Trabajo Fin de Máster

Herramienta de análisis para problemas de forecasting

Carolina Paz Arias Morales

Máster en Técnicas Estadísticas

Curso 2022-2023

Propuesta de Trabajo Fin de Máster

Título en galego: Ferramenta de análise para a predición de problemas
Título en español: Herramienta de análisis para problemas de forecasting
English title: Analysis tool for forecasting problems
Modalidad: Modalidad B
Autor/a: Carolina Paz Arias Morales, Universidad de Santiago de Compostela
Director/a: José Ameijeiras Alonso, Universidad de Santiago de Compostela
Tutor/a: Elena Doctor, SDGroup; Miguel Arias, SDGroup
Breve resumen del trabajo: Este trabajo desarrolla el uso de dos librerías para el análisis de series temporales en Python: *Sktime* y *Skforecast*, rescatando sus características más importantes. En el primer Capítulo, se presenta la justificación de la revisión de ambas librerías. El segundo y tercer Capítulo, describen matemáticamente los métodos de predicción a usar, seguido por una variedad de métricas para su evaluación. En el Capítulo 4 se examinan las características distintivas de ambas librerías. Por último, en el Capítulo 5 se lleva a cabo una aplicación práctica utilizando datos suministrados por la empresa SDGroup
Recomendaciones:
Otras observaciones:

Don José Ameijeiras Alonso, Profesor ayudante Doctor del Departamento de Estadística, Análisis Matemático y Optimización de la Universidad de Santiago de Compostela, doña Elena Doctor, Data Science de SDGroup, y don Miguel Arias, Data Science de SDGroup, informan que el Trabajo Fin de Máster titulado

Herramienta de análisis para problemas de forecasting

fue realizado bajo su dirección por don/doña Carolina Paz Arias Morales para el Máster en Técnicas Estadísticas. Estimando que el trabajo está terminado, dan su conformidad para su presentación y defensa ante un tribunal.

En Santiago de Compostela, a xx de Enero de 2024.

El/la director/a:
Don/doña José Ameijeiras Alonso

El/la tutor/a:
Don/doña Elena Doctor

El/la tutor/a:
Don/doña Miguel Arias

El/la autor/a:
Don/doña Carolina Paz Arias Morales

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.

Índice general

Capítulo 1

Contexto y justificación

En el ámbito de la ciencia de datos, muchos proyectos se centran en la manipulación y análisis de datos temporales, los cuales son ubicuos en diversas disciplinas como finanzas, salud, meteorología y más, dado que involucran múltiples variables interrelacionadas que evolucionan con el tiempo. Dentro de este contexto, para la empresa colaboradora SDGroup surge la necesidad de contar con modelos precisos y eficientes para su análisis y por ende se hace imperativo el acceso a herramientas especializadas que permitan el manejo de los datos de manera sencilla y precisa.

En la práctica, los proyectos de ciencia de datos tienen plazos ajustados que pueden limitar la exploración exhaustiva de múltiples enfoques de modelado desde el inicio. La selección del método más adecuado para abordar un problema de series temporales a menudo se ve restringida por estas limitaciones de tiempo. Por ende, es crucial contar con una herramienta que permita comparar y evaluar diversos enfoques de manera eficiente, sin sacrificar la calidad del análisis ni la exhaustividad en la consideración de opciones viables.

El desarrollo de un *framework* basado en la revisión de librerías proporciona una solución estratégica para estas problemáticas. Este *framework* permitiría la comparación y evaluación sistemática de distintos enfoques y técnicas para el análisis de series temporales. Al seguir las mejores prácticas de la industria y aprovechar las funcionalidades de librerías especializadas como *Sktime* y *Skforecast*, se puede ofrecer una plataforma que permita explorar y comparar rápidamente múltiples métodos de análisis, seleccionando aquellos que mejor se adapten a las características específicas del problema y los límites de tiempo del proyecto. Esto posibilitaría un proceso de toma de decisiones más informada y ágil en la selección de enfoques adecuados, manteniendo altos estándares de calidad en la resolución de problemas de series temporales.

Estos dos puntos hacen que sea interesante para la empresa disponer de una herramienta que, siguiendo las buenas prácticas que se consideran imprescindibles en este tipo de proyectos, compare distintos enfoques, de forma que podamos ser menos restrictivos en ese sentido pero teniendo la capacidad de adaptación a los tiempos de los proyectos.

Capítulo 2

Revisión de los métodos

En este capítulo se introducirá el concepto de series temporales y de algunos de sus diferentes modelos de predicción. En la primera sección, se abordarán los conceptos fundamentales de las series temporales, comenzando con la definición misma y discutiendo los aspectos clave para trabajar con ellas. Al comprender estas definiciones, estaremos equipados para adentrarnos en el análisis más detallado de las series temporales y sus métodos de predicción. En la segunda parte del capítulo, se estudiarán y compararán algunos de los diversos métodos de modelado y predicción.

2.1. Justificación de elección de modelos

El análisis de series temporales es una disciplina matemática y estadística esencial que se centra en la comprensión y modelado de datos secuenciales a lo largo del tiempo. Estas secuencias de datos, conocidas como series temporales, se encuentran en una amplia gama de campos, ofreciendo información valiosa sobre patrones, tendencias y comportamientos que cambian con el tiempo.
Por lo anterior, resulta beneficioso estudiar y desarrollar métodos de predicción de series temporales para así gestionar una toma de decisiones informada a partir de las predicciones lo más precisas posibles, las cuales permiten a las empresas y organizaciones planificar estrategicamente a largo plazo, optimizar recursos muchas veces limitados, evaluar riesgos, entre otros.

Los métodos elegidos para este trabajo son: ARIMA (Autoregressive Integrated Moving Average), ETS (Error-Trend-Seasonality), Suavización exponencial, Prophet, XGBoost de ML y Perceptrón multicapa de NN. Cuya elección se basa en la complementariedad de sus fortalezas.
Los métodos clásicos ETS y ARIMA son fundamentales debido a su amplia aplicación en la modelización de series temporales. Estos enfoques han sido ampliamente utilizados y estudiados durante décadas, lo que les otorga una sólida base teórica y práctica. ETS es versátil, se destaca por su capacidad para descomponer y modelar los componentes de error, tendencia y estacionalidad presentes en las series temporales, lo que permite capturar patrones complejos y variaciones en los datos (Peña, 2005).
Por otro lado, ARIMA es conocido por su capacidad para manejar datos estacionarios y no estacionarios, lo que lo convierte en una herramienta versátil para modelar diferentes tipos de series, incluyendo aquellas con tendencias y patrones estacionales. De hecho, Peña menciona que hay una amplia gama de casos que pueden ser cubiertos por procesos integrados (2005, Capítulo 6), por ende, suelen ser útiles en muchas situaciones.
Ambos métodos proporcionan una especie de base para comprender y modelar una amplia gama de comportamientos presentes en las series temporales, conviertiendolos en pilares importantes tanto para su análisis como para su predicción. Estas característcicas permiten establecer puntos de comparación con otros enfoques más modernos o especializados en la predicción de series temporales.

El método de Suavización exponencial se enfoca en la adaptabilidad a diferentes patrones. Es especialmente útil para datos con patrones de tendencia y estacionalidad, ofreciendo métodos como Holt-Winters que se adaptan a diferentes comportamientos, incluyendo datos con ruido (Hyndman y Snyder, 2008, Capítulo 2).
Prophet, desarrollado por Facebook, es robusto para manejar datos con estacionalidad, efectos de días festivos y cambios en la tendencia. Es útil cuando se trabaja con datos de series temporales con múltiples fuentes de incertidumbre sin la necesidad de espaciar regularmente las mediciones (Taylor y Letham, 2018).

En cuanto a aprendizaje automático o ML, se escogió XGBoost, abreviatura de *Extreme Gradient Boosting*, la cual es una técnica de ML basada en árboles de decisión que se ha destacado como una de las herramientas más usadas en el campo de la ciencia de datos y ML. Esta técnica no es específicamente un modelo diseñado exclusivamente para series temporales, sino que se utiliza para problemas de regresión y clasificación. Su capacidad para aprender patrones complejos provenientes de múltples predictores y realizar predicciones precisas lo hace relevante en el contexto de las series temporales (Amat, 2020).
XGBoost se distingue por su capacidad para mejorar las predicciones al considerar variables externas o características adicionales. A través de la integración de relaciones no lineales, puede capturar patrones no triviales en los datos, lo que resulta útil cuando se busca mejorar la precisión predictiva al incorporar información adicional que pueda influir en la serie (Amat, 2020). Además, su capacidad para manejar grandes volúmenes de datos y paralelizar ciertas partes del algoritmo lo convierten en una herramienta atractiva en el análisis de series temporales.

Finalmente, las Redes Neuronales (NN), con énfasis en arquitecturas especializadas como las LSTM (Memoria a Corto y Largo Plazo) y otros modelos diseñados específicamente para abordar series temporales, destacan por su capacidad para comprender y aprender relaciones complejas en los datos temporales. En particular, las LSTM, son una variante de las redes neuronales recurrentes (RNN) que han sido especialmente diseñadas para manejar secuencias temporales y datos con dependencias a largo plazo. Su capacidad para recordar y aprender patrones de datos a largo plazo las hace valiosas en el contexto de series temporales donde existen patrones no lineales o relaciones complejas que evolucionan a lo largo del tiempo (Géron, 2022).
El autor Géron (2022), recalca que estas redes neuronales son útiles cuando se enfrentan patrones temporales que no pueden ser capturados fácilmente por métodos más tradicionales, al ser capaces de aprender de manera adaptativa a partir de los datos. Por tanto, se convierten en una herramienta poderosa para modelar y predecir comportamientos complejos, no lineales o con dependencias temporales extensas.

Al seleccionar este conjunto diverso de modelos, se busca abordar una amplia gama de comportamiento de los datos temporales, evitando la redundancia al aprovechar las fortalezas específicas de cada enfoque con el fin de mejorar la precisión predictiva y la capacidad de generalización en diferentes contextos.

2.2. Introducción a las series temporales

A continuación, exploraremos las definiciones matemáticas fundamentales relacionadas con las series temporales. Comenzaremos examinando los conceptos básicos:

Definición 1.1. Consideremos el proceso estocástico $\{X_t\}_{t\in\mathbb{Z}}$, donde t representa el instante de tiempo en el que es observada cada variable aleatoria. Una observación de dicho proceso se conoce como una realización o trayectoria y es denotada por $\{x_t\}_{t\in\mathbb{Z}}$.

Consideraremos como serie de tiempo $\{x_t\}_{t\in\mathbb{Z}}$ a un trozo finito de una trayectoria, definiéndola como una realización parcial de un proceso estocástico en intervalos regulares $1, 2, \ldots, T$.
Dado que el objetivo de estudiar una serie temporal es determinar qué tipo de proceso estocástico la ha generado, para caracterizar su comportamiento y poder hacer predicción, se hace necesario que la estructura probabilística del proceso generador sea estable en el tiempo, característica conocida como estacionariedad.
Según Peña (2005, Capítulo 1), una serie estacionaria es aquella que oscila alrededor de un nivel constante. En cambio, las series no estacionarias además de tener un nivel no constante pueden albergar un comportamiento repetitivo a lo largo del tiempo llamado comúnmente como estacionalidad. Podemos desglosar las principales características de una serie temporal como sigue:
Tendencia: es el comportamiento a largo plazo de la serie respecto a su nivel. Cuando existe una variación de aumento en el valor medio de la serie, la tendencia se dice creciente. Estacionalidad: es el comportamiento periódico de la serie, de frecuencia fija. La existencia de un patrón repetitivo es debido a que la serie se ve afectada por factores estacionales como el mes del año o día de la semana. Heterocedasticidad: es la presencia de variabilidad no constante. Si la dispersión de los datos alrededor de la media es uniforme a lo largo del tiempo, la serie se dice homocedástica.
Por lo tanto, consideraremos una serie estacionaria cuando esta no presente tendencia ni estacionalidad y sea homocedástica. Estas características se pueden observar en el gráfico secuencial de la serie en estudio, donde se representa cada observación x_t frente al instante t en el que es observada.

Para presentar los modelos que analizaremos necesitamos las siguientes definiciones, las cuales se se pueden encontrar en Aneiros (2023). Dado un proceso estocástico $\{X_t\}_{t\in T}$ se definen:
Definición 2.1. Función de medias de $\{X_t\}_{t\in T}$ a $\mu_t = \mathbb{E}[X_t]$, con $t \in T$, la cual es una medida de posición central de la serie temporal.
Definición 2.2. Función de varianzas de $\{X_t\}_{t\in T}$ a $\sigma_t^2 = Var[X_t]$, con $t \in T$, la cual mide la variabilidad de la serie temporal.
Definición 2.3. Función de autocovarianzas de $\{X_t\}_{t\in T}$ a

$$\gamma_{t_1,t_2} = Cov(X_{t_1}, X_{t_2}) = \mathbb{E}[(X_{t_1} - \mu_{t_1})(X_{t_2} - \mu_{t_2})].$$

Esta función mide la dependencia lineal entre la serie en el instante anterior t_1 y el instante t_2.

Las siguientes dos funciones son particualarmente interesantes, pues usaremos sus gráficas para definir la estacionalidad de las series con datos reales:
Definición 2.4. Función de autocorrelación simple (fas) a

$$\rho_{t_1,t_2} = \frac{\gamma_{t_1,t_2}}{\sqrt{\sigma_{t_1}^2 \sigma_{t_2}^2}},$$

la cual mide la dependencia lineal existente entre X_{t_1} y X_{t_2} y toma sus valores en el intervalo $[-1, 1]$.
Si se sustrae el efecto lineal que ejercen las variables entre los instantes t_1 y t_2 sobre X_{t_1} y X_{t_2}, se puede medir la dependencia linal entre X_{t_1} y X_{t_2} con la siguiente función:
Definición 2.5. Función de autocorrelaciones parciales (fap) a

$$\alpha_{t_1,t_2} = \frac{Cov(X_{t_1} - \hat{X}_{t_1}^{(t_1,t_2)}, X_{t_2} - \hat{X}_{t_2}^{(t_1,t_2)})}{\sqrt{Var(X_{t_1} - \hat{X}_{t_1}^{(t_1,t_2)}, X_{t_2} - \hat{X}_{t_2}^{(t_1,t_2)})}},$$

donde $\hat{X}_{t_1}^{(t_1,t_2)}$ y $\hat{X}_{t_2}^{(t_1,t_2)}$ denotan al mejor predictor lineal de X_{t_1} y X_{t_2} respectivamente, construidos mediante la regresión lineal tanto de X_{t_1} como de X_{t_2} respecto a las otras variables tomadas en el intervalo $[t_1, t_2]$ que minimizan la suma de los cuadrados de las diferencias entre los valores observados y los valores predichos por el modelo. Al igual que antes, esta función toma sus valores en el intervalo $[-1, 1]$.

Estas funciones nos ofrecen perspectivas sobre el proceso estocástico que crea la serie temporal, en lugar de revelar directamente información acerca de la serie en sí. Es por esto que estas funciones están ligadas al proceso $\{X_t\}_{t\in\mathbb{Z}}$, el cual no conocemos directamente y, por ende, necesitamos estimarlas a partir de los datos disponibles. Para realizar la estimación nos apoyamos en las versiones muestrales de las funciones definidas anteriormente, las cuales son las siguientes:
Media muestral: $\bar{x} = \frac{1}{T}\Sigma_{t=1}^{T} x_t$.
Función de autocovarianzas muestrales: $\hat{\gamma}_k = \frac{1}{T}\Sigma_{t=1}^{T-k}(x_t - \bar{x})(x_{t+k} - \bar{x})$, con $\hat{\gamma}_k = \hat{\gamma}_{-k}$ para todo $k = 0, 1, \ldots, T-1$.
Función de autocorrelaciones simples muestrales: $\hat{\rho}_k = \frac{\hat{\gamma}_k}{\hat{\gamma}_0}$.
Función de autocorrelaciones parciales muestrales: $\hat{\alpha}_k = \hat{\alpha}_{kk}$, donde $\hat{\alpha}_{kk}$ es el estimador mínimo cuadrático de α_{kk} en la regresión $x_t = \alpha_{k0} + \alpha_{k1}x_{t-1} + \ldots + \alpha_{kk}x_{t-k} + \epsilon$.

Las funciones recién expuestas ofrecen información sobre la estructura y la dependencia temporal, por lo que son herramientas para entender y modelar los procesos subyacentes a los datos secuenciales a lo largo del tiempo. Al estar basadas en conceptos como la autocorrelación y la autocorrelación parcial, proporcionan perspectivas sobre patrones, tendencias y comportamientos dinámicos en las series temporales.
Al analizar la autocorrelación simple, obtenemos una medida directa de la relación entre una observación y sus valores anteriores a intervalos específicos de tiempo. Por otro lado, la autocorrelación parcial elimina la influencia de las observaciones intermedias, lo que nos permite identificar la relación directa entre dos observaciones, controlando las correlaciones con los puntos intermedios. Esto nos permite entender cómo cada observación se relaciona con sus valores anteriores y también para la construcción de modelos. Los modelos AR (Autorregresivos) y MA (Medias Móviles) son justamente dos de los modelos fundamentales que utilizamos para modelar series temporales basándonos en esta comprensión.
Los modelos AR utilizan la información de las observaciones pasadas para predecir el valor actual, a través de una combinación lineal de observaciones previas y un término de error. Por otro lado, los modelos MA consideran una combinación lineal de errores de predicción anteriores para predecir el valor actual. Al basarnos en la estructura de dependencia y autocorrelación identificada a través de las funciones como fas y fap, ahora estamos en condiciones de definir estos modelos más detalladamente como sigue:
Definición 2.6. Un proceso estacionario $\{X_t\}_{t\in T}$ tal que

$$X_t = c + \phi_1 X_{t-1} + \ldots + \phi_p X_{t-p} + a_t,$$

donde $c, \phi_1, \ldots, \phi_p$ son constantes y $\{a_t\}_t$ ruido blanco[1] se denomina proceso autorregresivo de orden p o $AR(p)$.
Aquí, la función de autocorrelaciones parciales es cero para todo retardo mayor que p. Concretamente, se tiene que $\hat{\sigma}_k$ debería pertenecer al intervalo $\left(-\frac{1{,}96}{\sqrt{T}}, \frac{1{,}96}{\sqrt{T}}\right)$ obtenido como intervalo de confianza considerando la distribución normal asintótica con $\alpha = 0{,}05$, donde T es la cantidad de observaciones en la serie temporal.
Definición 2.7. Un proceso estacionario $\{X_t\}_{t\in T}$ tal que

$$X_t = c + a_t + \theta_1 a_{t-1} + \ldots + \theta_q a_{t-q},$$

donde $c, \theta_1, \ldots, \theta_q$ son constantes y $\{a_t\}_{t\in T}$ ruido blanco, se denomina proceso de medias móviles de orden q o $MA(q)$. En este caso, la fas se anula para todo retardo mayor que q.

Estos modelos AR y MA se combinan en el modelo ARMA (Autorregresivo de Medias Móviles), que integra tanto la dependencia de las observaciones anteriores como la dependencia de los errores de predicción pasados para modelar la serie temporal. Como bien menciona Peña (2005, Capítulo 5), si

[1] Ruido blanco: proceso estocástico $\{a_t\}_t$ cuyas variables aleatorias son incorreladas, con media cero y varianza finita.

añadimos estructura de un proceso MA a un proceso auto regresivo AR o viceversa, estamos presentes ante un proceso mixto autorregresivomedia móvil de orden (p, q) o $ARMA(p, q)$. Así, se define entonces como sigue:

Definición 2.8. Un proceso estocástico estacionario $\{X_t\}_{t\in T}$ será $ARMA(p, q)$ si permite la representación

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

donde $c, \phi_1, \ldots, \phi_p, \theta_1, \ldots, \theta_q$ son constantes y $\{a_t\}_{t\in T}$ ruido blanco.

Definición 2.9. Consideraremos a B como el operador retardo tal que $BX_t = X_{t-1}$, $B^2X_t = X_{t-2}, \ldots$. Con ella podemos reescribir la ecuación de un proceso $ARMA(p, q)$ como

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

donde $\phi(B) = 1 - \phi_1 B - \phi_2 B^2 - \ldots - \phi_p B^p$ y $\theta(B) = 1 + \theta_1 B + \theta_2 B^2 + \ldots + \theta_q B^q$.

Para modelar un modelo $ARMA(p, q)$ con el fin de hacer predicción, nos guiaremos por los pasos referidos en Peixeiro (2022, Capítulo 6), los cuales aparecen ilustrados en la Figura 2.1:

1. Probar la estacionariedad: si su media, varianza y autocorrelación no cambian con el tiempo en el gráfico secuencial y/o aplicando la prueba de Dickey-Fuller aumentada (ADF), la cual establece como hipótesis nula que está presente una raíz unitaria y por lo tanto serie temporal no es estacionaria. Si la serie es heterocedástisca aplicar transformaciones Box-Cox[2] para estabilizar la varianza.
2. Definir una lista de posibles valores para p y q, a partir de simulación. Un ejemplo de esto en Python es definir previamente una secuencia de números enteros en un rango específico con la función *range* de la librería base para los valores de p y q, para luego generar una lista con la función *list* que contenga todas las combinaciones únicas de (p, q) con la función *product* de la librería *Intertools*, como se muestra en Peixeiro (2022).
3. Ajustar cada combinación ARMA(p,q) usando la muestra.
4. Seleccionar el modelo ARMA(p,q) con el criterio de información AIC[3] más bajo.
5. Realizar el análisis de los residuos del modelo: comprobar que son ruido blanco, es decir, que sean incorreladas de media cero y varianza finita. Sino, volver al paso dos, ajustando los valores de p y q.
6. Realizar predicciones con el modelo ARMA ajustado en el paso cuatro.

La elección del modelo adecuado depende de la naturaleza de los datos y la estructura subyacente de la serie temporal. Estos modelos se utilizan ampliamente en la predicción, el análisis y la comprensión de una amplia gama de fenómenos que evolucionan con el tiempo. Lamentablemente, no abundan series reales generadas por procesos estacionarios ya que suelen presentar tendencia y/o patrones repetitivos. Si a pesar de aplicar transformaciones como la transformación Box-Cox, aún persiste la falta de estacionalidad en los datos de una serie temporal, se pueden considerar otras estrategias o enfoques para abordar este problema como la diferenciación, que tiene como objetivo principal hacer que la serie temporal sea estacionaria, simplificando el modelado y dando paso un nuevo modelo $ARIMA$, cuyo proceso se puede encontrar detalladamente en Peña (2005, Capítulo 4).

2.2.1. Modelos ARIMA

Nos centraremos en un proceso no estacionario cuando no lo es en media, en varianza y en las autocorrelaciones. Cuando el nivel de la serie cambia a lo largo del tiempo dando origen a una tendencia se

[2] La transformación Box-Cox es una técnica utilizada para estabilizar la varianza y hacer que los datos se aproximen más a una distribución normal. Esta transformación es una familia de funciones que se define mediante la fórmula: $y(\lambda) = \begin{cases} \frac{y^\lambda - 1}{\lambda} & \text{si } \lambda \neq 0, \\ \log(y) & \text{si } \lambda = 0. \end{cases}$, donde y es la variable a transformar y λ es el parámetro de transformación, según Brockwell y Davis (1991).

[3] Criterio de información de Akaike o AIC es una medida de la calidad de un modelo en relación con otros, usado para la selección de modelos y definido como $AIC = 2k - 2ln(\hat{L})$, con k número de parámetros en el modelo, y el valor $\hat{L}$ de la función de máxima verosimilitud, Peixeiro (2022, Capítulo 6).

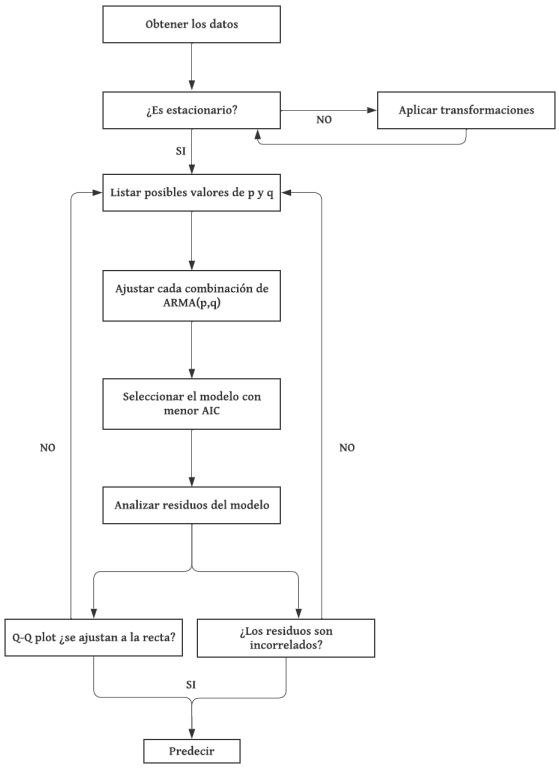

Figura 2.1: Algoritmo para modelar con un modelo $ARMA(p,q)$ una serie temporal. Fuente: Peixeiro (2022, Capítulo 6), traducida.

pueden aplicar sucesivamente diferenciaciones regulares para eliminar dicha característica.
Definición 2.10. Dado un proceso estocástico $\{X_t\}_{t\in T}$, aplicar el operador diferencia ∇ de orden 1 se define como:

$$\nabla X_t = X_t - X_{t-1}.$$

En general,

$$\nabla^d = (1-B)^d = \nabla(\nabla^{d-1}) = (1-B)(1-B)^{d-1},$$

donde B es el operador retardo (definición 2.9) y d el orden de integración, el cuál es el número de diferencias regulares necesarias para obtener un proceso estacionario. Por tanto, si $d=0$ el proceso es $ARMA$. Si la serie diferenciada es estacionaria, la modelizaremos mediante un $ARMA(p,q)$.
Definición 2.11. Un proceso $ARIMA(p,d,q)$ es aquel que se convierte en un $ARMA(p,q)$ después de aplicar d diferenciaciones regulares. 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 y además se cumple la siguiente propiedad:

$$\{X_t\}_{t\in T} \ \ es \ \ ARIMA(p,d,q) \Leftrightarrow (1-B)^d X_t \ \ es \ \ ARMA(p,q).$$

En la práctica, siguiendo lo planteado por Aneiros (2023), propondremos un modelo ARIMA cuando la no estacionariedad esté provocada únicamente por la presencia de tendencia. Dicha tendencia se podrá observar tanto en el gráfico secuencial de la serie y en la fas, la cual tendrá un decrecimiento lineal, tomando valores positivos cercanos a 1 en los primeros retardos que luego decreceran lentamente.

Este modelo si bien captura no estacionariedades generadas por presencia de tendencia, no lo hace para las generadas por presencia de componente estacional, la cual hace que la media de los datos no sea constante en el tiempo, sin embargo tiene un comportamiento cíclico de período s. Hasta ahora los procesos $ARMA(p,q)$ no estacionales modelizan la dependencia regular entre X_t y observaciones consecutivas del pasado inmediato.
Si se quiere modelizar la dependencia estacional entre X_t y observaciones en instantes múltiplos del período estacional s se da paso a los $ARMA$ estacionales.
Definición 2.12. Llamaremos al modelo ARMA estacional o $ARMA(P,Q)_s$ a un proceso estacionario $\{X_t\}_{t\in T}$ tal que

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

donde $c, \Phi_1, \ldots, \Phi_P, \Theta_1, \ldots, \Theta_Q$ son constantes y $\{a_t\}_{t\in T}$ ruido blanco.
Equivalentemente, podemos escribir

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

donde $\Phi(B^s) = (1 - \Phi_1 B^s - \Phi_2 B^{2s} - \ldots - \Phi_p B^{Ps})$, $\Theta(B^s) = (1 + \Theta_1 B^s + \Theta_2 B^{2s} + \ldots + \Theta_Q B^{Qs})$, y B^s operador retardo estacional defnido como $B^s X_t = X_{t-s}$.
Cabe mencionar que un $ARMA(P,Q)_s$ es, en particular, un $ARMA(sP, sQ)$ con muchos coeficientes nulos. Por tanto, las condiciones de estacionariedad, causalidad e invertibilidad se deducen de las de los $ARMA(p,q)$ no estacionales.

Peña (2005) menciona en el Capítulo 7 que si existen procesos tanto con dependencia regular como estacional, se pueden modelizar de forma sencilla como procesos ARMA estacionales multiplicativos o $ARMA(p,q)\times(P,Q)_s$, el cual modeliza de forma separada ambas dependencias para luego incorporarlas de forma multiplicativa.
Definición 2.13. Llamaremos al modelo ARMA estacional multiplicativo o $ARMA(p,q)\times(P,Q)_s$ a un proceso $\{X_t\}_{t\in T}$ tal que

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

Un modelo ARIMA estacional multiplicativo se basa en la hipótesis de relación de dependencia estacional igual para todos los periodos de tiempo. Sin embargo, y como menciona Peña (2005) en el Capítulo 7, esto no sucede con frecuencia en la práctica y se debe disponer de una muestra suficientemente grande para constrastarla construyendo modelos $(1 - \Phi_1 B^s - \ldots - \Phi_P B^{sP})(1 - B^s)X_t = (1 - \Theta_1 B^s - \ldots - \Theta_Q B^{sQ})a_t$ y viendo si presentan la misma estructura.

La función de autocorrelación simple para este modelo es una combinación de la fas de la parte regular y de la parte estacional. Es por esto, que en los retardos bajos $(1, 2, \ldots, [s/2])$ se observará la fas de la parte regular del modelo y en los retardos estacionales $(s, 2s, 3s, \ldots)$ la fas de la parte estacional. A ambos lados de los retardos estacionales se repetirá la fas de la parte regular, es decir, si la parte regular es un $MA(q)$ habrá a ambos lados de cada retardo estacional q coeficicentes no nulos y si es un $AR(p)$ habrá un decrecimiento impuesto por la estructura AR.
Algo similar ocurre con la función de autocorrelación parcial, pero a la derecha de cada retardo estacional aparecerá la fap de la parte regular (invertida, si la fap en el retardo estacional es positiva) y a la izquierda aparecerá la fas de la parte regular (invertida, si la fap en el retardo estacional es negativa).

Finalmente, se presenta el modelo ARIMA estacional multiplicativo o $ARIMA(p,d,q)\times(P,D,Q)_s$, el cual viene a generalizar todos los modelos vistos anteriormente, puesto que son capaces de capturar no estacionalidades provocadas tanto por la presencia de tendencia ($d > 0$), como por la componente estacional ($s > 1$ y $D > 0$) y modelar la dependencia regular (p o $q \neq 0$) y estacional ($s > 1$, y P o $Q \neq 0$). Características por las cuales son posiblemente los más usados en el contexto univariante.

Definición 2.14. Llamaremos al modelo ARIMA estacional multiplicativo o $ARIMA(p,d,q)\times(P,D,Q)_s$ a un proceso a un proceso $\{X_t\}_{t\in T}$ al que después de aplicarle d diferencias regulares y D diferencias estacionales de periodo s, se corvierte en un proceso $ARMA(p,q)\times(P,Q)_s$ y admite la siguiente representación:

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

donde el polinomio $\phi(z)\Phi(z^s)$ no tiene raíces de módulo 1.
En este caso, la fas presenta una fuerte correlación positiva en el retardo estacional y posiblemente en sus múltiplos, decreciendo linealmente a cero a medida que aumenta el retardo y tiene periodicidad del mismo periodo que la serie.
Dejamos a modo de resumen la Figura 2.2 donde se indican los pasos a seguir para realizar este ajuste con el fin de realizar predicciones.

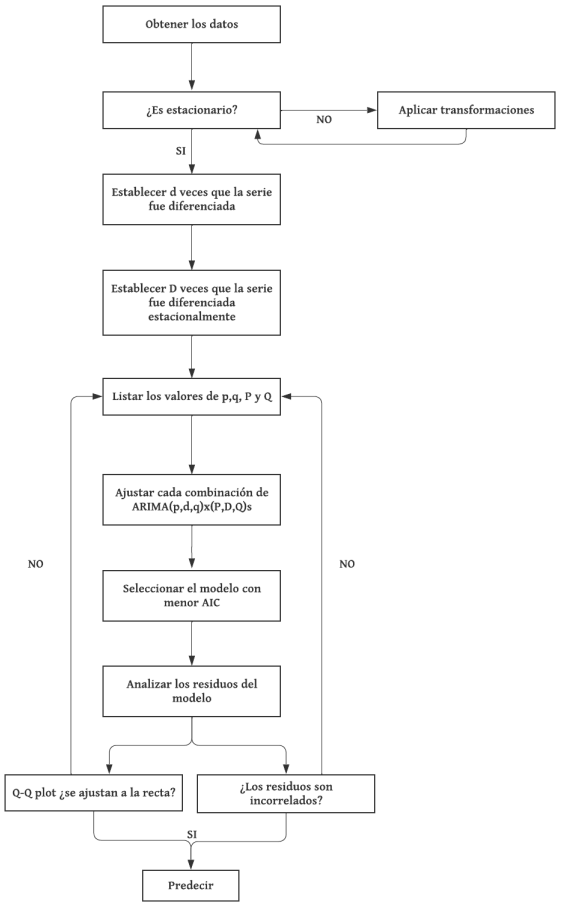

Figura 2.2: Algoritmo para modelar con un modelo $ARIMA(p,d,q)\times(P,D,Q)_s$ una serie temporal. Fuente: Peixeiro (2022, Capítulo 8), traducida.

Predicciones

La importancia de considerar tanto la predicción puntual como la predicción por intervalos en procesos ARIMA estacionales y no estacionales radica en la necesidad de comprender la incertidumbre asociada con las predicciones de series temporales.

La predicción puntual proporcionan un único valor estimado para el futuro de la serie temporal en un punto específico. Esto es valioso para tener una idea general de la dirección y magnitud de las tendencias previstas. Sin embargo, estas predicciones puntuales no transmiten información sobre la incertidumbre asociada a la predicción. Pueden subestimar la variabilidad y no ofrecer una visión completa del panorama de posibles resultados.
La predicción por intervalos considera la incertidumbre alrededor de la predicción puntual, brindando un intervalo dentro del cual se espera que caiga el valor futuro con cierto nivel de confianza. Proporcionan una comprensión más completa de la variabilidad y la incertidumbre en las predicciones. Estos intervalos son herramientas útiles para la toma de decisiones, ya que permiten evaluar la incertidumbre y el riesgo asociado con las predicciones.
En modelos ARIMA, la incorporación de predicciones por intervalos es fundamental debido a la naturaleza estocástica y dinámica de las series temporales. En particular, los modelos ARIMA estacionales y no estacionales pueden presentar comportamientos complejos y variaciones que no se capturan completamente en una única predicción puntual.

Predicción puntual
Para analizar la predicción de los modelos ya mencionados nos basaremos en las predicciones del modelo ARIMA no estacional, pues se sigue la misma idea para todos los modelos anteriores.
Una vez ajustado un modelo $ARIMA(p,d,q)$ con parámetros conocidos, el cuál ha superado el análisis de residuos[4], podemos realizar la predicción de valores futuros con origen en T a horizonte h denotada por $\hat{x}_T(h)$.

Para ilustrar el procedimiento, nos guiaremos de un ejemplo con un modelo ARIMA(2,1,1) a horizonte $h = 2$. En principio, expandimos la representación del modelo ajustado, como sigue:

$$(1 - \hat{\phi}_1 B - \hat{\phi}_2 B^2)(1 - B)\hat{x}_t = \hat{c} + (1 + \hat{\theta}_1 B)\hat{a}_t,$$

que es equivalente a

$$\hat{x}_t = (1 + \hat{\phi}_1)\hat{x}_{t-1} - (\hat{\phi}_1 - \hat{\phi}_2)\hat{x}_{t-2} - \hat{\phi}_2\hat{x}_{t-3} + \hat{c} + \hat{a}_t + \hat{\theta}_1\hat{a}_{t-1}.$$

Considerando $t = T + h$, con h horizonte de predicción, la expresión anterior es equivalente a

$$\hat{x}_{T+h} = (1 + \hat{\phi}_1)\hat{x}_{T+h-1} - (\hat{\phi}_1 - \hat{\phi}_2)\hat{x}_{T+h-2} - \hat{\phi}_2\hat{x}_{T+h-3} + \hat{c} + \hat{a}_{T+h} + \hat{\theta}_1\hat{a}_{T+h-1}.$$

Como hemos considerado $h = 2$, reemplazando este valor tenemos la siguiente ecuación:

$$\hat{x}_{T+2} = (1 + \hat{\phi}_1)\hat{x}_{T+1} - (\hat{\phi}_1 - \hat{\phi}_2)\hat{x}_T - \hat{\phi}_2\hat{x}_{T-1} + \hat{c} + \hat{a}_{T+2} + \hat{\theta}_1\hat{a}_{T+1}.$$

A continuación, siguiendo lo expuesto en Peña (2005, Capítulo8), se reemplazan las observaciones actuales y pasadas de $\hat{x}_t$ por x_t. Por lo tanto, $\hat{x}_T$ y $\hat{x}_{T-1}$ serán reemplazadas por x_T y x_{T-1} respectivamente. Las innovaciones $\hat{a}_t$ futuras se reemplazan por la media. Este último reemplazo viene del hecho de considerar a_t como ruido blanco que tiene media $\mathbb{E}(a_t) = 0$ y no presentar ninguna información de la serie en cuestión por ser independientes. Asumiendo lo anterior, la ecuación es equivalente a

$$\hat{x}_{T+2} = (1 + \hat{\phi}_1)\hat{x}_{T+1} - (\hat{\phi}_1 - \hat{\phi}_2)x_T - \hat{\phi}_2 x_{T-1} + \hat{c},$$

donde todos los términos son conocidos, pudiendo generalizar el procedimiento para cualquier h y ordenes finitos de p y q.

En general, Peña (2005, Capítulo 8) resalta que las predicciones a horizontes altos para los procesos $ARMA(p,q)$ estacionarios tenderán hacia la media del proceso, es decir, a $\mu = c/(1-\phi_1-...-\phi_p)$. En

[4]Consideraremos que los residuos $\{a_t\}_{t\in T}$ del ajuste superan la diagnosis si pueden ser ruido blanco con distribución gaussiana. Como las innovaciones no son conocidas, los constrastes se aplicarán sobre sus estimaciones o residuos.

cambio, si tenemos un proceso ARIMA ($d = 1$) las predicciones lo harán hacia un término constante si $c = 0$ o a una recta si $c \neq 0$, es decir, las predicciones del modelo tienden a seguir una tendencia lineal a medida que se proyecta hacia el futuro.
Si aumentamos el orden de diferenciación, por ejemplo a $d = 2$, las predicciones tenderán a una recta si $c = 0$ o a una parábola si $c \neq 0$, esto significa que a medida que se proyecta en el tiempo, las predicciones del modelo exhibirán una tendencia que no es lineal ni constante, sino que sigue una forma curva o parabólica y así susecivamente[5].

Predicción por intervalos
Como bien menciona Peña (2005), en el Capítulo 8, el propósito de llevar a cabo una predicción por intervalos radica en la necesidad de contar con información acerca de la precisión de las predicciones puntuales realizadas en la sección anterior. Un acercamiento a dicha precisión son los intervalos de confianza, en este caso llamados intervalos de predicción o IP.
La construcción de dichos intervalos parten de la expresión de la serie $X_t = c+a_t+\varphi_1 a_{t-1}+\varphi_2 a_{t-2}+\ldots$ y de la distribución muestral del error de predicción a horizonte h, es decir, $\mathrm{e}_T(h) = X_{T+h} - \hat{X}_T(h)$. Si podemos asumir el valor de T como suficientemente grande y la distribución de a_t gaussiana, la distribución del error es aproximadamente normal de media 0 y de varianza $\sigma_a^2(1+\varphi_1^2+\ldots+\varphi_{h-1}^2)$, con φ_i los coeficientes de X_t, quedando el IP como sigue:

$$\hat{X}_T(h) \pm z_{1-\alpha}\sqrt{\sigma_a^2(1+\varphi_1^2+\ldots+\varphi_{h-1}^2)},$$

para un nivel α de significación.
Estos intervalos, que son estimados a partir del ajuste del modelo de la muestra disponible y, una vez hechas las predicciones junto con el calculo de los residuos del modelo para estimar la varianza del error de predicción, ofrecen una visión más completa de la variabilidad inherente en los datos y mejoran la toma de decisiones informada.

2.2.2. Modelos ETS y Suavización Exponencial

Comenzaremos esta sección con el análisis descriptivo de una serie de tiempo, siguiendo lo propuesto por Aneiros (2008), Hyndman et al. (2008) y Peña (2005). En principio, una serie temporal homocedástica puede escribirse en función de la tendencia T_t, de la componente estacional S_t y de un error a_t o E_t, .
Recordemos que T_t modeliza el comportamiento a largo plazo, S_t el comportamiento periódico y E_t es la combinación de efectos de diversos factores, que a menudo desconocemos, generando los modelos llamados ETS.
Los modelos más usados son:

- Modelo aditivo: $X_t = T_t + S_t + E_t$.
- Modelos multiplicativos:
  - Modelo multiplicativo puro: $X_t = T_t \times S_t \times E_t$.
  - Modelo mixto: $X_t = (T_t \times S_t) + E_t$ o cualquier otra combinación.

La elección del modelo será a partir del que mejor sea capaz de reunir la principales características de la serie en estudio.
De forma general, el modelo aditivo es un buen ajuste cuando la magnitud de las oscilaciones estacionales no varía al hacerlo la tendencia. En cambio, los modelos multiplicativos se ajustan mejor cuando la magnitud de las oscilaciones estacionales varían proporcionalmente con las variaciones de la tendencia. Además, la tendencia se modela generalmente mediante un polinomio del tiempo de orden menor o igual

[5]Si $d > 1$, generalmente no se incluye c en el modelo, pues una tendencia cuadrática (o de grado mayor que 2) es mucho más compleja de manejar a la hora de hacer predicciones. Revisar la sección 2 del Capítulo 8 de Peña (2005) para una demostración detallada de cada caso

a dos, y la estacionalidad como una función periódica que verifica $S_t = S_{t-s}$ con s el periódo de la serie.

La aproximación de las componentes T_t y S_t a partir de x_t se pueden hacer a partir de métodos parámetricos y no parámetricos. El primero, modeliza la relación entre estas dos componentes y el tiempo, para luego ajustar dicho modelo, por ejemplo, a través de mínimos cuadrados. El segundo, se basa en asumir suavidad en la relación entre estas dos componentes y el tiempo. Por ejemplo, suavizando a partir de la aplicación de filtros de medias móviles[6] o suaviazado exponencial.
En nuestro caso nos centraremos en estos últimos, ya que considerar la tendencia como una componente determinista puede no ser realista. En muchos casos las series no siguen un nivel lineal o polinomico respecto al tiempo, a pesar de esto, Peña (2005) en el Capítulo 2 ilustra que en estos casos se puede ajustar una tendencia polinómica por tramos y ajustar en cada trozo un modelo lineal, polinómico o constante. Sin embargo, este procedimiento tiene limitaciones porque es poco razonable asumir un crecimiento determinista lineal implícito del modelo en cada intervalo, ya que al hacerlo estariamos acordando que la predicción del crecimiento se haría a través de los crecimientos observados asignando, en el caso lineal, ponderaciones tales que el mayor peso se asigne al crecimiento en el centro del período observado y el peso mínimo al crecimiento en los extremos del intervalo. Así, si tuviesemos información de solo los últimos cinco años del crecimiento de una población los pesos asignados serían el 20 %, 30 %, 30 % y 20 % a los cuatro crecimientos anuales, pero si aumentamos la muestra a 100 datos anuales el peso pasa a ser un valor mucho menor e igual al crecimiento observado hace 100 años.

Los métodos de suavizado vienen a permitir que los últimos datos puedan tener influencia en las predicciones a través del aumento en sus ponderaciones, considerando que los parámetros de tendencias deterministas puedan ir variando en el tiempo.
Con lo anterior en mente, la manera en que definamos la evolución del nivel medio de la serie a lo largo del tiempo será determinante para la elección del modelo. Hyndman et al. (2008, Capítulo 2) define a la tendencia como una combinación de nivel (ℓ) y crecimiento (b), a menudo llamada pendiente, distinguiendo 5 tipos diferentes:

Definición 2.15. Sea T_h la tendencia a horizonte h y ϕ, $\phi \in (0, 1)$ parámetro de suavización, se definen los siguientes tipos:
Nula: $T_h = \ell$. Se espera que el nivel medio de la serie sea horizontal, en el cual la serie oscilará alrededor de su media, que es constante.
Aditiva: $T_h = \ell + bh$. Se espera un crecimiento o decrecimiento lineal con pendiente b, la cual se corresponde con el crecimiento esperado entre dos periodos.
Aditiva suavizada: $T_h = \ell + (\phi + \phi^2 + ... + \phi^h)b$. Dado que es una versión suavizada de la anterior, se espera una tendencia cada vez más pequeña, convergiendo a una horizontal a lo largo del tiempo. Asumiendo que la tasa de crecimiento al final de los datos históricos es poco probable que continúe más allá de un corto periodo de tiempo.
Multiplicativa: $T_h = \ell b^h$.
Multiplicativa suavizada: $T_h = \ell b^{(\phi+\phi^2+...+\phi^h)}$.

Una vez escogido el tipo de tendencia podemos añadir S_t de forma aditiva o multiplicativa, la cual está definida completamente a partir del periodo estacional s. El añadir los errores no supone una tarea ardua, pues no crea ninguna diferencia en las predicciones si los ingresamos de forma aditiva o multiplicativa. De hecho, a menudo son ignorados en esta estapa.
En el Cuadro 2.1 podemos ver un resumen de las quince combinaciones posibles de tendencia y estacionalidad que dan lugar a los métodos de suavizado exponencial. Para cada uno de los 15 métodos, hay dos modelos de espacio de estados posibles a partir de la añadidura de la componente E_t. Como esta añadidura se puede realizar de dos formas diferentes (aditiva o multiplicativa) tenemos en realidad 30

[6] Aplicar a la serie x_t un filtro de medias móviles con coeficientes $[b_{-k}, \ldots, b_{-1}, b_0, b_1, \ldots, b_k]$ consiste en obtener los valores $\hat{x}_t = b_{-k}x_{t-k} + \ldots + b_{-1}x_{t-1} + b_0x_t + b_1x_{t+1} + \ldots + b_kx_{t+k}$.

modelos, los cuales dan lugar a pronósticos equivalentes pero con distintos intervalos de predicción.

Tendencia/Estacionalidad	**Nula** N	**Aditiva** A	**Multiplicativa** M
Nula N	NN	NA	NM
Aditiva A	AN	AA	AM
Aditiva suavizada A_d	A_dN	A_dA	A_dM
Multiplicativa M	MN	MA	MM
Multiplicativa suavizada M_d	M_dN	M_dA	M_dM

Cuadro 2.1: Combinaciones posibles de T_t y S_t para la generación de métodos de suavizado exponencial. Fuente: Hyndmanet al. (2008), Capítulo 2.

Una consideración importante que hace Hyndman et al. (2008), en el mismo capítulo, es la diferencia que existe entre métodos de suavizado y modelos de espacio de estados. Un método de suavizado lo define como un algoritmo para predecir puntualmente. Por otra parte, un modelo de espacio de estados además nos proporciona un marco para calcular intervalos de predicción, al añadir la componente E, que desarrollaremos un poco más adelante.
A continuación describiremos algunos de los métodos de suavizado exponencial más conocidos, siguiendo lo expuesto en Hyndman et al. (2008, Capítulo 2), para lo cual llamaremos vector de elementos no observables en el instante t, a $v_t = (l_t, b_t, s_t)$.

Suavizado Exponencial Simple (N,N):
Este método tiene una tendencia nula $T = l$ y carece de componente estacional ($x = T$, T nula). En este caso, solo interviene el elemento no observable nivel (l). El nivel en el instante t, l_t, se determinará como una combinación convexa entre el nivel del instante anterior l_{t-1}, y la observación real en el instante t, para todo $t = 1, \ldots, T$ como sigue:

$$l_t = \alpha x_t + (1 - \alpha) l_{t-1}$$

con $\alpha \in [0, 1]$ que habrá que estimar.
Esto es como tomar el nivel actual de la serie y corregir del nivel anterior al añadir la nueva observación.
La predicción a horizonte h queda como

$$\hat{x}_T(h) = l_T$$

donde l_T se obtiene a partir de la siguiente iteración:

$$\begin{aligned} l_T &= \alpha x_T + (1 - \alpha) l_{T-1} \\ &= \alpha x_T + (1 - \alpha)[\alpha x_{T-1} + (1 - \alpha) l_{T-2}] \\ &= \alpha x_T + \alpha(1 - \alpha) x_{T-1} + (1 - \alpha)^2 l_{T-2} \\ &= \alpha x_T + \alpha(1 - \alpha) x_{T-1} + \alpha(1 - \alpha)^2 x_{T-2} + ... + \alpha(1 - \alpha)^{T-1} x_1 + (1 - \alpha)^T l_0 \end{aligned}$$

Podemos ver que la predicción de valores futuros es calculado como una media ponderada de todas las observaciones pasadas con sus pesos decreciendo de forma exponencial, donde nos hace falta conocer los valores de l_0 y α. Encontrar dichos valores se conoce como *problema de inicialización* y es subyacente a todos los métodos, por lo cual lo abordaremos más adelante.

Método lineal de Holt (A,N):
Este método sólo considera una tendencia aditiva $T_h = l + hb$ sin componente estacional. En este caso, intervienen dos elementos no observables: el nivel (l) y la pendiente (b). El nivel en el instante t, l_t, se calculará al igual que antes como combinación convexa entre x_t y l_{t-1}, añadiendo el efecto de la pendiente b_{t-1} como sigue:

$$l_t = \alpha x_t + (1-\alpha)(l_{t-1} + b_{t-1})$$

para todo $t = 1, \ldots, T$ y con $\alpha \in [0, 1]$. En este caso, es como considerar el nivel actual de la serie como una corrección tanto del nivel anterior como de la pendiente anterior al añadir la nueva observación.
En el instante t la pendiente b_t se calculará como una combinación convexa entre b_{t-1} y la diferencia de niveles que se produjo entre dichos instantes:

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

con $\beta \in [0, 1]$, donde la predicción a horizonte h es equivalente a

$$\hat{x}_T(h) = l_T + hb_T,$$

donde β y b_0 son desconocidos.

Método de Holt-Winter aditivo (A,A):
En este método intervienen todos los elementos no observables, es decir, el nivel (l), la pendiente (b) y la estacionalidad (s). Aquí la tendencia y la estacionalidad son aditivas.
El nivel de la serie en el instante t se calcula como en el caso anterior, pero las observaciones serán ajustadas estacionalmente.
Como la estacionalidad es añadida de forma aditiva, la serie ajustada estacionalmente es el resultado de la transformación $x - S$ que proviene de la siguiente idea: Si definimos la serie ajustada estacionalmente K como la resultante de extraer la componente estacional a la serie original $x = T + S + E$, dejando únicamente las componentes tendencia y error, para modelos aditivos bastará con calcular $K = x - S$ (para multiplicativos será $K = x/S$). Así nos queda que, en el caso aditivo, l_t como

$$l_t = \alpha(x_t - s_{t-m}) + (1-\alpha)(l_{t-1} + b_{t-1}),$$

con m la frecuencia de observación de la serie y $\alpha \in [0, 1]$.
En cuanto a la pendiente, dado que la tendencia es aditiva al igual que en el caso anterior se mantiene como:

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

con $\beta \in [0, 1]$.
El cálculo de la componente estacional en el instante t, s_t se calculará como una combinación convexa entre s_{t-m} que es la estacionalidad de ese instante en el ciclo anterior y la diferencia observada entre el valor de la serie y el valor esperado dado por la tendencia $x_t - l_{t-1} - b_{t-1}$. Es decir,

$$s_t = \xi(x_t - l_{t-1} - b_{t-1}) + (1-\xi)s_{t-m}$$

con $\xi \in [0, 1]$ desconocido, donde la predicción a horizonte h queda como sigue:

$$\hat{x}_T(h) = l_T + hb_T + s_{t-m+h_m^+},$$

con $s_0, \ldots, s_m$ desconocidos y $h_m^+ = [(h-1) \mod m] + 1$ que sería el número de ciclos enteros que han pasado durante el tiempo h.

Con estos tres métodos descritos ya podemos interpretar los demás, considerando el tipo de tendencia y el ajuste estacional de la serie, en caso de que exista. En el Apéndice sobre tablas de modelos ETS se encuentra la Figura A.1 que contiene las ecuaciones de todos los métodos de suavizado exponencial

provenientes de Hyndman et al. (2008).

Modelos de espacio de estados
Para cada método antes descrito hay dos modelos de espacios de estados dependiendo de la forma de agregación de la componente E_t: aditiva o multiplicativa. Para distinguir entre modelos seguiremos la notación presentada en Hyndman et al. (2008) en el Capítulo 2, como $ETS(.,.,.)$ donde la primera letra corresponderá a los errores y será "A" o "M"según sea la forma de añadidura. Lo mismo ocurre con la segunda y la tercera letra, donde cada una representa a la tendencia y a la estacionalidad respectivamente, con tres formas de añadidura: ninguna, aditiva o multiplicativa. Por ejemplo, el modelo $ETS(M,M,N)$ tendrá errores multiplicativos, tendencia multiplicativa y carecerá de estacionalidad.

Una vez definido un modelo ETS, podremos estudiar la distribución de probabilidad a horizonte h de nuestra serie y calcular diferentes medidas, por ejemplo, la media condicionada a horizonte h dados los últimos elementos no observables de los que disponemos. Esta media la denotaremos como $\mathbb{E}(x_{t+h}|v_t)$, con v_t vector de elementos no observables l_t, b_t y s_t.
Para varios modelos, la predicción puntual a horizonte h coincidirá con $\mathbb{E}(x_{t+h}|v_t)$. Sin embargo, para los modelos con tendencia o estacionalidad multiplicativa, la media condicional y la predicción puntual diferirán levemente dependiendo del horizonte.
A continuación estudiaremos uno de los Modelos de espacio de estados para el Método lineal de Holt (A,N), con errores aditivos, siguiendo muy de cerca lo planteado en Hyndman et al. (2008, Capítulo 2), dado que siguiendo el mismo procedimiento se pueden obtener las ecuaciones de todos los demás modelos.

Modelo con errores aditivos o ETS(A,A,N):
Sea $\hat{x}_t = l_{t-1} + b_{t-1}$ la predicción a un paso de x_t suponiendo que conocemos todos los parámetros, y sea $\varepsilon_t = x_t - \hat{x}_t$ el error de predicción a un paso en el instante t.
Recordando las ecuaciones del Método lineal de Holt tenemos:

$$l_t = \alpha x_t + (1-\alpha)(l_{t-1} + b_{t-1}),$$

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

y

$$\hat{x}_T(h) = l_T + hb_T.$$

De la tercera ecuación se tiene que, por la añadidura de los errores, nos queda la predicción en el instante t como

$$\hat{x}_t = l_{t-1} + b_{t-1} + \varepsilon_t.$$

De la cual, podemos expresar $l_{t-1} = \hat{x}_t - b_{t-1} - \varepsilon_t$ como $l_{t-1} = \hat{x}_t - b_{t-1} - (x_t - \hat{x}_t)$, lo que a su vez es equivalente a $l_{t-1} = 2\hat{x}_t - b_{t-1} - x_t$.
Además, el desarrollo de la primera ecuación es:

$$l_t = l_{t-1} + b_{t-1} + \alpha(x_t - l_{t-1} - b_{t-1}).$$

Dado que $\varepsilon_t = x_t - \hat{x}_t$ es equivalente a $\varepsilon_t = x_t - (l_{t-1} + b_{t-1})$ reemplazando el valor de $\hat{x}_t$, tenemos:

$$l_t = l_{t-1} + b_{t-1} + \alpha\varepsilon_t.$$

Por otro lado, desde la segunda ecuación, a través del álgebra se puede llegar a

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

donde reemplazando el valor de l_t desde la segunda ecuación, nos queda lo siguiente:

$$b_t = b_{t-1} + \beta[(l_{t-1} + b_{t-1} + \alpha\varepsilon_t) - l_{t-1} - b_{t-1}],$$

la cual es equivalente a

$$b_t = b_{t-1} + \alpha\beta\varepsilon_t.$$

Resumiendo, tenemos las siguientes tres ecuaciones

$$l_t = l_{t-1} + b_{t-1} + \alpha\varepsilon_t,$$

$$b_t = b_{t-1} + \alpha\beta\varepsilon_t,$$

y

$$\hat{x}_t = l_{t-1} + b_{t-1} + \varepsilon_t,$$

las cuales constituyen uno de los modelos de espacio de estados ETS subyaciente del Método lineal de Holt, cuyo modelo quedará completamente especificado cuando le otorguemos una distribución para ε_t. Normalmente, se asume que son independientes e identicamente distribuidos a $\varepsilon_t \sim N(0, \sigma^2)$.

Modelos de espacio de estados para todos los métodos
Siguiendo el mismo procedimiento podemos obtener las ecuaciones de cada uno de los 30 modelos. Dichas ecuaciones se presentan en las tablas 2 y 3 en el apéndice sobre tablas de modelos ETS, provenientes de Hyndman et al. (2008, Capítulo 2).
Un aspecto importante a considerar y como bien lo menciona Hyndman et al. (2008, Capítulo 2), es cuando la serie no está compuesta estrictamente por datos positivos, ya que podría producirse alguna división por cero, por ejemplo, para modelos que combinen errores aditivos y tendencia multiplicativa o estacionalidad multiplicativa, así también cuando los errores son añadidos de forma multiplicativa, tendencia multiplicativa y estacionalidad aditiva. Por ello, el autor recomienda aplicar solo modelos con errores aditivos cuando los datos temporales contengan ceros o valores negativos.

Hyndman et al. (2008), en el Capítulo 2, menciona que las predicciones puntuales de estos modelos se suelen obtener de forma sencilla iterando sus ecuaciones para $t = T + 1, ..., T + h$ y fijando $\varepsilon_{T+i} = 0, \forall\ i = 1, ..., h$. Donde, en la mayoría de casos, estas predicciones coincidirán con la media condicional $\mathbb{E}(x_{t+h}|v_t)$, con v_t vector de elementos no observables, a excepción de aquellos con tendencia o estacionalidad multiplicativa para horizontes mayores o iguales a 2.
Además, en el mismo Capítulo, y también en el Capítulo 6, se menciona que estos modelos también permiten la obtención de intervalos de predicción. En el caso de los modelos lineales, donde la distribución de las predicciones es Gaussiana, se puede calcular la varianza condicional $var_t(h) = Var(x_{t+h}|v_t)$ y y generar intervalos de predicción basados en ella. Los modelos no lineales también se pueden aproximar adecuadamente por una Gaussiana, por lo que el procedimiento anterior también funcionaría. Se puede encontrar más detalle en Hyndman et al. (2008), Hyndman et al. (2005) y Taylor (2003).

Inicialización y estimación
Una vez seleccionado el modelo, para su aplicación debemos estimar los parámetros que intervienen $(\alpha, \beta, \xi, \phi)$ y también los valores iniciales del vector de elementos no observables v_0 para la construcción de los métodos. Recordamos que $v_t = (l_t, b_t, s_t)$ dependerá del método empleado.
La selección de valores iniciales (v_0) generalmente suele hacerse en base a una selección "ad hoc"de valores o por medio de esquemas heurísticos. Uno de estos esquemas es la que se ilustra en Hyndman et al. (2002), la cual genera buenos resultados y describiremos a continuación:
Inicialización de la componente estacional: supone calcular una media móvil [7] para los primeros datos de la serie, denotando estos valores como $f_t,\ \ t = m/2 + 1, m/2 + 2, ...$ Recordando que si la estacionalidad es agregada de forma aditiva, se ajustará la transformación $x_t - f_t$ y si es multiplicativa x_t/f_t. Luego, se calculan los valores iniciales $s_{-m+1}, ..., s_0$ promediando los datos transformados o sin tendencia para cada estación.
Inicialización de la componente de nivel: suponiendo que ya se han obtenido los índices estacionales, se procede a calcular una tendencia lineal sobre los primeros diez valores ajustados estacionalmente

[7] La Media móvil simple de orden m de define como $\frac{1}{m}\sum_{j=-k}^{k} x_{t+j}$, con $m = 2k + 1$.

(o simplemente los primeros diez valores si no hay componente estacional) frente una variable temporal que va desde 1 hasta 10 ($t = 1, \ldots, 10$). La ecuación se puede expresar como $\hat{y}_t = \alpha'\beta' t$, donde $\hat{y}_t$ es el valor ajustado estacionalmente en el período t, t es el tiempo que va desde 1 hasta 10, α'es la intersección de la línea y β' es la pendiente de la línea, que representa la tasa de cambio promedio de la serie temporal por unidad de tiempo. Para calcular α' y β' se utiliza la regresión lineal sobre los primeros diez valores ajustados estacionalmente y sus respectivos tiempos. El valor inicial del nivel l_0 ($\hat{y}_0$) se toma como la intersección de la línea de tendencia con el eje vertical, es decir, cuando $t = 0$. Dado que estamos utilizando un índice de tiempo que comienza en 1, simplemente establecemos el nivel inicial como el valor de la intersección de la línea con el eje vertical.
Inicialización de la componente de crecimiento: si la tendencia es aditiva se fijará b_0 como la pendiente de la regresión mencionada antes. En cambio, si es multiplicativa será $b_0 = 1 + \hat{\beta}_0/\hat{\beta}_1$, con $\hat{\beta}_0$ el intercepto y $\hat{\beta}_1$ la pendiente de la regresión.

Por otro lado, Hyndman et al. (2008, Capítulo 2) menciona que la estimación de los parámetros se pueden realizar a través del criterio de máxima verosimilitud. De forma general, las ecuaciones para los modelos de espacios de estados para todos los métodos, se pueden formular como sigue:

$$x_t = \mu_t + r(v_{t-1})\varepsilon_t$$

$$v_t = f(v_{t-1}) + g(v_{t-1})\varepsilon_t$$

con $v_t = (l_t, b_t, s_t, s_{t-1}, ..., s_{t-m+1})$ traspuesto, r, f y g funciones sobre los elementos de v_t, y $\{\varepsilon_t\}$ ruido blanco gaussiano.
A partir de estas, se puede formular la función de verosimilitud de la siguiente manera:

$$\mathscr{L}^*(\theta, v_0) = n \log\left(\sum_{t=1}^{T} \varepsilon_t^2\right) + 2\sum_{t=1}^{T} \log|r(v_{t-1})|,$$

con θ vector de elementos iguales a los parámetros del modelo, los cuales pueden ser estimados a partir de la minimización de la función $\mathscr{L}^*$.

2.2.3. Prophet

A lo largo del tiempo se han desarrollado muchas bibliotecas para automatizar el proceso de pronóstico, algunas de las bibliotecas más populares son Pmdarima, Prophet, NeuralProphet y PyTorch Forecasting (Peixeiro, 2022, Capítulo 19).
En nuestro caso, nos centraremos en Prophet, que es un paquete de código abierto gratuito de Meta Open Source creado por Sean J. Taylor y Ben Lethan, que podemos encontrar en Taylor y Letham (2018), el cual se puede implementar en diferentes lenguajes de programación[8]. Esta biblioteca se creó a partir de la necesidad de producir pronósticos precisos rápidamente y posiblemente sea la biblioteca más conocida, ya que puede adaptarse a tendencias no lineales y combinar el efecto de múltiples estacionalidades.
El modelo de Prophet consiste en descomponer las series temporales en tres factores: tendencia, estacionalidad y días festivos o vacaciones, donde la tendencia no es lineal sino que se ajusta a la estacionalidad como sigue:

$$x(t) = g(t) + s(t) + h(t) + \varepsilon_t$$

donde $g(t)$ representa la tendencia, $s(t)$ la estacionalidad, $h(t)$ los efectos de las vacaciones y ε_t el error del modelo, asumido con distribución gaussiana.

A continuación, veremos los tres factores en los que se descompone el modelo Prophet. Sin embargo, antes de artir con cada uno, cabe recalcar que los autores Taylor y Letham (2018) mencionan que

[8]Implementación https://facebook.github.io/prophet/

el modelo Prophet es similar a un modelo aditivo generalizado usando solo el tiempo como regresor, donde la modelización de la estacionalidad de forma aditiva mantiene el mismo enfoque adoptado por el suavizado exponencial. En cambio, la estacionalidad multiplicativa, si se necesitase podría lograrse usando una transformación logarítmica. Es por ello, que funciona muy bien con series que tienen grandes efectos estacionales.
Además, este modelo se basa en ajustar una curva a los datos históricos, en lugar de encontrar el proceso subyacente, capturando patrones estacionales y tendencias de manera más intuitiva y asumiendo una pérdida de información predictiva. Esto significa que Prophet no tiene en cuenta la dependencia temporal de los datos como si lo hacen los modelos ARIMA, donde los valores futuros dependen de los valores pasados y por tanto, puede no ser adecuado para todos los conjuntos de datos, especialmente aquellos con dependencias temporales complejas o irregulares. Sin embargo, puede ser útil para conjuntos de datos donde los patrones estacionales y las tendencias son más evidentes y se desea una herramienta más simple y rápida para hacer predicciones.

Volviendo al primer factor del modelo, Prophet tiene implementados dos modelos de tendencias: un modelo de crecimiento saturado o logístico y un modelo de crecimiento lineal por partes. Revisaremos ambos modelos a continuación:
Tendencia saturada
Para la predicción del crecimiento se suele usar un enfoque similar a lo que sucede en la naturaleza, respecto al crecimiento de la población, el cual es no lineal y tiene un límite máximo de crecimiento. A este último concepto, los autores lo acuñan a una saturación de una capacidad de carga, la cual no permite un crecimiento más allá de un límite determinado y a menudo puede ser modelizado por el crecimiento logístico, definido como sigue:

$$g(t) = \frac{C(t)}{1 + \exp(-(k + a(t)^\top \delta)(t - (m + a(t)^\top \gamma)))},$$

donde $g(t)$ representa la cantidad en el tiempo t, que, siguiendo con la analogía al crecimiento poblacional podría ser el número de usuarios de Facebook en una región en particular. $C(t)$ es el límite máximo de crecimiento en el tiempo t, el cual puede variar con el tiempo. Por ejemplo, podría reflejar el crecimiento de la infraestructura de Internet en una región. k es la tasa de crecimiento, que determina qué tan rápido se acerca la cantidad al límite máximo. Se asume que esta tasa de crecimiento puede cambiar, en cualquier instante t.
El vector S representa los puntos de cambio o *changepoints* ocurridos en los instantes $s_j, j = 1, .., S$, donde ocurren cambios significativos en la tasa de crecimiento. Por ejemplo, en el contexto de Facebook, podría representar el lanzamiento de una nueva característica o una campaña publicitaria importante que afecta el crecimiento de usuarios. El vector $a(t) \in (0, 1)^S$ se define como el vector de ajuste de tasa como sigue

$$a_j(t) = \begin{cases} 1 & \text{si } t \geq s_j \\ \\ 0 & \text{en otro caso} \end{cases}$$

Este vector indica si ha ocurrido un changepoint en un momento t dado. Para cada s_j, será 1 si t es mayor o igual que s_j, lo que indica que el changepoint ha ocurrido antes o en ese momento. De lo contrario, es 0.
Tasa de Crecimiento en el Instante t se ajusta teniendo en cuenta los changepoints y el vector de ajuste de tasa. Se calcula como $k + a(t)^\top \delta$ donde δ es un vector que contiene los cambios en la tasa en cada changepoint. Cada componente δ_j representa el cambio en la tasa en el changepoint s_j.
El parámetro de compensación m se puede interpretar como el valor de la tendencia en el instante 0 y cambia a medida que avanza el tiempo para ajustarse a las variaciones de la serie. En los puntos de cambio, m también se ajusta para conectar los segmentos y obtener una curva continua. El ajuste en

el changepoint j se calcula como

$$\gamma_j = \left(s_j - m - \sum_{l<j} \gamma_l \right) \left(1 - \frac{k + \sum_{l<j} \delta_l}{k + \sum_{l \leq j} \delta_l} \right),$$

donde γ es el vector de ajustes de compensación. El primer término en la multiplicación ajusta m en el changepoint j para conectar los segmentos anteriores y el segundo término modifica este ajuste para asegurar una transición suave entre los segmentos alrededor del changepoint j.
A modo de resumen, la utilización de este modelo de crecimiento es una forma de adaptación para ajustar la tasa de crecimiento y el parámetro de compensación a medida que se identifican los changepoints. Esto posibilita capturar de manera efectiva los cambios en la tendencia de la serie temporal de una forma flexible y continua.

Tendencia lineal con puntos de cambio
En este caso, se asume un crecimiento que no tiene un límite máximo. En otras palabras, no se espera que el crecimiento alcance un punto donde se detenga o se estabilice, como en el caso de la logística, o donde no se tiene información suficiente para modelar la saturación del crecimiento. Con lo anterior, el crecimiento se representa como una línea recta o una tendencia constante a lo largo del tiempo y una tasa constante a trozos del crecimiento proporciona un modelo parsimonioso, el cual se define como:

$$g(t) = (k + a(t)^\top \delta)t + (m + a(t)^\top \gamma),$$

donde k es la tasa de crecimiento constante, δ contiene los ajustes de tasa, m es el parámetro de compensación y γ_j se establece como $-s_j \delta_j$ para que la función sea continua y $a(t)$ es el vector de ajuste de changepoints, similar al modelo anterior. Aquí, la pendiente cambia como una función de t, por ello, a la pendiente k se le añade $a(t)^\top \gamma$.

Los puntos de cambio o changepoints se pueden especificar manualmente, por ejemplo, como comentabamos anteriormente si se conocen las fechas de lanzamientos de ciertas características nuevas que sean capaces de alteran el crecimiento, o de forma automática. Prophet considera una gran cantidad de posibles valores para los changepoints al hacerlo de forma automática, donde cada uno tiene una tasa de cambio distribuida según una distribución Laplace, es decir, $\delta_j \sim \text{Laplace}(0, \tau)$, con τ controlando la flexibilidad del modelo para alterar la tasa (Taylor y Letham, 2018). Particularmente cuando $\tau = 0$ el ajuste es equivalente al modelo logístico.
Cuando hacemos una predicción a horizonte h la tendencia se mantiene constante en términos de tasa de crecimiento. La incertidumbre en la tendencia se estima generalizando el modelo, asumiendo que en el futuro habrá la misma frecuencia promedio y magnitud de cambios en la tasa de crecimiento que en los datos históricos. Para esto, se simulan cambios futuros en la tasa de crecimiento que imitan los observados en el pasado, utilizando una varianza inferida a partir de los datos y reemplazando dicho valor en τ. Esta suposición puede llevar a intervalos de predicción que no son exactos, pero sirven como indicadores de posibles sobreajustes en el modelo.
En resumen, el modelo de tendencia lineal proporciona una forma parsimoniosa de capturar el crecimiento sin un límite máximo o saturación, utilizando una tasa de crecimiento constante y ajustando la tendencia en función de los changepoints identificados en los datos históricos. La incertidumbre en la predicción se estima considerando la variabilidad en la tasa de crecimiento observada en los datos históricos.

Estacionalidad
Para ajustar y pronosticar los efectos de estacionalidades anuales, mensuales, semanales, etc., debemos especificar modelos de estacionalidad que sean funciones periódicas del tiempo t. Esto significa que estos modelos consideran patrones que se repiten en intervalos regulares de tiempo, como días, semanas o años, para capturar adecuadamente las fluctuaciones estacionales en los datos. Basados en series de Fourier para proporcionar un modelo flexible, Taylor y Letham (2018) definieron el siguiente modelo:

Siguiendo la notación, sea P el período regular que esperamos tenga la serie, que podría ser un valor de 1 para datos mensuales, 7 para datos semanales, o cualquier otro valor que represente la periodicidad de los datos, podemos aproximar efectos estacionales suaves y arbitrarios con la siguiente formulación de $s(t)$

$$s(t) = \sum_{n=1}^{N} \left(a_n \cos\left(\frac{2\pi nt}{P}\right) + b_n \operatorname{sen}\left(\frac{2\pi nt}{P}\right)\right),$$

con N representando el número de términos en la serie de Fourier utilizados para aproximar los efectos estacionales en los datos. Aumentar N permite ajustar los patrones de los datos que suelen cambiar más rápidamente, pero asumiendo el riesgo de sobreajuste del modelo. Los autores considerán que, de forma general, para datos con patrones anuales y semanales los valores de N pueden ser $N = 10$ y $N = 3$, aunque se puede seleccionar el valor a través de un proceso automático como los criterios de información AIC o BIC.
Para estimar los parámetros del vector $\beta = (a_1, b_1, ..., a_N, b_N)$ traspuesto se realiza a partir de una matriz de estacionalidad con los vectores para cada valor de t de los datos históricos y los datos futuros. Por ejemplo, si hay estacionalidad semanal y N = 3, tenemos

$$x(t) = \left[cos\left(\frac{2\pi t}{7}\right), sen\left(\frac{2\pi t}{7}\right), ..., sen\left(\frac{2\pi(3)t}{7}\right)\right]$$

donde el componente estacional es $s(t) = x(t)\beta$ y $\beta \sim N(0, \sigma^2)$ para imponer un suavizado "previo" a la estacionalidad.

Vacaciones
Dado que los días festivos pueden provocar alteraciones en las actividades y por lo tanto en la serie temporal, forman una componente de interés para agregar en el modelo. Sumado a esto, se tiene en consideración que en muchas ocasiones cambia el día la celebración según el calendario actual, por ejemplo, el presente año la navidad se celebra el día miércoles 25 de diciembre mientras que el año 2023 fue el día lunes 25 de diciembre el cual estuvo mucho más cercano al fin de semana y por ello pudieron tener cierto tipo de influencia los días anteriores a la celebración en el efecto propio del día festivo como tal. Además, como a menudo no siguen un patrón periódico sus efectos no se pueden modelar por un ciclo suave. Sin embargo, el impacto de un día festivo anual en particular suele ser similar año tras año, por lo que es importante incorporarlo en la previsión.
Prophet permite ingresar manualmente los días festivos, incluso especificando por país, asumiendo que sus efectos son independientes. Los modelos de ARIMA y ETS, a diferencia de Prophet, tradicionalmente no incluyen una forma integrada de manejar días festivos de manera directa en el modelo. Aunque, se pudiese crear variables indicadoras que representen la presencia o ausencia de un día festivo en cada observación de los datos y luego incluirlas como variables exógenas en el modelo para capturar el impacto directo de ldichas festividades en los datos.
Prophet modela las vaciones o festividades a través de la siguiente matriz

$$Z(t) = [\mathbf{I}(t \in D_1), ..., \mathbf{I}(t \in D_L)],$$

donde se considera que para cada día festivo i, se define un conjunto de fechas pasadas y futuras D_i y se agrega una función indicadora al modelo para representar si un punto de tiempo dado t cae dentro de esa festividad. Luego, a cada festividad se le asigna un parámetro κ_i, que cuantifica su impacto en el pronóstico de la siguiente manera:

$$h(t) = Z(t)\boldsymbol{\kappa},$$

donde $\boldsymbol{\kappa} \sim N(0, \nu^2)$ con ν desviación estándar muestral de los datos.

Predicción

Una vez obtenidas las características estacionales, de vacaciones y los indicadores de punto de cambio $a(t)$ se procede a realizar predicciones. Para este punto nos enfocaremos en la realización de la predicción con Python, debido a que es el lenguaje de programación utilizado por la empresa, siguiendo muy de cerca lo planteado en Peixeiro (2022) en el Capítulo 19.
Inicialmente se debe realizar una preparación previa al formato de los datos temporales tal que se disponga de una columna de fecha, comúnmente llamada *ds* que sea aceptada por la librería *Pandas* de Python y otra columna numérica con los valores a pronosticar. Seguido a esto, se dividen los datos en un conjunto de entrenamiento y en otro de prueba. Luego, se inicializa un modelo Prophet creando una instancia de la clase Prophet, es decir, se crea un modelo de pronóstico utilizando Prophet a partir de la definición de un objeto en Python que representa el modelo de pronóstico, por ejemplo, de la siguiente manera $modelo = Prophet()$.
Una vez creado el modelo, se ajusta utilizando el método *fit* con los datos de entrenamiento. Dicho método es una función específica proporcionada por Prophet para ajustar el modelo a los datos históricos. Durante este proceso de ajuste, el modelo utiliza los datos de entrenamiento para estimar los parámetros y aprender las tendencias, estacionalidades y otros patrones presentes en la serie temporal. Una vez que el modelo ha sido ajustado correctamente, estará preparado para generar pronósticos.
Finalmente, se da paso a la generación de predicciones, donde se crean predicciones utilizando el método *predict* para un período futuro especificado. El modelo genera una estimación de la serie temporal para los puntos de tiempo futuros. Este método toma como entrada un rango de fechas futuras para las cuales se desean obtener predicciones, y devuelve una estimación de cómo se comportará la serie temporal en esos puntos de tiempo futuros. El modelo utiliza la información aprendida durante el ajuste, como las tendencias, estacionalidades y efectos de días festivos, para generar estas predicciones.

Para mejorar la calidad de las predicciones Prophet utiliza la validación cruzada y la optimización de hiperparámetros, con el fin de garantizar que los modelos de pronóstico sean estables y precisos. Esto permite a los usuarios encontrar la mejor configuración de modelo para sus datos y mejorar continuamente la calidad de las predicciones.
La validación cruzada es una técnica utilizada para evaluar el rendimiento de un modelo y garantizar su estabilidad. En este contexto, se lleva a cabo entrenando el modelo en un subconjunto de los datos de entrenamiento y pronosticando en un horizonte h determinado. En principio define un subconjunto del los datos de entrenamiento, ajusta el modelo y predice a horizonte h_1. Luego, añade más datos al subconjunto definido anteriormente, vuelve a ajustar el modelo y predice para otro horizonte h_2. Este proceso se repite hasta utilizar el conjunto de entrenamiento completo, como se ve en la Figura 2.3, siempre y cuando exista una cantidad suficiente de datos. Sino es el caso, Prophet proporciona valores predeterminados que son adecuados para la mayoría de los casos, sin embargo, cuando sea necesario ajustar estos parámetros para optimizar el rendimiento del modelo, el analista trabaja en conjunto con el proceso de ajuste del modelo para encontrar los valores óptimos.

Después de realizar la validación cruzada se utilizan métricas de evaluación del rendimiento del modelo, como el error absoluto medio (MAE) o el error porcentual absoluto medio (MAPE), para medir la precisión del modelo en cada período de predicción. Estas métricas proporcionan información sobre la calidad de las predicciones y si el modelo está mejorando o empeorando a medida que predice más en el futuro. Dichas métricas las revisaremos más en detalle en la siguiente sección.

Por otra parte, Prophet también permite ajustar los hiperparámetros del modelo para mejorar o refinar su rendimiento. Esto se hace experimentando con diferentes valores de hiperparámetros y evaluando cómo afectan la calidad de las predicciones en la validación cruzada. El objetivo es encontrar la combinación óptima de hiperparámetros que produzca las mejores predicciones para el problema específico en cuestión. Por lo general se pueden ajustar cuatro parámetros por los usuarios:
Escala de punto de cambio: determina la flexibilidad de la tendencia, es decir, cuánto cambia la tendencia en los puntos de cambio de tendencia. Si este parámetro es demasiado pequeño, la tendencia no se ajustará correctamente y la varianza observada en los datos se tratará como ruido. Si es demasiado

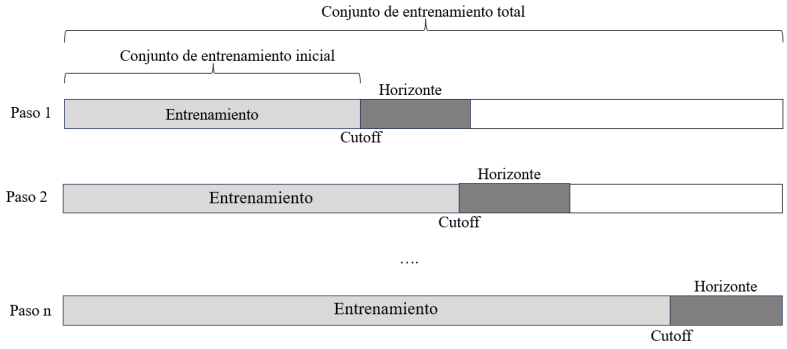

Figura 2.3: Procedimiento de validación cruzada en Prophet para una serie temporal. Fuente: Peixeiro (2022), Capítulo 19, traducida.

alto, la tendencia se sobreajustará a fluctuaciones ruidosas. Peixeiro (2022) en el Capítulo 19 menciona que el rango $[0{,}001, 0{,}01, 0{,}1, 0{,}5]$ es suficiente para tener un buen ajuste.
Escala de estacionalidad: determina la flexibilidad de la estacionalidad. En este caso, un valor grande permite que la estacionalidad se ajuste a fluctuaciones más pequeñas y un valor pequeño dará una estacionalidad más suave. Al igual que antes, Peixeiro (2022) en el Capítulo 19 sugiere el rango $[0{,}01, 0{,}1, 1{,}0, 10{,}0]$.
Escala de vacaciones: determina la flexibilidad de los efectos de las vacaciones y funciona igual que la escala anterior, con el mismo rango.
Modo de estacionalidad: la estacionalidad puede ser añadida de forma aditiva o multiplicativa. Por defecto es aditiva, pero si la fluctuación estacional se hace más grande con el tiempo se puede establecer como multiplicativa.
Entonces, se define el rango de valores a probar para cada parámetro y se genera una lista de combinaciones únicas de parámetros. Para cada combinación se entrena un modelo por validación cruzada. Luego, se evalúa el modelo a través de las métricas determinadas y se procede a realizar la predicción. La combinación de los dos métodos mencionados anteriormente dará como resultado una forma robusta de encontrar un modelo óptimo.
A modo de resumen, podemos considerar a Prophet como un modelo flexible que asume una pérdida de información predictiva debido al enfoque de ajuste de curva a los datos en vez del proceso subyacente como comentabamos al inicio, pero que garantiza la integración de múltiples periodos estacionales y cambios de tendencia. Por ello, trabaja mucho mejor con series que tienen un gran efecto estacional con algunas estacionalidades en los datos históricos. También, como comentamos en la sección de predicción del modelo, es un ajuste bastante rápido, es decir, que no suele usar tantas líneas de código para obtener resultados. Peixeiro (2022) menciona también que es robusto frente a outliers y datos faltantes en el Capítulo 19.

2.2.4. Modelo ML

Los árboles de clasificación y regresión llamados CART por sus siglas en inglés (*Classification and Regression Tree*) son modelos de aprendizaje supervisado en el cuál existe una variable objetivo predeterminada y a su vez forman una técnica de análisis predictivo. Su efectividad y su representación gráfica los han vuelto unos de los algoritmos más populares en Aprendizaje Automático o *Machine Learning*.
La construcción de un CART se hace a través de la división binaria recursiva con un nodo raíz[9] que contiene todo el conjunto de datos. Es decir, se analiza cada variable de los datos y se divide en dos,

[9]Llamaremos nodo raíz a la población completa o a su muestra respectiva.

buscando maximizar la pureza de los nuevos subnodos. Este proceso se vuelve a repetir hasta cumplir algún criterio de parada, como la profundidad máxima del árbol, el número mínimo de muestras requeridas en un nodo o la incapacidad para mejorar la homogeneidad con una nueva división.
La medida de homogeneidad más comúnmente utilizada en la construcción de árboles CART es el Índice de Gini, el cual se busca minimizar durante la construcción del árbol para la obtención de un conjunto más homogéneo. En el caso de árboles de regresión se suele utilizar la suma residual de cuadrados, buscando que en cada subdivisión del árbol se considere la que genere una mayor reducción en dicha medida.

En el caso de un problema de regresión, se suele usar un árbol de regresión donde la variable dependiente es continua y los valores de los nodos terminales son reducidos a la media de las observaciones presentes en esos nodos, cuyo razonamiento se puede aplicar a series temporales a través del método de ventana corrediza. Es decir, se reestructura la serie temporal del formato (tiempo, observacion)= $((t_0, t_1, ..., t_k), (x_0, x_1, ..., x_k))$ a un problema de aprendizaje supervisado utilizando el valor de la serie temporal en el instante anterior para predecir el valor en el siguiente instante, eliminando la columna tiempo como sigue:

x	y
	x_0
x_0	x_1
x_1	x_2
...	...
x_k	

Tanto la ventana de entradas como las de salidas esperadas se desplazan hacia adelante a través del tiempo para crear nuevas "muestras". Sin embargo, para realizar una predicción de un intervalo, y no solo de un valor, se suele utilizar el método multipaso recursivo o el método de multipaso directo.
El primer método se constituye por etapas, donde a cada paso de predicción se le conoce como *step* según Amat y Escobar (2023) y se realiza la predicción para el instante t_{T+1} a partir de un subconjunto de las últimas n observaciones anteriores. El siguiente *step* considera los $n-1$ valores anteriores más el valor pronósticado en el paso anterior $\hat{x}_{T+1}$ y así sucesivamente. La Figura 2.4 representa el algoritmo de forma gráfica.
Siguiendo con la idea de Amat y Escobar (2023) el método de multipaso directo crea un modelo diferente para cada predicción tomando $n \leq T$ observaciones consecutivas de la serie temporal para predecir el valor a horizonte h pasos hacia delante, como muestra la Figura 2.5. En este caso, si se quiere calcular todos los valores a horizontes intermedios entre el horizonte 1 y h es necesario entrenar h modelos distintos, aumentando el coste computacional.
Una forma de mejorar la eficiencia y efectividad de los CART es a través de la inclusión de nuevos métodos como *Random Forests* o *XGBoost* (*Extra gradient boosting*), los cuales se basan en múltiples árboles de decisión, en nuestro caso, nos centraremos en este último.

Gradient Boosting Trees
Un modelo Gradient Boosting está formado por un conjunto de árboles CART, entrenados de forma secuencial, tal que cada nuevo árbol mejore de alguna forma los errores de los árboles anteriores. La predicción de una nueva observación se obtiene a través de la combinación de las predicciones de todos

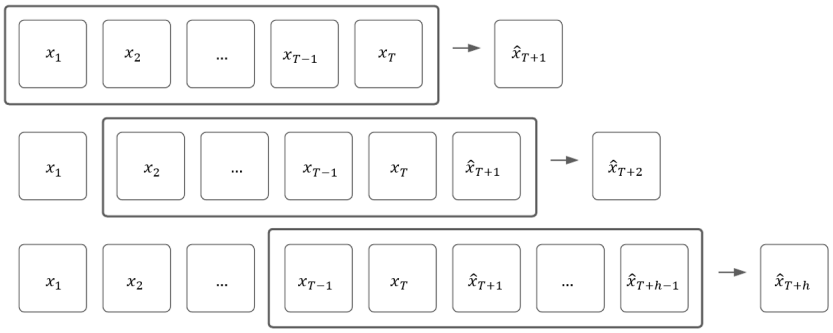

Figura 2.4: Método multipaso recursivo con $n = T$.

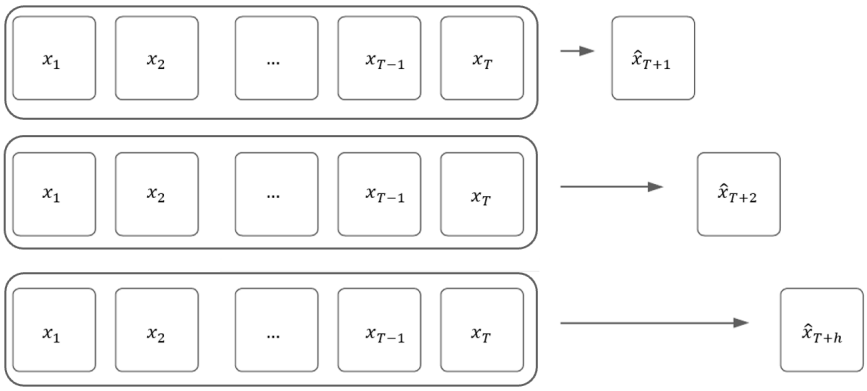

Figura 2.5: Método multipaso recursivo con $n = T$.

los árboles individuales, disponiendo de múltiples predictores que interactuan, generalmente, de forma no lineal.
Encontrar un modelo que represente la interacción antes mencionada es una tarea compleja, donde la utilización de árboles CART permiten segmentar el espacio de los predictores en regiones pequeñas y más simples, donde es más sencillo manejar las interacciones, obteniendo buenos resultados en la mayoría de los casos (Amat, 2020).

Para entender cómo funcionan los modelos Gradient Boosting Trees es necesario conocer primero los conceptos de *ensemble* y *boosting*. El primero de ellos combina múltiples modelos en uno nuevo con el objetivo de equilibrar el problema entre sesgo y varianza para obtener mejores predicciones que los modelos originales. El término sesgo[10] se refiere a cuánto se desvían, en promedio, las predicciones de un modelo con respecto a las observaciones reales. Por otro lado, la varianza hace referencia a cuánto cambia el modelo en función de los datos usados en su entrenamiento. Es decir, un modelo no debería tener grandes variaciones ante pequeños cambios en las observaciones de entrenamiento, si fuese así el modelo tendría alta varibilidad.
Al aumentar la complejidad del modelo se reduce el sesgo, pero puede llegar el caso donde el modelo se ajusta tanto a los datos de entrenamiento, como un polinomio de interpolación, que no puede predecir de forma correcta nuevos datos. Este problema se conoce como sobreajuste o *overfitting*, dando lugar a un alza en la varianza. Los modelos de árboles con pocas ramificaciones tienden a tener poca varianza pero pueden no captar la relación entre las variables de forma exacta debido a su construcción, lo

[10]Refleja la habilidad del modelo para capturar la verdadera relación entre los predictores y la variable de respuesta.

que se traduce en un aumento del sesgo. Por tanto, se intenta buscar un modelo óptimo que pueda equilibrar sesgo y varianza, donde los métodos de ensemble intentan dar solución a este problema de equilibrio.

Los métodos de ensemble más usados son: *Bagging* y *Boosting*. En el primero de ellos se ajustan múltiples modelos diferentes, cada uno ajustado a un subconjunto distinto de datos de entrenamiento. Para predecir, todos los modelos aportan su predicción y se considera media de dichas predicciones como valor final.
Por otro lado, en la segunda variante del método se ajustan secuencialmente múltiples modelos sencillos con pocas ramificaciones, llamados *weak learners*, de forma que cada modelo aprende de los errores del anterior y la importancia o peso de las observaciones va cambiando en cada iteración dando lugar a diferentes ajustes.
Se suelen usan modelos con baja varianza, lo que conlleva un valor de sesgo alto. Al realizar el ajuste de forma secuencial los modelos pueden llegar a reducir el sesgo y por ende reducir el error total[11]. Como menciona Amat (2020), considerar para cada método, ajustes individuales lo más diferentes posibles es lo que mejor contribuye a mejorar los resultados.

Gradient Boosting
Uno de los algoritmos de *boosting* más usados es *Gradient Boosting*, el cual entrena modelos secuencialmente, ajustando los residuos de los modelos anteriores.
Inicialmente se considera la variable respuesta y y se ajusta un primer modelo *weak learner* o iteración f_1, con f diferenciable, tal que:

$$f_1(x) \approx y.$$

Luego, se ajusta un segundo modelo f_2 que intenta predecir los residuos del modelo anterior, es decir, "intenta" corregir los errores del modelo anterior como sigue:

$$f_2(x) \approx y - f_1(x).$$

La siguiente iteración ajusta un tercer modelo f_3 que intenta predecir los residuos de ambos modelos anteriores como sigue:

$$f_3(x) \approx y - f_1(x) - f_2(x).$$

Este procedimiento se realiza M veces, minimizando los residuos iteración a iteración:

$$y \approx f_1(x) + f_2(x) + ... + f_M(x).$$

Además, este algoritmo contiene una gran cantidad de hiperparámetros cuyos valores óptimo se identifican mediante validación cruzada. A continuación, mencionaremos los más importantes según Amat (2020). El primero de ellos es la cantidad de *weak learner* utilizados. Como este proceso realiza un ajuste en cada iteración es susceptible a hacer un sobreajuste si hay un número excesivamente alto de modelos ajustados. Para evitarlo, se usa otro hiperparámetro llamado *learning rate* denotado por λ cuyo valor realiza una regularización, es decir, controla la influencia que tiene cada *weak learner* o modelo ajustado en el conjunto del ensemble, limitando su influencia. Generalmente λ toma valores entre 0.001 o 0.01, mientras menor sea su valor menor es el riesgo de sobreajuste, pero más árboles se necesitan para alcanzar buenos resultados. Otro hiperparámetro es en torno al tamaño de cada *weak learner*, el autor menciona de 1 a 10 como un rango aceptable de cada árbol.

El modelo ajustado por *learning rate* queda como sigue:

$$f_1(x) \approx y$$

$$f_2(x) \approx y - \lambda f_1(x)$$

[11] Error total: $sesgo + varianza + \varepsilon$

$$f_3(x) \approx y - \lambda f_1(x) - \lambda f_2(x)$$

...

$$y \approx \lambda f_1(x) + \lambda f_2(x) + ... + \lambda f_M(x).$$

Dada la gran cantidad de hiperparámetros que tienen los modelos de *gradient boosting*, se suelen combinar las estrategias de *backtesting* de validación cruzada con la búsqueda en cuadrícula o *grid search* para identificar la configuración de hiperparámetros óptima.
Amat y Escobar (2023)[12] describen la primera técnica como la evaluación del rendimiento del modelo aplicándolo a los datos históricos. Esencialmente, es una forma especial de validación cruzada aplicada a períodos anteriores. Su propósito principal es evaluar la precisión y eficacia de un modelo, identificando posibles problemas o áreas de mejora. El *backtesting* se puede realizar utilizando diversas técnicas, como divisiones de entrenamiento-prueba simples o métodos más sofisticados como ventanas de tiempo móviles o ventanas en expansión, seleccionando el método según las necesidades específicas del análisis y las características de los datos de series temporales.
Los autores mencionan además que el tiempo necesario para ajustar los hiperparámetros aumenta con el número de reentrenamientos realizados, donde para agilizar el proceso se recomienda realizarlo en dos fases. En la primera fase el modelo se entrena solo una vez y se utiliza secuencialmente sin actualizarlo, es decir, el modelo se entrena con un conjunto de datos de entrenamiento inicial y luego se utiliza el mismo conjunto para hacer predicciones secuenciales sobre datos futuros sin reentrenarse con las nuevas observaciones que se van obteniendo con el tiempo. Esto significa que el modelo no se actualiza con la información más reciente disponible y sigue utilizando los parámetros aprendidos durante el entrenamiento inicial para hacer todas las predicciones.
Esta aproximación tiene la ventaja de ser mucho más rápida que otros métodos que requieren el reentrenamiento del modelo en cada iteración. Sin embargo, se corre el riesgo de que el modelo puede perder su capacidad predictiva con el tiempo, ya que no incorpora la información más reciente disponible.
La segunda fase se centra en la región de interés identificada durante la primera fase. Esta región de interés es el conjunto más prometedor de valores de hiperparámetros que se han identificado durante la búsqueda inicial, donde es más probable encontrar la configuración óptima para el modelo, la cual se puede identificar a través de una búsqueda amplia y exhaustiva de diferentes combinaciones de hiperparámetros utilizando técnicas como la búsqueda en cuadrícula que registra el rendimiento del modelo para cada combinación de hiperparámetros.
Para esta fase, se aplica una estrategia de *backtesting* adaptada, que se ajusta a los requisitos específicos del caso de uso. Esto implica diseñar un proceso de evaluación más detallado y específico que permita evaluar a fondo el rendimiento del modelo dentro de esta región de interés. El objetivo de esta fase es obtener una evaluación más precisa y exhaustiva del rendimiento del modelo dentro de la región de interés identificada, lo que permite tomar decisiones más informadas sobre la configuración final de los hiperparámetros.
La búsqueda en cuadrícula o *Grid search* es una técnica de ajuste de hiperparámetros que evalúa una lista de combinaciones de hiperparámetros y observaciones pasadas utilizadas como predictores para predecir futuras observaciones con el fin de encontrar la configuración óptima para un modelo de pronóstico, basandose en el rendimiento del modelo evaluado mediante *backtesting*.
Por tanto, el modelo se entrena sobre la muestra de entrenamiento, evaluando el modelo mediante *backtesting* y se vuelve a entrenar con la mejor combinación de hiperparámetros y y observaciones pasadas encontrada. Se puede obtener más detalle en Amat y Escobar (2023).

Una estrategia alternativa es donde el *weak learner* se ajusta a una fracción del conjunto de entrenamiento, extraída de forma aleatoria y sin reemplazo, llamada *Stochastic Gradient Boosting* la cual consigue mejorar la capacidad predictiva. El modelo *XGBoost* (*Extreme Gradient Boosting*) es una implementación de este algoritmo, el cual entrena un modelo autorregresivo, que utiliza valores pasados

[12] Basados en la librería Skforecast.

de la variable de respuesta como predictores y va agregando variables exógenas al modelo, evalúando la mejora del rendimiento. Algunas de las principales caracteristicas del modelo XGBoost son:

Permite valores faltantes o missing values: *XGBoost* puede manejar automáticamente los valores faltantes en los datos durante el entrenamiento del modelo. Esto significa que no es necesario preprocesar manualmente los datos para llenar o eliminar los valores faltantes antes de entrenar el modelo, dado que puede manejarlos internamente de manera eficiente.

Permite el uso de GPUs: *XGBoost* ofrece la capacidad de utilizar unidades de procesamiento gráfico (GPUs) para acelerar el proceso de entrenamiento del modelo. Esto puede resultar en tiempos de entrenamiento más cortos, especialmente para conjuntos de datos grandes, ya que las GPUs están diseñadas para manejar cálculos matriciales de manera altamente paralela.

Entrenamiento paralelizado: *XGBoost* implementa un algoritmo de entrenamiento eficiente que puede paralelizar algunas partes del proceso de entrenamiento. Esto significa que puede aprovechar múltiples núcleos de CPU para acelerar el proceso de entrenamiento, lo que resulta en tiempos de entrenamiento más cortos en comparación con los métodos secuenciales.

Permite restricciones monotónicas: *XGBoost* ofrece la capacidad de imponer restricciones monotónicas en las variables predictoras. Esto significa que se puede especificar si se espera que una variable tenga un efecto monotónicamente creciente o decreciente en la variable de respuesta. Esta característica puede ser útil en escenarios donde se tiene conocimiento previo sobre la relación entre las variables.

Permite trabajar sobre matrices dispersas: *XGBoost* puede manejar eficientemente matrices dispersas, que son matrices en las que la mayoría de los elementos son cero. Esto es útil cuando se trabaja con conjuntos de datos que tienen una gran cantidad de características, pero la mayoría de ellas son cero, como en el caso de datos textuales o datos con codificación *one-hot* de variables categóricas. La codificación *one-hot* implica que cada categoría única en una variable categórica se convierte en una nueva variable binaria. Si una observación pertenece a una categoría específica, la variable correspondiente se establece en 1; de lo contrario, se establece en 0. La ventaja de la codificación es que preserva la información sobre las categorías sin asignarles un orden o valor numérico específico. Esto es útil para algoritmos de aprendizaje automático que no pueden trabajar directamente con variables categóricas.

Acepta variables categóricas: *XGBoost* puede manejar variables categóricas directamente, sin necesidad de convertirlas previamente en variables numéricas mediante técnicas de codificación. Esto simplifica el proceso de preparación de datos y puede mejorar la eficiencia del modelo, especialmente cuando se tiene muchas variables categóricas en los datos.

Todas estas características hacen que XGBoost sea una opción especialmente atractiva para el modelado predictivo de series temporales

2.2.5. Modelos NN

Las redes neuronales han emergido como una herramienta más en el ámbito del modelado y predicción de series temporales, ofreciendo un enfoque innovador y eficaz para abordar la complejidad inherente a estos datos. La capacidad de las redes neuronales para capturar relaciones no lineales y dependencias temporales las convierte en una opción atractiva y poderosa para analizar y predecir patrones en series temporales de diversas naturalezas, desde datos financieros hasta fenómenos climáticos. Su flexibilidad y capacidad para manejar grandes volúmenes de datos, así como su habilidad para adaptarse a cambios en los patrones temporales, las posicionan como una herramienta interesante en la investigación y la práctica en este campo. Al integrar conceptos de autorregresión con redes neuronales, se abre la puerta a modelos más sofisticados y precisos, capaces de capturar tanto las tendencias a largo plazo como las variaciones estacionales, lo que resulta en una mejora en la capacidad predictiva. En este contexto, explorar y comprender el potencial de las redes neuronales para el modelado y predicción de series temporales no solo amplía nuestro conocimiento teórico, sino que también al tener aplicaciones prácticas importantes en una variedad de campos, se vuelve una herramienta interante de abordar en nuestro trabajo.

En esta sección abordaremos de forma teórica las redes neuronales y sus características más importantes, junto con la conexión a series de tiempo para su modelado y posterior predicción.

La Figura 2.6 representa una red neuronal que a su vez está representando un proceso $AR(p)$. La capa de entrada o *input layer* recibe las observaciones, en este caso las p observaciones pasadas, la capa de salida u *output layer* genera la predicción combinando los valores de salida de la capa oculta o *hidden layer*, la cuale representa el ajuste del modelo $AR(p)$ para producir la predicción.

Cada nodo representa una neurona y están conectados tanto a las neuronas de la capa anterior como a las de la capa siguiente, a través de ponderaciones que modulan la información en las flechas de unión. En este caso, siguiendo el modelo $AR(p)$ antes mencionado, las ponderaciones son los parámetros $\phi_1, ..., \phi_p$ y la capa de salida es obtenida por una función lineal de la capa de entrada.

En particular, cuando una red tiene como capa de entrada a los valores observados de las observaciones pasadas de una serie temporal se conoce como autorregresión de red neuronal o modelo $NNAR(p, k)$ donde p son los valores de entrada pasados y k las neuronas en la capa oculta. Por otra parte, si la serie en estudio tiene datos estacionales es de gran utilidad añadir los últimos datos de la misma estacionalidad, en este caso se conoce como $NNAR(p, P, k)_m$ que es equivalente al modelo $(y_{t-1}, ..., y_{t-p}, y_{t-m}, ..., y_{t-Pm})$ o $ARIMA(p, 0, 0)(P, 0, 0)_m$[13].

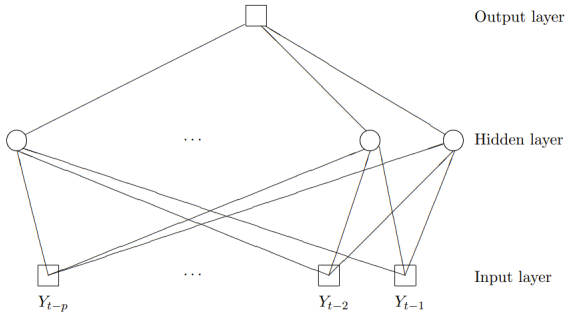

Figura 2.6: Representación red neuronal de un AR(p). Fuente: Makridakis et al. (2008), Capítulo 8.

Para definir y entrenar una red neuronal con el fin de realizar tareas específicas como la regresión de series temporales, debemos definir algunos elementos importantes previamente. La elección adecuada de estos elementos puede tener un gran impacto en el rendimiento y la eficacia de la red. Nos guiaremos por lo expuesto en Makridakis et al. (2008), Géron (2022), Hyndman (2014) y Peixeiro (2022).

El primer elemento es la arquitectura de una red neuronal, la cual representa el número de capas y neuronas que están presentes en una red y, también, la forma en la que están conectadas. Las arquitecturas más simples no tienen capas ocultas y equivalen a regresiones lineales.

La arquitectura más simple se llama *perceptrón*, la cual está basada en una sola capa de neuronas *threshold logic unit* (TLU) que tienen entradas y salidas dentro del conjuto real y cada valor de entrada a la neurona está asociado a un peso o ponderación, donde la neurona TLU calcula la suma de los productos de los valores de entrada por sus respectivas ponderaciones. Luego, al valor obtenido de la suma de dichos productos, se le aplica dentro de la misma neurona una función llamada *step function* $h(.)$ que genera un valor de salida o *output*.

Una de las funciones de paso más usadas es la función de escalón Heaviside o *Heaviside step function* y también la función de signo o *the sign function*,las cuales se aplican al resultado de la combinación lineal anterior y se definen como:

[13]Un modelo $NNAR(p, 0)$ es equivalente a $ARIMA(p, 0, 0)$ y $NNAR(p, P, 0)_m$ lo es a $ARIMA(p, 0, 0)(P, 0, 0)_m$, ambos sin restricciones sobre estacionariedad.

$$\text{heaviside}(x) = \begin{cases} 0 & \text{si } x < 0 \\ 1 & \text{si } x \geq 0 \end{cases} \qquad \text{sgn}(x) = \begin{cases} -1 & \text{si } x < 0 \\ 0 & \text{si } x = 0 \\ 1 & \text{si } x > 0 \end{cases}$$

Estas funciones entregan un resultado numérico que se puede emplear para problemas de clasificación. Por otro lado, también existe una neurona llamada *bias neuron* la cual representa el sesgo y que genera como salida un 1.
Cabe mencionar que también es posible considerar una combinación no lineal de neuronas, una vez que agregamos una capa oculta, la red neuronal se vuelve no lineal y se conoce como red *feed-forward multicapa* (FNN) donde el flujo de las entradas es unidireccional. Además, Géron (2022) en el Capítulo 10, menciona que muchas limitaciones de los perceptrones, como la incapacidad de resolver algunos problemas triviales y la incapacidad de capturar el orden temporal de una serie, son suprimidas considerando múltiples perceptrones, es decir, un perceptrón multicapa o *Multi-Layer Perceptron (MLP)*[14] que, además de ser un ejemplo de red FNN es donde recae la ventaja del uso de redes neuronales aplicadas en series de tiempo, pues al sumar múltiples capas ocultas producirán modelos mucho más complejos de predicción, aunque con un elevado coste computacional. Es decir, cuando se aplican redes neuronales a series temporales, estas entregan un modelo de predicción no lineal que requiere un gran número de observaciones, pero que, como mencionabamos antes, es flexible y es capaz de ajustar modelos complejos.

Cuando se tiene una capa de varias neuronas generando varias salidas, estas se pueden agrupar en una de la siguiente manera:

$$h_{w,b}(\mathbf{X}) = f(\mathbf{XW} + \mathbf{b}),$$

donde las matrices $\mathbf{X}$ y $\mathbf{W}$ representan los valores de entrada y las ponderaciones respectivamente, sin considerar el peso de la neurona del sesgo. El vector $\mathbf{b}$ representa los pesos entre la neurona sesgo y las otras neuronas, y la función $f(.)$ se llama *función de activación*, la cual es parte de los elementos importantes para especificar una red, en conjunto con su arquitectura. Comentaremos más profundamente sobre sus características más adelante.
Por ejemplo, si consideramos la neurona 1 de la capa oculta de la Figura 2.6, primer círculo de izquierda a derecha, podemos representar su información de entrada como:

$$\phi_1 Y_{t-1} + \phi_2 Y_{t-2} + ... + \phi_p Y_{t-p} + \varepsilon_t,$$

donde ε_t representa a los errores de la serie, asumidos homocedasticos.
Esta neurona, en función del resultado de la suma ponderada que recibe aplica una *función de activación* $f(.)$ que describe una salida hacia la capa oculta siguiente, si esque la hay, generando un *valor de activación* o salida denotado por a, de la siguiente manera:

$$Y_t = f(Y_{t-1}, ..., Y_{t-p}) + \varepsilon_t,$$

donde

$$a = f(Y_{t-1}, ..., Y_{t-p}) + \varepsilon_t$$

es equivalente a un modelo de autorregresión no lineal.
Puesto que en la capa de entrada queremos que los valores sean las mismas observaciones pasadas, la función $f(.)$ dentro de cada neurona es la función identidad. De forma general, para las neuronas que se encuentran en una misma capa, se les asigna una misma función de activación.

[14] En los 90' si en la red habían dos o más capas ocultas se consideraba como red neuronal profunda (deep neural network (DNN)), ahora una red profunda tiene docenas de capas o más.

Habitualmente, para una función de activación no lineal se elige una combinación entre una función lineal de las neuronas de entrada, seguida por una función no lineal llamada de función de aplastamiento.

Principales funciones de activación
Las funciones de activación determinan cómo una neurona combina sus entradas para producir una salida. Son generalmente funciones no lineales que introducen no linealidades en el modelo, lo que permite a la red aprender y modelar relaciones más complejas en los datos. A continuación revisaremos algunas las funciones de activación más comunes.
Si consideramos una red compuesta por una única neurona conocida como "perceptrón simple", el valor a se puede comparar con un umbral, para el cual la función de activación genera un resultado si a sobrepasa o no dicho umbral. Generalmente, las funciones de activación sencillas generan valores de $a \in (0, 1)$ o $a \in (-1, 1)$. En particular, cuando $a = 0$ se dice que la neurona está inactiva.
Cuando las neuronas son TLUs, la función de activación es la función de paso o *step function* mencionada anteriormente. Otras funciones de activación son las siguientes:

Rectified linear unit (ReLU): esta función de activación se considera sólo si el valor de entrada es mayor a cero, sino, el valor de salida es cero. Cuando el valor de entrada es mayor a cero, el valor de salida aumenta linealmente con el de entrada.

$$\text{ReLU}(x) = max(x, 0).$$

Exponential linear unit (ELU): esta función de activación entrega valores cercanos a cero si el valor de entrada es negativo, lo cual impide que los gradientes se anulen. Si $\alpha = 1$ la función es suave alrededor de 0, evitando saltos al usar el algoritmo de *gradient descent*.

$$\text{ELU}_\alpha(x) = \begin{cases} \alpha(e^x - 1) & \text{si } x < 0 \\ x & \text{si } x \geq 0. \end{cases}$$

Softmax: esta función de activación realiza una tranformación a los valores de entrada a probabilidades.

$$\text{Softmax}(x) = \frac{e^x}{\sum_{k=1}^{K} e^{xk}}.$$

En este caso, es importante notar que la suma de todos los valores de salida debe dar como resultado 1.

Sigmoide: esta función de activación transforma valores entre $(-\infty, \infty)$ a valores entre $(0, 1)$.

$$\text{sigmoid}(x) = \frac{1}{1 + \exp(-x)}.$$

Por ejemplo, si el valor de activación es $a = \sum_{i=1}^{p} \phi_i Y_{t-i} + \varepsilon$, este se convierte a su vez en un valor de entrada de la función $g(a) = \frac{1}{1+e(-a)}$. La gran ventaja es que esta función es capaz de reducir los efectos de valores de entrada extremos, proporcionando un mayor grado de robustez a la red.
Por último, consideramos a la función Tangente hiperbólica (Tanh). Esta función de activación transforma valores entre $(-\infty, \infty)$ a valores entre $(-1, 1)$.

$$\tanh(x) = \frac{1 - \exp(-2x)}{1 + \exp(-2x)}.$$

Otro elemento importante es la función de costo, la cual mide cuán bien está funcionando la red, es decir, la precisión de las predicciones. Generalmente, se trata de una función que mide la diferencia

entre las salidas predichas por la red y las salidas reales conocidas en el conjunto de datos de entrenamiento, es decir, la desviación entre el valor real de una observación y su estimación a partir de la red. Estas funciones se abordarán en la siguiente sección, donde será equivalente función de coste a medida de error.

Entrenamiento de la red

Según Makridakis et al. (2008) en el Capítulo 8, el entrenamiento de la red también forma parte de los elementos que se deben considerar para especificar una red neuronal, cuyo objetivo es minimizar esta función de costo.

El algoritmo de entrenamiento es el método utilizado para ajustar los parámetros de la red neuronal (como los pesos y sesgos de las conexiones entre neuronas) con el fin de minimizar la función de costo. Algunos de los algoritmos de entrenamiento más comunes son el Descenso del Gradiente (*Gradient Descent*) y sus variantes, como el Descenso del Gradiente Estocástico (*Stochastic Gradient Descent*), el Descenso del Gradiente con Momento (*Gradient Descent with Momentum*), etc.

En este contexto, en vez de hablar de estimación de parámetros, hablamos de entrenamiento de la red neuronal. Las ponderaciones o pesos que existen entre las uniones de neuronas y el sesgo se eligen mediante un algoritmo de entrenamiento, el cual intenta encontrar los valores de los parámetros, en el caso del $AR(p)$ el vector $(\phi_1, \ldots, \phi_p)$ asociado a las observaciones pasadas de la serie, que minimizan una función de coste o *loss function*.

Habitualmente se utiliza el algoritmo de entrenamiento *gradient descent* para ajustar dichos pesos y sesgo de cada neurona, el cual minimiza localmente una función diferenciable. A continuación, se seguirán los siguientes pasos para el entrenamiento con dicho algoritmo:

Especificar el número de capas ocultas y el número de neuronas en cada una de ellas. Generalmente seleccionados por validación cruzada o $k = (p + P + 1)/2$.

Preprocesado de las variable si es el caso, a través de transformaciones como el centrado y la estandarización para evitar diferencias entre escalas de la variables. El centrado resta a cada valor la media de la variable predictora a la que pertenece. La estandarización, por su parte, consiste en transformar los datos de forma que todos los predictores estén aproximadamente en la misma escala.

Iniciar la red con valores aleatorios de pesos y sesgo. La capa de entrada de una red suele constar de tantas variables explicativas como sea posible y los valores rezagados de la serie temporal. Para series con componente estacional, se busca tener tantas bservaciones pasadas de entrada como períodos. El número de neuronas de la capa de salida corresponde al número de variables que se van a predecir. Habitualmente, los pesos de cada capa se seleccionan en base a una distribución normal de media cero y varianza $1/n$ ó $2/n$, con n el número de entradas. El peso, por lo general, se inicializa en cero.

Se inicia un algoritmo de optimización de forma iterativa, el cual tratará de minimizar la diferencia entre el valor de activación y el valor real de la serie, es decir, el error que comete la red. Como comentabamos antes, uno de los más usados es *gradient descent*[15] en conjunto con el algoritmo propagación hacia atrás o *backpropagation*, el cual permite actualizar los pesos y sesgo del modelo para reducir su error de predicción propagandolos hacia atrás.

De forma general, *gradient descent* luego de calcular la función de coste, calcula el *gradiente* como las derivadas parciales de dicha función con respecto a todos los parámetros de la red, a través del algoritmo *backpropagation*. Este último, identifica y cuantifica la influencia de los pesos y del sesgo en las predicciones de la red basandose en la regla de la cadena para calcular el gradiente de error de la red con respecto a cada parámetro de la última capa del modelo, de forma automática. Esto se repite sucesivamente para las capas anteriores hasta llegar al inicio de la red. Como menciona Géron (2022) en el Capítulo 10, este algoritmo puede averiguar cómo cada conexión pesa y cada término de sesgo debe modificarse para reducir el error de la red.

[15] Existen varias versiones: gradiente descendente en lotes, estocástico y estocástico en mini-lotes. Muchas veces, por el coste computacional, se ocupa método de descenso de gradiente estocástico o estocástico en mini-lotes, que divide los datos en grupos o *batch* y actualiza los parámetros de forma progresiva con cada batch.

Una vez calculado el vector gradiente, se actualizan los parámetros de la red restando a su valor actual el valor del gradiente correspondiente, multiplicado por una tasa de aprendizaje o *learning rate*. Esta tasa es uno de los parámetros más difíciles de determinar, pues modela los cambios en los demás parámetros del modelo a medida que se van optimizando. De forma general, se usa la tasa variable más pequeña posible, con sus valores mayores al principio, los que van disminuyendo con el tiempo.

Una forma de prevenir el sobreajuste del modelo es regularizando los pesos de las neuronas. Las regularizaciones más usadas son L1, L2 o *weight decay*, las cuales evitan que los pesos tomen valores muy grandes regulando la presencia de las neuronas asociadas en el modelo. Por otra parte, la regularización *dropout* que reduce a cero los pesos de una fracción pequeña de neuronas por capa en cada iteración del entrenamiento.

Todos los pasos del entrenamiento se conjugan en la siguiente ecuación:

$$w_{i,j}^{next\ step} = w_{i,j} + \eta(y_j - \hat{y}_j)x_i,$$

donde $w_{i,j}$ es el peso entre las neuronas i-ésima y j-ésima, x_i el input o valor de entrada i-ésimo de la instancia actual, $\hat{y}_j$ el output j-ésimo o valor de salida para la instancia actual, y_j el valor j-ésimo de salida objetivo y η la learning rate, la cual se usa para calcular los pesos en cada iteración del entrenamiento de la red neuronal.

Predicción
Para hacer predicción, la red se aplica de forma iterativa. Es decir, para pronosticar a horizonte $h = 1$ se usan los datos históricos y para pronosticar a horizonte $h = 2$, se añade la predicción en el instante $t + 1$ hecha en el paso anterior como una nueva entrada a los datos históricos. Este proceso se realiza hasta que se hayan calculado todos los pronósticos requeridos.
La predicción de intervalos se realiza de forma diferente. Si la red neuronal ajustada se puede escribir como

$$y_t = f(y_{t-1}) + \varepsilon_t,$$

con el vector $y_{t-1} = (y_{t-1}, y_{t-2}, ..., y_{t-p})$ y ε_t asumido como homocedastico, podemos hacer la predicción utilizando *bootstrapped residuals*, el cual solo asume que los errores definidos como $\varepsilon_t = y_t - \hat{y}_{t-1}$ están incorrelados y no normalmente distribuidos.
Este método usa una simulación de la observación a horizonte $h = 1$ como $y_{T+1} = \hat{y}_{T+1} + \varepsilon_{T+1}$ donde $\hat{y}_{T+1}$ es la predicción de la observación y el error futuro ε_{T+1} es deconocido, en base a la suposición de que los errores futuros serán "parecidos" a los errores pasados, comunmente llamados residuos, y por ende se puede hacer el reemplazo de los errores futuros tomando muestras de los residuos. Este proceso se realiza hasta que se hayan calculado muchos futuros posibles, para luego calcular los intervalos de predicción a través de los percentiles para cada horizonte de pronóstico.
En este caso, podemos simular muestras futuras a través de valores aleatorios para ε_t remuestreando a partir de los valores históricos como se menciona anteriormente o también, a partir de una distribución normal. Luego de obtener ε_t la muestra futura queda como sigue:

$$y_{T+1}^* = f(\mathbf{y}_T) + e_{T+1}^*$$

$$y_{T+2}^* = f(\mathbf{y}_{T+1}) + e_{T+2}^*$$

$$...$$

De esta forma, según Hyndman (2014) en el Capítulo 11, generamos información o conocimiento de la distribución de todos los valores futuros basados en el ajuste de un modelo de redes neuronales.

Con lo expuesto anteriormente estamos en condiciones de especificar una red neuronal. Existe una clase de redes que tienen conexiones retroalimentadas que les permiten mantener y utilizar información sobre eventos pasados en la secuencia para predecir eventos futuros llamada Redes neuronales

recurrentes *(Recurrent Neural Network)* o RNN, por lo que se consideran una herramienta interesante para modelar y predecir datos temporales. Dentro de esta clase, hay varias arquitecturas específicas de RNN, como las RNN simples, las RNN bidireccionales, las RNN con puertas (como las LSTM y las GRU), entre otras, cada una con sus propias características y usos particulares. A continuación revisaremos sus principales características.

Redes neuronales recurrentes

Las redes neuronales recurrentes (RNN) son una clase de redes neuronales diseñadas para modelar datos secuenciales, como series temporales, texto y audio. Se utilizan para series temporales debido a su capacidad inherente para capturar dependencias temporales a lo largo del tiempo. A diferencia de las redes neuronales *feedforward* tradicionales, las RNN tienen conexiones retroalimentadas que les permiten mantener y utilizar información sobre eventos pasados en la secuencia para predecir eventos futuros. Esto las hace especialmente efectivas para capturar patrones de comportamiento a largo plazo en datos temporales, como tendencias, ciclos estacionales y correlaciones entre observaciones sucesivas. Las RNN pueden analizar y predecir series temporales con mayor precisión que otros modelos debido a su capacidad para modelar dependencias temporales complejas y adaptarse dinámicamente a diferentes patrones en los datos.

El tipo de redes neuronales añaden conexiones en las capas anteriores para que la neurona reciba como entrada la salida de la iteración anterior, modelando la dependencia temporal de las observaciones. Por ejemplo, la Figura 2.7 muestra la RNN más simple que tiene una neurona que recibe entradas, produciendo una salida, y enviando esa salida de vuelta a sí misma.

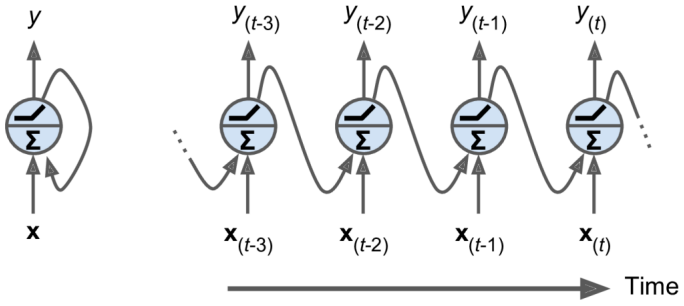

Figura 2.7: Ejemplo RNN. Fuente: Géron, A. (2022) Capítulo 15.

En este caso, para estimar el valor y_t la red RNN tiene en cuenta el valor anterior y_{t-1}, que a su vez tiene en cuenta el valor anterior y_{t-2} y así sucesivamente. Debido a esta construcción se tienen en cuenta las correlaciones entre observaciones pasadas y también se puede ajustar los pesos de forma tal que a los últimos valores se les asigne un mayor valor, similar a la idea del suavizado exponencial.

La Figura 2.8 muestra una capa de RNN desenrrollada a lo largo del tiempo, de la cual se puede representar su salida para el instante t como sigue:

$$\mathbf{y}_t = f(\mathbf{W}_x^\top \mathbf{x}_t + \mathbf{W}_y^\top \mathbf{y}_{t-1} + \mathbf{b}),$$

donde cada neurona tiene dos conjuntos de ponderaciones, los valores de entrada $\mathbf{x}_t$ y las salidas en los instantes anteriores $\mathbf{y}_{t-i}$ representados como los vectores $\mathbf{w}_x$ y $\mathbf{w}_y$. Si se extiende este concepto a una capa completa de neuronas, se consideran las matrices $\mathbf{W}_x$ y $\mathbf{W}_y$. Tanto $f(.)$ como $\mathbf{b}$ siguen siendo la función de activación y el vector de sesgos respectivamente.

Los valores de salida de las neuronas son, a su vez, inputs para para las interacciones siguientes, a esto Géron (2022) en el Capítulo 15 le llama *memoria* de sus instancias pasadas y este tipo de neuronas se conocen como neuronas *Long Short Term Memory (LSTM)*[16], las cuales son capaces de recordar

[16]La red neuronal LSTM toma el nombre de la neurona LSTM propuesta por Sepp Hochreiter y Jürgen Schmidhuber

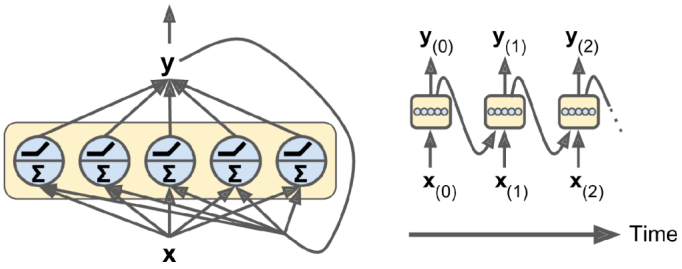

Figura 2.8: Ejemplo Salida de una capa de RNN. Fuente: Géron, A. (2022) Capítulo 15.

salidas a largo plazo como las salidas de iteraciones anteriores y también a corto plazo, como las iteraciones en instantes inmediatos.

La Figura 2.9 muestra el funcionamiento de una neurona LSTM de forma más exacta. Se añade una celula de estado C que permite mantener información pasada dentro de la red por más tiempo. Esencialmente es una estructura de memoria adaptable que permite que la red retenga y utilice información relevante de entradas anteriores a lo largo del tiempo, lo que facilita el aprendizaje de patrones a largo plazo en datos secuenciales como series temporales.
La idea clave es que la red puede aprender qué almacenar a largo plazo, qué no almacenar o desechar y qué leer de ella, Géron (2022). La neurona tiene tres vectores de entrada: vector de entradas x_t, vector de entrada de estado a corto plazo h_{t-1}, vector de entrada de estado a largo plazo C_{t-1}, y tres vectores de salida: vector de salida y_t, vector de salida de estado a corto plazo h_t y vector de salida de estado a largo plazo C_t.

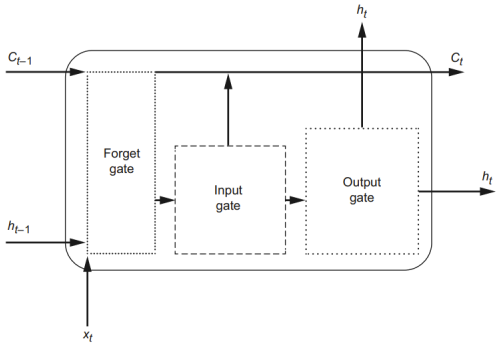

Figura 2.9: Ejemplo neurona LSTM. Fuente: Peixeiro, M. (2022) Capítulo 15.

El estado C_{t-1} atraviesa un filtro llamado *forget gate* en la cual olvidará parte de la información y luego agregará nueva memoria. Es decir, la celula olvida información del estado anterior, dando como resultado un valor de C_{t-1} actualizado o C'_{t-1} que finalmente tendrá la salida C_t a largo plazo. Sin embargo, C'_{t-1} se alimenta antes de salir de *forget gate* del filtro de x_t y h_{t-1} como se explica en el párrafo siguiente.
Las entradas x_t y h_{t-1} se combinan y se duplican, una copia pasa por una función de activación que determina qué información ha de ser eliminada en la puerta *forget gate*, cuyo resultado se combina con el proveniente de C_{t-1} generando como salida a C'_{t-1}. Otra copia pasa por *input gate* en la cual se determina qué información es relevante del elemento actual de la secuencia. En este paso, vuelve a existir una duplicación de x_t y h_{t-1}, pues una copia pasa por una función de activación *sigmoide* que

en 1997.

determina que información se conserva y otra por una función de activación tangente que regula la red computacionalmente, ambos resultados se combinan entre ellos generando un resultado que, a su vez se combina con C'_{t-1} proveniente de *forget gate* actualizando su valor. En es este punto, existen dos valores de salida desde *input gate** el valor C'_{t-1} actualizado y una copia de la combinación entre x_t y h_{t-1}, ambos son enviados a *output gate*.
En esta última estapa, la copia de x_t y h_{t-1} vuelve a pasar por una función de activación *sigmoide* que elimina memoria mientras que el valor C'_{t-1} ya actualizado se convierte en la entrada C_t la cual pasa por una función tangente, ambos resultados se combinan y generan la salida h_t. Este es el paso donde la información pasada, C_t, se utiliza para procesar la información del elemento presente de la secuencia.

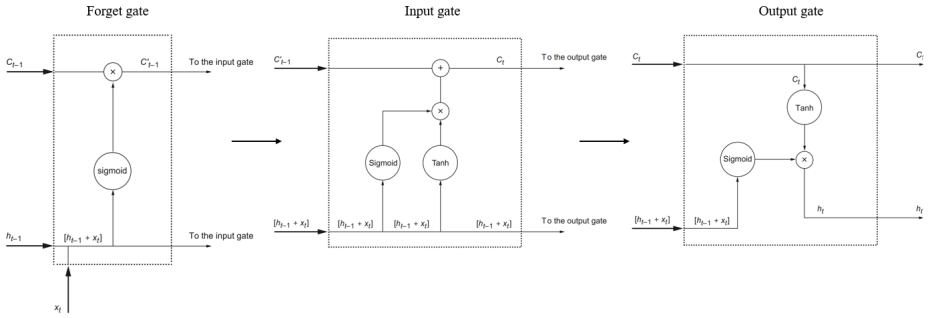

Figura 2.10: Funcionamiento neurona LSTM. Fuente: Peixeiro, M. (2022) Capítulo 15.

En resumen, *forget gate* determina qué información del pasado conservar, *input gate* por su parte determina qué información del paso actual se conserva para actualizar la memoria de la red. Por último, *output gate* utiliza la información del pasado guardada en memoria para procesar el elemento actual de una secuencia.
En particular, este tipo de neuronas evita el desvanecimiento del gradiente, a través de cambios en el estado oculto y salidas de una RNN clásica, lo que le permite capturar o recordar información por grandes periodos de tiempo, refinando cómo la nueva información se añade a la ya almacenada y también permitiendo el olvido de esta.
Finalmente, como menciona Makridakis et al. (2008) en el Capítulo 8, una desventaja de los modelos generados a partir de redes neuronales es que no permite el entendimiento exacto de los datos, pues no hay un modelo estocástico explícito sino que un acercamiento a la predicción mediante una "caja negra". Aunque, esto a su vez dota a las redes de patrones automáticos de los datos sin suposiciones teóricas sobre su distribución. Por otro lado, son capaces de predecir cuando modelos explicitos han fallado, adaptandose a las irregularidades y características inusuales de las series temporales. Por ejemplo, los modelos AR lineales pueden modelar la ciclicidad de forma simétrica, pero las NN son capaces de modelar ciclos asimétricos. Sumado a esto, se puede construir un modelo que capture la información de múltiples series temporales relacionadas entre sí, generando un buen rendimiento de las predicciones. Todo lo anterior, sumado a la gran capacidad de aumentar la multiplicidad de capas en conjunto con funciones no lineales le dan una flexibilidad a estos modelos, llegando a ser muy complejos y de gran coste computacional. En el desarrollo de pronósticos de series de tiempo, existen antecedentes de buenos resultados mediante el empleo combinado de modelos de series de tiempo (ARIMA) y redes neuronales.

2.3. Puntos débiles y puntos fuertes

El análisis de los métodos de predicción de series temporales revela una diversidad de ventajas y desventajas en su aplicación. Es por ello que comentaremos ciertas características más relevantes de

cada uno a tener en consideración.
ARIMA, conocido por su capacidad para modelar y predecir datos estacionales, incluyendo tendencia y cambios en la variabilidad a lo largo del tiempo, ofrece una estructura sólida y una comprensión intuitiva, con un modelo estadístico bien establecido, ampliamente utilizado y estudiado. Sin embargo, puede resultar sensible a la estacionalidad compleja, es decir, cuando hay múltiples estacionalidades o estacionalidades no lineales. Además, requiere datos estacionarios o que se puedan transformar en estacionarios mediante diferenciación. Esto puede ser una limitación en casos donde los datos tienen tendencias no lineales o cambios en la varianza a lo largo del tiempo.
En ocasiones, ajustar los parámetros de un modelo ARIMA puede ser un proceso complicado y laborioso que requiere experiencia y conocimiento específico. Sumado a esto, ARIMA no maneja naturalmente variables exógenas, lo que significa que puede tener dificultades para modelar series temporales que son influenciadas por factores externos no incluidos en la serie original.
A pesar de estas desventajas, sigue siendo una herramienta valiosa en el análisis de series temporales, especialmente cuando se combinan con técnicas adicionales como modelos de componentes estacionales o modelos de regresión.

Por otro lado, ETS es reconocido por su simplicidad y flexibilidad al manejar diferentes patrones de datos. Al igual que ARIMA, los modelos ETS pueden adaptarse a una amplia variedad de patrones, incluyendo tendencias, estacionalidades y cambios en la variabilidad.
ETS incorpora componentes clave al descomponer la serie temporal en error, tendencia y estacionalidad, lo que permite una comprensión más clara de la estructura de los datos y facilita la identificación de patrones importantes y de interpretación intuitiva. Aunque, aún cuando proporcionan dicha descomposición, la interpretación de los resultados puede ser complicada en casos donde hay interacciones complejas entre estas componentes o cuando los datos tienen estructuras poco usuales.
Una limitante de estos modelos es la sensibilidad a datos atípicos, lo que puede afectar negativamente la precisión de las predicciones. Los valores atípicos pueden distorsionar los componentes de error, tendencia y estacionalidad del modelo, lo que lleva a predicciones menos precisas.
Al igual que ARIMA, los modelos ETS asumen que los datos son estacionarios o pueden transformarse en estacionarios. Esto puede limitar su aplicabilidad en casos donde los datos tienen tendencias no lineales o cambios en la varianza a lo largo del tiempo y, también, pueden tener dificultades para capturar patrones de estacionalidad complejos en los datos, especialmente cuando hay múltiples estacionalidades o estacionalidades no lineales.

Los modelos de suavización exponencial presentan simplicidad y facilidad de implementación, lo que los hace accesibles incluso para aquellos sin experiencia avanzada en análisis de series temporales. También pueden adaptarse a una amplia variedad de patrones en series temporales, incluyendo tendencias, estacionalidades y cambios en la variabilidad a lo largo del tiempo.
Estos modelos son efectivos para capturar cambios graduales en los datos a lo largo del tiempo, lo que los hace útiles para predecir series temporales con patrones de cambio suave.
Pueden producir buenas predicciones con un bajo requerimiento de datos históricos, lo que los hace adecuados para pronosticar en situaciones donde se dispone de poca información histórica.
Sin embargo, estos modelos pueden tener dificultades para capturar cambios bruscos o repentinos en los datos, ya que tienden a suavizar las fluctuaciones en lugar de capturarlas explícitamente, es decir, tienden a ser especialmente útiles para series temporales con patrones de datos estables y mínima variabilidad.
Requieren la selección de parámetros de suavizado, como el factor de suavizado y el número de períodos de estacionalidad. La elección de estos parámetros puede afectar significativamente la calidad de las predicciones y puede ser, en ocasiones, subjetiva.
Al igual que ARIMA, los modelos de suavización exponencial no manejan naturalmente variables exógenas, lo que puede ser una limitación en casos donde los datos son influenciados por factores externos no incluidos en la serie original. Estos modelos pueden tener dificultades para capturar patrones de datos complejos, como aquellos que involucran múltiples estacionalidades o patrones no lineales.

Por su parte, XGBoost al ser un algoritmo de aprendizaje automático, puede superar las limitaciones de ARIMA y ETS al capturar relaciones no lineales y manejar grandes conjuntos de datos. Tiende a producir modelos con una alta precisión predictiva. Su capacidad para modelar relaciones complejas en los datos le permite capturar patrones sutiles y hacer predicciones precisas.
XGBoost está diseñado para manejar grandes volúmenes de datos de manera eficiente, lo que lo hace adecuado para analizar series temporales con muchas observaciones y/o muchas características, incluyendo técnicas de regularización incorporadas, como la penalización L1 (Lasso) y la penalización L2 (Ridge), que ayudan a prevenir el sobreajuste y mejorar la generalización del modelo.
Por otra parte, tiene varios hiperparámetros que deben ajustarse adecuadamente para obtener el mejor rendimiento del modelo. La selección inadecuada de hiperparámetros puede conducir a un sobreajuste o a un rendimiento no óptimo del modelo.
La interpretación de estos modelos puede ser más compleja en comparación con modelos más simples como ARIMA o suavización exponencial. Sumado a esto, puede ser sensible a la presencia de datos atípicos en la serie temporal, lo que puede afectar negativamente la precisión de las predicciones. Es importante preprocesar los datos adecuadamente para minimizar el impacto de los valores atípicos en el rendimiento del modelo.
Entrenar modelos de XGBoost puede ser computacionalmente costoso, especialmente cuando se utilizan conjuntos de datos grandes o se ajustan muchos hiperparámetros. Esto puede requerir recursos computacionales y tiempo de procesamiento significativos.

Prophet destaca por su capacidad para manejar eficazmente datos con múltiples fuentes de incertidumbre y estacionalidad. Puede modelar y capturar patrones de estacionalidad complejos, incluyendo efectos estacionales diarios, semanales y anuales, así como vacaciones y eventos especiales diferenciando por países. Sumado a esto, es fácil de implementar y de usar, lo que lo hace más accesible debido a que su interfaz intuitiva y su sintaxis no es altamente compleja.
Usa un enfoque de ajuste automático de parámetros que simplifica el proceso de modelado y reduce la necesidad de ajustes manuales, aunque sí presenta la opción de ajuste de parámetros de forma manual. Este método puede ser especialmente útil cuando se trabaja con conjuntos de datos con tendencias cambiantes y efectos estacionales complejos, ofreciendo un enfoque más automatizado y adaptable. Puede manejar datos faltantes y valores atípicos de manera efectiva, lo que lo hace robusto frente a la presencia de irregularidades en la serie temporal. Puede interpolar y suavizar datos faltantes, y puede modelar valores atípicos como eventos especiales en los datos.
Sin embargo, una desventaja de Prophet es que puede ser menos efectivo en la predicción de series temporales con patrones de corto plazo o datos con tendencias altamente no lineales. Su enfoque en capturar patrones de largo plazo puede limitar su capacidad para predecir cambios rápidos en los datos. Los datos de baja calidad o mal estructurados pueden afectar negativamente el rendimiento del modelo y la precisión de las predicciones. Es decir, puede manejar valores faltantes, pero si hay demasiados o si faltan datos en puntos críticos de la serie temporal, esto podría afectar la capacidad del modelo para capturar patrones y hacer predicciones precisas.
Aunque Prophet proporciona intervalos de confianza para las predicciones, la interpretación de estos intervalos puede ser menos intuitiva en comparación con otros modelos más simples como ARIMA o suavización exponencial. Es decir, dado que puede manejar eventos festivos y cambios de tendencia de manera flexible, siendo esto una ventaja en términos de modelado de datos complejos, también puede hacer que la interpretación de los intervalos de confianza sea más desafiante, ya que estos eventos pueden afectar la amplitud y la forma de los intervalos de confianza. En contraste, modelos más simples como ARIMA o suavización exponencial tienden a proporcionar intervalos de confianza más directos y fáciles de interpretar porque se basan en modelos más simples y no descomponen la serie temporal en componentes separadas.
Al igual que XGBoost, para conjuntos de datos grandes, Prophet puede requerir recursos computacionales y tiempo significativos.

Finalmente, las redes LSTM son capaces de modelar relaciones temporales complejas y no lineales en series de tiempo. Son capaces de capturar dependencias a largo plazo entre observaciones, lo que las hace efectivas para predecir patrones temporales complejos y capturar dinámicas no lineales en los datos.
Son flexibles y pueden adaptarse a una amplia variedad de datos, incluyendo series temporales con múltiples variables de entrada, datos de secuencia de texto y datos de secuencia de tiempo. La arquitectura de memoria a largo plazo de las redes LSTM permite retener y utilizar información relevante de entradas anteriores a lo largo del tiempo, lo cual permite modelar dependencias temporales a largo plazo.
Pueden aprender a ignorar valores atípicos y capturar patrones subyacentes en los datos, lo que les permite producir predicciones precisas incluso en presencia de irregularidades en la serie.
Aunque, tienden a requerir una cantidad significativa de datos para entrenar de manera efectiva, especialmente cuando se utilizan para problemas complejos o para capturar patrones a largo plazo en los datos. La falta de datos puede conducir al sobreajuste o a modelos poco confiables.
Son modelos que pueden llegar a ser muy difíciles de interpretar, no permiten un entendimiento exacto de los datos como comentábamos anteriormente, ya que no se genera un modelo estocástico explícito. La comprensión de cómo se toman las decisiones y se generan las predicciones en una red LSTM puede ser un desafío, lo que puede dificultar la confianza en los resultados del modelo.
Tienen varios hiperparámetros que deben ajustarse adecuadamente para obtener el mejor rendimiento del modelo. La selección inadecuada de hiperparámetros puede conducir a un sobreajuste o a un rendimiento no óptimo del modelo.
Al igual que en los últimos modelos mencionados, se sacrifica el coste computacional. Entrenar modelos LSTM puede ser computacionalmente costoso, especialmente cuando se utilizan conjuntos de datos grandes o se ajustan muchos parámetros, lo que puede requerir recursos computacionales y tiempo de procesamiento significativos.

A modo de resumen, la elección entre estos métodos dependerá de la naturaleza específica de los datos y del contexto de aplicación. Combinar estos enfoques puede resultar beneficioso, ya que cada uno tiene fortalezas y debilidades diferentes.
La consideración de estas herramientas ofrece un abanico de posibilidades para seleccionar el modelo más apropiado en función de las características y la complejidad de las series temporales, así como de los objetivos de predicción deseados y de los recursos disponibles.

Capítulo 3

Revisión de métricas

En el capítulo anterior, exploramos diversos métodos para modelar series temporales, cada uno con sus propias características que los hacen adecuados en función de la naturaleza de los datos. Sin embargo, también es importante reconocer que estos métodos pueden presentar desventajas, tanto en términos del modelo en sí como de los recursos disponibles.
Ahora, llegamos a un punto crucial: una vez que hemos seleccionado y ajustado los modelos utilizando la muestra de datos, surge la necesidad de evaluar su precisión mediante diversas métricas.
La evaluación precisa y confiable de modelos de predicción en series temporales es crucial para la toma de decisiones en una amplia variedad de campos. En este sentido, las métricas son herramientas para evaluar la precisión de estos modelos, permitiéndonos comprender la magnitud de los errores de pronóstico en relación con los valores reales, los cuales pueden expresarse como porcentajes. En este capítulo, se revisarán algunas métricas como el Error Relativo Porcentual Absoluto Medio (MAPE) y algunas modificaciones del mismo, incluido el Error Porcentual Absoluto Medio Simétrico (sMAPE), entre otros. Se seguirá muy de cerca lo expuesto por Hyndman et al. (2008) y Hyndman and Koehler (2006).

3.1. Criterios para la elección de métricas

La selección de métricas de evaluación para comparar métodos de predicción en series temporales, como MAPE, sMAPE, gMAPE, rMAPE y MASE, se fundamenta en una consideración de varios factores que abordan la casuística específica de los datos y la necesidad de evitar redundancias en la evaluación de la precisión de los modelos. A continuación se mencionarán algunos de ellos.

Representación de errores en diferentes contextos: cada métrica tiene una forma particular de representar los errores de predicción. Por ejemplo, MAPE y sMAPE evalúan el error en términos porcentuales, siendo útiles cuando se busca comprender el error relativo en relación con los valores reales. Por otra parte, gMAPE introduce una ponderación por el tamaño de las series, lo que puede ser beneficioso cuando se comparan conjuntos de datos de diferentes escalas. Asimismo, rMAPE mide el error relativo porcentual medio ajustado considerando la variabilidad de los datos, lo que es útil en situaciones donde la dispersión de los valores es alta. El MASE, por su parte, evalúa el error en relación con un modelo de referencia, proporcionando una medida de error relativo que es independiente de la escala y, por lo tanto, adecuada para comparar modelos en diferentes conjuntos de datos.
Evitar la redundancia en la evaluación: las métricas seleccionadas ofrecen enfoques distintos para medir la precisión de los modelos. Al elegir un conjunto diverso de métricas, se evita la redundancia al capturar aspectos diferentes y complementarios de la calidad de la predicción. Esto permite una evaluación más completa y equilibrada de la eficacia de los métodos de predicción, considerando múltiples dimensiones

de la precisión y ofreciendo una visión más detallada de su desempeño.
Adaptación a diferentes escenarios y requisitos: cada métrica responde mejor a ciertos contextos y requisitos específicos. Dependiendo de la casuística de los datos y los objetivos de la predicción, ciertas métricas pueden ser más adecuadas que otras. La variedad de métricas seleccionadas permite adaptarse a diferentes situaciones, brindando una evaluación más completa y precisa de la idoneidad de los modelos para cada escenario particular.

En resumen, la elección de estas métricas se basa en su capacidad para ofrecer perspectivas complementarias y no redundantes sobre la precisión de los modelos de predicción en series temporales. Esta selección permite una evaluación más holística y precisa de los modelos, teniendo en cuenta las peculiaridades de los datos y proporcionando una base sólida para la comparación entre diferentes enfoques de predicción.

3.2. Definición de las métricas de error

Basandonos en Hyndman et al. (2008) y Hyndman and Koehler (2006) se definen las siguientes medidas como sigue:

Errores absolutos
Consideraremos el error absoluto de predicción con horizonte h a $e_{t+h} = x_{t+h} - \hat{x}_{t+h}$, donde x_{t+h} representa el valor real de la serie temporal en el tiempo $t+h$ y $\hat{x}_{t+h}$ representa el valor pronosticado de la serie temporal en el tiempo $t+h$. Donde el error absoluto de predicción a un paso o a horizonte $h = 1$ queda como $e_{t+1} = x_{t+1} - \hat{x}_{t+1}$.
Ambos errores se encuentran en la misma escala que los datos originales y por consiguiente, las siguientes medidas también:
Error absoluto medio (MAE) = $media(|e_t|)$, $t = 1, \ldots, n$, con n número total de observaciones.
Error cuadrático medio (MSE) = $media(e_t^2)$, $t = 1, \ldots, n$, con n número total de observaciones.
Raíz del error cuadrático medio (RMSE) = $\sqrt{MSE}$, lo que es equivalente a RMSE = $\sqrt{media(e_t^2)}$, $t = 1, \ldots, n$, con n número total de observaciones.
Estas medidas son útiles al comparar diferentes modelos predictivos que se aplican a los datos con la misma escala. Hyndman et al. (2008) recomienda el uso de MAE por su facilidad de calculo.

Errores relativos
Consideraremos el error relativo de la predicción a un paso a $p_t = 100e_t/x_t$. En este caso, por su construcción la escala se vuelve independiente lo que significa que no depende directamente de la escala específica en la que se expresen los valores de la serie temporal y por ende se puede usar para comparar distintas series con escalas diferentes.
Error relativo porcentual absoluto medio (MAPE) = $media(|p_t|)$, $t = 1, \ldots, n$, con n número total de observaciones.
A pesar de su fácil interpretación, si los valores reales son cero o están próximos a cero esta medida produce valores infinitos o indefinidos. Además de tener una distribución sesgada o asimétrica cuando cualquier valor real de las observaciones es cercano a cero.
Error porcentual absoluto medio simétrico (sMAPE) = $media\,(200|x_t - \hat{x}_t|/(x_t + \hat{x}_t))$, $t = 1, \ldots, n$, con n número total de observaciones.
Esta medida sigue considerando una división por un número cercano a cero cuando x_t es igual a cero, ya que es probable que el valor pronosticado $\hat{x}_t$ también sea cercano a cero, puesto que los modelos de pronóstico tienden a predecir valores similares a los observados en los datos históricos. Esto significa que, incluso con el sMAPE, el cálculo del error aún implica la división por un número cercano a cero.

Otras medidas conocidas son:

Error Porcentual Absoluto Medio Geométrico (gMAPE) = $media(\sqrt[n]{\prod_{t=1}^{n} p_t})$, con n número total de observaciones. Este error calcula la media geométrica de los errores porcentuales absolutos, haciendola más equilibrada en comparación con MAPE y puede ser útil en situaciones donde los errores extremos deben ser considerados cuidadosamente.
Error Porcentual Absoluto Medio Relativo (rMAPE) = $\frac{MAPE}{\frac{1}{n}\sum |x_t|}$, con n número total de observaciones. Este error divide al MAPE por el promedio de los valores reales. Además, tiene en cuenta el tamaño relativo de los valores pronosticados, útil en casos en los que la magnitud de los valores pronosticados varía significativamente.
Robust MAPE (RMAPE) = $100\frac{1}{h-2N}\sum_{i=N+1}^{h-N} \frac{|e_t^*|}{|x_t^*|}$, con $2N < h$. Esta medida de error es una modificación propuesta por la empresa SDG Group que se diseñó para ser más robusta y resistente a la influencia de valores extremos en los datos de la serie temporal, la cual trunca el MAPE original simétricamente, considerando solo una parte de los datos.
Es decir, si los errores de predicción a horizonte h son representados por $e_1, .., e_h$, se consideran a $e_1^* \leq, .., \leq e_h^*$ como dichos errores ordenados de forma creciente y los valores de las observaciones de la serie $x_1^*, .., x_h^*$ asociados a dichos errores. El RMAPE se calcula con los $h - 2N$ errores centrales, sin considerar tanto a los N errores más grandes como a los N más pequeños. El valor de N se estableció en $N = 1$ para series cortas, es decir $h = 7$ y $N = 2$ para las series largas, con $h = 24$.
Una nueva medida de error que proponen Kim y Kim (2016) para intentar solventar el problema de la sensibilidad a los valores extremos y la asimetría en la penalización de los errores positivos y negativos del MAPE es incorporar la función arcotangente, dado que tiene la propiedad de comprimir los valores hacia el centro, lo que puede reducir el efecto de los valores extremos y suavizar la distribución de los errores. Esto puede hacer que la medida sea menos sensible a los valores extremos.
Error porcentual absoluto arctangente medio (MAAPE) = $media(arctan(|p_t|))$, $t = 1, \ldots, n$, con n número total de observaciones.
La función arcotangente tiene un rango finito, lo que evita que el MAAPE se dispare a infinito cuando los valores reales se aproximan a cero. Esto hace que sea más robusto y menos propenso a sesgos cuando se enfrenta a valores pequeños en la serie temporal.

Errores escalados
Mientras que los errores relativos son útiles para comparar la precisión relativa entre diferentes modelos, los errores escalados proporcionan una medida de la precisión del modelo en relación con un modelo de referencia específico en una escala más intuitiva. Por ejemplo, si el MAPE es del 10 %, significa que, en promedio los pronósticos difieren en un 10 % del valor real. En cambio, un error escalado de 0,5 indica que el modelo es dos veces más preciso que el modelo de referencia. En este caso, consideraremos la siguiente medida:
Error escalado absoluto medio (MASE) = $media(|q_t|)$ donde

$$q_t = \frac{e_t}{\frac{1}{n-1}\sum_{i=2}^{n} |x_i - x_{i-1}|},$$

es el error escalado para cada punto de tiempo t, con n número total de observaciones en la serie temporal. Al igual que antes, por su construcción es independiente de la escala de los datos originales. Tanto el sMAPE como el MASE solventan el problema de que el MAPE genere infinitos o indefinidos.

Resumiendo, cada métrica de evaluación de modelos de series temporales tiene sus propias ventajas y limitaciones. MAPE es particularmente útil cuando se necesita entender la precisión en términos porcentuales. brindando una comprensión directa de fácil interpretación. Sin embargo, su debilidad radica en su sensibilidad a valores atípicos, lo que puede distorsionar la evaluación en datos extremos mientras que gMAPE es apropiada en dicha situación, para datos con valores extremos, al ponderar las desviaciones porcentuales, lo que lo hace adecuado para pronósticos en mercados volátiles o con eventos inusuales.
RMAPE, al ser un truncamiento simétrico del MAPE, se centra en la evaluación de la precisión del pronóstico al trabajar solo con una porción central de los errores de predicción, lo que lo hace robusto

ante valores extremos. Es particularmente útil en entornos con una variabilidad significativa en los datos de pronóstico, como industrias con productos de diferentes tamaños o escalas. Sin embargo, su limitación reside dicha exclusión, es decir, puede llevar a una visión parcial de la precisión del modelo.
La métrica sMAPE es valiosa en contextos donde tanto las sobreestimaciones, entendidas como cuando el valor pronosticado excede al valor real, como las subestimaciones son críticas para evaluar la calidad de los pronósticos. Dado que la sMAPE penaliza tanto las sobreestimaciones y las subestimaciones de manera equitativa, se adapta especialmente bien a situaciones donde la dirección de los errores, ya sea sobreestimación o subestimación, impacta significativamente en la evaluación del pronóstico.
El MASE al proporcionar una medida de la precisión del modelo en relación con un modelo de referencia, es útil para evaluar la precisión relativa del modelo en diferentes contextos.
Por tanto, cada métrica ofrece un enfoque diferente para evaluar la precisión de los modelos de series temporales. La elección de la métrica adecuada debe basarse en la casuística de los datos y las necesidades específicas de la evaluación, teniendo en cuenta las fortalezas y debilidades de cada métrica para obtener una evaluación más completa y precisa de los modelos. Integrar múltiples métricas puede proporcionar una visión más holística de la calidad del modelo en diferentes escenarios y contextos de aplicación.

Para la aplicación a datos reales hemos elegido las siguientes métricas: MAPE, sMAPE, gMAPE, RMAPE y MASE. La selección de estas métricas se justifica, como comentabamos, por su no-redundancia, es decir, cada una aporta información valiosa y complementaria sobre la precisión del modelo de pronóstico. Al usar estas métricas en combinación, se evita depender únicamente de una métrica que podría pasar por alto ciertos aspectos importantes de la precisión del pronóstico, lo que garantiza una evaluación más completa y precisa.

Capítulo 4

Software para la modelización de series temporales

A la hora de elegir un lenguaje para trabajar en el análisis de series temporales, la elección de Python tiene fundamentos sólidos, especialmente si la empresa ya lo utiliza en su infraestructura. Peixeiro (2022) destaca su popularidad debida a la versatilidad que tiene, sin limitarse a la computación estadística, permitiendo la programación de sitios web, realizar aprendizaje automático, implementar modelos y más. A esto podemos agregar su facilidad y rapidez de aprendizaje y la disponibilidad de bibliotecas especializadas. Además, en un entorno donde la empresa ya usa Python, integrar nuevas soluciones basadas en el mismo lenguaje puede simplificar la gestión de recursos humanos y técnicos, así como la compatibilidad con el resto del ecosistema de la empresa. Esto facilita la colaboración entre equipos, la reutilización de código y el mantenimiento a largo plazo. En resumen, al optar por Python, se aprovecha la familiaridad y la robustez del ecosistema existente, lo que puede acelerar el desarrollo y la implementación de soluciones de análisis de series temporales.

Particularmente en el análisis de series temporales, el ecosistema de Python ofrece una amplia gama de bibliotecas y herramientas especializadas para implementar modelos y realizar predicciones precisas. Estas bibliotecas están diseñadas para manejar desde tareas básicas hasta técnicas avanzadas de predicción de series temporales. Entre las bibliotecas más prominentes Peixeiro (2022) en el Capítulo 19 menciona a *Pmdarima*, *Prophet*, *NeuralProphet* y *PyTorch Forecasting*. Sumado a esto, con nuestra investigación podemos añadir a *Statsmodels*, *TensorFlow*, y *Scikit-learn*, como opciónes recurrentes, cada una con sus propias fortalezas y capacidades específicas para abordar distintos aspectos de la modelización de series temporales.

Si bien existen múltiples opciones de bibliotecas en Python para trabajar con series temporales, es esencial considerar las que brinden una combinación equilibrada de funcionalidades avanzadas, facilidad de uso, interfaz sencilla y flexibilidad para abordar una amplia variedad de problemas. Entre las bibliotecas más completas y versátiles se destacan *Sktime* y *Skforecast*. La primera de ellas ofrece un marco de trabajo modular y extensible que abarca desde modelos clásicos hasta técnicas de aprendizaje automático más avanzadas, permitiendo una implementación flexible y eficiente de modelos de series temporales, siendo su principal objetivo la creación de una API unificada para múltiples tareas de series temporales (Löning et al.,2019). Por otro lado, Amat y Escobar (2024) destacan a *Skforecast* como un conjunto integral de herramientas para entrenamiento, validación y predicción de series temporales, proporcionando herramientas poderosas y especializadas para modelar y predecir datos temporales con una estructura clara y un enfoque intuitivo, teniendo acceso a una amplia gama de funcionalidades como ingeniería de características, selección de modelos, ajuste de hiperparámetros y más.

Al optar por estas dos librerías, se obtiene acceso a una variedad de métodos y herramientas robustas para el análisis y la predicción de series temporales. Estas bibliotecas no solo ofrecen una amplia gama de algoritmos y técnicas de modelado, sino también una sintaxis coherente y una documentación

detallada que facilita su uso y aprendizaje. La combinación de la versatilidad de *Sktime* y la especialización de *Skforecast* las posiciona como opciones interesantes para abordar diversos desafíos en la modelización y predicción de series temporales en Python.

4.1. Librerías en el mercado: Sktime y Skforecast

En esta sección se comentarán algunas de las variadas opciones y funcionalidades disponibles en *Sktime* y *Skforecast* para el análisis de series temporales, para luego poder aplicarlas a datos proporcionados por la empresa colaboradora en el siguiente Capítulo. Para este apartado nos guiaremos muy de cerca por Amat y Escobar (2024) y Löning et al. (2019).

Comenzando por *Sktime*, esta presenta una fácil la manipulación de series temporales, permitiendo cargar, almacenar y visualizar datos temporales en diferentes formatos. Por ejemplo, ofrece métodos para leer datos de archivos CSV, Excel o bases de datos, así como para transformar y preprocesar series temporales para su posterior análisis. Además, la biblioteca facilita la visualización de datos temporales mediante gráficos y herramientas interactivas que permiten explorar y comprender mejor los patrones y tendencias presentes en las series temporales. Ofrece herramientas para el preprocesamiento, limpieza y transformación de series temporales, incluyendo técnicas de imputación de valores faltantes y normalización de datos.
Una parte importante es que esta librería incluye una amplia gama de modelos de series temporales, desde métodos clásicos como ARIMA, ETS y suavización exponencial hasta enfoques más avanzados como modelos basados en aprendizaje automático, Prophet, XGBoost y NN. Los dos últimos modelos no vienen integrados, pero se pueden implementar a través de la función *make_reduction* que trae la librería y que produce un estimador *Sktime*. Esto nos permite ajustar los modelos que hemos revisado en el Capítulo 2. Además, tiene la capacidad de combinar múltiples modelos o estimadores de series temporales para crear flujos de trabajo más complejos y sofisticados.
De forma más detallada, *Sktime* utiliza la función *temporal-train-test-split* para dividir la muestra en un conjunto de entrenamiento y otro de prueba, para luego aplicar cada método en un objeto llamado *forecaster*, por ejemplo, para modelar utilizando suavización exponencial se tiene *forecaster = ExponentialSmoothing()*, a un horizonte de predicción en un objeto llamado *ForecastingHorizon* que define el horizonte utilizando cualquier tipo de índice admitido como argumento.
Esta librería unifica diferentes métodos de predicción desde otras librerías de uso común. Los métodos que trae por defecto son los siguientes: *ExponentialSmoothing* el cual es un pronosticador de suavizamiento exponencial de Holt-Winters y *autoETS* de la librería *Statsmodels*, *ARIMA* y *AutoARIMA* de la librería *Pmdarima*, *AutoARIMA* de *Statsforecast* y *Prophet* de *Facebookprophet*, para éste último el formato de los datos se debe cambiar al propuesto por Facebook, donde los índices deben ser del tipo *DatetimeIndex* de *Pandas*, es decir, utilizando la función *to-datetime()*.
Un punto interesante es que *Sktime* puede crear pronosticadores que no están unificados de otras librerías a través de la función *make_reduction* que mencionabamos anteriormente. Esta función funciona como un "metaestimador" que permite el uso de cualquier estimador compatible con la API[1] de la librería *Scikit-learn* para realizar pronósticos. En nuestro caso, la utilizamos para los métodos de ML *XGBoost* con *GradientBoostingRegressor* y para métodos de redes neuronales con *MLPRegressor*, ambos provenientes de la librería *Sklearn*.
La etiqueta *strategy="recursive"* dentro de *make_reduction* hace que el método se aplique secuencialmente a cada punto de la serie temporal dentro de las ventanas, utilizando iterativamente las predicciones previas como parte de las entradas para predecir el siguiente punto en la serie de la siguiente forma: la serie temporal se divide en ventanas de una longitud determinada con *window_length* igual a un número entero, estas ventanas se desplazan a lo largo de la serie temporal. Comenzando

[1] API: Application Programming Interface, se refiere a un conjunto de reglas, protocolos y herramientas que permiten la comunicación y la interacción entre distintos componentes de software. En este caso, siguen una interfaz de programación común.

desde el primer punto de la ventana, se utiliza el regresor o método (*GradientBoostingRegressor* o *MLPRegressor*) para hacer una predicción sobre ese punto utilizando la información histórica dentro de la ventana. La predicción obtenida para ese punto se agrega a la serie temporal parcial que contiene las predicciones anteriores y se avanza al siguiente punto dentro de la ventana para repitir el proceso, utilizando la serie temporal parcial (que ahora incluye las predicciones anteriores) como parte de las entradas para predecir el siguiente punto. Este proceso continúa iterativamente para cada punto dentro de la ventana, utilizando las predicciones previas como parte de las entradas para predecir el siguiente punto. Al final del proceso, se obtienen las predicciones para todos los puntos dentro de la ventana, utilizando la estrategia recursiva para aplicar el regresor secuencialmente en cada punto.
Cada método admite transformaciones de pre o posprocesamiento a través de *TransformedTargetForecaster* tanto para la tendencia como para la componente estacional con las funciones *Deseasonalizer* y *Detrender* respectivamente.
Por otro lado, también proporciona herramientas para la selección y ajuste de hiperparámetros de modelos mediante validación cruzada y optimización para mejorar la precisión predictiva. *Sktime* proporciona estrategias de ajuste de parámetros que siguen la filosofía de la librería *Scikit-learn*, por ejemplo, para la predicción utilizando validación cruzada se utiliza la función *ExpandingWindowSplitter* que crea particiones progresivamente más grandes a medida que avanza en la serie temporal. Por cada división, el conjunto de entrenamiento incluye todos los datos anteriores al punto de corte actual, mientras que el conjunto de prueba consta solo del siguiente punto en la serie temporal.
La función textitForecastingGridSearchCV se utiliza para realizar búsqueda de hiperparámetros en un diccionario definido como *param_grid* mediante la validación cruzada ajustada anteriormente en textitExpandingWindowSplitter. Se puede acceder a los parámetros ajustados con *best_params_*. Podemos mencionar que también existe *ForecastingRandomizedSearchCV* que en lugar de proporcionar una grilla con valores específicos para cada hiperparámetro, se especifica una distribución de búsqueda para cada hiperparámetro. Estas distribuciones pueden ser, por ejemplo, uniforme, normal o discreta, y el algoritmo de búsqueda aleatoria seleccionará aleatoriamente valores de hiperparámetros dentro de estas distribuciones.
Asimismo, ofrece métricas y herramientas para evaluar el rendimiento de los modelos de series temporales, incluyendo métricas clásicas como las que hemos revisado en el Capítulo 3, y herramientas para visualizar y comparar las predicciones con los valores reales. Las métricas gMAPE y rMAPE se deben transcribir de forma manual, adecuando un % de exclusión de los errores más grandes y más pequeños para ésta última métrica.
Esta librería cuenta con la utilización de series temporales exógenas, que no se pronostican, pero que son útiles para realizar el pronóstico en cuestión y también con pronósticos multivariados, utilizando más de una variable. Se pueden realizar intervalos de predicción con *predict_interval* y cuantiles de predicciones con *predict_quantiles*. También cuenta con una forma de ensamblaje para combinar múltiples métodos de pronóstico para realizar una selección automatizada con la función *MultiplexForecaster* que funciona como una lista de múltiples pronosticadores que permiten combinarlos para formar un "meta-estimador" que combina sus predicciones para generar una predicción final. La idea detrás de esta función es aprovechar las fortalezas de múltiples métodos de pronóstico para mejorar la precisión de las predicciones. Puede ser útil cuando se tienen varios algoritmos de pronóstico y se quiere aprovechar la diversidad de enfoques para mejorar la calidad general de las predicciones.
También podemos rescatar que cuenta con funciones *diff*, que permite calcular la diferencia de primer orden entre observaciones sucesivas en una serie temporal, *BoxCoxTransformer* para aplicar transformaciones Box-Cox que pueden ayudar a estabilizar la variabilidad de la serie y *SeasonalDifference* que realiza una diferenciación estacional de una serie temporal.
Finalmente, esta librería cuenta con una documentación detallada y ejemplos prácticos que ayudan a los usuarios a comprender y utilizar eficazmente la biblioteca. A esto se suma la comunidad activa que contribuye al desarrollo continuo de la biblioteca, lo que brinda soporte y actualizaciones constantes.

Por su parte, *Skforecast* permite visualizar las predicciones generadas por los modelos, así como las series temporales originales y sus características relevantes. Esta librería está diseñado para trabajar

en conjunto con otras bibliotecas de Python, como *Pandas*, facilitando la manipulación y preparación de datos.

Al igual que *Sktime*, hace uso de regresores compatibles con la API *Scikit-learn*. Incluye modelos clásicos como ARIMA y suavización exponencial, así como modelos de aprendizaje automático más avanzados, como *Random Forests* y *Gradient Boosting*. A diferencia de *Sktime*, utiliza la biblioteca *Statsmodels* para algunos de sus métodos que no están incluidos de forma directa, como suavización exponencial y ETS. Como solución, se puede definir un adaptador para hacer que el regresor *SimpleExpSmoothing* sea compatible con la librería. Esto implica una dependencia externa y la necesidad de envolver estas funciones para su uso.

De forma más detallada, aplica cada método en un objeto *Forecaster* a un horizonte de predicción dentro de la función *predict* llamado *steps* definido como número entero. Para modelos ARIMA unifica los pronosticadores de la librería *Statsmodels* solo con algunos elementos para mejorar la velocidad en comparación a lo que hace la librería *Pmdarima* que importa todos los elementos. El objeto *ForecasterSarimax* encapsula estos modelos donde la etiqueta regressor puede ser *Sarimax*, *auto_arima* u cualquier método compatible con *Statsmodels*.

Para la utilización de otros métodos, hace uso de los regresores de la librería *Scikit-learn*, por ejemplo para el método *XGBoost* se importa *XGBRegressor* de la librería *Xgboost* compatible con la API de *Scikit-learn* y para redes neuronales *MLPRegressor* directamente de *Sklearn*. Sin embargo, para los métodos de suavización exponencial, ETS y Prophet *Skforecast* no proporciona una integración directa por lo que no los hemos considerado en el desarrollo de este trabajo.

En el caso de las métricas, la librería hace uso del módulo *sklearn.metrics* de la biblioteca *Scikit-learn* (Sklearn) que proporciona una variedad de métricas para evaluar la calidad de los modelos de aprendizaje automático como *mean_absolute_percentage_error* y también permite el ingreso de métricas personalizadas de forma manual según sea necesario.

Ambas bibliotecas reconocen la importancia de la validación cruzada en el contexto de series temporales y brindan herramientas para implementarla de manera adecuada. La elección entre ellas puede depender de las preferencias personales del usuario, así como de otros aspectos específicos del análisis de series temporales que estén abordando. *Skforecast* tiene implementada un tipo especial de validación cruzada que se aplica al periodo(s) anterior(es) de la serie temporal con un objeto *Backtesting*, en el caso de modelos ARIMA *backtesting_sarimax*. Se puede realizar utilizando una variedad de técnicas, como simples divisiones de pruebas de tren, ventanas móviles o ventanas expandidas. En nuestro caso, se utilizarán para la aplicación a datos reales, *Backtesting* sin reacondicionamiento fijando el prámetro *fixed_train_size=False* donde el modelo se entrena una sola vez y se usa secuencialmente sin actualizarlo, siguiendo el orden temporal de los datos, pues es mucho más rápido que otros métodos que requieren volver a entrenar el modelo cada vez. Sin embargo, el modelo puede perder su poder predictivo con el tiempo al no incorporar la última información disponible como comentamos en el Capítulo 2 en el apartado de Gradient Boosting. Para suplir un poco esta desventaja utilizamos la etiqueta *refit=True* que después de completar el *backtesting* el modelo se reajustará utilizando todos los datos de entrenamiento disponibles (es decir, utilizando todo el conjunto de datos históricos) antes de devolver los resultados finales del *backtesting*. Este reajuste implica un aumento del tiempo.

También proporciona estrategias de ajuste de parámetros a través de la función *grid_search_sarimax*, que realiza una búsqueda exhaustiva de hiperparámetros utilizando la técnica de búsqueda de cuadrícula para encontrar la combinación óptima de hiperparámetros en el diccionario definido como *param_grid* que mejor se ajusta a los datos de la serie temporal. Además cuenta con *random_search_sarimax* que, a diferencia de la búsqueda en cuadrícula que prueba todas las combinaciones posibles de hiperparámetros y retrasos, realiza una búsqueda aleatoria que muestrea un número fijo de valores de las posibilidades especificadas. El número de combinaciones que se evalúan se define en la etiqueta *n_iter*. El proceso de búsqueda en cuadrícula se puede ampliar para comparar varios modelos a través de un bucle que itera sobre cada regresor y aplicando la función *grid_search_forecaster*. Este enfoque permite una exploración más exhaustiva y puede ayudar a seleccionar el mejor modelo.

Finalmente, al igual que con *Sktime*, cuenta con las funciones *differentiation* que permite calcular la diferencias hasta segundo orden. A diferencia de *Sktime*, esta librería no trae implementada transfor-

maciones Box-Cox de forma directa, pero se pueden realizar desde la librería *SciPy*. Además, existe la posibilidad de adaptar los parámetros con la etiqueta *Regressor parameters*, así como de incluir variables exógenas al ajustar el modelo con el argumento *exog*. Se pueden realizar intervalos de predicción con la función *predict_interval*.
En este caso también se cuenta con una documentación exhaustiva que proporciona ejemplos y explicaciones claras sobre su funcionamiento y uso y una comunidad y soporte activo para la colaboración continua.

A conitnuación se presentan algunas limitaciones provenientes de los datos de forma general que consideramos importante a tener en cuenta. Es decir, trabajar con conjuntos de datos de prueba muestrales muy acotados presenta varias limitaciones que pueden afectar la calidad y la generalización de los modelos desarrollados, muchas veces independiente de cual librería se esté usando.
Una de ellas es la representatividad insuficiente que puede tener la muestra. Un conjunto de datos de prueba pequeño puede no capturar completamente la diversidad y complejidad del mundo real. Esto puede llevar a que los modelos entrenados no sean representativos de la variabilidad presente en el dominio de aplicación, indepediente de cual librería se utilice.
Junto con lo anterior se produce una dificultad para identificar relaciones sutiles, dado que en conjuntos de datos pequeños, puede ser más difícil descubrir patrones o relaciones más sutiles y complejas. Esto es especialmente relevante en problemas donde se requiere un gran volumen de datos para revelar tendencias significativas.
Del mismo modo, se presenta la sensibilidad a *outliers*. La presencia de valores atípicos en un conjunto de datos pequeño puede tener un impacto desproporcionado en el rendimiento del modelo. Un *outlier* puede influir significativamente en las estadísticas descriptivas y afectar la capacidad del modelo para generalizar. En este caso, ambas librerías no incluyen funcionalidades específicas para manejar valores atípicos en series temporales de forma integrada.
Además de eso exite el riesgo de sobreajuste. Cuando se trabaja con conjuntos de datos pequeños, existe un mayor riesgo de sobreajuste. Los modelos pueden aprender patrones específicos del conjunto de datos de entrenamiento que no se generalizan bien a nuevos datos. Esto puede resultar en un rendimiento deficiente en situaciones del mundo real, indepediente de cual librería se utilice.
Respecto a la validación cruzada, la cual es una técnica importante para evaluar el rendimiento del modelo, con conjuntos de datos de prueba pequeños, la capacidad de realizarla de manera efectiva se ve comprometida o limitada, lo que puede llevar a evaluaciones menos confiables de la generalización del modelo.
Asimismo, las métricas de evaluación del rendimiento pueden ser más variables y menos confiables con conjuntos de datos pequeños, provocando inestabilidad en la evaluación del rendimiento, lo cual puede llevar a dificultar la toma de decisiones informadas sobre la eficacia de un modelo.
En general, es crucial ser consciente de las limitaciones asociadas con conjuntos de datos de prueba pequeños y adoptar estrategias adecuadas, como la selección cuidadosa de técnicas de validación, la consideración de técnicas de aumento de datos, y la interpretación crítica de los resultados del modelo.

Capítulo 5

Aplicación a datos reales

Comenzaremos esta sección con la presentación de los primeros datos que fueron proporcionados por la empresa colaboradora SDG Group. Podemos ver en la Figura 5.1 los primeros 10 datos diarios, en la columna titulada como "ds" la fecha y en la columna "y" el valor de la observación.

	ds	y
0	2017-09-10	292.0
1	2017-09-11	875.0
2	2017-09-12	714.0
3	2017-09-13	774.0
4	2017-09-14	510.0
5	2017-09-15	829.0
6	2017-09-16	371.0
7	2017-09-17	245.0
8	2017-09-18	864.0
9	2017-09-19	636.0

Figura 5.1: Serie temporal ingresada en Python a través de Pandas.

La Figura 5.2 muestra el gráfico secuencial de la serie. Dado que la serie presenta cierta variabilidad a lo largo del tiempo se le aplicó una transformación logarítmica para estabilizarla, cuyo resultado se aprecia en la Figura 5.3, donde la variabilidad de los datos a lo largo del tiempo más uniforme.

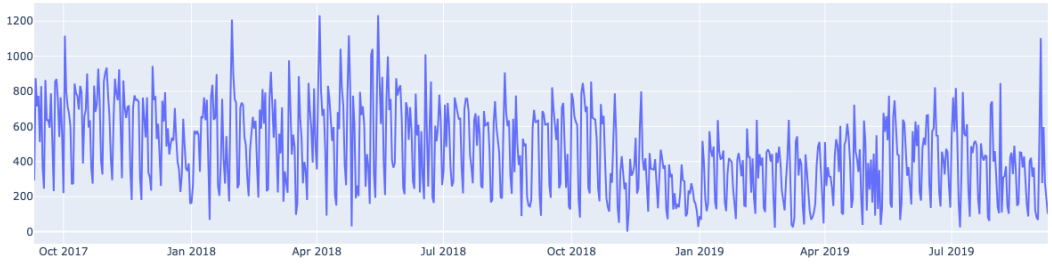

Figura 5.2: Gráfico secuencial serie temporal.

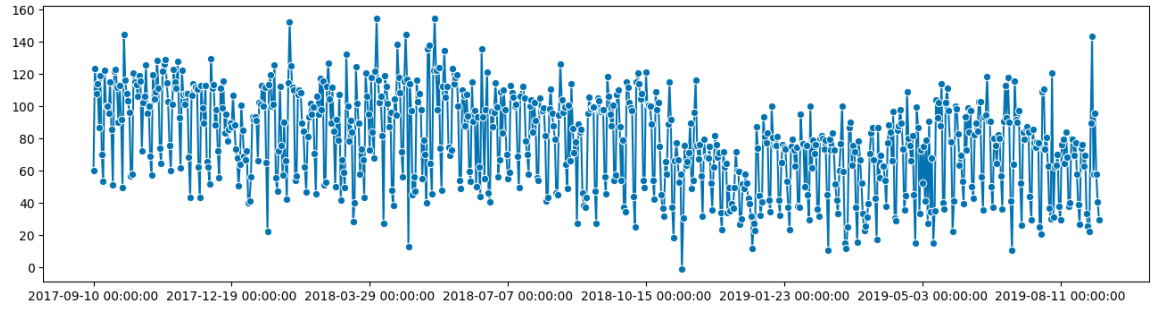

Figura 5.3: Gráfico secuencial del logaritmo de la serie temporal.

Con la serie temporal homocedástica, se llevó a cabo la prueba de Dickey-Fuller, mencionada en el Capítulo 2. Los resultados revelaron un valor p de 0.218605, indicando que no hay suficiente evidencia para rechazar la hipótesis nula de que la serie temporal tiene raíces unitarias, lo que implica que la serie no es estacionaria. Posteriormente, se procedió a diferenciar la serie una vez, obteniendo un valor p de 5.064584928230798e-19. Este resultado sugiere que la serie temporal después de la diferenciación es estacionaria.
Después, los datos se dividieron en una muestra de entrenamiento compuesta por 701 datos y una de prueba compuesta por 28 datos. Se emplearon los métodos de predicción mencionados en el Capítulo 2, evaluados con las métricas revisadas en el Capítulo 3. Cada método se aplicó de dos maneras: manualmente, ajustando los parámetros de forma manual, y automáticamente, permitiendo que la propia librería determine la mejor combinación de parámetros.

Con la librería *Sktime* se obtuvieron los resultados de las métricas para los métodos automáticos con validación cruzada que se muestran en el Cuadro 5.1, redondeados a cuatro decimales, mientras que en el Cuadro 5.2 se encuentran los resultados para los métodos manuales con validación cruzada. En ambos, el rMAPE se adecuó en un 5 % de exclusión de los errores más grandes y más pequeños. De ambos cuadros podemos observar que, en la mayoría de los casos, se obtuvieron errores más bajos para los métodos aplicados de forma manual.
Con la librería *Skforecast* se obtuvieron los resultados para los métodos automáticos con validación cruzada que se muestran en el Cuadro 5.3, redondeados a cuatro decimales, mientras que en el Cuadro 5.4 se encuentran los resultados para los métodos manuales con validación cruzada. De ambos cuadros podemos observar que, los métodos implementados de forma manual tienden a generar errores más bajos de forma general.

De forma particular, notamos que los métodos XGBoost y de NN tienden a obtener errores más bajos tanto en la implementación manual como en la automática. Aunque, ARIMA con *Skforecast* en la implementación manual tiende a tener los errores más bajos.
Dado que la serie temporal no contiene valores cercanos a cero, con una media de aproximadamente 462.32 y una desviación estándar de aproximadamente 243.96, una métrica que podría ser útil es el MAPE o sMAPE para evaluar la precisión de las predicciones en este contexto. La elección entre estas métricas dependerá de si desea considerar la simetría en la evaluación de errores (sMAPE) o si se desea una medida de error porcentual intuitiva y fácil de interpretar (MAPE).

Métricas para métodos automáticos con validación cruzada					
	MAPE	sMAPE	gMAPE	rMAPE	MASE
ARIMA	0.4170	0.3225	0.2376	0.3889	0.7845
Suavización Exponencial	0.4688	0.3425	0.2410	0.4266	0.8465
ETS	0.4688	0.3425	0.2411	0.4266	0.8465
Prophet	0.2730	0.2285	0.1813	0.2403	0.5991
XGBoost	0.2213	0.1947	0.1094	0.1992	0.5021
NN	0.1598	0.1573	0.0747	0.1403	0.4172

Cuadro 5.1: Métodos automáticos con validación cruzada con Sktime.

Métricas para métodos manuales con validación cruzada					
	MAPE	sMAPE	gMAPE	rMAPE	MASE
ARIMA	0.2708	0.2279	0.1965	0.2379	0.6232
Suavización Exponencial	0.1791	0.1653	0.0902	0.1515	0.4315
ETS	0.1776	0.1687	0.0895	0.1490	0.4392
Prophet	0.1809	0.1779	0.0988	0.1542	0.4510
XGBoost	0.1758	0.1778	0.1218	0.1522	0.4764
NN	0.1849	0.1714	0.1082	0.1614	0.4418

Cuadro 5.2: Métodos manuales con validación cruzada con Sktime.

Métricas para métodos automáticos con validación cruzada					
	MAPE	sMAPE	gMAPE	rMAPE	MASE
ARIMA	0.4264	0.3293	0.2476	0.3904	0.8031
XGBoost	0.2363	0.2270	0.1330	0.2090	0.5688
NN	0.2060	0.2072	0.1252	0.1868	0.5139

Cuadro 5.3: Métodos automáticos con validación cruzada con Skforecast.

Métricas para métodos manuales con validación cruzada					
	MAPE	sMAPE	gMAPE	rMAPE	MASE
ARIMA	0.2043	0.1925	0.1208	0.1746	0.4854
XGBoost	0.2213	0.2387	0.1260	0.2164	0.5612
NN	0.2024	0.2086	0.1412	0.1818	0.5355

Cuadro 5.4: Métodos manuales con validación cruzada con Skforecast.

Capítulo 6

Conclusiones

En el Capítulo 4, discutimos que la selección entre Sktime y Skforecast para el análisis de series temporales depende de las necesidades específicas del usuario y del contexto del problema. Ambas bibliotecas ofrecen herramientas poderosas para el análisis de series temporales en Python, aunque con enfoques ligeramente diferentes.
Sktime se destaca por su diseño modular y flexible, lo que facilita la combinación de varios métodos de series temporales y la construcción de modelos complejos. Su enfoque en la extensibilidad y la reutilización de componentes, incluyendo integraciones con bibliotecas como Statsmodels, Pmdarima, FacebookProphet y Sklearn, brinda a los usuarios un alto grado de control sobre sus análisis. Esta integración directa hace que Sktime sea una opción más completa en comparación con Skforecast, que aún carece de métodos implementados como Prophet y ETS. Como resultado, los usuarios de Skforecast pueden necesitar recurrir a anclajes manuales con otras bibliotecas para complementar sus análisis. Este aspecto motivó la investigación realizada en este trabajo, abordando las preguntas sobre la existencia de una biblioteca que permita un análisis completo de series temporales o la posibilidad de un framework que integre diversas herramientas para alcanzar este objetivo.
En este caso, podemos afirmar que Sktime ha demostrado ser una opción viable al permitir el ajuste de todos los modelos estudiados en el Capítulo 2 y de las métricas revisadas en el Capítulo 3. Sin embargo, es importante tener en cuenta que, debido a que Sktime también utiliza ciertas bibliotecas para algunos de sus métodos, es posible que no tenga implementados todos los métodos disponibles que si pueden tener otras bibliotecas y que no hemos explorado a lo largo de este trabajo. Por lo tanto, podría estar en desventaja frente a otras bibliotecas especializadas en ciertos métodos de análisis de series temporales. Con esto en mente, sería beneficioso explorar otras bibliotecas que ofrezcan una cobertura más amplia de métodos y características ajustables, o incluso crear un framework que integre la mayor cantidad posible de herramientas para el análisis de series temporales. Esto permitiría evitar anclajes y llevar a cabo proyectos completos con una sola interfaz fácil de usar y completa, que ayude a superar las limitaciones de tiempo y costo computacional de los proyectos.

Apéndice A

Tablas de modelos ETS

En esta sección se encuentran la tabla que resume las fórmulas para los cálculos recursivos y predicción puntual de todos los métodos de suavizado exponencial y las tablas de las ecuaciones de espacio de estados para los modelos según el tipo de añadidura de la componente de error.

Trend	Seasonal		
	N	A	M
N	$\ell_t = \alpha y_t + (1-\alpha)\ell_{t-1}$ $\hat{y}_{t+h\|t} = \ell_t$	$\ell_t = \alpha(y_t - s_{t-m}) + (1-\alpha)\ell_{t-1}$ $s_t = \gamma(y_t - \ell_{t-1}) + (1-\gamma)s_{t-m}$ $\hat{y}_{t+h\|t} = \ell_t + s_{t-m+h_m^+}$	$\ell_t = \alpha(y_t / s_{t-m}) + (1-\alpha)\ell_{t-1}$ $s_t = \gamma(y_t / \ell_{t-1}) + (1-\gamma)s_{t-m}$ $\hat{y}_{t+h\|t} = \ell_t s_{t-m+h_m^+}$
A	$\ell_t = \alpha y_t + (1-\alpha)(\ell_{t-1} + b_{t-1})$ $b_t = \beta^*(\ell_t - \ell_{t-1}) + (1-\beta^*)b_{t-1}$ $\hat{y}_{t+h\|t} = \ell_t + hb_t$	$\ell_t = \alpha(y_t - s_{t-m}) + (1-\alpha)(\ell_{t-1} + b_{t-1})$ $b_t = \beta^*(\ell_t - \ell_{t-1}) + (1-\beta^*)b_{t-1}$ $s_t = \gamma(y_t - \ell_{t-1} - b_{t-1}) + (1-\gamma)s_{t-m}$ $\hat{y}_{t+h\|t} = \ell_t + hb_t + s_{t-m+h_m^+}$	$\ell_t = \alpha(y_t / s_{t-m}) + (1-\alpha)(\ell_{t-1} + b_{t-1})$ $b_t = \beta^*(\ell_t - \ell_{t-1}) + (1-\beta^*)b_{t-1}$ $s_t = \gamma(y_t / (\ell_{t-1} + b_{t-1})) + (1-\gamma)s_{t-m}$ $\hat{y}_{t+h\|t} = (\ell_t + hb_t)s_{t-m+h_m^+}$
A_d	$\ell_t = \alpha y_t + (1-\alpha)(\ell_{t-1} + \phi b_{t-1})$ $b_t = \beta^*(\ell_t - \ell_{t-1}) + (1-\beta^*)\phi b_{t-1}$ $\hat{y}_{t+h\|t} = \ell_t + \phi_h b_t$	$\ell_t = \alpha(y_t - s_{t-m}) + (1-\alpha)(\ell_{t-1} + \phi b_{t-1})$ $b_t = \beta^*(\ell_t - \ell_{t-1}) + (1-\beta^*)\phi b_{t-1}$ $s_t = \gamma(y_t - \ell_{t-1} - \phi b_{t-1}) + (1-\gamma)s_{t-m}$ $\hat{y}_{t+h\|t} = \ell_t + \phi_h b_t + s_{t-m+h_m^+}$	$\ell_t = \alpha(y_t / s_{t-m}) + (1-\alpha)(\ell_{t-1} + \phi b_{t-1})$ $b_t = \beta^*(\ell_t - \ell_{t-1}) + (1-\beta^*)\phi b_{t-1}$ $s_t = \gamma(y_t / (\ell_{t-1} + \phi b_{t-1})) + (1-\gamma)s_{t-m}$ $\hat{y}_{t+h\|t} = (\ell_t + \phi_h b_t)s_{t-m+h_m^+}$
M	$\ell_t = \alpha y_t + (1-\alpha)\ell_{t-1}b_{t-1}$ $b_t = \beta^*(\ell_t / \ell_{t-1}) + (1-\beta^*)b_{t-1}$ $\hat{y}_{t+h\|t} = \ell_t b_t^h$	$\ell_t = \alpha(y_t - s_{t-m}) + (1-\alpha)\ell_{t-1}b_{t-1}$ $b_t = \beta^*(\ell_t / \ell_{t-1}) + (1-\beta^*)b_{t-1}$ $s_t = \gamma(y_t - \ell_{t-1}b_{t-1}) + (1-\gamma)s_{t-m}$ $\hat{y}_{t+h\|t} = \ell_t b_t^h + s_{t-m+h_m^+}$	$\ell_t = \alpha(y_t / s_{t-m}) + (1-\alpha)\ell_{t-1}b_{t-1}$ $b_t = \beta^*(\ell_t / \ell_{t-1}) + (1-\beta^*)b_{t-1}$ $s_t = \gamma(y_t / (\ell_{t-1}b_{t-1})) + (1-\gamma)s_{t-m}$ $\hat{y}_{t+h\|t} = \ell_t b_t^h s_{t-m+h_m^+}$
M_d	$\ell_t = \alpha y_t + (1-\alpha)\ell_{t-1}b_{t-1}^{\phi}$ $b_t = \beta^*(\ell_t / \ell_{t-1}) + (1-\beta^*)b_{t-1}^{\phi}$ $\hat{y}_{t+h\|t} = \ell_t b_t^{\phi_h}$	$\ell_t = \alpha(y_t - s_{t-m}) + (1-\alpha)\ell_{t-1}b_{t-1}^{\phi}$ $b_t = \beta^*(\ell_t / \ell_{t-1}) + (1-\beta^*)b_{t-1}^{\phi}$ $s_t = \gamma(y_t - \ell_{t-1}b_{t-1}^{\phi}) + (1-\gamma)s_{t-m}$ $\hat{y}_{t+h\|t} = \ell_t b_t^{\phi_h} + s_{t-m+h_m^+}$	$\ell_t = \alpha(y_t / s_{t-m}) + (1-\alpha)\ell_{t-1}b_{t-1}^{\phi}$ $b_t = \beta^*(\ell_t / \ell_{t-1}) + (1-\beta^*)b_{t-1}^{\phi}$ $s_t = \gamma(y_t / (\ell_{t-1}b_{t-1}^{\phi})) + (1-\gamma)s_{t-m}$ $\hat{y}_{t+h\|t} = \ell_t b_t^{\phi_h} s_{t-m+h_m^+}$

Figura A.1: Fórmulas para cálculos recursivos y pronósticos puntuales. Fuente: Hyndman et al. (2008), capítulo 2.

Trend	Seasonal		
	N	A	M
N	$\mu_t = \ell_{t-1}$ $\ell_t = \ell_{t-1} + \alpha\varepsilon_t$	$\mu_t = \ell_{t-1} + s_{t-m}$ $\ell_t = \ell_{t-1} + \alpha\varepsilon_t$ $s_t = s_{t-m} + \gamma\varepsilon_t$	$\mu_t = \ell_{t-1}s_{t-m}$ $\ell_t = \ell_{t-1} + \alpha\varepsilon_t/s_{t-m}$ $s_t = s_{t-m} + \gamma\varepsilon_t/\ell_{t-1}$
A	$\mu_t = \ell_{t-1} + b_{t-1}$ $\ell_t = \ell_{t-1} + b_{t-1} + \alpha\varepsilon_t$ $b_t = b_{t-1} + \beta\varepsilon_t$	$\mu_t = \ell_{t-1} + b_{t-1} + s_{t-m}$ $\ell_t = \ell_{t-1} + b_{t-1} + \alpha\varepsilon_t$ $b_t = b_{t-1} + \beta\varepsilon_t$ $s_t = s_{t-m} + \gamma\varepsilon_t$	$\mu_t = (\ell_{t-1} + b_{t-1})s_{t-m}$ $\ell_t = \ell_{t-1} + b_{t-1} + \alpha\varepsilon_t/s_{t-m}$ $b_t = b_{t-1} + \beta\varepsilon_t/s_{t-m}$ $s_t = s_{t-m} + \gamma\varepsilon_t/(\ell_{t-1} + b_{t-1})$
A_d	$\mu_t = \ell_{t-1} + \phi b_{t-1}$ $\ell_t = \ell_{t-1} + \phi b_{t-1} + \alpha\varepsilon_t$ $b_t = \phi b_{t-1} + \beta\varepsilon_t$	$\mu_t = \ell_{t-1} + \phi b_{t-1} + s_{t-m}$ $\ell_t = \ell_{t-1} + \phi b_{t-1} + \alpha\varepsilon_t$ $b_t = \phi b_{t-1} + \beta\varepsilon_t$ $s_t = s_{t-m} + \gamma\varepsilon_t$	$\mu_t = (\ell_{t-1} + \phi b_{t-1})s_{t-m}$ $\ell_t = \ell_{t-1} + \phi b_{t-1} + \alpha\varepsilon_t/s_{t-m}$ $b_t = \phi b_{t-1} + \beta\varepsilon_t/s_{t-m}$ $s_t = s_{t-m} + \gamma\varepsilon_t/(\ell_{t-1} + \phi b_{t-1})$
M	$\mu_t = \ell_{t-1}b_{t-1}$ $\ell_t = \ell_{t-1}b_{t-1} + \alpha\varepsilon_t$ $b_t = b_{t-1} + \beta\varepsilon_t/\ell_{t-1}$	$\mu_t = \ell_{t-1}b_{t-1} + s_{t-m}$ $\ell_t = \ell_{t-1}b_{t-1} + \alpha\varepsilon_t$ $b_t = b_{t-1} + \beta\varepsilon_t/\ell_{t-1}$ $s_t = s_{t-m} + \gamma\varepsilon_t$	$\mu_t = \ell_{t-1}b_{t-1}s_{t-m}$ $\ell_t = \ell_{t-1}b_{t-1} + \alpha\varepsilon_t/s_{t-m}$ $b_t = b_{t-1} + \beta\varepsilon_t/(s_{t-m}\ell_{t-1})$ $s_t = s_{t-m} + \gamma\varepsilon_t/(\ell_{t-1}b_{t-1})$
M_d	$\mu_t = \ell_{t-1}b_{t-1}^{\phi}$ $\ell_t = \ell_{t-1}b_{t-1}^{\phi} + \alpha\varepsilon_t$ $b_t = b_{t-1}^{\phi} + \beta\varepsilon_t/\ell_{t-1}$	$\mu_t = \ell_{t-1}b_{t-1}^{\phi} + s_{t-m}$ $\ell_t = \ell_{t-1}b_{t-1}^{\phi} + \alpha\varepsilon_t$ $b_t = b_{t-1}^{\phi} + \beta\varepsilon_t/\ell_{t-1}$ $s_t = s_{t-m} + \gamma\varepsilon_t$	$\mu_t = \ell_{t-1}b_{t-1}^{\phi}s_{t-m}$ $\ell_t = \ell_{t-1}b_{t-1}^{\phi} + \alpha\varepsilon_t/s_{t-m}$ $b_t = b_{t-1}^{\phi} + \beta\varepsilon_t/(s_{t-m}\ell_{t-1})$ $s_t = s_{t-m} + \gamma\varepsilon_t/(\ell_{t-1}b_{t-1}^{\phi})$

Figura A.2: Ecuaciones de espacio de estados para cada modelo de error aditivo en la clasificación. Fuente: Hyndman et al. (2008), capítulo 2.

Trend	Seasonal		
	N	A	M
N	$\mu_t = \ell_{t-1}$ $\ell_t = \ell_{t-1}(1 + \alpha\varepsilon_t)$	$\mu_t = \ell_{t-1} + s_{t-m}$ $\ell_t = \ell_{t-1} + \alpha(\ell_{t-1} + s_{t-m})\varepsilon_t$ $s_t = s_{t-m} + \gamma(\ell_{t-1} + s_{t-m})\varepsilon_t$	$\mu_t = \ell_{t-1}s_{t-m}$ $\ell_t = \ell_{t-1}(1 + \alpha\varepsilon_t)$ $s_t = s_{t-m}(1 + \gamma\varepsilon_t)$
A	$\mu_t = \ell_{t-1} + b_{t-1}$ $\ell_t = (\ell_{t-1} + b_{t-1})(1 + \alpha\varepsilon_t)$ $b_t = b_{t-1} + \beta(\ell_{t-1} + b_{t-1})\varepsilon_t$	$\mu_t = \ell_{t-1} + b_{t-1} + s_{t-m}$ $\ell_t = \ell_{t-1} + b_{t-1} + \alpha(\ell_{t-1} + b_{t-1} + s_{t-m})\varepsilon_t$ $b_t = b_{t-1} + \beta(\ell_{t-1} + b_{t-1} + s_{t-m})\varepsilon_t$ $s_t = s_{t-m} + \gamma(\ell_{t-1} + b_{t-1} + s_{t-m})\varepsilon_t$	$\mu_t = (\ell_{t-1} + b_{t-1})s_{t-m}$ $\ell_t = (\ell_{t-1} + b_{t-1})(1 + \alpha\varepsilon_t)$ $b_t = b_{t-1} + \beta(\ell_{t-1} + b_{t-1})\varepsilon_t$ $s_t = s_{t-m}(1 + \gamma\varepsilon_t)$
A_d	$\mu_t = \ell_{t-1} + \phi b_{t-1}$ $\ell_t = (\ell_{t-1} + \phi b_{t-1})(1 + \alpha\varepsilon_t)$ $b_t = \phi b_{t-1} + \beta(\ell_{t-1} + \phi b_{t-1})\varepsilon_t$	$\mu_t = \ell_{t-1} + \phi b_{t-1} + s_{t-m}$ $\ell_t = \ell_{t-1} + \phi b_{t-1} + \alpha(\ell_{t-1} + \phi b_{t-1} + s_{t-m})\varepsilon_t$ $b_t = \phi b_{t-1} + \beta(\ell_{t-1} + \phi b_{t-1} + s_{t-m})\varepsilon_t$ $s_t = s_{t-m} + \gamma(\ell_{t-1} + \phi b_{t-1} + s_{t-m})\varepsilon_t$	$\mu_t = (\ell_{t-1} + \phi b_{t-1})s_{t-m}$ $\ell_t = (\ell_{t-1} + \phi b_{t-1})(1 + \alpha\varepsilon_t)$ $b_t = \phi b_{t-1} + \beta(\ell_{t-1} + \phi b_{t-1})\varepsilon_t$ $s_t = s_{t-m}(1 + \gamma\varepsilon_t)$
M	$\mu_t = \ell_{t-1}b_{t-1}$ $\ell_t = \ell_{t-1}b_{t-1}(1 + \alpha\varepsilon_t)$ $b_t = b_{t-1}(1 + \beta\varepsilon_t)$	$\mu_t = \ell_{t-1}b_{t-1} + s_{t-m}$ $\ell_t = \ell_{t-1}b_{t-1} + \alpha(\ell_{t-1}b_{t-1} + s_{t-m})\varepsilon_t$ $b_t = b_{t-1} + \beta(\ell_{t-1}b_{t-1} + s_{t-m})\varepsilon_t/\ell_{t-1}$ $s_t = s_{t-m} + \gamma(\ell_{t-1}b_{t-1} + s_{t-m})\varepsilon_t$	$\mu_t = \ell_{t-1}b_{t-1}s_{t-m}$ $\ell_t = \ell_{t-1}b_{t-1}(1 + \alpha\varepsilon_t)$ $b_t = b_{t-1}(1 + \beta\varepsilon_t)$ $s_t = s_{t-m}(1 + \gamma\varepsilon_t)$
M_d	$\mu_t = \ell_{t-1}b_{t-1}^{\phi}$ $\ell_t = \ell_{t-1}b_{t-1}^{\phi}(1 + \alpha\varepsilon_t)$ $b_t = b_{t-1}^{\phi}(1 + \beta\varepsilon_t)$	$\mu_t = \ell_{t-1}b_{t-1}^{\phi} + s_{t-m}$ $\ell_t = \ell_{t-1}b_{t-1}^{\phi} + \alpha(\ell_{t-1}b_{t-1}^{\phi} + s_{t-m})\varepsilon_t$ $b_t = b_{t-1}^{\phi} + \beta(\ell_{t-1}b_{t-1}^{\phi} + s_{t-m})\varepsilon_t/\ell_{t-1}$ $s_t = s_{t-m} + \gamma(\ell_{t-1}b_{t-1}^{\phi} + s_{t-m})\varepsilon_t$	$\mu_t = \ell_{t-1}b_{t-1}^{\phi}s_{t-m}$ $\ell_t = \ell_{t-1}b_{t-1}^{\phi}(1 + \alpha\varepsilon_t)$ $b_t = b_{t-1}^{\phi}(1 + \beta\varepsilon_t)$ $s_t = s_{t-m}(1 + \gamma\varepsilon_t)$

Figura A.3: Ecuaciones de espacio de estados para cada modelo de error multiplicativo en la clasificación. Fuente: Hyndman et al. (2008), capítulo 2.

Bibliografía

Amat, Joaquín. (2020). *Gradient Boosting con Python*. Disponible en Web: https://cienciadedatos.net/documentos/py09_gradient_boosting_python.

Amat, Joaquín., y Escobar, Javier. (2023). *Skforecast: forecasting series temporales con Python y Scikitlearn*. Disponible en Web: https://cienciadedatos.net/documentos/py27-forecasting-series-temporales-python-scikitlearn.

Amat, Joaquín., y Escobar, Javier. (2024). Skforecast (Version 0.12.1) [Software informático]. Disponible en Web: https://skforecast.org/0.12.1/index.html.

Aneiros, Germán. (2008). Series de tiempo. Apuntes de la materia. Disponible en Web:http://eio.usc.es/eipc1/BASE/BASEMASTER/FORMULARIOS-PHP/MATERIALESMASTER/Tema1.pdf.

Aneiros, Germán. (2008). Series de tiempo. Apuntes de la materia. Disponible en Web: http://eio.usc.es/eipc1/BASE/BASEMASTER/FORMULARIOS-PHP/MATERIALESMASTER/Mat_11_Tema2.pdf.

Aneiros, Germán. (2008). Series de tiempo. Apuntes de la materia. Disponible en Web: http://eio.usc.es/eipc1/BASE/BASEMASTER/FORMULARIOS-PHP/MATERIALESMASTER/Mat_112852_Tema3_I.pdf.

Brockwell, Peter., y Davis, Richard. (1991). *Time series: theory and methods*. Springer science& business media.

Géron, Aurélien. (2022). *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow*. O'Reilly Media, Inc.

Hyndman, Rob., Koehler, Anne., Ord, Keith., y Snyder, Ralph. (2005). Prediction intervals for exponential smoothing using two new classes of state space models. *Journal of Forecasting*, 24(1):17-37.

Hyndman, Rob., y Koehler, Anne. (2006). Another look at measures of forecast accuracy. *International journal of forecasting*, 22(4):679-688.

Hyndman, Rob., Koehler, A.nne., Ord, Keith., y Snyder, Ralph. (2008). *Forecasting with exponential smoothing: the state space approach*. Springer Science & Business Media.

Hyndman, Rob., y Athanasopoulos, George. (2018). *Forecasting: principles and practice*, 2nd edition, OTexts: Melbourne, Australia. Disponible en web: https://otexts.com/fpp2/.

Kim, Sungil., y Kim, H.eeyoung. (2016). A new metric of absolute percentage error for intermittent demand forecasts. *International Journal of Forecasting*, 32(3), 669-679.

Löning, Markus., Bagnall, Anthony., Ganesh, Sajaysurya., Kazakov, Viktor., Lines, Jason., y Király, Franz. (2019). *sktime: A unified interface for machine learning with time series*.

Peixeiro, Marco. (2022). *Time series forecasting in python*. Simon and Schuster.

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

Taylor, James. (2003). Exponential smoothing with a damped multiplicative trend. *International journal of Forecasting*, 19(4):715- 725.

Taylor, Sean., y Letham, Benjamin. (2018). Forecasting at scale. *The American Statistician*, 72(1), 37-45.