Trabajo Fin de Máster

Técnicas de Machine Learning aplicadas a la Estadística Pública

Alexander Suárez Soto

Máster en Técnicas Estadísticas

Curso 2025-2026

Propuesta de Trabajo Fin de Máster

Título en galego: Técnicas de Machine Learning aplicadas á Estatística Pública
Título en español: Técnicas de Machine Learning aplicadas a la Estadística Pública
English title: Machine Learning Techniques applied to Public Statistics
Modalidad: Modalidad B
Autor/a: Alexander Suárez Soto, Universidade de Santiago de Compostela
Director/a: Alberto Rodríguez Casal, Universidade de Santiago de Compostela
Tutor/a: Marcos Rodríguez Rey, Instituto Galego de Estatística (IGE)
Breve resumen del trabajo:
Recomendaciones:
Otras observaciones:

Don Alberto Rodríguez Casal, catedrático de la Universidade de Santiago de Compostela y don Marcos Rodríguez Rey, subdirector xeral de síntese, análise e difusión de Instituto Galego de Estatística (IGE), informan que el Trabajo Fin de Máster titulado

Técnicas de Machine Learning aplicadas a la Estadística Pública

fue realizado bajo su dirección por don/doña Alexander Suárez Soto para el Máster en Técnicas Estadísticas. Estimando que el trabajo está terminado, dan su conformidad para su presentación y defensa ante un tribunal. Además, Don Alberto Rodríguez Casal, y don Alexander Suárez Soto

√ sí ☐ no

autorizan a la publicación de la memoria en el repositorio de acceso público asociado al Máster en Técnicas Estadísticas.

En Santiago de Compostela, a 31 de mayo de 2026.

El/la director/a:
Don/doña Alberto Rodríguez Casal

RODRIGUEZ CASAL, ALBERTO (AUTENTICACIÓN)
Firmado digitalmente por RODRIGUEZ CASAL, ALBERTO (AUTENTICACIÓN)
Nombre de reconocimiento (DN): c=ES, serialNumber=34896829X, sn=RODRIGUEZ, givenName=ALBERTO, cn=RODRIGUEZ CASAL, ALBERTO (AUTENTICACIÓN)
Fecha: 2026.06.01 14:42:00 +02'00'

El/la autor/a:
Don/doña Alexander Suárez Soto

SUAREZ SOTO ALEXANDER - 73058364Z
Firmado digitalmente por SUAREZ SOTO ALEXANDER - 73058364Z
Fecha: 2026.05.31 19:34:12 +02'00'

El/la tutor/a:
Don/doña Marcos Rodríguez Rey

INSTITUTO GALEGO DE ESTATÍSTICA
Marcos Rodríguez Rey
Santiago de Compostela
2026.06.01 14:53:33+02'00'

Declaración responsable. Para dar cumplimiento a la Ley 3/2022, de 24 de febrero, de convivencia universitaria, referente al plagio en el Trabajo Fin de Máster (Artículo 11, Disposición 2978 del BOE núm. 48 de 2022), **el/la autor/a declara** que el Trabajo Fin de Máster presentado es un documento original en el que se han tenido en cuenta las siguientes consideraciones relativas al uso de material de apoyo desarrollado por otros/as autores/as:

- Todas las fuentes usadas para la elaboración de este trabajo han sido citadas convenientemente (libros, artículos, apuntes de profesorado, páginas web, programas,...)
- Cualquier contenido copiado o traducido textualmente se ha puesto entre comillas, citando su procedencia.
- Se ha hecho constar explícitamente cuando un capítulo, sección, demostración,...sea una adaptación casi literal de alguna fuente existente.

Y, acepta que, si se demostrara lo contrario, se le apliquen las medidas disciplinarias que correspondan.

Agradecimientos

Me gustaría agradecer:

A mi director, Alberto Rodríguez, por su implicación en todos los sentidos, sinceridad en cada corrección y por acompañarme por segunda vez en uno de los proyectos más importantes de mi vida.

A mi tutor, Marcos Rodríguez, así como a toda la plantilla del Instituto Galego de Estatística, tanto por el apoyo brindado con el trabajo, como por hacerme sentir uno más del gran grupo que son, haciendo que el proceso de elaboración de este documento fuese más que ameno, divertido.

A todos mis profesores del máster, por haberme permitido profundizar más en lo que me gusta, facilitando siempre el proceso.

A Lucía Souto, por estar a mi lado cada día.

A Jacinto Suárez, Ana Soto, Iñaki Suárez y toda mi familia, porque sin ellos no podría estar donde estoy hoy.

A Javier Soto, por haber confiado siempre en mí.

Índice general

Resumen

Resumen en español

Este trabajo girará en torno al modelado y predicción de series de tiempo. Se estudiarán una serie de metodologías y algoritmos, partiendo desde las bases estadísticas clásicas, como puede ser la metodología Box-Jenkins y centrándonos principalmente en el machine learning, revisando técnicas de reducción de la varianza como pueden ser el bagging y el boosting. Además, se hará especial hincapié en un modelo asociado a cada una de estas técnicas: random forest por parte del bagging y XGBoost por parte del boosting. Por último, nos centraremos en el estudio de las redes neuronales recurrentes, que dada su idoneidad para la gestión de datos secuenciales nos serán de gran utilidad para el modelado de series de tiempo.

Después se llevará a cabo un análisis sobre conjuntos de datos de interés público proporcionados por el Instituto Galego de Estatística, mediante los cuales se tratarán de visualizar los puntos fuertes y débiles de los modelos expuestos de forma teórica.

English abstract

This project will focus on time series modelling and forecasting. We will examine a range of methodologies and algorithms, starting with classical statistical approaches such as the Box-Jenkins method and focusing primarily on machine learning, whilst reviewing variance-reduction techniques such as bagging and boosting. Furthermore, particular emphasis will be placed on a model associated with each of these techniques: random forest for bagging and XGBoost for boosting. Finally, we will focus on the study of recurrent neural networks, which, given their suitability for handling sequential data, will be of great use to us in time series modelling.

We will then carry out an analysis of datasets of public interest provided by the Galician Institute of Statistics, through which we will attempt to visualise the strengths and weaknesses of the models presented in theory.

Capítulo 1

Introducción

Los seres humanos no podemos ver el futuro. Esto nos impide aprovechar oportunidades, evitar catástrofes y anticiparnos a las situaciones del mañana. Todos hemos deseado en algún momento, en mayor o menor medida, saber qué nos depara, qué sucederá al cabo de un tiempo con el tema que nos preocupa en un determinado momento. Tanto es así que cantidad de cuentos y de historias fantasean con que los protagonistas pueden hacerlo, trasladándonos a un mundo que nos permite estar prevenidos de los eventos venideros. Si bien los seres humanos no podemos ver el futuro, sí podemos predecirlo, y aquí es donde convergen las matemáticas y la fantasía.

Para poder llevar a cabo esta tarea, usamos las series de tiempo, que son conjuntos de datos ordenados, indexados temporalmente, que nos permiten por ejemplo, ver la evolución del precio de un activo, de las temperaturas medias de un país o de otras muchas cosas a lo largo del tiempo. Esta evolución, este comportamiento pasado, nos puede dar más información de la que uno podría pensar a priori. Nos puede dar información sobre la tendencia que lleva el sujeto estudiado, o de si hay un factor estacional, que se repite a lo largo del tiempo. En este punto es donde entran el juego las matemáticas y la inteligencia artificial, que hoy en día está en boca de todos. Aplicando técnicas basadas en la estadística, somos capaces de crear modelos que captan los patrones en la evolución temporal del sujeto, y una vez se tiene esto, podemos aventurarnos a lanzar predicciones, y anticiparnos a si deberíamos comprar o vender una determinada acción, o si la demanda eléctrica en un determinado momento va a ser mayor o menor. Es decir, las series de tiempo nos permiten estar un paso más cerca de poseer ese superpoder imposible de adquirir, pero que ahora podemos rozar. A continuación presentamos un ejemplo de una serie de tiempo que muestra la evolución de la demanda eléctrica en españa, serie con la que posteriormente trabajaremos en el Capítulo 4. Se puede encontrar en la Figura 1.1.

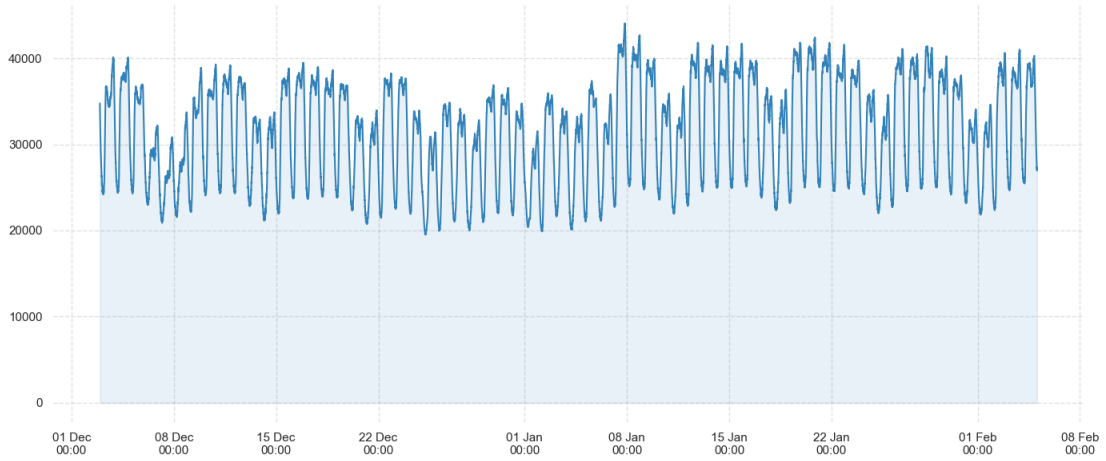


Figura 1.1: Evolución temporal de la demanda eléctrica real en el sistema eléctrico peninsular desde el 1 de noviembre de 2025 hasta el 8 de marzo de 2026. Se presenta un desglose detallado del 50 % central del conjunto de datos para facilitar una mejor apreciación de los detealles. En el eje X tenemos los instantes de tiempo (frecuencia de 5 minutos) y en el eje Y la demanda eléctrica (MW).

Por último, aprovechamos este capítulo introductorio para presentar lo que el lector podrá encontrar a lo largo del documento. En este primer capítulo veremos definiciones de nociones básicas, definiciones de conceptos de relevancia vistos en profundidad y la adecuación de otros muchos cuya utilidad ha de matizarse cuando se ve aplicada al campo de las series de tiempo. En la Sección 1.1 presentaremos las series de tiempo, dando una definición para las mismas y acompañándola de otras asociadas a conceptos muy relacionados. Lamentablemente, hay ciertos conceptos de carácter transversal que no pertenecen estricamente a esta última sección, pero que son necesarios para una correcta comprensión de todo lo que vendrá después. Así nace la Sección 1.2, en la que daremos los matices necesarios para adaptar conceptos de uso cotidiano del ámbito del machine learning a su utilización en el contexto de las series de tiempo. En estas dos primeras secciones se ha buscado sentar las bases relativas a las series de tiempo, que es uno de los temas principales de este documento. La Sección 1.3, por su parte, buscará sentar las bases del otro tema predominante, el machine learning, centrándose en particular en señalar las diferencias de este campo con el deep learning, motivando la necesidad de ambos, cada uno en su debido contexto, ya que precisamente será una combinación de ambos la que constituya el bloque central del estudio.

Cuando se habla de modelos de predicción, un tema que, dada la mención al campo de machine learning se puede deducir que será recurrente a lo largo del documento, se debe hablar también del error que cometen a la hora de predecir. Como sabemos, el error de predicción de un modelo es la diferencia entre el valor predicho por este y la realidad. La cuestión que deseamos abordar es la de cómo medir el error cometido para poder sacar provecho evaluando el rendimiento de los modelos y extrayendo conclusiones interesantes acerca de los datos. Con este objetivo surge la Sección 1.4, en la que revisaremos las métricas de error que emplearemos para la evaluación de las predicciones de nuestros modelos, diferenciando entre aquellas que ofrecen medidas absolutas y aquellas que ofrecen medidas relativas. Llegados a este punto, estaremos familiarizados con las métricas de error, pero en la Sección 1.5 iremos un paso más allá, mostrando un procedimiento que incorpora la medición del error en un proceso iterativo que garantiza la robustez estadística de los resultados. Estamos hablando de una técnica de remuestreo muy conocida, como es la validación cruzada, y de las variantes que esta nos trae consigo.

Una vez adquiridas las nociones globales, estamos en condiciones de pasar a estudiar los modelos de interés. En la Sección 2.1 del Capítulo 2 estudiaremos una metodología clásica de modelado de series de tiempo, la metodología Box-Jenkins, precursora de todo lo que llegó después, y mencionando por qué las metodologías más novedosas no han sido capaces de desplazarla. En las secciones posteriores (Sección 2.2, Sección 2.3 y Sección 2.4) del mismo seguiremos con el modelado y predicción de series de tiempo, pero esta vez desde otro punto de vista, empleando la inteligencia artificial y entrando en el terreno del machine learning, explotando las ventajas que este campo nos ofrece. En el Capítulo 3 nos centraremos en el deep learning, subcampo del machine learning cuya necesidad bajo ciertas condiciones habremos motivado, como decíamos en la Sección 1.3 del presente capítulo. En esencia trabajaremos las redes neuronales, que son la piedra angular del deep learning. Finalmente, la conclusión del estudio llegará en el Capítulo 4, en el que pondremos a prueba los modelos estudiados para ilustrar su comportamiento frente a conjuntos de datos reales, para ver su utilidad práctica, los puntos fuertes y flaquezas de cada uno.

1.1. Series de tiempo

En la introducción del capítulo vimos una primera noción de lo que son las series de tiempo, pero ahora buscamos introducir lo que son de forma precisa. Antes de continuar, cabe destacar que la información principal para la realización de esta sección se extrajo de: Ruiz Abellón (2026), Aneiros Pérez (2024), Rojas-Jimenez (2025) e IBM (2021)

Para poder dar una primera definición de lo que es un serie de tiempo, antes debemos definir el concepto del que se desprende, el *proceso estocástico*. Podemos encontrarla a continuación.

Definición 1. *Un proceso estocástico es una colección o familia de variables aleatorias* $\{X_t : t \in T\}$*, ordenadas según el subíndice* t *que en general se suele identificar con el tiempo.*

En base a la Definición 1, podemos tomar $T = \mathbb{Z}$ llegando de esta forma al proceso estocástico definido como $\{X_t : t \in \mathbb{Z}\}$. Considerando ahora una observación del mismo, que llamamos trayectoria, $\{\dots, x_{-2}, x_{-1}, x_0, x_1, x_2, \dots\}$, podemos definir una serie de tiempo $\{x_1, x_2, \dots\}$ como una realización o trayectoria parcial del proceso estocástico.

Cada serie tiene una esencia que la caracteriza y que nos permite agruparla con otras similares. Esta esencia viene determinada por sus componentes, que serán las que, a la hora de modelar las series, hagan que optemos por un enfoque u otro distinto.

- **Tendencia**. Definimos la tendencia como el patrón subyacente en los datos a lo largo del tiempo. Si hay tendencia, se podrá apreciar que el nivel de la serie, que es la base en torno a la que se mueve la misma sin tener en cuenta el ruido, no es constante.
- **Estacionalidad**. La estacionalidad se define como un comportamiento periódico de la serie. Decimos que está presente cuando la serie está influenciada por factores estacionales de periodo fijo como el día, mes, trimestre.
- **Heterocedasticidad**. Definimos la heterocedasticidad como varianza no constante. Esto es, decimos que la serie de tiempo es heterocedástica cuando la variabilidad de la misma no es constante y cambia, por ejemplo, con el nivel de la serie.

Además de estos, resulta fundamental definir el concepto de estacionariedad, que está íntimamente relacionado con los últimos, y es esencial conocerlo cuando se trabaja con series de tiempo.

Definición 2. *Un proceso estocástico* $\{X_t : t \in T\}$ *es estacionario si se verifican las siguientes condiciones:*

1. $\mu_t = \mu,\ \forall t \in T,$
2. $\gamma(t, t+k) = \gamma_k,\ \forall t, k \in T,$

donde:

- $\mu_t = \mathbb{E}(X_t)$
- $\gamma(s,t) = Cov(X_s, X_t) = \mathbb{E}\left((X_s - \mu_s)(X_t - \mu_t)\right)$

De la Propiedad 2 de la Definición 2 podemos deducir que la varianza de cada variable X_t no depende del instante temporal t. Es decir, tenemos que $\sigma_t^2 = \text{var}(X_t) = \mathbb{E}\left((X_t - \mu_t)^2\right) = \text{Cov}(X_t, X_t) = \gamma(t,t) = \gamma_0$. Adicionalmente, podemos enlazar esta definición con los conceptos que hemos dado previamente. Primero fijémonos en que una serie que posea tendencia o componente estacional, no va a cumplir la Definición 2, ya que su media cambiará con el instante de tiempo. Por otra parte, una serie heterocedástica tampoco la cumplirá, ya que σ_t^2 dependerá a su vez del instante.

Podemos volver momentáneamente sobre la serie de la Figura 1.1. Con un vistazo rápido, ya se pueden apreciar ciertos patrones repetitivos que harían que no fuese estacionaria. Por lo tanto, ilustremos una serie estacionaria, para poder hacernos una idea gráfica de cómo sería una serie que entre dentro de la Definición 2. Para ello, vamos a considerar un ejemplo muy sencillo y muy conocido dentro de las series estacionarias, y es una serie de ruido blanco. El ruido blanco, que denotaremos por $\{a_t\}_t$, es un proceso estacionario en el que se cumplen las siguientes propiedades.

- $\mu = 0.$
- $\gamma_k = \begin{cases} \sigma_a^2, & \text{si } k = 0 \\ 0, & \text{si } k \neq 0 \end{cases}$

Es decir, se puede apreciar que es una serie estacionaria con $\mu_t = \mu = 0$, con varianza constante σ_a^2 con covarianza nula entre variables aleatorias de instantes de tiempo distintos. En la Figura 1.2 podemos encontrar la representación gráfica de una realización que nos ayudará a ilustrar la definicion de ruido blanco y como consecuencia, también de estacionariedad.

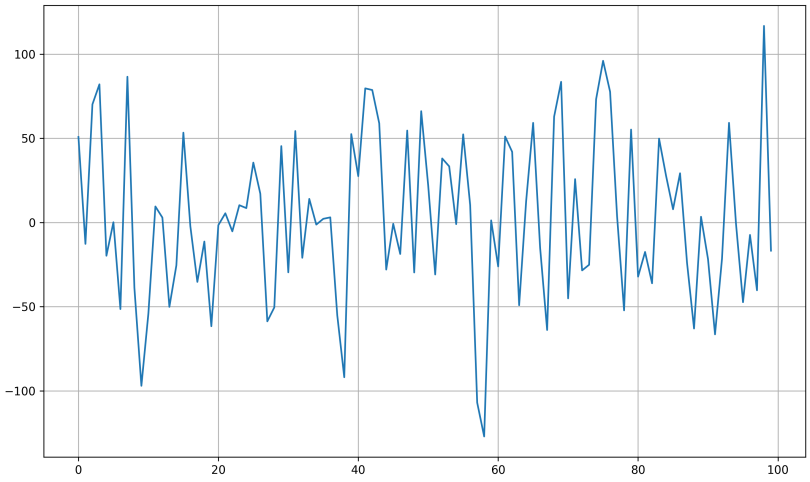


Figura 1.2: Representación gráfica de un proceso de una serie de ruido blanco generado en Python. En el eje X tenemos una rejilla temporal con valores entre 0 y 99. En el eje Y tenemos 100 numeros aleatorios generados mediante una distribución normal de media cero y desviación típica 50.

En la Figura 1.2 se puede ver claramente cómo las oscilaciones poseen una magnitud similar, indicando varianza constante y se producen en torno a su media, que es el cero. Por último, presentamos dos conceptos adicionales que tomarán mucha importancia sobre todo cuando hablemos de la metodología Box-Jenkins en la Sección 2.1 Capítulo 2, la función de autocorrelaciones simples (fas) y la función de autocorrelaciones parciales (fap).

Definición 3. *Definimos la función de autocorrelaciones simples (fas) como*

$$\rho(s,t) = \frac{\gamma(s,t)}{\sigma_s \sigma_t} \in [-1,1].$$

A partir de la Definición 3 vemos que la fas mide el grado de dependencia lineal que existe entre dos variables. Se puede definir como la autocorrelación entre los valores de las series que se encuentran a un determinado número de instantes de distancia. Análogamente, podemos definir la función de autocorrelaciones parciales.

Definición 4. *Definimos la función de autocorrelaciones parciales (fap) como*

$$\alpha(s,t) = \frac{Cov\left(X_s - \hat{X}_s^{(s,t)}, X_t - \hat{X}_t^{(s,t)}\right)}{\sqrt{var\left(X_s - \hat{X}_s^{(s,t)}, X_t - \hat{X}_t^{(s,t)}\right)}},$$

donde $\hat{X}_j^{(s,t)}$ denota el mejor predictor lineal de X_j construido a partir de las variables medidas en los instantes comprendidos entre s y t, sin considerar estos. Dado que estamos en un contexto de series de tiempo, suponemos que se cumple la relación de orden $s < t$, pero matemáticamente esto no tendría mayor relevacia. En la Definición 4 podemos ver esta vez que la fap es la autocorrelación entre los valores de las series que se encuentran a un determinado número de instantes de distancia, teniendo en cuenta los valores de los instantes intermedios. Por lo tanto, esta última mide la dependencia lineal entre dos variables, una vez se ha sustraido el efecto lineal que ejercen sobre ellas las variables medidas en los instantes intermedios.

1.2. Consideraciones generales en el modelado y predicción de series de tiempo

El modelado y predicción de series de tiempo no debe enfocarse como si se tratase de un conjunto de datos al uso. Las series de tiempo poseen una característica muy especial, que ha de tenerse presente a la hora de trabajar con las mismas: un orden. Los datos están indexados en un índice temporal, algo que no puede obviarse, ya que no tiene sentido práctico usar el futuro para predecir el pasado. Es por esto que se ha considerado oportuna la creación de una sección previa en la que se aclaren tanto estas peculiaridades que presentan los datos con los que se estará trabajando, como conceptos generales, presentes implícitamente a lo largo de todo el estudio y con los que, en consecuencia, es importante familiarizarse.

El objetivo de esta sección es definir propiamente y aclarar ciertos conceptos que aparecerán recurrentemente a lo largo del documento, ya sea cuando se presenten las distintas metodologías o cuando se hable de por qué una es preferiblle a otra en determinados contextos. A su vez, hay algunos conceptos básicos del machine learning cuyo funcionamiento se ve ligeramente modificado para adaptarse al contexto de las series de tiempo. Como haremos en cada sección del documento, destacamos que la información para la realización de la misma se extrajo principalmente de: Fernández-Casal et al. (2024), Lee (2026) y Hyndman y Athanasopoulos (2021).

El primer concepto del que hablaremos es el de conjunto de entrenamiento y conjunto de test. El conjunto de entrenamiento es aquel subconjunto de la totalidad de los datos que se emplea para que el modelo aprenda los patrones internos y la estructura de los datos, con el objetivo de que sea capaz de extrapolar las tendencias en base a los mismos y predecir otros nuevos una vez finalizado el entrenamiento. Se debe tener cuidado con el concepto de sobreajuste, que surge cuando el modelo aprende demasiado bien los datos de entrenamiento y pierde la capacidad de generalización, fallando cuando tiene que predecir datos nuevos. Para probar lo bien que predice el modelo en datos nuevos, se emplea el conjunto de test, en base al que se comparan los resultados predichos por el modelo, pudiendo así cuantificar cómo de bueno es este, si ha logrado capturar los patrones de forma satisfactoria o si se está produciendo un sobreajuste.

Si el número de observaciones no es muy grande, se podría entrenar el modelo con la totalidad de los datos, empleando después técnicas de remuestreo para la evaluación de la precisión. Sin embargo, si el conjunto de datos tiene un tamaño considerable, se suele particionar en los dos conjuntos disjuntos que hemos mencionado. Los datos de test deben utilizarse únicamente para la evaluación de los modelos finales, no para el entrenamiento o selección de hiperparámetros óptimos, también llamados parámetros estructurales, que son aquellos parámetros que determinan la estructura y complejidad del modelo. Para este último fin, se puede volver a particionar el conjunto de entrenamiento, obteniendo tres subconjuntos, donde el tercero es el conjunto de validación. Para cada combinación de hiperparámetros, se ajusta el modelo con los datos de entrenamiento, se emplean los de validación para evaluarlos y posteriormente seleccionar los óptimos. Por último, se emplean los datos de test para evaluar el rendimiento del modelo seleccionado. No obstante, lo más habitual para la selección de los hiperparámetros es emplear validación cruzada, sobre la que profundizaremos más adelante. En el contexto de las series de tiempo, se mantienen los significados de los conceptos que acabamos de explicar, pero con ciertas precauciones. Recordemos que las series de tiempo tienen una relación de orden, por lo que los conjuntos de entrenamiento (validación, si lo hay) y test, no se seleccionarán de forma aleatoria, si no siguiendo un orden temporal. No se pueden emplear observaciones del final para el entrenamiento y del inicio para el test.

Hablemos ahora del sesgo, la varianza y el compromiso entre ambos al que todo científico aspira a llegar a la hora de ajustar modelos de predicción. El sesgo y la varianza son dos fuentes de error de predicción. El primero mide cómo de lejos, en promedio, están las predicciones de un modelo con respecto a los valores reales. La varianza, por su parte, mide cuánto cambian las predicciones de un modelo con diferentes conjuntos de datos de entrenamiento. En base a esto, se puede deducir que un modelo demasiado simple, tiende a tener un alto sesgo y baja varianza, mientras que en uno complejo, con muchos parámetros, el sesgo será muy bajo, ya que las predicciones serán más precisas, pero la varianza será más alta. Se debe tener cuidado, ya que estos últimos pueden dar lugar al previamente mencionado sobreajuste. Encontraremos un ejemplo muy ilustrativo de estos conceptos cuando en la Sección 2.2 del Capítulo 2 veamos los árboles de decisión. Tal y como veremos, los árboles profundos son un ejemplo de modelo con bajo sesgo y alta varianza, mientras que los árboles poco profundos son el ejemplo de alto sesgo y baja varianza. Ahora que conocemos ambos conceptos, parece claro que a la hora de ajustar modelos no se busca uno que se posicione cerca de uno de los dos extremos (gran sesgo y poca varianza o gran varianza y poco sesgo), ya que el error disminuirá por un lado pero se verá compensado al aumentar por el otro. Lo que se busca es un equilibrio entre sesgo y varianza, que no es más que el punto en que la combinación de ambos conceptos da lugar al menor error posible.

Otro concepto interesante es el de espacio predictor. Cuando nos refiramos al espacio predictor, hacemos referencia al conjunto de posibles valores que pueden tomar nuestras variables predictoras, es decir, las variables que proporcionamos a nuestro modelo para tratar de predecir la variable de interés. Hablaremos de este espacio cuando hablemos, por ejemplo, de su segmentación en el contexto de los árboles de decisión, en el Capítulo 2, Sección 2.2.

Por otra parte, a lo largo del documento aparecerá en repetidas veces la palabra interpretabilidad, ya que representa una cualidad muy deseable para los modelos ajustados. Esta hace referencia a lo fácil

que resulta la comprensión del camino que ha seguido el modelo para llegar a los resultados a los que ha llegado. Es decir, cómo de comprensible es el proceso interno que le lleva a realizar las predicciones. Normalmente este concepto está íntimamente unido a la complejidad del modelo, ya que a mayor complejidad, menor suele ser la interpretabilidad. Cuando un modelo no es interpretable, decimos que es una caja negra, y suele ser un término recurrente cuando se habla de modelos altamente complejos o hiperparametrizados, como las redes neuronales.

Por último, tenemos el concepto de diferenciación de la serie de tiempo, que ocurre cuando calculamos las diferencias entre observaciones consecutivas de la misma. Esta es una forma habitual de hacer que las series de tiempo no estacionarias, debido a cambios de tendencia, pasen a serlo. La diferenciación puede ayudar a estabilizar la media de una serie de tiempo eliminando los cambios en el nivel de la misma, y reduciendo (o eliminando) como consecuencia la tendencia y estacionalidad. Cuando ajustamos un modelo sobre una serie de tiempo diferenciada, lo que hacemos realmente es que ese modelo aprenda las variaciones de los datos.

1.3. Machine Learning vs Deep Learning

Tal y como se puede leer en el título del trabajo, el tema principal del mismo es el machine learning (aplicado a series de tiempo). El machine learning es un campo muy amplio dentro de la inteligencia artificial, tanto que cuenta a su vez con un subcampo que ha tomado mucha relevacia en los últimos años dadas las características que tienen asociados los problemas que es capaz de abordar. Este subcampo es el deep learning, y el objetivo de esta breve sección es precisamente el de exponer las diferencias justificando el uso simultáneo de ambos. Para ello, tomamos como referencias principales: UDIT (2025) y Jonker y Gomstyn (2026).

El machine learning o aprendizaje automático es una rama de la inteligencia artificial que permite a las máquinas aprender de los datos sin ser programadas explícitamente. Para ello se basa en el uso de algoritmos que identifican patrones en los datos y en base a los mismos llevan a cabo predicciones o tomas de decisiones. Podemos clasificar los algoritmos propios de este campo en dos grandes grupos: aprendizaje supervisado y aprendizaje no supervisado. Los algoritmos pertenecientes al primer grupo se entrenan con datos etiquetados, donde cada entrada se asocia con una salida. Por su parte, en el aprendizaje no supervisado se trabaja con datos sin etiquetar y el objetivo es buscar patrones ocultos o agrupaciones en los datos, sin proporcionar indicaciones de partida.

Por otra parte, el deep learning o aprendizaje profundo es una subcategoría del machine learning que utiliza modelos complejos como redes neuronales artificiales que imitan la estructura del cerebro humano. Una característica que distingue a los algoritmos de deep learning es su capacidad para manejar grandes volúmenes de datos no estructurados, como pueden ser imágenes, audio o texto, y extraer características del proceso, limitando la intervención del investigador, e incluso logrando distinguir patrones que resultarían imperceptibles para el ojo humano.

Antes de continuar, no está de más hacer hincapié en qué son los datos estructurados, los no estructurados y las diferencias entre ambos, ya que como se ha podido ver en el párrafo anterior, son conceptos con los que conviene estar familiarizado cuando se busca diferenciar el machine learning y el deep learning. “Estructurado” y “no estructurado” son términos utilizados para clasificar los datos según su formato y reglas de esquema o falta de ellos. Los datos estructurados se organizan en un formato claro y predefinido, y pueden incluir tanto datos cuantitativos (precios) como cualitativos (fechas, nombres). Por el contrario, los datos no estructurados carecen de este formato predefinido. Suelen ser conjuntos grandes, que pueden representar el 90 % de los datos generados por una empresa. Pueden contener tanto datos textuales (correos electrónicos), como no textuales (archivos de imagen o multimedia), así como datos cualitativos (comentarios en redes) y cuantitativos (cifras incrustadas en texto) también.

Una vez vistas las definiciones de ambos grupos, y lo que son los datos estructurados, podríamos preguntarnos por las diferencias específicas entre ambos, que realmente serán las que harán que optemos por un algoritmo de uno u otro grupo. La primera diferencia llega en términos de estructura y complejidad. El machine learning emplea algoritmos más sencillos, como pueden ser algunos de los que veremos a continuación, a saber: árboles de decisión, bagging, boosting, etc. Por su parte, los algoritmos de deep learning son más complejos y más difíciles de interpretar, pudiendo incluso llegar a convertirse en auténticas cajas negras, como es el caso de las redes neuronales a las que dedicaremos el Capítulo 3.

Otra diferencia llega con los conjuntos de datos que requiere cada grupo. El machine learning puede funcionar con conjuntos de datos estructurados y que no sean de gran tamaño, mientras que el deep learning requiere conjuntos de datos amplios para entrenar sus modelos, que recordemos son realmente complejos, con una gran cantidad de parámetros y pueden llegar a sobreajustar si no cuentan con los datos suficientes para validación.

En cuanto a la capacidad de procesamiento, podemos deducir en base a las características que hemos dado hasta el momento que el machine learning necesita menos y puede llevarse a cabo en hardware convencional, mientras que el deep learning tiene otro tipo de requerimientos, como GPUs (unidades de procesamiento gráfico), dada la elevada demanda computacional y tiempo de entrenamiento.

La última diferencia que expondremos irá de la mano de la precisión y los resultados, y es que ambos grupos brillan en contextos diferentes. Por un lado, los modelos de machine learning proporcionan buenos resultados en tareas de menor complejidad y cantidad de datos, mientras que a los algoritmos de deep learning se les puede confiar tareas más complejas, en las que presenta mayor precisión, a cambio del precio a pagar en términos de coste computacional y tiempo de entrenamiento, como indicábamos anteriormente.

No todo son diferencias entre ambos grupos, y es que como es lógico dado que el deep learning pertenece al machine learning, también comparten ciertas características. La primera similitud y quizá la más evidente es que ambos grupos dependen de los datos, en menor o mayor cantidad, estructurados o desestructurados, para aprender de ellos y potenciar su rendimiento. La segunda similitud se da en el objetivo que ambos grupos comparten. Tanto uno como otro buscan entrenar modelos que sean capaces de generalizar a partir de los datos que han visto en el entrenamiento para predecir o decidir sobre datos nuevos, nunca antes vistos. A continuación, en la Figura 1.3 encontramos un esquema que resume las características que hemos comentado en los párrafos previos.

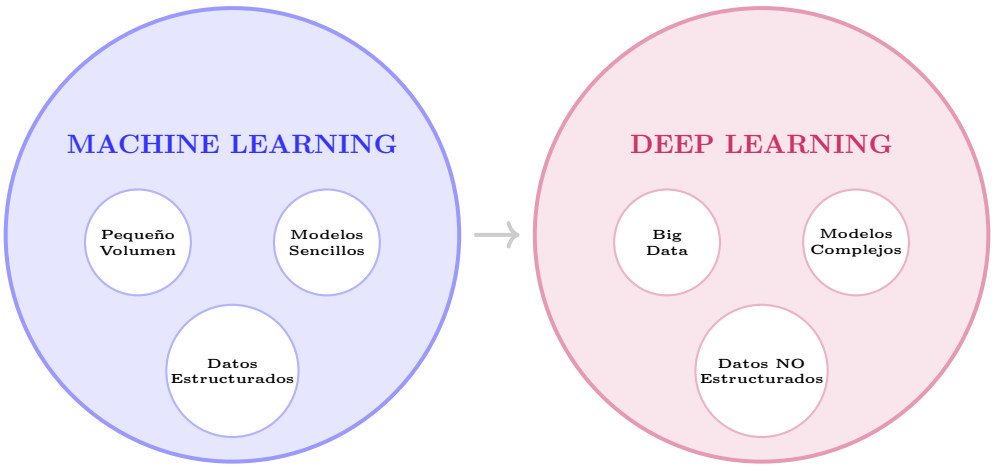


Figura 1.3: Machine Learning vs Deep Learning.

1.4. Métricas de Error

En esta subsección presentamos las métricas de error en base a las cuales se determinará la calidad de los resultados obtenidos con los distintos métodos. Más allá de las más habituales, nos veremos obligados a recurrir a otras menos conocidas, pero que nos ayudarán en caso de que los datos presenten características determinadas que invalidarían la interpretación del resto, como por ejempo la aparición de valores nulos en los datos.

Dividiremos las métricas de error en dos tipos: las absolutas y las relativas. Las primeras son aquellas que dependen de las unidades consideradas y que son más difíciles de interpretar si no se posee un conocimiento profundo de los datos con los que se trabaja. Por ese motivo trabajaremos también con las segundas, con las medidas relativas, que son más fácilmente interpretables ya que muestran un valor de error porcentual o proporcional. Veremos primero las métricas absolutas para después pasar a comentar las relativas, pero antes, cabe destacar que la información para la realización de esta sección se extrajo principalmente de las siguientes fuentes: MSMK (2024), Bonnin, Rodolfo (2017), Oracle (2026), INNOVATIANA (2026), Imperia SCM (2026), DataSphere (2025) y GeeksforGeeks (2025a).

La primera es el Error Absoluto Medio, o MAE por sus siglas en inglés (*Mean Absolute Error*). Se calcula como la media de las diferencias absolutas entre los valores reales y las predicciones del modelo:

$$MAE(y, \hat{y}) = \frac{1}{n}\sum_{i=1}^{n} |y_i - \hat{y}_i|,$$

donde y_i es el valor i-ésimo observado de la serie e $\hat{y}_i$ es el valor de la predicción asociada. Esta medida es crucial en la evaluación de modelos predictivos porque proporciona una medida clara y directa de la precisión de un modelo. Al calcular la media de las diferencias absolutas entre las predicciones y los valores reales, el MAE ayuda a identificar cuán cerca están las predicciones del modelo respecto a los datos reales, sin dar un peso extra a los errores más grandes. Esta medida es especialmente útil en contextos donde los errores grandes y pequeños deben ser considerados igualmente importantes. En cuanto a la interpretación del valor de esta métrica, tenemos que un valor bajo de la misma indica que las predicciones están, en promedio, cerca de los valores reales, sugiriendo un buen rendimiento. Esto, como comentábamos antes, está muy influenciado por la escala de los datos. No es lo mismo trabajar en euros y obtener un MAE de mil de euros, que trabajar en miles de euros y obtener un MAE de mil euros.

De la misma forma que hemos hablado del error absoluto medio, consideraremos el Error Absoluto Mediano o MedAE, por su nombre en inglés (*Median Absolute Error*). Esta medida es análoga a la anterior, pero se calcula tomando la mediana de las diferencias en valor absoluto del valor real y predicho:

$$MedAE(y, \hat{y}) = median(|y_1 - \hat{y}_1|, \ldots, |y_n - \hat{y}_n|).$$

Una ventaja de esta medida, con respecto a la anterior, es que es robusta frente a valores atípicos por el hecho de emplear la mediana. Nos basaremos también en una medida clásica como es el Error Cuadrático Medio o RMSE, nuevamente por su nombre en inglés (*Root Mean Square Error*). Esta medida representa la desviación estándar de los valores residuales o errores de predicción. Consiste en calcular la raíz cuadrada del promedio de los errores al cuadrado entre los valores predichos y los reales.

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

Cuenta con las ventajas de que es sencillo de interpretar por estar en las mismas unidades que la variable objetivo y que penaliza con fuerza los errores grandes, en contraste con lo que hacía el MAE. Así como esta última era más útil cuando queríamos penalizar todos los errores de forma uniforme, el RMSE brillará en contextos donde interese que los errores grandes tengan un gran peso. Como aspectos negativos podríamos destacar que es una medida muy sensible a valores atípicos y que no refleja realmente un error relativo en relación con los datos. Vemos que esto último puede interpretarse tanto como una ventaja como una desventaja. Será de interés ilustrar el error máximo de predicción. No se puede considerar una métrica de error al uso, pero es ilustrativo ver cuánto falló el modelo en su peor predicción.

Una vez presentadas las métricas absolutas, centrémonos en las relativas. Estas tienen la ventaja de que se presentan en términos porcentuales, por lo que tienen una parte intuitiva muy importante. La primera que comentamos es el Error Porcentual Absoluto Medio o MAPE (*Mean Absolute Percentage Error*), que es una métrica que mide el error porcentual medio entre el valor real y la predicción. En otras palabras, indica cuánto se desvía, en promedio, la predicción respecto al valor real:

$$MAPE(y, \hat{y}) = \frac{100}{n} \sum_{i=1}^{n} \left| \frac{y_i - \hat{y}_i}{y_i} \right| \%.$$

Su principal ventaja es que ofrece una interpretación sencilla. Un MAPE del $X\,\%$ significa que, de media, la predicción se desvía un $X\,\%$ del valor observado. La siguiente medida también nos proporciona una idea muy intuitiva del error cometido. Se trata del Error Relativo, y no es más que un reescalado del MAE por la media de la variable respuesta, multiplicado por cien. Es decir:

$$E.R.(y, \hat{y}) = 100 \frac{MAE}{\bar{y}} \%,$$

suponiendo $y \geq 0$. Esta suposición no es descabellada, ya que las series con las que se trabajará más adelante son de exportaciones y demanda eléctrica, cuyos valores serán no negativos. Esta medida es una propuesta muy interesante para poner en contexto el resultado que se obtiene del MAE. No es una medida quizá tan relevante como muchas otras de las expuestas en esta sección, pero para la implementación de la misma no se recurrió a ninguna bibliografía adicional, fue una propuesta del autor dada la necesidad, como decíamos, de poner en contexto el MAE. Continuamos revisando el Error Porcentual Absoluto Ponderado o WAPE (*Weighted Absolute Percentage Error*). Definimos esta métrica como la suma de los errores absolutos dividida por la suma de los valores reales:

$$WAPE(y, \hat{y}) = 100 \frac{\sum_{i=1}^{n} |y_i - \hat{y}_i|}{\sum_{i=1}^{n} |y_i|} \%.$$

A diferencia del MAPE, que promedia errores porcentuales por observación, WAPE usa la suma de la totalidad de errores dividida entre la suma de la totalidad de los datos reales. Esto hace que esta métrica evite el problema de los denominadores pequeños que podían surgir con el MAPE, convirtiéndola en una métrica más robusta. Por último, en cuanto a la interpretación, sería algo del estilo: En promedio, en todo el conjunto de datos, las predicciones del modelo se desvían en un $X\,\%$ de la realidad. Para concluir con las métricas relativas, añadimos el MedAE relativo, que resulta ser el MedAE relativizado por su mediana:

$$R.MedAE(y, \hat{y}) = \frac{MedAE}{median(y_i)} \%.$$

El motivo de considerarla es que nos interesa llevar una métrica tan interesante como es el MedAE a un terreno más visual, como el que nos proporciona un error porcentual. Además, este conservará

la robustez propia del error en mediana, haciéndola muy atractiva para sacar conclusiones analíticas. Finalmente, consideramos una medida que no se podría catalogar a priori de absoluta o relativa, ya que se trata de un coeficiente que tiene un significado particular. Estamos hablando del coeficiente de determinación, denotado por R^2. Esta métrica/coeficiente representa la proporción de la variabilidad de la variable respuesta explicada por un modelo de regresión, mediante el cual se pretende analizar la relación entre una variable respuesta, o dependiente, y una o más variables explicativas o independientes. Se calcula como la diferencia entre la unidad y el cociente entre la suma de los cuadrados de los errores residuales y la suma total de los errores. Estos dos últimos conceptos se definen formalmente como sigue:

$$SS_{res} = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2,$$
$$SS_{tot} = \sum_{i=1}^{n} (y_i - \bar{y})^2.$$

Así, podemos definir nuestra métrica de la forma siguiente.

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}},$$

donde, como decíamos SS_{res} es la suma de los cuadrados de los errores residuales y SS_{tot} la suma total de los errores. A pesar de que en estadística teórica a priori es un valor acotado entre cero y uno, siendo los más próximos a uno los más deseables, la realidad es que puede darse el caso de obtener un valor negativo para el mismo cuando se aplica a modelos de machine learning, lo que estaría indicando que el modelo es peor que predecir el promedio. Matemáticamente, esto ocurre cuando el modelo no permite predecir funciones constantes y es evaluado sobre datos que introducen un error residual mayor que la propia variabilidad total de la muestra.

Por último, cabe destacar que cuando implementemos estas métricas relativas en Python se fijará un ε muy pequeño ($\varepsilon = 1e-10$) para que en caso de toparse con valores nulos, que vemos que se situarán en el denominador, no se divida entre cero directamente, a pesar de que el error aumentará notablemente. Además, es interesante adelantar que si bien la comparación entre los resultados de las distintas métricas será de gran interés, lo primero en lo que nos fijaremos para medir el rendimiento de un modelo será en las métricas relativas, que nos dan una idea quizá más intuitiva, y dentro de las mismas, las más robustas (WAPE, MedAE relativo, ...) serán las primeras.

1.5. Validación Cruzada

A lo largo del documento se mencionará de forma recurrente el concepto de validación cruzada, por lo que le dedicamos una sección de este capítulo introductorio para familiarizarnos con el mismo y que la lectura más adelante sea más fluida evitando repeticiones. Antes de continuar, solo queda destacar que la información que se puede encontrar en esta sección se extrajo principalmente de: Fernández-Casal et al. (2024) y Hyndman y Athanasopoulos (2021).

Definimos la validación cruzada como una herramienta para medir la calidad predictiva de un modelo, ya que permite cuantificar el error de predicción utilizando una única muestra de datos. Existen diversas versiones de esta herramienta, siendo la más simple la validación cruzada dejando uno fuera o LOOCV (*leave-one-out cross-validation*). En esta variante, se realiza un ajuste para cada observación de la muestra empleando el resto de las observaciones (es decir, todas menos la observación en cuestión) y se mide el error de predicción en esa observación, que es el único dato que el modelo no ha visto para

realizar el ajuste. Cuando este proceso se ha repetido para cada una de las observaciones, se pueden combinar todos los errores individuales y obtener de esta forma medidas globales del error de predicción.

La validación cruzada dejando uno fuera requiere, como decíamos antes, un ajuste para cada observación. Se puede deducir que si se cuenta con un conjunto de datos grande, el coste computacional puede ser un factor limitante, lo que nos lleva a otra de las variantes de la validación cruzada, llamada *k-fold cross-validation*. La idea de este método es emplear grupos de observacioes en lugar de observaciones individuales. Se particiona el conjunto en k grupos, que típicamente suelen ser 5 o 10, y a continuación la idea es la misma que en el LOOCV, ajustar el modelo para cada grupo sin tomar únicamente los datos pertenecientes al grupo en cuestión. Notemos que la LOOCV es un ejemplo particular de *k-fold cross-validation* incluyendo un único dato en cada grupo. Pensemos en las implicaciones que puede tener la elección de k en el sesgo y la varianza. Si k es pequeño, significa que hay pocos grupos, y por lo tanto cada uno de ellos contendrá un mayor número de observaciones. Como consecuencia de esto, entrenaremos el modelo con un número menor de observaciones, el sesgo será más alto y la varianza más pequeña. Análogamente, cuando k es más grande hay menos observaciones por grupo y por lo tanto el modelo descarta menos para su ajuste en cada etapa, derivando en un menor sesgo y mayor varianza. Pensemos en un ejemplo práctico en el que tenemos 100 datos. Tomando $k = 2$ y asignando 50 datos a cada grupo, cuando descartemos cada uno de los folds, el modelo se perdera la mitad de los datos del conjunto en cada ajuste, mientras que si tomamos $k = 5$, con 20 datos en cada grupo, el modelo estaría entrenando con el 80 % de los datos en cada etapa.

Además de las mencionadas, existen otras muchas variantes de este método, como la de particionar repetidamente de forma aleatoria los datos en un conjunto de entrenamiento y otro de validación. Así, podría ser que algunas observaciones apareciesen repetidas veces en el conjunto de validación y otras ninguna. Notemos que todas las variantes que hemos mencionado se utilizan en un contexto en el que se busca la minimización del error, pero hay veces que no es tan importante esto, como buscar un equilibrio entre un error bajo y un modelo no muy complejo. Es decir, no siempre compensa reducir al máximo el error, a cambio de no acotar la complejidad del modelo, y más cuando muchas veces una diferencia que se considera insignificante en términos de error, puede reducir significativamente la complejidad del modelo, haciéndolo más interpretable y reduciendo el coste de computación. En este contexto surge la regla de un error estándar, propuesta en Breiman et al. (1984). La idea es que estamos trabajando con estimaciones de la precisión que pueden presentar variabilidad, por lo que la idea radica en seleccionar el modelo más simple dentro de un error estándar de la precisión correspondiente al valor óptimo. Como decíamos antes, se considera que no hay diferencias significativas en términos de precisión, pero además se mitigaría el efecto de la variabilidad debido a la aleatoriedad.

Por último, dado que el contexto en el que estamos es el de las series de tiempo, uno puede pensar que la selección aleatoria quizá no es la más adecuada, ya que no intentaremos predecir el pasado con datos futuros. En este caso, se suele emplear el *rolling forecasting*. En este procedimiento tenemos una serie de conjuntos de test, cada uno compuesto por una única observación. El conjunto de entrenamiento correspondiente está compuesto por todas las observaciones que ocurrieron antes de la observación que compone el conjunto de test, de forma que las observaciones futuras no pueden influir en la predicción. Por otra parte, para garantizar un tamaño considerable del conjunto de entrenamiento (ya que de otro modo las predicciones podrían no ser fiables), las primeras observaciones de la serie se descartan como conjunto de test. Como ocurría con otras variantes de la validación cruzada, la precisión de las predicciones se obtiene promediando sobre los conjuntos de test. Por otra parte, cuando predecimos en base a series de tiempo, las predicciones a un paso pueden no ser tan relevantes o interesantes cuando predecimos un horizonte de tiempo mayor, en cuyo caso es posible adaptar el rolling forecasting para que siga cumpliendo su función. Supongamos que queremos una predicción a horizonte 4. La idea es usar los datos hasta tiempo t para el entrenamiento para predecir lo que ocurrirá en tiempo $t + 4$ (ya sea prediciendo entre medias $t + 1$, $t + 2$ y $t + 3$, o yendo directamente a $t + 4$) y medir el error en dicho tiempo. Después se da un paso adelante, se toma el dato que ahora estaba en el tiempo no visto $t + 1$

dentro del conjunto de entrenamiento y se repite el proceso. En la Figura 1.4 podemos encontrar una representación gráfica de algunas de las variantes mencionadas a lo largo de la sección.

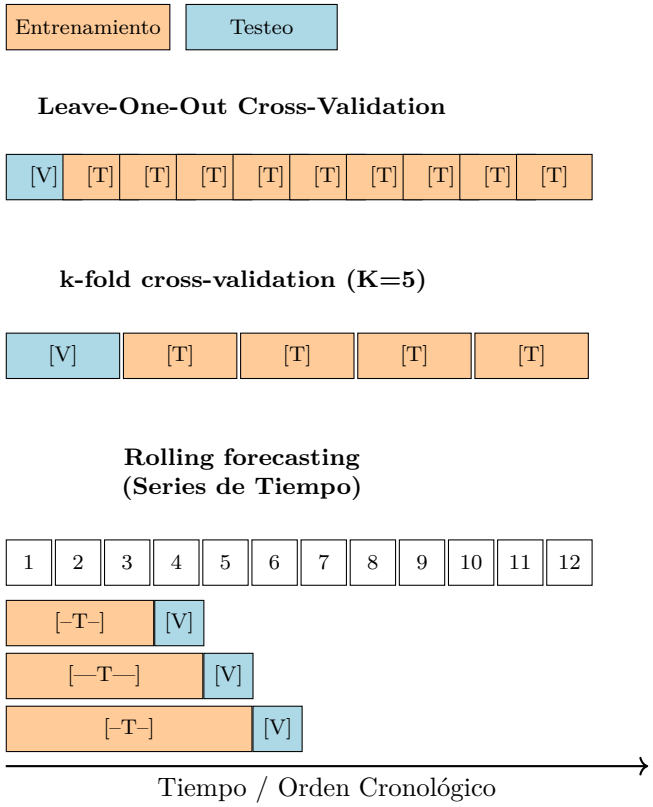


Figura 1.4: Representación gráfica de distintas variantes de validación cruzada. De arriba a abajo: LOOCV, *k-fold* y *rolling forecasting*.

Una vez visto esto, podemos continuar con el Capítulo 2, en el que principalmente estudiaremos diversas metodologías propias del machine learning al uso, dedicando también una sección al modelado estadístico clásico.

Capítulo 2

Modelado con Machine Learning

Las técnicas modernas nos han permitido realizar grandes avances en el modelado y predicción de series de tiempo, pero a diferencia de lo que ocurre en muchos ámbitos de la evolución, estas no han sido capaces de desplazar completamente a las metodologías clásicas, dejándolas obsoletas. De hecho, estos modelos paramétricos clásicos hacen que el investigador se beneficie de un nivel de interpretabilidad muy alto, inaccesible para modelos más novedosos y vanguardistas, como pueden ser XGBoost o las redes neuronales, dado que se basan en conceptos estadísticos claros. En esencia, como veremos más adelante, es aritmética lineal aplicada al contexto de evolución temporal. Por este motivo, dedicaremos la Sección 2.1 a hacer una breve revisión de una metodología clásica de gran relevancia, precursora de los avances que han ido llegando posteriormente, pero que sigue siendo una opción muy válida para la investigación de series de tiempo. Además de seguir siendo una herramienta competitiva, se puede usar también como sparring o modelo de comparación, para evaluar si el rendimiento de los modelos de machine y deep learning está siendo positivo o inferior de lo esperado.

Por otra parte tenemos el término machine learning o aprendizaje automático, en castellano, que se utiliza en el campo de la inteligencia artificial desde 1959 para hacer referencia, principalmente a algoritmos de predicción. En este Capítulo 2 nuestro objetivo principal será la predicción de series de tiempo desde el enfoque propio del aprendizaje automático. Un enfoque que hace un uso más que notable de la programación informática, y que por lo tanto lleva la predicción que podremos ver en la Sección 2.1 a un nuevo nivel en diversos aspectos. La primera diferencia y más notable ocurrirá en términos de escalabilidad, permitiendo que la generación de las predicciones sea más automática, y en consecuencia más abundante. Por otra parte se podrá apreciar un cambio evidente en el papel del investigador. Así como en los métodos estadísticos clásicos debía predominar el conocimiento teórico, estos nuevos métodos requerirán de un mayor esfuerzo a la hora de programar y preprocesar los datos, sin por supuesto dejar completamente de lado los conocimientos estadísticos. En definitiva, este capítulo nos permitirá explorar a fondo la aplicación del machine learning a la predicción de series de tiempo, así como las ventajas en términos de flexibilidad y automatización que este campo nos ofrece con respecto a metodologías clásicas, a pesar de sacrificar la gran interpretabilidad de estas a cambio.

Dado que estamos hablando de un concepto de una extensión considerable que abarca múltiples metodologías, nos vemos obligados a limitar el estudio seleccionando unas opciones muy interesantes y útiles en la práctica. En la Sección 2.2 revisaremos una de las metodologías más conocidas, base de muchas de las que veremos a continuación, los árboles de decisión. Después de esto, continuaremos con métodos más complejos, basados en la reducción de la varianza para reducir la variabilidad de las predicciones. En la Sección 2.3 veremos el método de bagging y en la Sección 2.4 veremos el boosting. A pesar de que se pueden emplear diferentes predictores para ilustrar estas técnicas de reducción de la varianza, aprovecharemos los conocimientos adquiridos en la Sección 2.2 y utilizaremos árboles de decisión.

2.1. Modelado ARIMA

En esta sección estudiaremos la metodología Box-Jenkins, que propone una alternativa muy interesante dentro de los métodos clásicos. Esta metodología está basada en el desarrollo de modelos estadísticos que tienen en cuenta la dependencia existente entre los datos. Estos modelos se conocen con el nombre de ARIMA por sus siglas en inglés, *AutoRegresive Integrated Moving Average*, o Autorregresivos Integrados de Media Móvil en castellano, y permiten expresar un cierto valor como una función lineal de datos anteriores y errores debidos al azar. Dependiendo de las características de la serie temporal con la que se esté trabajando, la mencionada combinación de datos previos y errores debidos al azar puede presentarse de diversas formas, que explicitaremos a continuación. Otra característica interesante de esta metodología es el carácter cíclico de la misma, mediante el cual reconoce patrones en la serie, propone un modelo ARIMA en base a los mismos, estima los parámetros, lo valida y después predice. Si el modelo seleccionado no pasa la validación, se ha de repetir el proceso. Este se puede encontrar ilustrado en la Figura 2.1.

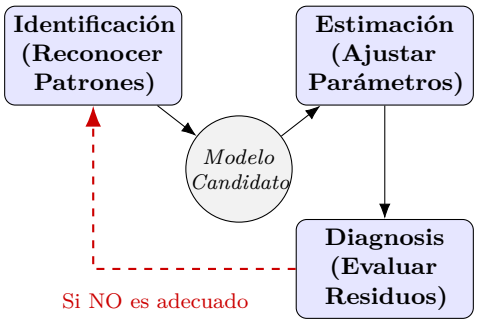


Figura 2.1: Diagrama de bucle iterativo de la metodología Box-Jenkins.

Como se puede intuir en base a la Figura 2.1, esta metodología constará de unas fases muy marcadas, que desglosaremos a lo largo de la sección. Lo primero que haremos será formalizar los modelos determinar las situaciones en las que uno resultaría preferible a otro, pero antes de continuar, cabe destacar que la información para la realización de esta sección se tomaron como referencias principales: de la Fuente Fernández (2025), Aneiros Pérez (2024), Shumway y Stoffer (2017), Rojas-Jimenez (2025).

2.1.1. Formulación de los modelos ARIMA

En esta primera subsección mostraremos la formulación matemática de los distintos modelos en los que se basa la metodología estudiada para realizar sus predicciones. La primera que veremos, que podremos encontrar definida a continuación, es la más simple, y se emplea cuando la serie proporcionada es estacionaria.

Definición 5. *Un proceso estacionario* $\{X_t\}_t$ *que admite la representación*

$$X_t = c + \phi_1 X_{t-1} + \cdots + \phi_p X_{t-p} + a_t + \theta_1 a_{t-1} + \cdots + \theta_q a_{t-q},$$

donde $c, \phi_1, \ldots, \phi_p, \theta_1, \ldots, \theta_q$ $(\phi_p \neq 0 \neq \theta_q)$ *son constantes, se conoce como un proceso ARMA(p,q). Además,* a_t *es el elemento correspondiente al instante de tiempo* t *de una serie de ruido blanco, representado en la Figura 1.2. Equivalentemente, tomando:*

$$\phi(B) = (1 - \phi_1 B - \phi_2 B^2 - \cdots - \phi_p B^p),$$
$$\theta(B) = (1 + \theta_1 B + \theta_2 B^2 + \cdots + \theta_q B^q),$$

donde B *es el operador retardo, definido por* $BX_t = X_{t-1}$*, podemos reescribir la ecuación previa como*

$$\phi(B)X_t = c + \theta(B)a_t.$$

Cabe destacar que, como podemos ver en Aneiros Pérez (2024), $\phi(z)$ no debe tener raíces de módulo 1, ya que esto es condición necesaria y suficiente para que el proceso no sea estacionario, incumpliendo así una hipótesis inicial. Adicionalmente cabe destacar que cuando $q = 0$, es decir, se tiene un ARMA(p,0), se dice que se tiene un proceso autorregresivo de orden p, denotado por AR(p), en el que solamente se tienen en cuenta los valores de la serie en los p instantes anteriores, ponderados por unos pesos que nos ayudan a regular la influencia del pasado en el presente, más un error de predicción en el instante de tiempo actual. Análogamente, cuando $p = 0$, lo que se tiene es un proceso de medias móviles de orden q, es decir, ARMA(0,q) $\Leftrightarrow$ MA(q), en el que solamente se tienen en cuenta los errores de predicción en los q instantes de tiempo anteriores, ponderados nuevamente por unos pesos que ayudan a regular la influencia de los mismos. Es decir, la ecuación vista en la Definición 5 es la unión de dos partes, una autorregresiva y otra de medias móviles.

Tal y como hemos hecho cuando hablamos del Suavizado Exponencial, supongamos ahora que la serie que estamos estudiando no es estacionaria y posee una tendencia. El enfoque que se le da en esta metodología a esta característica es el de aplicar sucesivamente d diferencias (que llamaremos diferencias regulares) a la serie para eliminar la tendencia, hasta obtener una serie que ya no la posea. Si la serie resultante es estacionaria, la modelaríamos a través de un proceso ARMA, como los vistos en la Definición 5. De esta forma surgen los modelos ARIMA, que encontramos definidos en la Definición 6.

Definición 6. *Un proceso ARIMA(p,d,q) es aquel que, después de aplicarle d diferencias regulares, se convierte en un proceso ARMA(p,q). Es decir:*

$$\{X_t\}_t \text{ es ARIMA(p,d,q)} \Leftrightarrow (1-B)^d X_t \text{ es ARMA(p,q).}$$

Equivalentemente, $\{X_t\}_t$ es un proceso ARIMA(p,d,q) si admite una representación del tipo:

$$\phi(B)(1-B)^d X_t = c + \theta(B)a_t,$$

donde el polinomio $\phi(z)$ no tiene raíces de módulo 1.

Por otra parte, un claro aspecto positivo de estos procesos que estamos revisando es que nos ofrecen la posibilidad de modelar, no solo la dependencia regular, sino también la dependencia estacional de la serie de tiempo. No se debe confundir este concepto con el de componente estacional, ya que la primera simplemente estaría asumiendo que hay una inercia estadística, pero la componente estacional asume un patrón rígido, definido por el calendario, que pone en riesgo la estacionariedad de la serie. Para modelar esta dependencia estacional, damos un paso más alla hacia los procesos ARMA estacionales, que podemos definir como sigue.

Definición 7. *Un proceso estacionario $\{X_t\}_t$ que admite la representación*

$$X_t = c + \Phi_1 X_{t-s} + \Phi_2 X_{t-2s} + \cdots + \Phi_P X_{t-Ps} + a_t + \Theta_1 a_{t-s} + \Theta_2 a_{t-2s} + \cdots + \Theta_Q a_{t-Qs},$$

con $c, \Phi_1, \ldots, \Phi_P, \Theta_1, \ldots, \Theta_Q$ ($\Phi_P \neq 0 \neq \Theta_Q$) constantes, se conoce como un proceso ARMA(P,Q)$_s$. Equivalentemente, tomando

$$\Phi(B^s) = (1 - \Phi_1 B^s - \Phi_2 B^{2s} - \cdots - \Phi_P B^{Ps}),$$
$$\Theta(B^s) = (1 + \Theta_1 B^s + \Theta_2 B^{2s} + \cdots + \Theta_Q B^{Qs}),$$

donde B^s denota al operador retardo estacional, definido por $B^s X_t = X_{t-s}$, podemos reescribir la ecuación previa como

$$\phi(B)X_t = c + \theta(B)a_t.$$

En la Definición 7 encontramos el caso análogo de un proceso ARMA, pero cuando la serie muestra una dependencia estacional. Como diferencia con el ARMA que modela la dependencia regular, podemos decir que ahora los coeficientes no nulos se encuentran en los múltiplos del periodo estacional, que ahora por comodidad denotamos por s. Esto a su vez influye en el proceso autorregresivo y el proceso de medias móviles de los que se compone. Notemos que, nuevamente, cuando fijamos $Q = 0$ se tiene un proceso autorregresivo estacional de orden P, ARMA(P,0) $\Leftrightarrow$ AR(P), en el que esta vez solamente influyen los P valores de la serie en los instantes de tiempo que son múltiplos del periodo estacional. A su vez, cuando fijamos $P = 0$, se tiene un proceso de medias móviles estacional de orden Q, ARMA(0,Q) $\Leftrightarrow$ MA(Q) en el que, al igual que nos ocurría con la parte autorregresiva, solamente se tienen en cuenta los Q errores de predicción que se encuentran en los instantes de tiempo que son múltiplos del periodo estacional.

El siguiente paso lógico se presenta cuando deseamos modelar ambos tipos de dependencia de forma simultánea. Para ello, basta con considerar el modelo siguiente.

$$\phi(B)\Phi(B^s)X_t = c + \theta(B)\Theta(B^s)a_t,$$

que denotamos por ARMA(p,q)$\times$(P,Q)$_s$. Hasta ahora, hemos visto que la metodología Box-Jenkins es capaz de lidiar con la dependencia regular, la dependencia estacional y la tendencia de la serie de tiempo que le proporcionemos. Podemos resumirlo de la siguiente manera:

- Un ARMA(p,q) si solo tenemos dependencia regular.
- Un ARMA(P,Q)$_s$ si solo tenemos dependencia estacional.
- ARMA(p,q)$\times$(P,Q)$_s$ si contamos con ambos tipos de dependencia.

Nos falta ver la forma en la que mitiga el efecto de la componente estacional de una serie. Con esta motivación, surgen los procesos ARIMA estacionales, que son aquellos capaces de incorporar la componente estacional de la serie de tiempo. Para ello, aplican a la serie una diferenciación estacional reiterada. Este es el motivo por el que se especificó previamente que la diferenciación para eliminar la tendencia era regular, es decir, la habitual, en la que se calcula la diferencia entre el valor de la serie en tiempo t y su inmediatamente anterior en tiempo $t-1$: $X_t - X_{t-1}$. La diferenciación estacional, por el contrario, es la diferencia entre el valor de la serie en tiempo t y el s instantes de tiempo anterior: $X_t - X_{t-s}$. Gracias a estos procesos, podemos definir uno que realmente serviría para englobar a todos los vistos hasta ahora. Si la serie considerada tiene tendencia y/o componente estacional, el proceso sugerido sería, primero eliminar la tendencia, aplicando d diferencias regulares $\left((1-B)^d\right)$, y posteriormente eliminar la componente estacional aplicando D diferencias estacionales $\left((1-B^s)^D\right)$. Una vez nuestra serie es estacionaria, podrá ser modelada a través de un proceso ARMA.

Definición 8. *Un proceso ARIMA(p,d,q)$\times$(P,D,Q)$_s$, o ARIMA estacional multiplicativo, es aquel que, después de aplicarle d diferencias regulares y D diferencias estacionales de periodo s, se convierte en un modelo ARMA(p,q)$\times$(P,Q)$_s$. Equivalentemente $\{X_t\}_t$ es un proceso ARIMA estacional multiplicativo si admite una representación del tipo:*

$$\phi(B)\Phi(B^s)(1-B)^d(1-B^s)^D X_t = c + \theta(B)\Theta(B^s)a_t,$$

donde el polinomio $\phi(z)\Phi(z^s)$ no tiene raíces de módulo 1.

Como decíamos antes, se puede ver que el proceso presentado en la Definición 8 sirve para englobar los que hemos visto hasta el momento, ya que es capaz de modelar una serie de tiempo arbitraria, sin importar la falta de estacionariedad debida a las causas estudiadas. Una vez vistos los procesos de esta subsección, estamos en condiciones de explicar las fases de la metodología, en la que se tomarán estos procesos y se ajustarán a los datos, dando lugar a los modelos mediante los cuales llevaremos a cabo las predicciones.

2.1.2. Fases de la metodología Box-Jenkins

Como se ha indicado previamente, una vez vista la forma en la que la metodología Box-Jenkins es capaz de lidiar con las distintas características de la serie, debemos ver más en profundidad el procedimiento que la compone. Este procedimiento está formado por cuatro fases, a saber: la fase de identificación, la fase de estimación, la fase de diagnosis y la fase de predicción. A continuación profundizaremos sobre las mismas, ilustrándolas en base a un conjunto de datos de turismo en Australia. Se trata de datos con una frecuencia trimestral, medidos en miles de viajes (a pesar de que posteriormente se cambia la medida a millones de viajes), que muestran el volumen total de personas que viajaron dentro de Australia con propósito vacacional. Estos datos se obtuvieron de Hyndman et al. (2025), y nos van a servir, como decíamos, para ilustrar las fases con un ejemplo sencillo y visual.

Fase de identificación

Esta primera fase consiste en determinar qué proceso de los que hemos visto en la subsección previa se ajustaría mejor a las características de la serie temporal con la que se está trabajando. Esto implica determinar las pautas de la tendencia y la componente estacional, para lograr mitigarlas y seleccionar los órdenes adecuados para nuestro ARIMA. Para este fin, será esencial la información que nos proporcionen la fas y la fap muestrales ($\hat{\rho}_k$ y $\hat{\alpha}_k$, respectivamente), definidas en la Sección 1.1 del Capítulo 1, además de la propia representación de la serie.

Así, una vez se tiene la representación gráfica de la serie se debe que juzgar, primeramente, si se ve homocedástica o heterocedástica. En caso de que se de el segundo de los casos, es habitual aplicar una transformación a la misma. Existen diversas opciones, pero la transformación logarítmica es la más extendida para mitigar esta característica que pone en peligro la estacionariedad de la serie. Después de esto, es cuando se juzga si la serie presenta tendencia y/o componente estacional. Si la serie presenta tendencia, algo que se observa mirando si los retardos decrecen lentamente en la fas muestral, se deben aplicar d diferenciaciones regulares a la misma, hasta lograr corregirla. Después, si presenta componente estacional, deberemos aplicar las D diferencias estacionales a la serie resultante, hasta lograr paliar el efecto de la misma.

Llegados a este punto ya estaremos ante una serie estacionaria por lo que representaremos sus fas y fap muestrales y observaremos los retardos que más repunten en ambos casos para determinar los coeficientes p, P, q y Q. Para la asignación de los dos primeros, es decir, los asociados a la parte autorregresiva, nos fijaremos en el gráfico asociado a la fap muestral. En particular, para la asignación de P nos fijaremos en el último retardo que sea múltiplo de s y que repunte significativamente (es habitual que se muestre una banda de confianza representando los intervalos de la Ecuación (2.1), por lo que significativamente quiere decir por encima de esta banda), y para p lo mismo, pero considerando únicamente los $s/2$ primeros retardos. La determinación de q y Q es totalmente análoga, solo que esta vez, para la parte de medias móviles, nos fijamos en el gráfico asociado a la fas muestral. Un último hecho importante que se debe comentar es que con este juicio visual no se podrá proponer un proceso en el que p y q y/o P y Q sean no nulos simultáneamente.

La significación que hemos mencionado cuando hemos hablado de los repuntes de los retardos no es arbitraria, sino que se fundamenta en propiedades asintóticas de los estimadores, bajo la hipótesis de que el proceso es ruido blanco. La explicación de esto la podemos encontrar en Aneiros Pérez (2024). Se afirma que bajo ciertas condiciones generales, como puede ser un tamaño muestral, n, grande, se cumple que:

$$\{X_t\} \text{ incorrelado} \Rightarrow \begin{cases} \hat{\rho}_k \approx N\left(0, \dfrac{1}{\sqrt{n}}\right), & \forall k, \\ \hat{\alpha}_k \approx N\left(0, \dfrac{1}{\sqrt{n}}\right), & \forall k. \end{cases}$$

Por lo tanto, se pueden utilizar los dos estadísticos anteriores para resolver el contraste de hipótesis H_0 : "$\{X_t\}$ *incorrelado*" frente a H_1 : "$\{X_t\}$ *correlado*".

En el caso de tener una serie incorrelada, tanto la fas como la fap muestrales se comportan igual. Son, en esencia, ruido que oscila en torno al cero con una varianza constante de $1/n$. Así, si un proceso es un $AR(p)$, sus coeficientes de autocorrelación parcial para retardos superiores a p deberían comportarse como los de una serie incorrelada. Análogamente, en un proceso $MA(q)$, la autocorrelación simple debe presentar este comportamiento a partir del retardo q, aunque ajustando la varianza por la persistencia de los instantes previos. Por tanto, bajo ciertas condiciones generales, incluyendo n grande, se verifica que:

- $AR(p) \Rightarrow \hat{\alpha}_k \approx N\left(0, \frac{1}{\sqrt{n}}\right), \quad \forall k > p,$
- $MA(q) \Rightarrow \hat{\rho}_k \approx N\left(0, \frac{\sqrt{1+2\sum_{i=1}^{q}\rho_i^2}}{\sqrt{n}}\right), \quad \forall k > q.$

En consecuencia, se pueden emplear los estadísticos anteriores para resolver los contrastes de hipóstesis siguientes: H_0 : "$\{X_t\}$ *es* $AR(p)$" y H_0 : "$\{X_t\}$ *es* $MA(q)$", respectivamente. Así, con un nivel de significación del 5 % no se rechazará H_0 si, para cada $k > p$ y para cada $k > q$, respectivamente, se cumple que:

$$\begin{aligned} \hat{\alpha}_k &\in \left(-\frac{1{,}96}{\sqrt{n}}, \frac{1{,}96}{\sqrt{n}}\right), \\ \hat{\rho}_k &\in \left(-1{,}96\sqrt{\frac{1+2\sum_{i=1}^{q}\hat{\rho}_i^2}{n}}, 1{,}96\sqrt{\frac{1+2\sum_{i=1}^{q}\hat{\rho}_i^2}{n}}\right). \end{aligned} \tag{2.1}$$

Tomemos ahora nuestro conjunto de datos de turismo en Australia. En la gráfica de la izquierda de la Figura 2.2 podemos encontrar una representación de nuestra serie de tiempo. Se puede argumentar la existencia de cierta heterocedasticidad, ya que parece que la amplitud de las oscilaciones varía con el nivel de la misma, por lo que se aplicará una transformación logarítmica a la serie. Además, se requerirá una diferenciación tanto regular, dada la presencia de tendencia, como estacional, ya que parece haber un patrón cíclico trimestral claro, es decir, $s = 4$. La gráfica de la derecha de la Figura 2.2 nos muestra la serie estacionaria, una vez aplicadas la transformación y las diferencias, tanto la regular como la estacional. En ella parecen haber desaparecido tanto la heterocedasticidad, como la tendencia y la componente estacional. Se pueden apreciar unos tramos centrales más amplios, quizá debido a atípicos.

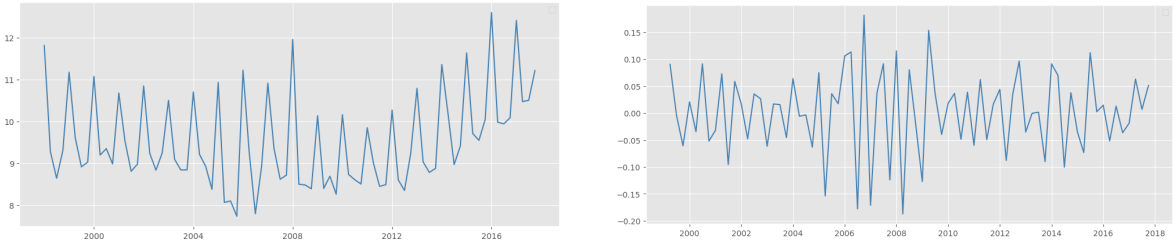


Figura 2.2: La gráfica de la izquierda muestra una representación gráfica de la serie de tiempo de interés. El eje X es el eje temporal con frecuencia trimestral, mientras que el eje Y representa el volumen total de personas que viajaron dentro de Australia con propósito vacacional, medido en millones de personas. La gráfica de la derecha nos muestra la misma serie una vez aplicada la transformación logarítmica, la diferencia regular y la diferencia estacional.

Una vez tenemos la serie estacionaria, representamos gráficamente las fas y fap, equivalentemente ACF y PAFC, muestrales, que se pueden encontrar en la Figura 2.3. Dado que, como se ha indicado previamente, no se pueden extraer conclusiones de ambos gráficos simultáneamente, nos centraremos, por ejemplo, en el ACF. Siguiendo la lógica indicada previamente, si observamos los $s/2$ primeros retardos, vemos que solamente el primero sale de las bandas. De la misma forma, si nos fijamos en los retardos que son múltiplos de $s = 4$, vemos que nuevamente solo sale el primero. En base a este criterio, proponemos un ARIMA$(0, 1, 1) \times (0, 1, 1)_4$. Ejecutando ahora la función *auto_arima*, la cual genera automáticamente el modelo más adecuado en base al AIC para la serie proporcionada, se llega al mismo modelo que hemos propuesto en base al ACF, solo que considerando adicionalmente un intercepto, denotado por μ.

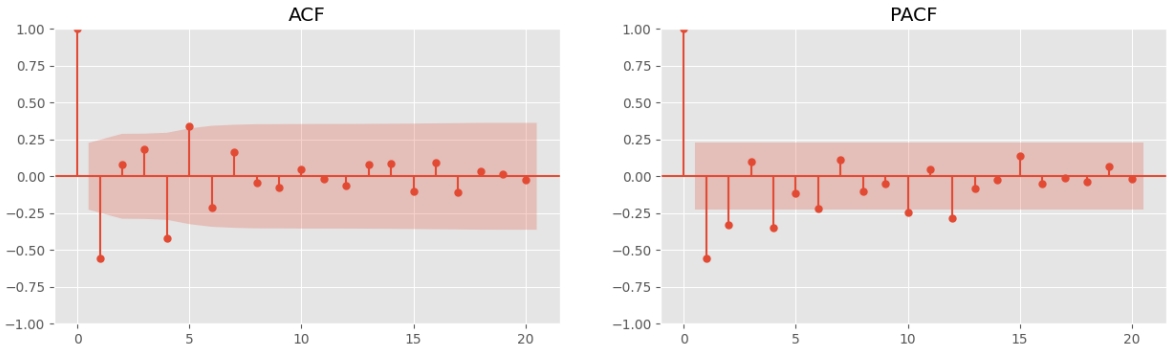


Figura 2.3: Funciones de autocorrelación simple (ACF) y autocorrelación parcial (PACF). Las bandas sombreadas indican el intervalo de confianza (típicamente al 95 %).

Fase de estimación

En esta segunda fase se estiman por máxima verosimilitud los coeficientes de proceso seleccionado de forma provisional en el paso previo. Además, se debe determinar si hay atípicos dentro de nuestra serie de tiempo, que puedan alterar el funcionamiento. Un valor atípico es un valor con poca probabilidad de ocurrencia dada la estructura habitual de evolución de la serie, y los diferenciaremos en dos clases. Si denotamos por $\{Y_t\}$ al proceso del que proviene la serie contaminada por el efecto del atípico y $\{X_t\}$ al proceso del que proviene la serie libre del efecto del valor atípico:

- Atípicos Aditivos: El valor de la serie en el instante $t = h$ ha sido generado de manera distinta al resto. Por lo tanto, se tiene que: $Y_t = w_A I_t^{(h)} + X_t$.
- Atípicos Innovativos: El valor de la innovación en el instante $t = h$ ha sido generado de manera distinta al resto. Así, las innovaciones del proceso $\{Y_t\}$ son: $e_t = w_I I_t^{(h)} + a_t$, siendo a_t las innovaciones del proceso $\{X_t\}$.

Por lo tanto, vemos que los valores atípicos modifican o bien el valor de la serie o de las innovaciones de la misma en un instante de tiempo determinado, para después dejar que continúen con el proceso natural. En la práctica, la idea para la detección de atípicos es buscarlos sobre la totalidad de los datos, si se encuentra algún atípico aditivo, se incorpora su efecto al modelo que hemos ajustado en este mismo apartado, y si se encuentra algún atípico innovativo, se retira del conjunto para que no se vuelva a detectar y se repite el proceso hasta que no queden más.

Volviendo sobre nuestro ejemplo de turismo en Australia, hemos encontrado un atípico aditivo en la fecha: 1998-01-01. Dado que se trata de un AO, incorporaremos su efecto al modelo, cuyos parámetros estimados por la función *auto_arima* son: $\mu = 0.0008$, $\theta_1 = -0.8380$, $\Theta_1 = -0.8677$ y $\sigma^2 = 0.0020$.

Por su parte, el coeficiente estimado para el atípico es AO = 0.0366, valor que tiene una interpretación directa. Dado que estamos trabajando con una serie de logaritmos, el coeficiente mostrado para el atípico implica que este valor fue $e^0.0366 - 1 \approx 0.0373$, es decir un 3.73 % más alto de lo esperado por el modelo. Una vez hecho esto, estamos en condiciones de pasar a la fase de validación o diagnosis.

Fase de diagnosis

Da comienzo la tercera fase, en la que se pondrá a prueba el modelo para ver si podemos tomarlo de forma definitiva. Para ello estudiaremos los residuos del mismo, comprobando si existen estructuras de dependencia o si por el contrario siguen un proceso de ruido blanco. Es decir, nuestro objetivo en esta fase es determinar si las innovaciones tienen media cero, varianza constante y están incorreladas. Si no verifica esto, se descarta el modelo seleccionado como posible generador de la serie de tiempo considerada, y se vuelve a la primera fase para buscar un nuevo candidato. Para comprobar si hay incorrelación se lleva a cabo el contraste de Ljung-Box, cuya hipótesis nula es $H_0 = \rho_1 = \cdots = \rho_H = 0$. Por lo tanto, rechazaremos H_0 frente a H_1 si existe un valor $\rho_j \neq 0$ para algún $j \in \{1, \ldots, H\}$, donde H es el número total de retardos que el test analiza de forma conjunta. Para comprobar si la media es cero se emplea un nuevo contraste. La hipótesis nula de este último sera $H_0 : \mu_a = 0$, por lo que rechazaremos H_0 frente a H_1 cuando μ_a sea significativamente distinta de cero.

Además, en este apartado es habitual tratar de ver si las innovaciones siguen una distribución normal. Esta una condición que aporta ciertas ventajas, como pueden ser que bajo normalidad el ruido blanco equivale a la independencia o que los estimadores utilizados (máxima verosimilitud gaussiana), son asintóticamente eficientes, pero que no es necesaria estrictamente hablando para la validación del modelo, como puede ser el carácter estacionario mencionado en el párrafo previo. Es decir, es conveniente pero no necesaria. Para su comprobación, como podemos ver en Aneiros Pérez (2024), se puede emplear el contraste de normalidad de Jarque-Bera, cuya hipótesis nula es que la serie de innovaciones es gaussiana. Por último, cabe destacar que además de contrastes, se emplean gráficas para asesorar la validez del modelo, como pueden ser un correlograma o una gráfica que muestre la evolución temporal de los residuos estandarizados, en la que además podremos ver si la oscilación de estos es homocedástica. También se suelen proporcionar un histograma y un QQ-plot para ver si las innovaciones se ajustan a la distribución normal estándar.

Retomando el ejemplo de Australia, tenemos que nuestro modelo supera con éxito tanto el test de Ljung-Box, como el de media cero, con p-valores respectivos de 0.3290 y 0.7295, convirtiéndolo en válido y permitiéndonos continuar con la fase de predicción. Sin embargo, no podemos asumir normalidad para los residuos de nuestro modelo, ya que el p-valor asociado al test de Jarque-Bera es 0. Estas conclusiones se ven respaldadas gráficamente por el panel que tenemos en la Figura 2.4. Las gráficas de arriba a la derecha y de abajo a la izquierda nos muestran que nuestros residuos no se adecúan a una distribución normal, mientras que las de arriba a la izquierda y abajo a la derecha nos permiten ver la incorrelación y media cero de los mismos, es decir, que son ruido blanco.

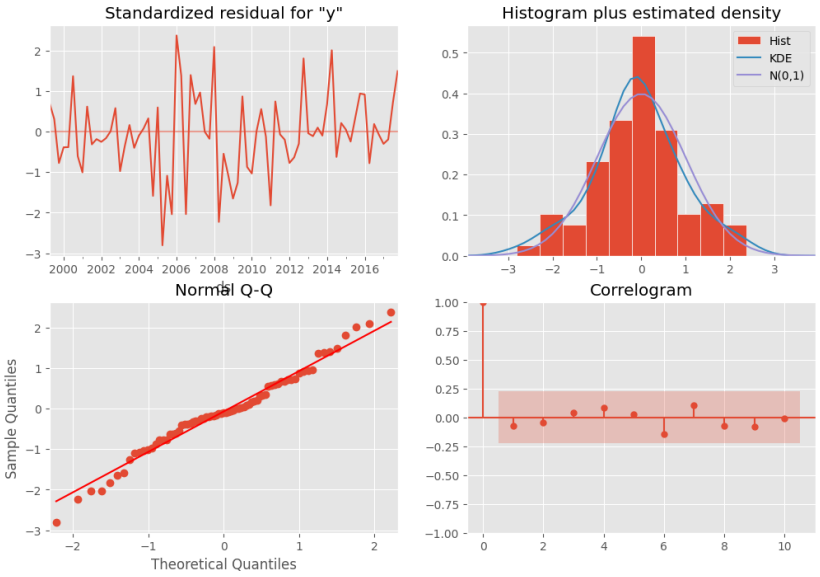


Figura 2.4: Panel de diagnóstico de los residuos del modelo ajustado. De arriba a abajo e izquierda a derecha: Gráfico que muestra residuos estandarizados en el tiempo, histograma y curva de densidad estimada (KDE) junto a la densidad de la distribución normal estándar, QQ-Plot, cuyo eje X representa los cuatiles teóricos, es decir, dónde deberían estar ubicados los datos si siguieran una distribución normal y cuyo eje Y representa dónde están situados realmente los residuos estandarizados, una vez ordenados de menor a mayor y por último correlograma de los residuos.

Fase de predicción

Con esto da comienzo la última fase, la fase de predicción. En esta, una vez obtenido un modelo adecuado, validado en la fase previa, se realizan las predicciones en base al mismo. Es habitual también, además de obtener y representar las predicciones, mostrar un intervalo de confianza para las mismas que nos ayude a medir la incertidumbre. Estas predicciones serán una buena forma de comprobar si el modelo ha sido capaz de capturar satisfactoriamente las características de la serie estudiada, como puede ser la tendencia o la componente estacional.

Las predicciones para el ejemplo considerado a lo largo de la presentación de las fases se pueden encontrar en la Figura 2.5. En ella vemos cómo el modelo ha sido capaz de captar el patrón de la serie de partida, ajustándose a su tendencia y componente estacional. Se muestran además los intervalos de confianza, que nos permiten hacernos una idea de la incertidumbre asociada a las predicciones.

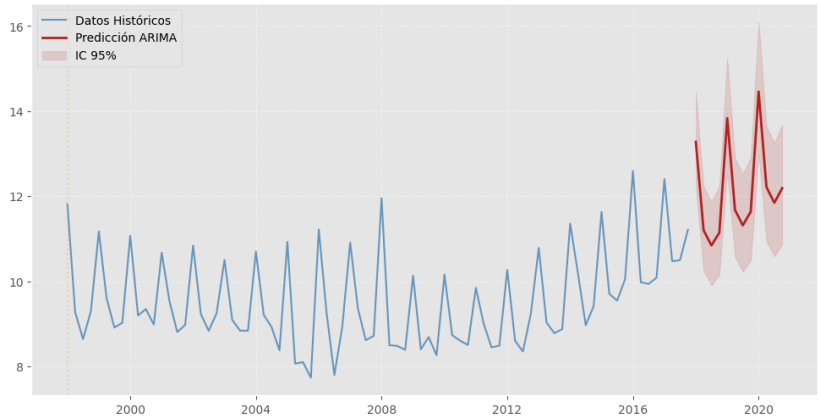


Figura 2.5: Gráfica que muestra las predicciones realizadas en base al modelo ARIMA seleccionado y validado para nuestra serie de tiempo, junto con un intervalo de confianza sombreado para las mismas.

Ahora que hemos adquirido un conocimiento más profundo de esta metodología, podemos expandir la gráfica vista en la Figura 2.1, mostrando una representación más completa del procedimiento.

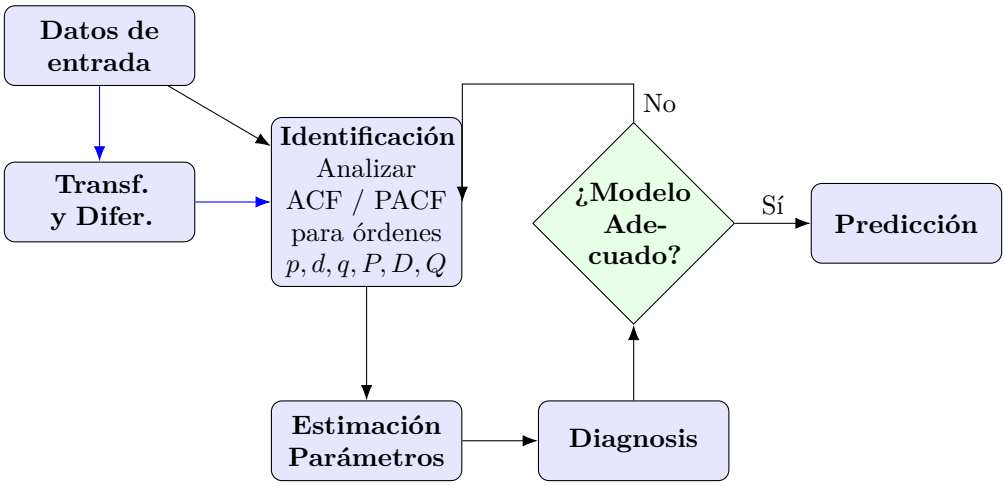


Figura 2.6: Diagrama de flujo que ilustra la metodología iterativa de Box-Jenkins para el modelado de series temporales ARIMA. Las líneas en azul indican que no siempre es necesario recurrir a ese camino.

2.2. Árboles de decisión

Los árboles de decisión representan uno de los métodos más sencillos y de mayor interpretabilidad para la realización de predicciones en problemas tanto de clasificación como de regresión. Se basan en la idea de particionar el espacio de variables predictoras en regiones muy simples y de forma que que el proceso se pueda representar mediante un árbol binario de decisión como el que se puede encontrar en la Figura 2.7.

En particular, nos centramos en la metodología CART (*Classification and Regression Trees*), que es la más popular en lo relativo a los árboles de decisión. Una de las ventajas que nos empujan al uso de

esta metodología es que es capaz de lidiar con datos faltantes dentro de las variables explicativas. La forma de hacerlo consiste en establecer particiones sustitutas, conocidas como *surrogate splits*, de tal forma que si hay un dato faltante en una variable que determina una división, se usa una alternativa que de lugar a una partición similar.

Por último, antes de adentrarnos más en el estudio del método, cabe destacar que las referencias principales empleadas para la obtención de la información han sido: Breiman et al. (1984), Hastie et al. (2009) y Fernández-Casal et al. (2024).

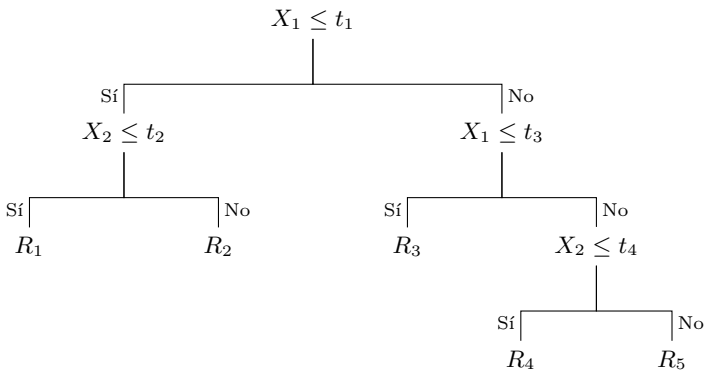


Figura 2.7: Divisiones de un árbol de decisión.

La Figura 2.7 nos sirve para ejemplificar el proceso. Como vemos, se parte de un nodo inicial, al que llamamos nodo raíz, que contiene el 100 % de la muestra de entrenamiento, y se realiza una primera división. Para realizarla, se toma la variable X_1 y se fija un valor, t_1, de forma que si el valor que toma esta variable es superior o igual al mismo, partira por una rama, mientras que si es inferior, se le asignará otra distinta. Ahora, cada camino llega a un nodo diferente, que cuenta con un porcentaje distinto de los datos de entrenamiento. Posteriormente, en los nodos a los que se llega, se realiza una nueva división, ya sea sobre la misma variable tomando un valor diferente, u optando por una variable distinta. Como podemos ver en la Figura 2.7, el proceso explicado se repite un número finito de veces hasta obtener los llamados nodos terminales u hojas, que tanto en esta figura como en la Figura 2.8 denotamos por R_i y son las regiones finales que se obtienen de la división repetida del árbol. En ellas se lleva a cabo la predicción, de forma que en problemas de regresión, que es el tipo de problemas que nos ocupa, será el promedio de los valores que han caído dentro de cada una.

Cuando este proceso finaliza, el espacio predictivo está secciónado en regiones rectangulares, dentro de las cuales, la predicción de la respuesta es, como decíamos, constante. En la Figura 2.8 se muestra la partición inducida por las divisiones ejemplificadas en la Figura 2.7.

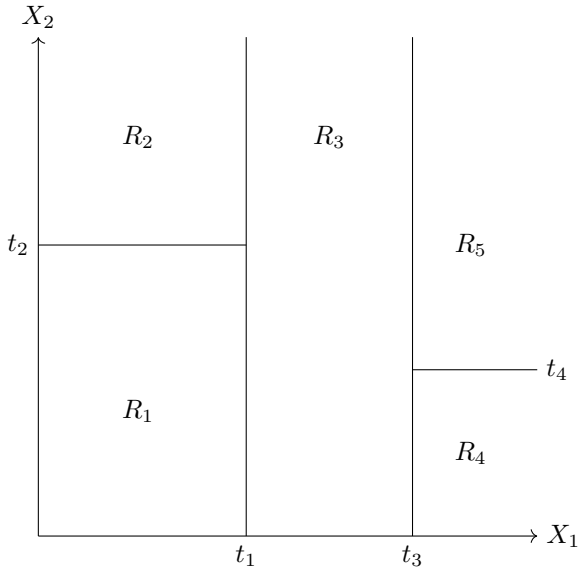


Figura 2.8: Partición inducida por el árbol de decisión previo.

Cabe destacar que si la relación entre las variables predictoras y la variable respuesta no se puede explicar de forma satisfactoria mediante rectángulos, la calidad predictiva del árbol se verá claramente mermada.

Como consecuencia de lo anterior, supongamos que tenemos una partición en M regiones, que denotamos por $R_1, R_2, \ldots, R_M$, y que en cada una de ellas modelamos la respuesta de forma constante, c_m. De esta forma, podemos definir la siguiente función f, que caracteriza las predicciones que nuestro modelo realizará:

$$f(x) = \sum_{m=1}^{M} c_m I(x \in R_m).$$

Vemos que esta función predice el valor c_m para los valores que caen en la región R_m, valiéndose de la función indicadora. Por último, antes de proceder con el proceso de crecimiento del árbol, cabe destacar un hecho importante acerca del sesgo y la varianza del modelo. Es buen momento comentarlo dado que estará íntimamente relacionado con el número de regiones o nodos terminales a los que se termina llegando. Un árbol de decisión que llega a tener muchos nodos terminales, es un árbol de decisión muy especializado. Además es más complejo, ya que ha realizado un número mayor de cortes y como consecuencia de esto, es un modelo con poco sesgo, pero mucha varianza. Análogamente, un árbol con pocas divisiones, y pocos nodos terminales es un modelo muy simple, con mucha varianza y poco sesgo. En consecuencia, las dos próximas subsecciones en esencia estarán lidiando con el problema del punto de encuentro entre sesgo y varianza de forma implícita.

2.2.1. Crecimiento del árbol

Abordemos la cuestión de cómo hacer que los árboles de regresión crezcan. Comenzamos el proceso con p entradas de datos, que serán nuestras variables explicativas, y una respuesta que denotaremos por Y, para cada uno de las n observaciones disponibles. Matemáticamente traduciríamos esto como: (x_i, y_i), con $i = 1, 2, \ldots, n$, y donde $x_i = (x_{i1}, x_{i2}, \ldots, x_{ip})'$. Como mencionabamos anteriormente, el algoritmo necesita decidir de forma automática sobre qué variables, y qué valores de las mismas dividirá el árbol.

Teorema 1. *Si adoptamos el criterio de mínimos cuadrados para minimizar el error de predicción:*

$$\sum_{i=1}^{n}(y_i - f(x_i))^2, \tag{2.2}$$

se puede comprobar que la elección óptima para c_m, $\hat{c}_m$, viene dada por la media de y_i en la región R_m.

Demostración. Partimos de la expresión vista en la Ecuación (2.2). Si sustituimos la función f por la expresión presentada anteriormente, nos quedaría lo que sigue.

$$\sum_{i=1}^{n}(y_i - f(x_i))^2 = \sum_{i=1}^{n}(y_i - \sum_{m=1}^{M} c_m I(x_i \in R_m))^2.$$

Tengamos en cuenta ahora que cada x_i pertenece a una única región, por haberse dividido el espacio de variables predictoras de una partición disjunta. Sea $R_{m(i)}$ dicha región. Entonces, podemos denotar $f(x_i) = c_{m(i)}$, y como consecuencia de esto último, la Ecuación (2.2) se transforma en lo siguiente.

$$\sum_{i=1}^{n}(y_i - c_{m(i)})^2.$$

Ahora separamos la suma en función de la región en la que cae x_i. Denotemos por I_m al conjunto de índices de los elementos que caen en R_m, de forma que: $I_m = \{i \mid m(i) = m\}$. De esta forma, podemos transformar la expresión de la ecuación previa en:

$$\sum_{i=1}^{n}(y_i - c_{m(i)})^2 = \sum_{m=1}^{M}\sum_{i\in I_m}(y_i - c_m)^2.$$

Notemos ahora que c_m solamente aparece en el término que corresponde a su región, por lo que podemos minimizar cada región de forma independiente. Así, fijamos la región R_m y suponemos sin pérdida de generalidad que $|I_m| = J$. En este caso, buscamos resolver el problema de optimización:

$$\min_{c} \sum_{i\in I_m}(y_i - c)^2.$$

Si derivamos la expresión anterior con respecto a c_m, tenemos lo siguiente.

$$\frac{d}{dc}\sum_{i\in I_m}(y_i - c)^2 = -2\sum_{i\in I_m}(y_i - c).$$

Igualando a cero la expresión anterior, llegamos al resultado que podemos encontrar a continuación.

$$-2\sum_{i\in I_m}(y_i - c) = 0 \Leftrightarrow \sum_{i\in I_m}(y_i - c) = 0 \Leftrightarrow \sum_{i\in I_m} y_i = \sum_{i\in I_m} c.$$

Notemos ahora que c es constante, por lo que $\sum_{i\in I_m} c = Jc$, y entonces:

$$\sum_{i\in I_m} y_i = Jc \Leftrightarrow \hat{c}_m = \frac{1}{J}\sum_{i\in I_m} y_i.$$

Es decir, $\hat{c}_m$ es la media de los y_i tales que $x_i \in R_m$. □

Del Teorema 1 deducimos que para el crecimiento del árbol, se deben seleccionar las regiones $R_1, R_2, \ldots, R_m$ que minimicen la siguiente expresión:

$$\sum_{m=1}^{M} \sum_{i \in I_m} (y_i - \hat{c}_m)^2.$$

Pero este problema es poco razonable e intratable en la práctica, ya que a pesar de las restricciones que puedan tener nuestras regiones, buscaría valores de M demasiado grandes. El método CART propone una alternativa avariciosa que busca un compromiso entre rencimiento, sencillez e interpretabilidad. Así, en lugar de hacer una búsqueda por todas las particiones posibles sigue un proceso recursivo en el que realiza cortes binarios. Comenzando con la totalidad de los datos, consideramos una variable y un punto para realizar la división, que denotaremos por j y s, respectivamente. Mediante los mismos, se define un par de semiplanos de la forma:

$$R_1(j, s) = \{x \mid x_j \leq s\} \text{ y } R_2(j, s) = \{x \mid x_j > s\}.$$

En consecuencia, buscamos los j y s que sean solución del problema de optimización:

$$\min_{j,s} \left(\min_{c_1} \sum_{x_i \in R_1(j,s)} (y_i - c_1)^2 + \min_{c_2} \sum_{x_i \in R_2(j,s)} (y_i - c_2)^2 \right).$$

Por el resultado probado en el Teorema 1, sabemos el resultado de la minimización de dentro del paréntesis para cualquier elección de j y s, por lo que el problema que abordamos se resumiría como sigue.

$$\min_{j,s} \left(\sum_{x_i \in R_1(j,s)} (y_i - \hat{c}_1)^2 + \sum_{x_i \in R_2(j,s)} (y_i - \hat{c}_2)^2 \right),$$

donde $\hat{c}_i$ es el promedio visto anteriormente. A diferencia del problema considerado originalmente, este tiene una solución rápida, ya que para cada variable de corte j, la determinación del punto s se puede encontrar de forma relativamente fácil, dado que solo hay un número finito de valores s que generen conjuntos $R_1(j, s)$ y $R_2(j, s)$. Por lo tanto, mediante el escaneo de todas las variables de entrada, encontrar el par (j, s) óptimo es posible. Una vez encontrada la escisión óptima, particionamos los datos entre las dos regiones resultantes y repetimos el proceso para cada una de ellas, continuando de forma sucesiva un número finito de veces.

2.2.2. Poda del árbol

La siguiente pregunta que podríamos hacernos es cuándo parar, es decir, cuándo dejamos de hacer crecer el árbol. Por un lado, como decíamos antes, si lo dejamos crecer demasiado dará lugar a un modelo demasiado complejo, difícil de interpretar y además se producirá un sobreajuste a los datos de entrenamiento. Esto quiere decir que cuando se evalúe el modelo fuera de la muestra con la que este se ha entrenado, los resultados serán malos. Es decir, lo que tendremos será un modelo con poco sesgo, pero mucha varianza, y como consecuencia de esto, un modelo inestable, ya que pequeños cambios en los datos dan lugar a modelos muy diferentes. Por otro lado, si paramos muy pronto obtenemos un modelo demasiado simple que será más interpretable y con menos varianza, es decir, menos inestable, pero con un gran sesgo, lo que hace que deje de ser una opción atractiva, ya que no se estaría capturando la estructura interna de los datos. Dicho de otra forma, el problema del tamaño del árbol en realidad resulta ser el problema clásico de buscar el equilibrio entre sesgo y varianza.

A pesar de haber diversas opciones utilizadas en la práctica, nos centraremos en la llamada *cost-complexity pruning*. Esta estrategia consiste en hacer crecer un árbol muy grande de forma que se minimice el error de entrenamiento, hasta que cada nodo t tenga un número de observaciones $N(t) \leq N_{min}$, donde habitualmente $N_{min} = 5$. Después, se toma el árbol completo, que denotaremos por T_0, y se lleva a cabo una poda. Definimos este concepto de la forma siguiente.

Definición 9. *El concepto de poda se refiere a colapsar cualquier cantidad de sus nodos internos, es decir, aquellos que no son terminales, dando lugar a otro ábol más pequeño que llamaremos subárbol del original.*

De la Definición 9 se desprendela necesidad de definir un nuevo concepto, el de subárbol.

Definición 10. *Denotemos por T_0 al árbol completo. Un subárbol $T \leq T_0$ se define como cualquier árbol que puede obtenerse colapsando un número determinado de sus nodos internos.*

Recordemos que la poda se realiza para buscar un equilibrio entre el sesgo y la varianza, es decir, buscando el subárbol que de lugar a un menor error al ser evaluado en datos que el árbol de decisión no ha visto durante el entrenamiento. Debido al gran número de subárboles, resulta evidente la imposibilidad de evaluarlos todos. Este problema se suple con el uso de un hiperparámetro, mediante el cual se controla el tamaño del árbol, o lo que es lo mismo, la complejidad del modelo, seleccionando el subárbol que resulta óptimo para los datos disponibles.

Sea el subárbol $T \leq T_0$ arbitrario, cumpliendo la Definición 10, obtenido de la poda del árbol original que da lugar a las regiones $R_1, R_2, \ldots, R_{|T|}$, donde $|T|$ es el número de nodos terminales del mismo (también se suele denominar la complejidad del subárbol). Consideremos, como hemos hecho hasta el momento, la suma de residuos al cuadrado como medida del error, pero ahora le añadiremos una penalización que depende de un hiperparámetro no negativo, llamado parámetro de complejidad del modelo y denotado por $\alpha \geq 0$:

$$C_\alpha(T) = \sum_{m=1}^{|T|} \sum_{x_i \in R_m} (y_i - \hat{c}_m)^2 + \alpha|T|, \tag{2.3}$$

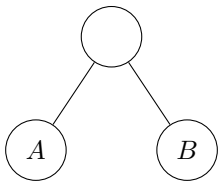
donde $\hat{c}_m$ sigue siendo el valor óptimo obtenido en el Teorema 1. Por un lado, tenemos un sumatorio que mide cómo de bien se ajusta el árbol a los datos de entrenamiento, cuyo error disminuye a medida que el árbol se hace más grande y por lo tanto más específico. Por otro lado $\alpha|T|$ es la penalización por complejidad, y aumenta con el número de nodos terminales del árbol. Notemos que $C_\alpha(T)$ es una combinación lineal del coste asociado al subárbol y de la complejidad del mismo, por lo que en lo que sigue nos referiremos a la misma como coste. Además, cabe destacar que se ha tomado la suma de residuos al cuadrado como medida de error asociado al árbol por coherencia con lo visto anteriormente. Aun así, para una mayor generalidad, en lo que sigue definiremos la función de coste asociada a un subárbol T como $R(T)$, pudiendo el lector asumir la igualdad de ambas expresiones si lo desea. De esta forma, ahora tenemos que:

$$C_\alpha(T) = R(T) + \alpha|T|.$$

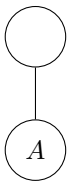
Para cada valor del parámetro α, existe un único subárbol más pequeño que minimiza este error. Si lo denotamos por $T(\alpha)$ para un valor dado α del parámetro de complejidad, lo definimos como aquel que cumple las condiciones siguientes:

1. $C_\alpha(T(\alpha)) = \min_{T \leq T_0} C_\alpha(T)$,
2. Si $C_\alpha(T) = C_\alpha(T(\alpha))$, entonces $T(\alpha) \leq T$.

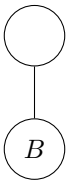
La condición 2 rompe los posibles empates que puedan darse en la minimización de C_α a base de seleccionar el más pequeño de ellos. Observemos que esta definición se apoya en el hecho mencionado de que el subárbol minimizador debe ser único, algo que comprobaremos a continuación. Lo primero es tener en cuenta que aunque α toma valores continuos, solamente existe una cantidad finita de subárboles, ya que si tenemos un árbol con, supongamos, M nodos, solamente disponemos de una cantidad combinatoria finita de formas de podarlo. Visualicémoslo con el ejemplo más sencillo, considerando 3 nodos totales, uno raíz y dos terminales. En ese caso, las opciones válidas se encuentran representadas en la Figura 2.9.



Opción A: Árbol original



Opción B: Subárbol 1



Opción C: Subárbol 2

Opción D: Subárbol 3 (Solo raíz)

Figura 2.9: Ejemplo visual de las opciones válidas de poda en un árbol con tres nodos.

Resulta inmediato extender esto a un número mayor de nodos. Cabe destacar que si el número en cuestión es mucho mayor, las opciones serán tremendamente abundantes, pero finitas. Por esta finitud que acabamos de ver ilustrada en un ejemplo trivial, lo que ocurre es que si $T(\alpha)$ es el subárbol minimizador para un valor de α dado, lo seguirá siendo para los siguientes valores continuos del parámetro hasta que se alcance un punto de salto en un nuevo valor, α', a partir del cual surgirá un nuevo subárbol minimizador, $T(\alpha')$, que lo seguirá siendo hasta llegar al siguiente punto α''. Lo siguiente es probar nuestro objetivo inicial contando con la hipótesis de que el número de subárboles posibles es finito.

Teorema 2. *Para cada valor de α, existe un único subárbol más pequeño que minimiza $C_\alpha(T)$.*

Demostración. Sean τ el conjunto finito de todos los subárboles obtenidos por la poda de un árbol inicial maximal y $\alpha_0 \geq 0$ un valor fijo del parámetro de complejidad. Tenemos que ver que el conjunto de subárboles que minimizan $C_{\alpha_0}(T)$, denotado por $M_{\alpha_0} = \{T \in \tau / C_{\alpha_0}(T) = \text{mín}\}$ tiene un elemento único bajo la relación "ser subárbol de".

La clave está en demostrar que si dos subárboles arbitrarios, T_1 y T_2, son minimizadores de $C_{\alpha_0}(T)$, es decir, $T_1, T_2 \in M_{\alpha_0}$, entonces $T_1 \cap T_2$ también lo es. Lo primero antes de continuar con la demostración es definir qué árboles resultan de la intersección y de la unión de T_1 y T_2. Para ello, consideremos la siguiente ecuación:

$$N(T_1 \cap T_2) = \{t \in N(T_{max}) \mid t \in N(T_1) \wedge t \in N(T_2)\},$$
$$N(T_1 \cup T_2) = \{t \in N(T_{max}) \mid t \in N(T_1) \vee t \in N(T_2)\},$$

donde $N(T)$ representa el conjunto de nodos que forman un árbol. De esta forma, tenemos que el subárbol formado por la intersección de los dos estará compuesto por los nodos presentes simultáneamente en los dos árboles. De esta forma, los nodos terminales de la intersección se encontrarán en los nodos donde la poda de T_1 o la de T_2 ocurrió más cerca de la raíz. Por el contrario, el subárbol unión estará formado por los nodos pertenecientes al menos a uno de los dos árboles. Los nodos terminales en este caso serán aquellos de T_1 o T_2 que estén a mayor profundidad en el árbol original.

Notemos que $T_1 \cap T_2$ es el árbol que contiene solo los nodos que son nodos internos tanto en T_1 como en T_2. Además, dado que los errores de estos dos árboles dependen únicamente de sus nodos terminales, podemos descomponer sus errores y tamaños basándonos en sus diferencias. Debido a la estructura aditiva del error en las hojas, se cumple una propiedad de supermodularidad:

$$\begin{aligned} R(T_1 \cup T_2) + R(T_1 \cap T_2) \leq R(T_1) + R(T_2), \\ |T_1 \cup T_2| + |T_1 \cap T_2| = |T_1| + |T_2|. \end{aligned} \tag{2.4}$$

La base de esta idea es que si tenemos un nodo arbitrario y lo dividimos en dos, el error de los hijos sumado siempre será menor o igual que el error del padre por sí solo. Como decíamos antes, a medida que el árbol crece, el error baja. Suponiendo que T_1 y T_2 no sean iguales, la unión de ambos da lugar al árbol más especializado de los cuatro, ya que se queda con todas las divisiones que hicieron ambos, mientras que la intersección es el más conservador de los cuatro, ya que solo mantiene las divisiones que ambos compartían. La idea de la desigualdad de la Ecuación (2.4) se puede explicar matemáticamente como sigue. Sea $R(root)$ el error inicial máximo cometido en el nodo raíz de un árbol. Cada vez que el árbol hace una división en un nodo arbitrario t, el error se reduce en $\delta(t) \geq 0$. Así, el error total de cualquier árbol se puede calcular como el error inicial menos las mejoras logradas al dividirse:

$$R(T) = R(root) - \sum_{t \in Int(T)} \delta(t), \tag{2.5}$$

donde $Int(T)$ representa el conjunto de nodos internos del árbol T. Ahora, aplicando el conjunto de inclusión-exclusión a las mejoras del error del caso considerado, obtendríamos:

$$\sum_{Int(T_1 \cup T_2)} \delta(t) + \sum_{Int(T_1 \cap T_2)} \delta(t) = \sum_{Int(T_1)} \delta(t) + \sum_{Int(T_2)} \delta(t).$$

Es inmediato ver que lo anterior es equivalente a la siguiente expresión

$$R(T_1 \cup T_2) + R(T_1 \cap T_2) = R(T_1) + R(T_2) \Rightarrow R(T_1 \cup T_2) + R(T_1 \cap T_2) \leq R(T_1) + R(T_2).$$

Es importante destacar que la igualdad que acabamos de probar se cumple por que estamos trabajando dentro del marco de la metodología CART, en la que el error es aditivo en las hojas, y por la Ecuación (2.5), que surge de haber supuesto que T_1 y T_2 surgen del mismo árbol completo. Es habitual ver el símbolo de desigualdad por un hecho de generalidad matemática y por que es la condición mínima necesaria para que funcione la poda.

Retomemos el proceso, donde lo habíamos dejado. Aplicando lo visto sobre la Ecuación (2.4) dentro de C_{α_0}, tenemos que:

$$C_{\alpha_0}(T_1 \cup T_2) + C_{\alpha_0}(T_1 \cap T_2) \leq C_{\alpha_0}(T_1) + C_{\alpha_0}(T_2).$$

Como consecuencia de esto, si T_1 y T_2 son ambos árboles minimizadores, se cumple que $C_{\alpha_0}(T_1) = C_{\alpha_0}(T_2) = C_{\alpha_0}$, y por lo tanto:

$$C_{\alpha_0}(T_1 \cup T_2) + C_{\alpha_0}(T_1 \cap T_2) \leq 2C_{\alpha_0}.$$

Ahora, razonando que ningún árbol puede tener un coste inferior al mínimo, que llegamos a que:

$$C_{\alpha_0}(T_1 \cap T_2) = C_{\alpha_0}.$$

Como consecuencia de esto último tenemos que la intersección de dos árboles óptimos también da lugar a un árbol óptimo, y por lo tanto, si intersecamos todos los árboles que minimizan el coste para un valor α_0 dado, se obtiene un único árbol minimizador. □

Por otra parte, se puede apreciar que cuando $\alpha = 0$, tendremos que $T = T_0$, algo que no tiene sentido considerar dada la naturaleza del problema de reducción o poda que estamos considerando. A medida que se incrementa el valor de α, se penalizan los subárboles con un mayor número de nodos terminales, dando lugar a un resultado más pequeño, evidenciando cómo este parámetro nos ayudará a buscar el punto de encuentro entre el sesgo y la varianza.

Notemos en base a los resultados expuestos hasta el momento que el punto de inicio para la poda no tiene por qué ser el árbol T_0, sino que más bien sería $T(0) = T_1$. Este representa al subárbol más pequeño de T_0 cumpliendo:

$$R(T_1) = R(T_0).$$

A continuación se estudiará el proceso de selección del árbol minimizador asociado a cada valor de α, que funciona basándose en los cortes sobre el eslabón más débil, también conocido como *weakest-link cutting*. Antes, debemos definir dos conceptos que aparecerán más adelante, los conceptos de rama de un árbol y subrama de la misma.

Definición 11. *Sea T un árbol y $t \in T$ un nodo no terminal del mismo. La rama que nace en t, denotada por T_t, se define como el subárbol que tiene a t como su nueva raíz e incluye a t y a todos sus descendientes.*

Definición 12. *Sea T_t una rama cumpliendo la Definición 11. Sea un nodo t' perteneciente a T_t sin incluir la raíz, entonces la subrama $T_{t'}$ es el nuevo subárbol que tiene como raíz a ese nodo t' y contiene a todos sus descendientes. Es decir, definimos una subrama como una rama dentro de una rama.*

Como a priori las definiciones pueden resultar confusas, podría ser interesante ilustrarlo gráficamente, ya que visualmete son dos conceptos más fáciles de comprender. Para ello, tomaremos el árbol de la Figura 2.7, ligeramente modificado para que quede una representación más completa y marcaremos una rama y una subrama del mismo. Esto se puede encontrar en la Figura 2.10.

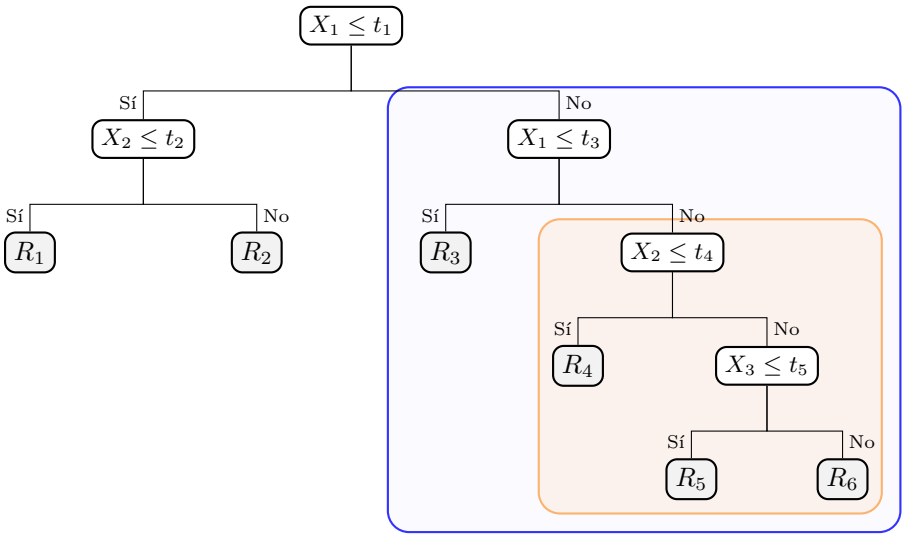


Figura 2.10: Visualización de una rama completa (azul) de un árbol y una subrama de la misma (naranja).

Ahora estamos en condiciones de definir el error asociado a una rama arbitraria de T_1, $R(T_t)$, como sigue:

$$R(T_t) = \sum_{t' \in \hat{T}_t} R(t'),$$

donde $\hat{T}_t$ es el conjunto de nodos terminales de T_t. Además, basándonos en la Sección 10.2 de Breiman et al. (1984), presentamos una propiedad que le pedimos a R, y es que cualquier nodo no terminal, t, de T_1, debe cumplir que el error del nodo es mayor o igual que la suma de los errores de sus descendientes terminales. Formalmente:

$$R(t) \geq R(T_t). \tag{2.6}$$

Comenzamos por lo tanto en el subárbol T_1. La esencia del proceso de poda que se está revisando es, como adelantábamos antes, entender que funciona cortando el eslabón más débil. Para cada nodo $t \in T_1$, denotamos por $\{t\}$ la subrama de T_t compuesta únicamente por el nodo en cuestión. De esta forma, tenemos que:

$$C_\alpha(\{t\}) = R(t) + \alpha.$$

Además, tenemos que para cualquier rama T_t definimos el coste como sigue.

$$C_\alpha(T_t) = R(T_t) + \alpha|\hat{T}_t|.$$

Siempre que se cumpla que $C_\alpha(T_t) < C_\alpha(\{t\})$, la rama T_t tiene un coste inferior que el nodo compuesto únicamente por $\{t\}$, pero para un determinado valor crítico de α, los dos costes se igualarán. En este punto la subrama $\{t\}$ es más pequeña que T_t, pero tiene el mismo coste, lo que la hace preferible. Para encontrar el mencionado valor crítico de α, solo debemos resolver la inecuación $C_\alpha(T_t) < C_\alpha(\{t\})$, obteniendo que:

$$\alpha < \frac{R(T) - R(T_t)}{|\hat{T}_t| - 1}.$$

En virtud de la propiedad presentada en la Ecuación (2.6) se puede garantizar que el valor crítico presentado a la derecha de la inecuación es positivo. Una vez visto esto, podemos definir la función $g_1(t), t \in T_1$ como sigue.

$$g_1(t) = \begin{cases} \dfrac{R(T) - R(T_t)}{|\hat{T}_t| - 1} & \text{si } t \notin \hat{T}_1, \\ \infty & \text{si } t \in \hat{T}_1. \end{cases}$$

En base a la misma podemos definir el eslabón más débil $\bar{t}_1$ en T_1 como aquel que cumple la siguiente igualdad:

$$g_1(\bar{t}_1) = \min_{t \in T_1} g_1(t).$$

Además, fijamos como nuevo umbral $\alpha_2 = g_1(\bar{t}_1)$. El nodo $\bar{t}_1$ es el eslabón más débil en el sentido de que a medida que aumenta el valor del parámetro de complejidad, es el primero en cumplir que $C_\alpha(T_t) < C_\alpha(\{t\})$. Entonces, como comentábamos antes, $\{\bar{t}_1\}$ es preferible a $T_{\bar{t}_1}$, y α_2 es el valor del parámetro de complejidad en el que eso ocurre.

Ahora definimos un nuevo árbol, $T_2 < T_1$ podando la rama $T_{\bar{t}_1}$, o lo que es lo mismo: $T_2 = T_1 - T_{\bar{t}_1}$. A contnuación buscamos el eslabón más débil en T_2 de la misma forma que acabamos de ver, definiendo de forma análoga una función g_2 y un nuevo umbral α_3. Si en algún punto damos con un empate en dos candidatos a eslabón más débil, es decir, si por ejemplo llegamos a un punto en el que ocurre: $g_k(\bar{t}_k) = g_k(\bar{t'}_k)$ entonces definimos $T_{k+1} = T_k - T_{\bar{t}_k} - T_{\bar{t'}_k}$. Reiterando este proceso a través de las sucesivas etapas obtendríamos una secuencia decreciente de subárboles: $T_1 > T_2 > T_3 > \ldots > \{t_1\}$. El proceso que acabamos de desglosar se puede resumir en el siguiente Teorema.

Teorema 3. *Consideremos $\{\alpha_k\}$ la sucesión creciente de forma que $\alpha_k < \alpha_{k+1}, k \geq 1$ y donde $\alpha_1 = 0$. Para $k \geq 1$, $\alpha_k < \alpha < \alpha_{k+1}$ se tiene que $T(\alpha) = T(\alpha_k) = T_k$.*

Dada la longitud de la prueba asociada a este resultado, si se desea comprobar la demostración del mismo se puede acudir a la Sección 10.2 de Breiman et al. (1984). El Teorema 3 resume el funcionamiento del sistema de poda estudiado, *cost-complexity pruning*. Comienza con T_1, encuentra la rama que hace las veces de eslabón más débil, $T_{\bar{t}_1}$ y poda para obtener T_2 cuando el valor del parámetro de complejidad alcanza α_2 y así se sigue sucesivamente. Además, una ventaja asociada a esta estrategia es que los pasos de poda recursivos son computacionalmente rápidos. Finalmente, cabe destacar, que el algoritmo inicialmente tiende a podar subramas grandes con muchos nodos terminales y, a medida que los árboles se hacen más pequeños, cada vez corta menos de forma simultánea.

El método de poda que acabamos de ver nos deja una sucesión decreciente de subárboles $T_1 > T_2 > T_3 > \ldots > \{t_1\}$, donde recordemos $T_k = T(\alpha_k)$ y $\alpha_1 = 0$. Nuestro objetivo ahora es el de seleccionar uno de estos como nuestro árbol de tamaño óptimo. Lo más habitual para abordar esta cuestión es proceder por validación cruzada u otro tipo de remuestreo en la muestra de entrenamiento para seleccionar el valor del parámetro α. Se puede seleccionar el valor que minimiza el error, o podemos forzar a que el modelo sea más sencillo aplicando la regla *one-standard-error*.

Concluimos la subsección indicando que también es habitual escribir la Ecuación (2.3) en los siguientes términos:

$$RSS_{\bar{\alpha}} = RSS + \bar{\alpha} RSS_0 t,$$

donde se ha reescalado el parámetro de complejidad de forma que: $\bar{\alpha} = \alpha / RSS_0$, siendo $RSS_0 = \sum_{i=1}^{n} (y_i - \bar{y})^2$ la suma de los residuos al cuadrado del árbol sin divisiones. De esta forma, podríamos

interpretar el nuevo parámetro $\bar{\alpha}$ como una penalización en la proporción de variabilidad explicada, ya que sin más que dividir la ecuación previa por RSS_0, se obtiene la proporción de variabilidad residual:

$$\frac{RSS_{\bar{\alpha}}}{RSS_0} = \frac{RSS}{RSS_0} + \bar{\alpha}t$$

Ahora, sabemos que se cumple que:

$$\frac{RSS}{RSS_0} = 1 - R^2; \ \frac{RSS_{\bar{\alpha}}}{RSS_0} = 1 - R^2_{\bar{\alpha}}.$$

Por lo tanto, podemos reescribir la igualdad anterior como sigue.

$$R^2_{\bar{\alpha}} = R^2 - \bar{\alpha}t.$$

Es decir, lo que se puede deducir de esta ecuación es que por cada hoja extra que añadamos a nuestro árbol, se restará $\bar{\alpha}$ de puntuación a nuestro R^2. Dicho de otra forma, para que valga la pena dividir una rama y crear una hoja nueva en nuestro árbol de regresión, esa nueva división tiene que mejorar el R^2 más que la penalización $\bar{\alpha}$. Si la mejora es demasiado pequeña, el árbol considera que no merece la pena y se poda.

2.2.3. Posibles problemas

A lo largo de la Sección 2.2 hemos visto un método sencillo e interpretable, que nos proporciona resultados razonables de forma comprensible. En esta Subsección 2.2.3 veremos los posibles problemas que puede acarrear el uso del método en cuestión.

El primer problema llega con el uso de los predictores categóricos. Cuando vamos a dividir un predictor con q posibles valores sin una relación de orden, existen $2^{q-1} - 1$ particiones posibles de esos q valores en dos grupos, algo que se puede convertir en un problema computacionalmente inaccesible para valores grandes de q. Sin embargo, si nos encontramos ante la situación de predicción de una variable binaria, los cálculos se nos simplificarían, ya que podríamos ordenar las clases del predictor en función de la proporción que pertenece a la clase que da resultado "1". Después solamente quedaría dividir este predictor como si estuviese ordenado. Lamentablemente, el caso de respuesta binaria no nos atañe actualmente, ya que entraría dentro del campo de los problemas de clasificación.

Otro problema asociado a modelos de este tipo es la dificultad a la hora de modelar estructuras aditivas. Supongamos que estamos ante un caso de regresión en el que $Y = c_1 I(X_1 < t_1) + c_2 I(X_2 < t_2) + \varepsilon$, donde ε es ruido de media cero. Un árbol binario haría la primera división sobre X_1, cerca de t_1. En el próximo nivel, tendría que dividir ambos nodos sobre la variable X_2, en t_2, para poder capturar la estructura aditiva. Esto podría ocurrir si se cuenta con un número suficiente de datos, pero el modelo no recibe ningún aliciente específico para este fin. Si en cambio quisiéramos modelar el efecto aditivo de diez variables en lugar de dos, tendríamos un problema más complicado de resolver. Para que el árbol capture esta estructura, debería hacer lo que hemos explicado antes con dos variables, después dividir las cuatro ramas resultantes por X_3 y así sucesivamente. Además, viendo el árbol gigante que resultaría el analista tendría verdaderos problemas para darse cuenta de que el modelo simplemente nos estaría diciendo que sumemos las 10 variables. Es decir, el árbol estaría ocultando la simplicidad de la suma tras una estructura jerárquica compleja.

Por último, destacamos un último problema, que de hecho será parte de nuestra motivación para estudiar los métodos propuestos en la Sección 2.3 y la Sección 2.4, y es la inestabilidad de los árboles.

Un problema muy presente en los árboles de decisión, especialmente es los grandes, es la alta varianza que poseen. Un pequeño cambio en los datos puede resultar en divisiones totalmente distintas, haciendo que la interpretabilidad sea menos rigurosa. La razón principal a la que se atribuye esta inestabilidad es a la estructura jerárquica que posee el proceso, ya que pensemos que el efecto de un error en una de las primeras divisiones se propaga hacia abajo sobre todas las divisiones posteriores. Se puede tratar de mitigar esta inestabilidad hasta cierto punto cambiando el criterio de división a uno más estable, pero la inestabilidad inherente del método no se puede suprimir. Es por este motivo, que nos interesamos en métodos de reducción de la varianza, como pueden ser el bagging o el boosting.

Aunque los árboles de decisión son un método de calidad predictiva mediocre, son la base de otros métodos que resultan ser más competitivos y usados en la práctica. En la Sección 2.3 veremos un ejemplo de esto, ya que estudiaremos el método bagging, que como veremos a continuación, nos ayudará a reducir la varianza de nuestras estimaciones, proporcionando resultados más estables.

2.3. Bagging

En la Sección 2.2 tuvimos la oportunidad de revisar un método de predicción muy interesante, sencillo e interpretable, así como las características asociadas al mismo. Vimos el dilema entre el sesgo y la varianza que nos planteaba el tamaño del árbol, regulado por el parámetro de complejidad. Vimos que cuando dejamos crecer mucho un árbol, se reduce mucho el sesgo del modelo, pero aumenta la variabilidad y el riesgo de sobreajuste. Esto supone un problema, ya que hace que nos perdamos un predictor muy interesante, con gran interpretabilidad y calidad predictiva aceptable, por culpa del mencionado problema con la variabilidad.

En este contexto surgen los métodos ensemble o combinados, que se empiezan a popularizar en la década de 1990, y se basan en la combinación de las las predicciones de diversos modelos para llegar a una más precisa. Uno de los primeros en aparecer fue el bagging, que es precisamente un método general de reducción de la varianza, basado en la utilización del bootstrap como técnica de remuestreo, junto con un modelo de clasificación o regresión. Un método así aparentemente soluciona el problema que nos surgía con los árboles aleatorios, así como también los de otros predictores que se debilitan debido al exceso de varianza en sus predicciones. En esta introducción de la sección, veremos lo que es el bagging de forma general, para después centrarnos en un caso específico en el que los predictores son árboles de decisión, llamado random forest, al que le dedicaremos la Subsección 2.3.1. También, antes de continuar es importante destacar que las fuentes principales para la obtención de la información de esta sección han sido: Fernández-Casal et al. (2024), Hastie et al. (2009), Carmona (2023), Breiman (1996) y Breiman (2001).

Antes de entrar a revisar el método en cuestión, dada la estrecha relación que guarda este con el bootstrap, parece interesante explicar un poco más esta técnica de remuestreo. Como decíamos el boostrap es una técnica de simulación por remuestreo que se puede usar para estimar determinadas características de un estadístico de interés como puede ser su varianza. Supongamos que contamos con una muestra aleatoria simple $X = \{x_1, x_2, \ldots, x_n\}$. El proceso consiste en remuestrear n veces bajo la distribución empírica F_n con reemplazamiento la muestra original X, obteniendo lo que llamamos una remuestra boostrap. Repitiendo este proceso B veces se obtienen un total de B remuestras bootstrap, denotadas por $X^{*b} = \{x_1^*, x_2^*, \ldots, x_n^*\}, b = 1, \ldots, B$, que comparten los elementos de la original pero no el orden de las mismas. Debido al reemplazamiento, tampoco se asegura que todos los elementos de la original aparezcan en cada remuestra. Esto se encuentra ilustrado en la Figura 2.11. Después veremos de qué forma incorporan el boostrap estos modelos a sus procedimiendos.

Figura 2.11: Ilustración de la técnica de remuestreo boostrap.

La idea detrás del bagging es muy intuitiva. Se dispone de un modelo que produce predicciones aceptables, pero con mucha variabilidad, convirtiéndolo en un predictor mediocre. Si dispusiéramos de muchas muestras, podríamos entrenar el modelo tantas veces como muestras haya, y después cada modelo entrenado nos daría una predicción. Después podríamos realizar el promedio de todas las predicciones, llegando a un resultado más fiable, dado que se reduciría la variabilidad en las mismas. El obstáculo que nos encontramos ahora es que contamos únicamente con una muestra, pero aquí es donde entra el juego el boostrap. Lo que hacemos es generar cientos de remuestras bootstrap a partir de la muestra de entrenamiento y entrenar nuestro modelo con cada una de ellas.

Consideremos ahora el problema de regresión en el que queremos ajustar un modelo a nuestros datos de entrenamiento, que podemos denotar por: $(x, y) = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$, obteniendo la predicción $\hat{f}(x)$ para la entrada x. El bagging promedia esta predicción sobre una colección de muestras bootstrap, reduciendo la varianza como consecuencia. Es decir, para cada muestra boostrap, que podemos denotar por $(x, y)^{*b}, b = 1, \ldots, B$, ajustamos nuestro modelo, llegando a una predicción $\hat{f}^{*b}(x)$. La estimación de la predicción que nos proporcionaría el método bagging sería el promedio de todas estas predicciones boostrap, o lo que es lo mismo:

$$\hat{f}_{bag}(x) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}^{*b}(x). \tag{2.7}$$

Lo que podemos encontrar en la Ecuación (2.7) es la estimación de la predicción que nos da el método bagging. Si estamos trabajando con una variable respuesta numérica, usar la esperanza matemática sobre la distribución del remuestreo parece lo más adecuado, pero como hemos podido ver, en su lugar se utiliza la media muestral. A continuación justificaremos la validez del uso de la misma como estimación de la predicción.

Si denotamos por F_n la distribución empírica, que asigna la misma probabilidad $1/n$ en cada uno de nuestros puntos (x_i, y_i), la verdadera predicción teórica del bagging se definiría como $\mathbb{E}_{F_n} \hat{f}^*(x)$, donde la muestra remuestreada sería $(x, y)^* = \{(x_1^*, y_1^*), (x_2^*, y_2^*), \ldots, (x_n^*, y_n^*)\}$, con cada $(x_i^*, y_i^*) \sim F_n$. Dado que habitualmente no podemos calcular esta esperanza de forma exacta, la expresión que podemos encontrar en la Ecuación (2.7) es una estimación Monte Carlo de la verdadera predicción bagging, convergiendo a ella a medida que $B \longrightarrow \infty$. Demostremos este hecho formalmente.

Teorema 4. *La expresión* $\hat{f}_{bag}(x) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}^{*b}(x)$ *converge a* $\mathbb{E}_{F_n}[\hat{f}^*(x)]$*, donde* $(X,Y)^* = \{(x_1^*, y_1^*), (x_2^*, y_2^*), \ldots, (x_n^*, y_n^*)\}$*, y cada* $(x_i^*, y_i^*) \sim F_n$ *cuando* $B \longrightarrow \infty$*.*

Demostración. La demostración del Teorema 4 es relativamente directa mediante la aplicación de la Ley Fuerte de los Grandes Números. Fijando un punto x, podemos definir la predicción del modelo que hemos entrenado con la remuestra boostrap b como una variable aleatoria:

$$Z_b = \hat{f}^{*b}(x).$$

Dado que cada muestra boostrap $(X,Y)^*$ se obtiene con reemplazo y de forma independiente de la misma distribución empírica F_n (el conjunto de datos original), las predicciones generadas por los distintos modelos son variables aleatorias independientes e idénticamente distribuidas. Es decir, $Z_1, Z_2, \ldots, Z_B$ son i.i.d. Notemos ahora que la esperanza de cada una de estas variables aleatorias bajo F_n, es por definición la estimación teórica del bagging, que denotamos por μ:

$$\mathbb{E}[Z_b] = \mathbb{E}_{F_n}[\hat{f}^*(x)] = \mu.$$

Por otra parte, la estimación Monte Carlo de la Ecuación (2.7) es simplemente la media muestral de estas B variables aleatorias i.i.d.:

$$\hat{f}_{bag}(x) = \frac{1}{B}\sum_{b=1}^{B} Z_b.$$

Ahora, sabemos que la Ley Fuerte de los Grandes Números establece que la media muestral de una sucesión de variables aleatorias i.i.d converge casi seguro a su valor esperado teórico a medida que el tamaño muestral, que en este caso es nuestro número de réplicas bootstrap, B, tiende a infinito. Formalmente esto sería:

$$\mathbb{P}\left(\lim_{B\to\infty} \frac{1}{B}\sum_{b=1}^{B} Z_b = \mu\right) = 1.$$

Ahora, sustituyendo los términos necesarios, podemos concluir la demostración. Con probabilidad uno,

$$\lim_{B\to\infty} \hat{f}_{bag}(x) = \lim_{B\to\infty} \frac{1}{B}\sum_{b=1}^{B} \hat{f}^{*b}(x) = \mathbb{E}_{F_n}[\hat{f}^*(x)].$$

□

Después de comprobar este resultado, podemos encontrar un ejemplo del funcionamiento del método ilustrado en la Figura 2.12.

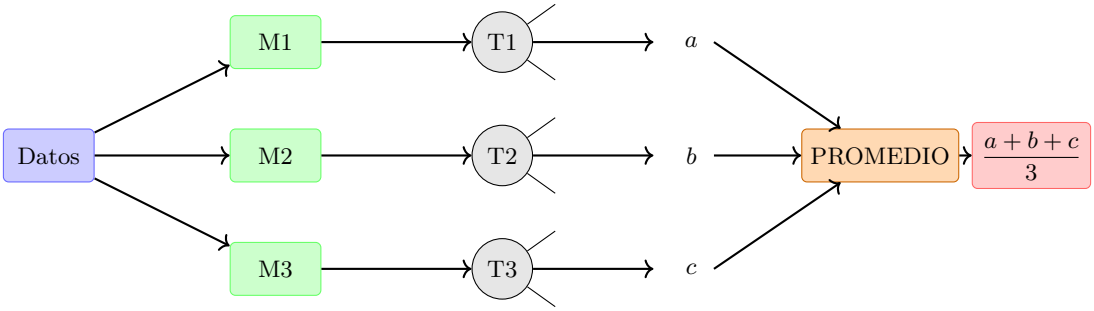


Figura 2.12: Diagrama que muestra el funcionamiento del método bagging empleando la técnica de remuestreo boostrap. Las cajas verdes representan las remuestras boostrap obtenidas a partir de la muestra de entrenamiento, representada por la caja azul. Los cículos representan los modelos que se van a combinar y las letras a, b y c sus respectivas predicciones.

Como se puede comprobar en la Sección 4 de Breiman (1996), este método, en efecto, funciona muy bien en el campo de las predicciones numéricas. Una ventaja adicional que tiene el bagging es que nos permite estimar el error de predicción de forma directa, sin necesidad de recurrir a una muestra de test u otras técnicas de remuestreo. Esto se debe a algo que ya ilustramos en la Figura 2.11, pero que no comentamos en su momento, la muestra OOB (por sus siglas en inglés, *out-of-the-bag*). Tal y como mencionamos anteriormente, las remuestras boostrap no tienen por qué incluir necesariamente todas las observaciones de la muestra de entrenamiento original. Por este motivo, fijémonos que hay unas observaciones de la muestra original que cada modelo de regresión ajustado no ha llegado a ver en ningún momento, y estas son precisamente las que constituyen la muestra OOB de cada uno de los modelos. Cada uno posee la suya, con unas observaciones determinadas. Se sabe que una muestra boostrap, en promedio, solamente utiliza dos tercios de los datos, o para ser más precisos, $1-(1-\frac{1}{n})^n$, que a medida que aumenta la muestra de entrenamiento se convierte en $1-\frac{1}{e}$. De esta forma, dado que cada observación no ha sido vista, aproximadamente, por un tercio de los modelos, a cuyas muestras OOB pertenece, se pueden emplear estos modelos para generar predicciones sobre la misma. De esta forma obtenemos una medida de error para esta observación, que se puede hacer global al repetir el proceso para las demás observaciones.

Ahora nos surge una nueva decisión, y es la de cuantas remuestras boostrap se han de generar, o lo que es lo mismo, cuantas predicciones deseamos promediar. A pesar de que Breiman (1996) mencionó que 25 réplicas boostrap le funcionaron bien en el ámbito de la regresión, también afirma que cuando se trabaja con la metodología CART (Classification and Regression Trees) de la que se habló en la Sección 2.2, la respuesta a esta cuestión no es demasiado relevante, ya que los tiempos de ejecución son insignificantes, incluso para un gran número de réplicas. Por otra parte, Fernández-Casal et al. (2024) propone una alternativa que es muy interesante en la práctica. Esta se basa en seleccionar B mediante la representación del error OOB, visualizando en qué punto se produce su convergencia. Normalmente cuando se aprecia la convergencia, aumentar el número de modelos no hará que el modelo mejore significativamente, pero como novedad, no aumentará el riesgo de sobreajuste a los datos de entrenamiento, dado que las remuestras generadas son independientes una de otra. Cabe destacar que esto no significa que el bagging, o el random forest que veremos después no puedan sobreajustar. Esto sí podría ocurrir bajo ciertas condiciones, como por ejemplo si los árboles que se emplean sean excesivamente profundos.

Se puede comprobar que un método con poca variabilidad tiene poco interés para la aplicación del bagging. Por este motivo encajan tan bien en este contexto los árboles de decisión sin podar. Tanto es así, que existe una variante del bagging diseñada específicamente para árboles de decisión, que además

incorpora alguna ventaja adicional. Estamos hablando del random forest, y lo veremos a continuación en la Subsección 2.3.1.

2.3.1. Random Forest

Como comentábamos previamente, el random forest es una variante del bagging en la que se toman árboles de decisión como modelos para la aplicación del mismo. La motivación para la creación de este modelo no surge únicamente de la idoneidad que presentan los árboles de decisión sin podar para combinar sus predicciones, si no de algunos inconvenientes que surgían al incorporarlos al bagging al uso.

En la demostración del Teorema 4 se asumió que cada variable aleatoria $Z_b = \hat{f}^{*b}(x)$ es una realización independiente del proceso bootstrap. Argumentamos que es idénticamente distribuida porque cada muestra se genera mediante un remuestreo con reemplazo sobre F y que es independiente en el sentido de que el proceso de remuestreo de un árbol no afectará al de otro. Esto fue suficiente para concluir la demostración, pero la realidad es que aunque los procesos de remuestreo son independientes entre sí, las variables Z_b están correlacionadas, es decir, $\rho \geq 0$, donde ρ representa la correlación positiva a pares, ya que los árboles comparten en promedio dos tercios de los datos con los que entrenan.

Esto provoca un problema y es que esta aleatoriedad no es suficiente en determinadas situaciones, dando lugar a árboles relativamente parecidos sobre todo en la parte más cercana a la raíz, aunque posteriormente se vayan diferenciando. Esta característica se conoce como correlación entre árboles y tiene lugar cuando un árbol es muy adeacuado para describir la relación entre los predictores y la respuesta, o cuando hay algún predictor muy fuerte, de forma que siempre se encuentra en el corte superior. Esta correlación entre árboles, se traduce en correlación entre las predicciones que estas generan, invalidando las ventajas que el bagging nos aportaba, ya que promediar variables correladas produce una reducción de varianza mucho menor que promediar variables incorreladas.

Formalmente tenemos que si promediamos B variables aleatorias i.i.d, cada una con varianza σ^2, obtenemos una varianza de σ^2/B. Si las variables están simplemente idénticamente distribuidas, con correlación a pares ρ, la varianza del promedio sería:

$$\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2.$$

A medida que B aumenta, el segundo término desaparece, pero el primero prevalece. Esta es la explicación de por qué hacer bagging sobre árboles correlados limita los beneficios del método. Antes de continuar veamos que, en efecto, esa es la varianza resultante de promediar B variables correlacionadas.

Proposición 1. *La varianza de la media de B variables correlacionadas es $\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$.*

Demostración. Sean $X_1, X_2, \ldots, X_B$ variables con la misma varianza $\text{var}(X_i) = \sigma^2$ y una correlación positiva par a par $Corr(X_i, X_j) = \rho$, para $i \neq j$. Recordemos ahora que $Cov(X_i, X_j) = \rho\sigma_i\sigma_j = \rho\sigma^2$. Ahora, tenemos que la varianza de una suma de variables es la suma de sus varianzas, sumando todas sus covarianzas. Formalmente:

$$\text{var}\left(\sum_{i=1}^{B} X_i\right) = \sum_{i=1}^{B} \text{var}\left(X_i\right) + \sum_{i \neq j} Cov\left(X_i, X_j\right).$$

Dado que tenemos B términos asociados a la varianza y $B(B-1)$ términos de covarianza, reescribimos:

$$\text{var}\left(\sum_{i=1}^{B} X_i\right) = B\sigma^2 + (B^2 - B)\rho\sigma^2.$$

Ahora, dado que lo que buscamos es la varianza del promedio, que sería $\bar{X} = \frac{1}{B}\sum_{i=1}^{B} X_i$, debemos aplicar la propiedad de la varianza que dice que: $\text{var}(aX) = a^2\text{var}(X)$. Así:

$$\text{var}\left(\bar{X}\right) = \frac{1}{B^2}\left(B\sigma^2 + (B^2 - B)\rho\sigma^2\right) = \frac{\sigma^2}{B} + \rho\sigma^2 - \frac{\rho\sigma^2}{B}.$$

Por último, reagrupando términos, podemos dar por concluida la demostración.

$$\text{var}\left(\bar{X}\right) = \rho\sigma^2 + \frac{\sigma^2 - \rho\sigma^2}{B} = \rho\sigma^2 + \frac{1-\rho}{B}\sigma^2.$$

□

Con esta problemática surgen nuevas modificaciones para tratar de paliarla basados en agregar aleatoriedad y reducir correlación. En Breiman (2001) el autor propone los bosques aleatorios. La idea que aporta es la de que en la construcción de cada uno de los árboles que constituirán el modelo final, se van haciendo cortes binarios, y en cada uno de los cortes, se seleccionarán al azar m de los p predictores con los que se contaba de inicio. Es decir, en cada uno de los cortes no van a estar disponibles todas las variables predictoras, introduciendo de esta forma una componente de aleatoriedad a mayores y una barrera adicional para el problema de la correlación entre árboles. Esta característica, que se encuentra ilustrada en la Figura 2.13, repercute a su vez en el número de árboles necesarios, haciendo que este aumente, dado que ahora la elección de los predictores no depende por completo del árbol en cuestión.

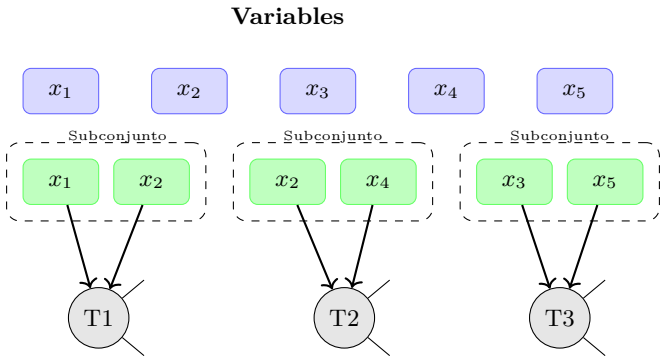


Figura 2.13: Diagrama que ilustra la mejora propuesta en random forest con respecto al bagging al uso con respecto a la selección de predictores.

Es habitual seleccionar $m = p/3$. Denotemos por Θ_b al "vector de parámetros" que define al árbol b, incluyendo las variables de corte del mismo, los puntos de corte en cada nodo y los valores de los nodos terminales. Así, $T(x;\Theta_b)$ representa el árbol b, donde x es el dato de entrada. Después de hacer crecer B árboles tales $\{T(x;\Theta_b)\}_1^B$, el predictor random forest para el caso de regresión sería:

$$\hat{f}_{\text{rf}}^B(x) = \frac{1}{B}\sum_{b=1}^{B} T(x;\Theta_b).$$

La intuición nos dice que a medida que disminuyamos el valor de m, disminuirá también la correlación entre cualquier par de árboles, y como consecuencia se reducirá también la varianza vista en la

Proposición 1. Ahora, dado que este será además uno de los modelos que ajustemos en el Capítulo 4, merece la pena explicitar los pasos que sigue el algoritmo, para que el lector lo tenga en cuenta cuando se llegue a usar. Ver Algoritmo 1.

Algoritmo 1 Algoritmo de Random Forest para regresión

1.
Para $b = 1, 2, \ldots, B$ **hacer**

a) Extraer una muestra bootstrap Z^* de tamaño n de los datos de entrenamiento.

b) Hacer crecer un árbol de regresión $T(x; \Theta_b)$ a los datos Z^*, repitiendo recursivamente los siguientes pasos para cada nodo terminal hasta alcanzar el tamaño mínimo de nodo n_{min}:

i. Seleccionar m variables al azar de entre las p disponibles.
ii. Elegir la mejor variable y punto de corte entre esas m.
iii. Dividir el nodo en dos nodos hijos.

2. Obtenemos así el conjunto de árboles $\{T(x; \Theta_b)\}_1^B$.

3. Para un nuevo dato x, la predicción es el promedio:

$$\hat{f}_{\text{rf}}^B(x) = \frac{1}{B} \sum_{b=1}^{B} T(x; \Theta_b)(x)$$

Además, el random forest cuenta con algunas ventajas más que lo convierten en una opción realmente atractiva para su uso en la práctica, y es que da lugar a modelos más eficientes computacionalmente que el bagging. Esto, a pesar de parecer contraintuitivo dado que antes hemos dicho que se necesita un número mayor de árboles, es así ya que la construcción de cada árbol es mucho más rápida, al evaluarse solamente en un número reducido de predictores en cada corte.

Por otra parte, no podemos perder de vista que si bien es cierto que el bagging o el random forest ayudan a reducir la varianza en las predicciones de los árboles de decisión, haciendo que estas sean más consistentes, existe un precio a pagar en términos de interpretabilidad. Recodemos que los árboles de decisión eran modelos muy interpretables, pero lamentablemente esta característica no se traslada a los estudiados en esta sección.

2.4. Boosting

En esta Sección 2.4 veremos un nuevo método de reducción de la varianza, pero cuya esencia es diferente a la del bagging (y, por supuesto, a la del random forest) que pudimos revisar en la Sección 2.3, llamado boosting. Como se vio en la misma, el bagging combinaba predictores relativamente precisos, debilitados por la inestabilidad asociada a los mismos. El enfoque del boosting es diferente, ya que se basa en combinar múltiples predictores débiles para, impulsados, dar lugar a un mejor predictor. Antes de continuar y profundizar más en el concepto de boosting, cabe destacar que para la realización de esta sección se han tomado como referencias principales las siguientes: Amat Rodrigo y Escobar Ortiz (2021), IBM (2026), Fernández-Casal et al. (2024), James et al. (2023), Chen y Guestrin (2016) y Belcic y Stryker (2026).

El boosting, como decíamos, es una técnica o metodolodía de aprendizaje lento en la que se combinan muchos modelos obtenidos mediante un método con poca capacidad predictiva para, impulsados, dar lugar a un mejor predictor. Los árboles de decisión pequeños (de poca profundidad) cumplen las características requeridas a la perfección, por lo que son una opción muy interesante y utilizada cuando se habla de esta técnica y será, en particular, en la que se centrará esta sección.

En el párrafo anterior mencionamos tanto el hecho de que esta metodología es de aprendizaje lento, como que está basada en impulsar predictores débiles. Esta tarea es llevada a cabo haciendo crecer árboles de forma secuencial, de forma que cada árbol nuevo se genera utilizando información de sus predecesores. A diferencia de otras metodologías también comúnmente utilizadas, como puede ser el bagging, el boosting no emplea técnicas de remuestreo. Cada árbol se ajusta en base a una versión modificada del conjunto de datos de partida.

Esta representa una de las ideas de aprendizaje más potentes de los últimos 20 años. Originalmente se diseñó para problemas de clasificación, destacando algoritmos como Adaboost (Adaptative Boosting), pero ahora, afortunadamente, abarca también los problemas de regresión. Lo hizo de la mano del método llamado Gradient Boosting Machine, perteneciente a la familia de los métodos iterativos de descenso de gradiente.

En estos métodos se desea minimizar una función de pérdida mediante el método de los gradientes, algo que no da lugar a mayores complicaciones si la función es diferenciable. Antes de continuar, conviene aclarar brevemente lo que es el método de los gradientes. Como sabemos, el gradiente es un vector que indica la dirección de máximo crecimiento, por lo que si deseamos minimizar, en este caso una función de pérdida que medirá el error cometido, debemos movernos en la dirección opuesta al gradiente. Así, este método funciona de forma iterativa. Primero se calcula el gradiente en un punto, se da un paso en la dirección opuesta y se repite de forma sucesiva.

Una ventaja de esta aproximación es que se pueden tomar diferentes funciones de pérdida, pero nosotros tomaremos la suma de los residuos al cuadrado como la pérdida de utilizar $f(x)$ para predecir la respuesta en el entrenamiento. Es decir:

$$L(f) = \sum_{i=1}^{n} L(y_i, f(x_i)) = \frac{1}{2}\sum_{i=1}^{n}(y_i - f(x_i))^2, \tag{2.8}$$

Así, multiplicando el sumatorio por $1/2$, los gradientes serían precisamente los residuos. Comprobémoslo:

$$-\frac{\partial L(f)}{\partial f(x_i)} = y_i - f(x_i) = r_i.$$

Esto es muy interesante, ya que como veremos a continuación cuando explicitemos el algoritmo para un problema de regresión, empleando como hemos dicho árboles de decisión, el proceso iterativo no ataca los datos de forma directa, sino los residuos, es decir, los gradientes, que van resultando de los distintos ajustes. Esto es, para cada una de las B iteraciones, se ajustará un árbol nuevo a los datos de entrenamiento, cuyo objetivo será predecir los residuos que resultaron del árbol de la iteración anterior. De esta forma se mejora la predicción de forma lenta y gradual, ajustando un nuevo árbol en cada iteración con el objetivo de mejorar lo que falló su predecesor, para después dar lugar al modelo final que será la combinación de todos ellos. El Algoritmo 2 explicita el proceso.

Algoritmo 2 Boosting para árboles de regresión

1. Inicializar $\hat{f}(x) = 0$ y los residuos $r_i = y_i$ para todo i en el conjunto de entrenamiento.

2.
Para $b = 1, 2, \ldots, B$ **hacer**

a) Ajustar un árbol $\hat{f}^b$ con profundidad máxima d a los datos de entrenamiento. Este no buscará la predicción del valor original, sino que busca la predicción de los residuos actuales, r. Es decir, busca mejorar lo que los árboles anteriores no pudieron resolver.

b) Actualizar $\hat{f}$ añadiendo una versión reducida:

$$\hat{f}(x) \leftarrow \hat{f}(x) + \eta \hat{f}^b(x),$$

con η denotando la tasa de aprendizaje.

c) Actualizar los residuos: $r_i \leftarrow r_i - \eta \hat{f}^b(x_i)$.

3. **Salida:** El modelo de boosting final:

$$\hat{f}(x) = \sum_{b=1}^{B} \eta \hat{f}^b(x)$$

El proceso que se muestra en el Algoritmo 2 nos permite destacar varios hechos. El primero es que, como decíamos, no se realiza un remuestreo sobre los datos de entrenamiento, sino que se toma la misma muestra de entrenamiento y se van mejorando las predicciones en base a los residuos de cada ajuste. Podríamos ilustrar el proceso gráficamente para que sea algo más visual. Esto se encuentra plasmado en la Figura 2.14.

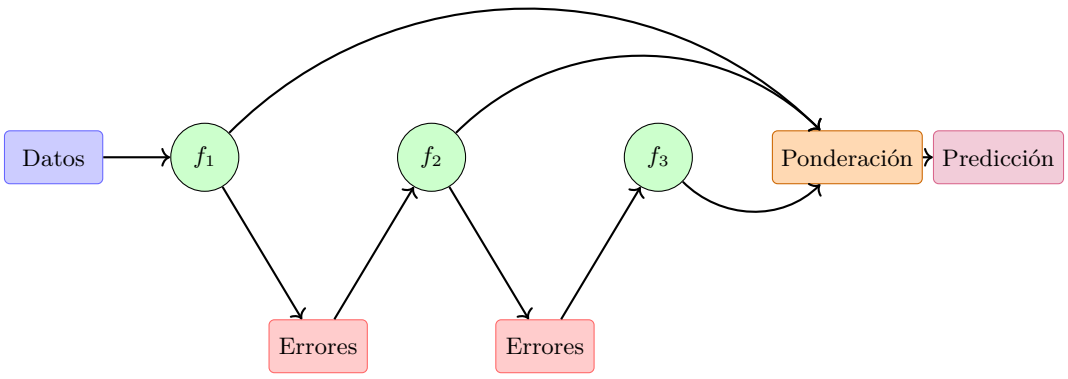


Figura 2.14: Diagrama que ilustra el funcionamiento del algoritmo de boosting.

Ahora que tenemos una idea visual de cómo funciona el algoritmo, podemos volver al Algoritmo 2 y fijarnos en más detalles del mismo. Lo primero que notamos es que el valor del hiperparámetro

d determinará el número de cortes, y por tanto el número de nodos terminales $(d+1)$. Respecto a este punto, cuando llegue el momento de ajustar los modelos a los datos, deberemos tener en cuenta que la profundidad de los árboles que se combinan en random forest debe ser mayor que la de los que se combinan en el algoritmo de boosting, ya que la motivación del primer método implica tener predicciones aceptables, pero inestables, y la del segundo es potenciar malos predictores.

Fijémonos ahora en el parámetro η, comúnmente llamado tasa de aprendizaje o *learning rate*. Este es un hiperparámetro que rige cuánto ajusta sus parámetros un modelo de machine learning en cada paso de su algoritmo de optimización. Es decir, es capaz de determinar si un modelo ofrece un rendimiento óptimo o no aprende durante el proceso de entrenamiento. El objetivo de un algoritmo de optimización es minimizar la mencionada función de pérdida que mide la diferencia entre las predicciones del modelo y la realidad. Cada vez que el modelo ejecuta su algoritmo de optimización, actualiza sus pesos en función del resultado y es precisamente este parámetro el que determina el tamaño de los cambios. Ayuda a garantizar que un modelo aprenda lo suficiente del entrenamiento para realizar ajustes significativos en sus parámetros sin corregir en exceso.

Se puede intuir la gran importancia que tiene la tasa de aprendizaje en el ajuste del modelo. Una tasa de aprendizaje alta provoca que el algoritmo sea víctima de un exceso de corrección en sus errores, llevando a un posible sobreajuste, y es que ahora, a diferencia de lo que ocurría con el bagging o el random forest, añadir más árboles sí que aumenta el riesgo de sobreajuste, ya que los árboles que se combinan no son independientes entre sí. Por otra parte, una tasa de aprendizaje baja no permite que el modelo aprenda lo suficiente en cada paso, actualizando sus parámetros con excesiva lentitud y tardando demasiado en alcanzar la convergencia. Cuando el valor de este parámetro es pequeño, se deberá suplir con un valor mayor de B, que representa el número de iteraciones o árboles, para que el algoritmo funcione de forma satisfactoria.

Por último, merece la pena destacar una característica más que se puede deducir del Algoritmo 2. Como hemos dicho a lo largo de la sección, en este algoritmo no se ataca directamente a los datos, sino al gradiente, es decir, a los residuos. Fijémonos en el paso 2.c. El residuo es por definición la parte de y_i que el modelo aún no ha sido capaz de explicar, es decir, el error de predicción actual. Si el modelo predice muy bien una observación, su residuo será cercano a cero, mientras que si lo hace mal, el residuo será grande (ya sea muy positivo o muy negativo). Por lo tanto, cuando se vuelve al paso 2.a., para que el árbol $\hat{f}^b$ logre reducir la pérdida total de forma más eficiente, está obligado a reducir primero los residuos más grandes, que son los que más pesan en la suma de cuadrados. De esta forma, implícitamente se otorga más peso a las observaciones en las que más se falla, y es en esta idea en la que se sustenta el boosting, la de potenciar los clasificadores mejorando las prediciones en las que más se ha fallado. Es decir, esta metodología puede verse como una ponderación iterativa de las observaciones, en la que se asigna un peso mayor a aquellas que dieron lugar a más problemas. Se puede encontrar esta idea ilustrada en la Figura 2.15.

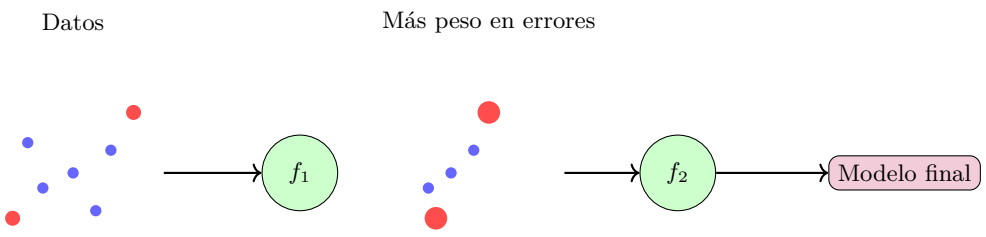


Figura 2.15: Diagrama que ilustra cómo los mayores errores reciben más atención.

Después de su introducción para problemas de regresión llegaron más modificaciones que potenciaron

su uso en este campo. Destacamos por ejemplo la variante conocida como Stochastic Gradient Boosting, cuya única modificación con respecto al Algoritmo 2 llega en la primera línea dentro del bucle, en la que al hacer el ajuste del nuevo árbol a los datos de entrenamiento buscando la predicción de los residuos pertinentes, no se considera toda la muestra de entrenamiento, sino que se selecciona al azar un subconjunto de la misma. Más mejoras fueron llegando, que hacían que el boosting fuese cada vez más competitivo en la práctica, hasta llegar al método que se ha seleccionado para el ajuste de los datos en este documento, el Extreme Gradient Boosting, o XGBoost.

2.4.1. XGBoost

Como comentamos anteriormente, el algoritmo de boosting seleccionado para su implementación práctica es el XGBoost. Este es un algoritmo que ha obtenido muy buenos resultados en la práctica, como podemos ver en Makridakis et al. (2022), y por ello le dedicaremos esta subsección.

A lo largo de la sección hemos podido ver el funcionamiento básico del boosting aplicado a problemas de regresión tomando árboles de decisión, pero lo interesante ahora es entender qué mejoras aporta XGBoost sobre esta base, que la conviertan en una opción tan interesante, y que hace que ofrezca un rendimiento tan competitivo con respecto a sus homólogos, a pesar de que, como veremos, su complejidad sea mayor.

Además de las modificaciones previamente mencionadas, como la aleatoriedad aportada por el Stochastic Gradient Boosting, el Extreme Gradient Boosting introduce una función objetivo compuesta esencialmente por dos partes. Por un lado tenemos una función de pérdida que mide la bondad del ajuste a los datos como la que pudimos ver en la Ecuación (2.8), $L(f)$, pero por otro lado se añade un término de regularización, denotado por $\Omega(f)$, que penaliza la complejidad del árbol, tratando así de paliar el sobreajuste. La expresión explícita de este último sería:

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda ||w||^2.$$

Fijémonos en la expresión asociada a $\Omega(f)$. Vemos que el parámetro γ, que junto a λ, representan los parámetros de regularización, acompaña al número total de nodos terminales que posee el árbol, denotado por T. De esta forma, estamos penalizando los árboles muy profundos. El vector w, por su parte, contiene los pesos de los nodos terminales y cumple que $w_j \in \mathbb{R}$. Al multiplicarlo por el parámetro λ penalizamos las hojas con pesos extremos, por lo que, de no existir, un solo nodo terminal podría tener asociado un peso enorme para tratar de ajustar un valor atípico, provocando un sobreajuste.

Sin embargo, donde XGBoost marcó la verdadera diferencia fue en el proceso de optimización. Mientras que el Stochastic Gradient Boosting al uso realiza un descenso de gradiente de primer orden, XGBoost incorpora un desarrollo de Taylor de segundo orden, logrando así una convergencia al óptimo más eficiente.

Ahora deseamos formalizar este proceso, por lo que denotemos por $l(f)^{(b)}$ la función objetivo en la iteración b. Recordemos que esta incorpora tanto la función de pérdida vista en la Ecuación (2.8), como el término de regularización, asociado a la iteración considerada. Como veremos a continuación, en la iteración b supondremos que las predicciones asociadas a la iteración anterior son conocidas, y las denotaremos por $\hat{y}_i^{(b-1)}$. En consecuencia, tal y como adelantábamos previamente, la optimización en la iteración actual se centra en hallar el nuevo árbol, f^b, empleando el gradiente, g_i, y la hessiana, h_i, que representan la primera y segunda derivada de la función de pérdida de la Ecuación (2.8) evaluadas en las predicciones de la iteración anterior, $\hat{y}_i^{(b-1)}$. La necesidad de plantear una aproximación de este estilo viene de la dificultad derivada de optimizar de forma directa una función objetivo compleja como la que se acaba de plantear. En concreto, en la Proposición 2 podremos ver que la función objetivo

general puede aproximarse mediante una forma cuadrática que nos permitirá obtener la estructura del árbol de una forma más eficiente.

Proposición 2. *En cada iteración b, el algoritmo XGBoost selecciona el árbol f^b que minimiza la aproximación de Taylor de segundo orden de la función objetivo global, definida como:*

$$l(f)^{(b)} \approx \sum_{i=1}^{n} [g_i f^b(x_i) + \frac{1}{2} h_i (f^b)^2(x_i)] + \Omega(f^b),$$

donde g_i y h_i son el gradiente y la hessiana de la función de pérdida vista en la Ecuación (2.8), respectivamente.

Demostración. Hasta el momento hemos trabajado con algoritmos que emplean solamente la aproximación de Taylor de primer orden (el gradiente) junto con la función de pérdida que pudimos ver anteriormente. Consideremos ahora una nueva función objetivo que añade a la misma un término de penalización por complejidad:

$$l(f) = \sum_{i=1}^{n} L(y_i, f(x_i)) + \sum_{b=1}^{B} \Omega(f^b),$$

donde, como hemos visto antes:

$$\Omega(f) = \gamma T + \frac{1}{2} \lambda ||w||^2.$$

Como hemos podido ver a lo largo de la sección, el boosting se entrena de forma aditiva, y el XGBoost no será distinto. Por este motivo, podemos definir la predicción de la i-ésima observación en la iteración b como sigue.

$$\hat{y}_i^{(b)} = \hat{y}_i^{(b-1)} + f^b(x_i).$$

Sustituyendo esto en la función objetivo general $l(f)$, obtenemos la expresión relativa al paso b.

$$l(f)^{(b)} = \sum_{i=1}^{n} L(y_i, \hat{y}_i^{(b-1)} + f^b(x_i)) + \Omega(f^b). \tag{2.9}$$

Notemos que aunque la función objetivo global regulariza todos los árboles ($\Omega(f)$), en el proceso de optimización asociado a la iteración b solamente nos interesa el término de regularización asociado al árbol correspondiente a la misma, ya que los términos asociados a árboles (iteraciones) previos se mantienen constantes. Ahora podemos realizar el desarrollo de Taylor de segundo orden sobre la función de pérdida en la expresión anterior. Para ello, considerando nuestra función L, tomando como punto base la predicción de la iteración anterior, $\hat{y}^{(b-1)}$, y $f^b(x_i)$ como incremento, llegamos a:

$$L(y_i, \hat{y}_i^{(b-1)} + f^b(x_i)) \approx L(y_i, \hat{y}_i^{(b-1)}) + \left[\frac{\partial L}{\partial \hat{y}_i^{(b-1)}} \right] f^b(x_i) + \frac{1}{2} \left[\frac{\partial^2 L}{\partial (\hat{y}_i^{(b-1)})^2} \right] (f^b(x_i))^2.$$

Ahora sustituyendo las derivadas como g_i (el gradiente) y h_i (la hessiana), la expresión nos queda:

$$L(y_i, \hat{y}_i^{(b-1)} + f^b(x_i)) \approx L(y_i, \hat{y}_i^{(b-1)}) + g_i f^b(x_i) + \frac{1}{2} h_i (f^b(x_i))^2.$$

Por último, sustituimos esta expresión dentro de la función objetivo original, $l(f)^{(b)}$:

$$l(f)^{(b)} \approx \sum_{i=1}^{n} \left[L(y_i, \hat{y}_i^{(b-1)}) + g_i f^b(x_i) + \frac{1}{2} h_i (f^b(x_i))^2 \right] + \Omega(f^b).$$

Observemos ahora un hecho clave y es que nuestro objetivo en la iteración b es encontrar el mejor árbol f^b. Por lo tanto, cualquier término que no dependa del mismo se podrá asumir constante. Fijémonos por ejemplo en $L(y_i, \hat{y}_i^{(b-1)})$, que representa el término de error del modelo hasta el paso anterior. Resulta evidente que ya no podemos cambiarlo, es un número fijo, y dado que al minimizar una función las constantes no influyen, lo eliminamos para simplificar los cálculos:

$$l(f)^{(b)} \approx \sum_{i=1}^{n} [g_i f^b(x_i) + \frac{1}{2} h_i (f^b)^2(x_i)] + \Omega(f^b),$$

donde $g_i = \partial_{\hat{y}^{(b-1)}} L(y_i, \hat{y}^{(b-1)})$ representa el gradiente y $h_i = \partial^2_{\hat{y}^{(b-1)}} L(y_i, \hat{y}^{(b-1)})$ la hessiana, tal y como queríamos demostrar. □

Cabe destacar que se ha omitido la presencia de η en el desarrollo de Taylor siguiendo la lógica expuesta en Chen y Guestrin (2016), donde la fase de optimización y de actualización se tratan como pasos independientes, permitiendo así simplificar el desarrollo algebraico expuesto. Notemos que la optimización de Taylor ya nos da la dirección de máximo descenso, mientras que η sería un hiperparámetro de control.

Observemos ahora un hecho interesante, y es que la expresión en la Ecuación (2.9) lo que hace es evidenciar que, al igual que ocurría con el boosting visto anteriormente, esta nueva variante también se centra en sumar la función f_b que más mejora nuestro modelo, de forma avariciosa. Ahora, la Figura 2.16 nos ilustra la relación que guarda el árbol con la función objetivo regularizada en cada iteración del método.

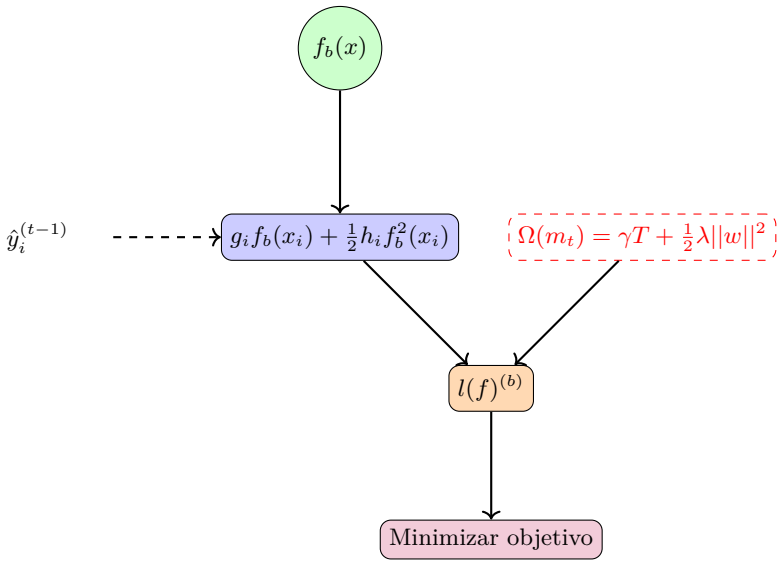


Figura 2.16: Diagrama que ilustra la correspondencia entre el árbol f_b y la función objetivo regularizada en XGBoost usando la aproximación de Taylor de segundo orden.

Capítulo 3

Redes Neuronales

En el Capítulo 2 se revisaron modelos y técnicas de machine learning que ofrecían diversas ventajas sobre los modelos clásicos. Cada uno con características propias, se convierten en una opción muy interesante para el investigador. En el Capítulo 3 tomaremos otro camino hacia metodologías que si bien comparten cualidades y objetivos con los anteriores, son diferentes en muchos aspectos. En particular en esta sección se estudiará la piedra angular del deep learning, las redes neuronales. Primero revisaremos las más simples, e iremos motivando la necesidad de ir recurriendo a opciones más complejas para llegar a los objetivos deseados. Por otra parte, tal y como hemos hecho a lo largo de todo el documento, indicamos que las referencias que se han consultado para la obtención de la información que se puede encontrar en este capítulo han sido: Fernández-Casal et al. (2024), James et al. (2023), Joseph (2022), Kulkarni et al. (2025), GeeksforGeeks (2026), ultralytics (2026a), Goodfellow et al. (2016), GeeksforGeeks (2025b), Jain (2024), ZeroMathAI (2026), ultralytics (2026b), Srivastava et al. (2014) y Huet (2024).

Las redes neuronales son una metodología adecuada para abordar problemas que posean una estructura subyacente compleja, capaces de captar patrones internos difíciles de percibir para el ojo humano. Con un nivel de especialización regulable a través del número de capas de procesamiento y unidades dentro de las mismas, fijadas de antemano por el investigador, resultan ser modelos hiperparametrizados y de baja interpretabilidad, cuyo funcionamiento se ve potenciado cuanto mayor es el conjunto de datos disponible. En la Sección 3.1 se estudiará la arquitectura más sencilla y más conocida de las redes neuronales, la Feed-forward network. Después, motivaremos la necesidad de la disponibilidad de memoria en las redes neuronales que utilicemos para el modelado de series de tiempo, por lo que en la Sección 3.2 se encontrará el estudio de las redes neuronales recurrentes, que nos ayudan a suplir esta carencia de las primeras. En la Sección 3.3 se revisará la arquitectura LSTM, que es una versión más moderna y refinada de las redes neuronales recurrentes clásicas, para finalmente concluir el capítulo con la Sección 3.4, en la que se estudiará la arquitectura GRU, que propone ciertas mejoras con respecto a la de la sección previa.

3.1. Feed-forward network

Comencemos hablando de la red neuronal más básica, la conocida como *feed-forward network* o FFN para abreviar. La idea que reside tras la FFN es la de tomar un vector de dimensión fija como *input* y procesarlo hacia delante, a través de un número determinado de capas ocultas que nos terminan proporcionando el resultado o salida deseada, a través del nodo final. A continuación mostramos un esquema de esta arquitectura.

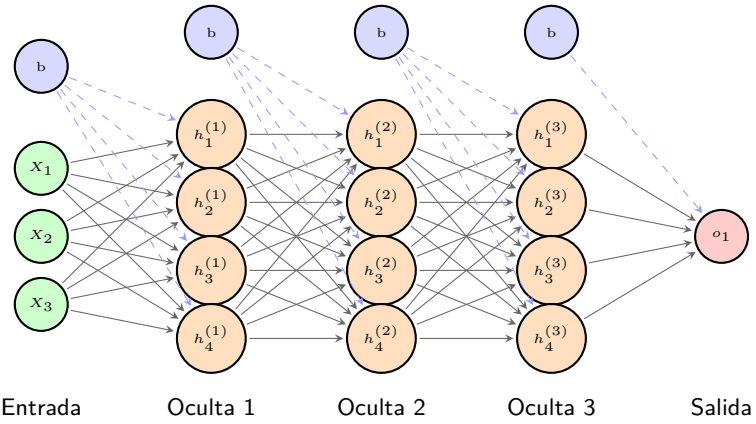


Figura 3.1: Arquitectura de una red neuronal *feed-forward* con tres capas ocultas.

Como se observa en la Figura 3.1, la arquitectura de las FFN destaca por su flujo unidireccional, en el que todos los nodos de la capa previa están conectados a todos los nodos de la capa posterior. A la primera capa se le conoce como capa de entrada o *input layer* y a la última como capa de salida o *output layer* y es a través de la cual nos llegará el resultado buscado. Todas las capas intermedias se conocen como capas ocultas o *hidden layers*, y son en las que se va realizando un procesado automático de los datos centrándose de forma sucesiva en los aspectos relevantes de los mismos. Una de sus debilidades es que conforme aumentan las capas se hace más difícil la interpretación del modelo, hasta convertirse en una auténtica caja negra. Por último, tenemos los nodos azules con una b dibujada dentro. Estos son los nodos de sesgo, cuya función sería la de otorgar una flexibilidad a la red. Podemos apreciar que en la Figura 3.1 se ha optado por ejemplificar esta arquitectura con tres nodos de entrada (*inputs*), tres capas ocultas con cuatro nodos cada una, y un único nodo de salida. A continuación profundizaremos en estas explicaciones, centrándonos la formalización matemática de las mismas.

El caso más sencillo de la FFN se da cuando esta tiene una única capa oculta, con un número de nodos que supondremos igual a M. Cada unidad oculta, h_m, que es como se suele llamar a los nodos de la capa oculta, es una variable resultado de combinar linealmente las p variables predictoras con unos parámetros w_{jm} (los parámetros w_{0m} son los asociados al sesgo), posteriormente transformada por una función no lineal que denotaremos por ϕ, denominada función de activación. Se suele tomar la función logística, aunque también es común optar por la función bisagra, denominada *ReLU (Rectified linear unit)*, que se define como sigue:

$$(x)_+ = \begin{cases} x & \text{si } x > 0, \\ 0 & \text{si } x \leq 0. \end{cases}$$

Así, cada unidad oculta sería de la forma siguiente.

$$h_m(\mathbf{x}) = \phi\left(\omega_{0m} + \omega_{1m}x_1 + \cdots + \omega_{pm}x_p\right).$$

En el caso de estar ante un problema de regresión, que es el que nos plantean las series de tiempo, el modelo final es una combinación lineal de las variables ocultas:

$$f(\mathbf{x}) = \gamma_0 + \gamma_1 h_1(\mathbf{x}) + \cdots + \gamma_M h_M(\mathbf{x}). \tag{3.1}$$

Es habitual también considerar una función de activación en el nodo final, para adaptar la predicción a distintos tipos de respuestas.

Dediquemos un momento a pensar en la no linealidad de las funciones de activación. Si queremos que nuestra red neuronal cumpla objetivos complejos, es una condición indispensable que las funciones de

activación sean no lineales ya que, matemáticamente, una combinación lineal de combinaciones lineales sigue siendo una combinación lineal. En consecuencia, una red de, supongamos, cinco capas intermedias, equivaldría a una de una sola capa que solo podría separar datos con líneas rectas. Es más, como se puede ver en el Teorema 1 de Cybenko (1989) una capa oculta es suficiente para aproximar cualquier función continua en un conjunto compacto usando funciones de activación sigmoideas, a pesar de que no se especifica el número de unidades ocultas o la forma de encontrar los pesos exactos. Este resultado fue posteriormente extendido en Hornik (1991), donde ya no se menciona la necesidad de funciones sigmoideas, sino simplemente acotadas y no constantes. Notemos que esto implícitamente se refiere a funciones esencialmente no lineales, ya que es sencillo comprobar que si una función es lineal y no constante, entonces no puede ser acotada. Sin embargo, existen casos donde está justificado el uso de las funciones de activación lineales, como por ejemplo en la capa final cuando se trata de predecir un resultado relativo a un problema de regresión.

Para obtener un mejor rendimiento en términos de memoria requerida y eficiencia, el algoritmo va tratando los datos por bloques de, por ejemplo, 32 o 64 datos, a los que llamamos *batches*. Recibe el nombre de *epoch* cada vez que el algoritmo completa el procesado de todos los datos. A este enfoque, que es quizá el más habitual, se le llama *Mini batch gradient descent*, y se puede considerar un punto intermedio en un espectro donde en el extremo izquierdo (tomar únicamente un dato por iteración) está el conocido como *Stochastic gradient descent*, que genera iteraciones muy rápidas pero mucho ruido y en el extremo derecho (tomar todos los datos) está el *Batch gradient descent*, que es muy lento y gasta mucha memoria, pero es más estable.

A continuación daremos una idea simplificada del proceso que sigue el ajuste de una red neuronal incorporando esos nuevos términos. Primero se ha de dividir la totalidad de los datos en *batches*. Cada uno de ellos dará lugar a una iteración del método y para cada una realizamos el *forward*, es decir, el proceso hacia delante hasta llegar a la predicción. Una vez se llega a la misma calculamos la pérdida, y con ella hacemos el *backward*, a partir del cual se obtiene el gradiente. Después, actualizamos los pesos asociados al modelo como la diferencia entre los pesos viejos y el producto entre la tasa de aprendizaje y el gradiente obtenido. Realizaríamos este proceso para cada *epoch*.

La tasa de aprendizaje, que denotamos nuevamente por η, tiene funciones muy importantes y similares a las que vimos cuando hablamos del boosting en la Sección 2.4 del Capítulo 2. Se encarga de controlar el volumen de la actualización y además de evitar la divergencia. Con una tasa de aprendizaje muy alta podríamos perdernos soluciones, pudiendo causar que la pérdida comience a oscilar o incluso que comience a aumentar de nuevo. Sin embargo, si aumentamos el tamaño del *batch*, deberíamos aumentar la tasa de aprendizaje. Cuando tenemos un *batch* pequeño el gradiente será más ruidoso que cuando tenemos uno grande, por lo que se necesita una tasa de aprendizaje más baja para que este ruido no desvíe a la red del camino que ha de seguir.

Hablemos de los parámetros asociados al modelo y el proceso *backward* mencionado anteriormente. Lo primero que debemos remarcar sobre este aspecto es algo que ya se mencionó en la introducción del capítulo, y es que las redes neuronales se consideran modelos hiperparametrizados, incluso para el modelo más sencillo con una cantidad moderada de variables predictoras. La estimación de los mismos se realiza minimizando una función de pérdida, de forma que se nos permite ver cómo de buenas son las predicciones de nuestro modelo. Como sabemos, si las predicciones son imprecisas, la función de pérdida tomaría un valor mayor, que disminuye a medida que estas se van acercando a la realidad. Además, como vimos en la Sección 2.4, se le suele pedir diferenciabilidad y la suma de los residuos al cuadrado es una elección extendida. Por lo tanto, dadas las observaciones $(x_i, y_i), i = 1, \ldots, n$, y denotando por $\theta = (\omega, \gamma)$, podríamos ajustar el modelo resolviendo un problema de mínimos cuadrados no lineal como el que se muestra a continuación.

$$R(\theta) = \min_{\theta} \frac{1}{2} \sum_{i=1}^{n} (y_i - f(x_i))^2, \tag{3.2}$$

donde $f(\mathbf{x})$ es la vista en la Ecuación (3.1). La solución exacta del problema suele ser imposible en la práctica, al tratarse de un problema no convexo, por lo que se ataca mediante un algoritmo heurístico de descenso de gradiente, llamado en este contexto *backpropagation*. Este algoritmo representa una estrategia para resolver el problema de la atribución del error. La lógica que sigue es la de propagar el residuo observado en la salida hacia atrás, a través de las capas de la red, para determinar cuánto ha contribuido cara parámetro al error total. Como veremos un poco más adelante, esta distribución se lleva a cabo al aplicar la regla de la cadena sobre la función de coste.

El algoritmo de *backpropagation* normalmente convergerá a un óptimo local, siendo rara la convergencia a uno global, por lo que es muy sensible a la solución inicial, que comúnmente se selecciona de forma aleatoria con valores próximos a cero. Otro problema asociado a la heurística de tipo gradiente es que se ve afectada de forma negativa por la correlación entre predictores. Cuando las correlaciones son muy altas, es usual preprocesar los datos, ya sea eliminando variables predictoras o utilizando componentes principales. A continuación, se presenta de manera formal la descomposición del gradiente de la función de coste, lo que permite observar cómo la regla de la cadena actúa como un mecanismo de asignación de residuos.

Proposición 3. *Para un modelo de red neuronal con arquitectura multicapa, el cálculo del gradiente de la función de coste asigna de forma proporcional una fracción del residuo de predicción $y_i - f(x_i)$ a cada parámetro del sistema.*

Demostración. El problema que se presenta es el de encontrar las direcciones en las que mover θ de forma que disminuya el objetivo en la Ecuación (3.2). Denotemos por θ_m el valor del que disponemos actualmente del conjunto de parámetros del modelo. El gradiente de $R(\theta)$ evaluado en el mencionado valor actual es el vector de derivadas parciales siguiente.

$$\nabla R(\theta_m) = \frac{\partial R(\theta)}{\partial \theta}_{|\theta=\theta_m},$$

donde el subíndice denota que el vector de derivadas parciales, una vez calculado, está evaluado en el valor de los parámetros del que disponemos actualmente. Esto nos proporciona la dirección de máximo ascenso del espacio en cuestión. Dado que la idea del descenso de gradiente es mover ligeramente el vector de parámetros en la dirección opuesta, hacemos lo siguiente.

$$\theta_{m+1} = \theta_m - \eta \nabla R(\theta_m).$$

En la ecuación previa, η representa la tasa de aprendizaje, y para un valor lo suficientemente pequeño de la misma, se producirá una disminución del objetivo $R(\theta)$. Como veremos más adelante, el cálculo del gradiente resulta ser bastante sencillo, incluso para redes con estructuras más complicadas, debido a la regla de la cadena. Como la función en cuestión es un sumatorio, su gradiente también lo será pero sobre las n observaciones, por lo que nos centraremos principalmente en el término siguiente:

$$R_i(\theta) = \frac{1}{2}\left(y_i - \gamma_0 - \sum_{m=1}^{M} \gamma_m \phi\left(\omega_{0m} + \sum_{j=1}^{p} \omega_{jm} x_{ij}\right)\right)^2.$$

Para simplificar las expresiones, consideremos: $z_{mi} = \omega_{0m} + \sum_{j=1}^{p} \omega_{jm} x_{ij}$. Ahora, podemos derivar primero con respecto a γ_m:

$$\frac{\partial R_i(\theta)}{\partial \gamma_m} = \frac{\partial R_i(\theta)}{\partial f(x_i)} \cdot \frac{\partial f(x_i)}{\partial \gamma_m} = -(y_i - f(x_i)) \cdot \phi(z_{mi}). \tag{3.3}$$

Ahora, con respecto a ω_{jm}:

$$\frac{\partial R_i(\theta)}{\partial \omega_{jm}} = \frac{\partial R_i(\theta)}{\partial f(x_i)} \cdot \frac{\partial f(x_i)}{\partial \phi(z_{mi})} \cdot \frac{\partial \phi(z_{mi})}{\partial z_{mi}} \cdot \frac{\partial z_{mi}}{\partial \omega_{jm}} = -(y_i - f(x_i)) \cdot \gamma_m \cdot \phi'(z_{mi}) \cdot x_{ij}. \tag{3.4}$$

Notemos que las dos expresiones a las que hemos llegado contienen el residuo $y_i - f(x_i)$. Esto apoya lo que comentamos previamente, ya que observemos en la Ecuación (3.3) que una fracción de este residuo se atribuye a cada unidad oculta, regulado por el valor de $\phi(z_{mi})$. Después, la Ecuación (3.4) nos permite ver una atribución similar para la entrada j a través de la unidad oculta m. Por lo tanto, acabamos de comprobar matemáticamente que el cálculo del gradiente asigna una fracción del residuo a cada uno de los parámetros mediante la regla de la cadena, tal y como hemos comentado cuando hemos esquematizado el proceso. □

Por otra parte, como hemos indicado anteriormente, las redes neuronales son modelos hiperparametrizados, por lo que tienen una gran tendencia al sobreajuste. Una forma de lidiar con este problema es penalizar el número de parámetros. En este contexto recibe el nombre de reducción de los pesos o *weight decay*:

$$\min_{\theta} RSS + \lambda \left(\sum_{m=1}^{M} \sum_{j=0}^{p} \omega_{jm}^2 + \sum_{m=0}^{M} \gamma_m^2 \right)$$

El parámetro regularizador, denotado por λ, suele seleccionarse por validación cruzada, confiando en que seleccione un valor que fuerce que un número considerable de parámetros sean nulos. Además, hay que tener en cuenta que al depender la penalización de una suma de pesos, es esencial que estos sean comparables, es decir, estén en escalas similares, para lo que se debe reescalar las variables predictoras antes de ajustar el modelo.

Además de la anterior, existe una nueva y eficiente forma de regularizar llamada *dropout*. Inspirada en el random forest, visto en la Subsección 2.3.1 del Capítulo 2, la idea se basa en suprimir de forma aleatoria una fracción, δ, de las unidades de una capa al ajustar el modelo, llevándose a cabo de forma independiente cada vez que una observación de entrenamiento es procesada. Las unidades restantes suplen a las suprimidas y sus pesos se escalan por un factor de $1/(1-\delta)$ para compensar. Lo que se consigue evitar con esta medida es que ciertos nodos se especialicen demasiado. En la práctica esto se lleva a cabo fijando a cero de forma aleatoria las activaciones de las unidades descartadas, manteniendo intacta la arquitectura de la red. Se proporciona una representación gráfica para ilustrar este concepto.

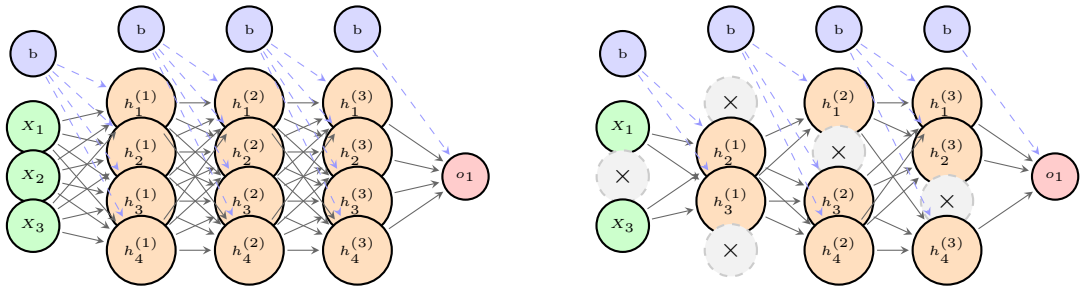


Figura 3.2: Comparación entre la arquitectura original (izquierda) y el efecto de *dropout* (derecha) durante el entrenamiento.

La Figura 3.2 nos permite ver la diferencia entre ambas situaciones durante el entrenamiento. Esta puntualización es importante, ya que si bien durante el entrenamiento se lleva a cabo la técnica como

hemos explicitado, durante el test no se realiza el *dropout*. Lo que se hace, en su lugar, es modificar los pesos. Si una unidad se retuvo con probabilidad p durante el entrenamiento, sus respectivos pesos se multiplicarán por p durante el test. Ilustramos esto en la Figura 3.3.

Entrenamiento: Prob. p de estar. Peso ω. Test: Siempre presente. Peso $p\omega$.

Figura 3.3: Regla de escalado de pesos durante el test.

Las implicaciones que esto tiene sobre la *backpropagation* son las más lógicas dadas las modificaciones realizadas. Centrémonos en un *batch* (después esto se extiende a cada uno de ellos, como ya se explicó anteriormente) concreto, en el que supongamos que hay un número D de observaciones. Cada una de esas D observaciones ha visto una red diferente, por lo que, a la hora de realizar la *backpropagation*, si un dato no ha visto un parámetro, aporta con gradiente nulo a la actualización del mismo. Después se promedian los resultados dentro del *batch* y se actualizan los pesos de la red completa.

Una vez sentadas las bases matemáticas de las FFN para el caso en el que solo contamos con una capa intermedia (a pesar de que la técnica del *dropout* se explicó en un contexto general de varias capas), podemos extender las nociones para una red con la misma arquitectura pero más capas ocultas. Esta extensión es necesaria, ya que las redes neuronales modernas típicamente tienen más de una capa oculta y varias unidades ocultas en cada una de ellas. Como vimos en Hornik (1991) y Cybenko (1989), una red con una única capa y un número suficiente de unidades ocultas es capaz de aproximar la mayoría de funciones, sin embargo, la tarea de dar con una solución satisfactoria se hace más sencilla con un número mayor de capas intermedias, cada una de ellas con un número más modesto de unidades ocultas.

A continuación exponemos las ecuaciones asociadas a un modelo FFN multicapa. En particular, las asociadas a las capas ocultas, permitiéndonos ver cómo se asocian unas con otras y haciendo evidente además la magnitud del número de parámetros que maneja este modelo.

Capa Oculta 1:
$$h_m^{(1)} = \phi\left(\omega_{0m}^{(1)} + \sum_{j=1}^{p} \omega_{jm}^{(1)} x_j\right), \quad m = 1, \ldots, M_1$$

Capa Oculta 2:
$$h_k^{(2)} = \phi\left(\omega_{0k}^{(2)} + \sum_{m=1}^{M_1} \omega_{mk}^{(2)} h_m^{(1)}\right), \quad k = 1, \ldots, M_2$$

Capa Oculta 3:
$$h_l^{(3)} = \phi\left(\omega_{0l}^{(3)} + \sum_{k=1}^{M_2} \omega_{kl}^{(3)} h_k^{(2)}\right), \quad l = 1, \ldots, M_3$$

$$f(\mathbf{x}) = \gamma_0 + \sum_{l=1}^{M_3} \gamma_l h_l^{(3)}$$

Por supuesto, el resultado y la estructura se pueden extender para el caso de una red neuronal con un número mayor de capas ocultas. En este caso, explicitamos las ecuaciones asociadas al ejemplo

que podemos ver en la Figura 3.1 ya que no se pierde generalidad y se considera más ilustrativo. En adelante, consideraremos el mismo número de capas para presentar los conceptos que siguen.

La lógica que hay tras esta extensión que estamos estudiando ahora, es la misma. Tanto en lo referente a propiedades del modelo, como al ajuste de los parámetros. El entrenamiento de redes profundas sigue el mismo principio de *backpropagation* detallado para el caso de una FFN con una sola capa oculta. La única diferencia es que el error se propaga recursivamente a través de varias capas ocultas en lugar de una aplicando, como dijimos anteriormente, la regla de la cadena sucesivamente desde la salida hasta la entrada.

Proposición 4. *En una red neuronal con múltiples capas ocultas, la atribución del error se mantiene y se generaliza de forma recursiva. El cálculo del gradiente asigna a cada parámetro una fracción del error de predicción, comenzando por la última capa oculta y retrocediendo hacia las anteriores.*

Demostración. Recuperemos el objetivo $R(\theta)$ que se deseábamos minimizar en el caso de una sola capa oculta, y extendámoslo al caso que tenemos actualmente, con tres capas ocultas. Ahora debemos considerar la ecuación que se puede encontrar a continuación, que presenta claras analogías en la forma con la asociada a una sola capa.

$$R_i(\theta) = \frac{1}{2}\left(y_i - \gamma_0 - \sum_{l=1}^{M_3} \gamma_l \phi\left(\omega_{0l}^{(3)} + \sum_{k=1}^{M_2} \omega_{kl}^{(3)} h_{ik}^{(2)}\right)\right)^2.$$

Nuevamente, simplifiquemos la expresión anterior mediante el siguiente cambio de notación:

$$z_{il}^{(3)} = \omega_{0l}^{(3)} + \sum_{k=1}^{M_2} \omega_{kl}^{(3)} h_{ik}^{(2)}$$

Primero, derivemos con respecto a γ_l:

$$\frac{\partial R_i(\theta)}{\partial \gamma_l} = \frac{\partial R_i(\theta)}{\partial f(x_i)} \cdot \frac{\partial f(x_i)}{\partial \gamma_l} = -(y_i - f(x_i)) \cdot \phi(z_{il}^{(3)}). \tag{3.5}$$

Ahora derivamos con respecto a $\omega_{kl}^{(3)}$:

$$\frac{\partial R_i(\theta)}{\partial \omega_{kl}^{(3)}} = \frac{\partial R_i(\theta)}{\partial f(x_i)} \cdot \frac{\partial f(x_i)}{\partial \phi(z_{il}^{(3)})} \cdot \frac{\partial \phi(z_{il}^{(3)})}{\partial z_{il}^{(3)}} \cdot \frac{\partial z_{il}^{(3)}}{\partial \omega_{kl}^{(3)}} = -(y_i - f(x_i)) \cdot \gamma_l \cdot \phi'(z_{il}^{(3)}) \cdot h_{ik}^{(2)}. \tag{3.6}$$

Se puede apreciar otra vez que las dos expresiones a las que hemos llegado contienen el residuo $y_i - f(x_i)$. Esta vez, se puede observar en la Ecuación (3.5) que una fracción de este residuo se atribuye a cada unidad oculta de la última capa, regulado por el valor de $\phi(z_{il}^{(3)})$. Después, la Ecuación (3.6) nos muestra una atribución similar para la activación k de la capa anterior a través de la unidad oculta l. Las analogías entre ambos casos son más que evidentes, y si se desease extender a un caso con un número tanto mayor como menor de capas intermedias, resulta inmediato. □

3.2. Recurrent Neural Networks

La arquitectura vista en la sección previa representa la base sobre la que construir lo que ha ido llegando posteriormente. Como es lógico, este tipo de redes neuronales se ha topado con ciertos problemas con los que no son capaces de lidiar de forma satisfactoria, haciendo que necesitemos expandir nuestras miras y conocimientos. Como ejemplo podríamos pensar en el campo del reconocimiento de imagen y vídeo. El desarrollo principal en este campo ha llegado de la mano de las redes neuronales convolucionales (CNN), que no abordaremos en este documento, pero pueden verse como redes neuronales para el procesamiento de datos en una rejilla.

Se sabe de la existencia de múltiples fuentes de datos que son secuenciales en la naturaleza, y que requieren de un tratamiento especial en la construcción de modelos predictivos. Podríamos pensar por ejemplo en documentos, como libros o reseñas. La secuencia y posiciones relativas de las palabras en un documento captan la narrativa, tema y tono y pueden ayudar en tareas como por ejemplo la de clasificación del tema que se trata en el documento, análisis del sentimiento que quiere transmitir el autor o incluso traducción de idiomas.

Por último, consideremos el ejemplo que más nos interesa dado el objetivo del estudio, las series de tiempo. Estas son otro ejemplo claro de datos secuenciales. Ya sean series de tiempo asociadas a factores climatológicos, a través de las que queremos predecir el tiempo meteorológico dentro de unos días, o series de tiempo financieras, por ejemplo, que a pesar de ser de difícil predicción pueden dar lugar a resultados razonables. Para todas estas hacen falta redes neuronales que posean una memoria para captar el carácter secuencial de los datos, y las que hemos estudiado en esta sección se quedan cortas en este aspecto. Por este motivo, vamos a redirigir el rumbo del estudio hacia nuevas arquitecturas que nos ofrezcan este tipo de funcionalidades. En particular, en adelante nos centraremos en las redes neuronales recurrentes (RNN por sus siglas en inglés), que son las predominantes en el campo de estudio en el que nos estamos enfocando.

Como decíamos, las RNN, son una familia de redes neuronales diseñada específicamente para el manejo de datos sencuenciales, a la que se le ha de proporcionar precisamente una secuencia, que podemos denotar por X, como objeto de entrada. Se puede pensar a modo de ejempo en un corpus de documentos. Cada documento se puede representar por una secuencia de, digamos, L palabras, de forma que $X = \{X_1, X_2, \ldots, X_L\}$, donde cada elemento representa una palabra. La posicion relativa de las palabras, así como el orden en que vienen dadas, proporcionan al escrito un sentido semántico. Las redes neuronales recurrentes están diseñadas para lidiar y sacar partido al formato secuencial de los datos de partida y producir una salida que puede ser tanto una nueva secuencia, como ocurre en la traducción de textos, como un escalar, como la etiqueta binaria sobre el sentimiento producido por algo plasmado en una reseña: me ha gustado o no me ha gustado. A continuación ilustramos gráficamente la arquitectura de este tipo de redes, que nos ayudará a visualizar mejor la forma en la que esta trabaja sobre estas sencuencias de datos.

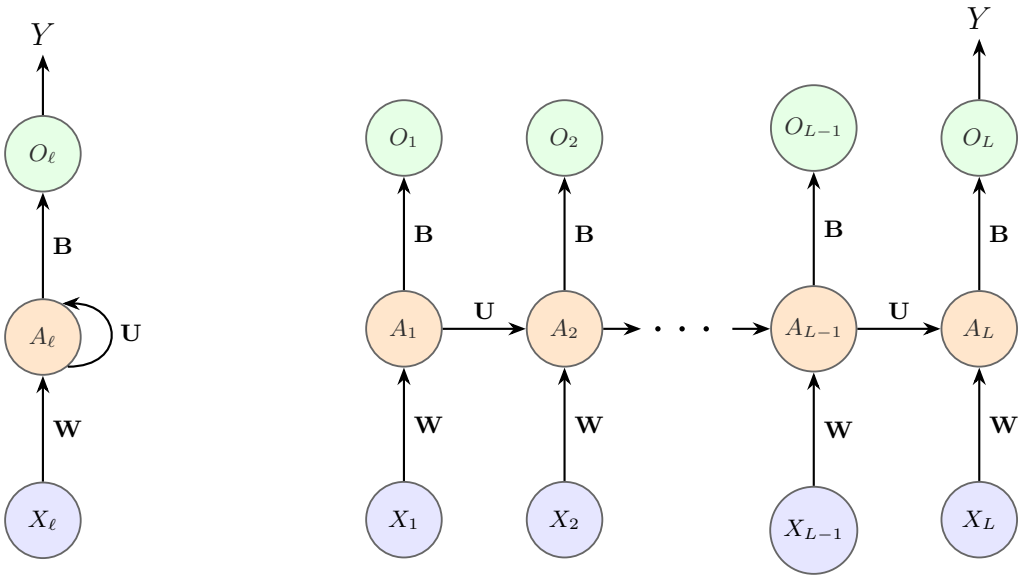


Figura 3.4: Arquitectura de una red neuronal recurrente (RNN). De izquierda a derecha: forma compacta con bucle y forma desdoblada a través del tiempo.

La estructura mostrada en la Figura 3.4 presenta una única capa oculta, por lo que se trata de un ejemplo sencillo. Se puede apreciar que como entrada se proporciona la secuencia $X = \{X_1, X_2, \ldots, X_L\}$ que antes representaba las palabras de nuestro documento, pero que en un futuro será la serie de tiempo sobre la que queramos aprender y predecir. Es importante destacar que cada elemento X_ℓ es un vector en sí mismo que contiene la información deseada. Así como el número de vectores, L, puede variar (una oración puede tener 5 palabras, y otra 10), la dimensión del vector es fija, y será la misma para todos.

Para entender esto bien necesitamos una noción de la estructura interna de los vectores, que varía con la naturaleza del problema considerado. Siguiendo con el ejemplo de las palabras, en este caso es habitual que el vector X_ℓ se presente como un *Word Embedding*. Esto es, un vector denso de números reales con una dimensión determinada, donde cada número representa una característica semántica aprendida de la palabra. En el caso del ejemplo que más nos interesa, las series de tiempo, sería un vector donde cada elemento representa una característica medida en ese instante de tiempo. Por ejemplo, para datos de exportaciones, podríamos tener un vector de dimensión tres, donde la primera componente representa el arancel promedio en el instante de tiempo considerado, la segunda el tipo de cambio en el mismo instante de tiempo y la tercera el PIB del país al que se exporta, en el mismo instante de tiempo.

A medida que esta secuencia se procesa (cada vector de forma individual), la red actualiza las activaciones A_ℓ de la capa oculta, tomando tanto el vector de entrada X_ℓ como la activación anterior A_{l-1} del paso previo de la secuencia. Como se puede apreciar también en la Figura 3.4, cada una de las activaciones de la secuencia $\{A_\ell\}_{l=1}^{L} = \{A_1, A_2, \ldots, A_L\}$ produce una predicción, O_ℓ, para Y, siendo la última, O_L, la principal.

Centrémonos ahora en entender matemáticamente lo que está ocurriendo. Supongamos que cada vector X_ℓ tiene p componentes $X_\ell^T = (X_{\ell 1}, X_{\ell 2}, \ldots, X_{\ell p})$, y la capa oculta posee K unidades $A_\ell^T = (A_{\ell 1}, A_{\ell 2}, \ldots, A_{\ell K})$. Cada una de estas unidades se especializa en detectar algo distinto. Cuanto mayor sea K, más memoria y capacidad de entender relaciones complejas tiene la red, pero hay que tener cuidado con seleccionar un valor demasiado alto, ya que esto podría derivar en un tiempo de entrenamiento muy elevado, así como en un mayor riesgo de sobreajuste. Observemos la matriz $\mathbf{W}$, que contiene un total de $K \times (p+1)$ pesos compartidos ω_{kj} para la capa de entrada. De forma similar, $\mathbf{U}$

es una matriz $K \times K$ de pesos u_{ks} para la comunicación interna entre las activaciones y $\mathbf{B}$ es un vector de pesos de longitud $K+1$ que contiene los pesos β_k para la capa de salida. Por lo tanto, se pueden enlazar estos conceptos mediante la fórmula siguiente.

$$A_{\ell k} = g\left(w_{k0} + \sum_{j=1}^{p} w_{kj} X_{\ell j} + \sum_{s=1}^{K} u_{ks} A_{\ell-1,s}\right).$$

Esta es una representación compacta de lo que ocurre en el proceso. Veamos el funcionamiento interno. La parte que está dentro de la función g, se llama la pre-activación. Podemos denotarla por $a_{\ell k}$ de forma que:

$$a_{\ell k} = w_{k0} + \sum_{j=1}^{p} w_{kj} X_{\ell j} + \sum_{s=1}^{K} u_{ks} A_{\ell-1,s}. \tag{3.7}$$

Esta pre-activación es una combinación del impacto del pasado, $A_{\ell-1}$, ponderado por $\mathbf{U}$ e impacto del momento actual, X_ℓ, ponderado por $\mathbf{W}$. Después, esta pre-activación se pasa por una función g, que suele ser la tangente hiperbólica, de forma que se comprime el valor de la misma al intervalo $[-1, 1]$.

$$A_{\ell k} = \tanh(a_{\ell k}) = g(a_{\ell k}) = g\left(w_{k0} + \sum_{j=1}^{p} w_{kj} X_{\ell j} + \sum_{s=1}^{K} u_{ks} A_{\ell-1,s}\right),$$

llegando nuevamente a la expresión que teníamos de inicio. De esta forma, gracias a la tangente hiperbólica, si $a_{\ell k}$ es un valor muy grande o muy pequeño, se saturará en el 1 o -1 respectivamente. Además, pequeños cambios en $a_{\ell k}$ no afectarán a $A_{\ell k}$. Así, la salida es de la forma siguiente.

$$O_\ell = \beta_0 + \sum_{k=1}^{K} \beta_k A_{\ell k}.$$

Por otra parte, notemos que estamos usando los mismos pesos $\mathbf{W}$, $\mathbf{U}$ y $\mathbf{B}$ para cada elemento de la secuencia. Esta es una característica que presentan las RNN, denominada uso compartido de parámetros. Además de un efecto regularizador, en el sentido de que se restringe el modelo al uso del mismo conjunto de pesos para múltiples tareas, el uso compartido de parámetros nos permite extender y aplicar el modelo a ejemplos de diferentes formas. Las RNN pueden procesar secuencias de mucha mayor longitud gracias a esto. En conclusión, resulta evidente que a medida que procedemos desde el comienzo hasta el final, las activaciones A_ℓ acumulan un historial de lo que han ido viendo, de forma que el contexto aprendido pueda ser usado para la predicción.

En cuanto a la cuantificación del error, o lo que es lo mismo, la diferencia que hay entre O_L e Y ilustrado en la Figura 3.4, tenemos que en problemas de regresión solo implica a la salida final O_L, indicando que las salidas intermedias no se utilizan explícitamente. Aun así, cuando ajustamos el modelo cada elemento X_ℓ de la secuencia de entrada contribuye a O_L a través de la cadena vista en la Figura 3.4, y como consecuencia de esto también contribuye al aprendizaje de los parámetros compartidos, $\mathbf{W}$, $\mathbf{U}$ y $\mathbf{B}$, mediante la función de pérdida gracias a la técnica de *backpropagation*. Si tenemos n pares de secuencias de entrada/respuestas, (X_i, Y_i), los parámetros se buscan mediante la minimización del siguiente sumatorio:

$$\sum_{i=1}^{n} (Y_i - O_{iL})^2 = \sum_{i=1}^{n} \left(Y_i - \left(\beta_0 + \sum_{k=1}^{K} \beta_k g\left(w_{k0} + \sum_{j=1}^{p} w_{kj} X_{iLj} + \sum_{s=1}^{K} u_{ks} A_{i,L-1,s}\right)\right)\right)^2.$$

No obstante, conviene hacer una aclaración, y es que las predicciones intermedias no solamente tienen una labor implícita dentro de la red, que es el rol principal que les hemos asignado en este texto. Además de 'funciones acumulativas' hay determinadas tareas de aprendizaje en las que la respuesta es también una secuencia, y por lo tanto $\{O_1, O_2, \ldots, O_L\}$ se necesita de forma explícita, como por ejemplo en labores de traducción, en las que es necesaria otra secuencia como salida.

Por otra parte, es importante dejar claro que cuando hablamos de *backpropagation* en el contexto de las RNN, no es lo mismo que cuando hablamos del mismo concepto en el contexto de las FFN. En la *backpropagation* estándar, los gradientes se propagan hacia atrás de una unidad oculta a otra, pero en el caso de las redes neuronales recurrentes ocurre algo distinto, y es que estos se propagan hacia atrás dentro de una única unidad, pero a través del tiempo. Se trata de un caso especial de *backpropagation* desarrollado para redes neuronales recurrentes, que recibe el nombre de *Back Propagation Through Time (BPTT)*.

El problema que presenta la *backpropagation* es que no sabe lidiar con bucles. Solo sabe ir en línea recta hacia atrás para el ajuste de los pesos. Por tanto, para que el algoritmo clásico funcione, lo que se hace es desdoblar la serie, convirtiendo este bucle que se puede apreciar en la gráfica de la izquierda de la Figura 3.4 en una línea temporal plana. Al hacer esto, como se puede apreciar en la gráfica de la derecha de la Figura 3.4, vemos que cada bloque temporal posee dos conexiones de salida, una hacia el *output* y otra hacia delante. Por lo tanto, el mismo bloque temporal debe recibir la suma de dos señales de error cuando se realice el viaje hacia atrás, una del error de predicción en ese instante de tiempo y otra del error que ha hecho que arrastren los futuros instantes de tiempo. En esto es en lo que consiste la idea del *BPTT*. Primero se realiza el paso hacia delante o *forward*, al final del cual se lleva a cabo la predicción y se mide el error. Posteriormente se toma este error y se envía hacia atrás. Por último, solamente queda modificar los pesos acorde a los gradientes calculados, de forma que estos se actualizan una sola vez. Si se desea más información o las ecuaciones explícitas de las mismas se puede consultar Werbos (1990), pero dado que ya se expusieron las asociadas a *backpropagation* y para el objetivo que nos ocupa en este momento no son estrictamente necesarias, se proporciona la idea general, facilitando la referencia por si se desean realizar futuras comprobaciones.

A su vez, existen problemas asociados a esta técnica recientemente presentada, como el desvanecimiento de gradiente. Este ocurre cuando los gradientes, que en esencia son los que dictan cuánto deben cambiar los parámetros de la red, se vuelven extremadamente pequeños a medida que se propagan hacia atrás. Dado que, como decíamos, estos gradientes son esenciales para actualizar los pesos del modelo, su desaparición significa que los primeros pasos de la red, dejan de aprender. Este fenómeno impide que el modelo capture patrones complejos en los datos, limitando la profundidad y rendimiento de las arquitecturas de aprendizaje profundo. Observemos cómo esto puede afectar de lleno a las RNN ya que al calcular los gradientes, la derivada de la función tangente hiperbólica se ve implicada, y sabemos que esta es de la forma:

$$\frac{d}{dx}\tanh(x) = 1 - \tanh^2(x),$$

que siempre es menor que la unidad. Además, cuanto mayor sea la magnitud de la activación, menor será este valor. Esto provocará que cuando se propague hacia atrás a lo largo de múltiples instantes temporales, los gradientes se hagan más pequeños de forma exponencial, haciendo que los primeros contribuyan mínimamente a la actualización de los pesos. Esto hace que la red olvide dependencias a largo plazo. Como veremos más adelante, arquitecturas más modernas de RNN, como la Long Short Term Memory (LSTM) y Gated Recurrent Unit (GRU), llegaron para, entre otras cosas, solucionar este problema propio de las *Vanilla RNN*, que es como se conoce a las RNN clásicas.

De la misma forma que el gradiente se desvanece, es conveniente tener en cuenta que si se propaga hacia atrás multiplicándose en múltiples ocasiones, también puede ocurrir lo opuesto, dando lugar a un

problema conocido como gradiente explosivo. En este caso, como decíamos, ocurre todo lo contrario, ya que se produce cuando los valores utilizados para actualizar los pesos de la red se acumulan, volviéndose excesivamente grandes. Esta inestabilidad impide que el modelo converja, interrumpiendo el proceso de aprendizaje. Para mitigar este efecto es habitual emplear la técnica de recorte de gradiente, que implica establecer un valor umbral, de forma que si la norma del vector de gradiente excede este umbral, se reduce para ajustarse al límite.

Hasta ahora hemos visto redes neuronales recurrentes que poseen una única capa oculta. La extensión lógica, análoga a la vista para las FFN, sería apilar más capas unas sobre otras, de forma que estas se conecten buscando un efecto colaborativo o de sinergia. A este concepto se le conoce como *stacked RNN*, y es un modelo especial que cuenta con múltiples redes neuronales recurrentes, una en cada una de las capas. Esto crea una pila (*stack* en inglés), donde cada capa procesa la secuencia de entrada. A continuación, en la Figura 3.5 encontramos una representación gráfica que nos ayudará a entender la arquitectura de la que estamos hablando.

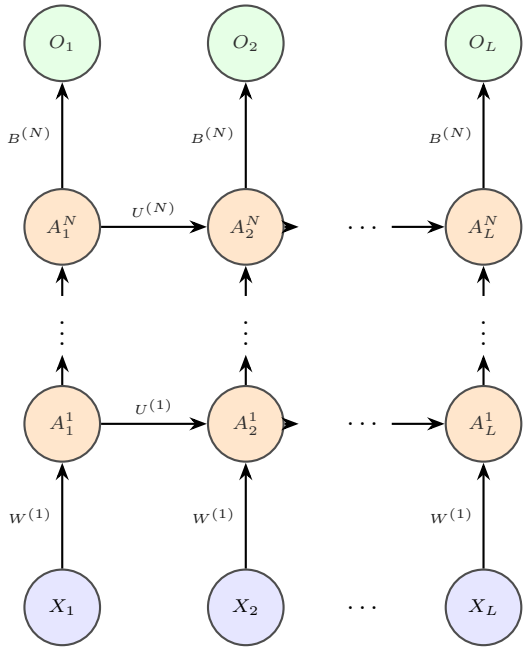


Figura 3.5: Esquema de una arquitectura *stacked RNN*.

La idea que hay detrás de esta novedosa arquitectura es que al apilar múltiples capas de redes neuronales recurrentes, una encima de otra, las salidas en cada instante de tiempo se convierten las entradas de la red de la capa siguiente. Cada capa poseerá una memoria, lo que habilita un aprendizaje jerárquico de características. Este describe la manera en la que los modelos profundos, como el que estamos revisando, van incorporando información cada vez más compleja, pasando de patrones simples de bajo nivel a significados estructurados de alto nivel a través de varias capas de procesamiento. Para ejemplificar esto, se puede pensar en que las primeras capas observan pequeños fragmentos de una pieza, como los bordes, y poco a poco se llega a las últimas, en las que se termina reconstruyendo la forma completa del objeto. Esto les permite aprender y recordar información en secuencias más largas. Cuantas más capas se añadan al modelo, esta estructura apilada será capaz de captar información cada vez más compleja. No obstante, esto también tiene sus limitaciones y es que resulta evidente que si el modelo se fija y depende de patrones muy específicos, pierde una capacidad de generalización que también era beneficiosa.

Vamos a tratar de desglosar lo que se puede ver en la Figura 3.5. Lo primero que notamos es que los pesos asociados a cada capa son diferentes, es decir, cada capa tiene una identidad propia. Por otra parte, para visualizar como fluye el procesamiento de información, lo más sencillo es pensarlo por instantes de tiempo. En el primer instante de tiempo, la primera capa recibe la entrada X_1 y crea su activación A_1^1, ponderando con los pesos correspondientes. Por su parte, la segunda capa recibe esta información proveniente de la primera capa y la pondera debidamente para obtener su activación, para después pasarlo hacia las capas superiores. En el segundo instante de tiempo, la primera capa recibe X_2 y la información de A_1^1, las pondera y le envía la información a la segunda capa, que debe ponderarla junto con la información que recibe de la misma capa pero del instante temporal previo. Este proceso ocurre de forma sucesiva, hasta llegar hasta el último instante de tiempo y la última capa, que nos proporcionará la salida deseada. Conceptualmente, el ajuste ocurre en diagonal, de forma entrelazada como acabamos de explicar, pero en la práctica es común realizar el ajuste por capas, a través de los instantes de tiempo.

El inconveniente de la arquitectura *stacked RNN*, además de la pérdida de generalización debida al aprendizaje jerárquico de características, es que arrastra dos de los problemas de las RNN simples y los empeora acorde con el número de capas que posea. Estamos hablando del desvanecimiento de gradiente y del gradiente explosivo. Recordemos que estos eran dos problemas que surgian al llevar a cabo la *BPTT*. El primero de ellos estaba asociado con una multiplicación reiterada de valores menores que uno, provocando que los primeros instantes de tiempo tuviesen poca influencia sobre el resultado final, mientras que el otro se asociaba con todo lo contrario, el producto reiterado de valores grandes debidos a los pesos asociados al modelo. Las *stacked RNN* incrementan drásticamente el número de unidades a través de las cuales debemos realizar la *backpropagation* provocando una situación más idónea para la ocurrencia de estos problemas. Por este motivo, en la práctica esta arquitectura suele usarse apilando redes neuronales recurrentes de tipo LSTM o GRU, que ya comentamos que implementaban herramientas para paliar, en particular, el desvanecimiento de gradiente.

3.3. Long Short Term Memory

Como hemos visto a lo largo de la Sección 3.2, las redes neuronales recurrentes son capaces de trabajar con datos secuenciales de forma eficiente. De hecho, en teoría también son capaces de capturar las dependencias a largo plazo de forma satisfactoria, pero el problema es que en la práctica esto no tiene por qué ser asi debido, en particular, a los problemas vistos, como el desvanecimiento de gradiente. En este contexto aparecen las redes Long Short Term Memory (LSTM), que son un tipo de arquitectura de red neuronal recurrente capaz de aprender estas dependencias a largo plazo que muchas veces se les escapan a las *vanilla RNN*. De hecho, las redes LSTM están específicamente diseñadas para evitar el problema de las dependencias a largo plazo. Antes de continuar, cabe destacar dos aspectos técnicos. El primero es que a partir de ahora, por simplicidad, para las dos secciones que quedan, denotaremos los instantes de tiempo por t, en lugar de ℓ. El segundo es que la información obtenida para la realización de esta sección fue obtenida principalmente de las siguientes fuentes: Olah (2015), Leo (2024b) y Pompas Gutiérrez (2023).

Las redes neuronales recurrentes que hemos estudiado hasta el momento comparten una estructura que encadena módulos simples uno detrás de otro, como pudimos ver por ejemplo en la Figura 3.4. Hasta ahora hemos observado la estructura global de como se trasporta la información entre los módulos, pero la clave ahora está en el interior de dichos módulos A_t. Los asociados a las redes vistas hasta el momento poseen una estructura interna muy simple, como la que se muestra a continuación en la Figura 3.6.

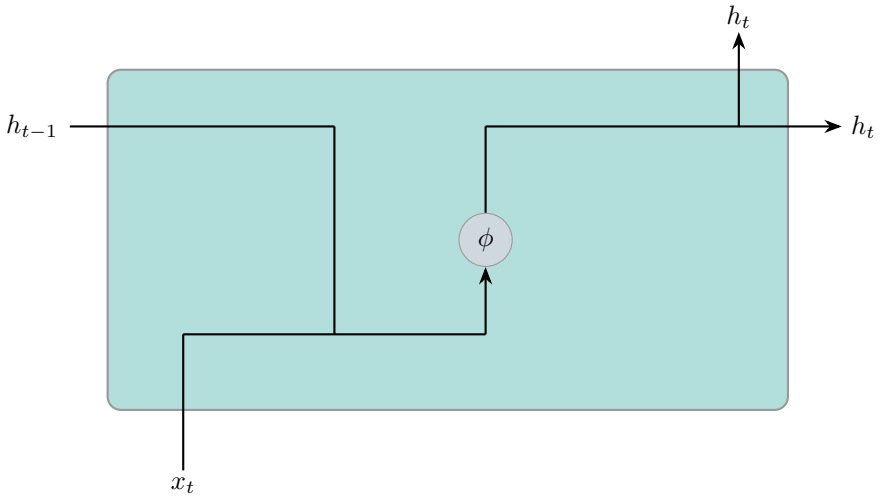


Figura 3.6: Diagrama de la celda de una red neuronal recurrente al uso.

Como vemos en la mencionada Figura 3.6 la estructura interna de las celdas encadenadas es muy sencilla. Cuando dos líneas convergen denota concatenación, mientras que si divergen denota que su contenido ha sido copiado y las copias se dirigen en direcciones distintas. Se puede apreciar que el flujo proveniente de la celda previa se combina con la entrada correspondiente al instante de tiempo actual, se pasa por una función de activación que denotamos por ϕ, aunque como ya se indicó previamente, suele ser la tangente hiperbólica, para después propagar el flujo hacia arriba, si procede, a otras capas, y al siguiente instante de tiempo.

Las LSTM, a pesar de tener también una estructura en la que se encadenan los distintos módulos, poseen celdas más complejas para tratar de solucionar el problema de la memoria a largo plazo. A continuación, en la Figura 3.7 mostramos el diagrama de una de las celdas correspondientes a esta arquitectura para poder entender de qué forma estas mejoran a las vistas en la Figura 3.6.

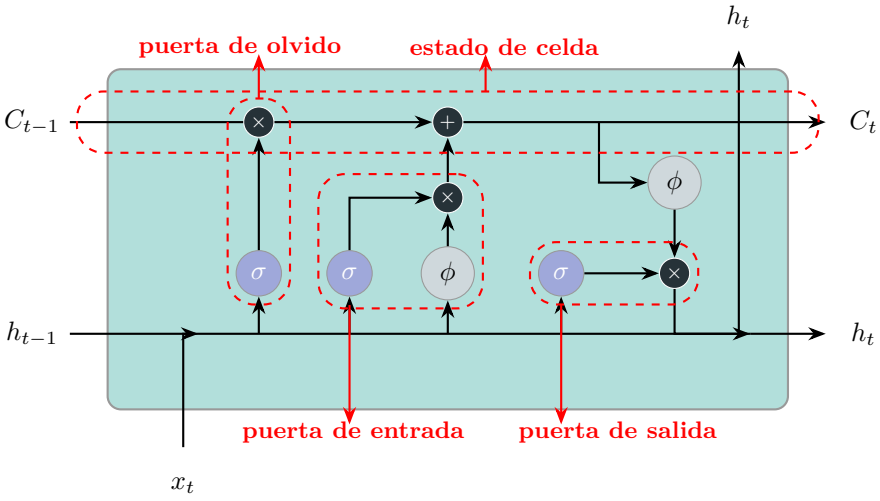


Figura 3.7: Diagrama de la celda de una red LSTM.

La Figura 3.7 puede parecer muy confusa, con muchas cosas pasando simultáneamente. La realidad es que es muy sencilla y organizada, a pesar de lo que pueda parecer debido al aumento de complejidad evidente con respecto a la de la Figura 3.6. La clave está en lo que llamamos estado de celda o "cell state", que es la línea horizontal que se mueve por la parte superior del diagrama. Esta hace las veces de cinta transportadora o memoria a largo plazo, ya que pasa a través de toda la celda (y toda la cadena, entre celdas) con ciertas interacciones menores, con el objetivo de transportar el contexto general a lo largo del tiempo. No debemos confundirla con el estado oculto o memoria a corto plazo h_{t-1}, que es la salida que produjo la celda en el paso inmediatamente anterior.

La red LSTM tiene la habilidad de modificar la información que pasa por el estado de celda mediante las estructuras llamadas puertas o "gates" que se pueden encontrar en la celda recuadradas en rojo. Estas representan una opción de dejar pasar información y están compuestas por una capa sigmoide y una operación de multiplicación punto a punto. La función sigmoide por la que se pasa cuando la información atraviesa la capa homónima, es de la forma:

$$\sigma(x) = \frac{1}{1 + e^{-x}}.$$

Esta función proporciona valores entre cero y uno, describiendo qué cantidad de cada componente se debe dejar pasar. Un valor de cero indica que no se debe dejar pasar nada de información, mientras que un valor de uno indica que se debe dejar pasar todo. Como podemos ver en la Figura 3.7 una red LSTM posee tres puertas de este estilo, y a continuación veremos más en profundidad su funcionamiento.

La primera de las puertas es la que en la imagen denominamos puerta de olvido o "forget gate". Su nombre es realmente muy ilustrativo, ya que su función es proteger la memoria a largo plazo, haciendo que esta se deshaga, o en otras palabras olvide, la información que ya no es relevante. Analicemos lo que ocurre dentro para entender cómo logra esto. Podemos ver que la red primero toma la memoria a corto plazo proveniente de la celda previa y la nueva entrada correspondiente a este instante de tiempo. Posteriormente, en la capa sigmoide se produce la toma de decisiones, es decir, se decide qué información se queda. Para ello, al pasar estos dos datos por la función sigmoide, los transformará en un vector con valores entre cero y uno al que llamaremos f_t. Ahora, para eliminar u olvidar la información, la red toma la memoria a largo plazo y la multiplica por f_t. Si el valor de este último era cero, ya que la sigmoide ha decidido que ya no importa, al multiplicarlo por C_{t-1} esta información desaparece. Por el contrario, si el valor es un uno la mantiene intacta, y si es por ejemplo 0.5 la información se desvanece parcialmente. Formalmente, esto se representaría como:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f),$$

donde h_{t-1} es la salida correspondiente a la celda anterior, x_t la entrada en este instante de tiempo y $[h_{t-1}, x_t]$ el vector que los concatena. Además tenemos W_f y b_f, que son la matriz de pesos y el vector de sesgos asociados a esta puerta, respectivamente.

Una vez realizado el primer paso, llega el momento de decidir qué información nueva, de la que está pasando en este instante de tiempo, se debe almacenar para producir el nuevo estado de celda que continuará hacia el siguiente módulo. De esto se encargará una nueva puerta, llamada puerta de entrada o "input gate". Este nuevo paso se divide en dos procesos. El primero está a cargo de la capa sigmoide que podemos ver dentro de la susodicha puerta, y es el que decide qué valores se deben actualizar. Se genera un vector que podemos denotar por i_t con número entre el cero y el uno. Un valor de cero significa no actualizar, mientras que un valor de uno significa actualizar por completo:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i),$$

donde $[h_{t-1}, x_t]$ representa el mismo vector concatenado de antes, y W_i y b_i la matriz de pesos y el vector de sesgos asociados a esta puerta, respectivamente. El segundo proceso está a cargo de la función ϕ, que sabemos que suele ser la tangente hiperbólica, y es crear un vector con los nuevos candidatos, representados por $\tilde{C}_t$. Tal y como se indicó anteriormente, la ventaja que aporta la tangente hiperbólica es que empuja los valores dentro del intervalo $[-1, 1]$, ayudando a mantener una estabilidad numérica dentro de la red:

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C),$$

donde $[h_{t-1}, x_t]$ representa el mismo vector concatenado que en las dos fórmulas previas, W_C y b_C la matriz de pesos y el vector de sesgos asociados a esta puerta, respectivamente, pero esta vez observemos que la función por la que se pasa el resultado es la tangente hiperbólica, ϕ, en lugar de la sigmoide, σ. Finalmente, ambas salidas se multiplican punto a punto y con toda la información recopilada en base a las puertas de olvido y de entrada, se puede actualizar la memoria a largo plazo para que siga hacia el siguiente módulo.

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t.$$

Con toda la información reunida llega el momento de decidir cuál va a ser la salida que exporte el módulo actual hacia posibles capas superiores y futuros instantes de tiempo. La respuesta es que lo que se pasará los próximos módulos será una versión filtrada del estado de celda actualizado. Fijémonos ahora en el recuadro llamado puerta de salida o "output gate". Lo primero es que la capa sigmoide perteneciente a esa puerta determine qué partes del estado de celda se van a sacar, es decir, qué partes de la memoria a largo plazo queremos emitir en este instante de tiempo. Podemos denotar este vector por o_t.

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o).$$

En la fórmula previa se pueden encontrar los mismos términos que veníamos observando, pero adaptados a la puerta actual. Esto es: $[h_{t-1}, x_t]$ es el vector que concatena h_{t-1} y x_t y , W_o y b_o la matriz de pesos y el vector de sesgos asociados a esta puerta, respectivamente, todo pasado por la función sigmoide. Después pasamos nuestro estado de celda por una función ϕ, que podemos asumir que es la tangente hiperbólica, para empaquetar de nuevo los valores dentro del intervalo $[-1, 1]$ y multiplicamos punto por punto ambos resultados, para emitir en este instante solamente las partes que hemos decidido, tanto hacia arriba, llegando a la siguiente capa si la hay, como hacia delante, llegando al módulo correspondiente al siguiente instante temporal. Formalmente traduciríamos esto como:

$$h_t = o_t * \tanh(C_t).$$

Ahora que hemos analizado internamente las celdas de las redes neuronales recurrentes, tanto de las más comunes como de las correspondientes a la arquitectura LSTM, tenemos una mejor comprensión de por qué estas últimas mejoran a las otras. Lo que haremos a continuación será tratar de ir más allá, dando un último paso que busque mejorar una red neuronal recurrente tan avanzada como es la LSTM. Para lograr esto, buscaremos esta mejora en términos de eficiencia computacional y facilidad a la hora de entrenar, introduciendo la arquitectura Gated Recurrent Unit o GRU. Con una menor cantidad de parámetros, las GRU poseen una estructura de puertas que les otorga la capacidad de capturar las características esenciales en series de tiempo y ayuda a mitigar el desvanecimiento de gradiente convirtiéndolas en una opción muy adecuada para secuencias largas de datos.

3.4. Gated Recurrent Unit

En esta sección revisaremos la arquitectura GRU, que si bien comparte la idea subyacente de LSTM, difiere en la forma de ejecutarla. La diferencia principal entre LSTM y GRU radica en la forma en la que gestionan el estado de celda de la memoria. Recordemos que las LSTM mantenían el estado de celda separado de la memoria a corto plazo, y se actualizaba con la ayuda de tres puertas. En GRU el estado de celda de memoria se reemplaza con un vector de activación candidato, que se actualiza mediante dos puertas, llamadas puerta de actualización y puerta de reinicio. Ilustremos la estructura interna de las celdas asociadas a la arquitectura GRU, para ver de qué forma logran sus objetivos. Pero antes, indicamos que la información obtenida para la realización de esta sección fue obtenida principalmente de las siguientes fuentes: Olah (2015), Kostadinov (2017), Anishnama (2023), Leo (2024a) y AllElectroHub (2025). Se puede encontrar la susodicha ilustración en la Figura 3.8.

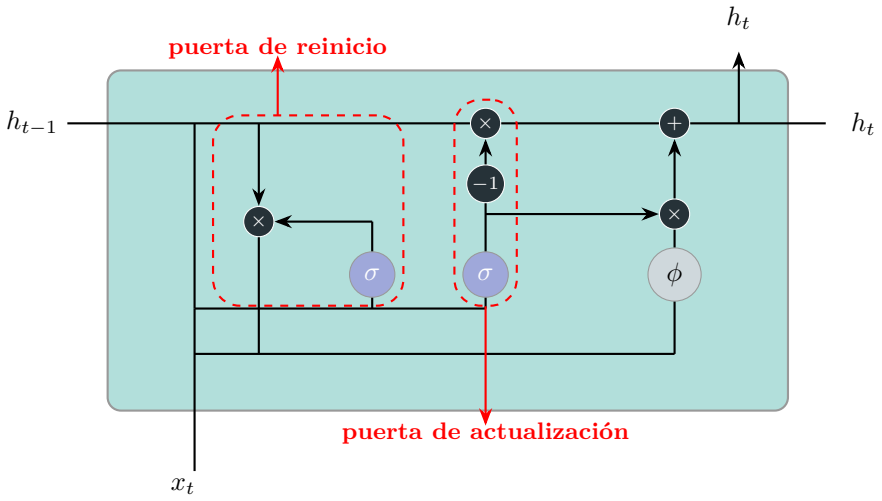


Figura 3.8: Diagrama de la celda de una red GRU.

En la Figura 3.8 se puede encontrar una estructura claramente más compleja que la expuesta en la Figura 3.6, pero menos recargada que la que pudimos ver en la Figura 3.7. El motivo es que en esta nueva celda se han combinado las puertas de entrada y olvido en una única puerta llamada puerta de actualización o "update gate". También fusiona el estado de celda y el estado oculto, haciendo que el modelo resultante sea, en efecto, más simple pero siga dando lugar a resultados excelentes.

Hablemos del proceso que sigue el módulo representado. Lo primero que se calcula es la puerta de actualización o "update gate" para el instante de tiempo t. Esta puerta determina cuánta información del pasado debería conservarse para influir en el estado oculto actual, h_t y es la que ayuda a GRU a retener la información importante en secuencias largas, haciéndola efectiva para el manejo de las mismas.

Para el cálculo asociado a esta puerta lo que hacemos es combinar de forma ponderada la entrada proporcionada en este instante de tiempo con la información proveniente de los $t-1$ instantes de tiempo anteriores, para después pasar esta combinación por la capa sigmoide. De esta forma, como vimos con LSTM, se obtiene un vector con valores entre cero y uno como salida, al que esta vez denotaremos z_t.

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t]).$$

En esta ecuación tenemos que z_t es, como decíamos, el resultado de la puerta de actualización en tiempo t, W_z es la matriz de pesos empleada para el cáldulo de esta puerta, h_{t-1} es el estado oculto

del instante de tiempo previo, x_t es la entrada en el instante de tiempo actual y $[h_{t-1}, x_t]$ representa la combinación o concatenación del estado oculto previo y la entrada actual. Por otra parte, al igual que ocurría con las LSTM, la función sigmoide se asegura que los valores del vector concatenado tomen valores comprendidos entre el cero y el uno, lo que permite a la puerta actuar como un filtro. Cuando z_t esté próximo a uno, GRU tenderá a optar más por la información obtenida en el presente, mientras que valores más próximos a cero harán que retenga más el pasado.

Una vez hecho esto, nos centramos en la puerta de reinicio o "reset gate" cuya función es, en esencia, ayudar al modelo a decidir cuánta información pasada debe olvidar. Para ello, calculamos un nuevo vector, denotado por r_t, obtenido como combinación ponderada los mismos elementos que antes, y nuevamente modificado a través de la capa sigmoide. Notemos que en esencia r_t se calcula igual que z_t, siendo los pesos que multiplican a los elementos la única diferencia:

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t]).$$

En la fórmula anterior r_t representa precisamente el resultado de la puerta de reinicio en el instante de tiempo t, W_r es la matriz de pesos para esta puerta, que recordemos es lo único que cambia y h_{t-1} vuelve a representar el estado oculto proveniente del instante de tiempo previo, igual que x_t vuelve a ser la entrada en el instante de tiempo actual y $[h_{t-1}, x_t]$ representa la concatenación de ambos. Se puede pensar en la puerta de reinicio como en un interruptor que emplea la función sigmoide para determinar cuánta información del pasado se debe ignorar cuando entra en juego información nueva. Si el valor de esta puerta esta próximo a cero, significa que el pasado se debe olvidar en favor de la novedad. Si, por el contrario, los valores son próximos a uno, la información del pasado se retendrá mayoritariamente. Dicho de otra forma, esta puerta permite a GRU resetear o reiniciar su memoria de forma selectiva, algo especialmente útil cuando hay, por ejemplo, un cambio drástico en el contexto de la secuencia estudiada.

Ahora estamos en condiciones de ver la forma en la que las mencionadas puertas afectarán a la salida final, siendo la puerta de reinicio la primera en aparecer con la llegada del estado candidato, denotado por $\tilde{h}_t$. Este representa la nueva información que podría potencialmente actualizar el estado oculto para que prosiga hacia futuros instantes, y es precisamente para la creación del mismo para lo que usamos la puerta de reinicio, en particular su decisión.

La creación del estado candidato representa el paso lógico para la incorporación de nuevo contenido en la memoria de la red. Estamos proponiendo un candidato que represente la información asociada a este instante de tiempo, pero para ello hacen falta tanto la entrada cruda del mismo, como la información del pasado que el modelo ha decidido mantener. Esta decisión de cuánta información del pasado mantener pasa, como hemos indicado antes, por la puerta de reinicio, que determina cuánta información del pasado el modelo ha decidido olvidar. Formalmente eso último se representaría como:

$$\tilde{h}_t = \tanh(W \cdot [r_t * h_{t-1}, x_t]).$$

En esta nueva ecuación tenemos nuestro estado candidato en el instante de tiempo t, $\tilde{h}_t$, la matriz de pesos para el mismo W, el estado oculto asociado al instante de tiempo anterior, h_{t-1} y la entrada en el actual, x_t. Fijémonos que ahí es donde aparece el valor asociado a la puerta de reinicio en el instante de tiempo t, que como decíamos y ahora podemos ver de forma nítida en el vector concatenado, determinará cuánta información del pasado prevalecerá. Valores próximos a uno harán que el producto con h_{t-1} sea más parecido a este último, mientras que valores próximos a cero harán que este desaparezca. Después, el vector concatenado multiplicado por la matriz de pesos se pasará por la función tangente hiperbólica para que los valores se compriman en el intervalo $[-1, 1]$, introduciendo una no linealidad en el modelo y evitando así problemas de inestabilidad numérica.

Finalmente necesitamos calcular el estado oculto, h_t, que pasaremos al siguiente módulo, tanto hacia arriba si procede como al del siguiente instante de tiempo. En este punto es cuando se necesita la puerta de actualización, ya que determinará cuánto tomar de nuestro estado candidato, que representa la información recopilada del instante actual y cuánto del contenido de pasos previos. Para entender la forma en la que esto se logra, veamos la siguiente ecuación.

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t.$$

Vemos que para la obtención del estado oculto final, h_t, se emplea tanto h_{t-1}, que es el estado oculto del paso previo y contiene información sobre lo que la red ha aprendido hasta el momento, como $\tilde{h}_t$, que es nuestro estado candidato para el instante de tiempo actual y representa la nueva información que podría potencialmente actualizar el resultado final. A la hora de sumarlos han sido ponderados empleando la salida de la puerta de actualización. El primero se ha multiplicado por $(1 - z_t)$, de forma que, cuando el vector asociado a la puerta de actualización esté próximo a cero, no se producirá prácticamente actualización en el estado oculto, prevalenciendo una mayor parte del del instante anterior. A su vez, nuestro estado candidato va multiplicado por z_t por lo que valores próximos a la unidad del mismo indicarán una actualización severa en el resultado final. Como hemos podido ver, el ajuste de z_t determinará cómo de suave será la transición entre conservar la información pasada e incorporar la nueva. Este balance es el que ayuda a GRU a retener contexto histórico relevante, a medida que se adapta a la nueva información que se le proporciona, convirtiéndola en una opción muy efectiva para el manejo de datos secuenciales.

Una vez revisado el contenido teórico, estamos en condiciones de emplearlo para estudiar conjuntos de datos de interés. Esto será lo que podremos encontrar en el Capítulo 4, en el que tomaremos los modelos vistos a lo largo del documento y veremos qué tal rinden en la práctica, comentando tanto sus fortalezas, como los puntos en los que parecen presentar más debilidades.

Capítulo 4

Aplicación práctica en Python

Comienza el Capítulo 4, en el que pondremos a prueba los modelos que hemos estudiado en los capítulos previos mediante su aplicación a datos reales. Los datos empleados, aunque de dominio público, fueron proporcionados por el Instituto Galego de Estatística, y la elección de los mismos vino motivada precisamente por las costumbres o necesidades dentro de la propia institución. Por otra parte, se ha empleado el lenguaje de programación Python para el modelado y predicción de las series de tiempo. El motivo es que este lenguaje ofrecía un abanico más amplio de posibilidades que otros que también hubieran podido cumplir las funciones deseadas, como por ejemplo R. Además de esto último, que fue la motivación principal, esta se consideró una oportunidad inmejorable para desarrollar habilidades y aprender, no solo de series de tiempo, sino también del mencionado lenguaje.

Dividiremos el Capítulo 4 en dos secciones, una por cada conjunto de datos utilizado. La Sección 4.1 estará dedicada a un conjunto que recoge las exportaciones totales de España a siete países distintos, cuatro que tienen el euro como moneda (Francia, Alemania, Portugal e Italia) y tres que no, procurando que haya variedad en continentes y distancias (Reino Unido, Estados Unidos y Japón). Primero ajustaremos datos con una frecuencia mensual, para después agregarlos trimestralmente y realizar el estudio correspondiente. De esta forma, podemos analizar las series combinándolas con la abundante información económica de carácter trimestral, como las cuentas económicas trimestrales que elaboran los Institutos de Estadística. Después, en la Sección 4.2 tomaremos los datos de demanda eléctrica de España (contando con las islas) y ajustaremos datos esta vez con una frecuencia de cinco minutos.

La estructura de ambas secciones será similar. Primero se dará un contexto sobre el conjunto que se va a estudiar y se especificarán las variables exógenas escogidas para este fin. Después nos centraremos en los datos, explicando su procedencia. Una vez hecho esto, se ahondará en la metodología de trabajo empleada para el ajuste de los respectivos conjuntos de datos. En este punto se especificarán también los modelos seleccionados para el ajuste de los datos, así como los parámetros asociados a cada uno de ellos. Finalmente se expondrán los resultados obtenidos, apoyando el análisis con representaciones gráficas que lo respalden.

Antes de continuar, solamente queda añadir que si el lector desea comprobar los archivos de Python mediante los cuales se ha llegado a estos resultados, los podrá encontrar en SuarezSoto (2026), junto con los conjuntos de datos empleados para ello.

4.1. Datos de Exportaciones

Comenzamos trabajando con los datos de comercio exterior relativos a exportaciones. Este estudio, como indicábamos en la introducción, consistirá en analizar las series de tiempo de las exportaciones desde España a siete países distintos. Cuatro de ellos, tienen el euro como moneda oficial, que son

Francia, Portugal, Italia y Alemania, y tres de ellos no, que serán Japón, Estados Unidos y Reino Unido. Dado que España exporta una gran variedad de productos a estos países, se ha realizado una agregación, de forma que la variable de interés contiene la totalidad de las exportaciones, medida en euros.

Además se ha añadido el efecto de ciertas variables exógenas para enriquecer el modelo y tratar de explicar mejor el comportamiento de nuestras siete series. Se ha considerado oportuna la inclusión de el PIB de cada uno de los países, el tipo de cambio de sus respectivas monedas al euro (siendo este 1 en el caso de Francia, Portugal, Italia y Alemania) y el arancel promedio de todos los países para los productos exportados desde España (siendo cero en todos los casos salvo Estados Unidos y Japón). Además, dado el gran cambio que ha tenido lugar en los aranceles impuestos por Estados Unidos, se ha considerado oportuna la inclusión de una variable a la que hemos llamado "$tariff\_shock$" y cuya función es simplemente indicar cuando un arancel es superior a 5, que solamente se cumple para Estados Unidos en el año 2025. Además, se han añadido dos variables exógenas adicionales para todos los modelos, salvo para ARIMA. Estas variables son indicadoras de ciclo y sirven para indicar que después del último mes/trimestre viene el primero. Para esto se han empleado el seno y el coseno de la forma siguiente:

$$\begin{aligned} \text{Trimestral:} \quad & \text{trim\_sin} = \sin\left(\frac{2\pi(q-1)}{4}\right) \quad \text{trim\_cos} = \cos\left(\frac{2\pi(q-1)}{4}\right), \\ \text{Mensual:} \quad & \text{mes\_sin} = \sin\left(\frac{2\pi(m-1)}{12}\right) \quad \text{mes\_cos} = \cos\left(\frac{2\pi(m-1)}{12}\right), \end{aligned}$$

donde q es el trimestre del 1 al 4 y m el mes del 1 al 12.

Por último, como podremos ver a continuación los resultados no son tan precisos como cabría esperar si solo nos fijamos en los aspectos matemáticos. La predicción de las series que estamos considerando es un objetivo realmente ambicioso, ya que el periodo temporal considerado abarca, entre otras cosas, una crisis financiera en Estados Unidos en el año 2007, la crisis del euro en el año 2009/2010 y una pandemia mundial en el año 2020. Por tanto, dada la naturaleza de los datos, se puede argumentar que las series poseen mucho ruido. Es decir, la componente impredecible de las mismas tiene un gran peso, dificultando la tarea de predicción. Por este motivo, se realizó un análisis posterior empleando los programas TRAMO y SEATS (Gómez y Maravall 1996), que dan lugar al software TRAMO-SEATS. Como podemos ver en Wikipedia contributors (2022), este software, recomendado por instituciones de prestigio, utiliza métodos de ajuste estacional paramétricos, basados en modelos ARIMA estacionales y en métodos de análisis espectral. Gracias a este programa, empleado por los profesionales del Instituto Galego de Estatística para la desestacionalización de series de tiempo, podremos descomponer nuestras series, comprobando así, entre otras cosas, si la calidad de los resultados está relacionada con la presencia de ruido.

4.1.1. Los datos

Lo primero que debemos indicar es que las series con las que estamos trabajando son las series brutas, sin desestacionalizar en un espacio de tiempo que abarca desde el año 2002, fecha de entrada en vigor del euro, hasta el año 2025. La diferencia principal entre estas y las desestacionalizadas radica en el resultado que se desea ver. Las series originales, con las que estamos trabajando, muestran los datos tal cual suceden en el mundo real y contienen, por ejemplo, variaciones que se repiten periódicamente. Por otro lado, a las series desestacionalizadas se les aplica un filtro estadístico para eliminar esos efectos cíclicos predecibles. El objetivo es limpiar la serie para observar la tendencia subyacente.

Para la obtención de los datos se buscó en determinadas fuentes conocidas por su fiabilidad y amplia oferta. Los datos de exportaciones se han tomado a través de DataComex (Secretaría de Estado de

Comercio 2026), herramienta web de la secretaría de estado de comercio. Es decir, proviene de los registros administrativos del "Departamento de Aduanas e Impuestos Especiales" de la "Agencia Tributaria del Estado", que registran mensualmente las operaciones de comercio exterior de mercancías que tienen lugar en España. Este registro administrativo, en lo que respecta al comercio con terceros países, se basa en la Declaración Única de Aduanas (DUA o Documento Único Administrativo). En cuanto al comercio intracomunitario, desde 1993 no existen fronteras entre los países que componen la Unión Europea, ni, por lo tanto, trámites aduaneros. Por lo tanto, en general, los operadores económicos están obligados a cumplimentar la declaración estadística Intrastat, que registra las correspondientes transacciones intracomunitarias. Esta declaración constituye la base para obtener datos estadísticos sobre el comercio entre Estados miembros. En concreto, el flujo utilizado ha sido: Exportaciones. Comprenden todas aquellas operaciones mediante las cuales un producto originario de Galicia (España) se vende a países que no forman parte de la UE. Para las definiciones de las zonas, se utiliza la geonomenclatura desarrollada por Eurostat (Instituto Galego de Estatística (IGE) 2026).

La variable exógena arancel promedio del resto de países a las exportaciones españolas se obtuvo del World Integrated Trade Solution (WITS) (World Integrated Trade Solution (WITS) 2026). El Banco Mundial —en colaboración con la Conferencia de las Naciones Unidas sobre Comercio y Desarrollo (UNCTAD) y en consulta con entidades, tales como el Centro de Comercio Internacional, la División de Estadística de las Naciones Unidas (UNSD) y la Organización Mundial del Comercio (OMC)— desarrollaron el programa informático "Solución Comercial Integrada Mundial" (WITS, por sus siglas en inglés). Este permite a los usuarios obtener acceso y recuperar información sobre comercio y aranceles. A continuación, se presenta el organismo internacional que compilan los datos recogidos: El Sistema de Análisis e Información Comercial (TRAINS) de la UNCTAD contiene información sobre importaciones, aranceles y medidas no arancelarias correspondientes. Los datos sobre aranceles y medidas no arancelarias están disponibles en el nivel más detallado de aranceles nacionales de los productos básicos (esto es, a nivel de líneas arancelarias), para más de 170 países y un nivel de desagregación de productos elevado. Los datos se registran según tres clasificaciones comerciales y arancelarias reconocidas internacionalmente.

Cabe destacar, como decíamos anteriormente, que de esta página solamente se obtuvieron los datos para Estados Unidos y Japón, ya que en el caso del resto de países es cero, incluso para Reino Unido después del Brexit, ver UK Government (2020). Por otra parte, en el World Integrated Trade Solution solo se pudieron encontrar los datos hasta el año 2023, por lo que nos seguían haciendo falta los correspondientes a los años 2024 y 2025. En el caso de Japón, nos pudimos permitir replicar el arancel promedio del año 2023 para los años siguientes también, dado que el cambio se espera mínimo y el arancel en sí mismo no es demasiado elevado. El problema llegaba con Estados Unidos, dados los grandes cambios que han tenido lugar desde la llegada del nuevo presidente. Para ello, nos basamos en las estimaciones de Cámara de Comercio de España (2025) para obtener los datos necesarios.

Por último, falta mencionar la obtención de las variables de PIB por país y tipo de cambio. En este caso el proceso de obtención fue más rápido. Se obtuvieron ambos conjuntos de datos, para los años demandados del Fondo Monetario Internacional (Fondo Monetario Internacional (FMI) 2026), que ofrece una gran variedad de estadísticas alineadas con sus principales objetivos, como la cooperación monetaria internacional o comercio internacional.

4.1.2. Metodología

En esta subsección revisaremos la metodología empleada para el estudio de las siete series. Antes de comenzar, cabe destacar que para este conjunto de datos se han probado dos enfoques. El primero, mensual y el segundo trimestral. Presentaremos los resultados de ambos, pero estos no poseen diferencias metodológicas más allá de un ajuste adicional en el apartado de lectura de datos, o cambiar la frecuencia en los puntos en los que hubo que especificarla. Algunas de las series de datos trabajadas venían en

formato mensual, otras en formato trimestral y otras en formato anual. Para adaptarlas se optó por la medida más oportuna en cada momento. Como ejemplo podemos poner el caso del arancel promedio que es un dato anual. En este caso se replicó para todos los meses/trimestres del año asociado.

Como comentamos anteriormente, el primer modelo ajustado será un modelo ARIMAX (la "X" al final proviene de que se añadirán las variables exógenas). Este modelo nos servirá como sparring, es decir, como modelo con el que compararemos los resultados y esperaremos mejorar. Para el ajuste de este modelo a las siete series se definirá una función que transforme los datos, los divida en entrenamiento-test y busque el modelo óptimo mediante la función *auto_arima*. Además, esta función se encargará de la búsqueda de datos atípicos, diferenciándolos entre aditivos, cuyo efecto deberá ser incorporado al modelo, e innovativos. Después, realizará también la validación, empleando los tests de Ljung-Box y media cero, así como la comprobación de la normalidad, aplicando el test de Jarque-Bera. Finalmente, continuará con la última etapa, en la que realizará predicciones sobre el conjunto de test.

Una dificultad que se presentó en este aspecto fue que algunos de los modelos obtenidos para ciertos países no superaban, llegado el momento, el test de Ljung-Box. Dado que se buscaba la automatización total del ajuste de los modelos, se descartó llevar a cabo una búsqueda individual de los modelos más adecuados para cada país en base a su ACF y PACF, y se creó otra función que probaba todas las combinaciones de modelos en un rango prefijado de los parámetros, aplicaba el test de Ljung-Box y seleccionaba, de entre los que lo superaban, el que daba lugar al mejor en términos de AIC. De esta forma aumentó significativamente el tiempo de computación, pero obtuvimos a cambio un código completamente automático.

Por su parte, el enfoque que siguen los modelos de machine learning tradicionales es distinto al visto para el modelo ARIMA. En este, se produce una transformación de serie de tiempo a problema de regresión estándar, y entra en luego el concepto de *lag*. Primero se lleva a cabo una transformación tabular. Para ello la función empleada, *ForecasterRecursiveMultiseries*, toma todas las series temporales de los distintos países y las apila. Ahora, dado que el modelo necesita identificar a qué país pertenece cada registro, añadimos la opción *encoding = 'ordinal'*, que añade una columna para este fin. Así, como se ilustra en la Figura 4.1 varias series se incorporan al proceso y se convierten en una sola matriz global, ajustando así un modelo único que aprende el comportamiento conjunto de todos los países.

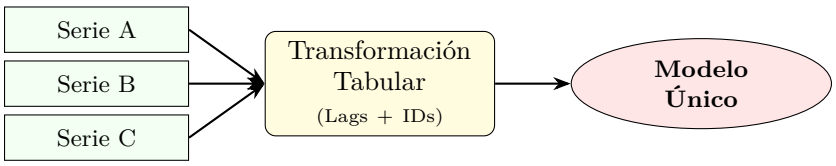


Figura 4.1: Diagrama que ilustra la transformación tabular que se lleva a cabo en el ajuste de los modelos.

Una vez estructurados los datos, llega el momento de la construcción de la matriz de variables independientes, X, y de la variable objetivo y. En este punto entra en juego el concepto de *lag*. Este representa el valor de la serie en el instante de tiempo anterior. Podemos apreciar en la Figura 4.2 que el modelo crea internamente una tabla, donde cada fila es un momento en el tiempo, mediante una ventana deslizante, a la que hicimos referencia cuando hablamos del rolling forecasting en la Sección 1.5 del Capítulo 1.

El número de columnas vendrá determinada por el número de lags que fijemos. En nuestro caso, atendiendo futuros intereses que podría tener el propio IGE, fijaremos *lags = 48* para el estudio mensual y *lags = 16* para el trimestral, lo que significa que en ambos casos estamos empleando cuatro años de historia previa para predecir el próximo (ya que predecimos a horizonte 12 en el caso mensual y 4 en el

trimestral). Además de esto, gracias a la función *RollingFeatures* añadimos una información extra, proporcionando al modelo tanto la media como la desviación típica en ventanas de 12 y 24 meses (caso mensual) y 4 y 8 trimestres (caso trimestral) para que cuente con esta información adicional de uno y dos años atrás.

Mencionamos una característica muy interesante, que además limita hasta cierto punto a estos modelos, y es que para ellos la columna asociada al primer lag y al último son variables independientes. No entiende que una va estrictamente después de la otra. Solamente busca correlaciones estadísticas no lineales entre las columnas y la variable objetivo de la misma fila.

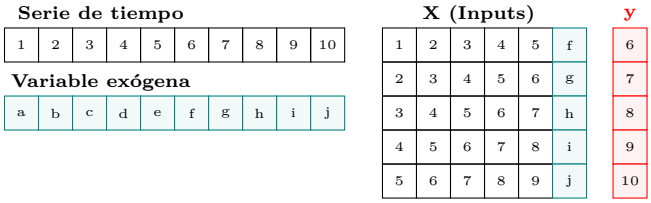


Figura 4.2: Diagrama que ilustra los lags junto con el rolling forecast.

Después tenemos el ajuste de los modelos LSTM y GRU. Estos llegan para suplir en cierto modo la problemática que nos surgía al tratar los *lags* como variables independientes sin una relación de orden estricta, ya que introducen arquitecturas de deep learning específicamente diseñadas para datos secuenciales.

A pesar de que se sigue utilizando $ForecasterRecursiveMultiseries$ para la preparación de los datos de forma tabular y multiserie, mediante la clase $RegresionRNN$ se define una forma de procesar los datos en el interior totalmente distinta. Primero tiene lugar la separación de variables, en la que, cuando la matriz entra a la red neuronal, el modelo separa los datos históricos (los *lags*) de las variables exógenas.

Después se lleva a cabo la restauración del orden temporal. A diferencia de lo que nos ocurría con random forest o XGBoost, las redes neuronales recurrentes, como vimos, necesitan leer la historia en su orden cronológico para actualizar su memoria interna. Para ello, el tensor de *lags* se invierte (espacialmente) para que la red procese los datos desde el momento más antiguo al más reciente.

Posteriormente llega el momento del procesamiento propiamente dicho y la combinación con las variables exógenas. Las capas de las redes neuronales recurrentes seleccionadas procesan la secuencia paso a paso, extrayendo los patrones complejos que consideren. El resultado del último paso temporal, que es el que más nos interesa, se concatena con las variables exógenas del momento actual, y es esta combinación de memoria y contexto actual obtenido con las exógenas la que pasa por una última capa lineal que emite la predicción. Por último, las predicciones realizadas en el conjunto de test serán las que se evalúen en base a las métricas expuestas en la Sección 1.4 del Capítulo 1.

Antes de continuar con la próxima subsección, en la que analizaremos los resultados obtenidos en base a la metodología expuesta, se debe indicar la configuración escogida para cada modelo de machine learning y deep learning (recordemos que ya se ha especificado la forma en la que se configura el modelo ARIMA), para que haya una completa transparencia cuando se visualicen los resultados. En el caso de los modelos random forest y XGBoost, la configuración no cambiará con la frecuencia de los datos, es decir, ambos poseerán la misma configuración para el caso mensual y trimestral. En el caso de random forest esta fue: 200 árboles a combinar, con una profundidad máxima de 5 y que un nodo terminal ha de tener como mínimo 8 datos, para evitar una gran especialización en las hojas y de esta forma

tratar de paliar el sobreajuste. Por su parte, en el modelo XGBoost combinaremos 500 árboles con una profundidad máxima de 3 y una tasa de aprendizaje de 0.01. Además, hemos añadido una serie de medidas para tratar de evitar el sobreajuste también en este modelo. Hemos indicado que para entrenar cada árbol solamente se puede emplear el 80 % de las filas y de las columnas, además de añadirle una penalización por complejidad.

En lo referente a las redes neuronales recurrentes, sí que hay diferencias en cuanto a la configuración para datos mensuales y trimestrales. Recordemos que estos modelos de deep learning son modelos hiperparametrizados, por lo que hay que tener un especial cuidado de no sobreajustar, sobre todo en el caso de no contar con muestras muy grandes. Por este motivo, en el caso mensual, para el que se dispone de más datos, hemos optado por una configuración de 64 unidades ocultas, dos capas apiladas y 50 *epoch*. Además, pensando en un posible sobreajuste hemos añadido un *dropout* de 0.3, cumpliendo la función que se explicó en el Capítulo 3. Por su parte, el modelo trimestral se ha configurado con 16 unidades ocultas, dos capas apiladas y 75 *epoch*, además de añadirle también el *dropout* de 0.3. Finalmente, cabe destacar que se probaron otras configuraciones, tanto con los modelos basados en árboles como con las RNN, para ver el impacto que estas tenían sobre los mismos. A pesar de no producir cambios demasiado significativos, tomamos las configuraciones que daban lugar a errores más bajos.

Una última indicación que merece la pena comentar antes de pasar a ver los resultados es que tanto la serie de tiempo como las variables exógenas se han reescalado para que los resultados no se viesen afectados por la diferencia de escala y magnitud de los valores. Como ejemplo de esto último pensemos en la diferencia que hay en estos términos entre las variables asociadas al PIB y al tipo de cambio. Además, se ha comprobado que los resultados al trabajar con los datos diferenciados en lugar de crudos da lugar a mejores resultados, por lo que es la configuración que se ha aplicado.

4.1.3. Resultados

En esta subsección analizaremos los resultados obtenidos en base a la aplicación de la metodología vista en la subsección previa y los datos expuestos. Se acompañará este análisis con imágenes que respalden los resultados numéricos y faciliten la visualización de los mismos. Lo primero que debemos hacer cuando trabajamos con un conjunto de datos nuevo es familiarizarnos con el mismo, por lo que a continuación graficaremos las siete series de datos, tanto para el caso mensual como para el caso trimestral. Se pueden encontrar en la Figura 4.3.

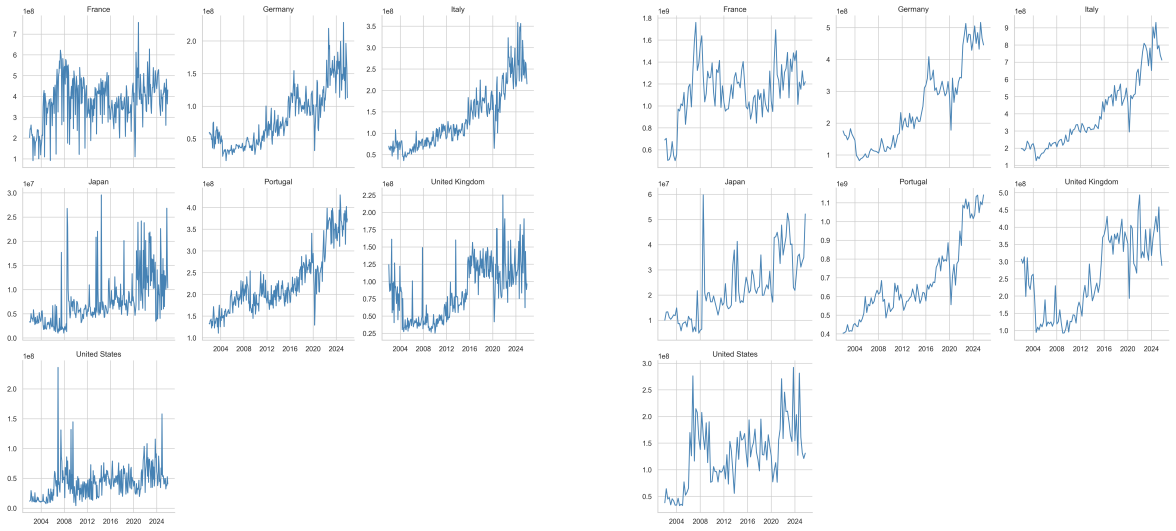


Figura 4.3: Representación gráfica de las series de tiempo correspondientes a exportaciones. En el eje X se muestra la evolución temporal (meses en el caso de la de la izquierda y trimestres en el caso de la de la derecha), y en el eje Y se muestran las exportaciones desde España al país en cuestión, medidas en euros.

Como decíamos, la Figura 4.3 nos permite ver ciertas características iniciales de cada serie. Notemos que la rejilla de puntos es más poblada en la gráfica de la izquierda, es decir, para el caso mensual ya que, como dijimos, contamos con más datos que en el caso trimestral, representado a la derecha. Por otra parte, se puede apreciar una cierta tendencia en todas las series, lo que explica y justifica el uso de la diferenciación que mencionamos previamente. Otro motivo para llevar a cabo la diferenciación es tratar el paliar el problema de random forest, XGBoost, y en menor medida de las redes neuronales recurrentes, con la extrapolación de la tendencia. En base a lo que hemos expuesto en las secciones teóricas se puede deducir que los modelos mencionados no serán capaces de predecir valores más altos que los que han visto durante el entrenamiento. En el caso de los modelos basados en árboles esto ocurre debido a que cada predicción viene a ser un promedio de las observaciones que estos han visto durante el entrenamiento, mientras que las redes neuronales recurrentes encuentran esta limitación en sus funciones de activación (la mencionada tangente hiperbólica), que al estar acotadas en un rango específico, tienden a saturarse cuando reciben valores más altos que los vistos en el entrenamiento. Estas últimas, como decíamos sufren menos este efecto ya que poseen memoria, que les permite capturar y proyectar la inercia de la serie. La diferenciación de la serie nos ayuda en este aspecto ya que, como se dijó en la Sección 1.2 del Capítulo 1, los modelos, en lugar de aprender los valores, aprenden variaciones dentro de la misma, evitando el efecto de techo que acabamos de mencionar.

Una vez realizado el primer juicio visual, ajustamos los modelos correspondientes pudiendo encontrar los resultados recogidos en la Tabla A.1 y en la Tabla A.2 de la Sección A.1 del Apéndice A. Estas tablas incluirán los resultados mensuales y trimestrales, respectivamente, asociados a todos los países y modelos, pero solamente para cuatro medidas seleccionadas como las más interesantes, que son el MAE, RMSE y MedAE como medias absolutas y WAPE como medida relativa. Si el lector desea comprobar los resultados de las métricas restantes, presentadas en la Sección 1.4 del Capítulo 1, se encuentran en la Tabla A.4 y en la Tabla A.5 en la Sección A.2 del Apéndice A. Interpretemos los resultados que podemos encontrar en las mismas.

Por un lado, lo más evidente parece que el modelo estadístico tradicional, el ARIMA, no ha sido capaz de capturar la complejidad de las series de tiempo proporcionadas en la mayoría de los casos, siendo

superado con creces por los modelos de machine y deep learning. Esto no es algo que nos coja por sorpresa, ya que desde un primer momento lo pensamos como un sparring para estos últimos. ARIMA muestra problemas graves en el ajuste de Francia, Reino Unido y Estados Unidos, especialmente. Fijémonos, por ejemplo, en el caso de Estados Unidos con datos mensuales: ARIMA tiene un WAPE de 993.91 %, mientras que un modelo de machine learning al uso como puede ser XGBoost lo reduce drásticamente a un 36.26 %. Esto evidencia que las series con las que se está trabajando presentan no linealidades o patrones que el modelo lineal no está siendo capaz de capturar. Los modelos de machine learning al uso, por su parte, se presentan como opciones muy sólidas. XGBoost y random forest muestran un rendimiento muy similar, superándose mutuamente dependiendo del país (a nivel mensual, XGBoost domina en Francia y Reino Unido, mientras que random forest lo hace en Alemania y Japón), aunque con métricas competitivas en general. Se puede comprobar su rendimiento mensual en la Figura 4.4.

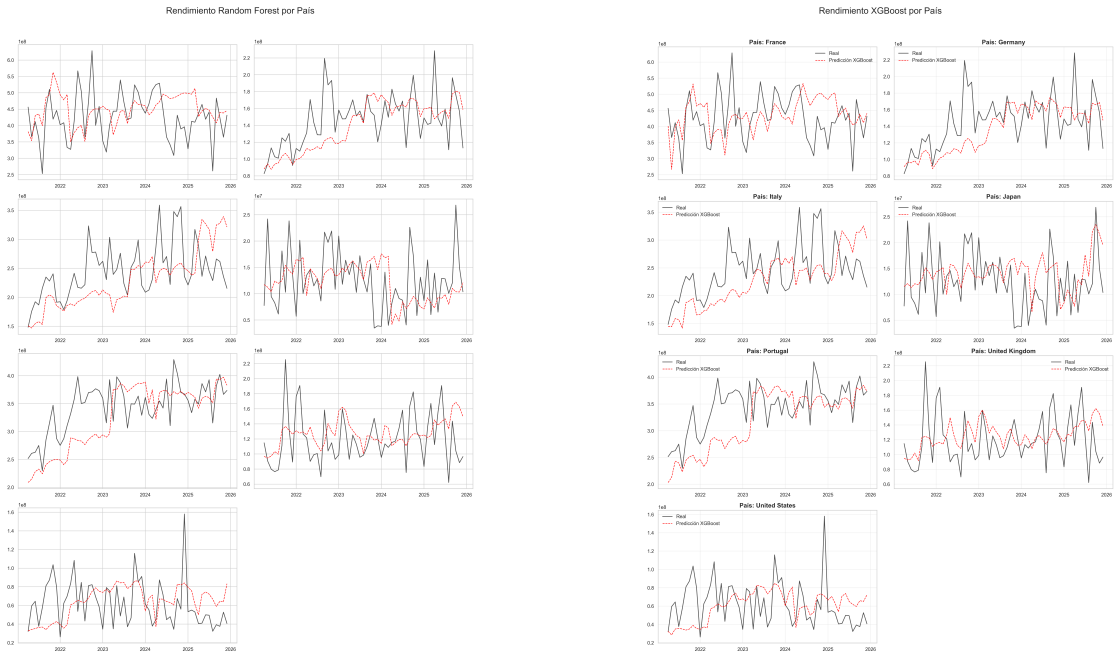


Figura 4.4: De izquierda a derecha: Predicciones con el modelo random forest y con el modelo XGBoost realizadas sobre el conjunto de test para los 7 países y frecuencia mensual.

A pesar de los buenos resultados logrados por ambos, destacando por ejemplo que random forest logró bajar el WAPE trimestral de Portugal hasta un 8.93 %, los modelos de deep learning han mostrado su robustez de forma global, aunque con matices interesantes, dado que tenemos dos situaciones enfrentadas. Por un lado, no son bases de datos muy extensas, algo que a priori es necesario para el correcto funcionamiento de las redes neuronales, pero dado que contamos con siete series, estos modelos pueden brillar un poco más, ofreciendo un rendimiento superior. De los modelos de deep learning podemos destacar por ejemplo que en el caso de los datos mensuales de Estados Unidos, que se podría decir que ha sido el país más complejo en conjunto, LSTM logra el mejor WAPE, reduciéndolo hasta 32,90 %. En la Figura 4.5 podemos encontrar una representación que capta el rendimiento de estos modelos tanto para el caso mensual como para el caso trimestral. No obstante, debemos destacar también que en países como Francia o Alemania con frecuencia mensual, los modelos basados en árboles han logrado superar mínimamente a las redes neuronales, mostrando una competencia ajustada entre los modelos.

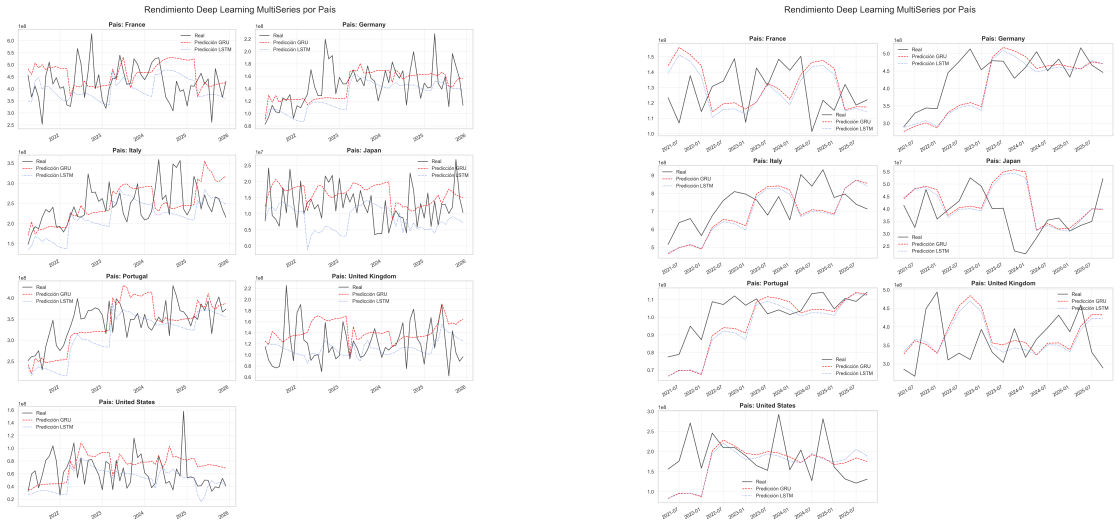


Figura 4.5: De izquierda a derecha: Predicciones con redes neuronales recurrentes (LSTM y GRU) realizadas sobre el conjunto de test para los 7 países para el caso de frecuencias mensuales y trimestrales.

Por otra parte, es interesante destacar que, cuando pasamos de datos mensuales a datos trimestrales, los errores absolutos aumentan, porque estamos prediciendo cantidades agregadas mayores (se juntan tres meses de datos). Sin embargo, como se puede intuir en la Figura 4.5, la clave está en el valor del WAPE, que vemos que disminuye en la mayoría de los casos, indicando que suele ser más fácil predecir el trimestre que el mes. Esto se puede deber a un suavizado de la varianza, ya que al agregar los datos estamos compensando la volatilidad mensual. Destacamos el caso de Japón como ejemplo de esto y el de Reino Unido como contrajemplo. Por un lado, todos los modelos sufren para la predicción mensual de Japón, con valores del WAPE que rondan o superan el 40 % (salvo el modelo ARIMA, que curiosamente ha rendido ligeramente mejor a nivel mensual que a nivel trimestral), pero vemos que a nivel trimestral LSTM logra disminuir este valor a un 23,94 %. Además, tenemos los casos de países como Alemania y Portugal, que mejoran el WAPE a nivel trimestral en cada modelo, aunque no con tanta holgura como Japón, salvo en la excepción mencionada. Por otra parte, vemos que en el caso de Reino Unido, modelos como random forest o XGBoost empeoran sus predicciones en términos de WAPE a nivel trimestral, indicando que quizá hay patrones dentro de los trimestres (mes a mes) de Reino Unido que las redes neuronales y árboles están siendo capaces de explotar. Por último, cabe mencionar el caso del modelo ARIMA para Estados Unidos. Como decíamos antes, a nivel mensual llega a un WAPE nada satisfactorio del 993.91 %, pero a nivel trimestral lo empeora significativamente, obteniendo el WAPE catastrófico de $2,852.31$ %. Ilustramos esta mejoría general en la Figura 4.6 en contraposición a lo visto en la Figura 4.4.

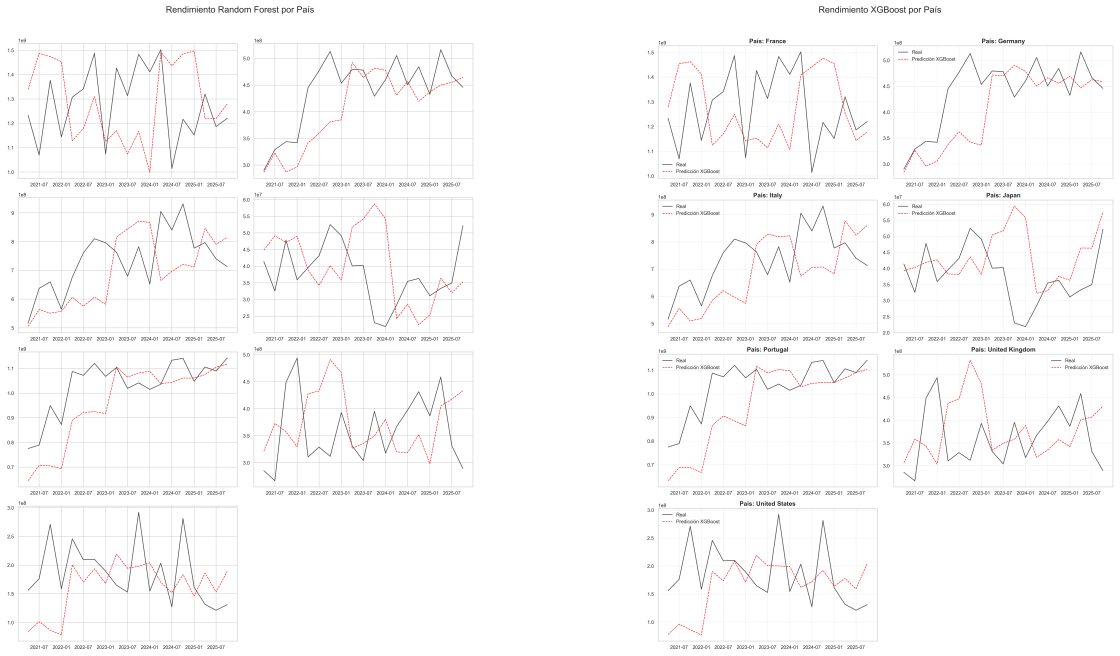


Figura 4.6: De izquierda a derecha: Predicciones con el modelo random forest y con el modelo XGBoost realizadas sobre el conjunto de test para los 7 países y frecuencia trimestral.

Centrándonos ahora en los países, particularmente en la facilidad de predicción de los mismos, podríamos decir que los que mejor se han predicho han sido Alemania y Portugal, que se presentan como los países más estables. Para Portugal, random forest logra el WAPE más bajo de todo el estudio, un 8.93 % en el caso trimestral, error más que satisfactorio. Para Alemania, este mismo modelo logra un WAPE también muy positivo de 10.53 %. Por otra parte, tal y como esperábamos antes de comenzar el análisis dada la lejanía de estos países y el escaso volumen exportado a los mismos, los más difíciles de predecir han sido Japón y Estados Unidos. Para este último, ningún modelo ha logrado bajar del 28 % en términos de WAPE en el caso trimestral y 32 % en el caso mensual. Desde un punto de vista objetivo, tampoco resultan ser demasiado negativos dada, como decíamos, la situación geográfica del mismo y el volumen de comercio. Si ahora nos fijamos en la predicción mensual de las exportaciones a Japón podremos ver una anomalía interesante. Como comentamos previamente, todos los modelos sufren para la predicción de Japón y curiosamente es ARIMA el que logra reducir el error al mínimo para esta frecuencia y país, con un valor del WAPE de 34.82 %, batiendo con holgura a modelos más complejos como las redes neuronales recurrentes, LSTM (52.94) y GRU (47.53). Algo similar ocurre con Alemania, y es que en este caso el WAPE asociado a ARIMA a nivel trimestral (16.47 %), si bien no mejora el asociado a modelos más complejos como GRU (12.11 %) o LSTM (11.50 %), vemos que se encuentra sorprendentemente cerca. Estos dos ejemplos sugieren que el algoritmo de deep learning no encuentra patrones diferenciales para estas frecuencias y países, siendo el modelo lineal una opción relativamente competitiva para la predicción de los mismos.

En el párrafo anterior hemos visto que algunos países, como Alemania y Portugal, resultan más fáciles de predecir que otros como Estados Unidos o Japón. Se puede achacar, como se decía en el apartado previo, a la lejanía de los mismos en relación a España, o al menor volumen de exportación que hay con los países en cuestión. De todas formas, como dijimos en la introducción, el ruido de las series puede estar jugando un papel importante en este problema, más aún tratándose de periodos de tiempo con eventos tan señalados, en los que la volatilidad está muy presente. Por este motivo, una vez llevada a cabo la descomposición de las series empleando el programa TRAMO-SEATS, estamos en condiciones

de exponer gráficas en las que se ha aislado la componente irregular e impredecible asociada al ruido de cada país. Comenzaremos exponiendo las gráficas para los dos países más difíciles de predecir, que como decíamos son Estados Unidos y Japón. Después, mostraremos los asociados a Francia y Reino Unido, que han estado un escalón por debajo en términos de dificultad, para finalmente exponer los de Alemania, Portugal e Italia, que se han situado globalmente como los países más fáciles de predecir.

Antes de continuar y mostrar las gráficas, se debe mencionar una última indicación en relación a las escalas de las mismas, esencial para su correcta comprensión. Como veremos, las escalas asociadas a las gráficas de todos los países se mueven en intervalos relativamente acotados, salvo Francia, que llega a tomar valores tremendamente altos. La explicación de esto está en el funcionamiento interno del programa TRAMO-SEATS. Este es el que decide si la serie es aditiva, y la descompone como *tendencia + estacionalidad + ruido* o si es multiplicativa y la descompone como *tendencia* $\cdot$ *estacionalidad* $\cdot$ *ruido*. En este último caso, emplea la transformación logarítmica dado que el logaritmo del producto es el sumatorio de logaritmos. Por lo tanto, cuando se tiene un modelo multiplicativo, el valor de referencia es el 1, valores por encima del mismo indican un *shock* positivo y valores por debajo un *shock* negativo. Por ejemplo, un valor de 1.5 indica que se exportó un 50 % más de lo predicho por las componentes regulares (tendencia y estacionalidad). Por otra parte, cuando se tiene un modelo aditivo como es el caso de Francia, el valor de referencia viene representado por el 0, y la gráfica se mueve en valores absolutos. Es decir un valor de X, implica que se exportaron X euros más de lo predicho por las componentes regulares. Veamos entonces las gráficas.

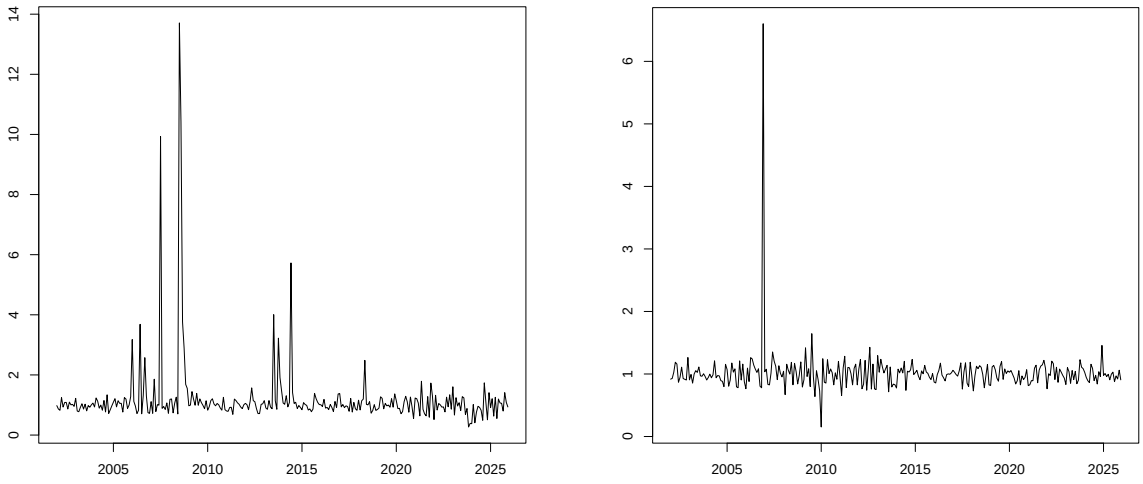


Figura 4.7: De izquierda a derecha: Componente irregular asociada a las series de exportaciones de Japón y Estados Unidos.

La Figura 4.7 nos muestra el ruido asociado a los dos países que han resultado más difíciles de predecir. De hecho, gracias a estas gráficas podemos, en efecto, deducir el por qué. Por un lado vemos que Japón muestra unas anomalías muy fuertes, llegando a multiplicar su valor por 14 en torno a 2008, seguido de otros picos muy fuertes. Esta volatilidad tan prominente, además de recurrente, hace que el modelo aprenda que pueden ocurrir sustos grandes e cualquier momento. Estados Unidos, a pesar de parecer más estable en torno al valor de referencia, muestra también un pico muy fuerte en torno a 2007, superando el valor 6 en la escala. Este dato extremo puede hacer que los modelos tiendan a ser conservadores, y por lo tanto inexactos. Resulta interesante ver que en el caso de estos dos países, los shocks más grandes se encuentran en torno a 2007, periodo en el que tuvo lugar la gran crisis financiera, pero apenas parece notarse el impacto del COVID-19 en el año 2020.

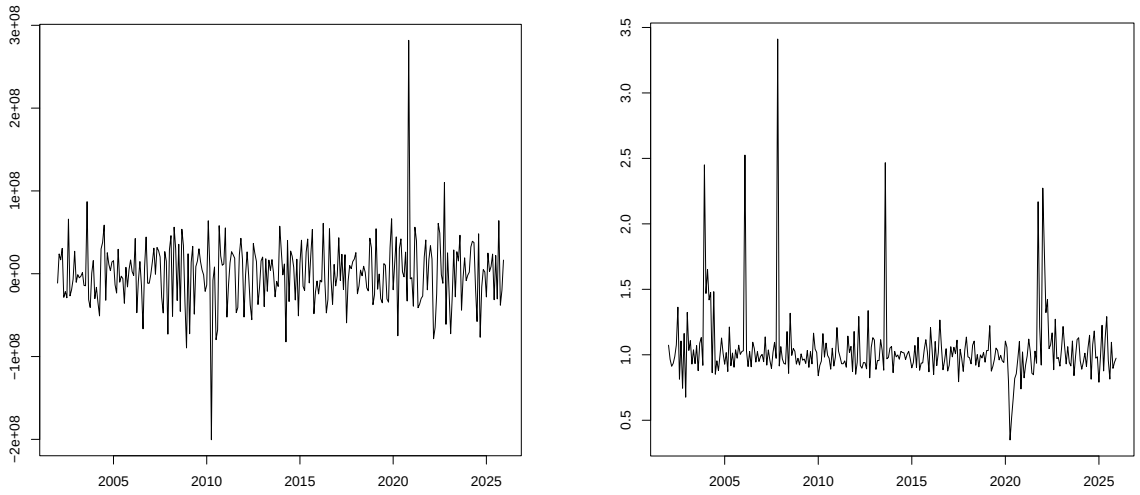


Figura 4.8: De izquierda a derecha: Componente irregular asociada a las series de exportaciones de Francia y Reino Unido.

Continuamos con la Figura 4.8, en la que encontramos las gráficas asociadas a Reino Unido y Francia, países que podríamos situar en el siguiente escalón en cuanto a la dificultad asociada a su predicción. En el caso de Reino Unido vemos que la dificultad reside no tanto en el tamaño de los *shocks* puntuales, sino en la cantidad de *shocks* moderados-altos. Ahora, en la gráfica de Reino Unido, sí que se puede ver la caída en torno al año 2020, que como decíamos está asociada al COVID-19, pero también podemos ver una rápida recuperación positiva, que quizá podría achacarse al *Brexit*. Dada la volatilidad de la serie, el modelo aprende esta característica, haciendo que las predicciones sean más imprecisas. En cuanto a Francia, vemos muchas oscilaciones significativas, destacando dos picos grandes, uno negativo en torno al 2010, y otro en torno al 2020, pero que sorprendentemente es positivo. Estos cambios de dirección tan fuertes y bruscos dificultan que el modelo pueda recrear una trayectoria limpia para la predicción de esta serie.

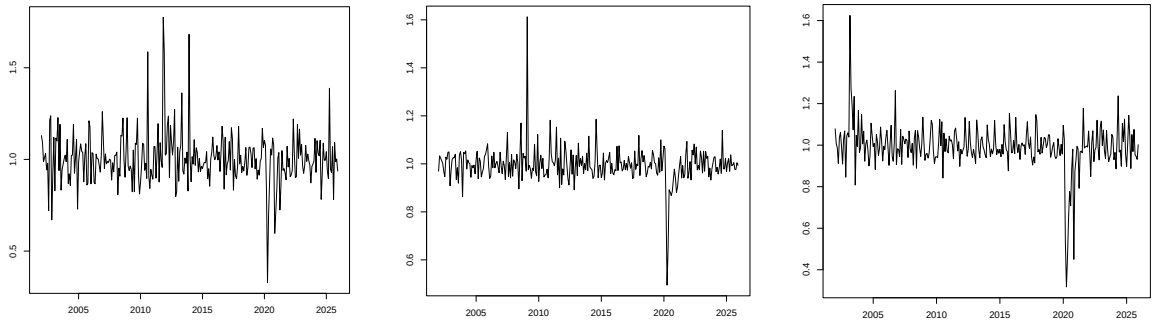


Figura 4.9: De izquierda a derecha: Componente irregular asociada a las series de exportaciones de Alemania, Portugal e Italia.

Finalmente, en la Figura 4.9 tenemos las gráficas asociadas a los países que han resultado más fáciles de predecir. Esto último se debe a que, como podemos ver en las mismas, la escala del ruido está mucho más contenida y acotada. Es decir, las variaciones irregulares de estos países se mueven en un intervalo más manejable que las series previas.

Por último, antes de concluir el análisis del conjunto de datos de exportaciones, extraeremos una serie de conclusiones en base a las métricas de interés. Fijémonos primero en las tres métricas absolutas del modelo LSTM asociadas a Francia, cuyos resultados nos sorprenden negativamente dado el gran volumen de comercio que hay con el mismo, pero se explican como decíamos, con la cantidad de ruido que pudimos ver en la Figura 4.8. Observemos que, para la frecuencia de datos mensual, el error asociado a la métrica MedAE es de aproximadamente 65 millones de euros, pero el obtenido para el MAE es de 72 millones de euros y el RMSE llega a 88 millones de euros. Como el RMSE eleva los errores al cuadrado, castiga las desviaciones grandes más que el MAE, por lo que si el primero es significativamente mayor que el segundo, denota errores puntuales muy grandes, sugiriendo la presencia de outliers. Por otra parte, MAE es un promedio y MedAE es una mediana, por lo que si el MAE es significativamente más alto que el MedAE, indica que hay unos pocos errores grandes arrastrando el error hacia arriba, sugiriendo una asimetría del error. Esto nos dice que aunque LSTM capta bien la tendencia general, con una mediana de error controlada, tiene momentos donde falla abruptamente, indicando que hay picos que el modelo no puede anticipar dando lugar a residuos grandes en meses determinados. Este mismo patrón se repite, por ejemplo, en Estados Unidos para la misma frecuencia y modelo.

Prestemos atención ahora a un fenómeno muy interesante que ocurre en Alemania con datos trimestrales, comparando los modelos random forest y LSTM. Fijémonos que LSTM presenta un MedAE notablemente más bajo (32 millones de euros aproximadamente) que el de random forest (más de 45 millones de euros). Sin embargo, al observar el RMSE ocurre lo contrario. Random forest logra un RMSE de aproximadamente 60.8 millones de euros, mientras que LSTM lo eleva hasta 68.7 millones de euros. La conclusión que podemos extraer en base a esto es que LSTM presenta una mayor precisión que random forest en la mayoría de las predicciones, de ahí que el MedAE sea menor, pero cuando se equivoca lo hace con errores más grandes, disparando su RMSE. Esto se puede tratar de intuir en la gráfica de la izquierda de la Figura 4.6 y en la gráfica de la derecha de la Figura 4.5. Por lo tanto, podríamos decir que los modelos de deep learning tienden a rendir mejor en términos medianos, pero son más susceptibles a fallar valores atípicos que hagan que se dispare su RMSE. Los métodos basados en árboles, como ya se comentó en el Capítulo 2 son capaces de lidiar mejor con esta varianza, ofreciendo predicciones más seguras en este aspecto.

4.2. Datos de la Red Eléctrica Española

Continuamos con el segundo conjunto de datos a estudiar que corresponde a la demanda de electricidad en España. Esta última es una aclaración importante, ya que no se trata de la demanda peninsular, sino española, en la que entran las islas. Este conjunto de datos será relativamente distinto del anterior, pero compartirán muchos detalles en la metodología, como veremos en la subsección posterior. La primera diferencia es que esta vez no trabajaremos con siete series de tiempo al mismo tiempo, sino con una sola: la demanda de electricidad en españa, medida en megavatios (MW), que es la unidad que en la que muestra los datos la propia página de la Red Eléctrica.

Debemos destacar dos detalles adicionales sobre este conjunto de datos. El primero es que no se tomarán variables exógenas para tratar de ayudar a predecir la demanda de electricidad, a pesar de que, como veremos a continuación, seguramente nos beneficiaríamos de la inclusión de ciertas variables cíclicas. Para esta predicción simplemente tendremos los valores históricos de nuestra serie, dado que el objetivo en este caso es dejar libres a los modelos, sin ayudas adicionales, para evaluar su rendimiento bajo estas condiciones. El segundo detalle está relacionado con la evaluación de la calidad del modelo. La propia página de la red eléctrica española muestra las predicciones de demanda que realiza la empresa responsable, por lo que además de medir la calidad de las predicciones realizadas en base a nuestros modelos, calcularemos las métricas de error en base a las predicciones de la propia Red Eléctrica, y las usaremos como un sparring extra para las nuestras, al que queremos aspirar a acercarnos.

4.2.1. Los datos

En lo referente a los datos, es muy interesante destacar un hecho que cambia enormemente con respecto al conjunto de exportaciones. Esta diferencia llega en términos de frecuencia de datos. Recordemos que el análisis previo se llevó a cabo para una frecuencia mensual y una frecuencia trimestral. Estos datos, en cambio, presentan una frecuencia de cinco minutos, así como un volumen mucho mayor, ya que se manejan datos desde el 1 de noviembre de 2025 hasta el 8 de marzo de 2026. Se puede apreciar que el periodo de tiempo escogido es muy particular. El motivo es que el día en que se tomaron los datos era 8 de marzo, y después se fueron haciendo pruebas tomando cada vez más datos (yendo más atrás en el tiempo). Finalmente se paró en el 1 de noviembre por tener suficientes datos y ser una fecha que marca el inicio de un nuevo mes.

Pudimos obtener los datos de la propia página de la Red Eléctrica Española (Red Eléctrica de España (REE) 2026), que es el operador del sistema eléctrico español encargado de asegurar el correcto funcionamiento, de la continuidad y de la seguridad del suministro eléctrico. Como operador de sistema, entre sus cometidos se incluye el ofrecer datos detallados sobre la demanda, generación, etc. de la energía eléctrica en España, ver Wikipedia contributors (2026).

4.2.2. Metodología

Llega el momento de desglosar las ideas de la metodología que se ha llevado a cabo para el ajuste de estos datos. A pesar de ser conjuntos muy diferentes, existen analogías metodológicas entre ambos que los podrán hacer comparables hasta cierto punto. Lo primero que se hará, también para este conjunto de datos, es ajustar nuestro modelo ARIMA que hará las veces de sparring, un sparring que esta vez aspiraremos a superar.

A pesar de que el proceso del ajuste es el mismo, en este caso se han tenido en cuenta unas consideraciones adicionales, debido a que el modelo que generaba la función *auto_arima* por sí sola no pasaba el test de media cero. Para que lo pudiera hacer, le proporcionamos algunas variables exógenas que la ayudasen a captar cambios de tendencia y componente estacional. En este caso, al decir componente estacional nos referimos al patrón semanal y diario claro que posee la serie. Este último fue el que decidimos atacar en primer lugar, como veremos a continuación. Primero le incluimos una variable *dummy*, como la que ya incluimos para el shock en los aranceles estadounidenses, cuya función era la de indicar que a partir del 7 de enero de 2026 parecía haber un aumento general en el nivel de la serie, asociado a la llegada del frío invernal, con las consecuencias que este tiene en la demanda de electricidad. Después, se le añadió una variable cíclica diaria, para tratar de ayudar a capturar una posible componente estacional del mismo periodo. Esta se calculó de manera análoga a las del apartado anterior, pero adaptada a la frecuencia actual:

$$\text{dia\_sin} = \sin\left(\frac{2\pi \cdot \text{minutos dia}}{\text{periodo dia}}\right),$$
$$\text{dia\_cos} = \cos\left(\frac{2\pi \cdot \text{minutos dia}}{\text{periodo dia}}\right).$$

Dado que tampoco se obtuvieron resultados incluyendo estas variables, ya que el modelo seguía sin superar el mencionado test, se optó por levantar las restricciones impuestas sobre los parámetros p y q, asumiendo de esta forma que el tiempo de computación iba a aumentar. Una vez hecho esto, el modelo al que llegó la función *auto_arima*, cuyos parámetros eran mayores que las cotas fijadas previamente, por lo que resulta evidente que estaban siendo una limitación, logró superar el test de media cero. Se probó entonces a retirar las variables exógenas y siguió dando resultado. Aun así buscando la mayor rigurosidad científica posible, se ha decidido dejar constancia de este proceso tanto por escrito en el documento actual, como en el código, aunque ligeramente adaptado para asegurar el correcto funcionamiento del mismo.

Posteriormente llega el ajuste de los modelos de machine learning propiamente dichos, el random forest y el XGBoost. Nuevamente estos comparten las ideas expuestas para el anterior conjunto de datos, y es este requisito de estructurar los datos como un problema de regresión estándar. Dado que en este caso, como decíamos anteriormente, analizamos una única serie de demanda eléctrica, lo que es el proceso se simplifica, pero aumenta en dimensionalidad y coste computacional debido al gran volumen de datos con el que se está trabajando. Para lograr nuestro objetivo aplicamos una transformación tabular análoga, pero esta vez empleamos la función que considera el caso recursivo para una única serie, $ForecasterRecursive$. Podemos ver una representación gráfica análoga a la vista anteriormente en la Figura 4.10.

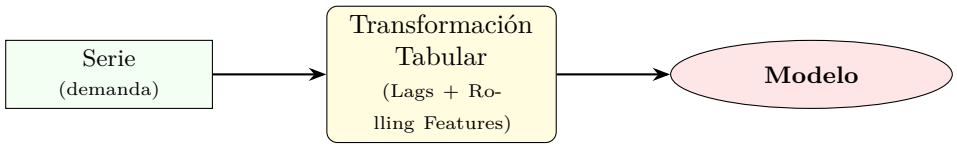


Figura 4.10: Diagrama del ajuste con una única serie temporal sin variables exógenas (explicitando la aparición de las rolling features).

Este proceso crea una matriz de características, X, y una variable objetivo utilizando nuevamente la técnica de ventana deslizante. Cada fila de la matriz representa un instante de tiempo, t, y sus columnas vuelven a estar formadas por los *lags* correspondientes. En nuestro diseño hemos optado por configurar un parámetro histórico enorme, ya que hemos fijado $lags = 8064$. Esto significa que para predecir la demanda de electricidad en el instante de tiempo t el modelo recibe como variables predictoras los 8064 registros anteriores, que en términos de datos con una frecuencia de cinco minutos, serían las cuatro semanas anteriores. Además, cabe destacar que se han tomado cuatro semanas dado que vamos a predecir a un horizonte de tiempo de un día. A esto le añadimos las rolling features, creadas nuevamente con la función $RollingFeatures$, que calculan la media y desviación típica de las últimas 1 y 2 horas, para no dejar de lado la inercia más inmediata. Vemos esto representado en la Figura 4.11, a la que ahora hemos incorporado la presencia de las rolling features.

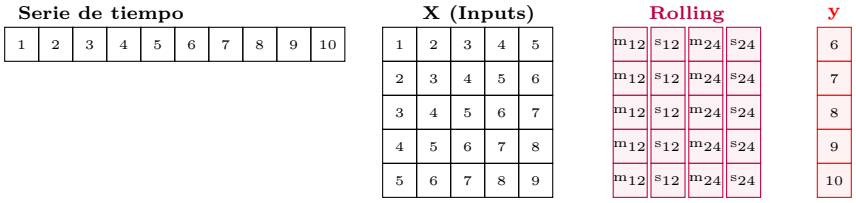


Figura 4.11: Diagrama que muestra la matriz de entrada con *lags* y rolling features (media y desviación típica).

Sin embargo, vuelve a ocurrirnos lo mismo que antes. Volvemos a tener la limitación conceptual de que estos modelos basados en árboles no interiorizan el orden cronológico de la secuencia, procesando los datos de la forma que hemos explicado previamente. Para abordar esta falta de conocimiento temporal volvemos a implementar las arquitecturas asociadas a las redes neuronales recurrentes LSTM y GRU.

A pesar de emplear $ForecasterRecursive$ (contamos con una única serie) para llevar a cabo el backtesting de forma tabular, mediante la clase $RegresionRNN$ cambiamos la forma en la que el algoritmo trata los datos. Al no existir variables exógenas que separar, el procesamiento se vuelve más directo. La red toma las columnas asociadas a los lags y las empaqueta en un tensor de tres dimensiones, que sigue la estructura: $[batch, secuencia, caracteristicas]$. En el caso que nos ocupa esto significa procesar un bloque donde la secuencia tiene una longitud de 8064 pasos temporales.

A diferencia de XGBoost o random forest, las capas de LSTM y GRU procesan la mencionada secuencia paso a paso. Para el modelo de demanda eléctrica, debido a la alta dimensionalidad de la ventana deslizante que hemos fijado en este estudio y la ausencia de variables exógenas, la secuencia se introduce en la red neuronal en orden cronológico inverso, desde el registro más reciente, hasta el más antiguo. La red neuronal es capaz de extraer los patrones temporales adaptando sus pesos a esta dirección. Esto puede extrañar dado el proceso seguido con los datos de exportaciones, pero es que en este último caso fue necesario fijar el orden de más antiguo a más reciente, dada la necesidad de fusionar la secuencia histórica con las variables exógenas en el último paso.

Finalmente, una vez que la red neuronal ha procesado el último *lag* de la secuencia temporal, el vector de memoria resultante pasa por una capa lineal que proyecta todo ese aprendizaje en un único valor, que será precisamente el valor estimado de demanda eléctrica para los próximos 5 minutos. Al igual que se indicó, las métricas de error que se emplearán para medir la calidad de las predicciones de los modelos en la muestra de test son las que se mostraron en la Sección 1.4 del Capítulo 1.

Antes de continuar con el análisis de los resultados obtenidos para este conjunto de datos, especificaremos la configuración de los modelos de machine y deep learning, tal y como hicimos con el conjunto de exportaciones. En este caso tenemos una preocupación diferente a la que teníamos con el anterior. El peligro ahora no es el riesgo de sobreajuste como tal, ya que disponemos de una base de datos más grande, el problema ahora es vigilar el coste computacional y el tiempo de entrenamiento, sobre todo en el caso de los modelos de deep learning.

Con esto en mente, la configuración escogida para random forest fue la de tomar 200 árboles para promediar, una profundidad máxima de 7 (mayor que la escogida para el conjunto de datos de exportaciones), y el número minimo de elementos por hoja lo mantenemos en 8. Por su parte, en XGBoost volvemos a usar la misma configuración, combinando 500 árboles con una tasa de aprendizaje de 0.01 y una profundidad máxima de 3, que sigue siendo menor que la escogida para random forest. Además, se ha vuelto a fijar que para entrenar cada árbol solamente se pueda usar el 80 % de las filas. Por último, para los modelos de deep learning hemos escogido una configuración de 32 unidades ocultas, tres capas y un *dropout* de 0.1. Notemos que ya no le tenemos ese miedo al sobreajuste pero sí cierto respeto al tiempo de computación, por lo que se han fijado 20 *epoch*, pudiendo ver que no se producía una mejoría del error en las últimas. Por último, cabe destacar que se ha reescalado y diferenciado la serie de tiempo para los cuatro modelos, ya que pudimos apreciar que esto potenciaba la calidad de los resultados obtenidos. Una vez realizadas estas últimas aclaraciones, procedemos con el análisis de los resultados en la siguiente subsección.

4.2.3. Resultados

Tal y como hicimos con el conjunto de datos de exportaciones, llevaremos a cabo un análisis de los resultados obtenidos en base al ajuste de los mismos modelos. Se pueden apreciar diferencias notables entre ambos conjuntos de datos, que repercutirán a su vez en la estructura del análisis. En este caso, como hemos dicho anteriormente, contamos con una serie de tiempo con una rejilla mucho mayor, pero es una única serie, y sin variables exógenas que ayuden en su predicción. Tal y como hicimos para el conjunto de datos previo, nos basaremos en la tabla que contiene las métricas más relevantes para extraer nuestras conclusiones, la Tabla A.3, que se encuentra en la Sección A.1 del Apéndice A. A su vez,la Tabla A.6 que contiene los resultados asociados al resto de métricas se encuentra en la Sección A.1 del mismo Apéndice. Se debe tener en cuenta que esta vez los valores dentro las mismas se miden en megavatios (MW). Podemos ver en las mismas que esta vez no hay países en las columnas, sino los datos que se han comparado. Por ejemplo, la columna de Modelo vs Real son las métricas de error calculadas para la comparación entre las predicciones de nuestros modelos y la demanda de electricidad real, Modelo vs Sparring contiene las métricas asociadas la comparación entre nuestras predicciones y las que muestran ellos en su página y finalmente Real vs Sparring, las métricas de comparación entre la

demanda real y las predicciones que ellos mismos realizaron. Comencemos por lo tanto con el análisis de los resultados.

Iniciamos el análisis de la misma forma que lo hicimos para el conjunto de datos de exportaciones, esta vez sin mostrar una gráfica de los datos, ya que se plasmó en la Figura 1.1 del Capítulo 1. A pesar de no poder apreciar, quizá, una tendencia tan clara como la que se podía distinguir en las series de exportaciones, volveremos a diferenciar la serie para ajustar tanto los modelos basados en árboles como las redes neuronales recurrentes, por el problema de estos con la extrapolación de la tendencia expuesto en la sección previa.

En lo referente al modelo ARIMA como representante del enfoque estadístico clásico, vemos en las métricas que el modelo es incapaz de capturar las no linealidades de la serie de tiempo. Como era de esperar, dado que nuevamente lo hemos planteado como un sparring fácil de batir (no confundir con el sparring potente, que son las predicciones que proporciona la propia página de la red eléctrica), se ha obtenido un WAPE del 21.64 % al compararlo con los datos reales de demanda. Esto en cierto modo valida nuestro enfoque inicial, de plantear este tipo de modelos como una base que los posteriores deberían mejorar.

Una vez visto esto, pasamos al campo del machine learning al uso y vemos que los modelos basados en árboles proporcionan opciones mucho más sólidas. Tal y como nos ocurría con el anterior conjunto de datos, XGBoost vuelve a superar a random forest, aunque no por demasiada diferencia, ya que logra un WAPE de 4.31 frente a un 4.75 del segundo. Esta mejora también repercute en una caída de los valores de las métricas absolutas, confirmando que XGBoost es más hábil en la captación de patrones que random forest. Esto se puede ver reflejado en la Figura 4.12.

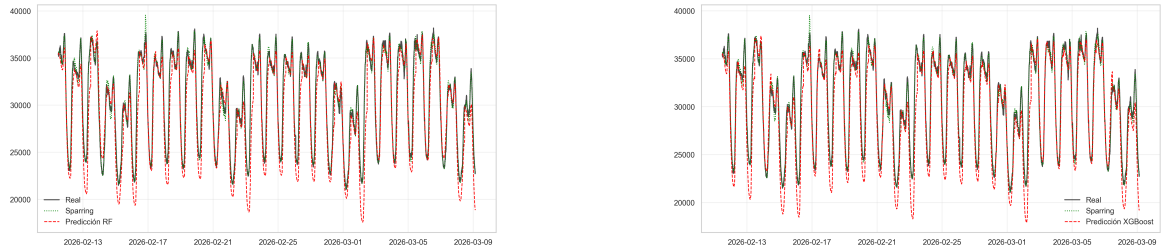


Figura 4.12: De izquierda a derecha: Predicciones con el modelo random forest y con el modelo XGBoost realizadas sobre el conjunto de test para los datos de demanda eléctrica (MW). La curva negra continua está asociada a la demanda real y la roja discontinua a las predicciones realizadas con nuestros modelos. También se muestra la curva verde discontinua, asociada a las predicciones de la propia página.

Por otra parte, los mejores resultados llegan de la mano de los modelos de deep learning. Tanto LSTM como GRU destacan sobre los modelos basados en árboles, logrando GRU el mejor ajuste con un WAPE de 3.56 %, seguido de cerca por LSTM con 3.74 %, ambos relativamente próximos al sparring de referencia al que aspiramos a acercarnos, 1.01 %, cuya precisión es digna de elogio. Este resultado apoya lo que comentamos en la teoría, y es que parece que la mayor simplicidad de GRU está dando lugar a un mejor rendimiento. Podemos visualizar los resultados asociados a los modelos de deep learning en la Figura 4.13.

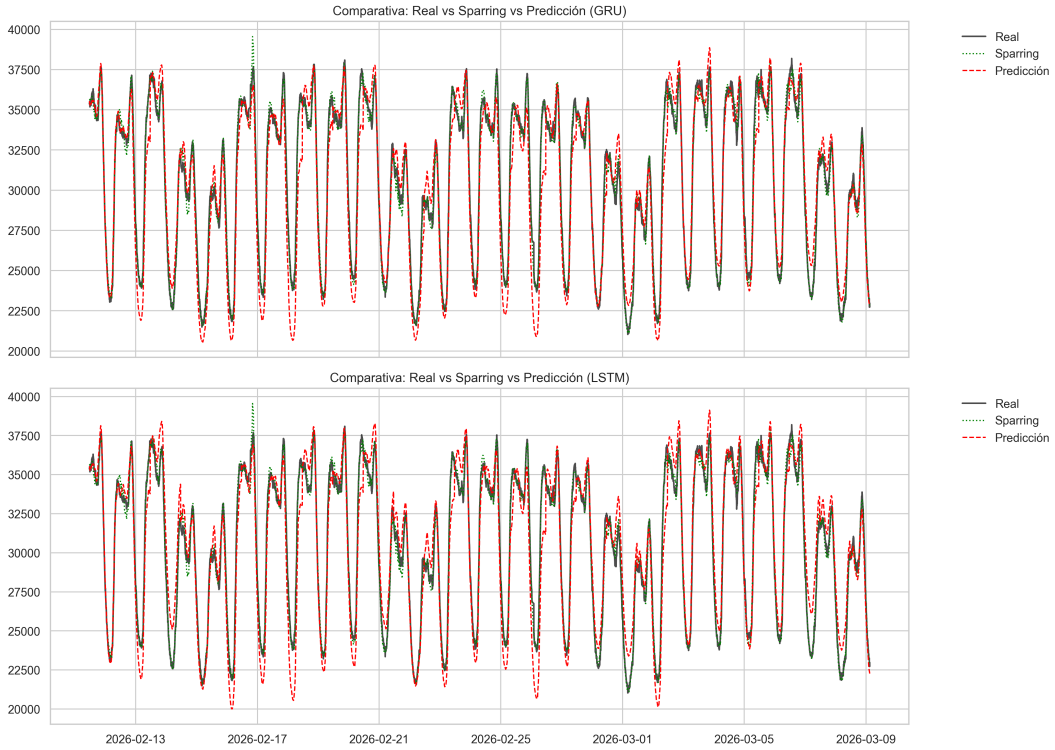


Figura 4.13: Predicciones con redes neuronales recurrentes (LSTM y GRU) realizadas sobre el conjunto de test para los datos de demanda eléctrica (MW). La curva negra continua está asociada a la demanda real y la roja discontinua a las predicciones realizadas con nuestros modelos. También se muestra la curva verde discontinua, asociada a las predicciones de la propia página.

Centrémonos ahora en las conclusiones que nos permiten extraer las métricas de forma directa. Por ejemplo, fijémonos en los resultados de XGBoost. Se puede apreciar que pasamos de un error mediano de 986 MW a un RMSE de 1792 MW, indicando la presencia de predicciones puntuales con errores muy grandes, que han arrastrado las medias hacia arriba. Por el contrario, los modelos de deep learning se muestran más estables en este aspecto, ya que la diferencia entre la mediana y los errores cuadráticos es menor. A pesar de esto, y como se puede ver también en la Figura 4.13, estos modelos siguen fallando en los picos de forma notable. Ponemos como ejemplo LSTM, que obtiene un MedAE parecido al de XGBoost, 985.46 %, y un RMSE de 1436.31 %, que es menor que el de XGBoost a pesar de que sigue estando desmarcado del error mediano.

Esto nos permite decir que si bien el MedAE de la red neuronal es comparable al de los modelos basados en árboles, los modelos de deep learning están destacando más en momentos de alta volatilidad, con más picos. A pesar de que el objetivo de los modelos ajustados basados en árboles es la reducción de la variabilidad, nos encontramos ante un conjunto de datos más grande, donde cada vez empieza a salir a relucir más el potencial de las redes neuronales. Estos últimos se muestran más estables, ya que cuando se equivocan lo hacen con desviaciones menores, evitando los errores grandes que probablemente estén disparando los de random forest y XGBoost.

Un último comentario que resulta relevante es que a lo largo del proceso de creación de los modelos se probó a realizar un ajuste tomando una y dos semanas de datos para entrenar y un horizonte de predicción de una hora. Lo interesante, es que para esta configuración los modelos XGBoost, LSTM y GRU mejoraban las predicciones de la propia página de la red eléctrica. Además, el tiempo de

entrenamiento era mucho menor que el que tenemos para la configuración actual. El motivo de cambiar la configuración fue estar en igualdad de condiciones, ya que las predicciones mostradas en la página de la red eléctrica se realizan, como mínimo, a un horizonte de un día, pero queda abierta la posibilidad de investigar en futuros estudios la implementación de estos modelos de forma que produzcan predicciones hora a hora de forma recursiva, dada la precisión de las mismas y la rapidez con la que se obtienen.

Apéndice A

Tablas de resultados

En la Sección A.1 mostramos las tablas con los resultados correspondientes a las métricas de error principales: MAE, RMSE, MedAE y WAPE, tanto para el conjunto de datos de exportaciones como para el conjunto de datos de demanda eléctrica. En la Sección A.2 mostramos las tablas con los resultados correspondientes a las métricas de error que se ha considerado que podían resultar redundantes una vez analizadas las primeras. Aun así, se incluyen por si fuesen del interés del lector.

A.1. Métricas principales

Tabla A.1: Métricas principales por país y modelo (Datos Mensuales)

Métrica	**France**	**Germany**	**Italy**	**Japan**	**Portugal**	**United Kingdom**	**United States**
RMSE (ARIMA)	600,609,323	88,330,544	88,483,155	5,555,664	155,192,074	211,193,220	1,304,441,234
MAE (ARIMA)	503,631,783	70,910,423	71,763,069	4,393,874	115,731,763	180,058,918	611,987,453
WAPE % (ARIMA)	117.83	48.61	29.31	34.82	33.79	150.15	993.91
MedAE (ARIMA)	497,320,401	61,890,110	68,230,923	3,554,999	104,977,325	180,137,304	34,866,116
RMSE (Random Forest)	86,600,724	30,199,733	55,907,532	6,432,232	48,919,278	34,828,595	27,778,622
MAE (Random Forest)	69,703,198	22,029,528	45,747,299	4,993,963	40,539,943	27,791,057	23,073,260
WAPE % (Random Forest)	16.36	15.05	18.62	39.59	11.78	23.13	37.16
MedAE (Random Forest)	58,102,913	18,269,220	40,600,125	3,960,243	34,838,325	24,006,420	20,800,856
RMSE (XGBoost)	83,060,664	29,863,434	52,752,617	6,226,888	50,248,037	34,234,585	28,076,557
MAE (XGBoost)	67,632,643	22,316,017	42,914,993	5,175,888	40,656,943	26,943,104	22,512,384
WAPE % (XGBoost)	15.87	15.25	17.47	41.03	11.82	22.42	36.26
MedAE (XGBoost)	55,019,728	17,371,235	34,346,976	4,839,648	32,831,901	19,674,716	18,895,093
RMSE (GRU)	95,895,053	30,060,674	56,647,670	7,341,158	48,175,827	44,637,168	31,707,400
MAE (GRU)	74,471,478	22,613,756	44,731,515	5,996,428	39,343,337	35,949,878	27,416,408
WAPE % (GRU)	17.48	15.45	18.21	47.53	11.43	29.92	44.16
MedAE (GRU)	63,696,973	17,137,495	35,704,001	4,867,428	37,311,693	33,592,147	27,804,209
RMSE (LSTM)	87,789,970	32,604,633	57,064,811	8,332,444	52,154,218	36,845,745	28,702,322
MAE (LSTM)	71,724,615	24,316,757	43,881,414	6,677,983	42,895,307	28,187,154	20,428,670

Continúa en la página siguiente...

Métricas principales por país y modelo (Datos Mensuales) (continuación)

Métrica	**France**	**Germany**	**Italy**	**Japan**	**Portugal**	**United Kingdom**	**United States**
WAPE % (LSTM)	16.84	16.61	17.86	52.94	12.47	23.46	32.90
MedAE (LSTM)	65,479,239	20,147,483	33,451,330	5,061,959	38,296,857	20,103,568	14,404,773

Tabla A.2: Métricas principales por país y modelo (Datos Trimestrales)

Métrica	**France**	**Germany**	**Italy**	**Japan**	**Portugal**	**United Kingdom**	**United States**
RMSE (ARIMA)	836,400,534	100,421,030	219,870,930	15,819,969	396,409,935	615,847,261	11,302,290,400
MAE (ARIMA)	777,024,309	71,275,208	204,389,034	13,450,882	291,702,624	540,410,694	5,156,262,270
WAPE % (ARIMA)	60.78	16.47	28.17	35.21	28.77	151.30	2,852.31
MedAE (ARIMA)	715,259,895	50,908,968	193,660,138	13,697,125	239,134,011	567,211,774	189,989,456
RMSE (Random Forest)	247,281,910	60,791,248	139,477,213	14,707,678	117,933,456	94,754,854	69,003,199
MAE (Random Forest)	208,740,589	46,224,367	118,714,594	11,376,608	92,137,700	83,899,271	57,348,024
WAPE % (Random Forest)	16.33	10.53	16.11	30.06	8.93	23.28	30.79
MedAE (Random Forest)	179,439,059	45,448,537	101,542,782	11,686,011	80,093,841	79,076,826	49,496,351
RMSE (XGBoost)	228,645,589	67,333,095	140,195,336	13,511,468	135,649,010	100,999,646	70,444,918
MAE (XGBoost)	196,155,094	48,539,983	123,665,752	9,663,004	106,827,088	85,378,463	58,378,482
WAPE % (XGBoost)	15.35	11.05	16.78	25.53	10.35	23.69	31.34
MedAE (XGBoost)	199,173,096	36,180,652	133,756,383	6,749,629	89,205,340	74,074,328	47,713,285

Continúa en la página siguiente...

Métricas principales por país y modelo (Datos Trimestrales) (continuación)

Métrica	**France**	**Germany**	**Italy**	**Japan**	**Portugal**	**United Kingdom**	**United States**
RMSE (GRU)	233,681,062	67,684,280	135,437,673	13,205,410	120,227,448	90,461,353	66,922,465
MAE (GRU)	197,597,487	53,193,504	122,607,808	9,330,155	97,264,109	78,882,076	53,687,498
WAPE % (GRU)	15.46	12.11	16.64	24.65	9.42	21.88	28.82
MedAE (GRU)	188,255,851	39,495,717	134,007,117	5,065,681	90,210,347	61,346,024	43,993,593
RMSE (LSTM)	224,135,372	68,708,060	137,357,853	12,636,014	127,140,570	86,099,701	69,566,043
MAE (LSTM)	192,737,014	50,476,499	122,604,910	9,061,586	101,954,433	75,047,036	56,151,376
WAPE % (LSTM)	15.08	11.50	16.64	23.94	9.88	20.82	30.15
MedAE (LSTM)	201,196,425	32,029,084	133,160,230	5,131,292	93,517,149	66,842,635	49,814,666

Tabla A.3: Métricas principales por comparación y modelo

Métrica	**Modelo vs Real**	**Modelo vs Sparring**	**Real vs Sparring**
RMSE (ARIMA)	8035.03	8005.59	414.2667
MAE (ARIMA)	6627.07	6604.41	309.8214
WAPE % (ARIMA)	21.64	21.60	1.0118
MedAE (ARIMA)	5778.10	5833.04	234
RMSE (Random Forest)	2059.0329	2022.4744	414.2667
MAE (Random Forest)	1454.2764	1427.1547	309.8214
WAPE % (Random Forest)	4.7494	4.6676	1.0117
MedAE (Random Forest)	1003.6689	975.4439	234
RMSE (XGBoost)	1792.7683	1751.0275	414.2667
MAE (XGBoost)	1319.6667	1276.9263	309.8214
WAPE % (XGBoost)	4.3098	4.1762	1.0118
MedAE (XGBoost)	986.4335	959.8618	234
RMSE (LSTM)	1436.3144	1424.1859	414.2667
MAE (LSTM)	1144.4717	1144.7427	309.8214
WAPE % (LSTM)	3.7377	3.7439	1.0118
MedAE (LSTM)	985.4623	1006.4728	234
RMSE (GRU)	1398.0036	1363.2449	414.2667
MAE (GRU)	1090.2960	1071.6021	309.8214
WAPE % (GRU)	3.5607	3.5047	1.0118
MedAE (GRU)	936.3279	934.2942	234

A.2. Métricas adicionales

Tabla A.4: Métricas adicionales por país y modelo (Datos Mensuales)

Métrica	**France**	**Germany**	**Italy**	**Japan**	**Portugal**	**United Kingdom**	**United States**
Max_Err (ARIMA)	1,453,621,819	197,844,195	192,483,504	14,253,338	327,759,218	414,801,199	3,342,563,936
MAPE % (ARIMA)	122.27	49.09	29.58	46.99	32.33	165.01	1,386.23
Rel_Err % (ARIMA)	117.83	48.61	29.31	34.82	33.79	150.15	993.91
Rel_MedAE % (ARIMA)	116.19	41.90	28.90	28.70	29.90	157.79	62.61
R^2 (ARIMA)	-68.86	-7.29	-2.79	0.02	-11.40	-38.92	-2,815.80
Max_Err (Random Forest)	181,900,393	97,671,974	118,731,755	16,464,773	114,358,373	88,660,180	73,903,012
MAPE % (Random Forest)	17.75	14.69	17.77	57.03	11.94	25.10	41.69
Rel_Err % (Random Forest)	16.36	15.05	18.62	39.59	11.78	23.13	37.16
Rel_MedAE % (Random Forest)	13.68	12.36	17.20	32.89	9.90	20.91	37.13
R^2 (Random Forest)	-0.46	0.03	-0.52	-0.29	-0.31	-0.07	-0.29
Max_Err (XGBoost)	194,896,908	97,802,268	119,637,161	13,511,543	116,367,377	100,981,659	87,097,052
MAPE % (XGBoost)	16.96	14.73	16.80	63.11	11.89	24.07	39.02
Rel_Err % (XGBoost)	15.87	15.25	17.47	41.03	11.82	22.42	36.26
Rel_MedAE % (XGBoost)	12.96	11.75	14.55	40.19	9.33	17.13	33.73
R^2 (XGBoost)	-0.34	0.06	-0.35	-0.21	-0.39	-0.03	-0.32
Max_Err (GRU)	246,999,127	94,184,857	127,932,487	15,922,039	118,202,852	97,773,169	75,605,506
MAPE % (GRU)	19.37	15.39	17.62	77.68	11.57	35.54	51.85
Rel_Err % (GRU)	17.48	15.45	18.21	47.53	11.43	29.92	44.16

Continúa en la página siguiente...

Métricas adicionales por país y modelo (Datos Mensuales) (continuación)

Métrica	France	Germany	Italy	Japan	Portugal	United Kingdom	United States
Rel_MedAE % (GRU)	15.00	11.60	15.12	40.42	10.60	29.25	49.63
R^2 (GRU)	-0.79	0.04	-0.56	-0.68	-0.28	-0.76	-0.68
Max_Err (LSTM)	257,617,270	103,044,876	138,063,927	18,357,326	121,425,209	115,239,701	104,159,231
MAPE % (LSTM)	17.00	15.67	17.05	58.84	12.54	23.78	32.06
Rel_Err % (LSTM)	16.84	16.61	17.86	52.94	12.47	23.46	32.90
Rel_MedAE % (LSTM)	15.42	13.63	14.17	42.03	10.88	17.51	25.71
R^2 (LSTM)	-0.50	-0.13	-0.58	-1.17	-0.49	-0.20	-0.37

Tabla A.5: Métricas adicionales por país y modelo (Datos Trimestrales)

Métrica	France	Germany	Italy	Japan	Portugal	United Kingdom	United States
Max_Err (ARIMA)	1,384,472,659	282,089,347	363,250,479	32,058,618	820,613,645	1,133,066,224	29,798,105,658
MAPE % (ARIMA)	62.49	16.15	27.16	32.21	27.10	155.09	3,820.35
Rel_Err % (ARIMA)	60.78	16.47	28.17	35.21	28.77	151.30	2,852.31
Rel_MedAE % (ARIMA)	55.23	11.26	25.81	36.08	22.60	171.19	116.44
R^2 (ARIMA)	-33.99	-1.17	-2.83	-2.50	-7.94	-92.85	-40,789.26
Max_Err (Random Forest)	422,327,026	132,045,248	240,061,114	35,691,484	244,274,866	179,177,225	184,462,454

Continúa en la página siguiente...

Métricas adicionales por país y modelo (Datos Trimestrales) (continuación)

Métrica	**France**	**Germany**	**Italy**	**Japan**	**Portugal**	**United Kingdom**	**United States**
MAPE % (Random Forest)	16.88	10.19	15.56	35.64	9.27	24.01	30.54
Rel_Err % (Random Forest)	16.33	10.53	16.11	30.06	8.93	23.28	30.79
Rel_MedAE % (Random Forest)	13.72	10.02	13.35	32.18	7.50	23.86	30.02
R^2 (Random Forest)	-1.91	0.09	-0.82	-1.98	-0.20	-1.22	-0.77
Max_Err (XGBoost)	427,467,461	171,350,637	231,561,381	36,408,989	261,716,070	220,350,371	184,652,247
MAPE % (XGBoost)	15.82	10.66	16.37	31.84	10.72	24.37	31.62
Rel_Err % (XGBoost)	15.35	11.05	16.78	25.53	10.35	23.69	31.34
Rel_MedAE % (XGBoost)	15.23	7.98	17.59	18.59	8.35	22.35	28.93
R^2 (XGBoost)	-1.48	-0.12	-0.84	-1.51	-0.59	-1.53	-0.84
Max_Err (GRU)	490,549,551	153,989,855	227,967,126	33,042,466	250,751,612	172,418,579	174,985,315
MAPE % (GRU)	16.14	11.98	16.50	30.16	9.76	22.67	29.18
Rel_Err % (GRU)	15.46	12.11	16.64	24.65	9.42	21.88	28.82
Rel_MedAE % (GRU)	14.40	8.71	17.62	13.95	8.45	18.51	26.68
R^2 (GRU)	-1.59	-0.13	-0.72	-1.40	-0.25	-1.03	-0.66
Max_Err (LSTM)	442,030,309	161,585,729	236,453,916	31,396,615	247,390,481	164,117,595	174,790,926
MAPE % (LSTM)	15.54	11.20	16.39	28.99	10.16	21.38	30.90
Rel_Err % (LSTM)	15.08	11.50	16.64	23.94	9.88	20.82	30.15
Rel_MedAE % (LSTM)	15.39	7.06	17.51	14.13	8.76	20.17	30.21
R^2 (LSTM)	-1.39	-0.16	-0.77	-1.20	-0.39	-0.84	-0.80

Tabla A.6: Métricas adicionales por comparación y modelo

Métrica	**Modelo vs Real**	**Modelo vs Sparring**	**Real vs Sparring**
Max_Err (ARIMA)	18347.34	17735.34	2311
MAPE % (ARIMA)	21.29	21.25	1.0056
Rel_Err % (ARIMA)	21.64	21.60	1.0118
Rel_MedAE % (ARIMA)	18.36	18.52	0.7434
R^2 (ARIMA)	-1.9242	-1.8784	0.9922
Max_Err (Random Forest)	8253.1629	8047.7405	2311
MAPE % (Random Forest)	4.9724	4.8845	1.0056
Rel_Err % (Random Forest)	4.7494	4.6676	1.0118
Rel_MedAE % (Random Forest)	3.1887	3.0973	0.7434
R^2 (Random Forest)	0.8080	0.8163	0.9922
Max_Err (XGBoost)	6753.0030	6748.3898	2311
MAPE % (XGBoost)	4.6067	4.4700	1.0056
Rel_Err % (XGBoost)	4.3098	4.1762	1.0118
Rel_MedAE % (XGBoost)	3.1339	3.0479	0.7434
R^2 (XGBoost)	0.8544	0.8623	0.9922
Max_Err (LSTM)	4763.0127	4436.8074	2311
MAPE % (LSTM)	3.9098	3.9152	1.0056
Rel_Err % (LSTM)	3.7376	3.7439	1.0118
Rel_MedAE % (LSTM)	3.1308	3.1959	0.7434
R^2 (LSTM)	0.9066	0.9089	0.9922
Max_Err (GRU)	5001.3560	4676.7363	2311
MAPE % (GRU)	3.6841	3.6285	1.0056
Rel_Err % (GRU)	3.5607	3.5047	1.0118
Rel_MedAE % (GRU)	2.9747	2.9667	0.7434
R^2 (GRU)	0.9115	0.9165	0.9922

Bibliografía

AllElectroHub (2025). *Understanding GRU in Neural Networks*. URL: https://www.allpcb.com/allelectrohub/understanding-gru-in-neural-networks (Accedido 23-03-2026).

Amat Rodrigo J y Escobar Ortiz J (2021). *Skforecast: forecasting de series temporales con Python, Machine Learning y Scikit-learn*. Actualizado en noviembre de 2025. URL: https://cienciadedatos.net/documentos/py27-forecasting-series-temporales-python-scikitlearn (Accedido 05-02-2026).

Aneiros Pérez G (2024). *Series de Tiempo*. Apuntes de clase, Máster Universitario en Técnicas Estadísticas, Universidade da Coruña. Material docente no publicado disponible en Campus Virtual.

Anishnama (2023). *Understanding Gated Recurrent Unit (GRU) in Deep Learning*. URL: https://medium.com/@anishnama20/understanding-gated-recurrent-unit-gru-in-deep-learning-2e54923f3e2 (Accedido 23-03-2026).

Belcic I y Stryker C (2026). *¿Cuál es la tasa de aprendizaje en el machine learning?* IBM Think. URL: https://www.ibm.com/es-es/think/topics/learning-rate (Accedido 19-03-2026).

Bonnin, Rodolfo (2017). *Median absolute error*. URL: https://www.oreilly.com/library/view/machine-learning-for/9781786469878/9f44e711-deb6-42de-abbd-524832ad32cc.xhtml (Accedido 04-03-2026).

Breiman L (1996). Bagging predictors. En: *Machine Learning* 24.2, págs. 123-140.

Breiman L (2001). Random forests. En: *Machine Learning* 45.1, págs. 5-32.

Breiman L, Friedman J, Olshen RA y Stone CJ (1984). *Classification and Regression Trees*. 1st. New York: Chapman y Hall/CRC. DOI: 10.1201/9781315139470.

Cámara de Comercio de España (2025). *Aproximación a los posibles efectos de una tasa arancelaria del 15 % en las exportaciones de bienes españoles a Estados Unidos*. Publicado el 24 de julio de 2025. URL: https://www.camara.es/aproximacion-posibles-efectos-tasa-arancelaria-15-exportaciones-bienes (Accedido 26-03-2026).

Carmona F (2023). *Bootstrap*. Universitat de Barcelona. URL: https://www.ub.edu/cursosR/files/bootstrap.html (Accedido 17-03-2026).

Chen T y Guestrin C (2016). XGBoost: A Scalable Tree Boosting System. En: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. ACM, págs. 785-794. DOI: 10.1145/2939672.2939785. URL: http://dx.doi.org/10.1145/2939672.2939785.

Cybenko G (1989). Approximation by superpositions of a sigmoidal function. En: *Mathematics of Control, Signals and Systems* 2.4, págs. 303-314. DOI: 10.1007/BF02551274. URL: https://doi.org/10.1007/BF02551274.

DataSphere (2025). *WAPE (Weighted Absolute Percentage Error)*. URL: https://insightful-data-lab.com/2025/08/17/wape-weighted-absolute-percentage-error/ (Accedido 05-03-2026).

de la Fuente Fernández S (2025). *Series temporales, modelo ARIMA metodología de Box-Jenkins*. Diapositivas de la asignatura Instrumentos Estadísticos Avanzados. Universidad Autónoma de Madrid. URL: https://www.estadistica.net/ECONOMETRIA/SERIES-TEMPORALES/modelo-arima.pdf (Accedido 22-12-2025).

Fernández-Casal R, Costa J y Oviedo de la Fuente M (2024). *Métodos predictivos de aprendizaje estadístico*. Manuais 42. Versión digital. A Coruña: Servizo de Publicacións da Universidade da

Coruña. ISBN: 978-84-9749-893-7. DOI: 10.17979/spudc.9788497498937. URL: http://hdl.handle.net/2183/37227.

Fondo Monetario Internacional (FMI) (2026). *Fondo Monetario Internacional (FMI)*. URL: https://www.imf.org/es/home (Accedido 21-03-2026).

GeeksforGeeks (2025a). *Python - Coefficient of Determination-R2 score*. URL: https://www.geeksforgeeks.org/machine-learning/python-coefficient-of-determination-r2-score/ (Accedido 05-03-2026).

GeeksforGeeks (2025b). *Stacked RNNs in NLP*. URL: https://www.geeksforgeeks.org/nlp/stacked-rnns-in-nlp/ (Accedido 15-02-2026).

GeeksforGeeks (2026). *Backpropagation Through Time - RNN*. URL: https://www.geeksforgeeks.org/machine-learning/ml-back-propagation-through-time/ (Accedido 12-02-2026).

Gómez V y Maravall A (1996). *Programs TRAMO and SEATS. Instructions for the User*. Working Paper 9628. Banco de España. URL: https://www.bde.es/f/webbde/SES/Secciones/Publicaciones/PublicacionesSeriadas/DocumentosTrabajo/96/Fich/dt9628e.pdf (Accedido 05-05-2026).

Goodfellow I, Bengio Y y Courville A (2016). *Deep Learning*. http://www.deeplearningbook.org. MIT Press, págs. 390-392.

Hastie T, Tibshirani R y Friedman J (2009). *The Elements of Statistical Learning*. New York: Springer.

Hornik K (1991). Approximation capabilities of multilayer feedforward networks. En: *Neural Networks* 4.2, págs. 251-257. ISSN: 0893-6080. DOI: 10.1016/0893-6080(91)90009-T. URL: https://doi.org/10.1016/0893-6080(91)90009-T.

Huet P (2024). *Embeddings: Qué son y cómo transforman datos en información*. OpenWebinars. URL: https://openwebinars.net/blog/embeddings/#:~:text=Los%20embeddings%20transforman%20palabras%2C%20frases,y%20mejora%20la%20eficiencia%20computacional. (Accedido 24-04-2026).

Hyndman RJ y Athanasopoulos G (2021). *Forecasting: Principles and Practice*. 3rd. Accessed on March 7, 2026. Melbourne, Australia: OTexts. URL: https://OTexts.com/fpp3.

Hyndman RJ, Athanasopoulos G, Garza A, Challu C, Mergenthaler M y Olivares KG (2025). *Forecasting: Principles and Practice, the Pythonic Way*. URL: https://otexts.com/fpppy/ (Accedido 25-12-2025).

IBM (2021). *Funciones de autocorrelación y autocorrelación parcial*. URL: https://www.ibm.com/docs/es/spss-modeler/saas?topic=data-autocorrelation-partial-autocorrelation-functions (Accedido 21-12-2025).

IBM (2026). *¿Qué es XGBoost?* IBM Think. URL: https://www.ibm.com/es-es/think/topics/xgboost (Accedido 05-02-2026).

Imperia SCM (2026). *MAPE en Supply Chain: mide y mejora la precisión de tu forecast*. URL: https://imperiascm.com/es-es/blog/mape-prevision-demanda (Accedido 05-03-2026).

INNOVATIANA (2026). *Root Mean Square Error (RMSE)*. URL: https://www.innovatiana.com/es/glossary/root-mean-square-error-rmse (Accedido 04-03-2026).

Instituto Galego de Estatística (IGE) (2026). *Informe de Comercio Exterior de Galicia: decembro 2025*. Inf. téc. Accedido el 21 de marzo de 2026. Xunta de Galicia. URL: https://www.ige.gal/estatico/estatRM.jsp?c=0307004001&ruta=html/gl/OperacionsConxunturais/InformeComercioExterior2025_12.html.

Jain A (2024). *Deep RNNs, Stacked RNNs, Stacked LSTMs*. URL: https://medium.com/@abhishekjainindore24/deep-rnns-stacked-rnns-stacked-lstms-79cf652f451a (Accedido 15-02-2026).

James G, Witten D, Hastie T, Tibshirani R y Taylor J (2023). *An Introduction to Statistical Learning: with Applications in Python*. 1.ª ed. Springer Texts in Statistics. Switzerland: Springer Cham. ISBN: 978-3-031-38747-0. DOI: 10.1007/978-3-031-38747-0. URL: https://link.springer.com/book/10.1007/978-3-031-38747-0.

Jonker A y Gomstyn A (2026). *Datos estructurados vs. no estructurados: ¿cuál es la diferencia?* IBM Think. URL: https://www.ibm.com/es-es/think/topics/structured-vs-unstructured-data (Accedido 07-03-2026).

Joseph M (2022). *Modern Time Series Forecasting with Python: Explore industry-ready time series forecasting using modern machine learning and deep learning*. Independently Published.

Kostadinov S (2017). *Understanding GRU Networks*. URL: https://medium.com/data-science/understanding-gru-networks-2ef37df6c9be (Accedido 23-03-2026).

Kulkarni NN, Mack J y Sabato A (2025). Comparative analysis of data-driven autoencoder networks for full-field expansion from sparse measurements. En: *Mechanical Systems and Signal Processing* 235, pág. 112957. ISSN: 0888-3270. DOI: https://doi.org/10.1016/j.ymssp.2025.112957. URL: https://www.sciencedirect.com/science/article/pii/S0888327025006582.

Lee F (2026). *¿Qué es el equilibrio sesgo-varianza?* IBM Think. URL: https://www.ibm.com/es-es/think/topics/bias-variance-tradeoff (Accedido 16-04-2026).

Leo C (2024a). *The Math Behind Gated Recurrent Units*. URL: https://towardsdatascience.com/the-math-behind-gated-recurrent-units-854d88aded65/ (Accedido 24-03-2026).

Leo C (2024b). *The Math Behind LSTM*. URL: https://towardsdatascience.com/the-math-behind-lstm-9069b835289d/#ae46 (Accedido 24-03-2026).

Makridakis S, Spiliotis E y Assimakopoulos V (2022). M5 accuracy competition: Results, findings, and conclusions. En: *International Journal of Forecasting* 38.4. Special Issue: M5 competition, págs. 1346-1364. ISSN: 0169-2070. DOI: https://doi.org/10.1016/j.ijforecast.2021.11.013. URL: https://www.sciencedirect.com/science/article/pii/S0169207021001874.

MSMK (2024). *Mean Absolute Error*. URL: https://msmk.university/mean-absolute-error/ (Accedido 04-03-2026).

Olah C (2015). *Understanding LSTM Networks*. URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs/ (Accedido 21-03-2026).

Oracle (2026). *Oracle Fusion Cloud EPM Trabajo con Planning*. URL: https://tinyurl.com/oraclemetrics (Accedido 08-03-2026).

Pompas Gutiérrez J (2023). *Descubriendo las redes neuronales recurrentes: Gestión de la Memoria a Corto y Largo Plazo*. URL: https://www.linkedin.com/pulse/descubriendo-las-redes-neuronales-recurrentes-gesti%C3%B3n-jordi/ (Accedido 21-03-2026).

Red Eléctrica de España (REE) (2026). *Demanda eléctrica en tiempo real*. URL: https://demanda.ree.es/visiona/peninsula/nacionalau/total (Accedido 21-03-2026).

Rojas-Jimenez K (2025). *Análisis de Series de Tiempo*. URL: https://bookdown.org/keilor_rojas/CienciaDatos/an%C3%A1lisis-de-series-de-tiempo.html (Accedido 23-12-2025).

Ruiz Abellón MC (2026). *Apuntes de Procesos Estocásticos*. Universidad Politécnica de Cartagena (UPCT). URL: https://www.dmae.upct.es/~mcruiz/Telem06/Teoria/apuntes_procesos.pdf (Accedido 14-03-2026).

Secretaría de Estado de Comercio (2026). *DataComex: Estadísticas de comercio exterior de bienes de España y la UE*. URL: https://datacomex.comercio.es/ (Accedido 21-03-2026).

Shumway RH y Stoffer DS (2017). ARIMA models. En: *Time series analysis and its applications: with R examples*. Springer, págs. 75-163.

Srivastava N, Hinton G, Krizhevsky A, Sutskever I y Salakhutdinov R (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. En: *Journal of Machine Learning Research* 15, págs. 1929-1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

SuarezSoto (2026). *TFM_SuarezSoto*. GitHub. URL: https://github.com/SuarezSoto/TFM_SuarezSoto (Accedido 28-04-2026).

UDIT (2025). *Machine Learning vs Deep Learning: diferencias, similitudes y aplicaciones*. URL: https://www.udit.es/machine-learning-vs-deep-learning-diferencias-similitudes-y-aplicaciones/ (Accedido 07-03-2026).

UK Government (2020). *UK-EU Trade and Cooperation Agreement: Summary*. URL: https://assets.publishing.service.gov.uk/media/602cf3dbd3bf7f031ce1360e/TCA_SUMMARY_PDF_V1-.pdf (Accedido 07-04-2026).

ultralytics (2026a). *Glosario - Desvanecimiento del gradiente*. URL: https://www.ultralytics.com/es/glossary/vanishing-gradient (Accedido 12-02-2026).

ultralytics (2026b). *Glosario - Gradiente Explosivo*. URL: https://www.ultralytics.com/es/glossary/exploding-gradient (Accedido 15-02-2026).

Werbos P (1990). Backpropagation through time: what it does and how to do it. En: *Proceedings of the IEEE* 78.10, págs. 1550-1560. DOI: 10.1109/5.58337.

Wikipedia contributors (2022). *TRAMO-SEATS*. Wikipedia, la enciclopedia libre. URL: https://es.wikipedia.org/wiki/TRAMO-SEATS (Accedido 05-05-2026).

Wikipedia contributors (2026). *Red Eléctrica de España*. Wikipedia, la enciclopedia libre. URL: https://es.wikipedia.org/wiki/Red_El%C3%A9ctrica_de_Espana (Accedido 21-03-2026).

World Integrated Trade Solution (WITS) (2026). *Acerca de WITS*. URL: https://wits.worldbank.org/es/about_wits.html (Accedido 21-03-2026).

ZeroMathAI (2026). *Aprendizaje Jerárquico de Características — Cómo el aprendizaje profundo construye representaciones significativas a partir de patrones simples (Capítulo 18)*. URL: https://zeromathai.com/es/hierarchical-feature-learning-es/ (Accedido 15-02-2026).