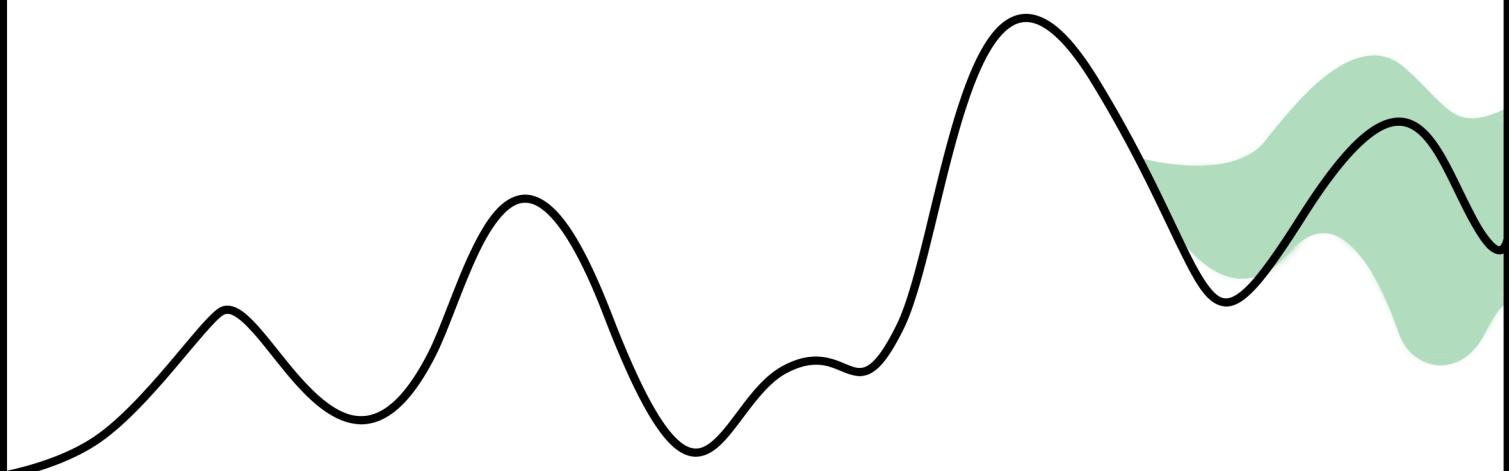


Comparación del desempeño de modelos estadísticos tradicionales, de aprendizaje automático y aprendizaje profundo para la predicción de series temporales

Tesina de grado



Alumno: Roncaglia Andrés Iván

Directora: Mag. Méndez Fernanda

Carrera: Licenciatura en Estadística



UNR Universidad
Nacional de Rosario

Agradecimientos

Resumen

Palabras clave: series temporales, predicción, ARIMA, aprendizaje automático, redes neuronales, transformers, TimeGPT.

Índice

| | |
|---|-----------|
| 1. Introducción | 1 |
| 2. Objetivos | 2 |
| 2.1 Objetivo general | 2 |
| 2.2 Objetivos específicos | 2 |
| 3. Metodología | 3 |
| 3.1 Conceptos básicos de series de tiempo | 3 |
| 3.2 Modelos estadísticos tradicionales | 3 |
| 3.2.1 SARIMA | 3 |
| 3.3 Algoritmos de aprendizaje automático | 5 |
| 3.3.1 Introducción a árboles de decisión y ensamblado | 5 |
| 3.3.2 Diferencias entre XGBoost y LightGBM | 7 |
| 3.3.3 Intervalos de confianza en algoritmos de aprendizaje automático | 8 |
| 3.4 Modelos de aprendizaje profundo | 8 |
| 3.4.1 Introducción a redes neuronales | 8 |
| 3.4.2 <i>Long Short Term Memory (LSTM)</i> | 10 |
| 3.4.3 Modelos transformadores | 12 |
| 3.4.4 Diferencias entre TimeGPT y Chronos | 16 |
| 3.5 Métricas de evaluación | 17 |
| 3.6 Técnicas para la estimación de parámetros | 18 |
| 4. Aplicación | 19 |
| 4.1 Series a utilizar | 19 |
| 4.2 Modelado | 21 |
| 4.2.1 ARIMA | 21 |
| 4.2.2 Modelos de aprendizaje automático | 27 |
| 4.2.3 LSTM | 29 |
| 4.2.4 Modelos fundacionales | 30 |
| 4.3 Comparaciones | 31 |
| 5. Conclusiones | 33 |
| 6. Bibliografía | 34 |
| 7. Anexo | 36 |
| 7.1 Gráficos estacionales | 36 |
| 7.2 Salidas de modelos arima | 36 |
| 7.3 Comprobación de supuestos de modelos arima | 42 |
| NOTAS | 44 |
| Mejoras a la investigación (en las que no me voy a centrar) | 45 |
| Preguntas nanda | 45 |

1. Introducción

La predicción de valores futuros en series de tiempo es una herramienta clave en múltiples ámbitos, tales como la economía, el comercio, la salud, la energía y el medio ambiente. En estos contextos, anticipar el comportamiento de una variable permite mejorar la planificación, asignar recursos de forma más eficiente y reducir la incertidumbre.

Actualmente, la ciencia de datos se encuentra en una etapa de constante expansión e innovación, impulsada por la gran cantidad de datos generados diariamente, por lo que en un contexto creciente de complejidad y exigencia temporal, resulta conveniente contar con herramientas que faciliten y acorten los tiempos de trabajo. Si bien los métodos más conocidos para trabajar series de tiempo son precisos, los modelos tradicionales como ARIMA son difíciles de automatizar y requieren de amplios conocimientos para encontrar un buen ajuste, mientras que los algoritmos de aprendizaje automatizado que se utilizan actualmente pueden demandar largos tiempos de entrenamiento y un gran coste computacional. Frente a estas limitaciones, han surgido recientemente modelos capaces de seleccionar de forma automática el mejor ajuste para una serie temporal dada, sin requerir entrenamiento previo ni conocimientos especializados en análisis de series de tiempo. Estos son los denominados modelos fundacionales preentrenados, tales como TimeGPT o Chronos.

Sin embargo, aún persisten interrogantes sobre el desempeño de estos nuevos modelos y la falta de acceso al código fuente de algunos de estos limita la posibilidad de auditar sus resultados o replicar su implementación. Por ello, esta tesina propone realizar una comparación sistemática de modelos de pronóstico para series de tiempo, abordando tres enfoques metodológicos: modelos estadísticos tradicionales, algoritmos de *machine learning* y modelos de aprendizaje profundo. El objetivo es evaluar su desempeño en distintos contextos, utilizando métricas como el porcentaje del error absoluto medio (MAPE) y el *Interval Score*, con el fin de analizar ventajas, limitaciones y potenciales usos de cada uno.

Este análisis busca aportar una mirada crítica e informada sobre el uso de nuevas tecnologías en la predicción de series de tiempo, contribuyendo a la toma de decisiones metodológicas más sólidas desde una perspectiva estadística.

2. Objetivos

2.1 Objetivo general

El objetivo de esta tesina es, en primer lugar, comparar la precisión, eficiencia y facilidad de pronosticar series de tiempo con distintos modelos, incluyendo enfoques estadísticos clásicos, algoritmos de *machine learning* y modelos de *deep learning*, analizando al mismo tiempo sus ventajas, limitaciones y condiciones de uso más apropiadas.

2.2 Objetivos específicos

- Implementar modelos clásicos de series de tiempo, como ARIMA y SARIMA, explicando y garantizando el cumplimiento de los fundamentos teóricos y supuestos que los sostienen.
- Aplicar modelos de aprendizaje automático supervisado, como XGBoost y LightGBM, explorando distintas configuraciones para garantizar el mejor ajuste.
- Desarrollar modelos de aprendizaje profundo, en particular redes LSTM, dando introducción a las redes neuronales y modelos de pronóstico más complejos.
- Realizar pronósticos con modelos fundacionales(TimeGPT,Chronos) y comprender su funcionamiento.
- Definir y aplicar métricas de evaluación (MAPE, *Interval Score*) para comparar el rendimiento de todos los modelos bajo un mismo conjunto de datos.
- Reflexionar valorativamente sobre los criterios de selección de modelos en función del contexto de aplicación, la complejidad computacional y la interpretabilidad de los resultados.

3. Metodología

El enfoque metodológico adoptado en esta tesina consiste en comparar el desempeño de distintos modelos de pronóstico aplicados a series temporales. Para ello, se seleccionan modelos representativos de tres enfoques principales: modelos estadísticos tradicionales, algoritmos de aprendizaje automático (*machine learning*) y modelos de aprendizaje profundo (*deep learning*).

El análisis se estructura en tres componentes fundamentales: una descripción conceptual de los modelos, su implementación práctica sobre series con diferentes características, y una evaluación cuantitativa comparativa a través de métricas de error.

3.1 Conceptos básicos de series de tiempo

Se denomina serie de tiempo a un conjunto de observaciones $\{z_1, z_2, \dots, z_t, \dots, z_n\}$ cuantitativas ordenadas en el tiempo, usualmente de forma equidistante, sobre una variable de interés. El análisis de series de tiempo tiene como objetivo sintetizar y extraer información estadística relevante, tanto para interpretar el comportamiento histórico de la variable como para generar pronósticos $\{z_{n+1}, \dots, z_{n+l}, \dots, z_{n+h}\}$.

Dado que las series temporales pueden exhibir diversos patrones subyacentes, resulta útil descomponerlas en componentes separadas, cada una de las cuales representa una característica estructural específica del comportamiento de la serie.

- Estacionalidad: corresponde a las fluctuaciones periódicas que se repiten a intervalos regulares de tiempo. Un ejemplo típico es la temperatura, que tiende a disminuir en invierno y aumentar en verano, repitiendo este patrón anualmente.
- Tendencia (o tendencia-ciclo): refleja la evolución a largo plazo de la media de la serie, asociada a procesos de crecimiento o decrecimiento sostenido. Por ejemplo, la población mundial exhibe una tendencia creciente a lo largo del tiempo.
- Residuos: representa las variaciones no sistemáticas que no pueden ser explicadas por la tendencia ni la estacionalidad. Estas fluctuaciones, que suelen deberse a eventos impredecibles o factores exógenos, se asumen como aleatorias.

3.2 Modelos estadísticos tradicionales

Son llamados modelos tradicionales a aquellos que surgen antes del auge del *machine learning* y los modelos de aprendizaje profundo. Son caracterizados por sus fuertes fundamentos estadísticos y su capacidad en capturar dependencias temporales en los datos.

3.2.1 SARIMA

Se dice que una serie es débilmente estacionaria si la media y la variancia se mantienen constantes en el tiempo y la correlación entre distintas observaciones solo depende de la distancia en el tiempo entre estas. Por comodidad, cuando se mencione estacionariedad se estará haciendo referencia al cumplimiento de estas propiedades.

Se denomina función de autocorrelación a la función de los rezagos, entendiendo por rezago a la distancia ordinal entre dos observaciones, que grafica la autocorrelación entre pares de observaciones. Es decir que para cada valor k se tiene la correlación entre todos los pares de

observaciones a k observaciones de distancia. En su lugar, la función de autocorrelación parcial calcula la correlación condicional de los pares de observaciones, removiendo la dependencia lineal de estas observaciones con las que se encuentran entre estas.

Los modelos *ARIMA* (*AutoRegressive Integrated Moving Average*) son unos de los modelos de pronóstico tradicionales mejor establecidos. Son una generalización de los modelos autoregresivos (AR), que suponen que las observaciones futuras son función de las observaciones pasadas, y los modelos promedio móvil (MA), que pronostican las observaciones como funciones de los errores de observaciones pasadas. Además, estos modelos pueden adaptarse a series no estacionarias mediante la aplicación de diferenciaciones de orden d , las cuales implican restar a cada observación el valor registrado d períodos anteriores.

Formalmente un modelo $ARIMA(p, d, q)$ se define como:

$$\psi_p(B)(1 - B)^d z_t = \theta_0 + \theta_q(B)\alpha_t \quad (1)$$

Donde z_t es la observación t -ésima, $\psi_p(B)$ y $\theta_q(B)$ son funciones de los rezagos (B), correspondientes a la parte autoregresiva y promedio móvil respectivamente, d es el grado de diferenciación y α_t es el error de la t -ésima observación.

Se debe tener en cuenta estos aspectos importantes:

- Se dice que una serie es invertible si se puede escribir cada observación como una función de las observaciones pasadas más un error aleatorio. Por definición, todo modelo AR es invertible.
- Por definición, todo modelo MA es estacionario.
- $\psi_p(B) = 1 - \psi_1 B - \psi_2 B^2 - \dots - \psi_p B^p$ es el polinomio característico de la componente AR y $\theta_q(B) = 1 - \theta_1 B - \theta_2 B^2 - \dots - \theta_q B^q$ de la componente MA. Si las raíces de los polinomios característicos caen fuera del círculo unitario, entonces un proceso AR se puede escribir de forma MA y es estacionario, y a su vez un proceso MA se puede escribir de forma AR y es invertible.
- Un proceso *ARIMA* es estacionario e invertible si su componente AR y MA lo son respectivamente.

Sin embargo este tipo de modelos no tienen en cuenta la posible estacionalidad que puede tener una serie, es por esto que se introducen los modelos $SARIMA(p, d, q)(P, D, Q)_s$ que agregan componentes AR, MA y diferenciaciones a la parte estacional de la serie con período s .

Un buen modelo *SARIMA* debe cumplir las siguientes propiedades:

- Sus residuos se comportan como ruido blanco, es decir, están incorrelacionados y siguen una distribución normal, con media y variancia constantes.
- Es admisible, es decir que es invertible y estacionario.
- Es parsimonioso, en el sentido de que sus parámetros son significativos.
- Es estable en los parámetros, que se cumple cuando las correlaciones entre los parámetros no son altas.

3.3 Algoritmos de aprendizaje automático

El aprendizaje automático (machine learning) es una rama de la inteligencia artificial que permite a las computadoras aprender de los datos y realizar tareas de forma autónoma. Aunque los métodos presentados no fueron diseñados específicamente para datos temporales, han demostrado ser útiles en múltiples contextos mediante diversas pruebas empíricas.

Los métodos de *machine learning*, a diferencia de los modelos tradicionales, se enfocan principalmente en identificar los patrones que describen el comportamiento del proceso que sean relevantes para pronosticar la variable de interés, y no se componen de reglas ni supuestos que tengan que seguir. Para la identificación de patrones, estos modelos requieren la generación de características.

3.3.1 Introducción a árboles de decisión y ensamblado

Los árboles de decisión pueden ser explicados sencillamente como un conjunto extenso de estructuras condicionales *if-else*. El modelo pronosticará un cierto valor x si una cierta condición es verdadera, u otro valor y si es falsa. Es importante ver que no hay una tendencia lineal en este tipo de lógica, por lo que los árboles de decisión pueden ajustar tendencias no lineales. El resultado que se obtiene al aplicar esta técnica puede resumirse gráficamente como un camino que toma diferentes bifurcaciones (o un tronco con diferentes ramas), y de esta característica surge su nombre.

Un árbol puede tener distinta cantidad de divisiones en un mismo nivel, llamadas hojas, y profundidad, las cuales determinan en qué medida el modelo se ajusta a los datos con los que se entrena. Lógicamente árboles más profundos y con más hojas suelen generar sobreajuste, es decir, un modelo que se adapta demasiado a los datos de entrenamiento y generaliza mal.

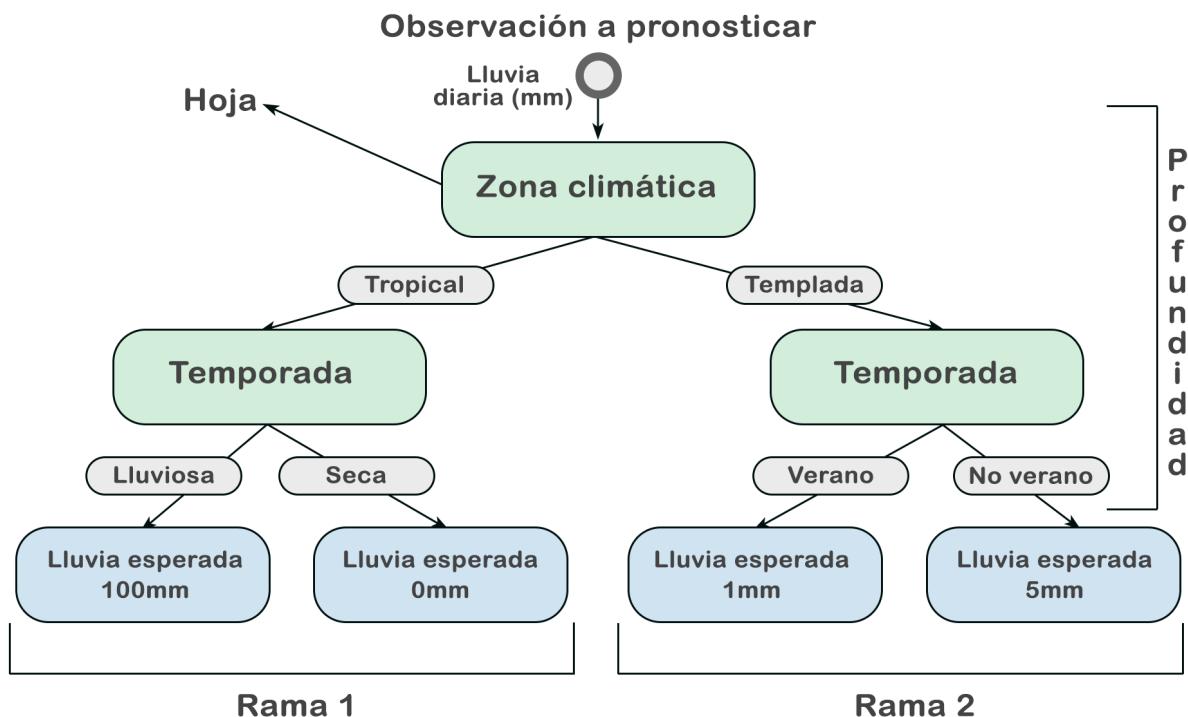


Figura 1: Ejemplo de árbol de decisión

Los métodos de ensamblaje combinan la predicción de varios estimadores base con el objetivo de mejorar la robustez de la predicción. Existen numerosos métodos de ensamblaje entre los cuales se encuentran los bosques de decisión y los árboles potenciados por gradiente (del inglés *Gradient-boosted trees*).

Boosting es un proceso iterativo, que consiste en la construcción de árboles de forma secuencial donde cada nuevo árbol busca predecir los residuos de los árboles anteriores. Es así entonces que el primer árbol buscará predecir los valores futuros de la serie, mientras que el segundo intentará predecir los valores reales menos los pronosticados por el primer árbol, el tercero tratará de inferir la diferencia entre los valores reales y el valor pronosticado del primer árbol menos los errores del segundo, y así sucesivamente. En cada iteración se pesan los puntos y se corrigen aquellos que tengan un mayor error por medio del descenso del gradiente.

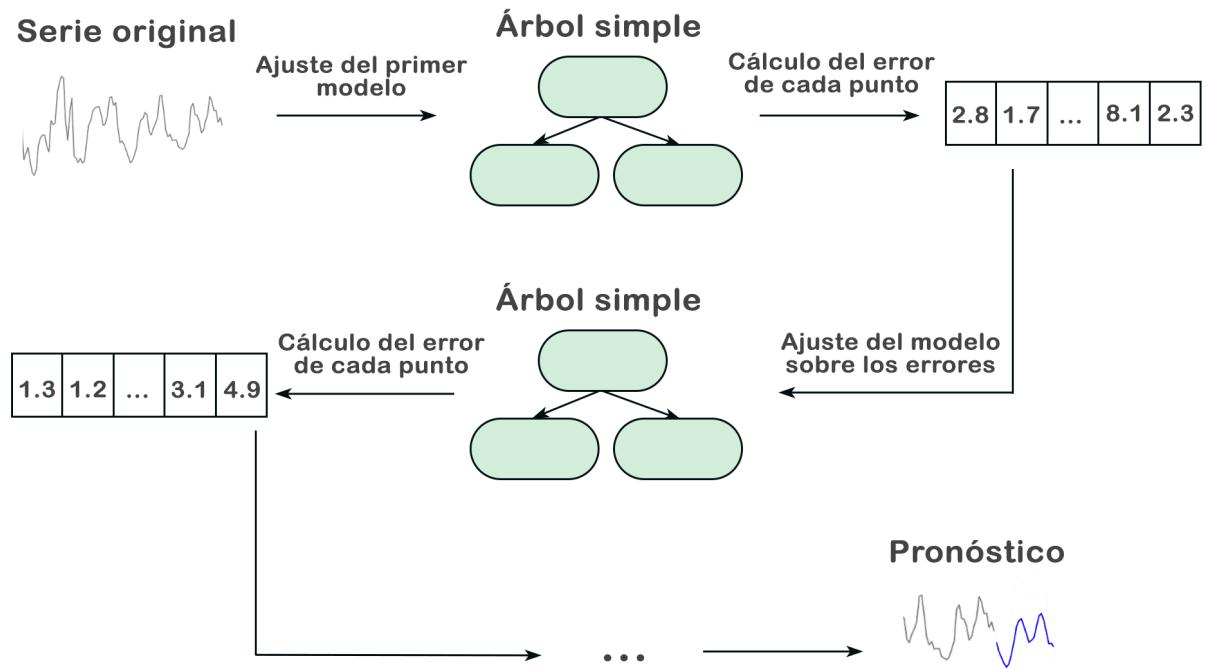


Figura 2: Proceso de ensamblado

La dirección de máximo crecimiento para una función está determinada por su gradiente, y del mismo modo, la dirección contraria a este gradiente es la dirección de máximo decrecimiento. De este modo, el descenso del gradiente busca encontrar los valores más bajos para una función de pérdida. El algoritmo de descenso de gradiente propone calcular el gradiente de la función de costo bajo el valor actual de parámetros, para luego modificarlo moviéndose en la dirección de mayor descenso. Este resultado se basa en derivadas, tasas de cambio instantáneo, por lo que conviene desplazarse una magnitud pequeña η , llamada “tasa de aprendizaje”. Es un algoritmo iterativo, en el que en cada paso la regla de actualización consiste en calcular la derivada de la función de costo respecto a cada parámetro y desplazarse una cierta magnitud η en la dirección contraria. Esto se repite un cierto número de veces hasta alcanzar la convergencia.

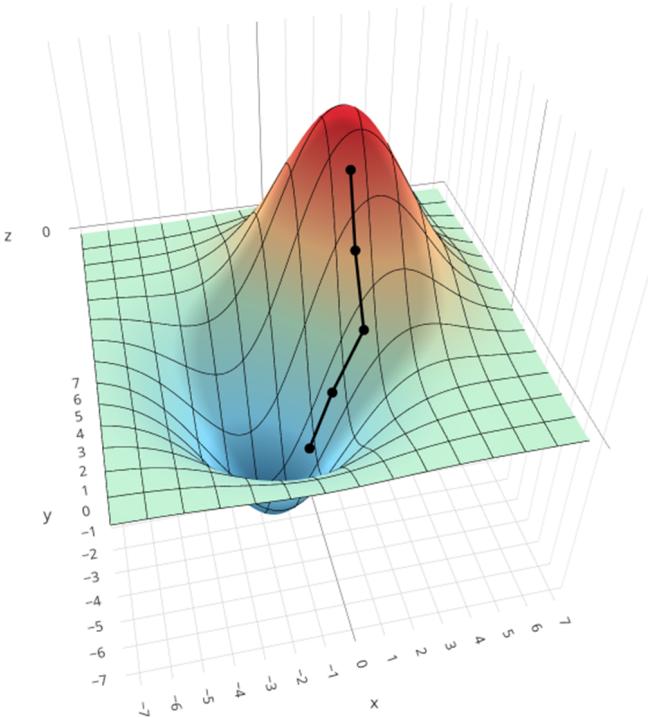


Figura 3: Ejemplo del descenso del gradiente en una función de pérdida

Sin embargo, los modelos no se construyen infinitamente, sino que se busca minimizar una función de pérdida que incluye una penalización por la complejidad del modelo, limitando así la cantidad de árboles que se producen. Existen múltiples métodos de ensamblaje (XGBoost, LightGBM, CatBoost, AdaBoost, entre otros) que se diferencian en la forma en la que se construyen los árboles. En esta tesina se usarán los algoritmos *eXtreme Gradient Boosting* (XGBoost) y *Light Gradient-Boosting Machine* (LightGBM).

3.3.2 Diferencias entre XGBoost y LightGBM

Las diferencias entre XGBoost y LightGBM radican en la forma en la que cada uno identifica las mejores divisiones dentro de los árboles y de que forma los hacen crecer.

Mientras que XGBoost usa un método en el que se construyen histogramas para cada una de las características generadas para elegir la mejor división por característica, LightGBM usa un método más eficiente llamado *Gradient-Based One-Side Sample* (GOSS). GOSS calcula los gradientes para cada punto y lo usa para filtrar afuera aquellos puntos que tengan un bajo gradiente, ya que esto significaría que estos están mejor pronosticados que el resto y no es necesario enfocarse tanto en ellos. Además, LightGBM utiliza un procedimiento que acelera el ajuste cuando se tienen muchas características correlacionadas de las cuales elegir.

A la hora de hacer crecer los árboles, XGBoost lo hace nivel a nivel, es decir que primero se crean todas las divisiones de un nivel, y luego se pasa al siguiente, priorizando que el árbol sea simétrico y tenga la misma profundidad en todas sus ramas. LightGBM, en cambio, se expande a partir de la hoja que más reduce el error, mejorando la precisión y eficiencia en series largas, pero arriesgándose a posibles sobreajustes si no se limita correctamente la profundidad de los árboles.

3.3.3 Intervalos de confianza en algoritmos de aprendizaje automático

No depender del cumplimiento de los supuestos distribucionales de los errores es una gran ventaja, pero conlleva la consecuencia de no poder obtener pronósticos probabilísticos. Para obtener intervalos de confianza que acompañen los pronósticos puntuales existen varias alternativas.

La regresión cuantil con *boosting* consiste en forzar al modelo a pronosticar los cuantiles deseados en lugar de la media. Sin embargo, esta opción no es fácilmente aplicable a XGBoost.

Una alternativa que recientemente gana popularidad es *conformal predictions*, que es una familia de métodos no paramétricos para construir intervalos de predicción. En general, esta familia de métodos requieren el cumplimiento del supuesto de intercambiabilidad, es decir, que las observaciones no deben ser dependientes del tiempo, algo que no se cumple en series de tiempo. Sin embargo, *Ensemble Batch Prediction Intervals* (EnmPI) es un método de conformal prediction diseñado específicamente para series de tiempo que no requiere este supuesto.

Para aplicar EnmPI primero se debe elegir un estimador por ensamblado y formar B muestras con *Bootstrap* por bloques. Esto último consiste en dividir el conjunto de datos de entrenamiento en bloques y formar un nuevo conjunto remuestreando sobre estos mismos bloques para mantener la correlación de los datos. Se ajustan B modelos, uno para cada muestra bootstrap. Para cada punto en el conjunto de entrenamiento se calcula el residuo usando únicamente aquellos estimadores que no contenían al punto en cuestión. Se generan los pronósticos puntuales usando la media de los pronósticos de los B modelos. Los intervalos de confianza se consiguen sumando a los pronósticos puntuales los cuantiles de la distribución de los residuos.

$$IC_{Z_{n+i};1-\alpha} = Z_n(l) \pm Q_{1-\alpha}(e) \quad (2)$$

Este último método será el que se utilizará para calcular los intervalos de confianza en los algoritmos de aprendizaje automático.

3.4 Modelos de aprendizaje profundo

El *deep learning* (aprendizaje profundo) es una rama del *machine learning* que tiene como base un conjunto de algoritmos que intentan modelar niveles altos de abstracción en los datos usando múltiples capas de procesamiento, con complejas estructuras o compuestas de varias transformaciones no lineales.

Entre estos algoritmos se encuentran las redes neuronales, que imitan el funcionamiento del cerebro humano usando procesos que simulan la forma biológica en la que trabajan las neuronas para identificar fenómenos, evaluar opciones y llegar a conclusiones.

3.4.1 Introducción a redes neuronales

Una red neuronal está compuesta en grandes rasgos de 3 capas: entrada, oculta y salida. Dentro de cada capa se pueden encontrar neuronas y conexiones entre estas, donde cada neurona representa una variable y cada conexión un peso, y es por esto que a estas conexiones las llamaremos así en adelante. La suma de los pesos y las neuronas que no formen parte de la capa de entrada dan el total de parámetros que tiene que ajustar el modelo.

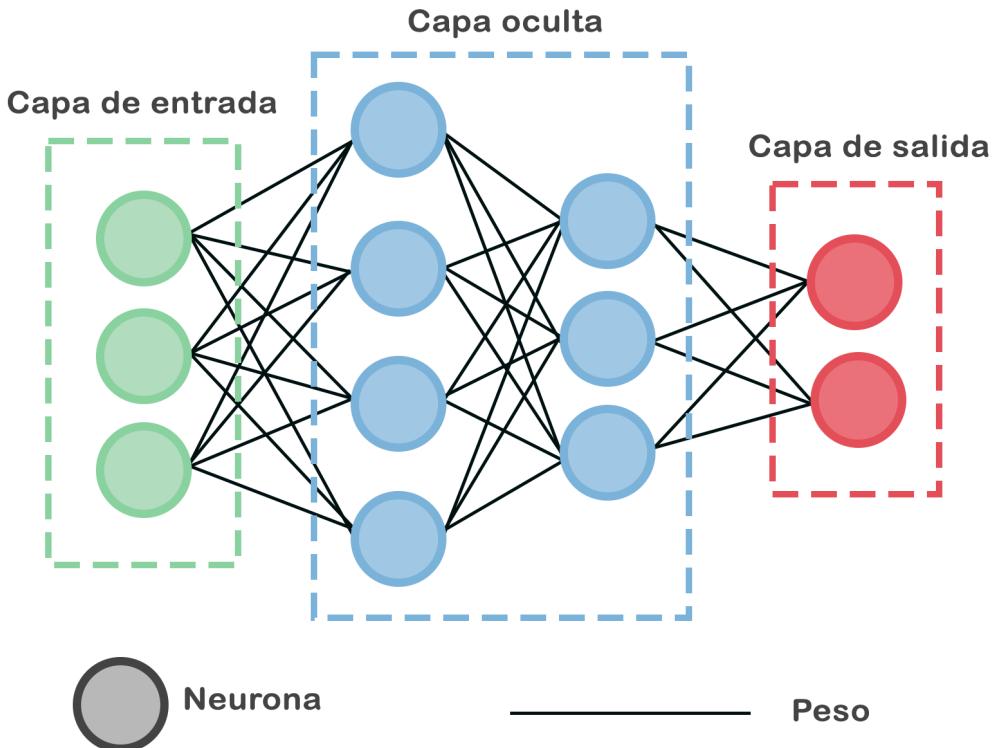


Figura 4: Red neuronal completamente conectada

En la capa de entrada se introducen las variables explicativas, y luego cada neurona fuera de esta capa es una función de las neuronas anteriores conectadas a la misma. Estas funciones son llamadas funciones de activación.

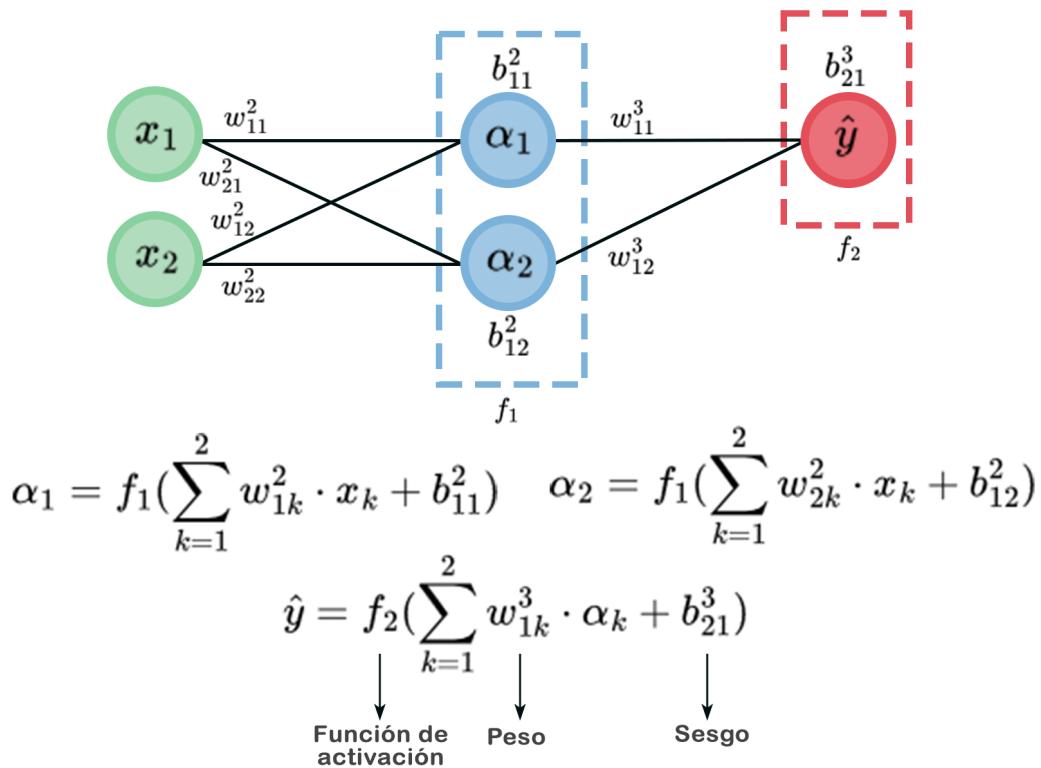


Figura 5: Red neuronal completamente conectada

Los parámetros se estiman buscando minimizar una función de costo, esto se logra con el descenso del gradiente y por medio de retropropagación. La retropropagación consiste en realizar una estimación inicial de la variable respuesta con los valores iniciales de la red neuronal, que pueden estar dados por ejemplo por una distribución normal, y de manera inversa a la dirección de la red neuronal calcular derivadas para encontrar la dirección de máximo decrecimiento de la función de costo para cada parámetro en la red neuronal.

Existen distintos tipos de redes neuronales según la forma en la que se conectan las neuronas. En esta tesina son de interés especialmente las *Convolutional Neural Networks* (CNN) y las *Recurrent Neural Networks* (RNN), en español, redes neuronales convolucionales y recurrentes respectivamente. Las primeras son útiles para el reconocimiento de patrones en los datos, mientras que las últimas son especialmente buenas en la predicción de datos secuenciales.

3.4.2 Long Short Term Memory (LSTM)

Lo que caracterizan a las redes neuronales recurrentes son los bucles de retroalimentación que se presentan en la Figura 6. Mientras que cada neurona de entrada en una red neuronal completamente conectada es independiente, en las redes neuronales recurrentes se relacionan entre ellas y se retroalimentan.

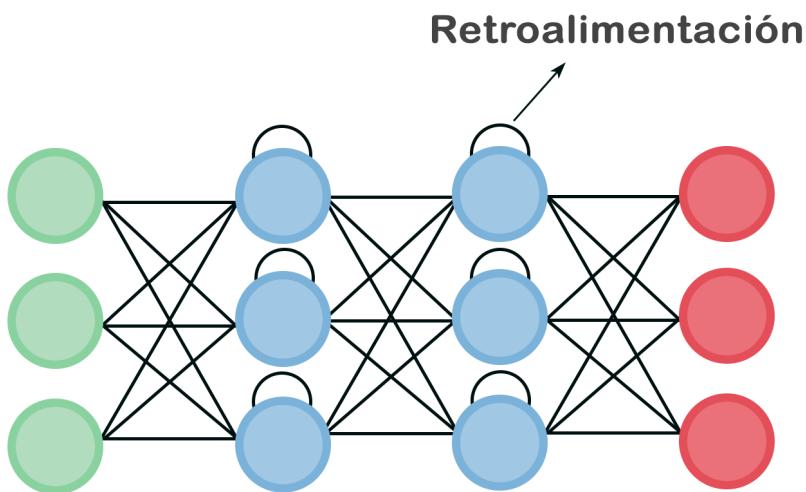


Figura 6: Ejemplo de RNN

Un problema frecuente en las RNN es su dificultad para capturar dependencias de largo plazo. Esto puede tener 2 causas, el desvanecimiento o la explosión del gradiente. El desvanecimiento del gradiente ocurre cuando, iteración tras iteración, el gradiente se aproxima a cero y se estabiliza, evitando que la red siga aprendiendo. Por el contrario, cuando el gradiente crece exponencialmente se habla de una explosión, esto lleva a inestabilidades en el aprendizaje, haciendo que las actualizaciones de los parámetros sean erráticas e impredecibles.

Las redes neuronales con memoria a corto y largo plazo (LSTM) son un tipo de RNN que solucionan este problema mediante un algoritmo logístico de 3 puertas, y usando una neurona que guarda la información histórica necesaria para la red.

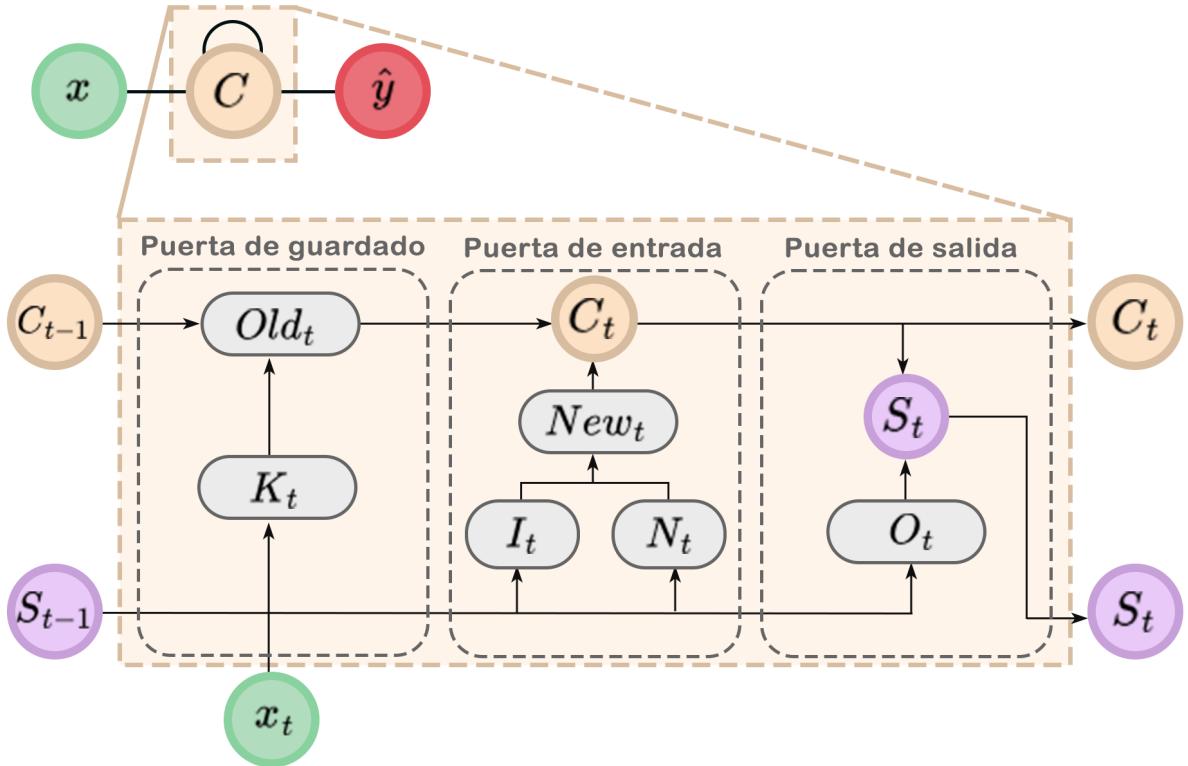


Figura 7: Estructura *Long Short Term Memory*

Puerta de guardado

La puerta de guardado se encarga de decidir si mantener o descartar la información actualmente guardada en la neurona de memoria de la red neuronal. Esta puerta recibe la entrada y el estado de la RNN, y las pasa como argumentos de una función sigmoide.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad (3)$$

$$K_t = \sigma(W_k \times [S_{t-1}, x_t] + B_k) \quad (4)$$

Si K_t es igual a 1, significa que la información guardada debe ser mantenida perfectamente. Si K_t fuera igual a 0, la información guardada debe ser descartada completamente.

Sean S_{t-1} el estado actual de la RNN, x_t la entrada actual, y W_t y B_t los pesos y sesgos de la puerta de guardado respectivamente:

$$Old_t = K_t \times C_{t-1} \quad (5)$$

Donde C_{t-1} es la información guardada actualmente y Old_t lo que se mantendrá para la próxima iteración de la red.

Puerta de entrada

La puerta de entrada controla que información añadir a la neurona de memoria. Sean W_i y B_i los pesos y sesgos de la puerta de guardado:

$$I_t = \sigma(W_i \times [S_{t-1}, x_t] + B_i) \quad (6)$$

$$New_t = I_t \times N_t \quad (7)$$

Donde N_t es el nuevo valor propuesto por la red neuronal y New_t es la información que se va a agregar a la neurona de memoria. Luego, el nuevo valor de la neurona de memoria es:

$$C_t = Old_t + New_t$$

Puerta de salida

La puerta de salida se encarga de extraer la información más importante del estado actual de la neurona para usar como salida. Sean W_o y B_o los pesos y sesgos de la puerta de salida y $tanh(x)$ la función tangente:

$$O_t = \sigma(W_o \times [S_{t-1}, x_t] + B_o) \quad (8)$$

$$S_t = O_t \times \tanh(C_t) \quad (9)$$

Donde S_t es el nuevo estado de la red neuronal.

Las 3 puertas son lógicas para que sea sencillo aplicar la retropropagación. Este sistema de puertas evita los problemas de desvanecimiento y explotación del gradiente, y evita que se acumulen muchos estados por largos períodos de tiempo eligiendo qué información es relevante guardar.

3.4.3 Modelos transformadores

Otro tipo de modelo de aprendizaje profundo son los *transformer models* (modelos transformadores), los cuales son significativamente más eficientes al entrenar y realizar inferencias que las RNNs; gracias al uso de mecanismos de atención, presentados en la publicación '[Attention is all you need](#)' de Google. La autoatención captura dependencias y relaciones en la secuencias de valores que se alimentan al modelo, logrando poner en contexto a cada observación.

Los modelos transformadores fueron creados originalmente con el propósito de generar texto. Sin embargo, tanto TimeGPT como Chronos explotan esta tecnología para el pronóstico de series de tiempo. Ambos modelos son preentrenados, lo cual significa que la optimización de parámetros y pesos fue realizada antes de usarse el modelo. Esto se logra entrenando y generalizando el modelo en un conjunto de datos extenso, por lo general de fuentes públicas. El preentrenamiento permite que el modelo adquiera conocimientos generales sobre la estructura y los patrones de los datos, los cuales luego pueden ser reutilizados en tareas concretas mediante técnicas como *fine-tuning* (ajuste fino). Los modelos preentrenados constituyen una gran innovación, lo que mejora la accesibilidad, precisión, eficiencia computacional y velocidad del pronóstico.

Dado que los modelos de lenguaje de texto utilizan diccionarios de *tokens*, que son segmentos de caracteres representados vectorialmente según ciertos parámetros, es necesario tokenizar los valores de la serie temporal. El diccionario de *tokens* con el que operan los modelos de

lenguaje no es infinito, por lo tanto es necesario proyectar las observaciones a un set finito de *tokens*. Para cumplir esto, Chronos escala y discretiza las observaciones. TimeGTP por su parte usa las mismas observaciones como *tokens*, esto dado que, si bien su arquitectura es la de un modelo transformador, esta no está basada en ningún modelo de lenguaje existente, y en cambio trabaja con un modelo especializado en series de tiempo entrenado para minimizar el error de pronóstico.

Para el escalado se aplica a las observaciones una transformación del tipo $f(x_i) = (x_i - m)/s$. Existen variadas técnicas de escalado eligiendo apropiadamente m y s , pero se opta por elegir $m = 0$ y $s = \frac{1}{C} \sum_{i=1}^C |x_i|$ debido a que preserva los valores iguales a cero, los cuales pueden ser importantes de destacar en numerosas aplicaciones.

Sin embargo estos valores siguen siendo números reales y no pueden ser procesados directamente por un modelo de lenguaje. Es por esto que se discretizan las observaciones. Se seleccionan B centros de intervalos en la recta real, c_1, c_2, \dots, c_B , y $B - 1$ extremos b_i que los separen, $c_i < b_i < c_{i+1}$ para $i \in \{1, \dots, B - 1\}$. Las funciones de discretización $q : \mathfrak{R} \rightarrow \{1, 2, \dots, B\}$, y de descuantificación $d : \{1, 2, \dots, B\} \rightarrow \mathfrak{R}$ se definen como:

$$q(x) = \begin{cases} 1 & \text{si } -\infty \leq x < b_1 \\ 2 & \text{si } b_1 \leq x < b_2 \\ \vdots & \\ B & \text{si } b_{B-1} \leq x < \infty \end{cases} \quad \text{y} \quad d(j) = c_j \quad (10)$$

Una vez se hayan transformado las observaciones para poder ser leídas por el modelo, el funcionamiento de un transformador es el siguiente:

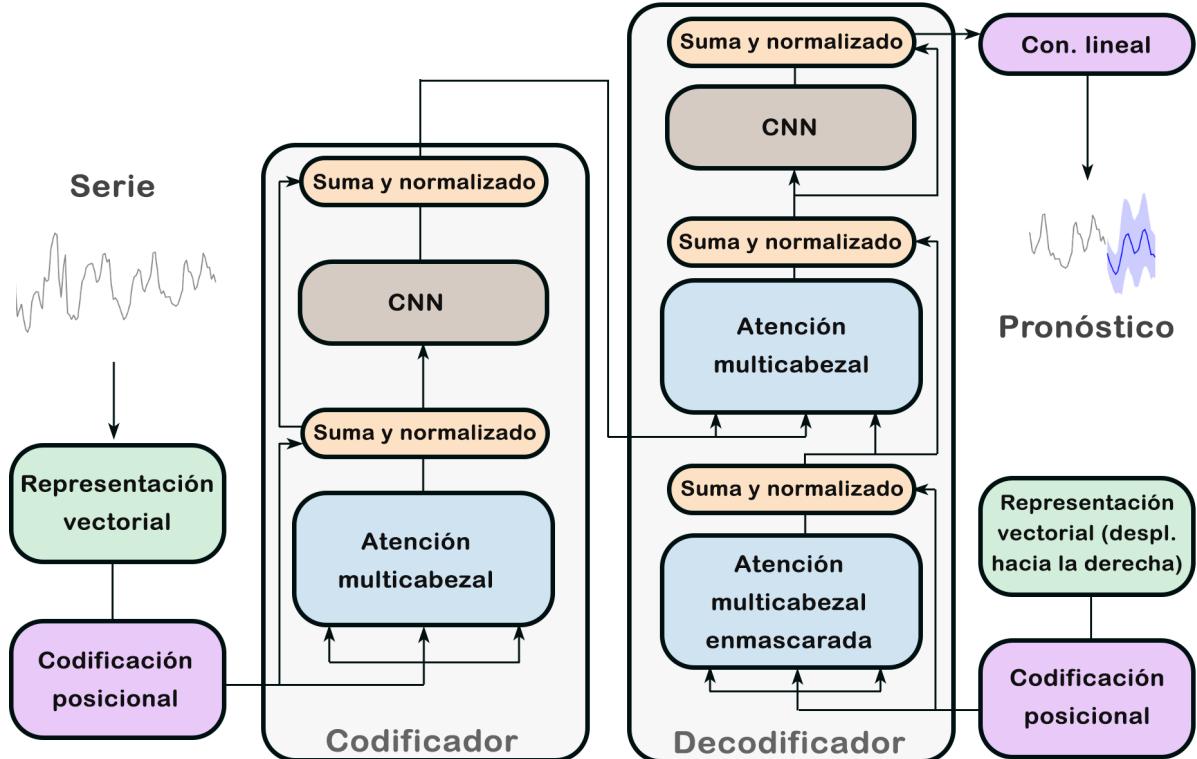


Figura 8: Diagrama de la estructura del modelo transformador de TimeGPT

Codificador

1. Representación vectorial: Cada *token* es transformado en un vector (vector de entrada) con muchas dimensiones (\vec{E}). Las dimensiones corresponden a diferentes características que el modelo definió en el preentrenado con una numerosa cantidad de parámetros.
2. Codificación posicional: Un set de valores adicionales o vectores son añadidos a los vectores de entrada antes de alimentarlos al modelo ($\vec{E} \leftarrow \vec{E} + \vec{P}$). Estas codificaciones posicionales tienen patrones específicos que agregan la información posicional del token.
3. Atención multi-cabezal (del inglés *Multi-Head Attention*): La autoatención opera en múltiples ‘cabezales de atención’ para capturar los diferentes tipos de relaciones entre tokens. Una cabeza de atención verifica el contexto en el que se presenta el token, y manipula los valores del vector que lo representa para añadir esta información contextual.

La verificación del contexto funciona gracias a una matriz denominada *Query* (W_Q) que examina ciertas características definidas con anterioridad en el preentrenado. El vector de entrada (\vec{E}) es multiplicado por esta matriz, resultando en un vector de consultas (\vec{Q}) para cada token. Una matriz de claves (W_K) que comprueba las relaciones con las características en la matriz de consultas, y tiene las mismas dimensiones que esta, es también multiplicada por \vec{E} generando así el vector de claves (\vec{K}). Luego, se forma una nueva matriz a partir de los productos cruzados entre los vectores \vec{K} y \vec{Q} de cada token, se divide por la raíz de la dimensión de los vectores¹ ($\sqrt{d_k}$) y se normaliza con *softmax*² por columna³ (\vec{S}), valores más altos indican que un token (de las columnas) está siendo influenciado por el comportamiento de otro token (de las filas).

| | a | fluffy | blue | creature | roamed | the | verdant | forest |
|---|------------------|------------------|------------------|------------------|------------------|------------------|------------------|------------------|
| | \vec{E}_1 | \vec{E}_2 | \vec{E}_3 | \vec{E}_4 | \vec{E}_5 | \vec{E}_6 | \vec{E}_7 | \vec{E}_8 |
| | $\downarrow W_Q$ |
| | \vec{Q}_1 | \vec{Q}_2 | \vec{Q}_3 | \vec{Q}_4 | \vec{Q}_5 | \vec{Q}_6 | \vec{Q}_7 | \vec{Q}_8 |
| $[a] \rightarrow \vec{E}_1 \xrightarrow{W_k} \vec{K}_1$ | +0.7 | -83.7 | -24.7 | -27.8 | -5.2 | -89.3 | -45.2 | -36.1 |
| $[\text{fluffy}] \rightarrow \vec{E}_2 \xrightarrow{W_k} \vec{K}_2$ | -73.4 | +2.9 | -5.4 | +93.0 | -48.2 | -87.3 | -49.7 | +7.8 |
| $[\text{blue}] \rightarrow \vec{E}_3 \xrightarrow{W_k} \vec{K}_3$ | -53.4 | -5.7 | +1.8 | +93.4 | -55.6 | -56.0 | -26.1 | -62.1 |
| $[\text{creature}] \rightarrow \vec{E}_4 \xrightarrow{W_k} \vec{K}_4$ | -21.5 | -29.7 | -56.1 | +4.9 | -32.4 | -92.3 | -9.5 | -28.1 |
| $[\text{roamed}] \rightarrow \vec{E}_5 \xrightarrow{W_k} \vec{K}_5$ | -20.1 | -40.9 | -87.8 | -55.4 | +0.6 | -64.7 | -96.7 | -18.9 |
| $[\text{the}] \rightarrow \vec{E}_6 \xrightarrow{W_k} \vec{K}_6$ | -87.9 | -33.3 | -22.6 | -31.4 | +5.5 | +0.6 | -4.6 | -96.8 |
| $[\text{verdant}] \rightarrow \vec{E}_7 \xrightarrow{W_k} \vec{K}_7$ | -41.2 | -55.5 | -42.3 | -59.8 | -79.0 | -97.9 | +3.7 | +93.8 |
| $[\text{forest}] \rightarrow \vec{E}_8 \xrightarrow{W_k} \vec{K}_8$ | -58.9 | -75.5 | -91.1 | -90.6 | -75.6 | -89.0 | -70.8 | +4.7 |

Figura 9: (before applying softmax) for example: Fluffy and Blue contribute to creature

Ahora que se sabe que tokens son relevantes para otros tokens, es necesario saber como son afectados. Para esto existe otra matriz de valores (W_V) que es multiplicada por cada vector de

¹Es útil para mantener estabilidad numérica, la cual describe cómo los errores en los datos de entrada se propagan a través del algoritmo. En un método estable, los errores debidos a las aproximaciones se atenúan a medida que la computación procede.

² $\text{softmax}(x) = \frac{e^{x_i/t}}{\sum_j e^{x_j/t}}$

³Aplicar *softmax* hace que cada columna se comporte como una distribución de probabilidad.

entrada resultando en los vectores de valor (\vec{V}) que son multiplicados a cada columna. La suma por filas devuelven el vector ($\Delta \vec{E}$) que debe ser sumado al vector de entrada original de cada token.

| Value matrix | | a | fluffy | blue | creature | roamed | the | verdant | forest | |
|--------------|---|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|--|
| W_V | | \vec{E}_1 | \vec{E}_2 | \vec{E}_3 | \vec{E}_4 | \vec{E}_5 | \vec{E}_6 | \vec{E}_7 | \vec{E}_8 | |
| a | $\rightarrow \vec{E}_1 \xrightarrow{W_V} \vec{V}_1$ | | | | | 0.00 | \vec{V}_1 | | | |
| fluffy | $\rightarrow \vec{E}_2 \xrightarrow{W_V} \vec{V}_2$ | | | | | 0.42 | \vec{V}_2 | | | |
| blue | $\rightarrow \vec{E}_3 \xrightarrow{W_V} \vec{V}_3$ | | | | | 0.58 | \vec{V}_3 | | | |
| creature | $\rightarrow \vec{E}_4 \xrightarrow{W_V} \vec{V}_4$ | | | | | 0.00 | \vec{V}_4 | | | |
| roamed | $\rightarrow \vec{E}_5 \xrightarrow{W_V} \vec{V}_5$ | | | | | 0.00 | \vec{V}_5 | | | |
| the | $\rightarrow \vec{E}_6 \xrightarrow{W_V} \vec{V}_6$ | | | | | 0.00 | \vec{V}_6 | | | |
| verdant | $\rightarrow \vec{E}_7 \xrightarrow{W_V} \vec{V}_7$ | | | | | 0.00 | \vec{V}_7 | | | |
| forest | $\rightarrow \vec{E}_8 \xrightarrow{W_V} \vec{V}_8$ | | | | | 0.00 | \vec{V}_8 | | | |

$$\Delta \vec{E}_j = \sum_i S_j \cdot \vec{V}_i \quad (11)$$

Con múltiples ‘cabezas’, cada una con sus propias matrices W_K , W_Q y W_V , se generan varios $\Delta \vec{E}$ que se suman y se añaden al vector de entrada original.

$$\vec{E} \Leftarrow \vec{E} + \sum_h \Delta \vec{E}_h \quad (12)$$

De forma resumida

$$\text{Atención}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (13)$$

Es importante notar que todas las matrices Q , K y V son preentrenadas.

4. Sumar y normalizar (*Add and norm*): En lugar de simplemente pasar los datos por las capas, las conexiones residuales se añaden sobre el vector de entrada en la salida de cada capa. Esto es lo que se hizo cuando se sumaron los cambios al vector de entrada, en lugar de directamente modificar el vector.

Dado que las redes neuronales profundas sufren de inestabilidad en los pesos al actualizarlos, una normalización al vector estabiliza el entrenamiento y mejora la convergencia.

5. Red neuronal convolucional (CNN): A diferencia de otros modelos transformadores tradicionales, TimeGPT incorpora *CNNs* para descubrir dependencias locales y patrones de corto plazo, tratando de minimizar una función de costo (MAPE, MAE, RMSE, etc.).

Decodificador

1. *Output embedding* (desplazado hacia la derecha): La entrada del decodificador son los tokens desplazados hacia la derecha.
2. Codificación posicional
3. *Masked multi-head attention* (Atención multicabezal enmascarada): Ahora que las predicciones deben hacerse únicamente con los valores previos a cada token, en el proceso de ‘atención’ y antes de aplicar la transformación softmax se debe reemplazar todos los valores debajo de la diagonal principal de la matriz QK por $-\infty$ para prevenir que los tokens sean influenciados por tokens anteriores.
4. *Multi-head attention*: En este caso, se usan las matrices de claves y valores que da como salida el codificador y la matriz de consultas es la salida de la capa de atención multicabezal enmascarada.
5. Conexión lineal: Es una capa completamente conectada que traduce las representaciones de atributos aprendidas en predicciones relevantes.

3.4.4 Diferencias entre TimeGPT y Chronos

TimeGPT es un modelo transformer preentrenado (de aquí las siglas GPT, *generative pretrained transformer*) para el pronóstico de series de tiempo que puede producir predicciones en diversas áreas y aplicaciones con gran precisión y sin entrenamiento adicional. El mismo fue desarrollado por Nixtla y tuvo su primera beta privada en Agosto de 2023, volviéndose accesible a todo público desde el 18 de julio de 2024.

Chronos es una familia de modelos transformadores preentrenados para series de tiempo basados en arquitecturas de modelos de lenguaje, el cual fue desarrollado por Amazon y lanzado en marzo de 2024.

La primera diferencia entre ambos modelos es la accesibilidad al código fuente, mientras que Chronos es de código abierto, el modelo desarrollado por Nixtla no lo es.

Tal vez la diferencia más importante es como operan los modelos. Se mencionó anteriormente que TimeGPT utiliza la arquitectura transformer para diseñar un modelo que pueda trabajar directamente con series de tiempo. Chronos en cambio, aprovecha los modelos de lenguaje existentes para aplicarlos a datos temporales. Esto conlleva a otras diferencias clave, como que Chronos debe transformar los datos antes de poder procesarlos, y por trabajar con datos discretos, está entrenado para minimizar la entropía cruzada entre las distribuciones de las categorías reales contra las predichas.

La función de pérdida utilizada por Chronos está dada por:

$$\ell(\theta) = \sum_{l=1}^{h+1} \sum_{i=1}^{|\nu_{ts}|} 1_{z_{n+l+1}=i} \log p_\theta(z_{n+l+1} = i | z_{1:n+l}) \quad (14)$$

Donde $|\nu_{ts}|$ es el tamaño del diccionario de tokens, el cual depende del número de intervalos creados. z_{n+h+1} es la serie transformada en *tokens* de la cual las primeras n observaciones se usarán como entrenamiento para pronosticar las siguientes h , y se agrega al final un token **EOS** que se utiliza comúnmente en los modelos de lenguaje para denotar el final de la secuencia. p_θ es la probabilidad estimada por el modelo bajo la parametrización θ .

Es importante notar que no es una función que detecta distancias, por lo que se espera que el modelo asocie a los intervalos cercanos gracias a la información en el conjunto de entrenamiento. Es decir que Chronos aplica regresión por clasificación.

Otra diferencia entre TimeGPT y Chronos es el tipo de red neuronal que utilizan para detectar patrones en los datos, mientras que el primero hace uso de las CNN, el segundo aplica *Feed-Forward Networks*. Luego de la conexión lineal, Chronos necesita volver a aplicar softmax para obtener las probabilidades del pronóstico, procedimiento que no es necesario por parte de TimeGPT.

3.5 Métricas de evaluación

Para comparar el rendimiento de los modelos se utilizan métricas cuantitativas. Para los pronósticos puntuales se usará el porcentaje del error absoluto medio (MAPE), mientras que para los pronósticos probabilísticos se aplicará el *Interval Score*, propuesto por Gneiting y Raftery (2007), que penaliza tanto la amplitud de los intervalos como la falta de cobertura. Estas comparaciones permiten evaluar la precisión, la robustez y la eficiencia de cada enfoque.

Sea $e_l = Z_{n+l} - \hat{Z}_n(l)$ el error de la l -ésima predicción, donde $\hat{Z}_n(l)$ representa el pronóstico l pasos hacia adelante, algunas medidas del error para pronósticos h pasos hacia adelante son:

- Error Cuadrático Medio (*Mean Square Error, MSE*):

$$MSE = \frac{1}{h} \sum_{l=1}^h e_l^2 \quad (15)$$

- Error absoluto medio (*Mean Absolute Error, MAE*):

$$MAE = \frac{1}{h} \sum_{l=1}^h |e_l| \quad (16)$$

- Porcentaje del error absoluto medio (*Mean Absolute Percentage Error, MAPE*)

$$MAPE = \left(\frac{1}{h} \sum_{l=1}^h \left| \frac{e_l}{Z_{n+l}} \right| \right) \cdot 100\% \quad (17)$$

El problema de estos errores es que solo tienen en cuenta la estimación puntual, y por lo general es buena idea trabajar con pronósticos probabilísticos para cuantificar la incertidumbre de los valores futuros de la variable. Gneiting y Raftery (2007, JASA) propusieron en *Strictly Proper Scoring Rules, Prediction, and Estimation* una nueva medida del error que tiene en cuenta los intervalos probabilísticos de la estimación, llamándola *Interval Score*:

$$S = \frac{1}{h} \sum_{l=1}^h (W_l + O_l + U_l) \quad (18)$$

Donde:

$$W_l = IS_l - II_l \quad (19)$$

$$O_l = \begin{cases} \frac{2}{\alpha} (Z_n(l) - Z_{n+l}) & \text{si } Z_n(l) > Z_{n+l} \\ 0 & \text{en otro caso} \end{cases} \quad U_l = \begin{cases} \frac{2}{\alpha} (Z_{n+l} - Z_n(l)) & \text{si } Z_n(l) < Z_{n+l} \\ 0 & \text{en otro caso} \end{cases} \quad (20)$$

Siendo IS_l e II_l los extremos superior e inferior del intervalo del l-ésimo pronóstico respectivamente. Es fácil darse cuenta que W es una penalización por el ancho del intervalo, y que O y U son penalizaciones por sobre y subestimación respectivamente.

Se considera mejor al modelo que minimiza estas métricas.

3.6 Técnicas para la estimación de parámetros

La correcta elección de los parámetros del modelo constituye una de las tareas más importantes para lograr un buen ajuste de los datos. No resulta conveniente utilizar todos los datos disponibles para este fin, ya que esto puede conducir a un sobreajuste (*overfitting*). Se habla de sobreajuste cuando el modelo ase adapta excesivamente a los datos de entrenamiento, lo que perjudica su capacidad para generalizar sobre datos nuevos. Para evitar este problema se aplican técnicas específicas de validación que permiten seleccionar los parámetros sin comprometer la capacidad predictiva del modelo.

La validación *holdout* consiste en reservar una parte del conjunto de entrenamiento como validación, e ir probando las métricas de evaluación de las distintas configuraciones de parámetros del modelo, ajustado sobre el resto de los datos de entrenamiento, sobre este nuevo conjunto. Luego, para ajustar el modelo con la mejor combinación de parámetros, se utilizarán tanto los datos de entrenamiento como de validación. Por la ordinalidad de los datos, el conjunto de validación tendrá que ser más reciente que el conjunto de entrenamiento.

$$\text{Conjunto de entrenamiento total : } \{ \underbrace{z_1, \dots, z_c}_{\text{Entrenamiento}}, \underbrace{z_{c+1}, \dots, z_n}_{\text{Validación}} \} \quad (21)$$

Si alguna serie presentara estacionalidad, se priorizará que el conjunto de validación tenga el largo del ciclo, para poder evaluar el ajuste del modelo en todo el ciclo.

Para ARIMA se usará el método de Box-Jenkins. En este método se compararán aquellos modelos que cumplan con los supuestos, y se elegirá como mejor combinación de parámetros aquella que minimize el AIC, medida de ajuste que penaliza por la cantidad de parámetros.

4. Aplicación

La aplicación empírica de esta tesina tuvo como objetivo implementar, ajustar y comparar los modelos presentados en la sección 3, utilizando un conjunto de series temporales seleccionadas. Para ello, se empleó el lenguaje de programación Python, junto con diversas librerías de código abierto.

Se trabajó con series temporales reales, obtenidas de bases de datos públicas y confiables. Cada serie fue modelada desde tres enfoques metodológicos distintos: modelos estadísticos tradicionales, algoritmos de aprendizaje automático y modelos de aprendizaje profundo. Para cada uno de ellos, se exploraron distintas configuraciones de parámetros, explicando su significado y función en el ajuste.

- Los modelos estadísticos clásicos, como ARIMA y SARIMA, serán ajustados utilizando la librería `pmdarima`. La selección de parámetros se realizó manualmente y mediante procedimientos automáticos, tomando como criterio principal el Criterio de Información de Akaike (AIC).
- Para el enfoque de aprendizaje automático, se emplearon algoritmos de boosting, específicamente XGBoost y LightGBM, implementados con las librerías `xgboost` y `lightgbm` respectivamente.
- En el caso de los modelos de aprendizaje profundo, se entrenaron redes LSTM utilizando la librería `neuralforecast`.
- Finalmente, se exploraron dos modelos fundacionales preentrenados: TimeGPT, accedido a través de la API de Nixtla mediante la librería `nixtla`, y Chronos, una familia de modelos preentrenados desarrollada por *Amazon Web Services* cuya implementación se llevó a cabo con la librería `autogluon`.

Cada modelo fue evaluado en función de su desempeño predictivo, utilizando métricas como el error absoluto porcentual medio (MAPE) y el *Interval Score*. Además, se consideraron aspectos adicionales como el tiempo de cómputo, la facilidad de implementación y la interpretabilidad de los resultados. Los resultados fueron presentados en tablas comparativas y visualizaciones gráficas, acompañadas de un análisis crítico.

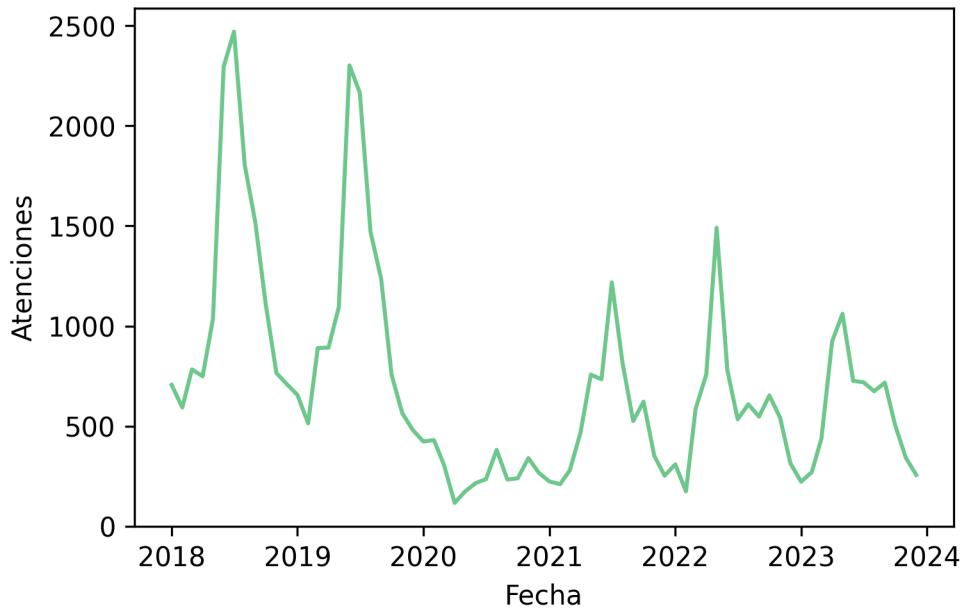
4.1 Series a utilizar

Se analizaron 3 series con distintas características con el objetivo de evaluar los resultados bajo distintas circunstancias, estas son:

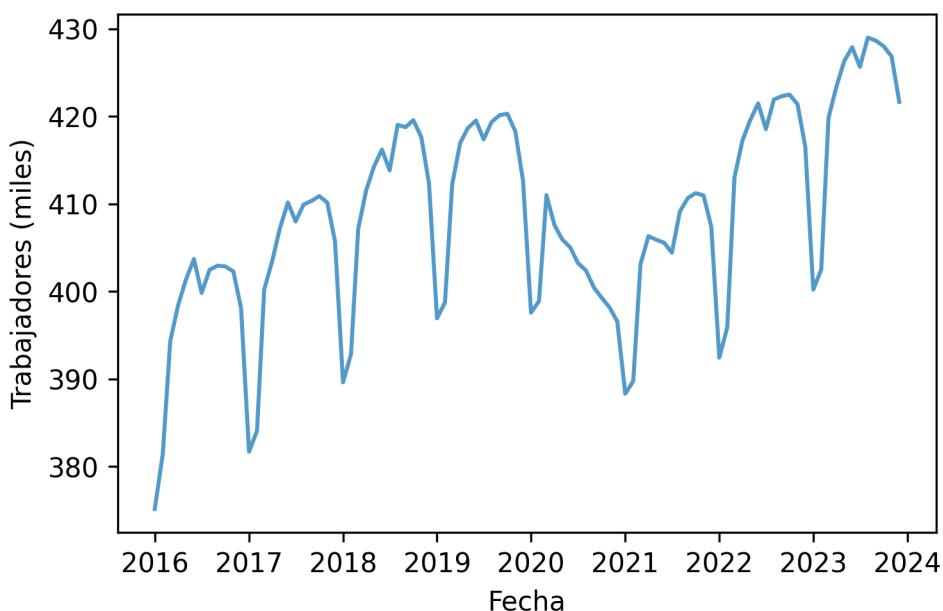
- Atenciones de guardia mensuales por patologías respiratorias en un hospital de Rosario.
- Personas registradas con empleo asalariado por mes en el sector educativo privado de Argentina.
- Temperatura horaria en la ciudad de Rosario.

La primera serie analizada corresponde al número mensual de atenciones de guardia por patologías respiratorias (Códigos CIE10: J09–J18, J21, J22 y J44) en el Hospital de Niños Víctor J. Vilela de la ciudad de Rosario. Esta información fue provista por la Dirección General de Estadística de la Municipalidad de Rosario. La serie mostró una marcada estacionalidad, con picos en los meses de invierno, y una fuerte caída en 2020, atribuida a las medidas sanitarias adoptadas durante la pandemia de COVID-19. Esta estacionalidad es más discernible en el

gráfico Figura 19 del anexo. El objetivo para esta serie es pronosticar las atenciones del año 2024.

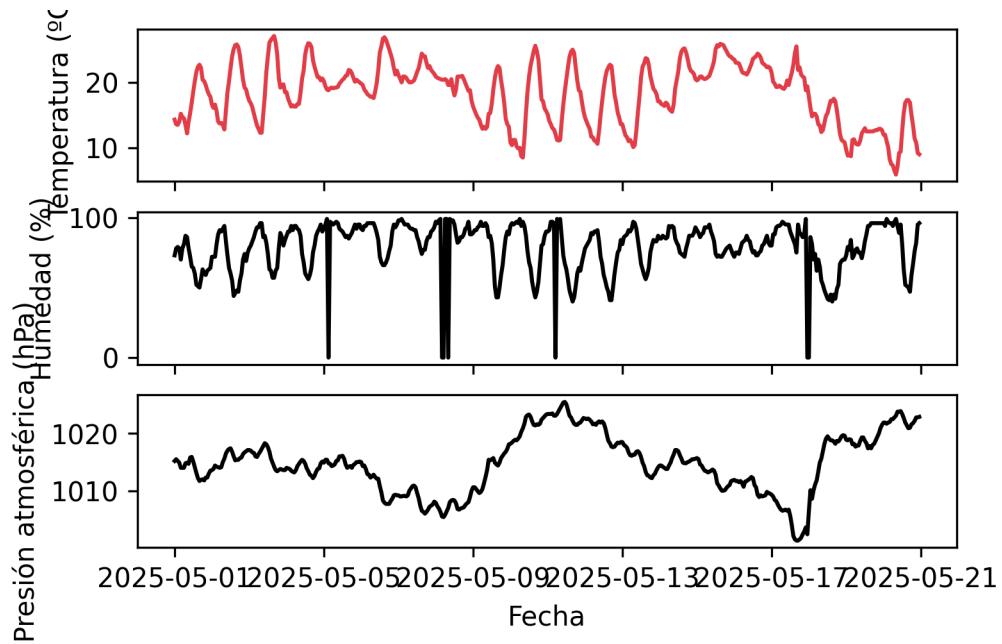


La cantidad de personas en Argentina con empleo asalariado en el área de enseñanza y registradas en el sector privado aumenta casi constantemente año tras año. Presenta descensos drásticos en los meses de diciembre y enero. En esta serie también se evidencian las consecuencias de la pandemia. El objetivo es pronosticar la cantidad de trabajadores registrados en el área de enseñanza en el año 2024. Los datos fueron relevados del informe de [Situación y evolución del Trabajo Registrado](#) elaborado por el Instituto Nacional de Estadísticas y Censos (INDEC).



Por último, se analizarán las temperaturas por hora a lo largo de los días del mes de marzo 2025, estudiando sus comportamientos en relación a la humedad relativa y a la presión atmosférica estándar. Gracias al gráfico Figura 20 del anexo, se puede observar el patrón estacional diario que tiene la temperatura, la cual se mantiene constante entre la noche y la mañana, pero a la

tarde sube pronunciadamente. Se estimarán las temperaturas horarias del 21 y 22 de mayo de 2025. La información necesaria fue extraída de la página del [Servicio Meteorológico Nacional](#).



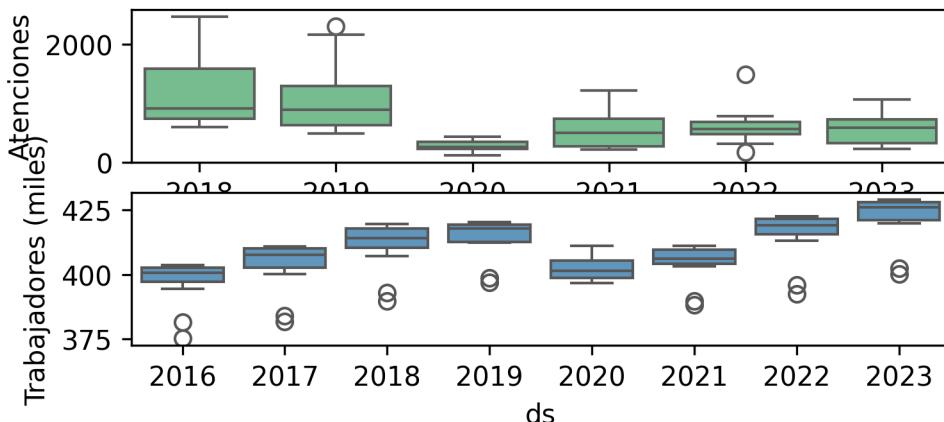
4.2 Modelado

4.2.1 ARIMA

Por la complejidad agregada de pronosticar usando ARIMA en series con periodicidad diaria y variables exógenas, la serie de temperatura se modelizará únicamente con el método automático que se menciona más adelante.

Antes de proponer modelos, se debe hacer un breve análisis exploratorio de las series. En primer lugar se debe observar que la variancia en cada período es la misma. De lo contrario se debería transformar la variable que se desea pronosticar.

```
Text(0, 0.5, 'Trabajadores (miles)')
```

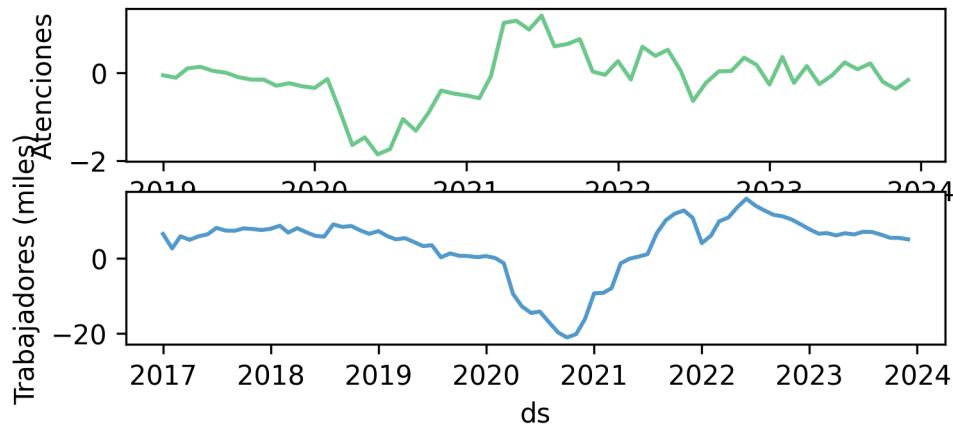


Dadas las diferencias en la variabilidad anual de las atenciones por año, se calculó λ para la transformación de Box y Cox concluyendo que las atenciones en guardia por patologías

respiratorias necesitan ser transformadas aplicando la función exponencial ($\lambda = \text{python round(atenciones_lambda, 2)}$).

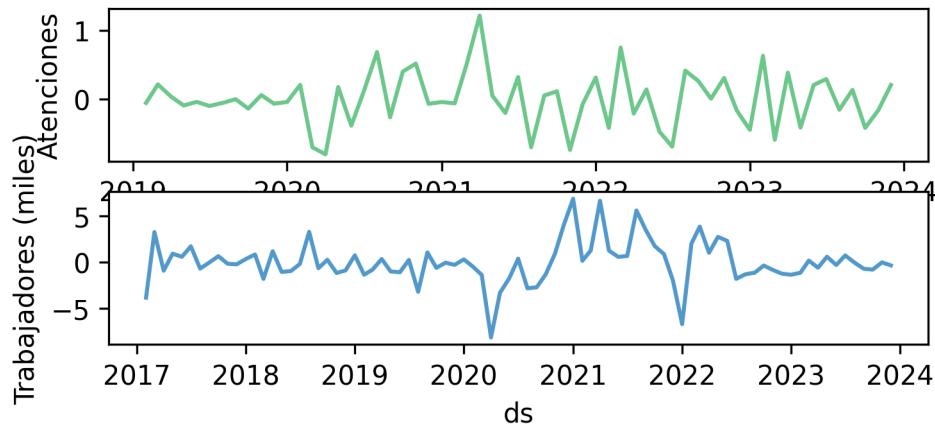
El primer paso de la modelización es estudiar las estacionalidades. Se mencionó anteriormente que las 2 series tienen estacionalidad, es por esto que se diferencian y se grafican, buscando que no sea visible ningún patrón estacional.

Text(0, 0.5, 'Trabajadores (miles)')



Una vez diferenciadas estacionalmente, es importante que no exista una tendencia clara, por lo que se estandarizan estacionariamente.

Text(0, 0.5, 'Trabajadores (miles)')



Para descubrir las componentes AR y MA que afectan a las series se grafican las autocorrelaciones y autocorrelaciones parciales de cada una.

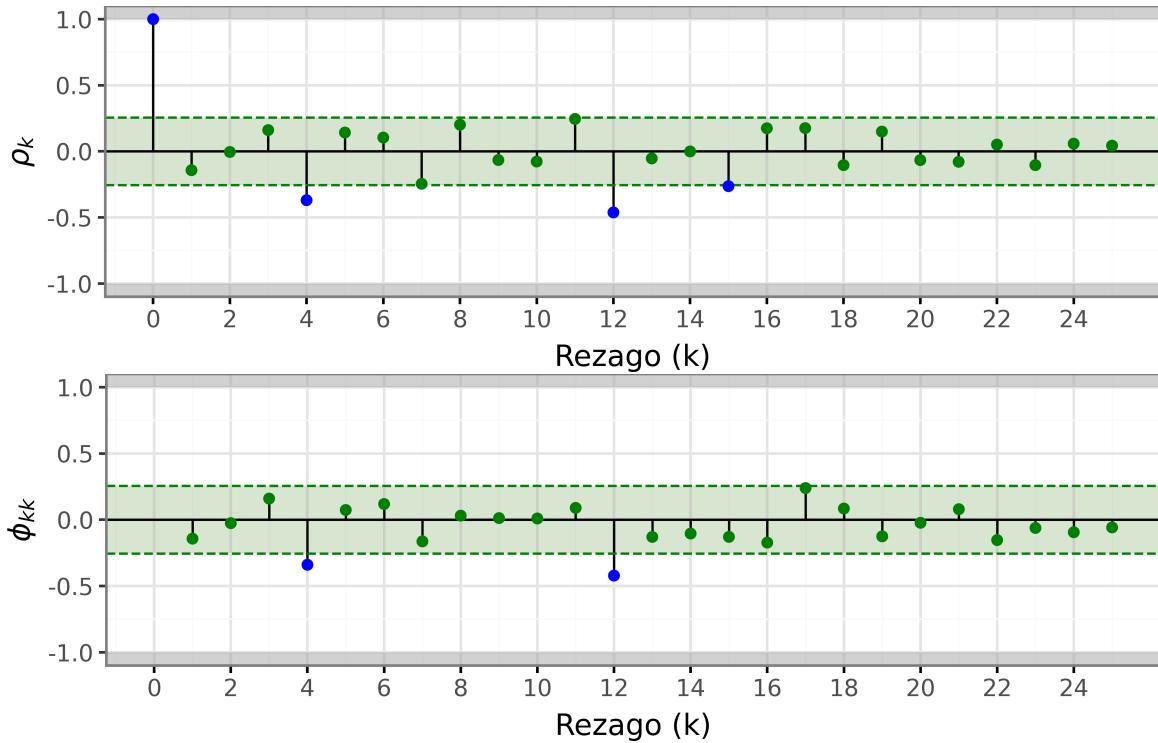


Figura 10: Autocorrelaciones de la serie de atenciones

En la serie de atenciones se puede ver que tanto en las autocorrelaciones como en las autocorrelaciones parciales, hay valores significativos en el quinto y doceavo rezago, lo que podrían significar que existen componentes MA y AR en la parte estacional o estacionaria de la serie.

Se proponen entonces los modelos $SARIMA(0, 1, 1)(0, 1, 0)_{12}$ y $SARIMA(0, 1, 0)(0, 1, 1)_{12}$.

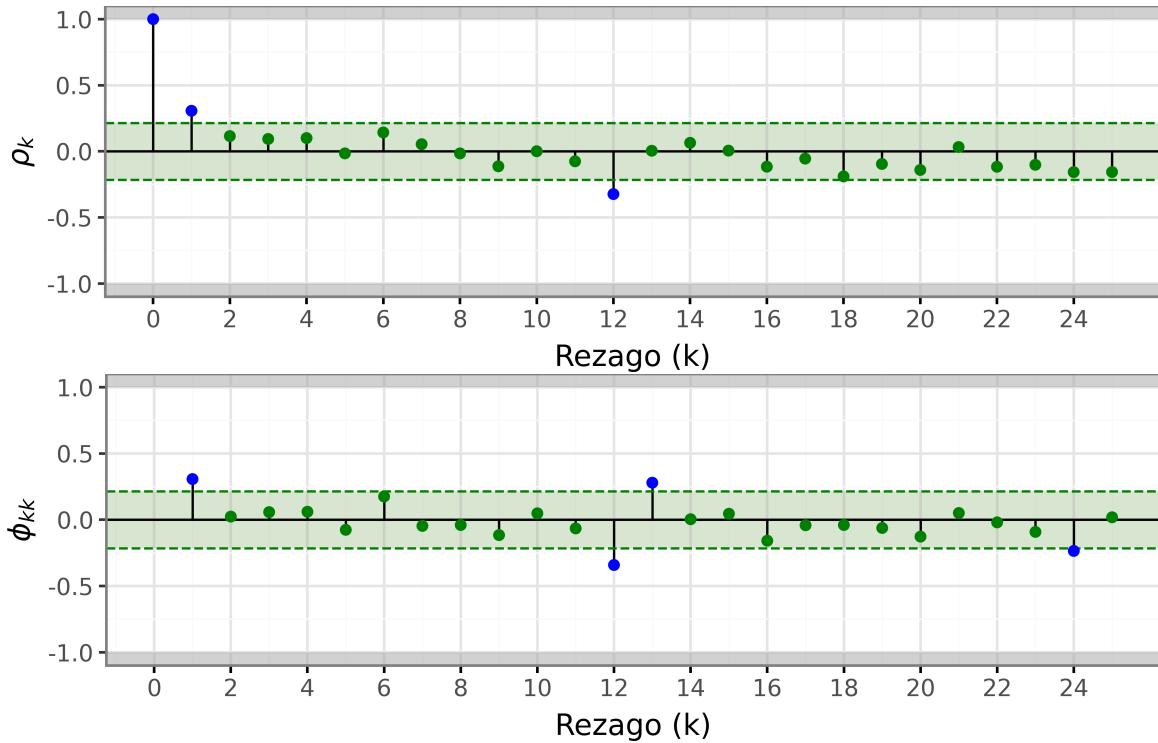


Figura 11: Autocorrelaciones de la serie de trabajadores

En las autocorrelaciones de la serie de trabajadores resultan significativos el primer y doceavo rezago, mientras que en las autocorrelaciones parciales se destacan los rezagos 1, 12, 13 y 24. Se puede suponer que el rezago 13 no es en realidad significativo y que la serie presenta una componente MA en la parte estacionaria y una AR en la estacional, formando el modelo $SARIMA(0, 1, 1)(1, 1, 0)_{12}$.

Además, por medio de la función `auto_arima` de la librería `pmdarima` se propusieron modelos automáticos. Se elegirá como mejor modelo aquel que minimice el Criterio de Información de Akaike (AIC) y cumpla con las propiedades del modelo ARIMA.

Tabla 1: Cumplimiento de las condiciones de estacionariedad e invertibilidad de los modelos ajustados.

| Modelo | AIC | Parte regular | | Parte estacional | |
|----------------------------------|--------|---------------|------|------------------|------|
| | | Est. | Inv. | Est. | Inv. |
| Atenciones en guardia | | | | | |
| $SARIMA(0, 1, 1)(0, 1, 0)_{12}$ | 857.6 | Si | Si | Si | Si |
| $SARIMA(0, 1, 0)(0, 1, 1)_{12}$ | 852.1 | Si | Si | Si | Si |
| $SARIMA(0, 1, 0)(1, 0, 0)_{12}$ | 1013.5 | Si | Si | Si | Si |
| Trabajadores | | | | | |
| $SARIMA(0, 1, 1)(1, 1, 0)_{12}$ | 359.9 | Si | Si | Si | Si |
| $SARIMA(2, 0, 0)(2, 1, 0)_{12}$ | 356.7 | Si | Si | Si | Si |
| Temperatura | | | | | |
| $SARIMAX(1, 1, 1)(2, 0, 1)_{24}$ | 1067.1 | Si | Si | No | Si |

Para la serie de atenciones se seguirá trabajando con el segundo modelo propuesto y con el seleccionado de manera automática, ya que el parámetro MA del primer modelo propuesto no es significativo. Por otro lado, los dos modelos probados para la serie de

trabajadores seguirán siendo utilizados. El modelo automático para la serie de temperaturas no goza de estacionariedad, por lo que se proponen los modelos $SARIMAX(1, 1, 1)(1, 0, 1)_{24}$, $SARIMAX(1, 1, 0)(2, 0, 1)_{24}$ y $SARIMAX(1, 1, 0)(1, 1, 0)_{24}$, de los cuales únicamente el último cumple con todas las propiedades buscadas, con un AIC de 1119.3.

Las salidas de los modelos se pueden ver en el [anexo](#) para comprobar los resultados.

Lo siguiente es comprobar que los residuos estandarizados se comporten como ruido blanco.

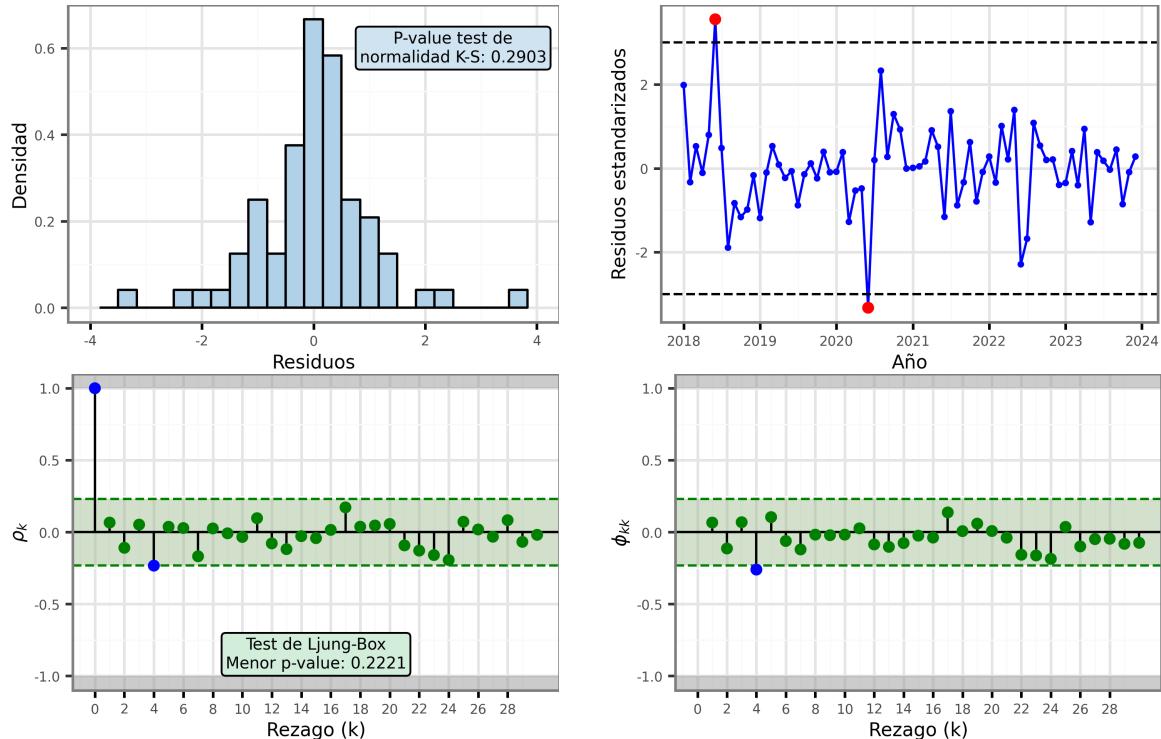


Figura 12: Comprobación de supuestos del segundo modelo arima manual para la serie de atenciónes

El histograma muestra la distribución de los residuos, pudiendo ver y comprobar con el test de Kolmogorov-Smirnov que es normal. La serie de los residuos muestra como estos no tienen un patrón particular ni variabilidad dependiente del tiempo, solo 2 *outlayers* fáciles de ignorar. Por último, los gráficos inferiores muestran las autocorrelaciones y autocorrelaciones parciales, esperando que no haya ninguna significativa. Por medio del test de Ljung-Box se comprueba que ninguna lo es y se concluye que los residuos no están correlacionados. Por lo tanto todos los supuestos se cumplen para este modelo.

Con el propósito de no poblar de gráficos el documento, el resto de los gráficos para la comprobación de supuestos de los demás modelos se encuentran en el [anexo](#). En estos se destaca que todos los modelos, con excepción del seleccionado automáticamente para los trabajadores, superaron la comprobación de supuestos. El problema de este modelo es la existencia de correlación en los residuos, supuesto que es sumamente importante al ajustar modelos ARIMA. Si bien los residuos del modelo seleccionado manualmente para los trabajadores no cumple con normalidad, es sencillo ver que esto se debe a unos pocos valores extremos, y que sin estos es muy probable que se sí se distribuyan normalmente.

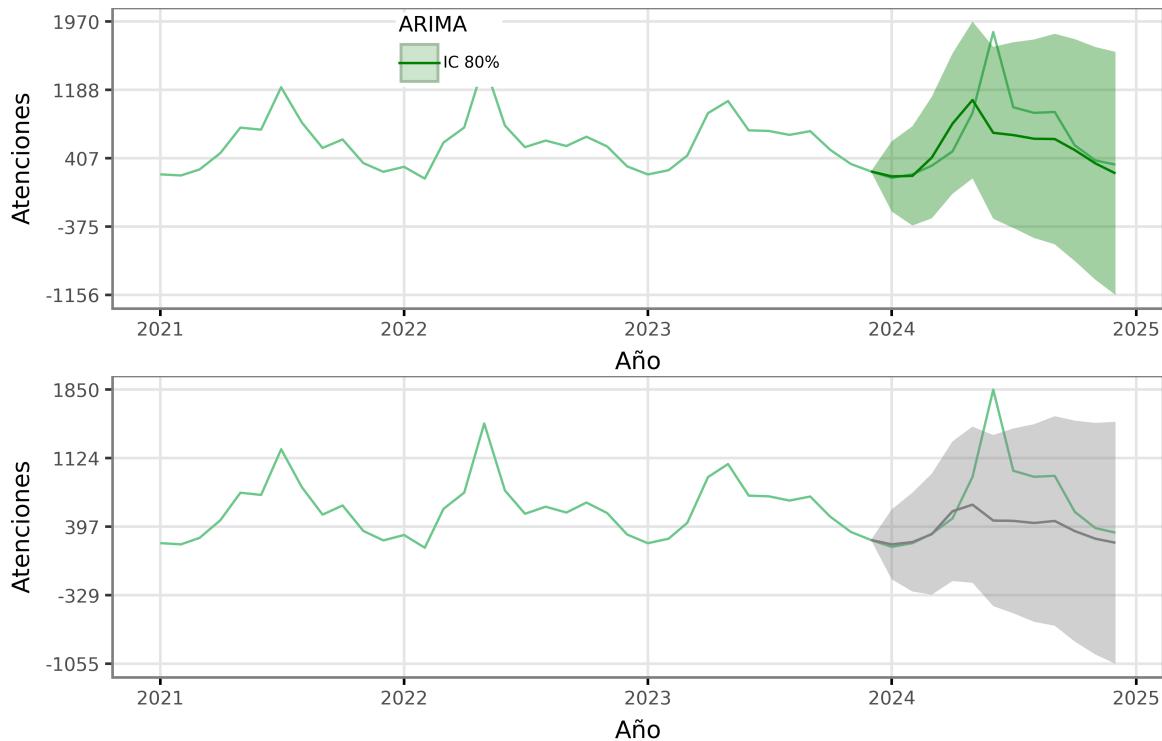


Figura 13: Pronostico de atenciones con ARIMA

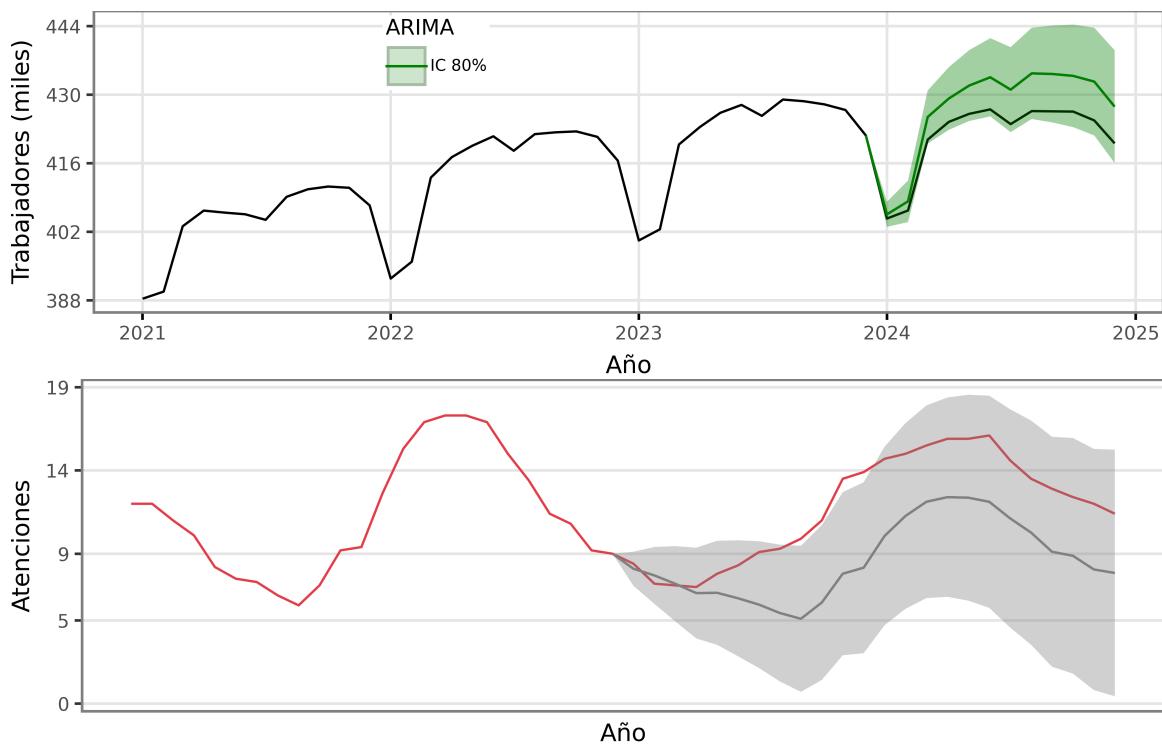


Figura 14: Pronostico de trabajadores y temperatura con ARIMA

Uno de los problemas de pronosticar con ARIMA es que para trabajar con variables exógenas se debe conocer los valores futuros de estas. Para este caso se asume que los valores de humedad y presión atmosférica son conocidos para el día 21, sin embargo, para el resto de aplicaciones esto no será así y se considera esto como una desventaja de este modelo de pronóstico.

Otro problema que se encuentra es que el intervalo de confianza para las atenciones por guardia toma valores negativos, algo claramente ilógico. Esto se puede resolver ajustando modelos sobre el logaritmo de la variable, y luego transformando los valores pronosticados a la escala original. Esta solución queda propuesta para un futuro trabajo y se usan los resultados como exposición de los problemas para pronosticar usando la familia de modelos ARIMA.

Tabla 2: Métricas de evaluación de los modelos ARIMA.

| Modelo | MAPE | <i>Interval Score</i> |
|--|--------|-----------------------|
| Atenciones en guardia | | |
| <i>SARIMA</i> (0, 1, 0)(0, 1, 1) ₁₂ | 0.2794 | 2096.9139 |
| <i>SARIMA</i> (0, 1, 0)(1, 0, 0) ₁₂ | 0.3328 | 2205.1062 |
| Trabajadores registrados | | |
| <i>SARIMA</i> (0, 1, 1)(1, 1, 0) ₁₂ | 0.0137 | 15.7902 |
| Temperatura | | |
| <i>SARIMA</i> (1, 1, 0)(1, 1, 0) ₂₄ | 0.2606 | 10.7507 |

4.2.2 Modelos de aprendizaje automático

Los modelos de pronóstico basados en el aprendizaje automático no tienen supuestos que cumplir, por lo que el modelaje se vuelve mucho más sencillo y automatizable. Los únicos aspectos en los que se debe tener especial cuidado y atención es en la selección de características y parámetros del modelo.

Tanto para XGBoost como para LightGBM las características elegidas para todas las series fueron las siguientes:

- Identificación temporal
- El promedio de las 3 observaciones anteriores
- El desvío estándar de las 3 observaciones anteriores
- El valor del primer rezago
- El valor del segundo rezago
- El valor del rezago estacional

Con el objetivo de seleccionar los mejores hiperparámetros, se elaboró una grilla con diferentes combinaciones de valores para cada parámetro. Se ajustaron modelos cumpliendo cada combinación en la grilla sobre un conjunto de validación.

XGBoost y Lightgbm permiten parametrizar los modelos de varias formas, los que se eligieron ajustar en este trabajo son los siguientes:

- Número de árboles que se contruyen paralelamente en cada iteración (A). Opciones: 20, 50, 100, 150.
- Profundidad máxima del árbol (P). Opciones: 2, 3, 4, 5.
- Número máximo de hojas del árbol (H). Opciones: 2, 4, 8, 16.
- Tasa de aprendizaje en el método del gradiente (η). Opciones: 0.001, 0.1, 0.2.
- Proporción de características que se usa en cada árbol (C). Opciones: 0.7, 1.

Tabla 3: Modelos XGBoost seleccionados y métricas de evaluación.

| Serie | A | P | H | η | C | MAPE | Interval Score |
|--------------------------|-----|-----|-----|--------|-----|--------|----------------|
| Atenciones en guardia | 100 | 2 | 2 | 0.2 | 1.0 | 0.3684 | 1515.9289 |
| Trabajadores registrados | 150 | 5 | 2 | 0.2 | 1.0 | 0.0069 | 21.9308 |
| Temperatura | 50 | 2 | 2 | 0.2 | 1.0 | 0.0583 | 2.7636 |

Tabla 4: Modelos LightGBM seleccionados y métricas de evaluación.

| Serie | A | P | H | η | C | MAPE | Interval Score |
|--------------------------|-------|-----|-----|--------|-----|--------|----------------|
| Atenciones en guardia | 100.0 | 2.0 | 4.0 | 0.2 | 0.7 | 0.5132 | 1527.5973 |
| Trabajadores registrados | 100.0 | 2.0 | 4.0 | 0.1 | 0.7 | 0.0186 | 40.8785 |
| Temperatura | 100.0 | 2.0 | 2.0 | 0.2 | 1.0 | 0.0931 | 4.6673 |

En las tablas Tabla 3 y Tabla 4 se muestran para cada serie que combinación de parámetros, de las 384 posibles, fue la que menor MAPE produjo sobre el conjunto de validación aplicando XGBoost o LightGBM respectivamente. Además, se presenta el MAPE e *Interval Score* que devuelve el modelo entrenado con todos los datos de entrenamiento y evaluado sobre el conjunto de prueba.

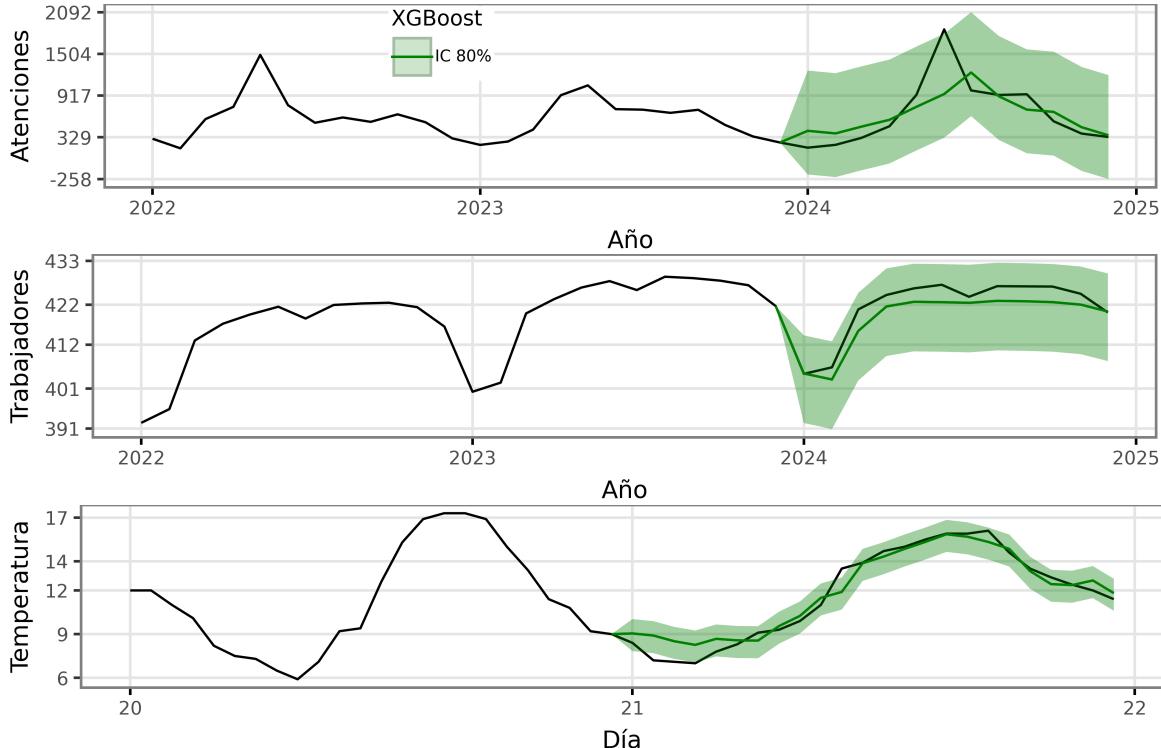


Figura 15: Pronósticos con XGBoost

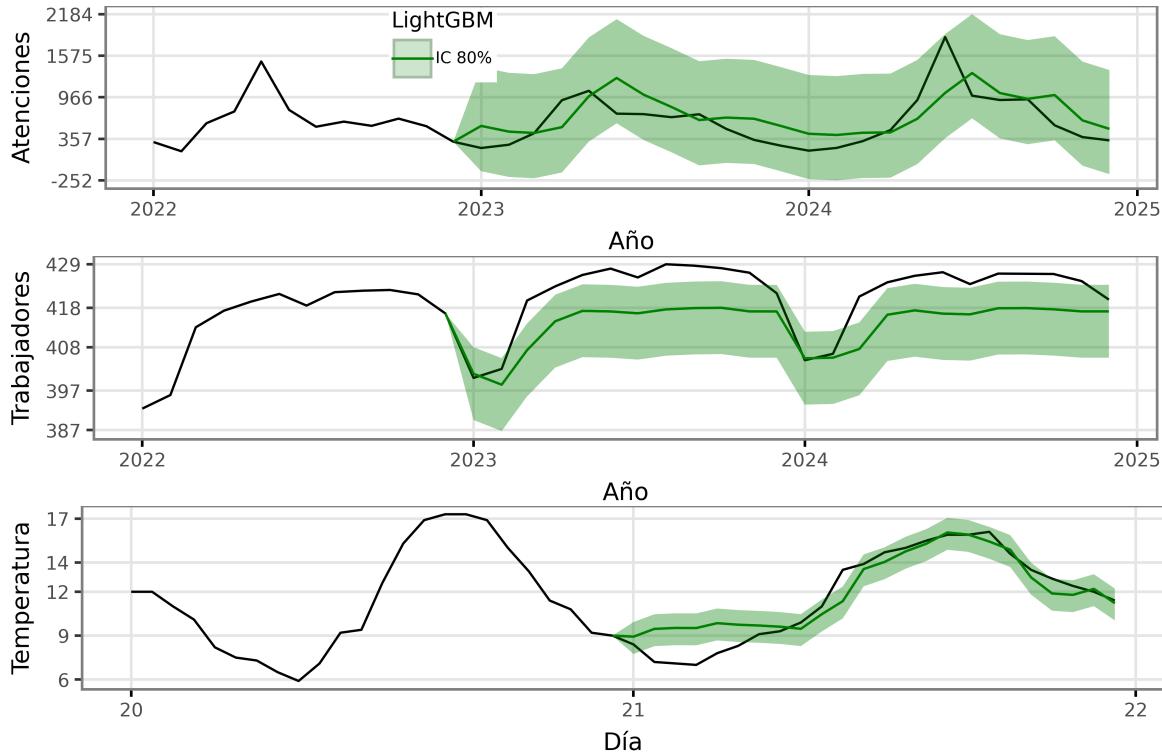


Figura 16: Pronosticos con LightGBM

4.2.3 LSTM

Los modelos basados en aprendizaje profundo se encargan de definir las características más relevantes y los parámetros más adecuados de forma automática. En las redes neuronales, como es el caso de LSTM, solo se tendrá que elegir la forma del modelo, es decir, las capas y la cantidad de iteraciones.

Para evitar el sobreajuste en redes neuronales existen numerosas alternativas. Entre estas se puede optar por frenar el entrenamiento antes de completar todas las iteraciones si es que no se ve mejora en la función de pérdida, esto se conoce como *early stop*. Otra opción es *ignorar* una cierta proporción de neuronas en cada iteración, que es equivalente a entrenar distintas redes neuronales. Esta última técnica es denominada *dropout*.

- Número de iteraciones (N). Opciones: 50, 100, 200, 500.
- Número de capas del codificador (C). Opciones: 1, 2, 3.
- Número de capas del decodificador (D). Opciones: 1, 2, 3.
- Tasa de aprendizaje (η). Opciones: 0.001, 0.1, 0.2.
- Paciencia para el *early stop* (P). Opciones: 5, 10, ∞ .
- *Dropout* (O). Opciones: 0, 0.1, 0.3.

Cada 10 iteraciones se evalúa la función de perdida, que la paciencia sea igual a 5 significa que si la función de pérdida no mejora en las últimas 5 iteraciones, se detiene el entrenamiento. Que el *dropout* sea 0.1 lleva a que cada neurona tenga una probabilidad de 0.1 de que se apague en cada iteración.

Codificador hace referencia al modelo LSTM, mientras que decodificador a un modelo perceptrón multicapa simple que garantiza eficiencia y estabilidad.

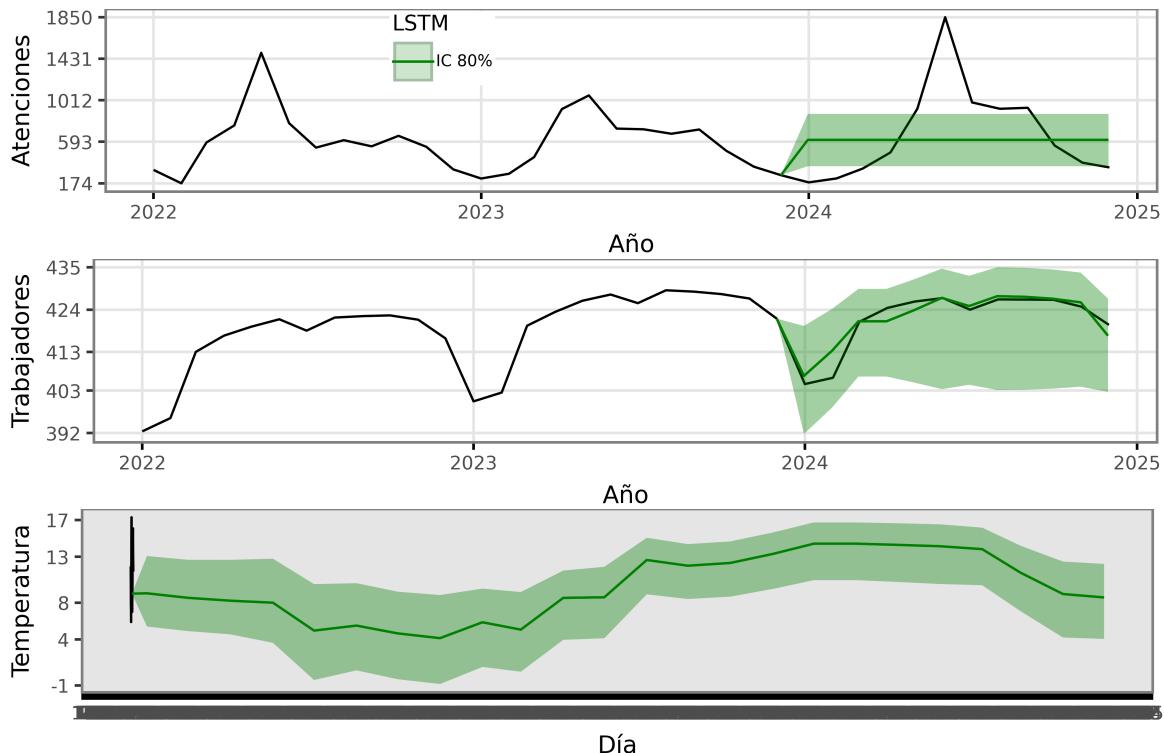


Figura 17: Pronosticos con LSTM

4.2.4 Modelos fundamentales

La mejoría de los modelos fundamentales por sobre las redes neuronales convencionales radica en que el ajuste de parámetros ya fue realizado con un preentrenamiento en grandes conjuntos de datos. Aún así, es posible utilizar la arquitectura transformer y realizar un ajuste más personalizado conocido como *fine tuning*.

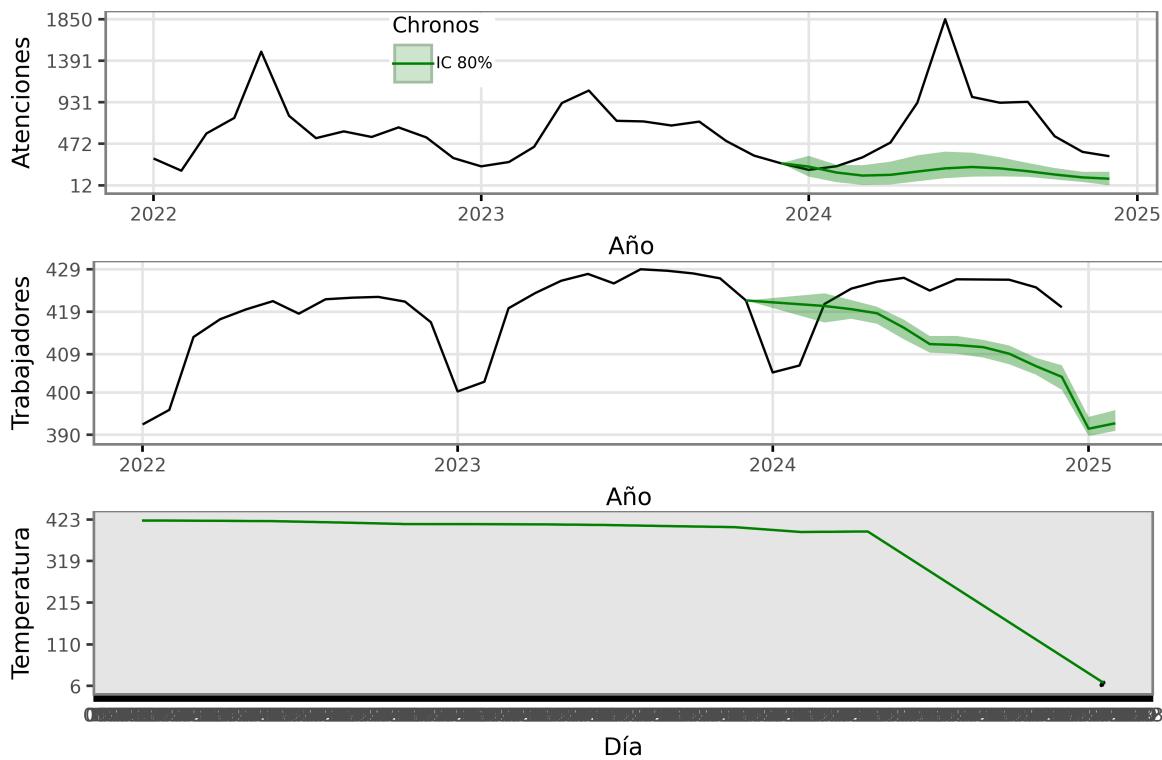


Figura 18: Pronosticos con Chronos

4.3 Comparaciones

Serie 1

| | Modelo | MAPE | Interval Score | Tiempo |
|---|----------|----------|----------------|------------|
| 0 | ARIMA | 0.332828 | 2205.106225 | 4.721362 |
| 1 | XGBoost | 0.368438 | 1515.928887 | 0.448006 |
| 2 | LightGBM | 0.513173 | 1527.597301 | 0.396870 |
| 3 | LSTM | 0.739148 | 6069.787537 | 53.060509 |
| 4 | Chronos | 0.685150 | 4363.816800 | 150.599766 |

Serie 2

| | Modelo | MAPE | Interval Score | Tiempo |
|---|----------|----------|----------------|------------|
| 0 | ARIMA | 0.007843 | 15.533570 | 22.484971 |
| 1 | XGBoost | 0.006866 | 21.930821 | 1.160061 |
| 2 | LightGBM | 0.018553 | 40.878508 | 0.362787 |
| 3 | LSTM | 0.004358 | 249.896690 | 5.727015 |
| 4 | Chronos | 0.029152 | 107.245707 | 157.101573 |

Serie 3

| | Modelo | MAPE | Interval Score | Tiempo |
|---|--------|----------|----------------|------------|
| 0 | ARIMA | 0.241747 | 9.332746 | 345.815492 |

| | Modelo | MAPE | Interval Score | Tiempo |
|---|----------|----------|----------------|-------------|
| 1 | XGBoost | 0.058265 | 2.763566 | 0.492999 |
| 2 | LightGBM | 0.093115 | 4.667262 | 0.336756 |
| 3 | LSTM | 0.234521 | 72.359019 | 14.237005 |
| 4 | Chronos | 0.142925 | 8.885428 | 1461.018784 |

5. Conclusiones

Problemas arima: consume mucho tiempo, necesita muchos conocimientos teoricos para problemas específicos (variables de conteo, evitar negativos, variancia no constante), puede que se trabaje hasta llegar a un modelo que luego no cumpla los supuestos, no es util cuando se tienen variables exogenas, porque solo permite variables ya conocidas

En este documento se presentan 3 contribuciones claves para el campo de la estadística. En primer lugar y como objetivo principal de esta tesina, la introducción de los modelos transformadores para el pronóstico de series temporales. Además se incluyeron aportes como evaluar el desempeño de los modelos con una nueva medida del error, el *interval score*, el cual no tiene en cuenta únicamente el pronóstico puntual sino también probabilístico. El último gran aporte es la construcción de intervalos de pronóstico por medio de *conformal predictions*, que no depende de conocer la distribución de los residuos.

6. Bibliografía

- Ansari et al.** (2024). *Chronos: Learning the Language of Time Series*. Transactions on Machine Learning Research. <https://arxiv.org/abs/2403.07815>
- Awan, A.** (2 de septiembre de 2024). *Time Series Forecasting With TimeGPT*. Datacamp. <https://www.datacamp.com/tutorial/time-series-forecasting-with-time-gpt>
- Bermejo, J.** (21 de mayo de 2024). *Redes neuronales*. Facultad de Ciencias Económicas y Estadística de la Universidad Nacional de Rosario.
- Elhariri, K.** (1 de marzo de 2022). *The Transformer Model*. Medium. <https://medium.com/data-science/attention-is-all-you-need-e498378552f9>
- Gilliland, M., Sglavo, U., & Tashman, L.** (2016). *Forecast Error Measures: Critical Review and Practical Recommendations*. John Wiley & Sons Inc.
- Gneiting, T., & Raftery A. E.** (2007). *Strictly Proper Scoring Rules, Prediction, and Estimation*. Journal of the American Statistical Association, 102(477), 359–378. <https://doi.org/10.1198/016214506000001437>
- Hyndman, R. J., & Athanasopoulos, G.** (2021). *Forecasting: principles and practice (3rd ed.)*. OTexts. <https://otexts.com/fpp3/>
- IBM.** (s.f.). *Explainers*. Recuperado el 14 de marzo de 2025 en <https://www.ibm.com/think/topics>
- Kamtziris, G.** (27 de febrero de 2023). *Time Series Forecasting with XGBoost and LightGBM: Predicting Energy Consumption*. Medium. <https://medium.com/@geokam/time-series-forecasting-with-xgboost-and-lightgbm-predicting-energy-consumption-460b675a9cee>
- Korstanje, J.** (2021). *Advanced Forecasting with Python*. Apress.
- Nielsen, A.** (2019). *Practical Time Series Analysis: Prediction with Statistics and Machine Learning*. O'Reilly Media.
- Nixtla.** (s.f.-a). *About TimeGPT*. Recuperado en diciembre de 2024 de https://docs.nixtla.io/docs/getting-started-about_timegpt
- Nixtla.** (s.f.-b). *LSTM*. Recuperado el 9 de abril de 2025 en <https://nixtlaverse.nixtla.io/neuralforecast/models.lstm.html#lstm>
- Parmezan, A., Souza, V., & Batista, G.** (1 de mayo de 2019). *Evaluation of statistical and machine learning models for time series prediction: Identifying the state-of-the-art and the best conditions for the use of each model*. Information Sciences. <https://www.sciencedirect.com/science/article/abs/pii/S0020025519300945>
- Prunello, M., & Marfetán, D.** (12 de mayo de 2024). *Árboles de decisión*. Facultad de Ciencias Económicas y Estadística de la Universidad Nacional de Rosario.
- Sanderson, G.** [3Blue1Brown]. (2024). *Attention in transformers, step-by-step / DL6 [Video]*. Youtube. <https://www.youtube.com/watch?v=eMlx5fFNoYc&t=1204s>
- Sanderson, G.** [3Blue1Brown]. (2024). *Transformers (how LLMs work) explained visually / DL5 [Video]*. Youtube. <https://www.youtube.com/watch?v=wjZofJX0v4M>
- Shastri, Y.** (26 de abril de 2024). *Attention Mechanism in LLMs: An Intuitive Explanation*. Datacamp. <https://www.datacamp.com/blog/attention-mechanism-in-langs-intuition>

Silberstein, E. (7 de noviembre de 2024). *Tracing the Transformer in Diagrams*. Medium. <https://medium.com/data-science/tracing-the-transformer-in-diagrams-95db68160c>

Vaswani et al. (2017). *Attention is all you need*. Google. <https://arxiv.org/pdf/1706.03762>

7. Anexo

7.1 Gráficos estacionales

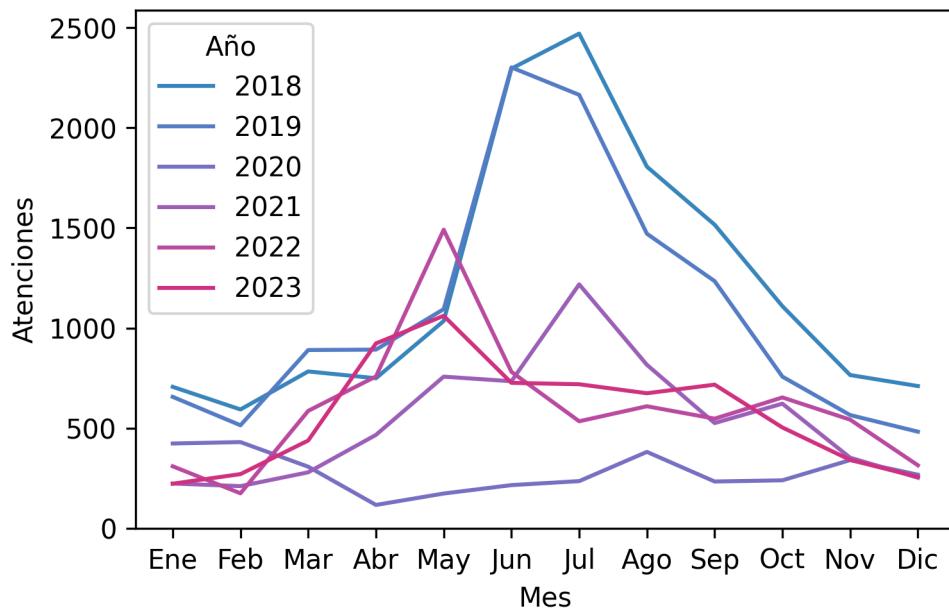


Figura 19: Atenciones por guardia mensuales por patologías respiratorias por año

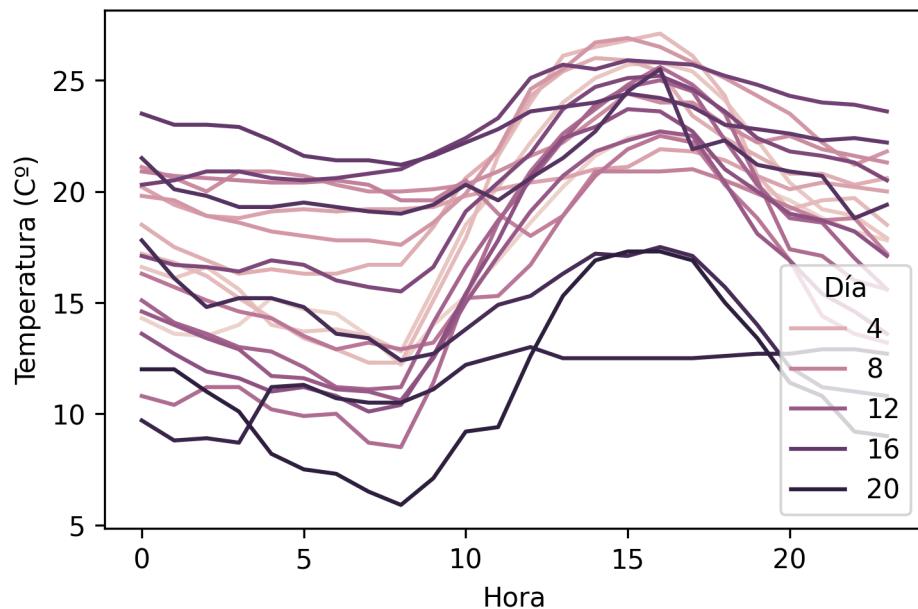


Figura 20: Atenciones por guardia mensuales por patologías respiratorias por año

7.2 Salidas de modelos arima

Atenciones modelo 1

SARIMAX Results

| | | | |
|------------------|----------------------------------|-------------------|----------|
| Dep. Variable: | y | No. Observations: | 72 |
| Model: | SARIMAX(0, 1, 1)x(0, 1, [], 12) | Log Likelihood | -425.795 |
| Date: | Thu, 03 Jul 2025 | AIC | 857.589 |
| Time: | 06:32:32 | BIC | 863.822 |
| Sample: | 0 - 72 | HQIC | 860.022 |
| Covariance Type: | opg | | |

| | coef | std err | z | P> z | [0.025 | 0.975] |
|-----------|-----------|----------|--------|-------|----------|----------|
| intercept | -0.3257 | 44.115 | -0.007 | 0.994 | -86.789 | 86.137 |
| ma.L1 | -0.0577 | 0.129 | -0.448 | 0.654 | -0.310 | 0.195 |
| sigma2 | 1.086e+05 | 1.56e+04 | 6.973 | 0.000 | 7.81e+04 | 1.39e+05 |

| | | | |
|-------------------------|------|-------------------|-------|
| Ljung-Box (L1) (Q): | 0.00 | Jarque-Bera (JB): | 16.54 |
| Prob(Q): | 0.97 | Prob(JB): | 0.00 |
| Heteroskedasticity (H): | 0.90 | Skew: | -0.80 |
| Prob(H) (two-sided): | 0.82 | Kurtosis: | 5.04 |

Warnings:

[1] Covariance matrix calculated using the outer product of gradients (complex-step).

Atenciones modelo 2

SARIMAX Results

| | | | |
|------------------|----------------------------------|-------------------|----------|
| Dep. Variable: | y | No. Observations: | 72 |
| Model: | SARIMAX(0, 1, 0)x(0, 1, [1], 12) | Log Likelihood | -423.031 |
| Date: | Thu, 03 Jul 2025 | AIC | 852.062 |
| Time: | 06:32:32 | BIC | 858.299 |
| Sample: | 0 - 72 | HQIC | 854.499 |
| Covariance Type: | opg | | |

| | coef | std err | z | P> z | [0.025 | 0.975] |
|-----------|---------|----------|--------|-------|----------|----------|
| intercept | 0.6941 | 33.473 | 0.021 | 0.983 | -64.911 | 66.299 |
| ma.S.L12 | -0.2919 | 0.107 | -2.720 | 0.007 | -0.502 | -0.082 |
| sigma2 | 9.8e+04 | 1.26e+04 | 7.789 | 0.000 | 7.33e+04 | 1.23e+05 |

| | | | |
|-------------------------|------|-------------------|-------|
| Ljung-Box (L1) (Q): | 0.13 | Jarque-Bera (JB): | 26.75 |
| Prob(Q): | 0.72 | Prob(JB): | 0.00 |
| Heteroskedasticity (H): | 0.79 | Skew: | -0.85 |
| Prob(H) (two-sided): | 0.61 | Kurtosis: | 5.82 |

Warnings:

[1] Covariance matrix calculated using the outer product of gradients (complex-step).

Atenciones modelo selección automática

```
SARIMAX Results
=====
Dep. Variable:                      y      No. Observations:             72
Model:                 SARIMAX(0, 1, 0)x(1, 0, 0, 12)   Log Likelihood       -504.749
Date:                   Thu, 03 Jul 2025      AIC                  1013.497
Time:                           06:32:32        BIC                  1018.023
Sample:                            0        HQIC                 1015.297
                                  - 72
Covariance Type:                opg
=====
```

| | coef | std err | z | P> z | [0.025 | 0.975] |
|----------|-----------|----------|-------|-------|----------|----------|
| ar.S.L12 | 0.5009 | 0.067 | 7.487 | 0.000 | 0.370 | 0.632 |
| sigma2 | 8.341e+04 | 1.11e+04 | 7.542 | 0.000 | 6.17e+04 | 1.05e+05 |

```
Ljung-Box (L1) (Q):                  0.41    Jarque-Bera (JB):           16.65
Prob(Q):                             0.52    Prob(JB):                     0.00
Heteroskedasticity (H):              0.65    Skew:                         0.60
Prob(H) (two-sided):                0.31    Kurtosis:                     5.05
=====
```

Warnings:

[1] Covariance matrix calculated using the outer product of gradients (complex-step).

Trabajadores modelo 1

```
SARIMAX Results
=====
Dep. Variable:                      y      No. Observations:             96
Model:                 SARIMAX(0, 1, 1)x(1, 1, [], 12)   Log Likelihood       -175.940
Date:                   Thu, 03 Jul 2025      AIC                  359.880
Time:                           06:32:32        BIC                  369.556
Sample:                            0        HQIC                 363.767
                                  - 96
Covariance Type:                opg
=====
```

| | coef | std err | z | P> z | [0.025 | 0.975] |
|-----------|---------|---------|--------|-------|--------|--------|
| intercept | -0.0347 | 0.314 | -0.111 | 0.912 | -0.649 | 0.580 |
| ma.L1 | 0.3242 | 0.110 | 2.958 | 0.003 | 0.109 | 0.539 |
| ar.S.L12 | -0.3532 | 0.081 | -4.337 | 0.000 | -0.513 | -0.194 |
| sigma2 | 3.9792 | 0.467 | 8.514 | 0.000 | 3.063 | 4.895 |

```
Ljung-Box (L1) (Q):                  0.12    Jarque-Bera (JB):           16.66
Prob(Q):                            0.73    Prob(JB):                     0.00
Heteroskedasticity (H):              1.28    Skew:                         -0.23
Prob(H) (two-sided):                0.52    Kurtosis:                     5.15
=====
```

Warnings:

[1] Covariance matrix calculated using the outer product of gradients (complex-step).

Trabajadores modelo selección automática

SARIMAX Results

| Dep. Variable: | y | No. Observations: | 96 | | | |
|-------------------------|--------------------------------|-------------------|----------|-------|--------|--------|
| Model: | SARIMAX(2, 0, 0)x(2, 1, 0, 12) | Log Likelihood | -173.349 | | | |
| Date: | Thu, 03 Jul 2025 | AIC | 356.697 | | | |
| Time: | 06:32:32 | BIC | 368.851 | | | |
| Sample: | 0 - 96 | HQIC | 361.583 | | | |
| Covariance Type: | opg | | | | | |
| | coef | std err | z | P> z | [0.025 | 0.975] |
| ar.L1 | 1.3724 | 0.090 | 15.176 | 0.000 | 1.195 | 1.550 |
| ar.L2 | -0.3937 | 0.081 | -4.890 | 0.000 | -0.552 | -0.236 |
| ar.S.L12 | -0.5199 | 0.097 | -5.373 | 0.000 | -0.710 | -0.330 |
| ar.S.L24 | -0.3363 | 0.119 | -2.827 | 0.005 | -0.570 | -0.103 |
| sigma2 | 3.3072 | 0.426 | 7.767 | 0.000 | 2.473 | 4.142 |
| Ljung-Box (L1) (Q): | 0.14 | Jarque-Bera (JB): | 13.80 | | | |
| Prob(Q): | 0.71 | Prob(JB): | 0.00 | | | |
| Heteroskedasticity (H): | 1.04 | Skew: | -0.32 | | | |
| Prob(H) (two-sided): | 0.92 | Kurtosis: | 4.88 | | | |

Warnings:

[1] Covariance matrix calculated using the outer product of gradients (complex-step).

Temperatura modelo selección automática

SARIMAX Results

| Dep. Variable: | y | No. Observations: | 480 | | | |
|------------------|--------------------------------|-------------------|----------|-------|--------|--------|
| Model: | SARIMAX(1, 1, 1)x(2, 0, 1, 24) | Log Likelihood | -524.526 | | | |
| Date: | Thu, 03 Jul 2025 | AIC | 1067.051 | | | |
| Time: | 06:32:32 | BIC | 1104.597 | | | |
| Sample: | 0 - 480 | HQIC | 1081.811 | | | |
| Covariance Type: | opg | | | | | |
| | coef | std err | z | P> z | [0.025 | 0.975] |
| intercept | -0.0009 | 0.006 | -0.135 | 0.892 | -0.013 | 0.012 |
| HUM | -0.0076 | 0.001 | -5.217 | 0.000 | -0.010 | -0.005 |
| PNM | -0.1338 | 0.061 | -2.210 | 0.027 | -0.253 | -0.015 |

| | | | | | | |
|-------------------------|---------|-------|-------------------|-------|--------|--------|
| ar.L1 | 0.2251 | 0.097 | 2.324 | 0.020 | 0.035 | 0.415 |
| ma.L1 | 0.1805 | 0.097 | 1.857 | 0.063 | -0.010 | 0.371 |
| ar.S.L24 | 0.9792 | 0.087 | 11.194 | 0.000 | 0.808 | 1.151 |
| ar.S.L48 | -0.0339 | 0.069 | -0.490 | 0.624 | -0.170 | 0.102 |
| ma.S.L24 | -0.7494 | 0.074 | -10.152 | 0.000 | -0.894 | -0.605 |
| sigma2 | 0.4960 | 0.025 | 19.884 | 0.000 | 0.447 | 0.545 |
| <hr/> | | | | | | |
| Ljung-Box (L1) (Q): | | 0.03 | Jarque-Bera (JB): | | 102.24 | |
| Prob(Q): | | 0.87 | Prob(JB): | | 0.00 | |
| Heteroskedasticity (H): | | 1.09 | Skew: | | -0.08 | |
| Prob(H) (two-sided): | | 0.59 | Kurtosis: | | 5.26 | |
| <hr/> | | | | | | |

Warnings:

[1] Covariance matrix calculated using the outer product of gradients (complex-step).

Temperatura modelo 1

| SARIMAX Results | | | | | | |
|-------------------------|--------------------------------|-------------------|-------------------|-------|--------|--------|
| Dep. Variable: | y | No. Observations: | 480 | | | |
| Model: | SARIMAX(1, 1, 1)x(1, 0, 1, 24) | Log Likelihood | -515.068 | | | |
| Date: | Thu, 03 Jul 2025 | AIC | 1046.136 | | | |
| Time: | 06:32:32 | BIC | 1079.509 | | | |
| Sample: | 0 - 480 | HQIC | 1059.255 | | | |
| Covariance Type: | opg | | | | | |
| <hr/> | | | | | | |
| | coef | std err | z | P> z | [0.025 | 0.975] |
| <hr/> | | | | | | |
| intercept | -4.422e-05 | 0.001 | -0.037 | 0.971 | -0.002 | 0.002 |
| HUM | -0.0072 | 0.001 | -5.035 | 0.000 | -0.010 | -0.004 |
| PNM | -0.2318 | 0.062 | -3.755 | 0.000 | -0.353 | -0.111 |
| ar.L1 | 0.5224 | 0.084 | 6.246 | 0.000 | 0.358 | 0.686 |
| ma.L1 | -0.1066 | 0.089 | -1.193 | 0.233 | -0.282 | 0.068 |
| ar.S.L24 | 0.9869 | 0.011 | 90.703 | 0.000 | 0.966 | 1.008 |
| ma.S.L24 | -0.8863 | 0.049 | -17.915 | 0.000 | -0.983 | -0.789 |
| sigma2 | 0.4682 | 0.024 | 19.347 | 0.000 | 0.421 | 0.516 |
| <hr/> | | | | | | |
| Ljung-Box (L1) (Q): | | 0.02 | Jarque-Bera (JB): | | 118.46 | |
| Prob(Q): | | 0.88 | Prob(JB): | | 0.00 | |
| Heteroskedasticity (H): | | 1.08 | Skew: | | -0.06 | |
| Prob(H) (two-sided): | | 0.64 | Kurtosis: | | 5.43 | |
| <hr/> | | | | | | |

Warnings:

[1] Covariance matrix calculated using the outer product of gradients (complex-step).

Temperatura modelo 2

SARIMAX Results

| Dep. Variable: | y | No. Observations: | 480 | | | |
|-------------------------|----------------------------------|-------------------|----------|-------|--------|--------|
| Model: | SARIMAX(1, 1, 0)x(2, 0, [1], 24) | Log Likelihood | -517.974 | | | |
| Date: | Thu, 03 Jul 2025 | AIC | 1051.948 | | | |
| Time: | 06:32:32 | BIC | 1085.322 | | | |
| Sample: | 0 - 480 | HQIC | 1065.068 | | | |
| Covariance Type: | opg | | | | | |
| | coef | std err | z | P> z | [0.025 | 0.975] |
| intercept | 0.0003 | 0.002 | 0.150 | 0.880 | -0.004 | 0.004 |
| HUM | -0.0077 | 0.001 | -5.315 | 0.000 | -0.011 | -0.005 |
| PNM | -0.1264 | 0.064 | -1.980 | 0.048 | -0.251 | -0.001 |
| ar.L1 | 0.4565 | 0.035 | 13.071 | 0.000 | 0.388 | 0.525 |
| ar.S.L24 | 1.0639 | 0.093 | 11.453 | 0.000 | 0.882 | 1.246 |
| ar.S.L48 | -0.0892 | 0.070 | -1.267 | 0.205 | -0.227 | 0.049 |
| ma.S.L24 | -0.8827 | 0.083 | -10.609 | 0.000 | -1.046 | -0.720 |
| sigma2 | 0.5096 | 0.028 | 18.016 | 0.000 | 0.454 | 0.565 |
| Ljung-Box (L1) (Q): | 0.44 | Jarque-Bera (JB): | 126.11 | | | |
| Prob(Q): | 0.51 | Prob(JB): | 0.00 | | | |
| Heteroskedasticity (H): | 1.03 | Skew: | -0.00 | | | |
| Prob(H) (two-sided): | 0.86 | Kurtosis: | 5.51 | | | |

Warnings:

[1] Covariance matrix calculated using the outer product of gradients (complex-step).

Temperatura modelo 3

SARIMAX Results

| Dep. Variable: | y | No. Observations: | 480 | | | |
|------------------|--------------------------------|-------------------|----------|-------|--------|--------|
| Model: | SARIMAX(1, 1, 0)x(1, 1, 0, 24) | Log Likelihood | -545.785 | | | |
| Date: | Thu, 03 Jul 2025 | AIC | 1103.570 | | | |
| Time: | 06:32:32 | BIC | 1128.292 | | | |
| Sample: | 0 - 480 | HQIC | 1113.309 | | | |
| Covariance Type: | opg | | | | | |
| | coef | std err | z | P> z | [0.025 | 0.975] |
| intercept | -0.0075 | 0.039 | -0.195 | 0.845 | -0.083 | 0.068 |
| HUM | -0.0071 | 0.002 | -4.181 | 0.000 | -0.010 | -0.004 |
| PNM | -0.0905 | 0.070 | -1.290 | 0.197 | -0.228 | 0.047 |
| ar.L1 | 0.3353 | 0.035 | 9.558 | 0.000 | 0.267 | 0.404 |
| ar.S.L24 | -0.4432 | 0.034 | -12.878 | 0.000 | -0.511 | -0.376 |
| sigma2 | 0.6372 | 0.032 | 20.082 | 0.000 | 0.575 | 0.699 |

| | | | |
|-------------------------|------|-------------------|-------|
| Ljung-Box (L1) (Q): | 0.04 | Jarque-Bera (JB): | 70.82 |
| Prob(Q): | 0.84 | Prob(JB): | 0.00 |
| Heteroskedasticity (H): | 1.29 | Skew: | -0.18 |
| Prob(H) (two-sided): | 0.12 | Kurtosis: | 4.90 |
| ===== | | | |

Warnings:

[1] Covariance matrix calculated using the outer product of gradients (complex-step).

7.3 Comprobación de supuestos de modelos arima

Modelo atenciones selección automática

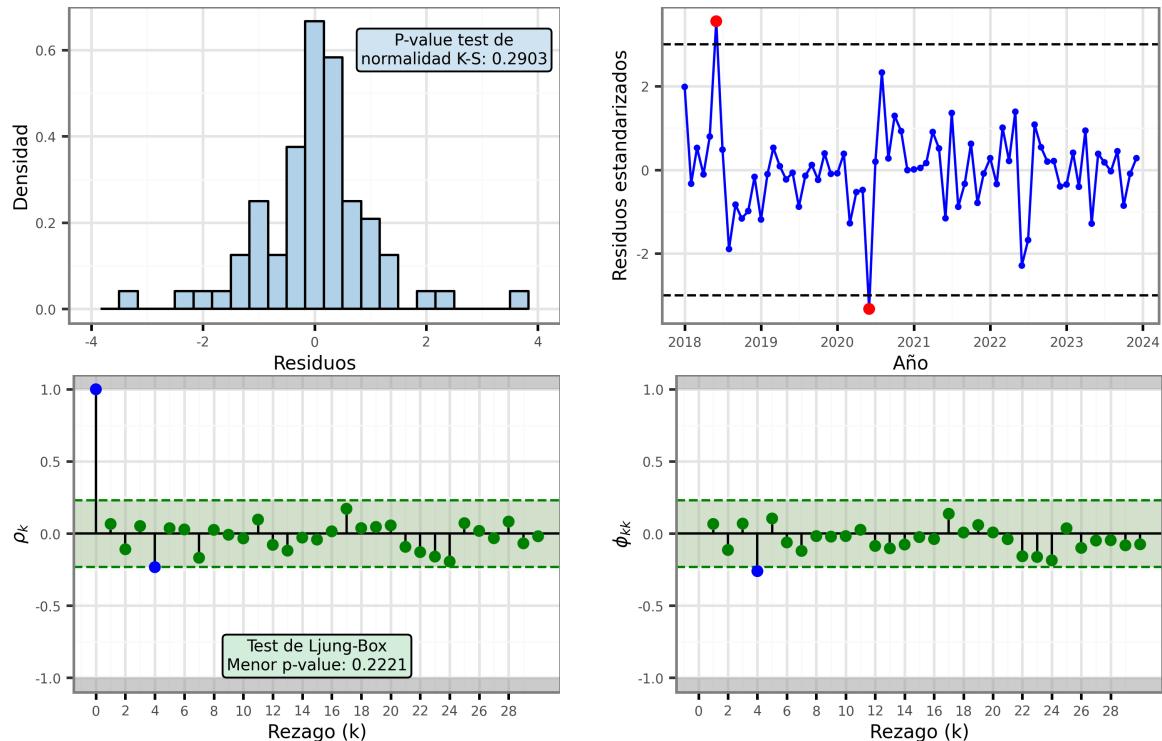


Figura 21: Comprobación de supuestos del modelo arima automático para la serie de atenciones

Modelo trabajadores 1

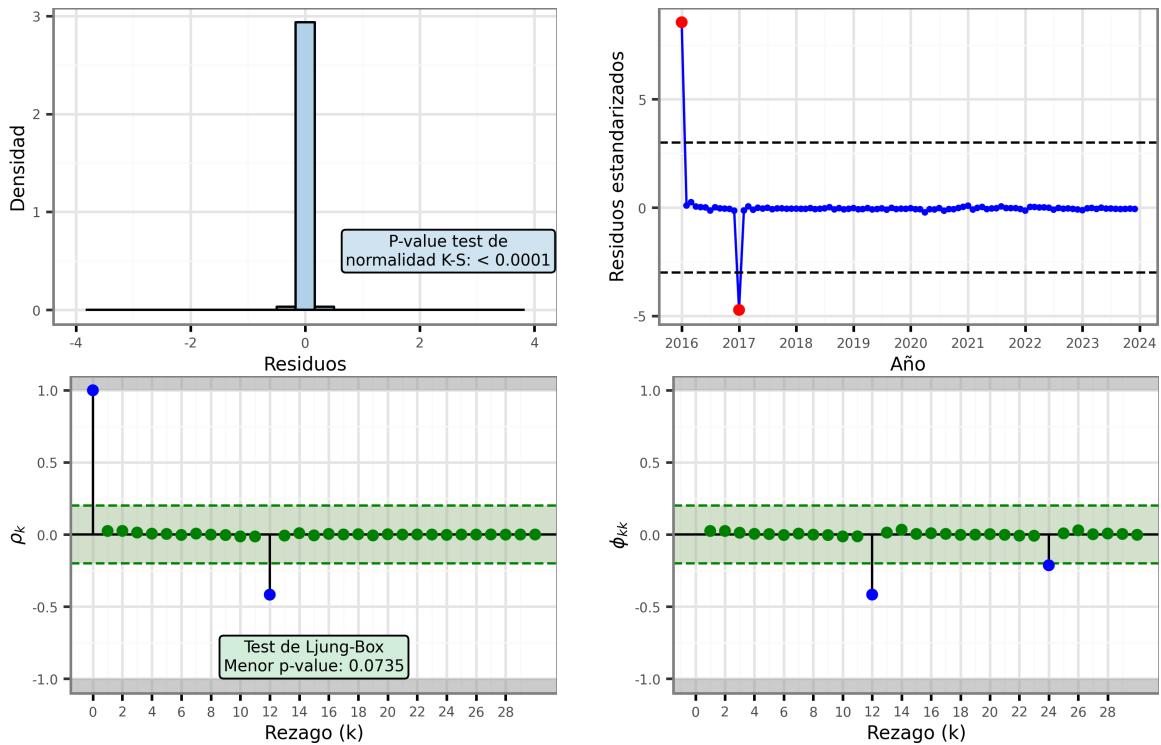


Figura 22: Comprobación de supuestos del modelo arima manual para la serie de trabajadores

Modelo trabajadores selección automática

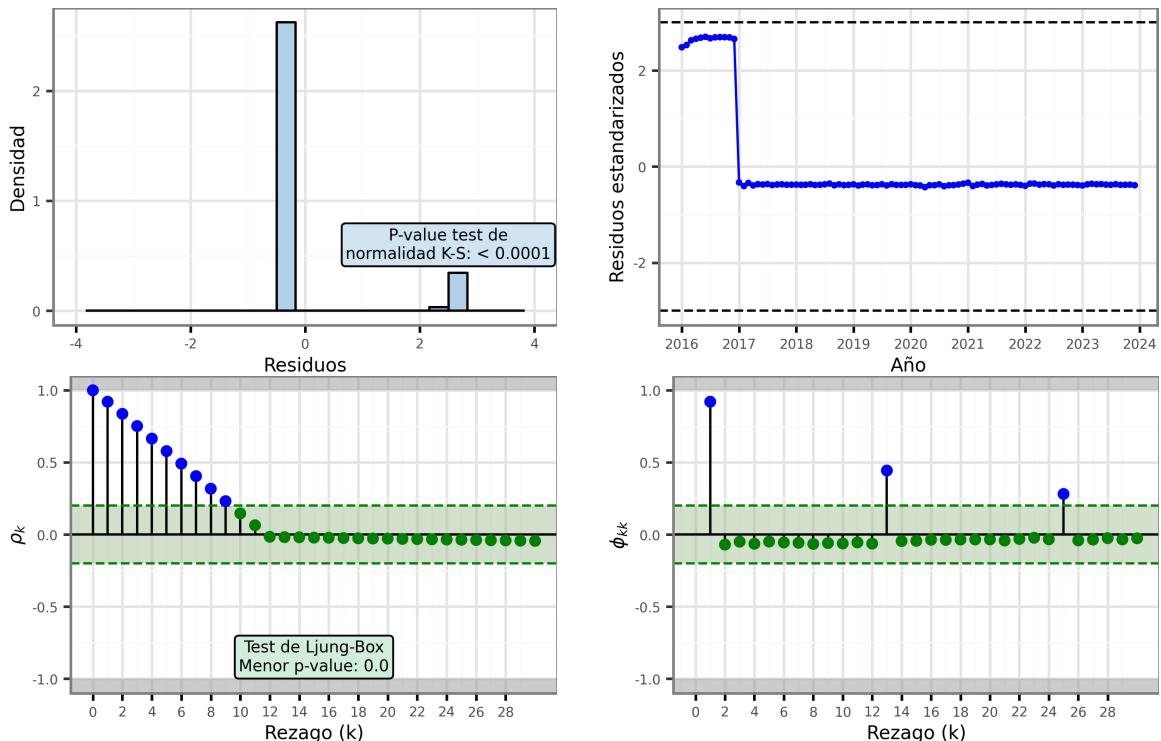


Figura 23: Comprobación de supuestos del modelo arima automático para la serie de trabajadores

Modelo Temperatura 3

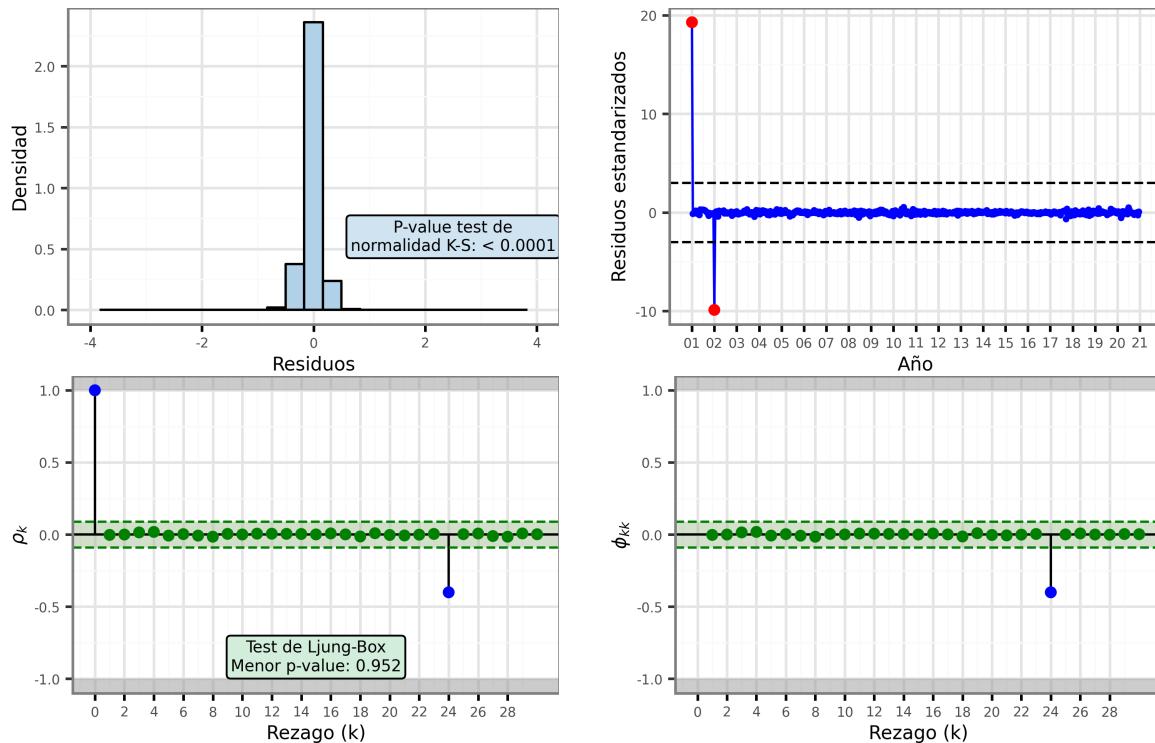


Figura 24: Comprobación de supuestos del tercer modelo arima manual para la serie de temperatura

NOTAS

- Series a utilizar: Estacionalidad y tendencia > trabajadores registrados en educación en el ámbito privado (tmb puede ser aalim/tabaco, servicios de seg/lim/inv,) (<https://www.argentina.gob.ar/trabajo/estadisticas/situacion-y-evolucion-del-trabajo-registrado>) ACTUALIZAR HAY DATOS NUEVOS Estacionalidad > Demanda mensual residencial del consumo de luz o algún emae estacional, ENFERMEDADES RESPIRATORIAS variable exógena > Temperatura + humedad + presión atmosférica (<https://www.smn.gob.ar/descarga-de-datos>)
- TO DO: corregir graficos residuos Volver a correr lstm (todos) Corregir pronósticos de chronos Corregir tablas pasar todos los graficos a plotnine Explicar como se eligen los modelos (mape), y como se registra el tiempo Explicar Aplicación, parámetros usados Revisar Arima revisar transformers revisar medidas de evaluación Volver a correr toda la aplicación y verificar resultados verificar luego de la aplicación cuales son las librerías que termino usando Verificar, actualizar y completar las descripciones de las funciones Actualizar los requirements

Agregar fuentes a los primeros graficos Autocorrelaciones en la descripciones de las series Puedo si utilizar los pronósticos de a poco

- Hacer dendograma de importancia de variables para los modelos de machine learning, timegpt (variables exógenas)
- Guardar las versiones de librerías, software y hardware en la que se corre el código
- Hacer equivalencia del documento en GitBook

- leer fpp3 que tiene unas cosas que pueden servir , tambien <https://otexts.com/fpppy/nbs/15-foundation-models.html#chronos>

Tomar el tiempo como el ultimo modelo PERO EXPLICARLO

nomenclatura: - conjunto de observaciones $z_1, \dots, z_t, \dots, z_n$ - pronostico $z_{n+1}, \dots, z_{n+l}, \dots, z_{n+h}$

Mejoras a la investigacion (en las que no me voy a centrar)

- Hacer que los modelos elijan el mejor ajuste con otras medidas del error en lugar de minimizar el mapeo, como el interval score
- explorar otras medidas del error
- implementar boosting y ensamblaje con otros métodos
- Explorar otras características para los métodos de ensamblaje
- medir el tiempo de ejecución de los algoritmos de otra manera
- Series con distintas periodicidades (semestrales, trimestrales, etc) y distintas unidades (diarias, anuales, horarias, etc)
- Explorar más sobre conformal predictions y demás opciones para crear intervalos de confianza
- usar ngboost para obtener pronósticos probabilísticos
- Validación cruzada temporal para el entrenamiento de los modelos

Preguntas nanda

- En realidad no es correcto tomar los pronósticos de a poco para comparar al corto y largo plazo porque los modelos se eligen según qué tan bien ajustan a cierto horizonte, si se cambia el horizonte esto podría modificarse