de Akaike** o **AIC** (de sus siglas en inglés *Akaike Information Criterion*), utilizado como medida de calidad relativa del modelo estadístico. En el caso general, el AIC evalúa cada unos de los modelos de la familia paramétrica de parámetros p, d y q, y selecciona aquel modelo para el que se minimice la expresión

$$AIC = 2k - 2\,log(L)$$

donde k representa el número de parámetros del modelo estadístico, y L es el máximo valor de la función de verosimilitud para el modelo estimado. En el caso particular de los modelos ARIMA, el número de parámetros es $p + q + k + 1$, donde k toma el valor 1 si $c \neq 0$ o 0 si $c = 0$ (nótese que queda incluido el parámetro σ_ϵ^2 de la varianza de los residuos). De esta forma, se tiene

$$AIC = 2(p + q + k + 1) - 2\,log(L)$$

Se deberá escoger el modelo con menos valor de AIC. El Criterio de Información de Akaike recompensa el ajuste (evaluado mediante la función de verosimilitud), pero sin embargo incluye una penalización como función creciente del número de parámetros estimados, tratando de evitar el sobreajuste del modelo. En los casos en los cuales la muestra es pequeña, hay una probabilidad sustancial de que el AIC seleccione modelos con un alto número de parámetros, provocando un aumento de la complejidad del modelo junto con el correspondiente riego de sobreajuste. Para controlar este hecho, se utiliza el AIC corregido, o AICc. La ecuación que define el AICc depende del modelo estadístico, y para el caso de los modelos ARIMA es

$$AICc = AIC + \frac{2\,(p + q + k + 1)\,(p + q + k + 2)}{N - p - q - k - 2}$$

con N el tamaño de la muestra. Otro posible criterio para la selección del modelo es el **Criterio de Información Bayesiano** o **BIC** (*Bayesian Information Criterion*), cuyo valor por modelo se calcula siguiendo la ecuación

$$BIC = log(N)\,(p + q + k + 1) - 2\,log(L) = AIC + [log(N) - 2]\,(p + q + k + 1)$$

que de nuevo penaliza la complejidad del modelo, manteniendo como regla de selección aquel modelo que minimice esta cantidad. Nótese además que la penalización de la complejidad de este criterio es mayor que en los casos previos.

Limitación de los modelos ARIMA

Si la serie temporal presenta patrones estacionales, el modelo ARIMA no se ajustará correctamente a estas fluctuaciones periódicas, dado que la técnica de diferenciación estudiada no permite transformar una serie temporal con patrones estacionales en una serie estacionaria. Es por ello que se requiere un proceso alternativo de transformación, siendo necesario añadir nuevos términos al modelo con el fin de ajustar el carácter estacional. Para su modelización, se presentan a continuación dos nuevos modelos de predicción derivados del modelo ARIMA: los modelos SARIMA y los modelos ARIMA con términos de Fourier como variables exógenas externas.

5.2.1. Modelos SARIMA

Como extensión de los modelos ARIMA surgen los modelos ARIMA estacionales o **modelos SARIMA** (*Seasonal ARIMA models*), los cuales tratan de solventar el problema derivado de la estacionalidad de las series temporales, dado que la técnica de transformación mediante diferencias recursivas, tal y como ya se ha mencionado, no permite lograr la estacionariedad de una serie con patrones estacionales. Derivado de este nuevo modelo aparece el concepto de **diferenciación estacional**, o serie de diferencias estacionales. Para lograr eliminar la variabilidad en media de una serie con patrones estacionales se debe tomar, en lugar de la serie original, la serie obtenida de la diferenciación entre observaciones separadas a distancia el periodo de estacionalidad, esto es, debe tomarse la serie de diferencias estacionales

$$y_t = x_t - x_{t-m}$$

donde m representa el periodo de la estacionalidad. De forma análoga a lo ocurrido con la diferenciación ordinaria, si la serie original se compone de un total de N observaciones, la serie de diferencias estacionales con periodo m posee un total de $N - m$ observaciones, dado que no es posible calcular las m primeras diferencias.

Surge de esta forma la necesidad de definir un nuevo operador que actúe sobre elementos de la serie y represente la operación de diferenciación estacional previa. Se define así el operador de diferencias estacionales, ∇_m, el cual toma como entrada un elemento de la serie temporal y devuelve la diferencia entre el propio elemento y el elemento m veces anterior, esto es,

$$\nabla_m(x_t) = x_t - x_{t-m}$$

Nótese que se verifica $\nabla_m = 1 - B^m$, dado que $\nabla_m(x_t) = x_t - x_{t-m} = x_t - B^m(x_t) = (1 - B^m)(x_t)$. Además, el operador se puede aplicar de forma recursiva, obteniendo así el operador de diferencias estacionales acumuladas de orden l, $\nabla_m^l(x_t) = \nabla_m\big(\nabla_m^{l-1}(x_t)\big)$.

El modelo SARIMA cuenta con un total de 7 parámetros, los 3 primeros correspondientes al modelo ARIMA subyacente, y 4 nuevos parámetros para modelar la estacionalidad de la serie temporal a estudio. Se obtiene así una nueva clase paramétrica de modelos de predicción, los modelos $\boldsymbol{SARIMA(p,d,q)(P,D,Q)_m}$. Los nuevos parámetros a estudio son:

- El parámetro $\boldsymbol{m}$ denota el periodo de estacionalidad de la serie.
- El parámetro $\boldsymbol{P}$ denota el número de retardos estacionales (observaciones periódicas pasadas x_{t-m}, x_{t-2m},x_{t-3m}...) utilizados para la estimación del siguiente valor de la serie.

- El parámetro $\boldsymbol{Q}$ denota el número de errores de predicción pasados a distancia periódica (ϵ_{t-m}, ϵ_{t-2m},ϵ_{t-3m}...) utilizados para la estimación del siguiente valor de la serie.
- El parámetro $\boldsymbol{D}$ denota el orden de diferenciación estacional de la serie.

Haciendo uso de los operadores ya estudiados, un modelo SARIMA$(p,d,q)(P,D,Q)_m$ viene dado por la expresión

$$\begin{aligned}(1-\phi_1 B-\cdots-\phi_p B^p)(1-\Phi_1 B^m-\cdots-\Phi_p B^{Pm})\,\nabla_m^D\,\nabla^d\,(x_t)\\ =c+(1-\theta_1 B-\cdots-\theta_q B^q)(1-\Theta_1 B^m-\cdots-\Theta_q B^{Qm})\,\epsilon_t\end{aligned}$$

Cuando se aplican tanto diferencias estacionales como diferencias ordinarias, el orden de aplicación de las mismas no altera la serie resultante. Sin embargo, para aquellos casos en los que la serie posee un importante patrón estacional, es preciso estudiar en primer lugar la serie de diferencias estacionales, dado que puede ocurrir que la serie resultante ya sea estacionaria y no sea necesario aplicar diferenciación clásica. En cambio, si se realiza en primer lugar una transformación por diferencias consecutivas y la serie original presentaba un comportamiento estacional, la serie resultante no será estacionaria.

Se puede observar que los modelos SARIMA, por cómo se han construido, son capaces de modelar una única estacionalidad de la serie. Este hecho limita su rango de aplicación práctica a series temporales que presenten un único patrón en su registro histórico.

5.2.2. Modelo ARIMA con series de Fourier

El matemático y físico francés Jean-Baptiste Joseph Fourier, demostró a principios del siglo XIX que toda función periódica puede representarse como suma de funciones sinusoidales de distinta amplitud y frecuencia. Este hecho permite extender el estudio de los modelos SARIMA al caso de múltiples estacionalidades, modelizando la estacionalidad de la serie mediante combinaciones lineales de variables exógenas construidas a partir de funciones armónicas con distintas frecuencias. De esta forma, dejando a las componentes sinusoidales la modelización de la estacionalidad, podemos centrar el modelado de la serie obtenida tras eliminar la estacionalidad a la serie original con un modelo ARIMA.

Para la modelización de múltiples estacionalidades siguiendo esta técnica, basta con fijar el número de patrones estacionales que posee la serie (M), el periodo de cada una de las estacionalidades (m_i para cada $i=1,...,M$), y fijar el número de términos de Fourier (pares de funciones seno y coseno con frecuencias incrementales) a incluir para cada una de las estacionalidades (K_i para cada $i=1,...,M$). De este modo, la estacionalidad puede modelarse mediante la expresión

$$y_t = a + \sum_{i=1}^{M} \left[\sum_{k=1}^{K_i} \left[\alpha_{i,k}\, sin\Big(\frac{2\,k\,\pi}{m_i}\, t\Big) + \beta_{i,k}\, cos\Big(\frac{2\,k\,\pi}{m_i}\, t\Big) \right] \right]$$

Para definir el número de términos de Fourier a utilizar para cada uno de las estacionalidades de la serie (los K_i para cada $i = 1, ..., M$) es necesario estudiar previamente dos fenómenos matemáticos ligados al concepto de aproximación funcional discreta, para los que se mantiene la terminología inglesa dada su extensión en la literatura: ***sampling*** (o muestreo) y ***aliasing***.

Cuando se limita el estudio de una función únicamente sobre un determinado conjunto de puntos, esta función pasa a ser interpretada exclusivamente como un vector, de tal forma que los valores de la función en los puntos que no pertenezcan al conjunto a estudio pasan a ser ignorados. Este técnica de restricción de una función a un conjunto de puntos se conoce como *sampling* o muestreo, y como consecuencia inmediata de él se encuentra el fenómeno de *aliasing*. El *aliasing* es el efecto causado cuando funciones distintas se vuelven indistinguibles sobre los puntos de una muestra, independientemente de lo ocurrido fuera de ella.

Estos fenómenos ocurren con frecuencia al trabajar con funciones sinusoidales. Si consideramos una red equiespaciada de N puntos en el intervalo $[0, L]$, de forma que cada punto se puede escribir como $t_j = j \cdot h$, con $j = 0, ..., N-1$, entonces las funciones seno y coseno verifican las dos siguientes propiedades:

- Si N es par
  - $cos\left(\frac{2\pi}{L}\left(\frac{N}{2} - k\right) t_j\right) = cos\left(\frac{2\pi}{L}\left(\frac{N}{2} + k\right) t_j\right)$
  - $sin\left(\frac{2\pi}{L}\left(\frac{N}{2} - k\right) t_j\right) = -sin\left(\frac{2\pi}{L}\left(\frac{N}{2} + k\right) t_j\right)$
- Si N es impar
  - $cos\left(\frac{2\pi}{L}\left(\frac{N-1}{2} - k\right) t_j\right) = cos\left(\frac{2\pi}{L}\left(\frac{N+1}{2} + k\right) t_j\right)$
  - $sin\left(\frac{2\pi}{L}\left(\frac{N-1}{2} - k\right) t_j\right) = -sin\left(\frac{2\pi}{L}\left(\frac{N+1}{2} + k\right) t_j\right)$

En la Figura 5.4 se puede observar un ejemplo del fenómeno de aliasing, en el cual sobre el conjunto de puntos $\{0, 1, 2, 3, 4, 5\}$ las funciones $sin(\frac{\pi}{3}t)$, $-sin(\frac{5\pi}{3}\,t)$ y $sin(\frac{7\pi}{3}\,t)$ coinciden.

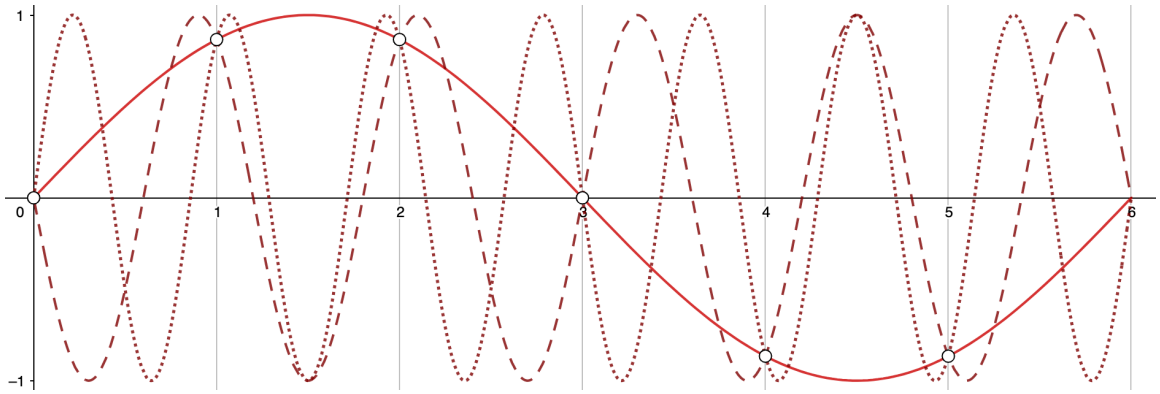


FIGURA 5.4: FENÓMENO DE ALIASING SOBRE UNA RED DE 6 PUNTOS EQUIESPACIADOS EN EL INTERVALO $[0, 6]$

De estos fenómenos se deduce que para la modelización de cada una de las estacionalidades, no será necesario hacer uso de un número de términos de Fourier superior a la mitad del periodo de estacionalidad, dado que las propiedades anteriores muestran que de no ser así se estaría produciendo *aliasing* y las nuevas variables exógenas no aportarían nada al resultado. Por ello la modelización de la estacionalidad con términos de Fourier junto con el modelado de la serie restante con un modelo ARIMA puede representarse en la forma

$$x_t = a + \sum_{i=1}^{M} \left[\sum_{k=1}^{\lfloor m_i/2 \rfloor} \left[\alpha_{i,k}\, sin\Big(\frac{2\,k\,\pi}{m_i}\,t\Big) + \beta_{i,k}\, cos\Big(\frac{2\,k\,\pi}{m_i}\,t\Big) \right] \right] + N_t \tag{5.2}$$

donde N_t representa el resultado del modelo ARIMA, y $\lfloor m_i/2 \rfloor = max\,\{k \in \mathbb{Z} \,/\, k \leq m_i/2\}$, abarcando así los casos de periodos de estacionalidad par e impar.

Entre las ventajas de este modelo híbrido destacan, como es evidente, la posibilidad de modelar múltiples estacionalidades, así como la facilidad de modelar grandes periodos de estacionalidad. Esta segunda ventaja se hace notable en relación al rendimiento de las implementaciones de los modelos SARIMA, dado que en la práctica el manejo de modelos SARIMA con periodos estacionales por encima de las 200 unidades de tiempo deriva en problemas de memoria.

Como inconveniente de este enfoque resalta el hecho de la rigidez en el modelado de la estacionalidad, dado que se toma una estructura constante y fija de los periodos y de la forma del ciclo, en contraposición a los modelos SARIMA en los cuales las modificaciones en los efectos de la estacionariedad sí se contemplan, dado que el comportamiento se calcula con respecto a lo sucedido en estaciones previas. Sin embargo, en muchos casos esta rigidez en la estacionalidad en un hecho de la serie, manteniéndose constante y sin grandes alteraciones. En aquellos escenarios en los que la estacionalidad de la serie a estudio no sufre notables cambios con el paso del tiempo este inconveniente del modelo híbrido queda en un segundo plano.

5.3. Modelo TBATS

En 2010, los profesores de Economía y Estadística Empresarial de la Universidad de Monash, Alysha M. De Livera, Rob J. Hyndman y Ralph D. Snyder (2011), desarrollaron un enfoque alternativo a los modelos ARIMA, que denominaron **TBATS** (de sus siglas en inglés, *Trigonometric seasonality, Box-Cox transformation, ARMA errors, Trend and Seasonal components*), orientados al modelado de series temporales univariantes con patrones estacionales complejos.

Los modelos TBATS son una extensión de los **modelos de alisado exponencial** propuestos a finales de la década de los cincuenta. Los modelos de predicción de alisado exponencial utilizan promedios ponderados de observaciones pasadas, cuyos pesos decaen de forma exponencial a medida que las observaciones son más lejanas. En otras palabras, cuanto más próxima es la observación, mayor es el peso asociado. Uno de los métodos de alisado exponencial orientado a estacionalidades más importante, y que además el modelo TBATS toma como referente en su construcción, es el método estacional de Holt-Winters (1957, 1960). La versión aditiva del modelo se representa como

$$\begin{aligned} y_t &= l_{t-1} + b_{t-1} + s_{t-m} + d_t \\ l_t &= l_{t-1} + b_{t-1} + \alpha\, d_t \\ b_t &= b_{t-1} + \beta\, d_t \\ s_t &= s_{t-m} + \gamma\, d_t \end{aligned}$$

donde m denota el periodo de estacionalidad y d_t es una variable de ruido blanco que representa el error de predicción. Por otro lado, las componentes l_t, b_t y s_t representan el nivel (valor medio de las observaciones de la serie), la tendencia y la estacionalidad de la serie, respectivamente. Los coeficientes α, β y γ representan los llamados parámetros de alisado, y l_0, b_0 y $\{s_{1-m}, ..., s_0\}$ son las variables de estado inicial.

El modelo TBATS añade a la estructura del modelo estacional de Holt-Winters tres nuevas componentes: la transformación de Box-Cox, correcciones de errores con modelos ARMA, y el modelado estacional mediante representaciones de Fourier:

$$\begin{aligned} y_t^{(\lambda)} &= l_{t-1} + \varphi\, b_{t-1} + \sum_{i=1}^{M} s_{t-m_i}^{(i)} + d_t \\ l_t &= l_{t-1} + \varphi\, b_{t-1} + \alpha d_t \\ b_t &= (1-\varphi)\, b + \varphi\, b_{t-1} + \beta d_t \\ d_t &= \sum_{i=1}^{p} \phi_i\, d_{t-i} - \sum_{i=1}^{q} \theta_i\, \epsilon_{t-i} + \epsilon_t \end{aligned}$$

y la componente estacional

$$
\begin{aligned}
s_t^{(i)} &= \sum_{j=1}^{K_i} s_{j,t}^{(i)} \\
s_{j,t}^{(i)} &= s_{j,t-1}^{(i)} cos\Big(\frac{2\,\pi\,j}{m_i}\Big) + s_{j,t-1}^{*(i)} sin\Big(\frac{2\,\pi\,j}{m_i}\Big) + \gamma_1^{(i)}\,d_t \\
s_{j,t}^{*(i)} &= -s_{j,t-1}^{(i)} sin\Big(\frac{2\,\pi\,j}{m_i}\Big) + s_{j,t-1}^{*(i)} cos\Big(\frac{2\,\pi\,j}{m_i}\Big) + \gamma_2^{(i)}\,d_t
\end{aligned}
$$

donde $m_1, ..., m_M$ denotan los periodos estacionales, l_t representa el nivel local, b y b_t son respectivamente las tendencias a largo y corto plazo, s_t^i representa la i-ésima componente estacional, d_t denota un proceso ARMA(p,q) con coeficientes ϕ_i para $i = 1, ..., p$ y θ_j para $j = 1, ..., q$, y ϵ_t es un proceso de ruido blanco con distribución normal de media 0 y varianza σ_ϵ^2 constante. Se tienen además los parámetros de alisado α, β, γ_i, $\gamma_1^{(i)}$ y $\gamma_2^{(i)}$, para $i = 1, ..., M$, y un parámetro de amortiguamiento (*damping*) de la tendencia, φ.

El parámetro λ denota el coeficiente de la transformación de Box-Cox, la cual se utiliza principalmente para estabilizar la varianza y asemejar más los datos a una distribución normal. La transformación de Box-Cox se define como una función continua con respecto al parámetro λ, definida por partes para mantener la continuidad en el punto de singularidad ($\lambda = 0$)

$$
y_t^{(\lambda)} = \begin{cases} \dfrac{y_t^\lambda - 1}{\lambda}, & \lambda \neq 0 \\ log(y_t), & \lambda = 0 \end{cases}
$$

El número de términos de Fourier requeridos para modelar la i-ésima componente estacional se denota por K_i, y de forma análoga al caso del modelado directo con términos de Fourier (Sección 5.2.2), su valor deberá ser inferior a $m_i/2$ si m_i es par o bien $(m_i - 1)/2$ si es impar, con el fin de evitar el fenómeno de *aliasing*. En la práctica, sin embargo, el número de términos que se requieren en la mayoría de componentes estacionales suele ser menor, lo que reduce el número de parámetros a estimar.

Para dar más detalles sobre su estructura, a la hora de denotar el modelo TBATS se incluyen los parámetros que configuran el modelo, de tal forma que se hace referencia a ellos con la notación TBATS($\lambda, \varphi, p, q, \{m_1, K_1\}, \{m_2, K_2\}, ..., \{m_M, K_M\}$).

Una de las principales ventajas reseñables, además de la posibilidad de modelar múltiples estacionalidades, es la capacidad de modelar los efectos de la estacionalidad con periodos no enteros. Por ejemplo, dada una serie con datos diarios, es posible modelar años bisiestos tomando como periodo $m = 365{,}25$.

Estimación de coeficientes

Para la estimación de los coeficientes se utiliza de nuevo la estimación por máxima verosimilitud condicionada. Dada la serie temporal de observaciones $y = (y_1, ..., y_N)$, la función de verosimilitud condicionada se deriva del hecho de suponer los errores $\epsilon_t \sim N(0, \sigma_\epsilon^2)$. Si denotamos por $\mathfrak{v} = (\tilde{\lambda}, \tilde{\varphi}, \tilde{\alpha}, \tilde{\beta}, \tilde{\gamma}_1^1, \tilde{\gamma}_2^1, ..., \tilde{\gamma}_1^M, \tilde{\gamma}_2^M, \tilde{\phi}_1, ..., \tilde{\phi}_p, \tilde{\theta}_1, ..., \tilde{\theta}_q)$ al vector de coeficientes, calculando el estimador máximo verosímil de la varianza y tomando la función de verosimilitud de la distribución conjunta de las variables aleatorias que componen el proceso estocástico a estudio, los estimadores de máxima verosimilitud son aquellos valores de $\mathfrak{v}$ que maximizan la función de log-verosimilitud (aplicación de la función logarítmica sobre la función de verosimilitud)

$$\mathscr{L}_c(\mathfrak{v}) = 2(\lambda - 1)\sum_{t=1}^{N} log(y_t) - N\, log\left(\sum_{t=1}^{N} \epsilon_t^2\right)$$

condicionada a los valores iniciales $x_0 = (l_0, b_0, s_0^1, ..., s_0^{(N)}, d_0, d_{-1}, ..., d_{1-p}, \epsilon_0, \epsilon_{-1}, ..., \epsilon_{1-q})$ donde $s_0^{(i)}$ es el vector $s_0^{(i)} = (s_{1,0}^{(i)}, s_{2,0}^{(i)}, ..., s_{K_i,0}^{(i)}, s_{1,0}^{*(i)}, s_{2,0}^{*(i)}, ..., s_{K_i,0}^{*(i)})$ (véase De Livera y col. (2011) para más información sobre cómo obtener la función de verosimilitud y de log-verosimilitud). Nótese que la aparición de subíndices negativos se debe únicamente al criterio de indexación, dado que para este caso se utilizan índices menores o iguales a cero para indicar los coeficientes previos que son necesarios para poder comenzar la estimación del resto de parámetros.

Criterios de selección de modelos

De nuevo para la selección del modelo óptimo se hace uso del Criterio de Información de Akaike, tomando el modelo que con menor valor $AIC = 2k - 2\, log(L)$, donde $log(L)$ es el máximo valor de la función de verosimilitud, esto es, el valor de la función evaluada en los estimadores de máxima verosimilitud; y k es el número total de parámetros de $\mathfrak{v}$ junto con el número de grados de libertad del vector de estados iniciales x_0. También se pueden utilizar criterios alternativos como el AICc o el BIC. Por otro lado, para la selección de los parámetros p y q del modelo ARMA, se siguen los dos siguientes pasos:

- En primer lugar se supone un modelo TBATS sin componente ARMA. Tras ajustar este modelo, se incorpora un modelo ARMA sobre la serie de errores, utilizando el procedimiento de estimación y selección del mejor modelo siguiendo el procedimiento estándar para modelos ARIMA (Sección 5.2), suponiendo la serie de errores estacionaria ($d = 0$).
- En segundo lugar, tras haber determinado los valores de p y q, se vuelve a estimar los coeficientes del modelo TBATS, incluyendo ahora sí los coeficientes del modelo ARMA(p,q) junto con el resto de parámetros. La componente ARMA se retiene únicamente si el resultado del segundo modelo posee un AIC inferior al modelo previo sin componente ARMA.

5.4. Redes Neuronales Recurrentes

Los modelos clásicos de predicción de series temporales estudiados hasta el momento se basan en el análisis de las series, estudiando las componentes de tendencia y estacionalidad existentes. Sin embargo, la aplicación práctica de estos modelos en estudios reales no permite modelar la influencia del factor humano y de factores externos que de forma directa influyen en el comportamiento de la serie. Estas influencias producen relaciones no lineales en las observaciones a las cuales los modelos clásicos no se pueden adaptar. Este hecho ha derivado en el crecimiento del interés en utilizar modelos de inteligencia artificial, destacando en este contexto la aplicación de Redes Neuronales Recurrentes, sobre series temporales.

Las Redes Neuronales Recurrentes o **RNN** (de sus siglas en inglés *Recurrent Neural Network*), propuestas a lo largo de la década de los ochenta (Rumelhart y col., 1986; Elman, 1990; Werbos, 1988) constituyen un tipo específico de red neuronal artificial, enmarcada dentro del campo del aprendizaje profundo o *deep learning* (por su término anglosajón), cuyo paradigma de aprendizaje es el aprendizaje supervisado.

El aprendizaje profundo, incluido dentro de una familia más amplia de modelos de aprendizaje automático, busca emular la forma de aprendizaje de los seres humanos tratando para ello de imitar el funcionamiento del cerebro humano. En particular en el caso de las RNN, es posible establecer una asociación o analogía con el lóbulo temporal del cerebro humano, el cuál interviene en el proceso de reconocimiento, procesando información y ayudando a pasar de memoria a corto plazo a memoria a largo plazo, evitando así que la información se pierda a lo largo del tiempo.

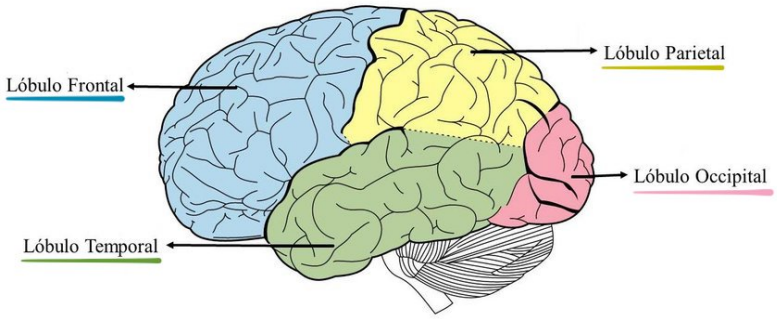


Figura 5.5: Lóbulos del cerebro

Los humanos no comienzan cada segundo a pensar desde cero. Al leer por ejemplo estas frases de este proyecto, es posible comprender cada palabra basándose en la comprensión de las palabras previas. Las redes neuronales tradicionales no permiten hacer esto, lo que puede suponer un gran problema. Basta con imaginarse por ejemplo la clasificación de los tipos de eventos ocurridos a

lo largo de una película. Una red neuronal tradicional no puede utilizar el conocimiento sobre los eventos previos para ayudarse con las clasificación de los eventos posteriores.

Las redes neuronales recurrentes abordan el problema de persistencia, para lo cual se constituyen como redes neuronales artificiales cuyas conexiones entre nodos forman un grafo dirigido a lo largo de una secuencia temporal, permitiendo modelar un comportamiento temporal dinámico. Las RNN alimentan las entradas a la red con información del instante temporal previo, formando bucles o ciclos sobre ellas, lo que permite así que la información persista a lo largo del tiempo. La Figura 5.6 muestra en la parte izquierda la representación gráfica de una RNN, mostrando el bucle de retroalimentación de la red. En el instante t la red toma como entrada el valor x_t y la retroalimentación del paso $t-1$, y devuelve una salida, h_t.

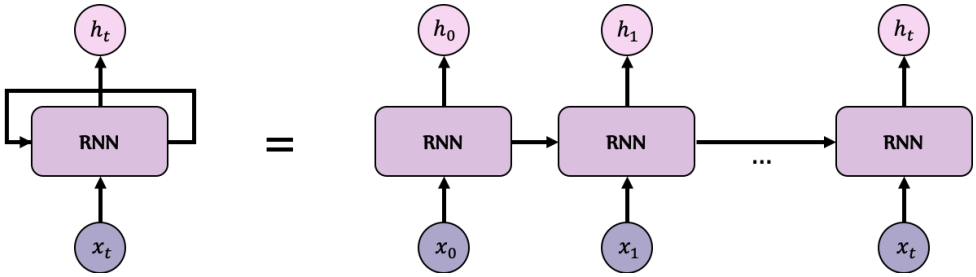


FIGURA 5.6: REPRESENTACIÓN GRÁFICA DE UNA RNN Y SU REPRESENTACIÓN "DESPLEGADA" EN EL TIEMPO

La parte derecha de la igualdad de la Figura 5.6 muestra el "despliegue" (*unrolled*) de la red para cada uno de los instantes de tiempo. En esta representación es importante recalcar que la RNN observada en cada uno de los instantes de tiempo es siempre la misma, únicamente se observan copias de ella misma. Este hecho es de vital importancia de cara más adelante en la compresión de la estructura interna de la red y del proceso de entrenamiento. Este enfoque revela la capacidad de estas redes tanto de procesar como de producir secuencias de datos, lo que las hace idóneas para su uso con series temporales.

Las redes neuronales en el contexto de la predicción de series temporales, permiten trabajar con múltiples variables, obteniendo así **modelos de predicción de series temporales multivariantes**.

Además de los beneficios generales que pueden aportar las redes neuronales como modelos de predicción de secuencias, las redes neuronales recurrentes también pueden aprender y aprovechar la dependencia temporal de los datos. En el caso más simple, la red toma como entrada en cada instante de tiempo una observación de una secuencia de la serie temporal, y puede aprender cuales de las observaciones previas de la secuencia son relevantes y de qué forma son relevantes. Las RNN tratarán de aprender la dependencia temporal y la información contextual de los datos de entrada.

Bajo el supuesto de predicción de secuencias, existen tres modelos arquitectónicos de construcción de las redes neuronales recurrentes: uno-a-muchos, muchos-a-uno y muchos-a-muchos.

- El modelo **uno a muchos** (*one to many*) produce múltiples salidas $[h_t, h_{t+1}, ..., h_{t+q}]$ a partir de una única entrada $[x_t]$ (véase Figura 5.7).

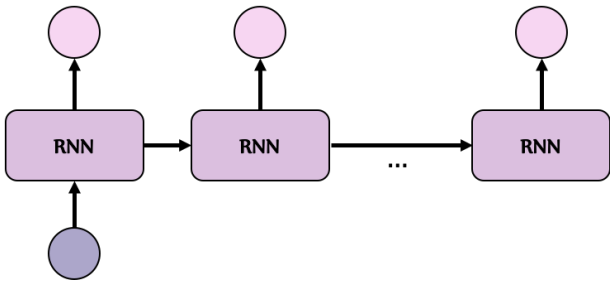


Figura 5.7: Modelo de predicción uno-a-muchos con redes neuronales recurrentes

El estado interno se acumula a medida que se van generando cada uno de los valores de la secuencia de salida. Este modelo es apropiado para aquellos problemas de predicción en los que se desea producir una secuencia de salida para cada una de las entradas en instantes de tiempo consecutivos.

- El modelo **muchos a uno** (*many to one*) produce una única salida $[h_t]$ tras recibir una secuencia de entrada múltiple $[x_{t-p}..., x_{t-1}, x_t]$ (véase Figura 5.8).

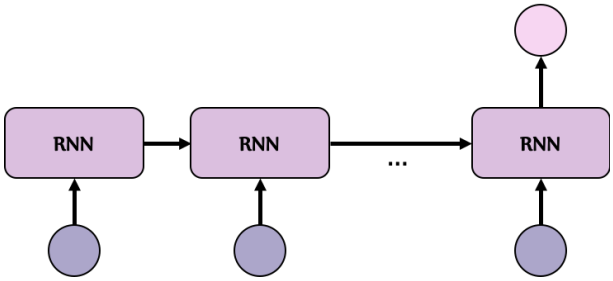


Figura 5.8: Modelo de predicción muchos-a-uno con redes neuronales recurrentes

El estado interno se acumula con cada valor entrante hasta producir un valor final. Este modelo es apropiado para aquellos problemas de predicción de secuencias en los que se requiere una secuencia con múltiples observaciones para llevar a cabo la predicción del siguiente valor de la secuencia. Otro ejemplo de aplicación de esta arquitectura es la clasificación de series temporales, consistente en asignar una etiqueta a cada serie o secuencia temporal.

- El modelo **muchos a muchos** (*many to many*) produce una secuencia con múltiple salidas $[h_t, h_{t+1}, ..., h_{t+q}]$ tras recibir una entrada múltiple $[x_{t-p}..., x_{t-1}, x_t]$ (véase Figura 5.9).

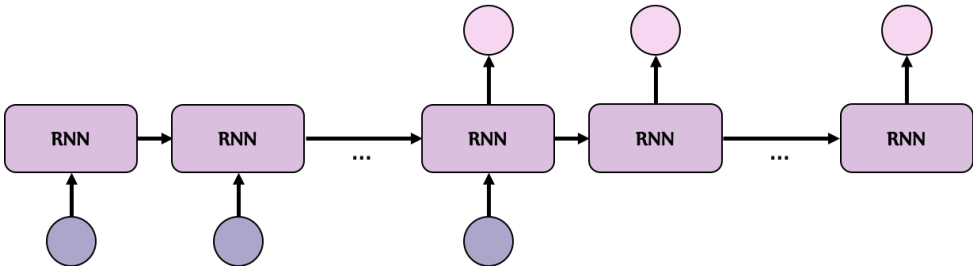


FIGURA 5.9: MODELO DE PREDICCIÓN MUCHOS-A-MUCHOS CON REDES NEURONALES RECURRENTES

  Al igual que en el caso del modelo muchos-a-uno, el estado se acumula hasta obtener la primera salida, sin embargo en este caso el estado interno sigue acumulándose a medida que se lleva acabo la predicción de la secuencia de salida. Es importante remarcar que el número de elementos de la secuencia de entrada no tiene porqué coincidir con el número de elementos de la secuencia de salida. Es modelo es apropiado cuando se pretende realizar predicciones de secuencias tomando como secuencia de entrada las observaciones previas. Este modelo en el ámbito de las series temporales se denomina también modelo *seq2seq* (*sequence to sequence*).

Aunque en principio puede parecer que las redes neuronales recurrentes poseen una estructura sencilla y destacan por su gran potencial de análisis de dependencias temporales, en la práctica desafortunadamente estas redes son difíciles de entrenar correctamente. Entre las principales razones que dificultan la puesta en práctica de este modelo destacan los problemas de desvanecimiento de gradiente (*vanishing gradient*) y explosión de gradiente (*exploding gradient*) (Bengio, Simard & Frasconi, 1994). Antes de entrar en detalles del proceso de entrenamiento de una RNN y de los problemas derivados de él, es preciso introducir la terminología común a cualquier implementación de red neuronal recurrente, asociada a la estructura interna de la red.

Estructura interna de una RNN

Una red neuronal recurrente genérica con entrada $x_t \in \mathbb{R}^m$ y **estado interno** $h_t \in \mathbb{R}^n$ para el instante de tiempo t, viene descrita por la ecuación recurrente

$$h_t = f\left(W_{rec}\, h_{t-1} + W_{in}\, x_t + b\right) \tag{5.3}$$

donde $W_{rec} \in \mathbb{R}^{n\times n}$ representa la matriz de pesos asociados al vector recurrente que actúa como estado interno de la red, $W_{in} \in \mathbb{R}^{n\times m}$ representa la matriz de pesos asociados al vector de

entrada, y $b \in \mathbb{R}^n$ representa el vector de sesgos (en inglés *bias*). La ecuación 5.3 deriva de la estructura interna de la red neuronal recurrente genérica que, tal y como puede observarse en las Figuras 5.10 y 5.11, se compone de un simple red neuronal interna con una capa de entrada con las $n + m$ entradas obtenidas de la concatenación de los vectores x_t y h_{t-1}, y una capa de salida con las n salidas que conformarán el nuevo estado interno h_t, utilizando como función de activación una función genérica f.

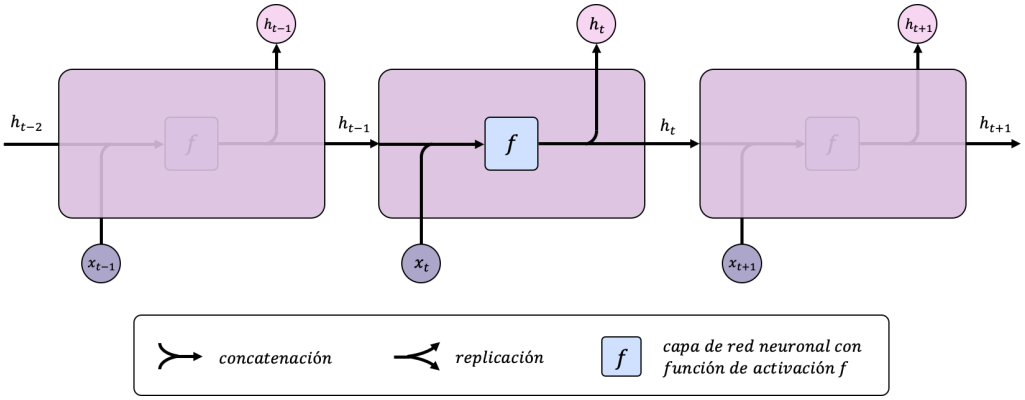
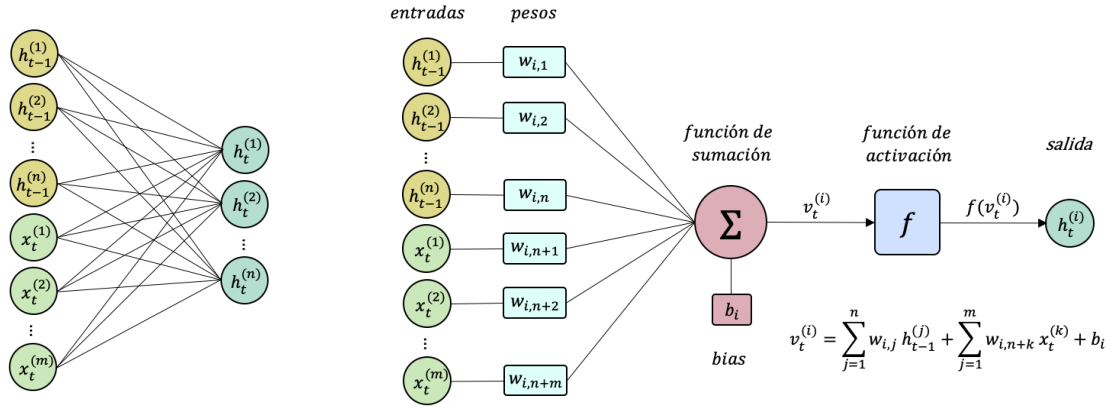


Figura 5.10: Estructura interna de una RNN genérica

Tomando como matriz de pesos de la red neuronal interna la matriz obtenida de la concatenación de las matrices W_{rec} y W_{in}, esto es, $W = [W_{rec}, W_{in}] \in \mathbb{R}^{n\times(n+m)}$, la salida de la red se representa por la ecuación

$$h_t = f\left(W\left[h_{t-1}, x_t\right] + b\right) \tag{5.4}$$



(A) (B)

Figura 5.11: Representación gráfica de (A) la red neuronal interna que constituye la RNN, y (B) la estructura detallada de las neuronas que componen la red interna

Las **funciones de activación** de un nodo de una red neuronal son funciones matemáticas que definen la salida del nodo dada una determinada entrada. En el caso más simple, la función de activación determina si la neurona debe ser activada (tomando la salida el valor 1), o no (tomando en este otro caso el valor 0), basándose en si la entrada de cada neurona es relevante para la predicción del modelo. En otros contextos las funciones ayudan a normalizar la salida de cada neurona, tomando un rango entre 0 y 1 o entre -1 y 1. En la Tabla 5.1 se muestran las funciones de activación que serán relevantes en este proyecto.

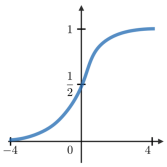
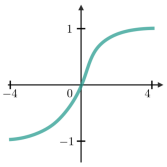
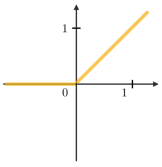
Función de activación	Ecuación	Rango	Gráfica
Sigmoide	$\sigma(z) = \frac{1}{1+e^{-z}}$	$(0,1)$	
Tangente hiperbólica	$tanh(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$	$(-1,1)$	
ReLU (Rectified Linear Unit)	$\phi(z) = max\{0,z\}$	$[0,+\infty)$	

TABLA 5.1: FUNCIONES DE ACTIVACIÓN

De la Figura 5.10 se observa que la salida obtenida con una RNN genérica se corresponde realmente con el estado oculto devuelto por la red. Sin embargo, de forma general, el tamaño del estado oculto es independiente del tamaño del vector de predicciones de salida que se desea obtener. El número de elementos (también llamados unidades) que constituyen el estado oculto es una característica a configurar en la construcción de la red, y su valor se encuentra relacionado con el tamaño de la memoria necesaria para modelar las dependencias y patrones temporales de la serie. Nótese por tanto que, aun existiendo una posible relación entre el tamaño de la memoria y el número de variables a predecir, ambas características no necesariamente coinciden. Por este motivo, la arquitectura básica de toda RNN requiere la utilización de una capa de salida que transforme el vector de estado interno en el vector de predicciones deseado. Surge así la necesidad de añadir una capa densa con entrada en cada instante de tiempo t el estado interno $h_t \in \mathbb{R}^n$, y salida el vector de predicciones con dimensión el número de variables a predecir, $y_t \in \mathbb{R}^l$. Las **capas densas**, también llamadas completamente conectadas, unen todos los elementos de

entrada con los elementos de salida, permitiendo así transformaciones y cambios de dimensión entre los vectores de entrada y salida de la red.

En la Figura 5.12 se muestra la adición de la capa densa a la salida de la red, con función de activación genérica g. Con esta nueva estructura, el resultado de aplicación de la red neuronal viene dado por las ecuaciones

$$\begin{aligned} h_t &= f\left(W_h\left[h_{t-1}, x_t\right] + b_h\right) \\ y_t &= g\left(W_y\, h_t + b_h\right) \end{aligned} \tag{5.5}$$

donde $W_h \in \mathbb{R}^{n\times(n+m)}$ y $b_h \in \mathbb{R}^n$ son respectivamente la matriz de pesos y el vector de sesgos de la capa interna que da como resultado el vector $h_t \in \mathbb{R}^n$ de estado oculto, y $W_y \in \mathbb{R}^{n\times l}$ y $b_y \in \mathbb{R}^l$ son respectivamente la matriz de pesos y el sesgo de la capa densa de salida que da como resultado el vector $y_t \in \mathbb{R}^l$.

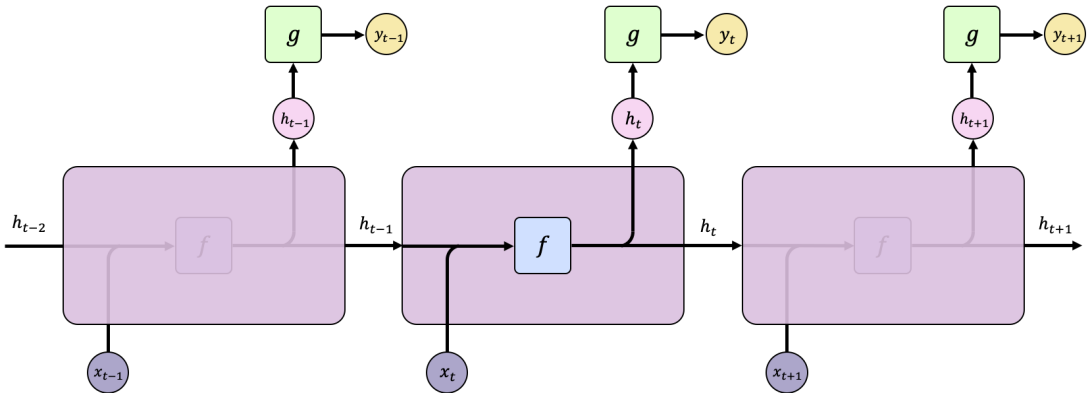


Figura 5.12: Estructura interna de una RNN genérica con capa densa de salida

Observando detalladamente las ecuaciones 5.5 se observa el concepto de recurrencia y la memoria asociada a las redes recurrentes. La salida y_t depende del estado interno actual h_t, pero este a su vez depende no solo de la entrada actual x_t, sino también del estado interno previo, h_{t-1}. Esta estructura es el que aporta la memoria a la red, permitiendo conservar y compartir la información entre instantes de tiempo.

Número de variables y tamaño de las secuencias de entrada y salida

Nótese que en la notación de las figuras previas no debe confundirse la cardinalidad de las entradas y salidas con los tamaños de las secuencias. Dado que el estudio de las RNN se centra en el contexto de aplicación sobre series temporales, el tamaño de la entrada dependerá de si la serie temporal a estudio es una serie univariante o una serie multivariante. En el caso general, en

cada instante de tiempo t, la entrada se interpretará como un vector $x_t \in \mathbb{R}^m$ cuyos elementos se corresponden con las observaciones de cada una de las variables a estudio. Por su parte, la salida estará constituida por un vector $y_t \in \mathbb{R}^l$ compuesto por las predicciones de las l variables de las que se desea conocer sus valores futuros. Sin embargo, el número de variables l de salida no tiene porqué coincidir con el número de variables de entrada. Considérese como ejemplo el uso de la temperatura y la presión atmosférica para realizar predicciones de la temperatura. En este caso, para cada instante de tiempo los valores de entrada de la red estarán constituidos por vectores de dos elementos, $x_t = [x_t^{(1)}, x_t^{(2)}] \in \mathbb{R}^2$, donde $x_t^{(1)}$ y $x_t^{(2)}$ representan respectivamente la temperatura y la presión en el instante de tiempo t. Por su parte, la salida $y_t \in \mathbb{R}$ representará la predicción de temperatura.

Independientemente del cardinal de los vectores x_t e y_t se tiene el concepto de **tamaño de secuencia**, que hace referencia al número de instantes de tiempo utilizados en la entrada como histórico para hacer la predicción (que de forma general denotaremos por p), y en la salida como número de predicciones a considerar (denotado por q). Volviendo al ejemplo previo, de forma independiente a los tamaños de entrada y salida considerados, deberán fijarse los tamaños de las secuencias de entrada y salida. Puede considerarse el uso de una secuencia de entrada de 10 periodos de tiempo previos al instante a estudio, y una predicción de 5 valores de temperatura, obteniendo así finalmente un modelo *seq2seq* de predicción multivariante.

A mayores, estos conceptos previos son de nuevo independientes con respecto al tamaño del vector interno de la RNN, tal y como ya se ha mencionado previamente. Por este motivo ha de prestarse especial atención en el momento de construir la RNN dado que en su definición intervienen diferentes conceptos de dimensionalidad que pueden generar confusión.

Entrenamiento de las RNN

Los parámetros que componen la red (las matrices de pesos W_{rec}, W_{in} y W_y, y los vectores de sesgos b_h y b_y de las ecuaciones 5.5) se ajustan mediante el proceso de entrenamiento de la red, con el objetivo de minimizar los errores de predicción.

Existen diversos algoritmos para el entrenamiento de los pesos de las RNN. La gran mayoría de los métodos existentes basan su fundamentación en la técnica de descenso por gradientes[2] (véase Qian, 1999). Para encontrar un mínimo local de una función diferenciable f siguiendo una técnica basada en el descenso por gradientes, se deben tomar de forma iterativa valores del dominio de la

[2] El gradiente de un campo escalar f, denotado por ∇f, es un campo vectorial cuyas funciones coordenadas son las derivadas parciales del campo escalar, esto es, $\nabla f = \left(\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n}\right)$. El vector gradiente evaluado en un punto genérico del dominio de f, $\nabla f(x)$, indica la dirección en la cual el campo f varía más rápidamente, y su módulo representa la tasa de crecimiento en dicha dirección.

función proporcionales al opuesto del gradiente, esto es, tomando iterativamente $x = x - \alpha \cdot \nabla f(x)$ donde α representa una constante denominada *learning rate* o tasa de aprendizaje.

Para el entrenamiento de la red, el proceso constará de dos partes. En primer lugar, se aplicará un algoritmo para calcular los gradientes de la función de pérdida de la red, y posteriormente se utilizará un algoritmo de optimización basado en el descenso por gradientes para estimar los coeficientes que reducen el error de predicción.

Para el desarrollo de esta sección se considera la versión más simple de una RNN, la cual en cada instante de tiempo toma una entrada y genera una salida (el equivalente a un modelo de predicción uno-a-uno que, si bien no ha sido introducido en el inicio de la sección dado que no es práctico en el contexto de la predicción de series temporales, resultará de utilidad en la comprensión del proceso de entrenamiento de las RNN).

Uno de los algoritmo más extendidos en redes neuronales recurrentes para el cálculo del gradiente de la función de error es el algoritmo de retropropagación a través del tiempo, más conocido como **BPTT** (por sus siglas en inglés, Backpropagation Through Time) (véase Werbos, 1990). Conceptualmente, el algoritmo BPTT trabaja sobre la red neuornal recurrente en su representación desplegada (véase la Figura 5.13).

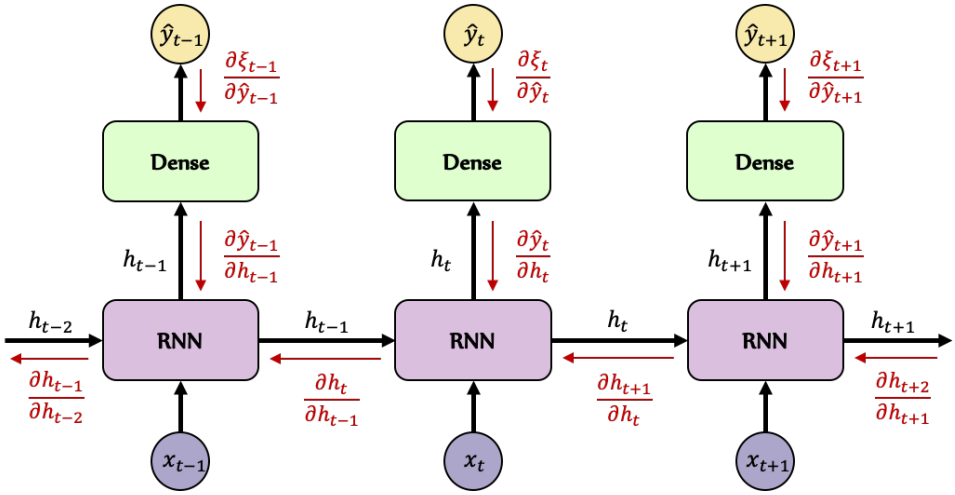


Figura 5.13: Red neuronal recurrente desplegada en el tiempo con propagación de errores

Para el entrenamiento de la red básica considerada se debe suponer que se dispone de pares de valores entrada/salida. Si en cada instante de tiempo t denotamos por x_t la entrada, y_t la salida esperada e $\hat{y}_t$ la salida obtenida con la red neuronal, se debe definir una función de error que evalúe el grado de inexactitud cometido por la red en cada una de las predicciones, relacionando los valores y_t e $\hat{y}_t$. Si denotamos por $\xi_t(y_t, \hat{y}_t)$ la función de pérdida en el instante t, el error

acumulado hasta el instante de tiempo actual T será igual a la suma de los errores acumulados en los instantes de tiempo previos,

$$\xi(\theta) = \sum_{t=1}^{T} \xi_t(y_t, \hat{y}_t) \tag{5.6}$$

donde θ es un vector que agrupa todos los parámetros de la red (en el caso a estudio actual estaría formado por los pesos de la red W_{rec}, W_{in} y W_y y los vectores de sesgos b_{rec}, b_{in} y b_y). Se obtiene así que la función de error será un función dependiente de los parámetros que configuran la red en cada paso de entrenamiento. Durante el entrenamiento de la red el objetivo será encontrar la configuración de parámetros que minimiza el error $\xi(\theta)$.

El objetivo del algoritmo BPTT es el cálculo del gradiente $\nabla_\theta \xi = \frac{\partial \xi}{\partial \theta}$. Una vez la red neuronal recurrente se ha desplegado a lo largo del eje temporal, el proceso de cálculo del gradiente de la función de error es análogo al proceso seguido en el entrenamiento estándar de un perceptrón multicapa (MLP, *Multilayer Perceptron*). Nótese que una vez se ha desplegado la red recurrente ya no existen bucles en la retroalimentación de la red, si bien las distintas redes obtenidas son todas copias de una misma, y por lo tanto los parámetros a configurar son los mismos. Si embargo, a diferencia del proceso de retropropagación de errores de un MLP, en el caso de las RNN hay que considerar la dependencia temporal y la recurrencia en las ecuaciones que definen la red. Manteniendo la nueva terminología para denotar las salidas de la red, las dos ecuaciones que modelan la RNN básica a estudio son

$$\begin{aligned} h_t &= f\left(W_{rec}\, h_{t-1} + W_{in}\, x_t + b_h\right) \\ \hat{y}_t &= g\left(W_y\, h_t + b_h\right) \end{aligned} \tag{5.7}$$

Para el cálculo de los gradientes siguiendo el algoritmo numérico de cálculo BPTT, los errores deben calcularse teniendo en cuanta las dependencias recurrentes que conforman la red. Aplicando las reglas básicas que rigen la derivación de funciones compuestas, se obtiene

$$\begin{aligned} \nabla_\theta \xi = \frac{\partial \xi}{\partial \theta} &= \sum_{t=1}^{T} \frac{\partial \xi_t}{\partial \theta} \\ \frac{\partial \xi_t}{\partial \theta} &= \sum_{k=1}^{t} \left(\frac{\partial \xi_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial^+ h_k}{\partial W} \right) \\ \frac{\partial h_t}{\partial h_k} &= \prod_{i=k+1}^{t} \frac{\partial h_i}{\partial h_{i-1}} \end{aligned} \tag{5.8}$$

donde $\frac{\partial^+ h_k}{\partial W}$ denota la derivada parcial "inmediata" del estado oculto h_k con respecto al vector de parámetros θ, esto es, tomando h_{k-1} constante con respecto a θ. Haciendo uso de las ecuaciones 5.7, se tiene

$$\begin{aligned} \frac{\partial h_i}{\partial h_{i-1}} &= f'(W_{rec}\, h_{i-1} + W_{in} x_i + b_h) \cdot \frac{\partial}{\partial h_{t-1}} [W_{rec}\, h_{i-1} + W_{in} x_i + b_h] \\ &= f'(W_{rec}\, h_{i-1} + W_{in} x_i + b_h) \cdot W_{rec} \end{aligned} \tag{5.9}$$

Juntando las ecuaciones 5.8 y 5.9, se obtiene

$$\frac{\partial \xi_t}{\partial \theta} = \sum_{k=1}^{t} \left[\frac{\partial \xi_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial h_t} \left(\prod_{i=k+1}^{t} f'(W_{rec}\, h_{i-1} + W_{in} x_i + b_h) \cdot W_{rec} \right) \frac{\partial^+ h_k}{\partial W} \right] \tag{5.10}$$

Cada uno de los términos $\frac{\partial \xi_t}{\partial \theta}$ en los que se descompone el gradiente del error es suma de términos, denominados contribuciones temporales o componentes temporales. La contribución temporal k-ésima mide cómo afectan los parámetros θ en el instante k a la función de coste en el instante $t > k$. Es posible llevar a cabo una vaga distinción entre las contribuciones de corto y largo plazo en función del valor de k, hablando de largo plazo para las contribuciones tales que $k \ll t$, y corto plazo al resto.

Las funciones más comunes en la construcción de redes neuronales recurrentes son la función sigmoide y la tangente hiperbólica (recuérdese la Tabla 5.1). Si analizamos las derivadas de estas funciones, observamos que en ambos casos su rango se encuentra acotado a lo sumo por 1 (veáse la Tabla 5.2).

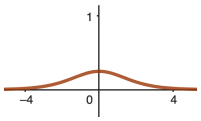
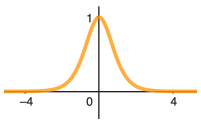
Función de activación	Derivada	Rango	Gráfica
Sigmoide	$\frac{d}{dz}\sigma(z) = \sigma(z)\big(1 - \sigma(z)\big)$	$(0, 1/4]$	
Tangente hiperbólica	$\frac{d}{dz} tanh(z) = 1 - tanh^2(z)$	$(0, 1]$	

Tabla 5.2: Derivadas de las funciones de activación

Además de estar ambas funciones acotadas por 1, sus valores a medida que se alejan del origen tienen rápido hacia 0. Este hecho trasladado a la evaluación de las derivadas de las funciones

de activación en cada una de las contribuciones temporales del gradiente del error, hace que las contribuciones temporales tiendan exponencialmente hacia 0 a medida que aumenta la distancia entre t y k. Este fenómeno, conocido como desvanecimiento del gradiente (***vanishing gradient*** por su término anglosajón), hace imposible a las RNN aprender correlaciones a largo plazo.

Por otro lado, dependiendo de la función de activación y de los parámetros de la red, es posible obtener un fenómeno contrario al desvanecimiento del gradiente. Si la matriz de pesos asociada al estado interno posee valores suficientemente grandes como para paliar el factor de multiplicación dado por la derivada de la función de activación, las contribuciones temporales de largo plazo pueden crecer exponencialmente, por encima de las contribuciones a corto plazo. Este fenómeno se conoce como explosión del gradiente (***exploding gradient*** por su término inglés), y su trato es más simple que en el caso del desvanecimiento (véase como ejemplo la solución aportada por Pascanu, Mikolov y Bengio, 2013).

5.4.1. LSTM

Las redes neuronales **LSTM** (acrónimo de su denominación anglosajona ***Long Short-Term Memory neural networks***) constituyen una tipo específico de red neuronal recurrente, propuesto por Sepp Hochreiter y Jürgen Schmidhuber (1997). Tal y como ya se ha comentado, teóricamente las RNN fueron concebidas para modelar las relaciones temporales de las secuencias de entrada. Sin embargo, en el entrenamiento de las RNN genéricas utilizando retropropagación, el cálculo de los gradientes puede derivar en los fenómenos de desvanecimiento y explosión, el primero de los cuales provoca una limitación en cuanto al modelado de las correlaciones a largo plazo. Tal y como se verá a lo largo de la sección, la estructura interna de la red LSTM está creada para eliminar el problema de desvanecimiento durante el proceso de entrenamiento, lo que aporta a estas redes un gran potencial en cuanto a la detección de patrones a largo plazo.

Las redes neuronales LSTM, al igual que las RNN, poseen también una estructura interna en cadena, pero sin embargo en esta ocasión la celda LSTM en lugar de disponer de una única capa de red neuronal como en el caso de la RNN genérica, posee cuatro capas de red neuronal interactuando de un modo específico (véase Figura 5.14).

Entre las diferencias existentes entre la red LSTM y la RNN genérica destaca la utilización de un nuevo estado interno, llamado **celda de estado**. La celda de estado actúa como una cinta transportadora de información, e interactúa con la red mediante operaciones concretas con el fin de preservar la información más relevante a largo plazo. En el instante t, la celda de estado está constituida por un vector $c_t \in \mathbb{R}^n$ cuya dimensión coincide con la dimensión del estado oculto $h_t \in \mathbb{R}^n$. El estado oculto también interviene en esta nueva estructura con el objetivo de modelar aquellos aspectos más relevantes relativos a la memoria a corto plazo de la red.

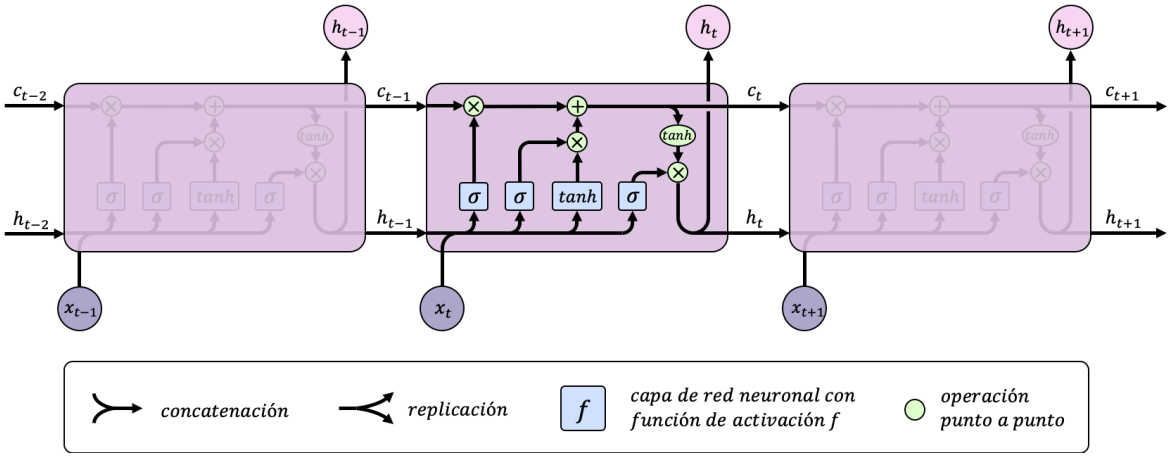


Figura 5.14: Estructura interna de una red LSTM

La red LSTM posee la capacidad de eliminar o añadir nueva información a la celda de estado, regulando estas modificaciones mediante una estructura denominada puerta (en inglés *gate*). Las puertas tienen como finalidad proteger y regular el intercambio de información con la celda de estado. La red LSTM consta de tres puertas:

- ***Forget gate*** o puerta de olvido: encargada de decidir qué detalles se deben descartar de la celda de estado. La capa asociada a esta puerta toma como entradas la concatenación de los valores del estado oculto previo (h_{t-1}) y el nuevo vector de observaciones (x_t), y haciendo uso de una función de activación sigmoide produce un vector con n valores, uno por cada elemento de la celda de memoria, comprendidos entre 0 (que significa que se debe eliminar por completo la correspondiente información de la celda de estado) y 1 (indicando que esa información debe permanecer inmutable) (véase la Figura 5.15).

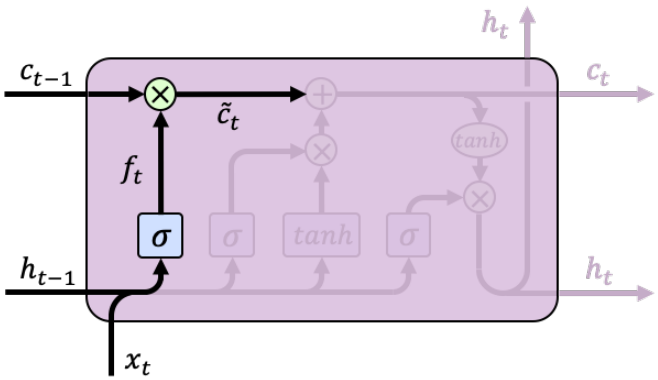


Figura 5.15: Forget gate

Para hacer efectivo el cambio en la celda de estado, se deberá multiplicar punto a punto[3] el vector c_{t-1} y el vector que determina el grado de olvido, f_t, obteniendo así una nueva celda de estado auxiliar ($\tilde{c}_t$) ya sin los valores descartados. Las ecuaciones que rigen este proceso son

$$\begin{aligned} f_t &= \sigma\left(W_f\left[h_{t-1}, x_t\right] + b_f\right) \\ \tilde{c}_t &= c_{t-1} * f_t \end{aligned} \tag{5.11}$$

donde $W_f \in \mathbb{R}^{n\times(n+m)}$ y $b_f \in \mathbb{R}^n$ representan la matriz de pesos y el vector de sesgos de la capa relativa a la *forget gate*.

- ***Input gate*** o puerta de entrada: decide qué valores de la entrada se utilizan para actualizar la memoria. La capa asociada a esta puerta toma de nuevo como entrada la concatenación de los vectores h_{t-1} y x_t y, tras pasar por una función sigmoide, devuelve un vector de n posiciones, una por cada elemento de la celda de estado, indicando las nuevas entradas a utilizar (nuevamente empleando un rango de 0 a 1). El proceso de actualización de la celda de memoria lleva asociado una segunda capa, que vuelve a tomar como entrada el vector concatenado $[h_{t-1}, x_t]$ y, haciendo uso de una tangente hiperbólica como función de activación, devuelve un vector candidato a ser añadido a la celda de estado (g_t). El producto punto a punto de los vectores i_t y g_t da como resultado los nuevos valores candidatos escalados según el grado de actualización de cada elemento de la celda de estado. Este candidato será sumado al vector de estado auxiliar obtenido en el paso previo ($\tilde{c}_{t-1}$), para dar como resultado la nueva celda de estado c_t (véase la Figura 5.16).

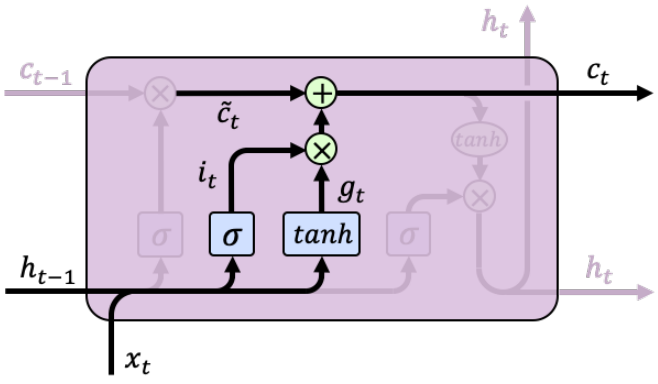


FIGURA 5.16: INPUT GATE

[3] El producto punto a punto de vectores es un caso particular del producto de Hadamard de matrices (véase Montalvo, 2020)

Las ecuaciones que rigen este proceso de actualización son

$$\begin{aligned} i_t &= \sigma\left(W_i\left[h_{t-1}, x_t\right] + b_i\right) \\ g_t &= tanh\left(W_g\left[h_{t-1}, x_t\right] + b_g\right) \\ c_t &= \tilde{c}_t + i_t * g_t \end{aligned} \tag{5.12}$$

donde $W_i \in \mathbb{R}^{n\times(n+m)}$ y $b_i \in \mathbb{R}^n$ representan la matriz de pesos y el vector de sesgos de la capa relativa a la *input gate*, y $W_g \in \mathbb{R}^{n\times(n+m)}$ y $b_g \in \mathbb{R}^n$ representan la matriz de pesos y el vector de sesgos de la capa complementaria que selecciona los candidatos a nuevos valores a añadir a la celda de estado.

- ***Output gate*** o puerta de salida: decide cuál debe ser la salida de la red basándose en las entradas y en la celda de estado. Para ello se utilizará una versión filtrada de la celda de memoria, dado que en lugar de utilizar sus propios valores se utilizará el resultado de aplicar la tangente hiperbólica a cada uno de los valores, lo que producirá salidas en el rango $(-1, 1)$. La capa asociada a la puerta de salida tomará una vez más como entrada la concatenación de h_{t-1} y x_t, y haciendo uso de una función sigmoide como función de activación devolverá un vector $o_t \in \mathbb{R}^n$ con los valores entre 0 y 1 qué determinan qué elementos se devolverán como salida de la red. Por último la salida se obtendrá del producto punto a punto de los vectores o_t y $tanh(c_t)$. Las ecuaciones que rigen este proceso de selección de la salida son

$$\begin{aligned} o_t &= \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right) \\ h_t &= o_t * tanh(c_t) \end{aligned} \tag{5.13}$$

donde $W_o \in \mathbb{R}^{n\times(n+m)}$ y $b_o \in \mathbb{R}^n$ representan la matriz de pesos y el vector de sesgos de la capa relativa a la *output gate*.

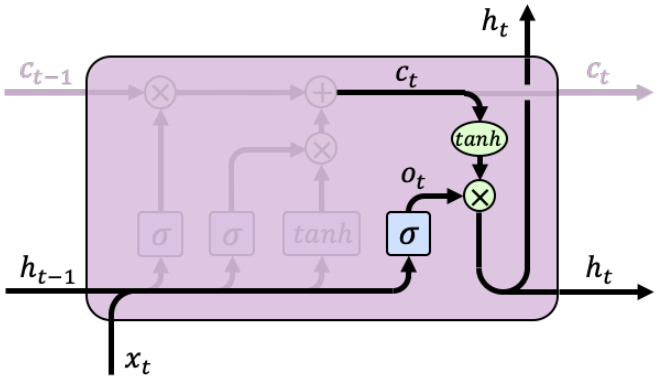


Figura 5.17: Oputput gate

Agrupando las ecuaciones 5.11, 5.12 y 5.13, se obtienen las ecuaciones que definen la estructura de la red LSTM:

$$
\begin{aligned}
f_t &= \sigma\left(W_f\left[h_{t-1}, x_t\right] + b_f\right) \\
i_t &= \sigma\left(W_i\left[h_{t-1}, x_t\right] + b_i\right) \\
o_t &= \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right) \\
g_t &= tanh\left(W_g\left[h_{t-1}, x_t\right] + b_g\right) \\
c_t &= f_t * c_{t-1} + i_t * g_t \\
h_t &= o_t * tanh(c_t)
\end{aligned}
$$

Tal y como se estudió en la sección relativa al entrenamiento de las RNN, el problema del desvanecimiento del gradiente viene propiciado por el valor de la derivada recursiva de la función que define el estado interno de la red. En el caso de las LSTM, la solución al problema viene dada por la estructura de la celda de estado. La función que define la celda de estado c_t, es función de f_t (la puerta de olvido), i_t (la puerta de entrada) y g_t (el candidato a celda de estado), y cada uno de ellos es a su vez función de c_{t-1} (dado que son funciones de h_{t-1} y por tanto también de c_{t-1}). Aplicando la regla de la cadena se obtiene

$$
\begin{aligned}
\frac{\partial c_j}{\partial c_{t-j}} = \; & c_{j-1}\,\sigma'(\cdot)\,W_f \,*\, o_{j-1}\,tanh'(c_{j-1}) \;+\; g_j\,\sigma'(\cdot)\,W_i \,*\, o_{j-1}\,tanh'(c_{j-1}) \\
& + i_j\,tanh'(\cdot)\,W_g \,*\, o_{j-1}\,tanh'(c_{j-1}) \;+\; f_j
\end{aligned}
\tag{5.14}
$$

Para propagar los errores k pasos, bastará con multiplicar los término en la forma 5.14 k veces. De esta expresión se puede observar la gran diferencias con respecto al gradiente recursivo de las RNN genérica. En las RNN genéricas los término $\frac{\partial h_j}{\partial h_{j-1}}$ toman un valor que o bien siempre estará por encima de 1, o bien siempre estará en el intervalo $[0, 1]$, lo que deriva en los problemas de desvanecimiento y explosión del gradiente. Sin embargo en el caso de las LSTM, se observa que la derivada recursiva (Ecuación 5.14) puede tomar para cualquier instante de tiempo previo tanto valores por encima de 1 como valores en el intervalo $[0, 1]$. Por tanto si se hace tender a infinito el número de lapsos de tiempo previo, en ningún caso está garantizado el desvanecimiento o la explosión del gradiente, esto es, no se garantiza que la norma del vector tienda a 0 o a infinito. Si durante el proceso de entrenamiento las contribuciones temporales comienzan a tender a 0, la red posee la capacidad de modificar los valores de alguna de las puertas para que el valor de $\frac{\partial c_j}{\partial c_{j-1}}$ crezca y se acerque más a 1. Esta capacidad de modificación de los pesos de la red para regular el desvanecimiento del gradiente es la que permite a la propia red decidir hasta qué punto del pasado es interesante considerar de cara a la detección de patrones estacionales o tendencias.

Tras conocer la estructura interna de una red LSTM, el siguiente paso de cara a la construcción de los modelos de predicción consiste en definir la arquitectura de la red neuronal. En función del tipo de problema de predicción se deberá elegir la arquitectura que mejor se alinee con los objetivos de predicción buscados. Para el procesamiento de series temporales, las dos arquitecturas clave son la arquitectura *vanilla* (dedicada a la implantación de un modelo de predicción *many-to-one*) y la arquitectura *encoder-decoder* (para la construcción de modelos de predicción *seq2seq*).

Red LSTM *Vanilla*

La configuración más básica de una red LSTM es la conocida como LSTM *Vanilla*. Esta arquitectura fue la definida originalmente por Hochreiter y Schmidhuber (1997), y está orientada a problemas de predicción con secuencias pequeñas. La arquitectura de una red LSTM *Vanilla* se compone de 3 capas: una capa de entrada, una capa LSTM oculta y una capa densa de salida.

Para problemas de predicción de series temporales, de cara a implantar un modelo de predicción de secuencias *many-to-one*, se requiere procesar una secuencia de tamaño fijo y predecir únicamente el siguiente valor de la secuencia. Por este motivo la capa LSTM deberá procesar todas las entradas que componen la secuencia, y la salida obtenida tras procesar el último elemento de la secuencia será la entrada que recibirá la capa densa con la finalidad de realizar la transformación oportuna según el contexto y obtener así la predicción del siguiente elemento de la serie temporal (véase la Figura 5.18).

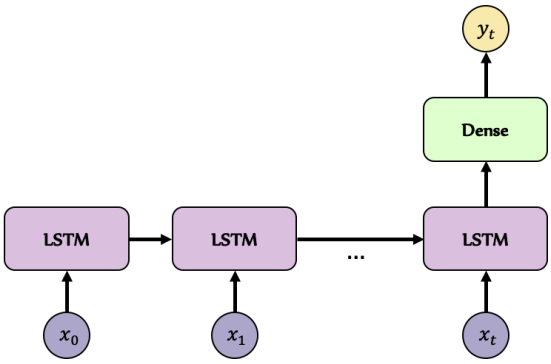


Figura 5.18: LSTM con arquitectura Vanilla para problemas de predicción muchos-a-uno

En función del contexto del problema de predicción a tratar, se pueden tomar adaptaciones sobre la arquitectura *Vanilla* para poder transformar el modelo de predicción *many-to-one* estudiado en un modelo *many-to-many*, sin más que analizar el origen de las variables que intervienen en la entrada de la red, $x_t \in \mathbb{R}^n$, así como las variables que se desea predecir, $y_t \in \mathbb{R}^m$.

En el caso de que la red utilice para predecir las mismas variables que son introducidas como entrada, esto es, si se verifica que $n = m$ y además una a una las variables de entrada y salida coinciden, es posible realizar predicciones con secuencias de salida de tamaño variable retroalimentando la red con los valores de salida de los pasos previos (Figura 5.19).

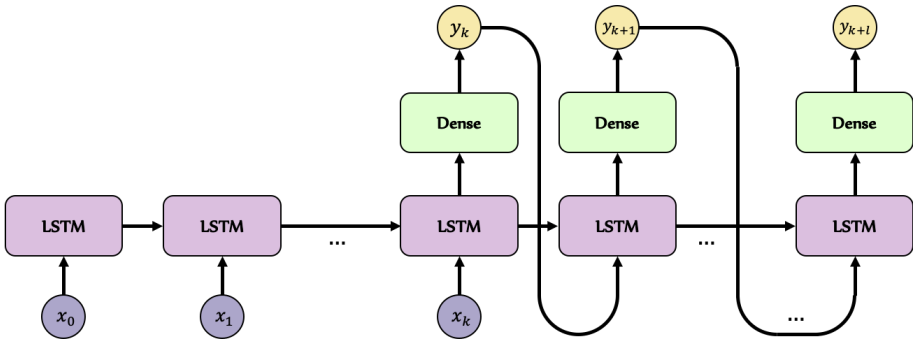


FIGURA 5.19: APLICACIÓN DE UNA RED LSTM VANILLA PARA PREDECIR UN NÚMERO ARBITRARIO DE VALORES FUTUROS

En incontables problemas de predicción de series temporales, a mayores de la serie temporal a estudio (bien sea univariante o multivariante), es posible disponer de variables auxiliares externas que se encuentran íntimamente relacionadas con la variable a estudio, y de las que además su comportamiento no es aleatorio, es decir, se conocen los valores que tomará la variable en cualquier instante de tiempo, pasado, presente o futuro. Este tipo de variables, a las cuales se hará referencia en adelante con el término **variable exógena**, serán de gran utilidad en el proceso de predicción. Algunos ejemplos representativos de variables exógenas que pueden ser de gran utilidad en la predicción de series temporales son la fecha de las observaciones, el día de la semana y la hora correspondientes a cada observación, si el día de la observación es o no festivo, etc. Obsérvese que todos los valores de estas variables de ejemplo son conocidos en cualquier instante de tiempo. Otras variables como la temperatura o la humedad no pueden ser utilizadas como variables exógenas, dado que se desconoce sus valores futuros.

La forma de integrar estas variables exógenas en una red LSTM con arquitectura *vanilla* utilizando retroalimentación consiste en entrenar la red incluyendo las variables exógenas en los vectores de entrada. De esta forma, si mantenemos la notación y llamamos x_t al vector de entrada compuesto por las observaciones en el instante t de las variables que componen la serie temporal a estudio, en este nuevo caso la entrada será la concatenación del vector x_t con un nuevo vector $f_t \in \mathbb{R}^s$ que incluye las observaciones de las s variables exógenas ligadas a la serie temporal.

Tras entrenar la red para realizar predicciones de un único valor tras procesar una secuencia de observaciones, la forma de utilizar la red para obtener una secuencia de predicciones es de nuevo retroalimentar las entradas con las salidas producidas por la red. Sin embargo, dado

que los valores de las variables exógenas son conocidos para cualquier instante de tiempo, se utilizan también para alimentar la entrada de la red. De esta forma la red neuronal dispondrá de datos reales relacionados con las variables a predecir en el momento de realizar cada una de las predicciones de la secuencia de salida, y no sólo los valores de las predicciones realizadas (véase Figura 5.20)

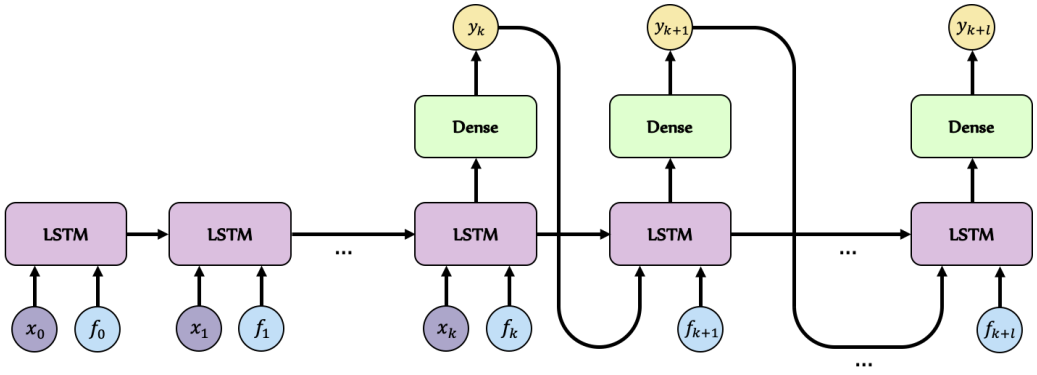


Figura 5.20: Aplicación de una red LSTM Vanilla para predecir un número arbitrario de valores futuros utilizando variables exógenas

Red LSTM *Encoder-Decoder*

Para problemas de predicción *seq2seq*, aquellos que toman como entrada una secuencia de observaciones y tratan de predecir la secuencia siguiente de observaciones[4], la opción de utilizar una arquitectura *Vanilla* con retroalimentación puede generar problemas de aproximación puesto que dicha arquitectura no ha sido concebida expresamente para este tipo de escenarios. La acumulación de los errores de predicción en el caso de retroalimentar las entradas con valores aproximados provoca una notable bajada de rendimiento de la red. Este hecho deriva en la necesidad de utilizar una arquitectura alternativa que sí permita trabajar con secuencias de salida, en lugar de restringir la predicción a valores unidimensionales.

Una aproximación a los problemas de predicción *seq2seq* que ha demostrado ser muy efectiva es la llamada red LSTM *Encoder-Decoder*. Esta arquitectura se compone de dos modelos:

- un modelo de lectura que codifica la entrada y devuelva un vector de tamaño fijo, y
- un modelo que decodifica el vector de tamaño fijo y construye la salida de la red.

[4]recuérdese que el tamaño de la secuencia de entrada no tiene porqué coincidir con el tamaño de la secuencia de salida

La red LSTM *Encoder-Decoder* fue concebida en su origen para problemas de procesamiento de lenguaje natural (NLP, *Natural Language Processing*), aportando grandes avances, en especial en el paradigma de traducción automática estadística, enmarcado dentro de la traducción automática de textos (véase Cho y col., 2014 para más información).

Tal y como se puede apreciar en la Figura 5.21, la información relativa a la secuencia de entrada viaja por toda la capa de codificación (*encoder*), hasta obtener los vectores de estado interno c_k y h_k obtenidos como salida tras el procesamiento del último elemento de la secuencia de entrada. La capa LSTM de decodificación (*decoder*) procesa en cada instante de tiempo el estado interno recibido del paso previo. Durante cada uno de estos instantes la red actualiza la celda de memoria interna y genera un nuevo estado interno, que servirá como entrada de la red en el siguiente instante y a su vez se utilizará para generar la salida del correspondiente instante de tiempo. Estas salidas son procesadas por una capa densa que lleva a cabo las transformaciones oportunas y genera una salida unitaria. Aplicando la capa LSTM de decodificación tantas veces como se desee es posible obtener secuencias de salida de tamaño arbitrario, tal y como se buscaba.

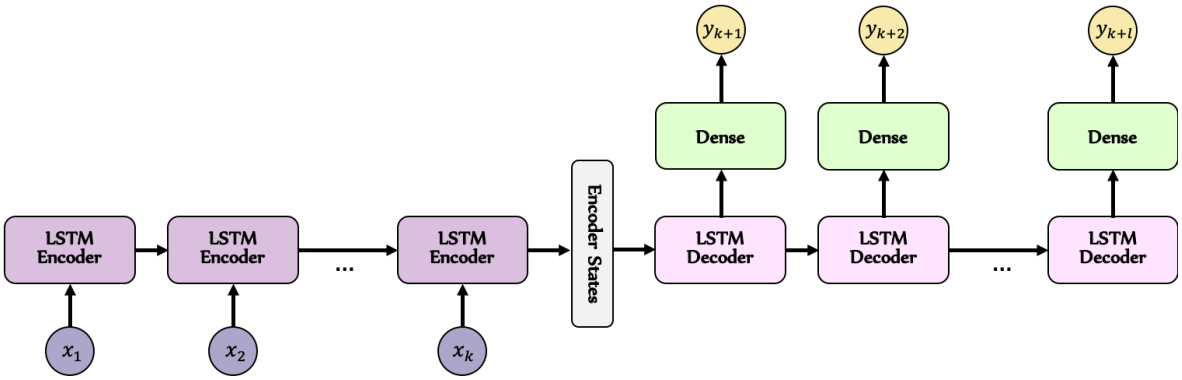


FIGURA 5.21: LSTM ENCODER-DECODER

De forma análoga a la construcción de la red LSTM con arquitectura *Vanilla* y variables exógenas, es posible extender las ventajas de dichas variables auxiliares a esta nueva arquitectura. El gran potencial que aporta el disponer de variables cuyo conocimiento es accesible en cualquier instante de tiempo permite mejorar las capacidades de predicción de la red dado que se permite a la red la posibilidad de establecer nuevas relaciones temporales entre las variables a estudio y las variables auxiliares.

En este caso para la construcción de la red LSTM *Encoder-Decoder* con variables exógenas se requiere por un lado incorporar en las entradas de la capa *Encoder* los valores de dichas variables, y además se deberán introducir en la capa *Decoder* los valores futuros de dichas variables. De este modo las entradas de la red *Decoder* se construyen a partir del estado interno previo y

los valores que toman las variables exógenas en el correspondiente instante de tiempo (véase la Figura 5.22).

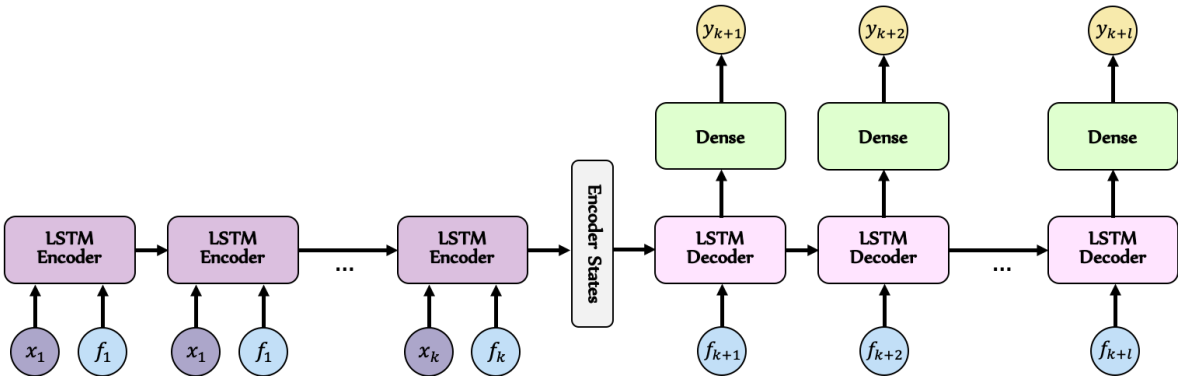


Figura 5.22: LSTM Encoder-Decoder con variables exógenas

Capítulo 6

Modelo de datos

6.1. Caso de estudio

Para la puesta en práctica de los modelos de predicción se ha utilizado un conjunto de datos de referencia que contiene las lecturas de consumos de energía (en kWh) de los bloques de oficinas 3 y 4 de Wood Quay del Ayuntamiento de Dublín entre Abril de 2011 y Febrero de 2013. Estos datos son públicos y se encuentran accesibles desde la web de la iniciativa *Smart Dublin*[1], bajo una licencia *Creative Commons Attribution*.

Figura 6.1: Bloques 3 y 4 de las Oficinas del Ayuntamiento de Dublín en Wood Quay

Smart Dublin es una iniciativa promovida por las cuatro autoridades locales cuya jurisdicción abarca colectivamente el área geográfica de la región de Dublín, con el apoyo de proveedores de tecnología, el mundo académico y los ciudadanos cuya única finalidad es mejorar la calidad de vida de la región. El proyecto trata de aportar soluciones innovadoras a los problemas locales agrupados en seis grandes temas: medio ambiente, personas, movilidad, economía, gobierno y bienestar.

[1] https://data.smartdublin.ie/dataset/energy-consumption-gas-and-electricity-civic-offices-2009-2012

Entre los objetivos de la iniciativa *Smart Dublin* se encuentra la eficiencia energética y la implantación de un modelo de edificación inteligente orientado a la reducción del consumo de energía y al aumento en la proporción de uso de energía renovables en comparación con las fuentes de energía no renovables.

En el año 2014 los edificios de oficinas principales del ayuntamiento de Dublín conformaban los edificios que más energía consumían de entre el total de activos del Ayuntamiento, con una elevada factura de electricidad que rondaba los 4,8GWh de electricidad al año, y un coste aproximado de 0,13 - 0,15 €/kWh de electricidad consumida durante los días de verano e invierno, respectivamente.

6.2. Raw data

Las mediciones de consumos se realizan en periodos de 15 minutos, incluyendo el consumo total de sendos bloques del edificio. La Tabla 6.1 muestra la estructura del fichero con extensión *.csv* con los datos "en bruto" (*raw data*), tal y como han sido extraídos de la web.

Civic Offices Blocks 3 and 4 KWh				···	
Date	Values	00:00	00:15	···	23:45
29/03/2011	96	nan	nan	···	33,60000229
30/03/2011	96	32,10000229	32,40000153	···	33,60000229
...	...	...	...	...	...
20/02/2013	96	25	24,5	···	26,5

TABLA 6.1: REGISTROS DEL CONJUNTO DE DATOS DE CONSUMOS (EN KWH) DEL AYUNTAMIENTO DE DUBLÍN

Se dispone de datos de un total de 695 días, entre el 29/03/2011 y el 20/02/2013. Para cada uno de los días se dispone de un total de 96 registros, desde las 00:00 a las 23:45 horas. Esto conlleva un total de 66.720 registros de consumo. En la Tabla 6.2 se pueden ver las características de los datos en crudo de los consumos medidos cada 15 minutos.

	Tipo de dato	Registros nulos	Media	Desviación típica	Valor mínimo	Valor máximo
Consumo (kWh) (cada 15 min.)	*Float*	209	42,288	17,889	0	89,550

TABLA 6.2: CARACTERÍSTICAS Y ESTADÍSTICOS DE LOS REGISTROS DE CONSUMO CAPTURADOS CADA 15 MIN.

6.3. Generación de nuevos datos

A partir de los datos en bruto es necesario llevar a cabo un proceso de preprocesado de los registros con el objetivo de obtener unos datos acorde con las necesidades del problema y sobre los que poder aplicar los modelos de predicción. Para ello será necesario un proceso de compuesto de tres etapas:

1. Transformación de datos.
2. Eliminación de valores ausentes (`NaN`, *Not a Number*) y valores atípicos (*outliers*).
3. División de datos para entrenamiento y pruebas.

Tras este proceso de tres etapas los datos obtenidos serán ya los datos sobre los que comenzar a construir los modelos de predicción. Sin embargo, dado que los procesos de transformación de datos y de eliminación de valores ausentes y atípicos en el presente contexto están íntimamente ligados, tal y como se detalla a continuación, ambas etapas se llevarán a cabo conjuntamente.

6.3.1. Transformación

La granularidad de los datos excede las necesidades requeridas de cara a la aplicación de los modelos con el fin de predecir consumos de energía futuros del edificio. Es por este motivo por el cual la primera transformación a realizar sobre el conjunto de datos consistirá en la agrupación de los registros de consumos en intervalos de una hora. Esto implica agrupar los consumos cada 4 valores, de forma que el nuevo conjunto de datos pase a estar indexado por el día y la hora del registro. Además, tras esta transformación, se deben almacenar los datos en un objeto de tipo `pandas.DataFrame`, utilizando como índice instancias de `pandas.Timestamp` construidas a partir de la fecha y hora del registro.

Durante el proceso de transformación se deberán descartan también los valores nulos de la serie, es decir, aquellos registros de los que no se dispone de lectura de consumo, junto con los registros de los que no se disponga el consumo en alguno de los cuatro intervalos de 15 minutos que componen el registro final. También se descartarán aquellos registros finales cuyo consumo se ha obtenido a partir de la suma de algún segmento con un consumo inferior a 10 kWh, dado que estos se consideran valores atípicos dada la naturaleza de los datos y el análisis generar obtenido en la Tabla 6.2.

El motivo de llevar a cabo la eliminación de registros ausentes y valores atípicos de forma conjunta con la transformación de los datos es debido a la necesidad de eliminar también los registros finales que se obtienen de la suma de entre los cuatro segmentos de 15 minutos de registros válidos y registros con valores atípicos o ausentes.

En el fragmento de Código 6.1 en Python se puede observar la implementación conjunta de los procesos de transformación y eliminación de valores ausentes y atípicos llevados a cabo sobre el conjunto de datos en bruto extraído del fichero *.csv*.

```
# Carga de datos
data = pd.read_csv("dcelectricitycivicsblocks34p20130221-1840.csv")

# Fecha y hora de los registros
date = data[["Civic Offices Blocks 3 and 4 KWh"]].iloc[1:,]
hour = data.iloc[:1, 2:].transpose()

# Se construye un nuevo dataframe repitiendo las fechas tantas veces como instantes fecha:hora existen
df = pd.DataFrame(np.repeat(date.values, len(data.columns)-2, axis=0), columns=['date'])
df['hour'] = pd.concat([hour]*(len(data)-1)).values

# Se almacenan en un array unidimensional todos los registros
data2 = data.iloc[1:,2:].to_numpy(dtype=float).flatten()

# Se parsea la fecha y hora de los registros para crear un dataframe indexado por fecha y hora
date_format = '%d/%m/%Y %H:%M'
index = pd.to_datetime(df['date'] + ' ' + df['hour'], format=date_format).map(lambda t: t.replace(minute=0))
df = pd.DataFrame(index=index, columns=['energy_consumption'])

# Se añade la columna de los valores al DataFrame
df['energy_consumption'] = data2.tolist()

# Se almacena el índice de los registros nulos y de consumo < 10
index_to_remove = df[(np.isnan(df['energy_consumption'])) | (df['energy_consumption']<10)]
index_to_remove = list(dict.fromkeys(index_to_remove.index))

# Se agruoan los registros por horas
df = df.groupby(df.index).sum()

# Se eliminan los registros nulos y de consumo < 10
df = df.loc[~(df.index.isin(index_to_remove))]
```

CÓDIGO 6.1: PREPROCESADO DE DATOS

Tras el proceso de preprocesado, se han obtenido un total de 16.614 registros de consumo por hora comprendidos entre el 29/03/2011 a las 17:00 y el 20/02/2013 a las 23:00.

Si analizamos las características de estos registros (véase Tabla 6.3), se observa mediante el análisis de la desviación típica y de los extremos del rango de valores una alta variabilidad en los datos. Las fluctuaciones diarias en los consumo producen una gran diferencia entre los consumos en horas laborales (rondando los 300 kWh) y los consumos fuera de dicho horario (estabilizándose los registros en torno a 100-150 kWh).

	Tipo de dato	Registros eliminados	Media	Desviación típica	Valor mínimo	Valor máximo
Consumo (kWh) (cada hora)	*Float*	66	169,159	71,070	89,5	346,500

Tabla 6.3: Características y estadísticos de los registros de consumo por horas

Con vistas a la posterior aplicación de los modelos de predicción, es necesario reducir esta variabilidad, para lo cual se requiere la aplicación de una transformación logarítmica de los registros. La Tabla 6.4 muestra los estadísticos de los nuevos datos tras aplicar la transformación, y el fragmento de Código 6.2 muestra la transformación logarítmica de los datos realizada en *Python*.

	Media	Desviación típica	Valor mínimo	Valor máximo
Consumo (kWh) (cada hora)	5,050	0,391	4,494	5,848

Tabla 6.4: Estadísticos de los logaritmos de los registros de consumo por horas

```
1 # Transformación logarítmica
2 df_sta = df.copy()
3 df_sta = np.log(df_sta['energy_consumption'])
```

Código 6.2: Transformación logarítmica

6.3.2. División de datos

Ante la necesidad de evaluar el grado de acierto en las predicciones realizadas con cada uno de los modelos de predicción, se requiere dividir el conjunto de datos total en dos subconjuntos:

- un **conjunto de entrenamiento** (*training set*) dedicado para buscar el modelo que mejor se ajuste a estos datos, y
- un **conjunto de prueba** (*test set*) sobre el cual probar la exactitud del modelo ya ajustado.

Cuando se realiza la subdivisión, es necesario asegurarse por un lado que el conjunto de prueba sea lo suficientemente grande como para generar resultados significativos desde el punto de vista estadístico, y por otro lado ha de ser representativo del conjunto de datos.

Para el conjunto de datos actual se reservarán el 85 % de los registros iniciales para entrenamiento (un total de 14.121 registros) y el 15 % final para pruebas (2.493 registros) (véase el fragmento de Código Python 6.3).

```
# División en entrenamiento (85%) y prueba (15%)
limit = int(len(df_sta)*0.85)
df_sta_train, df_sta_test = df_sta[:limit], df_sta[limit:]
```

CÓDIGO 6.3: DIVISIÓN DE DATOS EN ENTRENAMIENTO Y PRUEBAS

6.4. Análisis del modelo de datos final

Para la representación gráfica, los registros se interpretan como una serie temporal univariante donde la variable a estudio es el consumo de energía por horas (véase Figura 6.2).

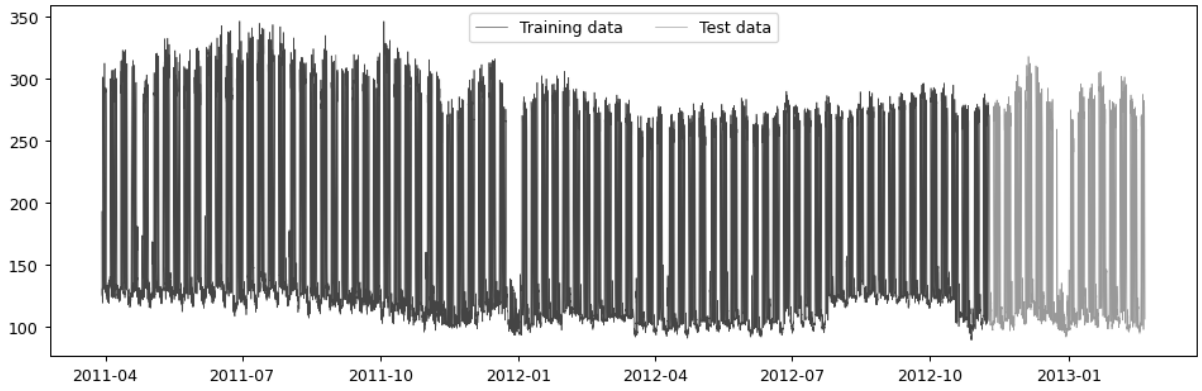


FIGURA 6.2: REPRESENTACIÓN GRÁFICA DE LA SERIE TEMPORAL COMPLETA A ESTUDIO

Si nos limitamos a observar los primeros meses de registros (Figura 6.3), es directo apreciar un comportamiento estacional en la serie que se repite de forma periódica todas las semanas. Durante los días laborables, el consumo de energía crece, comportamiento contrario al mostrado durante fines de semana (incluidos también días festivos) en los que se produce una notable bajada en el consumo de energía. Evidentemente estas apreciaciones concuerdan con la finalidad del edificio, dado que su utilización está marcada por los horarios de trabajo de los empleados del ayuntamiento. En otro tipo de infraestructuras, como puede ser el caso de edificios comerciales o

bloques de viviendas, el patrón de consumo podrá ser opuesto, destacando los picos de consumo en fines de semana y festivos.

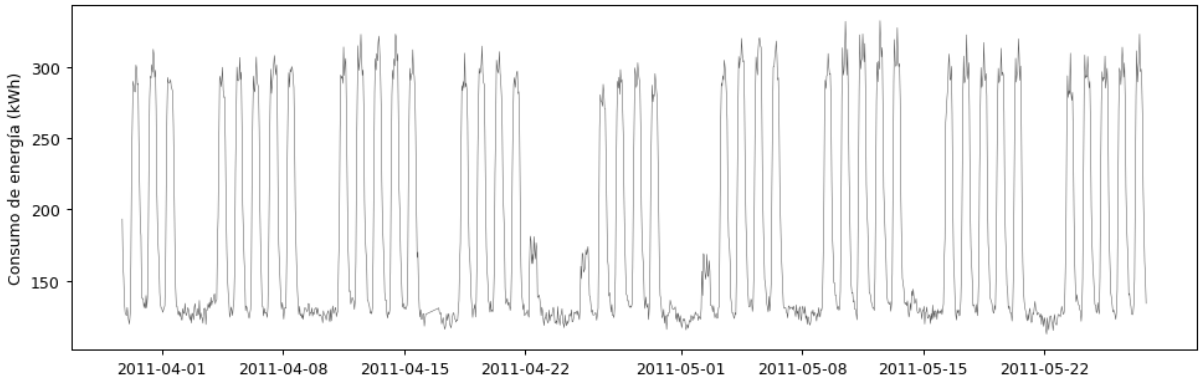


Figura 6.3: Representación gráfica de los dos primeros meses de la serie temporal a estudio

Centrando la atención en los consumos diarios (Figura 6.4) se aprecia un nuevo patrón de comportamiento correspondiente a una estacionalidad diaria. Efectivamente, el consumo de energía dentro de un mismo día sufre una variación durante las horas del día, mientras que por la noche el consumo del edificio se desploma, y de nuevo es a causa del uso que recibe el edificio, estando así ligado el consumo al correspondiente horario de trabajo de los empleados. Nótese sin embargo que durante los días no laborables el consumo se mantiene estable, sin fluctuaciones a causa de la inactividad laboral en el edificio. Esto podrá suponer un inconveniente de cara a configurar en los modelos de predicción los parámetros relacionados con la estacionalidad diaria, puesto que el comportamiento en estos días difiere del comportamiento habitual del edificio.

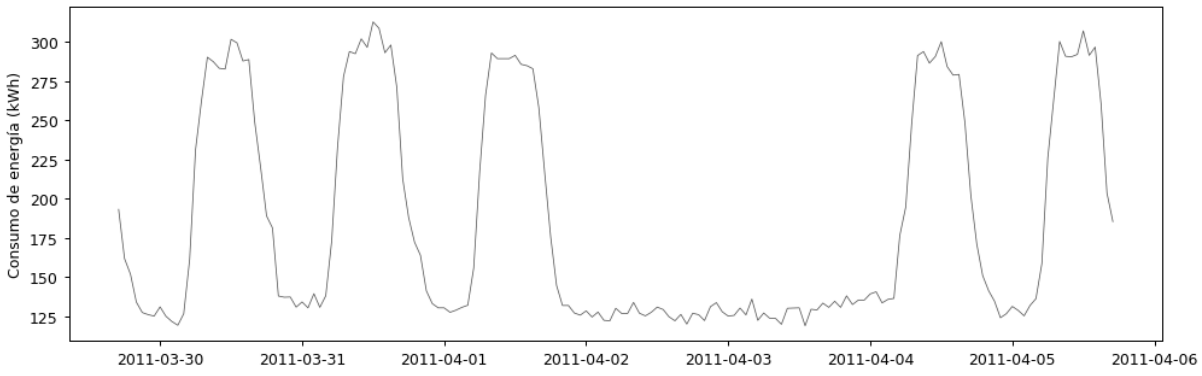


Figura 6.4: Representación gráfica de la primera semana de la serie temporal a estudio

Capítulo 7

Construcción de modelos en Python

En este capítulo se incluyen las construcciones de los modelos de predicción estudiados en el Capítulo 5 utilizando el lenguaje de programación *Python* y las librerías específicas detalladas en la sección 2.2. Tras explicar los pasos para la construcción de los modelos y la estimación de los parámetros que mejor se adaptan a la serie temporal a estudio de consumos de energía (analizada en el Capítulo 6), se detallan las construcciones necesarias para la puesta en práctica del modelo y la realización de predicciones del consumo de energía siguiendo un **esquema de ventanas móviles** (véase Figura 7.1).

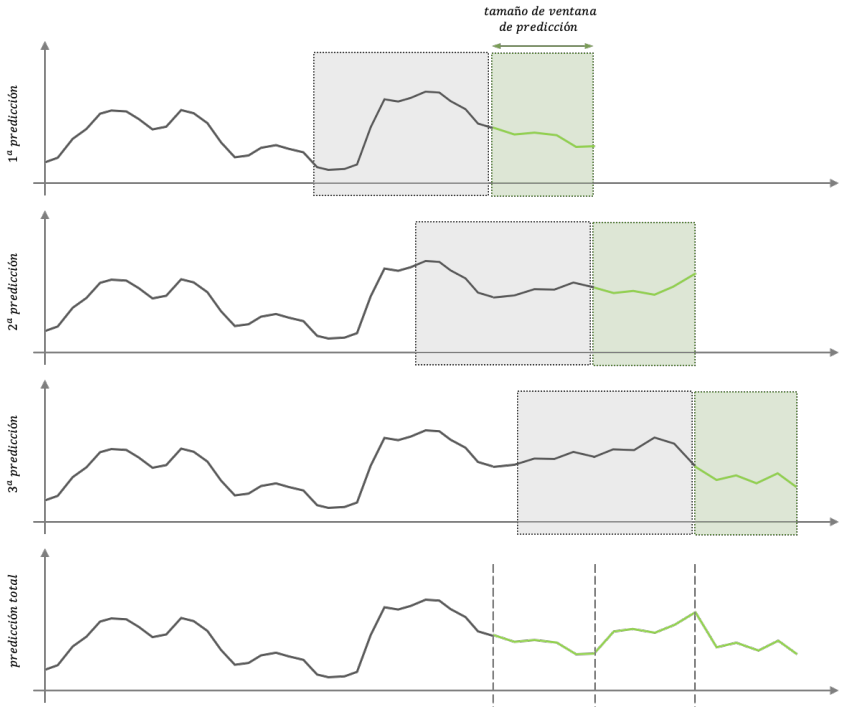


Figura 7.1: Procedimiento de predicción de secuencias siguiendo un modelo de ventanas móviles

El tamaño de la ventana hace referencia al número de predicciones a realizar tras analizar una secuencia de entrada con valores de consumos reales. El modelo de ventanas móviles lleva a cabo a partir de una secuencia de consumos reales la predicción del mismo número de valores que el correspondiente tamaño de la ventana. Tras una primera predicción, la ventana se desplaza hasta colocar el inicio de la nueva secuencia de predicciones seguida del final de la secuencia predicha en el paso previo. Los valores que preceden a la nueva ventana y que se utilizarán como entrada al modelo para realizar las predicciones deben ser valores reales y no las predicciones del paso previo.

En la Figura 7.1, la línea en color verde representa las predicciones realizadas por el modelo, mientras que la línea gris representa valores reales de la secuencia. Tras desplazar la ventana tantas veces como sea necesaria, las predicciones obtenidas en cada paso se agrupan para dar lugar a la secuencia total de predicciones del modelo (último gráfico de la figura).

Este modelo de ventanas móviles trata de simular el uso que posteriormente se dará al modelo de predicción. Si fijamos por ejemplo un tamaño de ventana de 24, dado que los registros de la serie temporal se indexan por horas, el modelo estaría llevando a cabo predicciones de consumos de energía para cada una de las horas que conforman el siguiente día a estudio. Un modelo de predicción con tamaño de ventana 24 permitirá realizar una vez al día una estimación del consumo en cada una de las horas del día siguiente, utilizando para ello en todo momento registros de consumos reales.

7.1. ARIMA

Para la construcción de los modelos ARIMA, tal y como se ha estudiado en la sección 5.2, el primer paso consiste en estudiar la estacionariedad y la autocorrelación del proceso estocástico asociado a la serie temporal a estudio, con el objetivo de determinar el valor de los parámetros p, d y q e inferir el modelo ARIMA(p,d,q) que mejor se ajuste a los datos de entrenamiento.

Tal y como se ha explicado en la sección 6.3, los datos utilizados para el proceso de inferencia del modelo ya han sido transformados mediante la aplicación de la función logaritmo natural, reduciendo la inestabilidad en varianza. Por este motivo, el único posible factor a tener en cuenta para obtener una serie estacionaria es la variabilidad en media.

Con el fin de analizar la estacionariedad se hará uso como herramientas alternativas de la función de autocorrelación (**ACF**, *Auto-Correlation Function*) y la función de autocorrelación parcial (**PACF**, *Partial Auto-Correlation Function*), las cuales miden el grado de dependencia temporal entre variables, esto es, el grado de influencia de las observaciones pasadas en las observaciones futuras. Desde el punto de vista probabilístico, estas funciones miden el grado

de correlación entre las variables aleatorias que componen el proceso estocástico asociado a la serie temporal a estudio. La **autocorrelación** mide la correlación entre dos variables separadas k periodos, mientras que la **autocorrelación parcial** mide la correlación entre dos variables separadas k periodos sin considerar la dependencia creada por los retardos intermedios existentes entre ambas. Ambos estadísticos toman valores entre -1 y 1. Si el valor es 0 esto significa que no existe relación entre sendos instantes de tiempo. En cambio si el valor es menor estrictamente que 0 (resp. mayor estrictamente que 0), existe una relación entre ambos instantes de tiempo y además esta relación es negativa (resp. positiva).

Para la estimación de los valores de estas funciones a partir de la realización del proceso estocástico se deben tomar las funciones de autocorrelación y autocorrelación parcial empíricas. Si se define la autocovarianza muestral como

$$\hat{\gamma}_k = \frac{1}{N} \sum_{t=k+1}^{N} (x_t - \bar{x})(x_{t-k} - \bar{x}),$$

donde $\bar{x}$ representa la media muestral de las observaciones y N el número total de observaciones, la autocorrelación se define a partir de la autocovarianza muestral como

$$\hat{\rho}_k = \frac{\hat{\gamma}_k}{\hat{\gamma}_0} = \frac{\sum_{t=k+1}^{N} (x_t - \bar{x})(x_{t-k} - \bar{x})}{\sum_{t=1}^{N} (x_t - \bar{x})^2}$$

El estudio probabilístico relativo a la construcción de los valores de autocorrelación parcial excede los objetivos del presente proyecto, y por lo tanto se limitará a considerar su definición, delegando en la librería *statsmodels* de *Python* su cálculo y omitiendo los detalles de su construcción empírica (puede consultarse la sección 3.5.2 del libro de Box y Jenkins (1970) para ampliar los detalles sobre la construcción de la función de autocorrelación parcial empírica).

Las funciones empíricas se construyen tomando los distintos valores de retardos (k) y para cada uno de ellos se estiman los coeficientes de autocorrelación y autocorrelación parcial. La Figura 7.2 muestra los valores obtenidos tras la aplicación de las funciones de autocorrelación y autocorrelación parcial sobre la serie temporal de consumos, para los retardos entre 0 y 200, haciendo uso para ello de las funciones `plot_acf` y `plot_pacf` de la librería *statsmodels*.

Se observa que la función de autocorrelación muestra un comportamiento cíclico, lo que demuestra la falta de estacionariedad de los datos. De ello se deriva que será necesario tomar al menos la serie de diferencias de primer orden para tratar de estabilizar los datos y eliminar estas

fluctuaciones. Por otra parte, si observamos la función de autocorrelación parcial, se observa que en los retardos en torno a los valores 24 y 168 el valor se aleja de 0 tomando valores negativos. Este hecho concuerda con la estacionalidad diaria y semanal que posee la serie (véase 6.4).

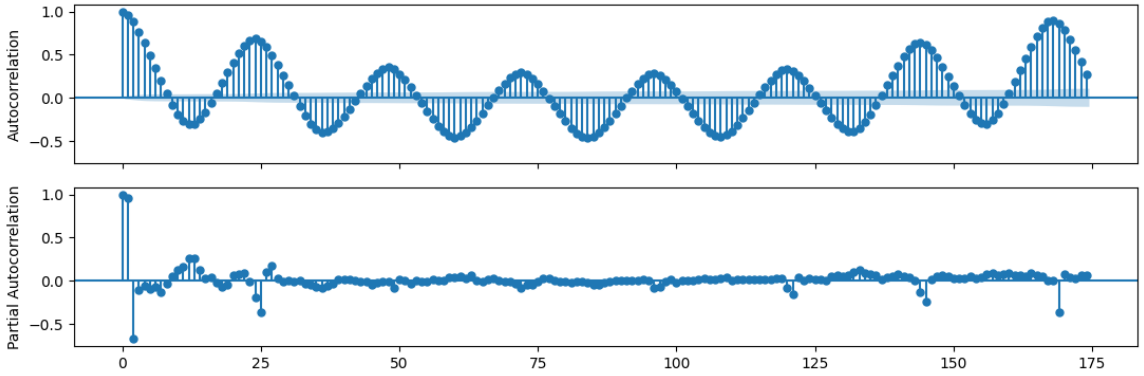


FIGURA 7.2: ACF Y PACF DE LA SERIE TEMPORAL DE CONSUMOS

Para inferir el modelo ARIMA que mejor se ajusta a los datos de entrenamiento se hará uso de la función `auto_arima` de la librería *pmdarima*. El Código 7.1 muestra la llamada a la función con los datos de entrenamiento.

```
autoarima_model = pm.auto_arima(y=df_sta_train,
                                 seasonal=False,
                                 stationary=False,
                                 information_criterion='aic',
                                 test='kpss',
                                 method='lbfgs',
                                 trace=True,
                                 error_action='ignore',
                                 suppress_warnings=True,
                                 stepwise=True)
```

CÓDIGO 7.1: ESTIMACIÓN DEL MODELO ARIMA

Algunos de los parámetros de la función toman los valores por defecto, no obstante se incluyen en la llamada a la función para remarcar la importancia del papel que desempeñan en la obtención del modelo. El argumento `seasonal=False` se utiliza para indicar que nos encontramos antes la estimación de un modelo ARIMA y no de un modelo SARIMA, y por tanto no deben estimarse los parámetros P, D y Q. En cuanto al argumento `stationary=False`, este se utiliza para remarcar que las observaciones de la serie introducida no constituyen una serie estacionaria, y por tanto se requiere estimar el valor del parámetro d relativo al orden de diferenciación.

Los argumentos `information_criterion='aic'` y `method='lbfgs'` hacen referencias al uso del Criterio de Información de Akaike como criterio de selección de modelos y al algoritmo L-BFGS como algoritmo de estimación de parámetros óptimos, respectivamente. El argumento `test='kpss'` hace referencias al uso del test de contraste de hipótesis de Kwiatkowski – Phillips – Schmidt – Shin (1992) para decidir si se requiere o no de diferenciación (parámetro d del modelo) con el fin de convertir la serie original es una serie estacionaria. Los argumentos restantes hacen referencias a parámetros de configuración de la salida obtenida con la ejecución de la función. La salida obtenida con la función aplicada sobre el caso a estudio es:

```
Performing stepwise search to minimize aic

 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=-30679.237, Time=2.50 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=-23409.143, Time=1.60 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=-30661.854, Time=0.90 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=-28391.133, Time=6.56 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=-23411.141, Time=0.70 sec
 ARIMA(1,1,2)(0,0,0)[0] intercept   : AIC=-30679.988, Time=7.85 sec
 ARIMA(0,1,2)(0,0,0)[0] intercept   : AIC=-30004.455, Time=7.77 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=-30666.491, Time=4.88 sec
 ARIMA(1,1,3)(0,0,0)[0] intercept   : AIC=-30678.193, Time=16.15 sec
 ARIMA(0,1,3)(0,0,0)[0] intercept   : AIC=-30385.590, Time=9.88 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=-30674.601, Time=9.45 sec
 ARIMA(2,1,3)(0,0,0)[0] intercept   : AIC=-30726.130, Time=34.00 sec
 ARIMA(3,1,3)(0,0,0)[0] intercept   : AIC=-31344.044, Time=35.30 sec
 ARIMA(3,1,2)(0,0,0)[0] intercept   : AIC=-31857.872, Time=27.03 sec
 ARIMA(3,1,1)(0,0,0)[0] intercept   : AIC=-30725.816, Time=21.56 sec
 ARIMA(4,1,2)(0,0,0)[0] intercept   : AIC=-31480.394, Time=32.04 sec
 ARIMA(4,1,1)(0,0,0)[0] intercept   : AIC=-30729.189, Time=30.86 sec
 ARIMA(4,1,3)(0,0,0)[0] intercept   : AIC=-30918.078, Time=42.95 sec
 ARIMA(3,1,2)(0,0,0)[0]             : AIC=inf, Time=11.20 sec

Best model:  ARIMA(3,1,2)(0,0,0)[0] intercept
Total fit time: 303.211 seconds
```

Observamos que tras probar distintas combinaciones de parámetros, el modelo que menor AIC obtiene es el modelo `ARIMA(3,1,2)` + constante .

Observación: El objeto obtenido con la función `auto_arima` es de tipo `pmdarima.arima.ARIMA` de la librería *pmdarima*. Sin embargo, este objeto se ha construido a bajo nivel a partir de un objeto de tipo `statsmodels.tsa.arima.model.ARIMAResults` de la librería *statsmodels*. Esta segunda librería posee una gran variedad de funciones sobre las que trabajar con modelos ARIMA, y por este motivo se hará uso del objeto `ARIMAResults` contenido en el atributo `arima_res_` del modelo de *pmdarima* para poder hacer uso así de dichas funciones (véase Código 7.2).

```
# pmdarima object -> statsmodels object
arima_model = autoarima_model.arima_res_
```

CÓDIGO 7.2: TRANSFORMACIÓN DE UN OBJETO DE PMDARIMA EN UN OBJETO DE STATSMODEL

En cuanto al análisis de los errores residuales de predicción sobre el conjunto de entrenamiento, la Figura 7.3 muestra la serie temporal de los errores residuales (obtenida como la diferencias de los valores reales y las predicciones, $e_t = x_t - \hat{x}_t$), así como la distribución empírica de los mismos. Se observa como la distribución empírica de los datos se asemeja a una distribución normal de media 0 y varianza finita, aunque con una pequeña diferencias en cuanto a las longitudes de las colas en sendos sentidos. Nótese que la cola derecha de la distribución es más larga, lo que se puede apreciar también en serie de residuos, con un incremento de los valores positivos con respecto a los negativos. Esto demuestra que, si bien el modelo aproxima correctamente a los datos, hay un cierto patrón que no logra modelar.

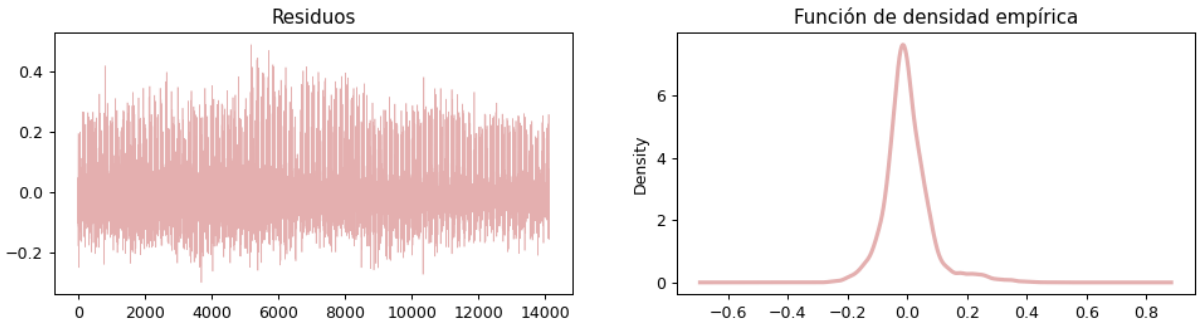


FIGURA 7.3: DISTRIBUCIÓN DE LOS RESIDUOS DEL MODELO ARIMA

Tras haber obtenido el modelo y haber comprobado su la validez, el siguiente paso consiste en realizar las predicciones (véase Figura 7.4).

El modelo ARIMA tal y como se construye está pensado para la realización de predicciones de un único valor futuro, lo que se correspondería a un tamaño de ventana unitario en el correspondiente enfoque de ventanas móviles. Nótese sin embargo que con la técnica de retroalimentación de predicciones es posible ampliar el rango de aplicación a cualquier tamaño de ventana. La Figura 7.4 muestra las predicciones para los tamaños de ventana 1, 6 y 24, junto con una predicción de tamaño igual al número de elementos del conjunto de prueba (lo que equivale a un tamaño de ventana de 2493).

En la figura se puede apreciar cómo la precisión de las predicciones empeora a medida que se aumenta el tamaño de la ventana, lo que concuerda con lo mencionado previamente. Nótese que la predicción realizada sin retroalimentar el modelo con predicciones y tomando como tamaño del ventana todo el conjunto de entrenamiento aporta unos resultados muy alejados de la

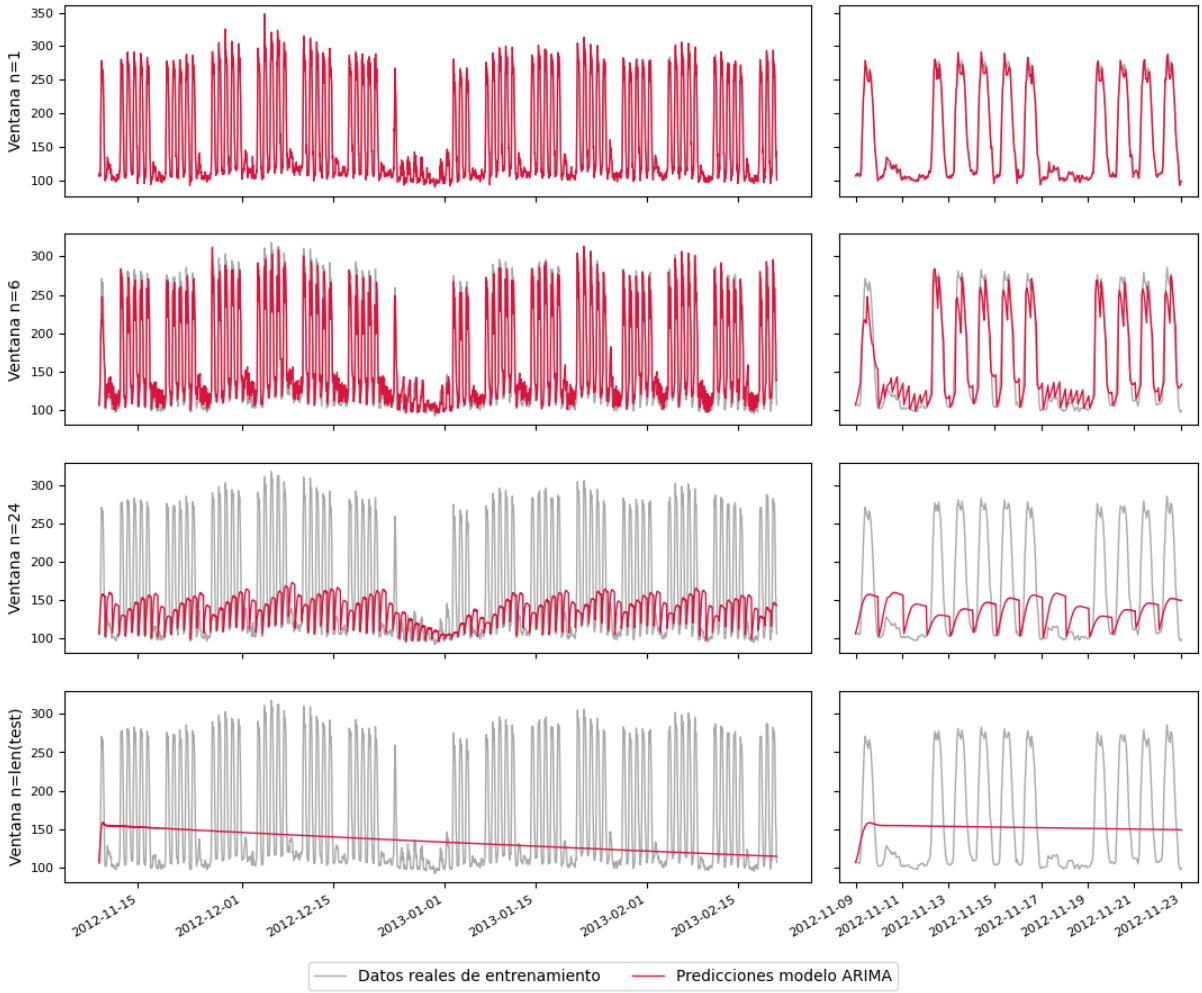


Figura 7.4: Predicciones con el modelo ARIMA para tamaños de ventana 1, 6 y 24, junto con la predicción sobre el total de elementos del conjunto de prueba, ampliando sobre las gráficas derechas las predicciones de las dos primeras semanas

realidad. En parte este hecho se debe a la falta de capacidad de los modelos ARIMA a modelar comportamientos estacionales. Un análisis más profundo sobre estas conclusiones se pospone al capítulo 8 de este mismo documento.

7.2. SARIMA

Tal y como se comentaba anteriormente, la falta de capacidad de modelado de estacionalidades de los modelos ARIMA lleva a obtener resultados muy poco efectivos a largo plazo. Es por este motivo por el cual se requiere de modelos de predicción alternativos que permitan modelar los periodos estacionales. Para la construcción del modelo SARIMA se debe en primer lugar

marcar el periodo de estacionalidad a modelar, ya que, tal y como se estudio en la sección 5.2.1, estos modelos sólo admiten un periodo de estacionalidad.

Ya se ha visto en 6.4 que los dos periodos de estacionalidad son 24 (estacionalidad diaria) y 168 (estacionalidad semanal). Sin embargo, las implementaciones de los algoritmos de estimación de los coeficientes óptimos no permiten trabajar con periodos de estacionalidad elevados, ya que su puesta en práctica se ve topada con problemas de capacidad de la memoria RAM durante su ejecución. Por este motivo, esta sección se centra en la construcción de un modelo de predicción que modele periodos de estacionalidad diarios, dejando el modelado de la estacionalidad semanal en los modelos ARIMA con series de Fourier (véase la sección 7.3).

De forma análoga a la forma de proceder con los modelos ARIMA, para comenzar el estudio del modelo SARIMA óptimo es necesario estudiar los valores de la ACF y PACF, pero en este caso el estudio se complementará con el análisis de las funciones de autocorrelación y autocorrelación parcial de la serie de diferencias estacionales, con periodo $m = 24$ (véase Figura 7.5).

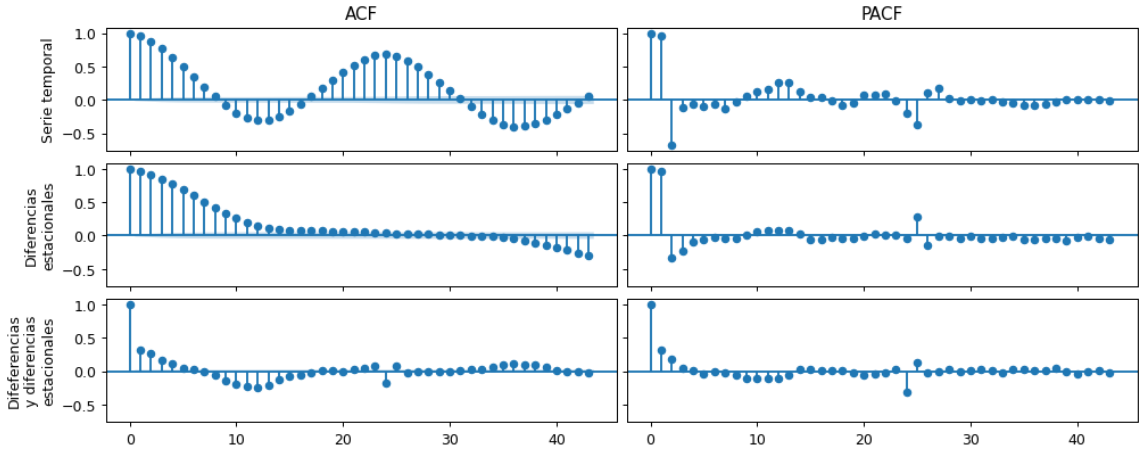


FIGURA 7.5: ACF Y PACF DE LA SERIE TEMPORAL DE CONSUMOS, LA SERIE DE DIFERENCIAS ESTACIONALES DE PERIODO 24 Y LA SERIE DE DIFERENCIAS Y DIFERENCIAS ESTACIONALES DE PERIODO 24

En la figura se pueden apreciar tres casos a estudio, cada uno de los cuales ligado a una posible configuración de los parámetros d y D del modelo SARIMA:

- La serie original, relativa a utilizar los valores de los parámetros $d = D = 0$.
- La serie de diferencias estacionales con periodo de estacionalidad 24, que se corresponde a utilizar los parámetros $d = 0$ y $D = 1$.
- La serie de diferencias aplicada sobre la serie de diferencias estacionales de periodicidad 24, que se corresponde con los valores $d = D = 1$.

Analizando los valores de ACF y PACF para cada una de las series a estudio, los valores más bajos de autocorrelación se obtienen para la última serie. No obstante, en los tres casos sigue existiendo un repunte en la correlación para valores próximos a 24, lo cual viene propiciado por la estacionalidad semanal dado que no se contempla en este modelo y su periodicidad afecta también al comportamiento diario.

Por este motivo previo en la llamada la función `auto_arima` se fijarán los valores de los parámetros d y D del modelo SARIMA a 1, con el fin de reducir el tiempo de ejecución del algoritmo de estimación de los coeficientes óptimos. El Código 7.3 muestra la función con los respectivos argumentos de entrada para la búsqueda del modelo SARIMA óptimo.

```
autoarima_seasonal_model = pm.auto_arima(y=df_sta_train,
                                         seasonal=True,
                                         m=24,
                                         D=1,
                                         d=1,
                                         max_p=5,
                                         stationary=False,
                                         information_criterion='aic',
                                         method='lbfgs',
                                         trace=True,
                                         error_action='ignore',
                                         suppress_warnings=True,
                                         stepwise=True)
```

Código 7.3: Estimación del modelo SARIMA

Nótese que para prevenir problemas de memoria durante la ejecución del algoritmo, el valor del parámetro p se ha limitado para que no exceda de 5. Los nuevos parámetros que aparecen en la llamada a la función con respecto al caso previo del modelo ARIMA son el periodo de estacionalidad (`m=24`), y los valores de los parámetros de diferenciación, que en este caso quedan fijados con antelación (`d=1` y `D=1`).

La salida obtenida tras ejecutar la función ha sido:

```
Performing stepwise search to minimize aic

 ARIMA(2,1,2)(1,1,1)[24]             : AIC=inf, Time=119.45 sec
 ARIMA(0,1,0)(0,1,0)[24]             : AIC=-31590.397, Time=7.95 sec
 ARIMA(1,1,0)(1,1,0)[24]             : AIC=-33921.556, Time=30.75 sec
 ARIMA(0,1,1)(0,1,1)[24]             : AIC=inf, Time=34.14 sec
 ARIMA(1,1,0)(0,1,0)[24]             : AIC=-32819.372, Time=3.77 sec
 ARIMA(1,1,0)(2,1,0)[24]             : AIC=-34499.067, Time=93.04 sec
 ARIMA(1,1,0)(2,1,1)[24]             : AIC=inf, Time=165.47 sec
```

```
    ARIMA(1,1,0)(1,1,1)[24]             : AIC=inf, Time=52.44 sec
    ARIMA(0,1,0)(2,1,0)[24]             : AIC=-32956.994, Time=50.54 sec
    ARIMA(2,1,0)(2,1,0)[24]             : AIC=-34888.098, Time=116.71 sec
    ARIMA(2,1,0)(1,1,0)[24]             : AIC=-34344.391, Time=64.60 sec
    ARIMA(2,1,0)(2,1,1)[24]             : AIC=inf, Time=246.75 sec
    ARIMA(2,1,0)(1,1,1)[24]             : AIC=inf, Time=87.84 sec
    ARIMA(3,1,0)(2,1,0)[24]             : AIC=-34897.923, Time=122.79 sec
    ARIMA(3,1,0)(1,1,0)[24]             : AIC=-34350.171, Time=48.46 sec
    ARIMA(3,1,0)(2,1,1)[24]             : AIC=inf, Time=204.40 sec
    ARIMA(3,1,0)(1,1,1)[24]             : AIC=inf, Time=88.99 sec
    ARIMA(4,1,0)(2,1,0)[24]             : AIC=-34900.099, Time=146.64 sec
    ARIMA(4,1,0)(1,1,0)[24]             : AIC=-34350.445, Time=64.92 sec
    ARIMA(4,1,0)(2,1,1)[24]             : AIC=inf, Time=349.30 sec
    ARIMA(4,1,0)(1,1,1)[24]             : AIC=inf, Time=122.30 sec
    ARIMA(5,1,0)(2,1,0)[24]             : AIC=-34916.101, Time=190.14 sec
    ARIMA(5,1,0)(1,1,0)[24]             : AIC=-34367.685, Time=63.56 sec
    ARIMA(5,1,0)(2,1,1)[24]             : AIC=inf, Time=314.62 sec
    ARIMA(5,1,0)(1,1,1)[24]             : AIC=inf, Time=119.27 sec
    ARIMA(5,1,1)(2,1,0)[24]             : AIC=-34913.839, Time=176.56 sec
    ARIMA(4,1,1)(2,1,0)[24]             : AIC=-34906.375, Time=289.76 sec
    ARIMA(5,1,0)(2,1,0)[24] intercept   : AIC=-34914.101, Time=882.91 sec

Best model:  ARIMA(5,1,0)(2,1,0)[24]
Total fit time: 2188.351 seconds
```

Tras probar distintas combinaciones de parámetros, el modelo que menor AIC obtiene es el modelo `SARIMA(5,1,0)(2,1,0)[24]`. La Figura 7.6 muestra la serie temporal de errores residuales tras la aplicación del modelo, junto con la función de densidad empírica de dichos residuos. En este caso de nuevo la distribución sigue un modelo normal de media cero, pero esta vez la serie es mucho más estable y no presenta asimetrías en sus colas. Esto es un indicador de un mejor ajuste y una correcta interpretación de los datos.

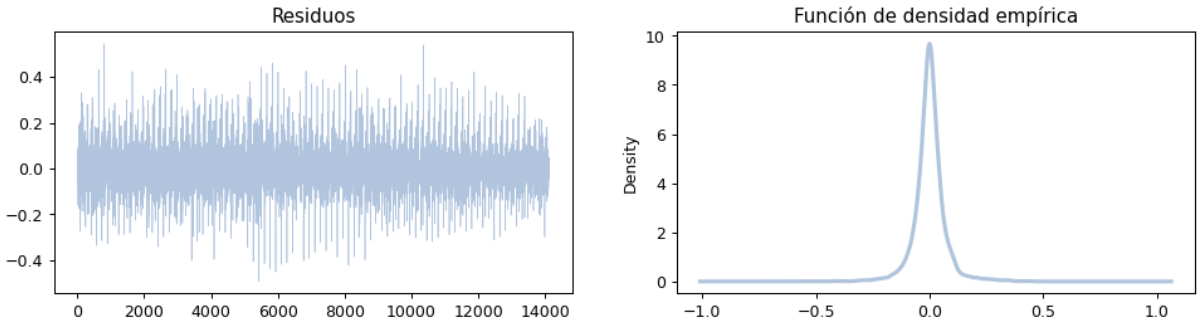


FIGURA 7.6: DISTRIBUCIÓN DE LOS RESIDUOS DEL MODELO SARIMA

Tras haber construido el modelo, el último paso radica en su aplicación al problema de predicción sobre los registros de entrenamiento. La Figura 7.7 muestra las predicciones realizadas con el modelo SARIMA óptimo para los tamaños de ventana 1, 6 y 24, así como la predicción sin retroalimentación del total de registros del conjunto de prueba.

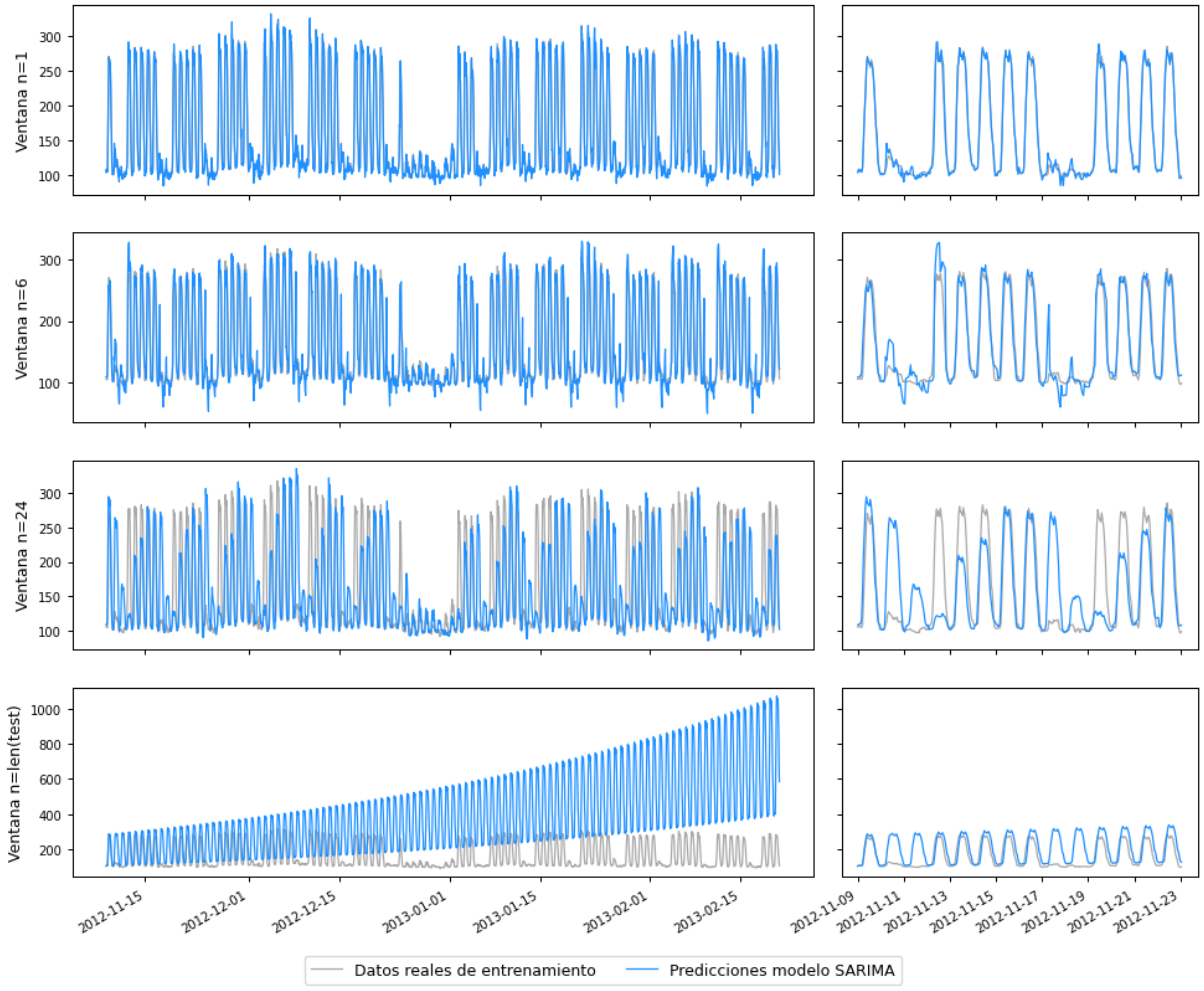


Figura 7.7: Predicciones con el modelo SARIMA para tamaños de ventana 1, 6 y 24, junto con la predicción sobre el total de elementos del conjunto de prueba, ampliando sobre las gráficas derechas las predicciones de las dos primeras semanas

El modelo aporta importantes resultados en cuanto a aproximación de los valores reales, en los tres casos en los que se utilizan ventanas pequeñas, incluso los resultados son bastante buenos a simple vista en el caso de un modelo de ventanas diario. Por el contrario el modelo no arroja resultados nada esperanzadores para la predicción total, en parte debido a que su uso no está pensado para predicciones a largo plazo.

7.3. ARIMA con series de Fourier

Para la construcción del modelo ARIMA con series de Fourier se requiere la definición de las variables auxiliares que actuaran como series temporales tomando los valores de los respectivos términos de Fourier con distintas frecuencias y amplitudes. Recordando la ecuación 5.2, los valores de las series temporales auxiliares que se necesitan calcular son

$$f_{k,t}^{(24)} = sin\Big(\frac{2\,k\,\pi}{24}\,t\Big), \quad g_{k,t}^{(24)} = cos\Big(\frac{2\,k\,\pi}{24}\,t\Big)$$

para la estacionalidad de 24 periodos, para cada $k = 1, ..., 12$, y

$$f_{k,t}^{(168)} = sin\Big(\frac{2\,k\,\pi}{168}\,t\Big), \quad g_{k,t}^{(168)} = cos\Big(\frac{2\,k\,\pi}{168}\,t\Big)$$

para la estacionalidad de 168 periodos, para cada $k = 1, ..., 84$. Para su cálculo se hará uso de la función auxiliar `FourierFeaturizer` de la librería *pmdarima*, la cual recibe como parámetros el tamaño del periodo y el número de términos (pares de seno y coseno) a considerar. Si m_i denota el periodo de la estacionalidad, el valor por defecto utilizado como número de periodos de la serie de Fourier es $\lfloor m_i/2 \rfloor$, que será el utilizado en este caso (véase Código 7.4).

```
# Términos de Fourier
exog_24 = FourierFeaturizer(m=24).fit_transform(df_sta_train)
exog_168 = FourierFeaturizer(m=24*7).fit_transform(df_sta_train)
exog = pd.concat([exog_24[1], exog_168[1]], axis=1, join='inner')
```

CÓDIGO 7.4: CÁLCULO DE LOS TÉRMINOS DE LAS SERIES DE FOURIER

Una vez se dispone de estos valores, el siguiente paso en la construcción del modelo consiste en la utilización de nuevo de la función `auto_arima` de la librería *pmdarima* para buscar el modelo ARIMA que mejor ajuste la serie a estudio, utilizando además las variables exógenas contenidas en el la variable de Python `exog`, derivadas de las series de Fourier (véase 7.5).

```
# Auto-arima model + Fourier
autoarima_fourier_model = pm.auto_arima(y=df_sta_train,
                                        exogenous=exog,
                                        seasonal=False,
                                        stationary=False,
                                        trace=True,
                                        error_action='ignore',
                                        information_criterion='aic',
                                        test='kpss',
```

```
10                                          method='lbfgs',
11                                          suppress_warnings=True,
12                                          stepwise=True)
```

Código 7.5: Estimación del modelo ARIMA con términos de Fourier

Nótese que en este caso se hace uso de un nuevo parámetro, `exogenous`, utilizado para incorporar al modelo ARIMA las nuevas variables exógenas. El resto de argumentos se mantienen invariantes con respecto a la llamada a la función en el caso de un modelo ARIMA simple (véase el Código 7.1). La salida obtenida por la función es:

```
Performing stepwise search to minimize aic

 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=-32426.114, Time=169.49 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=-26197.076, Time=162.14 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=-32394.686, Time=148.69 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=-30588.703, Time=467.30 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=-26199.069, Time=245.48 sec
 ARIMA(1,1,2)(0,0,0)[0] intercept   : AIC=-32424.585, Time=81.20 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=-32408.492, Time=781.19 sec
 ARIMA(3,1,2)(0,0,0)[0] intercept   : AIC=-32500.394, Time=624.84 sec
 ARIMA(3,1,1)(0,0,0)[0] intercept   : AIC=-32427.119, Time=70.35 sec
 ARIMA(4,1,2)(0,0,0)[0] intercept   : AIC=-32417.602, Time=737.33 sec
 ARIMA(3,1,3)(0,0,0)[0] intercept   : AIC=-32415.208, Time=736.79 sec
 ARIMA(2,1,3)(0,0,0)[0] intercept   : AIC=-32409.978, Time=666.96 sec
 ARIMA(4,1,1)(0,0,0)[0] intercept   : AIC=-32424.846, Time=285.30 sec
 ARIMA(4,1,3)(0,0,0)[0] intercept   : AIC=-32418.149, Time=639.71 sec
 ARIMA(3,1,2)(0,0,0)[0]             : AIC=-32503.084, Time=524.18 sec
 ARIMA(2,1,2)(0,0,0)[0]             : AIC=-32428.114, Time=126.97 sec
 ARIMA(3,1,1)(0,0,0)[0]             : AIC=-32429.120, Time=273.54 sec
 ARIMA(4,1,2)(0,0,0)[0]             : AIC=-32419.523, Time=593.10 sec
 ARIMA(3,1,3)(0,0,0)[0]             : AIC=-32417.208, Time=615.01 sec
 ARIMA(2,1,1)(0,0,0)[0]             : AIC=-32410.493, Time=496.48 sec
 ARIMA(2,1,3)(0,0,0)[0]             : AIC=-32411.861, Time=609.61 sec
 ARIMA(4,1,1)(0,0,0)[0]             : AIC=-32426.846, Time=271.32 sec
 ARIMA(4,1,3)(0,0,0)[0]             : AIC=-32420.211, Time=627.03 sec

Best model:  ARIMA(3,1,2)(0,0,0)[0]
Total fit time: 9954.394 seconds
```

El modelo que mejor se ajusta a la serie es el modelo `ARIMA(3,1,2)` + Fourier . Un análisis básico de los errores residuales muestra una asimetría en la colas de la función de densidad, lo que implica, al igual que en el caso del modelo ARIMA básico, que el modelo no ajusta del todo correctamente los datos, si bien sus diferencias no son excesivamente notables.

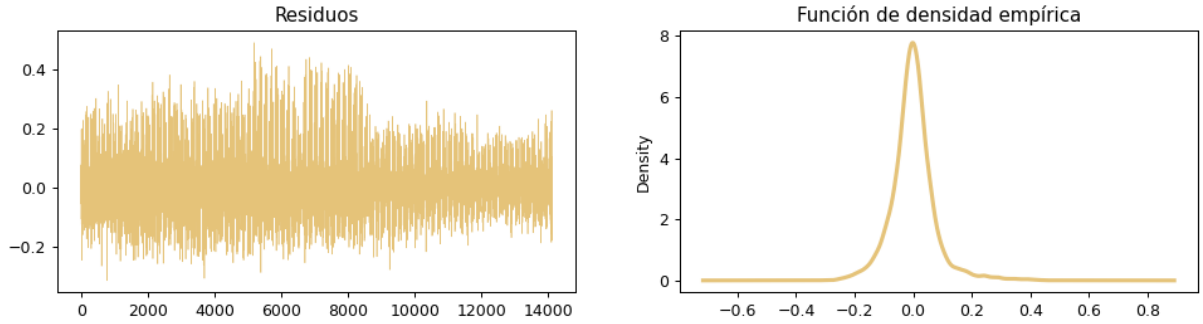


FIGURA 7.8: DISTRIBUCIÓN DE LOS RESIDUOS DEL MODELO ARIMA

Tomando los valores estimados de los coeficientes que acompañan a las series temporales de las variables de Fourier en el modelo estimado, es posible dar una representación de la estacionalidad estimada de la serie. Es decir, si nos limitamos a tomar exclusivamente la parte del modelo relativa a los términos de Fourier, el valor resultante de la combinación de todos ellos representa la componente estacional del modelo. De forma muy simple, utilizando el atributo `_params_exog` del modelo, el cual almacena los coeficientes de las variables exógenas, es posible representar la serie de estacionalidades sin más que tomar la combinación lineal de estos coeficientes y las columnas de la variable `exog` con los valores de los términos de Fourier (véase Código 7.6).

```
# Modelado de la estacionalidad
fourier_terms = np.zeros(len(exog.columns))
for i in range(len()):
    fourier_terms += fourier_model._params_exog[i]*exog.iloc[:,i]
```

CÓDIGO 7.6: CÁLCULO DE LA SERIE DE ESTACIONALIDADES DEL MODELO

La Figura 7.9 muestra la representación gráfica de componente estacional de la serie a estudio aproximada mediante los términos de Fourier que complementan al modelo ARIMA construido.

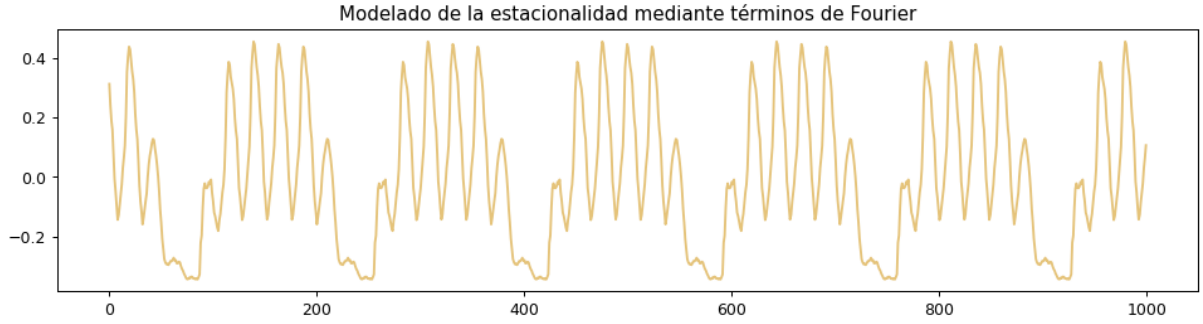


FIGURA 7.9: MODELADO MEDIANTE TÉRMINOS DE FOURIER DE LA ESTACIONALIDAD DE LA SERIE TEMPORAL

Para concluir el último paso consiste en aplicar el modelo para realizar las predicciones oportunas. Del mismo modo que se ha hecho en secciones previas, se llevará a cabo una predicción con ventanas de tamaño 1, 6 y 24, así como la predicción total sobre el conjunto de prueba (véase Figura 7.10).

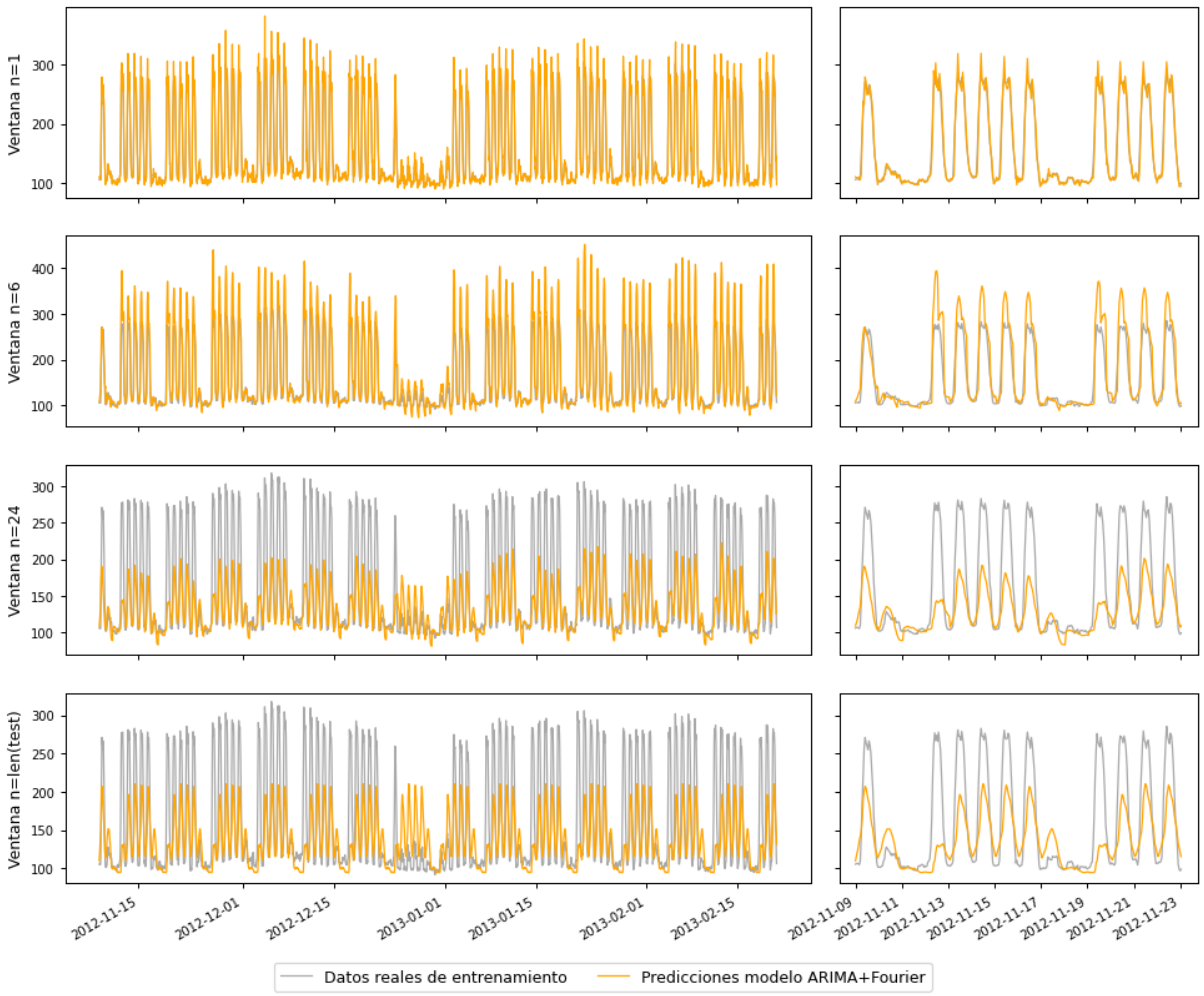


Figura 7.10: Predicciones con el modelo ARIMA con términos de Fourier para tamaños de ventana 1, 6 y 24, junto con la predicción sobre el total de elementos del conjunto de prueba, ampliando sobre las gráficas derechas las predicciones de las dos primeras semanas

Nótese que a medida que se aumenta el tamaño de la ventana, las predicciones tienden a estabilizarse en torno al comportamiento estacional modelado por los términos de Fourier, sin llegar a notarse apenas diferencia mediante la observación de las gráficas entre una predicción con tamaño de ventana 24 y las predicciones sin retroalimentación sobre el total del conjunto de datos de prueba.

7.4. TBATS

La simplicidad del modelo TBATS, en cuanto a configuración se refiere, hace que su proceso de construcción sea simple. Todo el peso de la construcción recae sobre el algoritmo de búsqueda de parámetros óptimos. Para ello se hará uso de la librería *tbats*, la cual implementa el algoritmo de estimación de coeficientes siguiendo los procedimientos teóricos vistos en la Sección 5.3 mediante las funciones `TBATS` y `fit` (véase el Código 7.7).

```
# Modelo TBATS
estimator = TBATS(seasonal_periods = (24,168),
                  use_arma_errors=None,
                  use_box_cox=False,
                  use_trend=False,
                  use_damped_trend=False)

# Ajuste del modelo con datos de entrenamiento
model_fit = estimator.fit(df_sta_train)
```

CÓDIGO 7.7: ESTIMACIÓN DEL MODELO TBATS ÓPTIMO

Para la configuración del modelo se han tomado como argumentos de la función `TBATS` los dos periodos estacionales, `seasonal_periods = (24,168)`, junto con las características de los modelos TBATS que aplican al presente caso. Dado que los datos ya han sido transformados mediante una transformación logaritmica, puede prescindirse de la aplicación de la transformada de Box-Cox, tomando para ello el argumento `use_box_cox=False`. De forma análoga, dado que la serie a estudio no presenta tendencias claras a corto y largo plazo, se omiten del modelo ambas características (`use_trend=False` y `use_damped_trend=False`). Sin embargo sí se mantiene el modelado de errores mediante un modelo ARMA, y por ello el correspondiente argumento se deja sin precisar. De este modo la decisión de si utilizar o no el modelo ARMA para dar forma a los errores residuales estará condicionada a si su uso permite una mejora de los resultados.

A continuación se muestran las características del modelo obtenido tras el ajuste sobre los datos de entrenamiento, obtenidas tras aplicar la función `summary` sobre el modelo ajustado.

```
Use Box-Cox: False
Use trend: False
Use damped trend: False
Seasonal periods: [ 24. 168.]
Seasonal harmonics [5 3]
ARMA errors (p, q): (0, 0)
Smoothing (Alpha): 1.343351
Seasonal Parameters (Gamma): [ 0.01746634  0.0230017  -0.02343262  0.01269913]
```

```
AR coefficients []
MA coefficients []
Seed vector [ 4.98892643e+00  1.21588639e-01 -1.40627150e-02 -1.13436889e-02
  9.53098373e-03 -1.41501880e-03 -1.51381940e-01 -7.11439341e-03
  1.73225728e-02 -1.27028187e-04 -4.43798047e-03  2.07354056e-01
 -6.07882211e-02  3.34783249e-03 -9.37401339e-02  7.91182756e-02
 -1.81477431e-02]
AIC 60240.717553
```

Manteniendo la notación utilizada en 5.3 para denotar el modelo a partir de su caracterización de parámetros, en el caso a estudio actual el modelo ajustado se corresponde con un modelo `TBATS(0,0,0,0,{24,5},{168,3})`. Nótese que tras haber dejado la responsabilidad de la elección del modelado de los residuos con un modelo ARMA, el algoritmo no ha hecho uso de ellos, lo que implica que su aplicación al presente modelo no producía una mejora los resultados.

La Figura 7.11 muestra la serie de errores residuales calculada a partir de los datos de entrenamiento y de las predicciones sobre realizadas sobre ellos con el modelo TBATS óptimo.

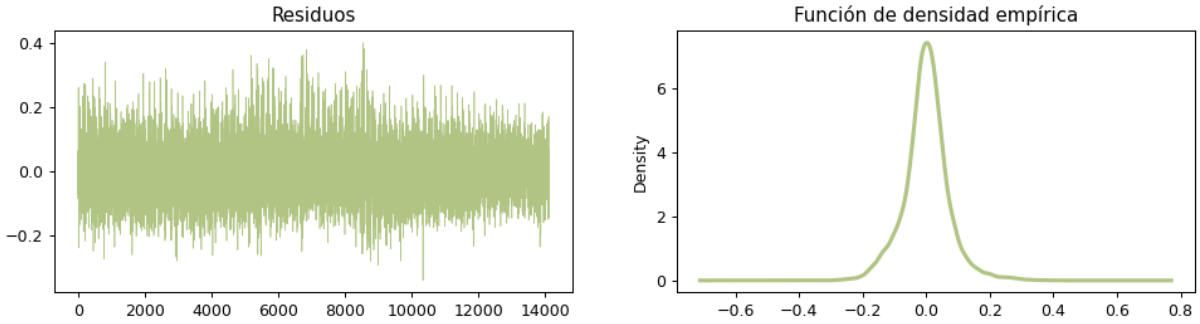


Figura 7.11: Distribución de los residuos del modelo TBATS

En este caso la función de distribución empírica de los residuos no presenta asimetrías en sus colas, y además sigue una distribución aproximadamente normal de media 0. De este hecho se deduce que el modelo TBATS estimado aproxima correctamente los datos de entrenamiento, al menos en el escenario de predicciones sin retroalimentación con tamaños de ventana unitarios para el cual ha sido construido.

El último paso una vez más consiste en realizar las predicciones bajo los cuatro escenarios que vienen utilizándose a lo largo de toda la sección: predicciones con tamaños de ventana 1, 6 y 24, y la predicción sobre el conjunto total de datos de prueba (véase Figura 7.12).

Fíjese que los resultados visualmente se asemejan a los resultados obtenidos en 7.10 obtenidos con el modelo ARIMA con términos de Fourier, ya que ambos modelan la estacionalidad como

combinación lineal de funciones sinusoidales, y por tanto para predicciones largas el comportamiento tiende a estabilizarse en torno a dicho modelo estacional.

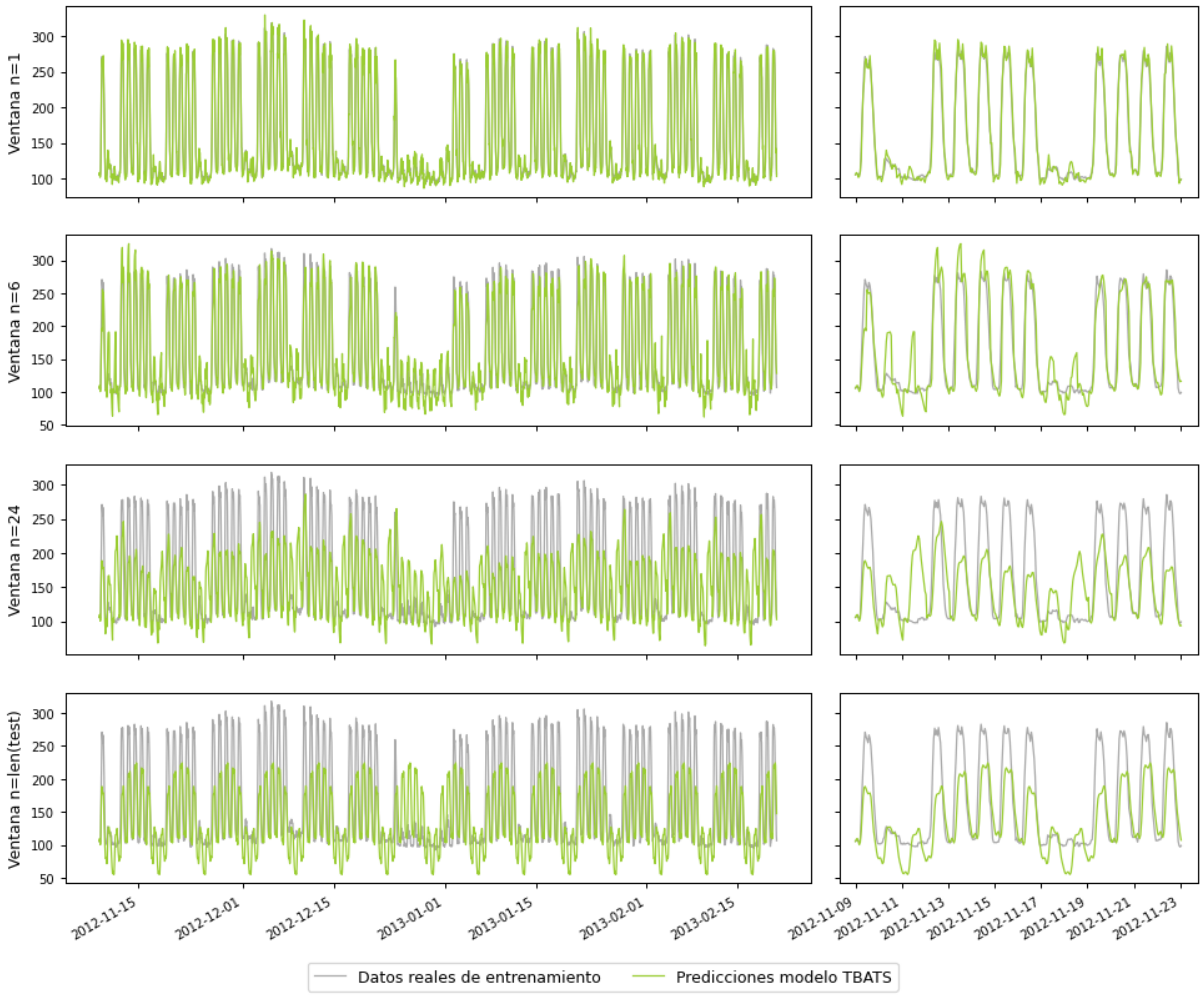


FIGURA 7.12: PREDICCIONES CON EL MODELO TBATS PARA TAMAÑOS DE VENTANA 1, 6 Y 24, JUNTO CON LA PREDICCIÓN SOBRE EL TOTAL DE ELEMENTOS DEL CONJUNTO DE PRUEBA, AMPLIANDO SOBRE LAS GRÁFICAS DERECHAS LAS PREDICCIONES DE LAS DOS PRIMERAS SEMANAS

7.5. Redes LSTM

Para la construcción de las redes neuronales LSTM, independientemente de la arquitectura y de la configuración que se vaya a utilizar, en un primer paso es imprescindible llevar a cabo una **normalización** de los datos de cara a reducir los tiempos de aprendizaje y convergencia de las redes. La normalización no es más que el reescalado de los datos originales de tal forma que los nuevos valores se encuentren en el rango de 0 a 1. Para el presente proyecto se utiliza una

normalización Min-Max, la cual requiere del conocimiento, o en su defecto de la aproximación, de los valores extremos de la serie. Esta normalización vienen expresada por la ecuación

$$x'_t = \frac{x_t - x_{min}}{x_{max} - x_{min}}$$

donde x_{max} y x_{min} representan los valores extremos de la serie. Para su implementación en *Python* se hará uso del objeto `MinMaxScaler` de la librería *scikit-learn* (véase el Código 7.8).

```
# Normalización de valores
values = values.reshape((len(values), 1))
scaler = MinMaxScaler(feature_range=(0, 1))
scaler = scaler.fit(values)
normalized = scaler.transform(values)
```

Código 7.8: Normalización de la serie a estudio

7.5.1. Arquitectura *Vanilla*

Como primera aproximación a las RNN se lleva a cabo la construcción de una red LSTM con arquitectura Vanilla, cuya única variable de entrada es la secuencia de consumos. Antes de comenzar con la construcción del modelo se deben preparar los datos de entrenamiento de cara al ajuste óptimo de la red. Para ello, el primer paso consiste en fijar el tamaño de las secuencias de entrada, para lo que se requiere un conocimiento previo del dominio de aplicación. En este caso es lógico pensar que para llevar a cabo las predicciones se requiera considerar al menos el consumo de varios días previos. Además, dado que se ha estudiado que el mayor periodo de estacionalidad es de 168, parece sensato tomar todos los valores de la última semana para entrenar el modelo, dado que así se da la posibilidad a la red de encontrar patrones dentro de dicho periodo. Por todo ello se fija como tamaño de las secuencias de entrada `input_seq_length=168`.

Para entrenar la red se requiere de dos conjuntos de entrenamiento, `training_x`/`training_y`, construidos a partir de los registros dedicados al entrenamiento de la red. Sendos conjuntos contendrán por un lado las secuencias de entrada de la red, y por otro los valores de salida que se desea predecir para cada una de las secuencias de entrada. Estos últimos serán los valores utilizados por el algoritmo BPTT de entrenamiento del modelo para estimar los gradientes a partir de la diferencia entre las predicciones y los valores reales esperados (véase la sección de entrenamiento de las RNN). Vistos los requisitos necesarios para entrenar el modelo, la construcción de ambos conjuntos se realiza siguiendo un modelo de ventana deslizante sobre la serie de registros de consumos de entrenamiento. En *Python* la construcción se realiza del siguiente modo:

- El conjunto de valores de entrada, `training_x`, contendrá todas las secuencias de tamaño `input_seq_length`, obtenidas siguiendo un esquema de ventanas deslizantes sobre la serie temporal, desplazando la ventana una unidad por desplazamiento y considerando un total de `L-input_seq_length` secuencias, donde `L` denota el número total de elementos del conjunto de entrenamiento, es decir, `L=len(normalized_training_set)`. Este conjunto deberá representarse en *Python* como un tensor[1] de orden 3 con dimensión `(L-input_seq_length, input_seq_length, 1)`.
- El conjunto de salidas, `training_y`, contendrá los consumos asociados al siguiente instante de tiempo de cada una de las secuencias del conjunto `training_x`. Para su representación en *Python* se deberá utilizar una matriz con dimensiones `(L-input_seq_length, 1)`.

La Figura 7.13 muestra la representación gráfica de la serie de registros de entrenamiento (`normalized_training_set`), representada como un array unidimensional de tamaño `L`, junto con el tensor `training_x` y la matriz `training_y`. Nótese que se utiliza la escala de colores para representar el desplazamiento de la ventana sobre la serie temporal para construir los receptivos conjuntos de entrenamiento. Cada una de las fibras 1-modales[2] del tensor `training_x` representa una secuencia de entrada a la red, y la correspondiente fila de la matriz `training_y` representa el siguiente valor real de la secuencia. El algoritmo BPTT itera sobre el conjunto de fibras 1-modales. Si toma como entrada la i-ésima fibra 1-modal (`training_x[i,:,:]`), el algoritmo analiza el error entre el resultado obtenido por la red y el valor de la i-ésima fila de la matriz `training_y[i,:]`, y en función de él lleva a cabo la actualización de los pesos de la red.

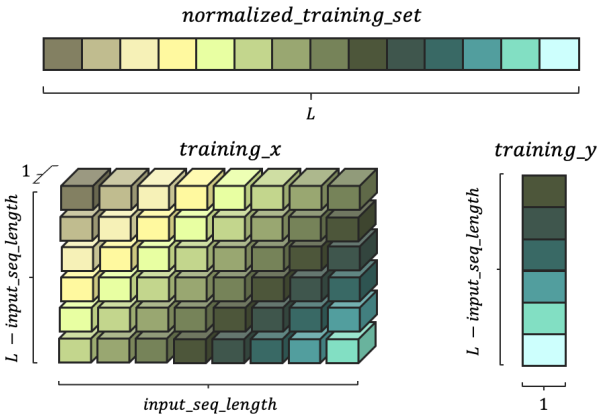


FIGURA 7.13: REPRESENTACIÓN GRÁFICA DEL PROCESO DE CONSTRUCCIÓN DE LOS CONJUNTOS DE ENTRENAMIENTO DE UNA RED NEURONAL VANILLA PARA UNA SERIE TEMPORAL UNIVARIANTE

[1] véase Montalvo (2020) para ampliar los fundamentos sobre tensores

[2] Las fibras 1-modales son subtensores de orden 1 obtenidos al dejar fijos todos los índices del tensor menor el primero. En Python las fibras 1-modales se obtienen haciendo uso de la notación $tensor[i,:,:]$

El fragmento de Código 7.9 muestra el procedimiento seguido en *Python* para construir ambos conjuntos de entrenamiento. En el caso a estudio las dimensiones de estos conjuntos serán `(13953, 168, 1)` para el tensor `training_x` y `(13953,1)` para la matriz `training_y`.

```
# Tamaño de la secuencia de entrada
input_seq_length = 168

# Conjuntos de entrenamiento
training_x = [normalized_training_set[i:i+input_seq_length] for i in
              range(0, len(normalized_training_set)-input_seq_length)]
training_x = np.array(training_x)
    .reshape(len(training_x),input_seq_length,1)

training_y = np.array(normalized_training_set[input_seq_length:])
    .reshape(len(training_x), 1)
```

Código 7.9: División de datos de entrenamiento

El siguiente paso en la construcción de la red neuronal consiste en definir su modelo arquitectónico. En *Keras* la forma de construir el modelo es mediante una secuencia de capas. El contenedor sobre el que se construyen estas capas es un objeto de la clase `Sequential`, y mediante la función `add` se permite añadir diversas capas al modelo. En el presente caso el modelo estará compuesto por una capa LSTM y una capa densa, cuyas construcciones se realizan a partir de los objetos `LSTM` y `Dense` de *Keras*. Para la construcción de la capa LSTM se requiere fijar el tamaño del vector de estado interno, que tal y como se estudió en 5.4.1 coincidirá con el tamaño de la celda de memoria y su valor marcará la capacidad de memoria de la red. Basando su definición en la experimentación, se fijará como tamaño final del vector de estado 10 unidades.

```
# Tamaño del estado interno
hidden_state_length = 10

# LSTM Vanilla
model = Sequential()
model.add(LSTM(units=hidden_state_length,
               activation='tanh',
               recurrent_activation='sigmoid',
               input_shape=(input_seq_length, 1)))

model.add(Dropout(rate=0.3))

model.add(Dense(units=1,
                activation='relu'))
```

Código 7.10: Construcción del modelo Vanilla

Para la configuración de la capa `LSTM` se toma como función de activación de las tres puertas que componen la capa LSTM la función `sigmoide`, y como función de activación de la capa auxiliar de la *input gate* la función `tanh`, tal y como se estudió en la sección 5.4.1. El argumento `units=hidden_state_length` indica el tamaño del vector asociado al estado interno de la red, y `input_shape=(input_seq_length, 1)` indica el tamaño de las fibras 1-modales utilizadas como secuencias de entrada.

Para la capa `Dense`, el argumento `units=1` especifica el tamaño de la salida de la capa, que en este caso es 1 puesto que se desea predecir un único elemento tras procesar toda la secuencia. El segundo argumento `activation='relu'` especifica la función de activación.

Entre ambas capas se ha añadido un nuevo elemento llamado `Dropout`. Este elemento tiene como fin ajustar un factor de olvido (*dropout*) en la entrada de la capa densa. Este factor se utiliza para reducir el sobreajuste de la red, y su forma de actuar consiste en desactivar aleatoriamente por cada nueva entrada en la red en la fase de entrenamiento un porcentaje de las neuronas de entrada, acorde con el factor de olvido especificado. Con este procedimiento se consigue que ninguna neurona "memorice" parte de la entrada, reduciendo así el riesgo de sobreajuste. En este caso se toma como factor de olvido el valor `rate=0.2`.

Una vez se ha construido el modelo, el siguiente paso es compilarlo. La compilación del modelo es un paso orientado a la eficiencia, ya que transforma la secuencia de capas previamente definidas en un conjunto de matrices y tensores muy eficientes y orientados a su ejecución en CPU o GPU, dependiendo de la configuración. La compilación del modelo requiere de la especificación de ciertos parámetros adaptados específicamente a las características de la red neuronal. Para llevar a cabo la compilación se debe aplicar la función `compile` sobre el propio modelo ya construido (véase el Código 7.11)

```
1 # Compilación del modelo
2 model.compile(loss='mse',
3                optimizer='adam')
4 model.summary()
```

CÓDIGO 7.11: COMPILACIÓN DEL MODELO VANILLA

Los argumentos que recibe la función `compile` para su aplicación sobre el presente modelo son `loss='mse'` para utilizar el Error Cuadrático Medio (MSE, *Mean Squared Error*) como función de pérdida del modelo a utilizar por el algoritmo BPTT para estimar de forma iterativa su gradiente, y `optimizer='adam'` como algoritmo específico para la optimización eficiente de los parámetros de la red. El algoritmo *Adam* (abreviatura de *Adaptive Moment Estimation*) (Kingma & Ba, 2014) es una variante del algoritmo de Descenso del Gradiente Estocástico (SGD, acrónimo del

término inglés *Stochastic Gradient Descent*), utilizado para actualizar los parámetros del modelo a partir del gradiente estimado de la función de pérdida.

Tras la compilación se ejecuta la función `summary` sobre el modelo para obtener un resumen de sus capas y sus correspondientes parámetros. A continuación se muestra la salida concreta obtenida con el modelo que se ha ido construyendo a lo largo de la sección.

```
Model: "sequential"

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 10)                480
_________________________________________________________________
dropout (Dropout)            (None, 10)                0
_________________________________________________________________
dense (Dense)                (None, 1)                 11
=================================================================

  Total params: 491
  Trainable params: 491
  Non-trainable params: 0
```

A partir de lo estudiado en la sección 5.4.1 acerca de la estructura interna de la celda LSTM y de las capas densas es posible deducir la procedencia de los 491 parámetros que componen el modelo. Dado que únicamente se ha considerado como variable de entrada el consumo de energía y además el tamaño del vector de estado interno se ha fijado en 10, la entrada de cada una de las capas internas de la red estará constituida por un vector con 11 posiciones $(1 + 10)$ y la salida será de 10 elementos, lo que hace un total en cada capa de $11 \times 10 = 110$ pesos y 10 sesgos, esto es, $W_f, W_i, W_o, W_g \in \mathbb{R}^{11\times10}$ y $b_f, b_i, b_o, b_g \in \mathbb{R}^{10}$. El total de parámetros de la capa LSTM es por tanto 480, que coincide con la salida de la función `summary`. Por su parte, la capa *Dense* toma como entrada el último vector de estado devuelto por la LSTM (véase la Figura 5.18), y como salida ha sido configurada para devolver un único elemento. Por tanto la capa densa tendrá un total de $10 \times 1 = 10$ pesos y 1 sesgo, obteniendo el total de los 11 parámetros buscados.

Tras haber construido el modelo y tener listos los conjuntos de registros de consumos de energía eléctrica para el entrenamiento, el último paso antes de la realización de las predicciones es el entrenamiento del modelo. El entrenamiento del modelo se lleva a cabo siguiendo el algoritmo BBPT y haciendo uso del algoritmo de optimización y la función de pérdida fijados durante el proceso de compilación del modelo (*Adam* y MSE en este caso). Para configurar el entrenamiento del modelo es importante tener presente dos conceptos comunes en el entrenamiento de cualquier red neuronal: ***epoch*** y ***batch***. El algoritmo de retropropagación requiere la red sea entrenada un

determinado número de veces sobre el total del conjunto de datos de entrenamiento. Este número se conoce como número de épocas de entrenamiento. A su vez, cada una de las épocas puede dividirse en grupos de pares de entradas y salidas de la red, llamados lotes (*batches*). El tamaño de estos lotes determina el número de secuencias que procesa la red antes de que se actualicen los pesos siguiendo el algoritmo de actualización de parámetros preestablecido.

Para el entrenamiento del modelo en *Keras* se debe hacer uso de la función `fit` de la clase `Sequential`, con argumentos `x=training_x` e `y=training_y` los correspondientes conjuntos de entrenamiento del modelo, junto con el número de épocas (`epochs`), y el tamaño del *batch* (`batch_size`). La función toma un último argumento, `verbose=1`, que hace referencia al tipo de salida dinámica mostrada durante el proceso de entrenamiento. El valor 1 permite visualizar una barra de progreso en cada una de las épocas junto con el valor de perdida acumulada durante el entrenamiento, el valor del argumento igual a 2 hará que se muestre únicamente el valor de pérdida, mientras que el valor del argumento a 0 anulará la salida dinámica (véase el Código 7.12).

```
epochs = 10
batch_size = 16

# Entrenamiento del modelo
model.fit(x=training_x,
          y=training_y,
          epochs=epochs,
          batch_size=batch_size,
          verbose=1)
```

CÓDIGO 7.12: ENTRENAMIENTO DEL MODELO VANILLA

A continuación se muestra la salida obtenida durante el proceso de entrenamiento, en la cual se puede observar cómo el error disminuye a medida que aumenta el número de épocas entrenadas.

```
Epoch 1/10
  3489/3489 [==============================] - 239s 68ms/step - loss: 0.0232
Epoch 2/10
  3489/3489 [==============================] - 268s 77ms/step - loss: 0.0085
Epoch 3/10
  3489/3489 [==============================] - 258s 74ms/step - loss: 0.0068
Epoch 4/10
  3489/3489 [==============================] - 238s 68ms/step - loss: 0.0065
Epoch 5/10
  3489/3489 [==============================] - 230s 66ms/step - loss: 0.0061
```

```
Epoch 6/10
  3489/3489 [==============================] - 216s 62ms/step - loss: 0.0060
Epoch 7/10
  3489/3489 [==============================] - 220s 63ms/step - loss: 0.0059
Epoch 8/10
  3489/3489 [==============================] - 235s 67ms/step - loss: 0.0059
Epoch 9/10
  3489/3489 [==============================] - 230s 66ms/step - loss: 0.0059
Epoch 10/10
  3489/3489 [==============================] - 247s 71ms/step - loss: 0.0057
```

Una vez entrenado el modelo es posible analizar el grado de aproximación del modelo estudiando los errores residuales tal y como viene haciéndose con todos los modelos previos. La Figura 7.14 muestra la distribución de la serie de errores residuales.

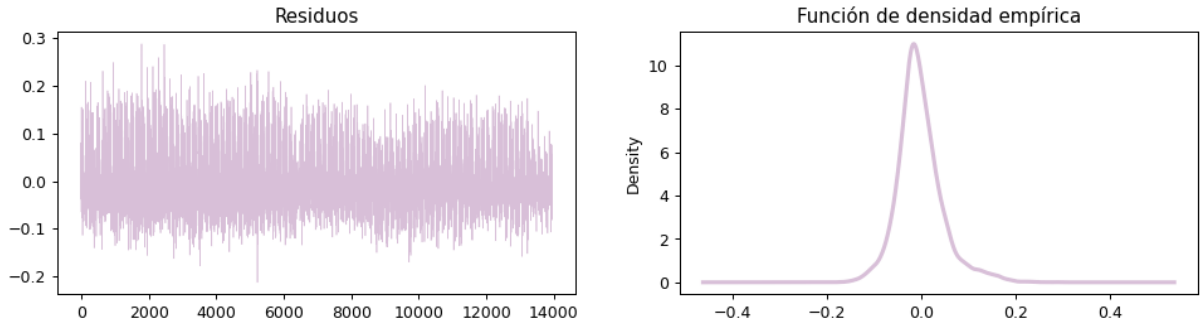


Figura 7.14: Distribución de los residuos de la red LSTM con arquitectura Vanilla

Una vez más mediante esta técnica de visualización de los residuos es posible deducir que el modelo aproxima correctamente los datos de entrenamiento, dado que la distribución de la serie de residuos se asemeja a una distribución normal de media 0 y varianza finita. Sin embargo se debe tener presente que este análisis de los errores hace referencia al modelo de predicciones con ventana unitaria, es decir, un modelo de predicción de únicamente el siguiente valor de la serie. Tal y como se verá a continuación este modelo sin embargo no ajusta correctamente para predicciones con tamaños de ventana grandes, pero no por ello este análisis de residuos ha de aportar malos resultados.

Ya por último, tras todo el proceso de modelado, construcción, entrenamiento y validación de la red, es posible realizar las predicciones deseadas. Una vez más se mantiene el enfoque de ventanas móviles con retroalimentación, siguiendo para este caso el esquema de la Figura 5.19.

En la Figura 7.15 se muestran las predicciones realizadas con la red construida para los tamaños de ventana 1, 6 y 24, así como la predicción sobre la totalidad del conjunto de prueba.

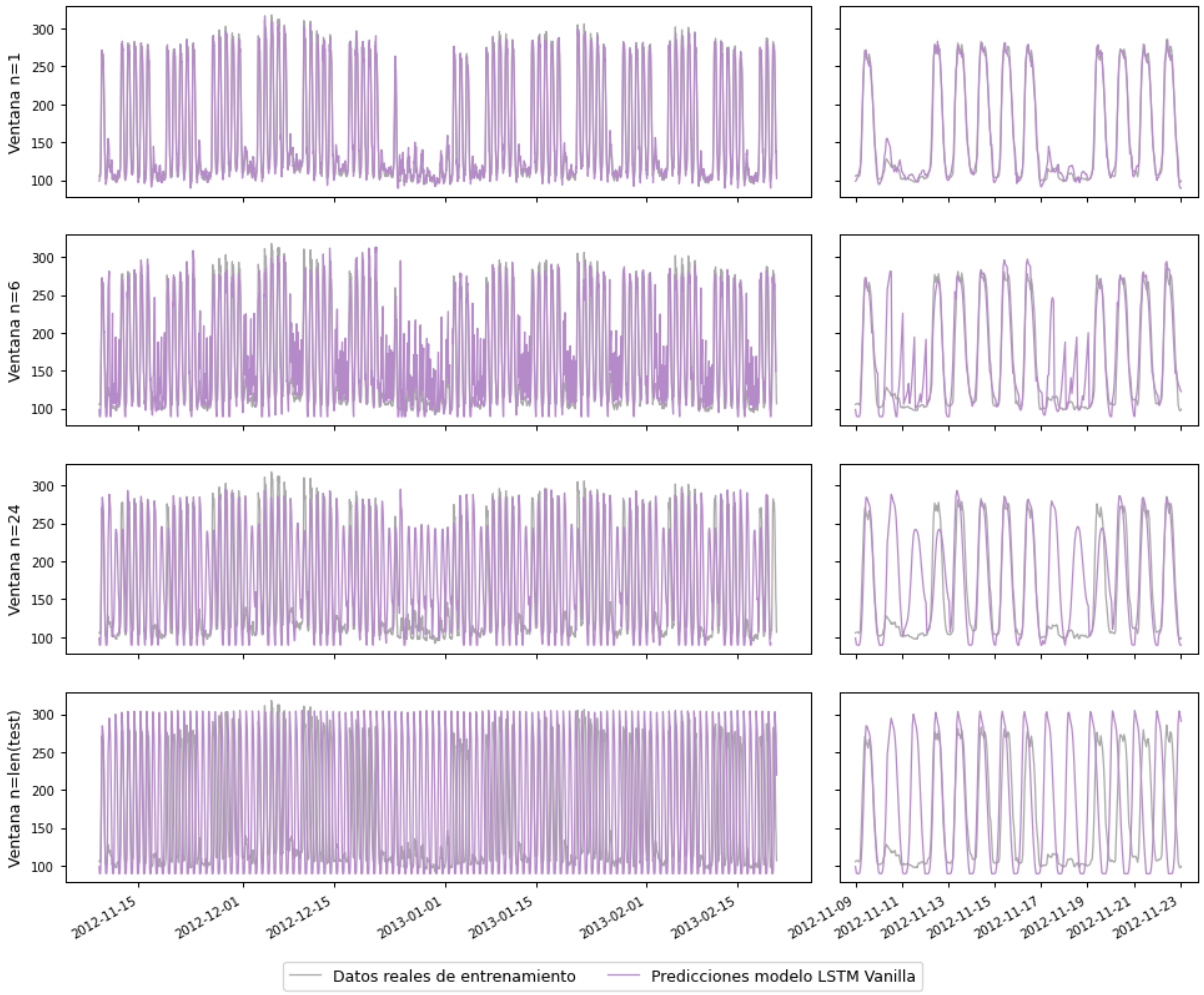


FIGURA 7.15: PREDICCIONES CON LA RED LSTM CON ARQUITECTURA VANILLA PARA TAMAÑOS DE VENTANA 1, 6 Y 24, JUNTO CON LA PREDICCIÓN SOBRE EL TOTAL DE ELEMENTOS DEL CONJUNTO DE PRUEBA, AMPLIANDO SOBRE LAS GRÁFICAS DERECHAS LAS PREDICCIONES DE LAS DOS PRIMERAS SEMANAS

Los resultados obtenidos distan mucho de lo que cabía esperar al estar aplicando una red neuronal, al menos en el caso de predicciones a largo plazo. A continuación se muestran variaciones en la construcción de la red realizadas con el objetivo de mejorar estos resultados.

7.5.2. Arquitectura *Vanilla* con variables exógenas

La siguiente construcción a realizar con vistas a mejorar las predicciones de la red anterior va a consistir en añadir a la estructura de la red variables exógenas con un componente temporal que permitan asociar los consumos de energía con características temporales. Los consumos de energía eléctrica, tal y como se ha estudiado en el análisis de las estacionalidades (véase la sección 6.4), poseen un comportamiento ligado a la hora del día y al día de la semana en el que han

ocurrido, que de hecho es lo que da pie a la existencia de las estacionalidades diaria y semanal. No obstante parece lógico pensar también que el día del mes o el respectivo mes del año pueden influir en el consumo, y de hecho en muchos casos así es. Por estos motivos las auxiliares a considerar en el presente proyecto son el **día de la semana**, el **día del mes**, el **mes** y la **hora** de los registros de consumo. Nótese que los valores que toman estas variables temporales son conocidos para cualquier instante de tiempo, lo que permite definirías no sólo como variables ligados al consumo de energía, sino de hecho como variables exógenas.

De nuevo esta vez es necesario construir los conjuntos `training_x` y `training_y` con los registros de entrenamiento, sin embargo esta vez es necesario considerar las observaciones de las variables exógenas como elementos del tensor `training_x`. Tal y como puede observarse en la Figura 7.16, en lugar de considerar las fibras 1-modales como secuencias de entrada de la red, se consideran las secciones horizontales[3] del tensor.

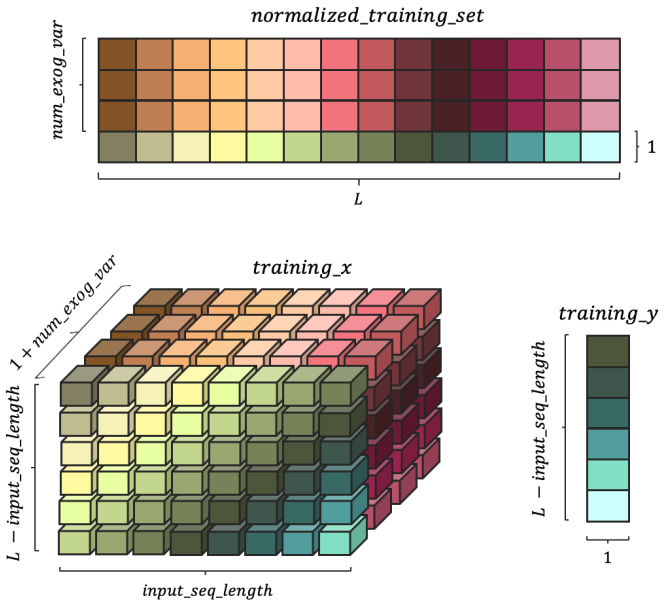


Figura 7.16: Representación gráfica del proceso de construcción de los conjuntos de entrenamiento de una red neuronal Vanilla para una serie temporal univariante con variables exógenas auxiliares

Nótese que en la figura no se hace distinción de escala de colores entre variables exógenas, lo cual no debe llevar a confusión. Simplemente se mantiene el mismo color por no dificultar su com-

[3] Las secciones son subtensores de orden 2 (matrices), obtenidos al fijar todos los índices del tensor menos dos de ellos. Las secciones de un tensor de orden 3, en función del índice fijado, se denominan secciones horizontales (fijando el primer índice, $T[i,:,:]$), verticales (fijando el segundo índice, $T[:,j,:]$) y frontales (fijando el tercer índice, $T[:,:,k]$).

prensión, pero no porque sus valores sean iguales. Puede observarse que el conjunto `training_y` permanece invariable con respecto a la Figura 7.13, dado que el objetivo de predicción perseguido sigue siendo el mismo. La diferencia está en las secuencias de entrada. En este caso las variables de entrada x_t tendrán como dimensión la suma del número de variables de la serie temporal a estudio (que podrá ser 1 si la serie es univariante o > 1 en el caso multivariante) y el número de variables exógenas. Para el presente caso el total de variables de entrada será 5 (1 variable con los consumos de energía y las 4 variables exógenas).

El fragmento de Código 7.13 muestra el proceso de reestructuración de los datos de entrenamiento para construir los conjunto `training_x` y `training_y`. Con respecto a la división realizada en 7.9 la diferencia radica en el manejo de la variable `normalized_training_set`, puesto que en este caso ha pasado de ser un array unidimensional a contener los valores de un total de 5 variables, tratándose así de un array bidimensional.

```
# Tamaño de la secuencia de entrada
input_seq_length = 84

# Conjuntos de entrenamiento
training_x = [normalized_training_set[:][i:i+input_seq_length] for i in
              range(0, len(normalized_training_set)-input_seq_length)]
training_x = np.array(training_x)
    .reshape(len(training_x),input_seq_length,5)

training_y = np.array(normalized_training_set[input_seq_length:,0])
    .reshape(len(training_x), 1)
```

Código 7.13: División de datos de entrenamiento con variables exógenas

En esta construcción se ha utilizado un tamaño de secuencia de entrada de 84 registros, correspondientes a media semana. En el momento de su construcción este hecho vino propiciado por los malos resultados a largo plazo obtenidos con el modelo previo. Por este motivo se ha tratado de utilizar una configuración distinta y así poder contrastar también ambos enfoques.

Para la construcción del modelo se mantiene la estructura implementada en 7.10, a excepción del tamaño de la secuencias de entrada, dado que en este caso el argumento que recibe el constructor del objeto `LSTM` será `input_shape=(input_seq_length,5)` para adaptar el tamaño de la entrada al número total de variables que intervienen. Mediante la experimentación se ha obtenido que los mejores resultados han sido alcanzados con un vector de estado interno con tamaño 12, por lo que la implementación del modelo deberá realizarse para el valor `hidden_state_length = 12`.

El proceso de compilación del modelo es idéntico al realizado en 7.11, manteniéndose el Error Cuadrático Medio como función de error de predicción y el optimizador *Adam* como algorit-

mo de actualización de parámetros del modelo. A continuación se muestra el resumen con la configuración final del modelo:

```
Model: "sequential"

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 12)                864
_________________________________________________________________
dropout (Dropout)              (None, 12)                  0
_________________________________________________________________
dense (Dense)                (None, 1)                 13
=================================================================

  Total params: 877
  Trainable params: 877
  Non-trainable params: 0
```

En este caso las dimensiones de las matrices de pesos y los vectores de sesgos han sido $W_f, W_i, W_o, W_g \in \mathbb{R}^{17\times 12}$ y $b_f, b_i, b_o, b_g \in \mathbb{R}^{12}$ para la capa `LSTM`, y $W_y \in \mathbb{R}^{12}$ y $b_y \in \mathbb{R}$ para la capa `Dense`, lo que hace un total de 877 parámetros.

Con respecto al entrenamiento se han tomado un total de 15 épocas segmentadas en lotes de 4, los cuales se ha configurado en la función `fit` del modelo gracias a los argumentos `epochs=15` y `batch=4`. El resultado obtenido con el entrenamiento del modelo ha sido:

```
Epoch 1/15
  3510/3510 [==============================] - 129s 37ms/step - loss: 0.0483
Epoch 2/15
  3510/3510 [==============================] - 132s 38ms/step - loss: 0.0128
Epoch 3/15
  3510/3510 [==============================] - 101s 29ms/step - loss: 0.0097
Epoch 4/15
  3510/3510 [==============================] - 98s 28ms/step - loss: 0.0087
Epoch 5/15
  3510/3510 [==============================] - 100s 29ms/step - loss: 0.0081
Epoch 6/15
  3510/3510 [==============================] - 104s 30ms/step - loss: 0.0076
Epoch 7/15
  3510/3510 [==============================] - 97s 28ms/step - loss: 0.0074
Epoch 8/15
  3510/3510 [==============================] - 95s 27ms/step - loss: 0.0073
```

```
Epoch 9/15
  3510/3510 [==============================] - 98s 28ms/step - loss: 0.0071
Epoch 10/15
  3510/3510 [==============================] - 101s 29ms/step - loss: 0.0069
Epoch 11/15
  3510/3510 [==============================] - 122s 35ms/step - loss: 0.0069
Epoch 12/15
  3510/3510 [==============================] - 120s 34ms/step - loss: 0.0070
Epoch 13/15
  3510/3510 [==============================] - 123s 35ms/step - loss: 0.0069
Epoch 14/15
  3510/3510 [==============================] - 143s 41ms/step - loss: 0.0066
Epoch 15/15
  3510/3510 [==============================] - 123s 35ms/step - loss: 0.0065
```

Tras entrenar el modelo el siguiente paso una vez más consiste en analizar la serie de errores residuales y su correspondiente función de densidad empírica, con la finalidad de llevar a cabo una primera validación sobre el grado de aproximación del modelo construido (véase Figura 7.17).

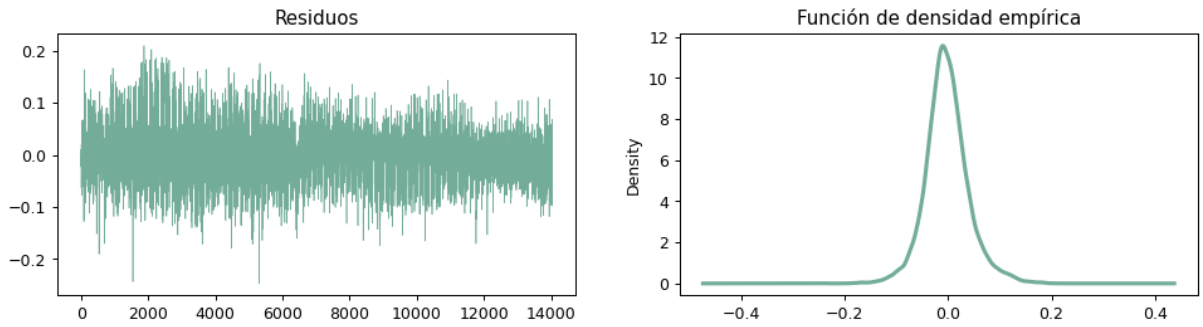


FIGURA 7.17: DISTRIBUCIÓN DE LOS RESIDUOS DE LA RED LSTM CON ARQUITECTURA VANILLA Y USO DE VARIABLES EXÓGENAS

Puede observarse que el comportamiento de los residuos puede aproximarse por un distribución normal centrada en 0, lo cual una vez más es un buen indicador de que el modelo aproxima correctamente los datos a estudio.

Ya sí, tras disponer del modelo entrenado, el siguiente paso consiste en la realización de predicciones (véase Figura 7.18). A diferencias de todos los modelos previos, en este caso la aproximación realizada, tanto para el caso de un tamaño de ventana de 24 como para las predicciones sobre el total de registros de prueba, los resultados se aproximan bastante a los valores reales. Al menos en estos casos la componente estacional del modelo refleja fielmente la verdadera estacionalidad de la serie, si bien la red sigue sin ser capaz de detectar el periodo vacacional entre diciembre y enero para el modelo de predicción total.

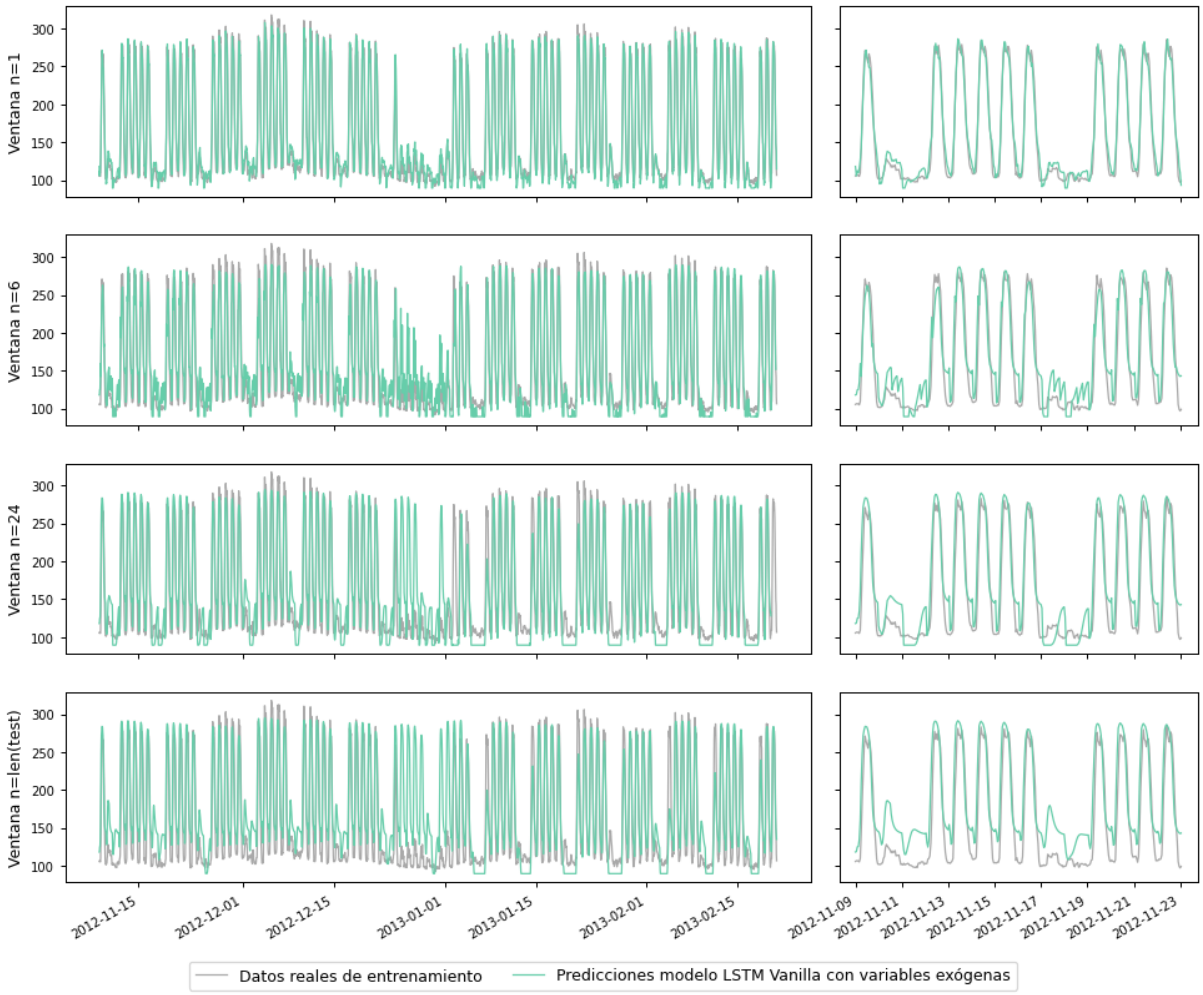


Figura 7.18: Predicciones con la red LSTM con arquitectura Vanilla y variables exógenas para tamaños de ventana 1, 6 y 24, junto con la predicción sobre el total de elementos del conjunto de prueba, ampliando sobre las gráficas derechas las predicciones de las dos primeras semanas

7.5.3. Arquitectura *Encoder-Decoder* con variables exógenas

Para concluir el presente capítulo se va a llevar a cabo la construcción de una red LSTM con arquitectura *Encoder-Decoder*, a fin de obtener un modelo propio de predicción *seq2seq*. El principal objetivo de este modelo será mejorar las predicciones a largo plazo. Dada la gran mejora de resultados que se ha obtenido con la red LSTM Vanilla con variables exógenas con respecto a su versión sin estas variables, la arquitectura *Encoder-Decoder* que se va a considerar contendrá ya desde el primer momento variables exógenas en la entrada tanto de la capa de codificación como en la capa de decodificación (véase la Figura 5.22).

La construcción de estos modelos requiere fijar tanto el tamaño de las secuencias de entrada como el tamaño de las secuencias de salida. Por este motivo la construcción de estos modelos y su posterior aplicación para la realización de predicciones no podrá seguir el mismo enfoque que el realizado en el resto de casos previos, puesto que estos modelos son entrenados para un tamaño de salida fijo y no admiten un enfoque de retroalimentación. Dado este posible inconveniente todas las implementaciones que es necesario realizar (construcción de los conjuntos de entrenamiento, construcción del modelo, etc.) se encapsularán en métodos, que generalmente tendrán como parámetros de entrada el tamaño de las secuencias de entrada y salida.

Tal y como puede derivarse de la Figura 5.22, para el entrenamiento de la red harán falta tres conjuntos de datos:

- Un conjunto `encoder_x` con las secuencias de entrada a la capa de codificación (*encoder*), incluyendo en ellas los valores de las variables exógenas junto con las observaciones de la serie temporal.
- Un conjunto `decoder_x` con las secuencias de entrada a la capa de decodificación (*decoder*) únicamente compuestas por los valores de las variables exógenas en los instantes de tiempo relativos a la secuencia de salida.
- Un conjunto `decoder_y` con las secuencias de salida esperadas para cada una de las secuencias de entrada.

La implementación realizada de la red para esta arquitectura requiere que los conjuntos construidos sean tensores (véase Figura 7.19). Las secciones horizontales del tensor `encoder_x` contendrán las secuencias de entrada, con tamaño `input_seq_length`. El número total de secuencias disponibles para el entrenamiento de la red será igual a `L-input_seq_length-output_seq_length+1`, donde `L` denota el número total de registros de entrenamiento. Por su parte las secciones horizontales de los tensores `decoder_x` y `decoder_y` contendrán las secuencias de salida de las variables exógenas y las secuencias de valores reales de salida, respectivamente, ambos con una longitud igual a `output_seq_length`.

En la Figura 7.19 una vez más se ha utilizado un gradiente de colores para facilitar la comprensión del proceso de construcciones de las secuencias de entrenamiento. De nuevo no se ha hecho distinción entre los colores de las diversas variables exógenas, lo cual no debe llevar a confusión dado que las variables exógenas no son todas iguales.

El fragmento de Código 7.14 muestra la implementación en *Python* del proceso de construcción de estos conjuntos de entrenamiento.

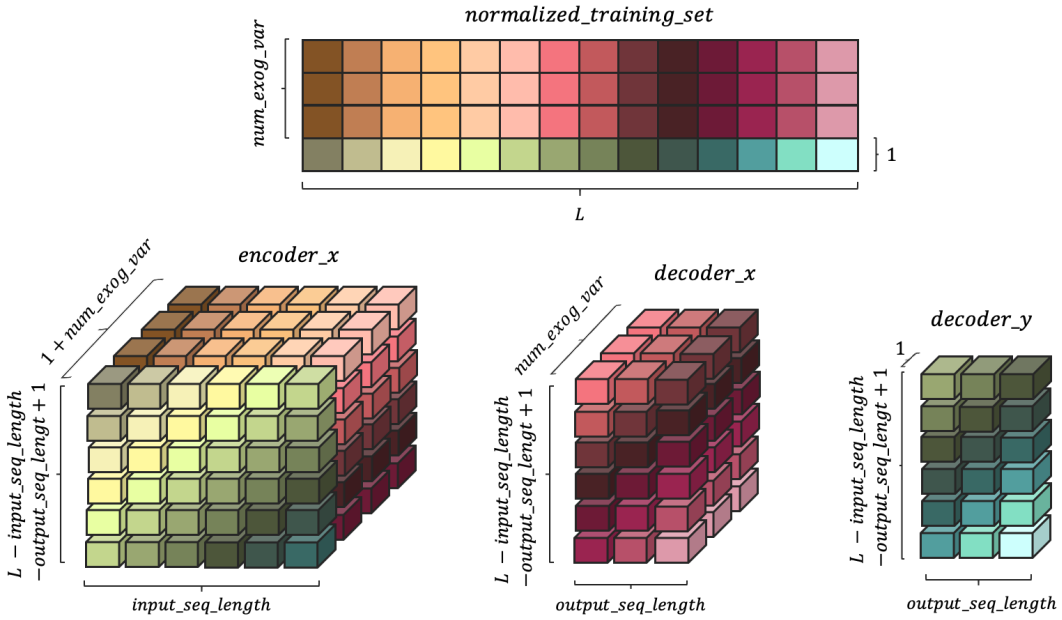


Figura 7.19: Representación gráfica del proceso de construcción de los conjuntos de entrenamiento de una red neuronal LSTM Encoder-Decoder para una serie temporal univariante con variables exógenas auxiliares

```python
def split_train_data(n_inputs, n_outputs):

    L = len(normalized_training_set)

    # Encoder input data
    encoder_x = [normalized_training_set[:][i:i+n_inputs] for i in
        range(L-n_inputs-n_outputs+1)]
    encoder_x = np.array(encoder_x).reshape(len(encoder_x), n_inputs, 5)

    # Decoder input data
    decoder_x = [normalized_training_set[i:i+n_outputs,1:] for i in
        range(n_inputs, L-n_outputs+1)]
    decoder_x = np.array(decoder_x).reshape(len(decoder_x), n_outputs, 4)

    # Decoder target data
    decoder_y = [normalized_training_set[i:i+n_outputs,0] for i in
        range(n_inputs, L-n_outputs+1)]
    decoder_y = np.array(decoder_y).reshape(len(decoder_y), n_outputs, 1)

    return (encoder_x, decoder_x, decoder_y)
```

Código 7.14: División de datos de entrenamiento con variables exógenas para la arquitectura Encoder-Decoder

Para la construcción de la red LSTM *Encoder-Decoder* con variables exógenas no es posible utilizar la estructura secuencial propuesta en las secciones previas haciendo uso de la clase `Sequential` de *Keras*. Esto es debido a que en este caso es necesario utilizar entradas en dos capas consecutivas de la red, además de ser necesario manejar el estado interno de la capa *encoder* para poder llevarlo como estado interno inicial de la capa *decoder*. Por este motivo el contenedor utilizado para organizar las capas de la red será un objeto de la clase `Model`. El Código 7.15 muestra la implementación genérica de la red LSTM *Encoder-Decoder* encapsulada en la función `define_model`.

```
def define_model(n_inputs, n_outputs, n_units):

    # Encoder
    encoder_inputs = Input(shape=(n_inputs, 5))
    encoder_lstm = LSTM(n_units, return_state=True)
    encoder_outputs, state_h, state_c = encoder_lstm(encoder_inputs)

    # Decoder
    decoder_inputs = Input(shape=(n_outputs, 4))
    decoder_lstm = LSTM(n_units, return_sequences=True)

    decoder_outputs = decoder_lstm(decoder_inputs, initial_state=[state_h,
        state_c])
    decoder_outputs = Dense(n_units, activation='relu')(decoder_outputs)
    decoder_outputs = Dense(1, activation='relu')(decoder_outputs)

    model = Model(inputs=[encoder_inputs, decoder_inputs],
        outputs=decoder_outputs)

    model.compile(loss='mse', optimizer='adam')

    return model
```

CÓDIGO 7.15: CONSTRUCCIÓN DEL MODELO ENCODER-DECODER CON VARIABLES EXÓGENAS

La función `define_model` toma como entrada el tamaño de la secuencia de entrada (`n_inputs`), el tamaño de la secuencia de salida (`n_outputs`) y el tamaño del vector de estado interno de la red (`n_units`).

Para construir la capa *encoder* se construye una capa de entrada `Input` sobre la que se indica el tamaño de las secuencias de entrada (en este caso `shape=(n_inputs,5)` dado que hay 4 variables exógenas más la variable de consumos). A continuación se define la capa de entrada con la configuración por defecto (que se corresponde con la configuración realizada en las construcciones previas en cuanto al tipo de funciones de activación se refiere), haciendo uso

del argumento `return_state=True` para indicar que la red debe devolver el estado interno tras procesar el último elemento de la secuencia. Tras ello se aplica sobre la capa LSTM la capa de entrada, lo que da como resultado una salida (`encoder_outputs`), el vector de estado oculto (`state_h`) y la celda de memoria (`state_c`).

Para la construcción de la capa *decoder* de nuevo se construye una capa de tipo `Input`, pero esta vez la dimensión será `shape=(n_outputs, 4)`, dado que las entradas únicamente se componen de valores de variables exógenas y no de consumos. Tras ello se construye la capa `LSTM` con argumento `return_sequences=True` para indicar que se requieren las salidas de la capa en cada instante de tiempo de la secuencia de salida. A continuación se aplica la capa de entrada a la capa LSTM, inicializando esta última con el estado interno devuelto por la capa de *encoder*, esto es, `initial_state=[state_h, state_c]`.

El último paso reside en procesar cada una de las salidas de la secuencia por una capa Densa. En este caso a raíz de los buenos resultados obtenidos por experimentación, se utilizan dos capas de tipo `Dense` a la salida, una con entrada y salida de tamaños igual al vector de estado interno de la red a fin de ponderar cada una de las salidas, y una segunda que devuelve únicamente un elemento, correspondiente ya sí al consumo de energía del instante de tiempo procesado.

A la vista de querer mejorar los resultados de predicción principalmente para tamaños de ventana grandes, es decir, mejorar las predicciones a largo plazo, se llevará a cabo la construcción de dos redes LSTM *Encoder-Decoder* con tamaños de secuencia de salida igual a 24 y al número total de registros de *test*, esto es, 2493.

Para el tamaño de secuencia de salida igual a 24, y haciendo uso de un tamaño de secuencia de entrada de 168 y de un vector de estado interno de tamaño 12, el resumen del modelo con los parámetros para cada una de las capas es el siguiente:

```
Model: "functional_1"


__________________________________________________________________________
Layer (type)                 Output Shape              Param #    Connected to
==========================================================================
input_1 (InputLayer)         [(None, 168, 5)]          0
__________________________________________________________________________
input_2 (InputLayer)         [(None, 24, 4)]           0
__________________________________________________________________________
lstm_1 (LSTM)                [(None, 12), (None,  864       input_1[0][0]
__________________________________________________________________________
lstm_2 (LSTM)                (None, 24, 12)            816       input_2[0][0]
                                                                 lstm_1[0][1]
                                                                 lstm_1[0][2]
```

```
__________________________________________________________________________
dense_1 (Dense)          (None, 24, 12)        156       lstm_2[0][0]
__________________________________________________________________________
dense_2 (Dense)          (None, 24, 1)         13        dense_1[0][0]
==========================================================================

  Total params: 1,849
  Trainable params: 1,849
  Non-trainable params: 0
```

Para el entrenamiento de la red se requieren dos conjuntos de entrada (`encoder_x` y `decoder_x`), para lo cual la función `fit` deberá tomar como argumento `x=[encoder_x, decoder_x]`, esto es, una lista obtenida de la concatenación de ambos tensores.

Entrenando el presente modelo un total de 5 épocas con un tamaño de *batch* igual a 4, el resultado de las predicciones de secuencias de salida de 24 mejora notablemente con respecto a los resultados obtenidos con los modelos previos (véase Figura 7.20).

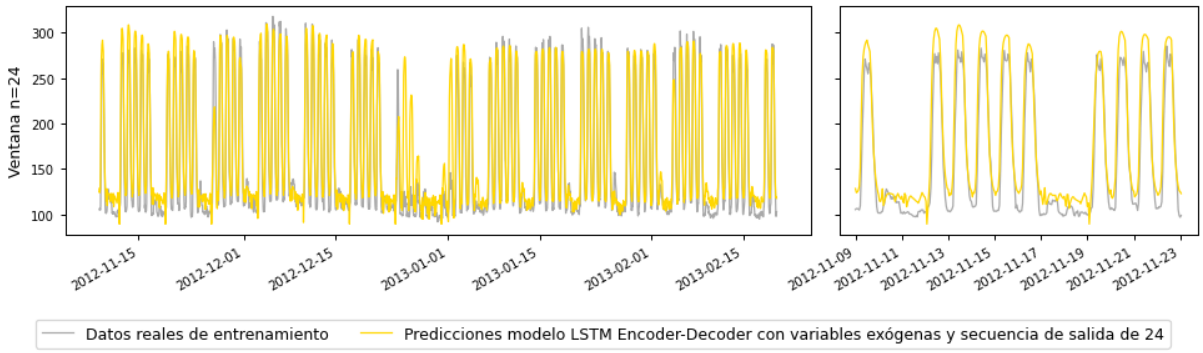


FIGURA 7.20: PREDICCIONES CON LA RED LSTM CON ARQUITECTURA ENCODER-DECODER Y VARIABLES EXÓGENAS PARA SECUENCIAS DE SALIDA DE TAMAÑO 24, AMPLIANDO SOBRE LA GRÁFICA DERECHA LAS PREDICCIONES DE LAS DOS PRIMERAS SEMANAS

De forma análoga se considera el modelo con tamaños de entrada y salida de 168 y 2493 respectivamente y tamaño del vector de estado interno 17, dando así como resultado una red neuronal con la siguiente configuración:

```
Model: "functional_2"


__________________________________________________________________________
Layer (type)                 Output Shape              Param #    Connected to
==========================================================================
input_1 (InputLayer)         [(None, 168, 5)]          0
```

```
_________________________________________________________________
input_2 (InputLayer)        [(None, 2493, 4)]         0
_________________________________________________________________
lstm_1 (LSTM)               [(None, 17), (None,       1564        input_1[0][0]
_________________________________________________________________
lstm_2 (LSTM)               (None, 2493, 17)          1496        input_2[0][0]
                                                                  lstm_1[0][1]
                                                                  lstm_1[0][2]
_________________________________________________________________
dense_1 (Dense)             (None, 2493, 17)          306         lstm_2[0][0]
_________________________________________________________________
dense_2 (Dense)             (None, 2493, 1)           18          dense_1[0][0]
=================================================================

  Total params: 3,384
  Trainable params: 3,384
  Non-trainable params: 0
```

Tras entrenar esta segunda red, es posible ya predecir los valores de consumo durante las 2493 horas que constituyen el conjunto de *test*, sin ningún tipo de retroalimentación ni conocimiento sobre los nuevos valores de prueba, únicamente introduciendo como secuencia de entrada a la red los últimos 168 consumos del conjunto de entrenamiento e introduciendo los valores de las variables exógenas relativas a los 2493 instantes de tiempo futuros. Tal y como puede observarse en la Figura 7.21, los resultados son extremadamente satisfactorios dado que las predicciones son capaces incluso de adaptarse al periodo festivo sin perjudicar a los consumos siguientes.

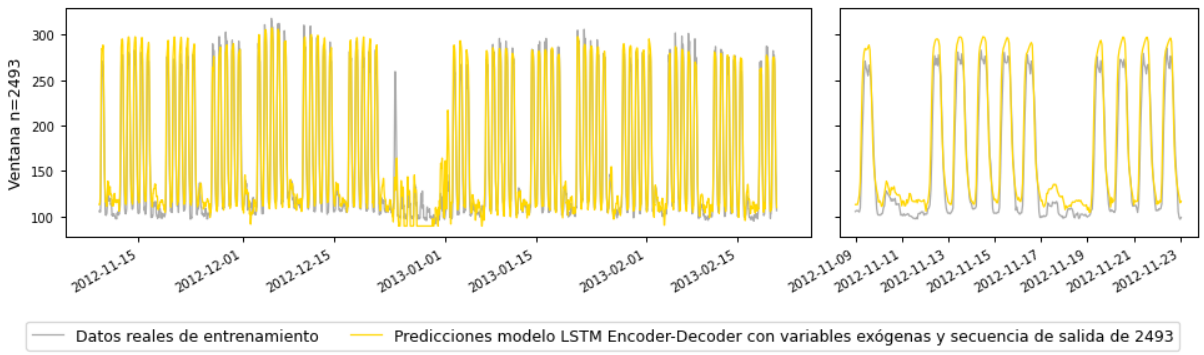


Figura 7.21: Predicciones con la red LSTM con arquitectura Encoder-Decoder y variables exógenas para secuencias de salida de tamaño 2493 (todo el conjunto de prueba), ampliando sobre la gráfica derecha las predicciones de las dos primeras semanas

Capítulo 8

Evaluación de resultados

8.1. Métricas de evaluación

Para analizar el grado de exactitud en las predicciones con los modelos que han sido desarrollados se requiere del uso de varias métricas de evaluación de errores. Para el desarrollo del capítulo se utiliza la siguiente notación:

y_t = observación de la variable en el instante de tiempo t

$\hat{y}_t$ = predicción realizada en el instante de tiempo t

$e_t = y_t - \hat{y}_t$ (error de predicción en el instante t)

N = número de observaciones del conjunto de prueba

Para el presente proyecto se hará uso de las dos siguientes métricas de evaluación de errores:

- Raiz Cuadrada del Error Cuadrático Medio, o **RMSE** (*Root Mean Squared Error*).

$$RMSE = \sqrt{\frac{\sum_{t=1}^{N}(y_t - \hat{y}_t)^2}{N}} = \sqrt{\frac{\sum_{t=1}^{N} e_t^2}{N}}$$

  El RMSE toma la raíz cuadrada del promedio de los errores de predicción al cuadrado. La doble transformación realizada provoca dos efectos: por un lado se da más peso a los errores grandes, y por otro se evita que los errores positivos y negativos se anulen, ya que se toma el cuadrado de estos valores. Su valor depende de la escala utilizada, por lo que no se recomienda su uso en aquellos casos en los que se comparen predicciones sobre distintas variables. Las unidades coinciden por tanto con las unidades de la variable a estudio.

- Error Absoluto Medio, o **MAE** (*Mean Absolute Error*).

$$MAE = \frac{1}{N}\sum_{t=1}^{N}|e_t| = \frac{1}{N}\sum_{t=1}^{N}|y_t - \hat{y}_t|$$

  También conocido como Desviación Media Absoluta, el MAE toma el promedio de los errores absolutos de las predicciones. Al igual que RMSE, esta métrica depende de la escala utilizada, y de nuevo sus unidades coinciden con las unidades de la variable estudiada.

En la comparación de ambas métricas, el RMSE posee como principal ventaja la penalización de grandes errores producidos, sin embargo en cuanto a su interpretación el MAE es más aconsejable. El RMSE no describe únicamente un promedio de errores, y posee otras implicaciones que son más difíciles de interpretar.

8.2. Análisis de resultados

Para llevar a cabo el análisis de los resultados mediante las métricas previas se han utilizado, a mayores de las predicciones mostradas en el Capítulo 7, predicciones con tamaños de ventana `[1,2,3,4,6,8,10,12,24,48,72,168]` para todos y cada uno de los modelos utilizados (a excepción de la red LSTM *Encoder-Decoder* cuyos resultados serás analizados más adelante, dado que su estructura difiere del resto de los modelos y no permite trabajar con una misma red sobre los diversos tamaños de ventana).

La Figura 8.1 y la Tabla 8.1 muestran los valores de la métrica RMSE para cada uno de los modelos y cada uno de los tamaños de ventana considerados.

Los mejores resultados se obtienen con la red LSTM *Vanilla* con variables exógenas, y de hecho la diferencia entre los resultados obtenidos con esta red distan bastante de los obtenidos con el resto de modelos, principalmente para periodos largos de predicción. De hecho a partir de un tamaño de ventana de 4 unidades la diferencia comienza a hacerse ya notable. Siguiendo esta misma métrica, los dos siguientes modelos que aportan mejores resultados son el modelo ARIMA con términos de Fourier y el modelo TBATS, justo aquellos que modelan tanto la estacionalidad diaria como la semanal. Los tres modelos restantes (ARIMA, SARIMA y LSTM *Vanilla* sin variables exógenas) aportan malas aproximaciones a los valores reales, debido principalmente a que son modelos para los cuales no se han considerado explícitamente todas las variaciones estacionales. En el caso de la red LSTM sin variables exógenas, la red no es capaz de encontrar por sí sola los patrones estacionales a largo plazo sobre la serie de consumos.

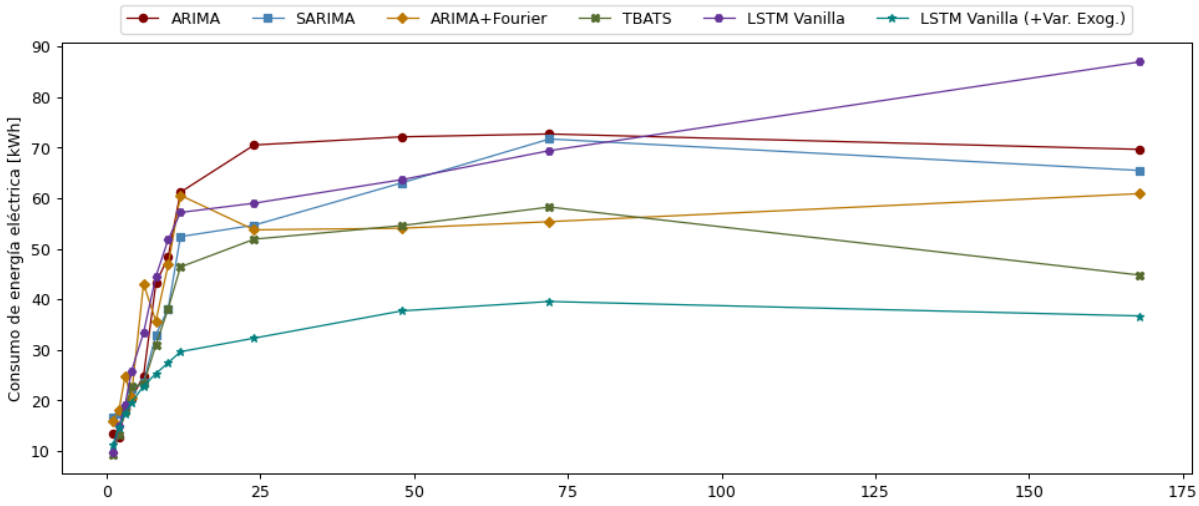


Figura 8.1: Representación gráfico de los errores de predicción con la métrica RMSE

Tamaño de ventana	ARIMA	SARIMA	ARIMA +Fourier	TBATS	LSTM Vanilla	LSTM Vanilla +Var. Exog.
1	13,3718	16,5893	16,0105	9,4014	9,8713	11,1917
2	12,6391	17,2841	18,1010	13,3320	14,9220	14,7784
3	18,4038	18,4812	24,8415	17,9071	19,1150	17,4770
4	20,2723	22,2066	20,7129	22,7349	25,6705	19,5480
6	24,8938	23,6400	43,0642	23,2483	33,4029	22,8644
8	43,1311	32,9274	35,7081	30,9703	44,5353	25,3110
10	48,3982	38,1878	46,8219	38,1464	51,7855	27,4499
12	61,1804	52,4005	60,5598	46,3995	57,1751	29,6511
24	70,5197	54,6709	53,7306	51,8946	59,0104	32,3258
48	72,1328	63,0365	54,0614	54,5790	63,6653	37,7181
72	72,6925	71,7028	55,3554	58,2424	69,3975	39,5704
168	69,6488	65,4753	60,8950	44,7956	86,9499	36,7189

Tabla 8.1: Análisis de los errores de predicción con la métrica RMSE

Por otro lado, para llevar a cabo un análisis complementario del grado de aproximación de las predicciones, la Figura 8.2 y la Tabla 8.2 muestran los valores del MAE para cada uno de los modelos y cada uno de los tamaños de ventana considerados. Con esta nueva métrica es posible hablar ya de promedio de errores, e interpretar los resultados obtenidos a partir de esta

interpretación.

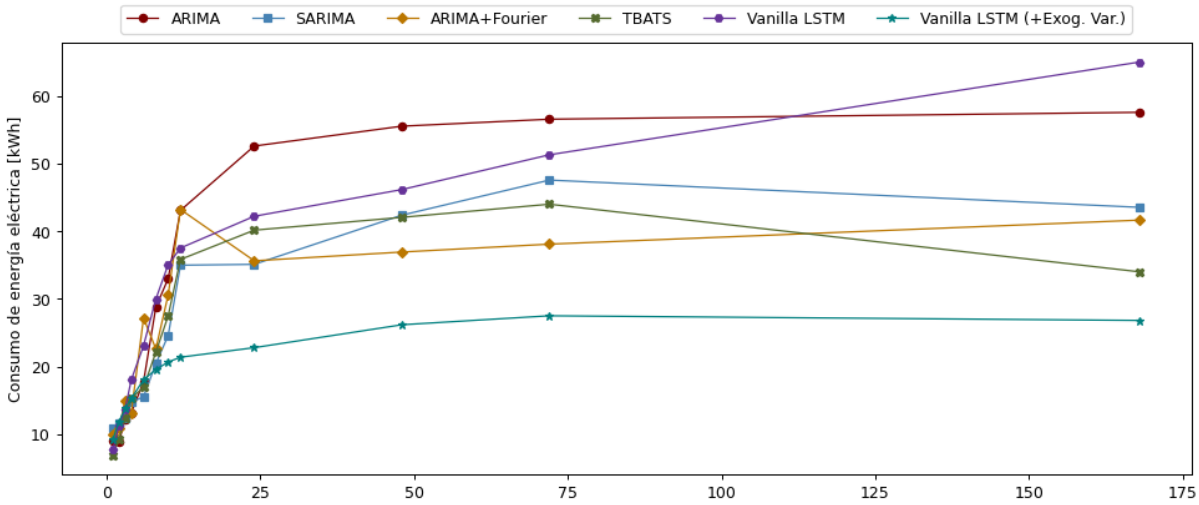


FIGURA 8.2: REPRESENTACIÓN GRÁFICO DE LOS ERRORES DE PREDICCIÓN CON LA MÉTRICA MAE

Tamaño de ventana	ARIMA	SARIMA	ARIMA +Fourier	TBATS	LSTM Vanilla	LSTM Vanilla +Var. Exog.
1	9,0158	10,8289	9,8558	6,8875	7,6339	9,1466
2	8,7827	11,5843	10,8254	9,2125	11,2043	11,8317
3	12,2285	12,3447	14,9849	12,3124	13,6639	13,7633
4	13,1243	14,6847	13,1667	15,3117	18,0156	15,2607
6	17,7437	15,5285	27,1724	16,9026	23,0964	18,0382
8	28,7994	20,4240	22,6017	22,1149	29,9932	19,5497
10	33,0218	24,5781	30,6091	27,5002	35,0852	20,6588
12	43,1305	35,0101	43,2642	35,8612	37,5534	21,3745
24	52,6657	35,1339	35,6591	40,1998	42,2596	22,7891
48	55,6073	42,4220	36,9581	42,0983	46,2162	26,1827
72	56,6417	47,6212	38,1291	44,0483	51,3702	27,5156
168	57,6598	43,5798	41,6898	34,0208	65,1177	26,8209

TABLA 8.2: ANÁLISIS DE LOS ERRORES DE PREDICCIÓN CON LA MÉTRICA MAE

La red LSTM *Vanilla* con variables exógenas es de nuevo con esta otra métrica el modelo de predicción que mejor aproxima los valores reales de consumos. Para predicciones con grandes tamaños de ventana, el error promedio de los errores absolutos cometidos se encuentra aproxi-

madamente entre 20 y 25kWh. Estos valores son bastante esperanzadores si los comparamos por ejemplo con los resultados obtenidos con el modelo de predicciones ARIMA, para el cual estos promedios se encuentran comprendidos entre 50 y 55kWh. En general estos errores son inferiores a los obtenidos con el RMSE, lo que lleva a la conclusión de que la penalización que ejerce el criterio RMSE a errores grandes es notable en la gran mayoría de modelos.

En general, para tamaños de ventana pequeños, o lo que es lo mismo, para predicciones a corto plazo, todos los modelos estudiados presentan comportamientos similares. Las predicciones con un horizonte temporal inferior a 4 lapsos de tiempo presentan errores pequeños.

Para el caso de la red LSTM *Encoder-Decoder* dirigida a problemas de predicción con grandes secuencias, los resultados de sus predicciones son notablemente mejores a los obtenidos con los modelos previos. Para el caso de las red LSTM *Encoder-Decoder* con variables exógenas construido en la sección 7.5.3 para un tamaño de secuencia de salida igual al tamaño del conjunto de prueba (2493), el RMSE es `16.1886` y el MAE `11.6689`. Estos valores son prácticamente iguales a los obtenidos con los modelos previos para tamaños de ventanas pequeños, sin embargo en este caso se trata de un tamaño de ventana de casi 2500 horas, lo que equivale a aproximadamente 15 semanas de horizonte temporal.

Capítulo 9

Conclusiones y trabajo futuro

Tras el análisis de resultados previo y un análisis global del presente documento, es claro que ha sido posible alcanzar los objetivos planteados en el proyecto. Por un lado ha sido posible estudiar y contrastar diversos modelos de predicción, tanto aquellos construidos con un enfoque probabilístico, como los modelo de aprendizaje automático materializados a través de las RNN, y en concreto mediante la construcción de las LSTM (OBJ-1). A la vista de los resultados se ha probado que los modelos clásicos aportan buenos resultados pero únicamente para predicciones a corto plazo. Por el contrario ha quedado demostrada la eficacia de las redes LSTM, a través de sus diversas arquitecturas, para llevar a cabo predicciones a largo plazo (OBJ-4). Si bien las redes LSTM con arquitectura *Vanilla* y variables exógenas han permitido realizar muy buenas predicciones a nivel general, tanto a corto como a largo plazo. Si únicamente se requieren predicciones a corto plazo (de 1 a 3 o 4 periodos), el esfuerzo de construcción de estas redes y de construcción de los conjuntos de datos para el entrenamiento y la realización de pruebas no compensa con respecto al uso de los modelos clásicos. Sin embargo para el caso de predicciones a largo plazo todo cambia, dado que los resultados de las redes neuronales han sido mucho mejores y el esfuerzo de construcción se ve más que recompensado.

De cara al cumplimiento del OBJ-2, el análisis y la investigación dedicados a encontrar un conjunto de datos sobre el que aplicar las predicciones dieron sus frutos, y gracias a los datos que el Ayuntamiento de Dublín en su momento hizo públicos ha sido posible realizar el proyecto. En cuanto a las implementaciones OBJ-3, ha sido posible construir todos los modelos apoyándose para ello en un gran número de librerías de *Python* que han sido previamente seleccionadas por su gran versatilidad y su adaptación al presente proyecto.

De cara a una continuación futura del proyecto, se deberá tratar de buscar nuevas alternativas de modelos predicción basados en modelos de aprendizaje automático, a la vista de haber sido los que mejores resultados han aportado. De cara a seguir profundizando en los modelos actuales, se puede tratar de incorporar nuevas variables exógenas al sistema, o incluso simplemente variables

temporales asociadas al consumo, pasando a trabajar de este modo con series temporales multivariantes. En esta línea pueden incorporarse nuevas variables propias de los edificios inteligentes, obtenidas directamente de sensores o medidores del edificio.

En cuanto a los datos, como trabajo futuro se puede intentar extender estos mismos modelos sobre nuevos *datasets*, con el fin de estudiar el grado de aproximación de los mismos al trabajar sobre patrones de estacionalidad más complejos.

Referencias bibliográficas

Angioi, A. (2020). Time Series Forecasting with an LSTM Encoder/Decoder in TensorFlow 2.0. Recuperado desde https://www.angioi.com/time-series-encoder-decoder-tensorflow/

Bengio, Y., Simard, P. & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, *5*(2), 157-166. doi:10.1109/72.279181

Box, G. E. & Jenkins, G. M. (1970). *Time series analysis: forecasting and control*. Holden-Day series in time series analysis and digital processing. Holden-Day.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H. & Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. doi:10.3115/v1/D14-1179

Choi, E., Cho, S. & Kim, D. K. (2020). Power Demand Forecasting using Long Short-Term Memory (LSTM) Deep-Learning Model for Monitoring Energy Sustainability. *Sustainability*. doi:10.3390/su12031109

De Livera, A. M., Hyndman, R. J. & Snyder, R. D. (2011). Forecasting time series with complex seasonal patterns using exponential smoothing. *Journal of the American statistical association*, *106*(496), 1513-1527. doi:10.1198/jasa.2011.tm09771

Elman, J. L. (1990). Finding structure in time. *Cognitive science*, *14*(2), 179-211. doi:10.1016/0364-0213(90)90002-E

Grenning, J. (2002). Planning poker or how to avoid analysis paralysis while release planning. *Hawthorn Woods: Renaissance Software Consulting*, *3*, 22-23.

Hicks, M. & Foster, J. S. (2010a). Adapting Scrum to Managing a Research Group. Recuperado desde http://hdl.handle.net/1903/10743

Hicks, M. & Foster, J. S. (2010b). Score: Agile research group management. *Communications of the ACM*, *53*(10), 30-31. doi:10.1145/1831407.1831421

Hochreiter, S. & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735-1780. doi:10.1162/neco.1997.9.8.1735

Holt, C. C. (1957). Forecasting seasonals and trends by exponentially weighted moving averages, Office of Naval Research. *Research memorandum*, *52*. doi:10.1016/j.ijforecast.2003.09.015

Hyndman, R. & Athanasopoulos, G. (2018). Forecasting: principles and practice. Recuperado desde OTexts.com/fpp2

Kingma, D. P. & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv: 1412.6980

Kwiatkowski, D., Phillips, P. C., Schmidt, P., Shin, Y. y col. (1992). Testing the null hypothesis of stationarity against the alternative of a unit root. *Journal of econometrics*, *54*(1-3), 159-178. doi:10.1016/0304-4076(92)90104-Y

Lu, Y. (2017). Industry 4.0: A survey on technologies, applications and open research issues. *Journal of industrial information integration*, *6*, 1-10. doi:10.1016/j.jii.2017.04.005

Marino, D. L., Amarasinghe, K. & Manic, M. (2016). Building energy load forecasting using Deep Neural Networks. *IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society*. doi:10.1109/iecon.2016.7793413

Medojevic, M., Villar, P. D., Cosic, I., Rikalovic, A., Sremcev, N. & Lazarevic, M. (2018). Energy management in Industry 4.0 ecosystem: a review on possibilities and concerns. *Annals of DAAAM & Proceedings*, *29*. doi:10.2507/29th.daaam.proceedings.097

Montalvo, D. (2020). Tensores y datos tensoriales.

Morilla, C. R. (2000). *Análisis de series temporales*. La muralla.

Naug, A. [Aviek], Ahmed, I. & Biswas, G. (2019). Online energy management in commercial buildings using deep reinforcement learning. En *2019 IEEE International Conference on Smart Computing (SMARTCOMP)* (pp. 249-257). IEEE. doi:10.1109/SMARTCOMP.2019.00060

Naug, A. [Avisek] & Biswas, G. (2018). Data driven methods for energy reduction in large buildings. En *2018 IEEE International Conference on Smart Computing (SMARTCOMP)* (pp. 131-138). IEEE. doi:10.1109/SMARTCOMP.2018.00083

Olah, C. (2015). Understanding LSTM Networks. Recuperado desde https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Ozturk, S. & Ozturk, F. (2018). Forecasting energy consumption of Turkey by Arima model. *Journal of Asian Scientific Research*, *8*(2), 52. doi:10.18488/journal.2.2018.82.52.60

Palachy, S. (2019). Stationarity in time series analysis. Recuperado desde https://towardsdatascience.com/stationarity-in-time-series-analysis-90c94f27322

Pascanu, R., Mikolov, T. & Bengio, Y. (2013). On the difficulty of training recurrent neural networks. En *International conference on machine learning* (pp. 1310-1318).

Peña, D. (2005). *Análisis de series temporales*. Alianza.

Qian, N. (1999). On the momentum term in gradient descent learning algorithms. *Neural networks*, *12*(1), 145-151. doi:10.1016/S0893-6080(98)00116-6

Reinsel, G. C. (2003). *Elements of multivariate time series analysis*. Springer Science & Business Media.

Roblek, V., Meško, M. & Krapež, A. (2016). A complex view of industry 4.0. *Sage Open*, *6*(2), 2158244016653987. doi:10.1177/2158244016653987

Rumelhart, D. E., Hinton, G. E. & Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, *323*(6088), 533-536. doi:10.1038/323533a0

Schwaber, K. (1997). Scrum development process. En *Business object design and implementation* (pp. 117-134). doi:10.1007/978-1-4471-0947-1_11

Tang, Z., De Almeida, C. & Fishwick, P. A. (1991). Time series forecasting using neural networks vs. Box-Jenkins methodology. *Simulation*, *57*(5), 303-310. doi:10.1177/003754979105700508

Werbos, P. J. (1988). Generalization of backpropagation with application to a recurrent gas market model. *Neural networks*, *1*(4), 339-356. doi:10.1016/0893-6080(88)90007-X

Werbos, P. J. (1990). Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, *78*(10), 1550-1560. doi:10.1109/5.58337

Winters, P. R. (1960). Forecasting sales by exponentially weighted moving averages. *Management science*, *6*(3), 324-342. doi:10.1287/mnsc.6.3.324

Apéndices

Apéndice A

Contenido adjunto

Junto a este documento, en el directorio de *OneDrive* proporcionado por la Escuela de Ingeniería Informática de Segovia para el deposito de los ficheros complementarios del proyecto, se incluye:

- `Dockerfile`: fichero de configuración del contenedor de *Docker* con el entorno de pruebas preparado.
- `dccelectricitycivicsblocks34p20130221-1840.csv`: fichero con los datos de consumo de de los bloques 3 y 4 de oficinas del ayuntamiento de Dublín.
- `/modelos`: directorio con los modelos de predicción ya ajustados.
- `notebook.ipynb`: fichero de *Jupyter Notebook* con todos las pruebas realizadas del proyecto.
- `start.sh`: script de Bash con los comandos para construir la imagen de *Docker* y crear un contenedor con el entorno del proyecto.
- `/variables`: directorio con las variables serializadas más relevantes del proyecto, creadas con el fin de agilizar las ejecuciones los modelos de cara a la realización de predicciones.

A continuación se muestra el árbol de directorios y ficheros adjuntos.

```
.
├── Dockerfile
├── dccelectricitycivicsblocks34p20130221-1840.csv
├── modelos
│   ├── dublin_autoarima-fourier.pkl
│   ...
│   └── dublin_tbats.pkl
├── notebook.ipynb
├── start.sh
└── variables
    ├── arima
    │   ├── arima_12_periods_forecast
    │   ...
    │   └── arima_out_of_sample_forecast
    ├── arima_fourier
    │   ├── fourier_12_periods_forecast
    │   ...
    │   └── fourier_out_of_sample_forecast
    ├── lstm
    │   ├── lstm_12_periods_forecast
    │   ...
    │   └── lstm_vanilla_forecasts
    ├── sarima
    │   ├── sarima_12_periods_forecast
    │   ...
    │   └── sarima_out_of_sample_forecast
    └── tbats
        ├── tbats_12_periods_forecast
        ...
        └── tbats_out_of_sample_forecast
```

Apéndice B

Instalación y versiones

B.1. Versiones

Toda las implementaciones del proyecto se han llevado a cabo utilizando *Python* como lenguaje de programación en su versión 3.8. En cuanto a las librerías requeridas, a continuación se incluyen las versiones que han sido utilizadas durante el desarrollo del proyecto.

- Pandas 1.1.1
- Numpy 1.18.5
- Matplotlib 3.3.1
- Joblib 0.16.0
- Statsmodels 0.12.0
- Pmdarima 1.7.0
- Tbats 1.1.0
- TensorFlow 2.3.0
- Keras 2.4.3

Todas las pruebas relativas a la implementación y construcción de los modelos, realización de predicciones y análisis de resultados se han llevado a cabo en un fichero con extensión *.ipynb* elaborado con *Jupyter Notebook* versión 6.1.3.

B.2. Instalación con Docker

Con el fin de poder cargar el fichero y visualizarlo e interactuar con él, se requiere la instalación de *Python*, *Jupyter Notebook* y el conjunto de librerías previamente listadas. Para facilitar la instalación se ha utilizado una imagen de *Docker* con el entorno preparado para la carga directa del fichero de *Jupyter*. El único requisito previo es disponer de *Docker*. La forma más sencilla de instalarlo es mediante la versión de escritorio, *Docker Desktop*[1].

[1] https://docs.docker.com/get-docker/

Una vez instalado, el siguiente paso consiste en la ejecución del script de Bash `start.sh` que se adjunta con el presente documento. Este archivo contiene los comandos de *Docker* necesarios para construir la imagen y crear un contenedor volátil que será eliminado automáticamente tras finalizar la ejecución.

Una vez se ha instalado la imagen en local y el contenedor se ha creado correctamente, se debe copiar la URL que aparece en la consola de comandos en un navegador web, con la estructura `http://127.0.0.1:8888/?token=my_token`. Esta dirección dará acceso a la interfaz gráfica de *Jupyter Notebook*, a partir de la cual se podrá acceder al fichero `notebook.ipynb` con todo los contenidos del proyecto.