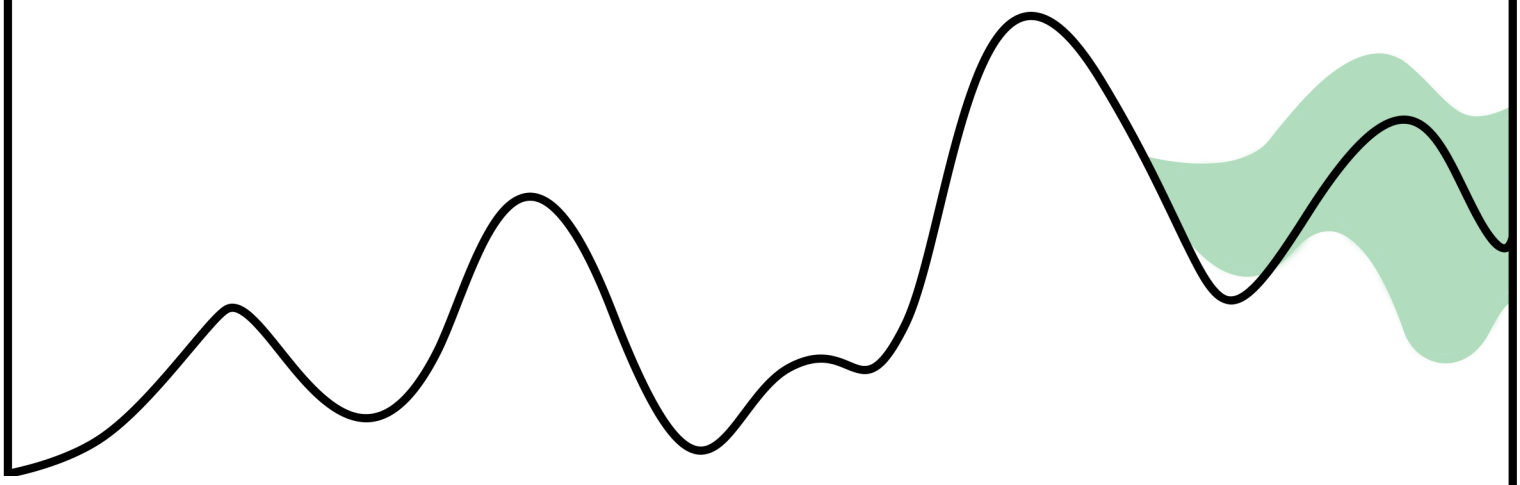
Comparación del desempeño de modelos estadísticos tradicionales, de aprendizaje automático y aprendizaje profundo para la predicción de series temporales

Anteproyecto de tesina



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

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 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. Es por esto que en esta tesina se 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 tales como TimeGPT y Chronos, buscando entender como funcionan.
- Definir y aplicar métricas de evaluación (MAPE, *Interval Score*) para comparar el rendimiento de todos los modelos bajo un mismo conjunto de datos.
- Reflexionar críticamente 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 se basa en la comparación del desempeño de distintos modelos de pronóstico aplicados a series temporales. Para ello, se seleccionarán modelos representativos de tres enfoques principales: estadísticos tradicionales, algoritmos de aprendizaje automático (machine learning) y modelos de aprendizaje profundo. El análisis se estructura en tres componentes fundamentales: la caracterización de los modelos, la implementación práctica sobre series con distintas características y la evaluación comparativa mediante métricas cuantitativas.

3.1 Conceptos básicos de series de tiempo

Se denomina serie de tiempo a un conjunto de observaciones 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 sobre su evolución futura.

Dado que las series temporales pueden exhibir diversos patrones subyacentes, resulta útil descomponerlas en componentes diferenciadas, 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 suelen deberse a eventos impredecibles o factores exógenos, y se asume que siguen un comportamiento aleatorio.

3.2 Modelos estadísticos tradicionales

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 $ARIMA$ (*AutoRegresive Integrated Moving Average*) son unos de los modelos de pronostico 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 generaliza en el sentido de los modelos diferenciados (I), en los que se resta a cada observacion los d-ésimo valores anteriores para estacionarizar en media.

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

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

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.
- $\psi_p(B) = 1 - \psi_1 B - \psi_2 B_2 - ... - \psi_p B^p$ es el polinomio característico de la componente AR y $\theta_q(B) = 1 - \theta_1 B - \theta_2 B^2 - ... - \theta_p B^q$ de la componente MA. Si las raíces de los polinomios característicos caen fuera del círculo unitario, entonces un proceso AR se puede esctribir 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, estan incorrelacionados y se distribuyen normal, con media y variancia constante.
- 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 apredizaje automático

El aprendizaje automático (del inglés *machine learning*) se define como una rama de la inteligencia artificial enfocada a permitir que las computadoras y máquinas imiten la forma en que los humanos aprenden, para realizar tareas de forma autónoma y mejorar la eficiencia y eficacia a través de la experiencia y la exposición a mas información. Si bien los métodos que se presentan no fueron diseñados especificamente para el análisis de datos temporales, como los modelos tradicionales o aquellos que utilizan aprendizaje profundo que se mencionarán más adelante, si probaron ser útiles a lo largo del tiempo y a través de distintas pruebas.

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 cúales determinan que tanto se ajusta el modelo a los datos con los cúales se entrena. Lógicamente árboles más profundos y con más hojas producirán un sobreajuste, y es esta la mayor desventaja que tienen este tipo de modelos.

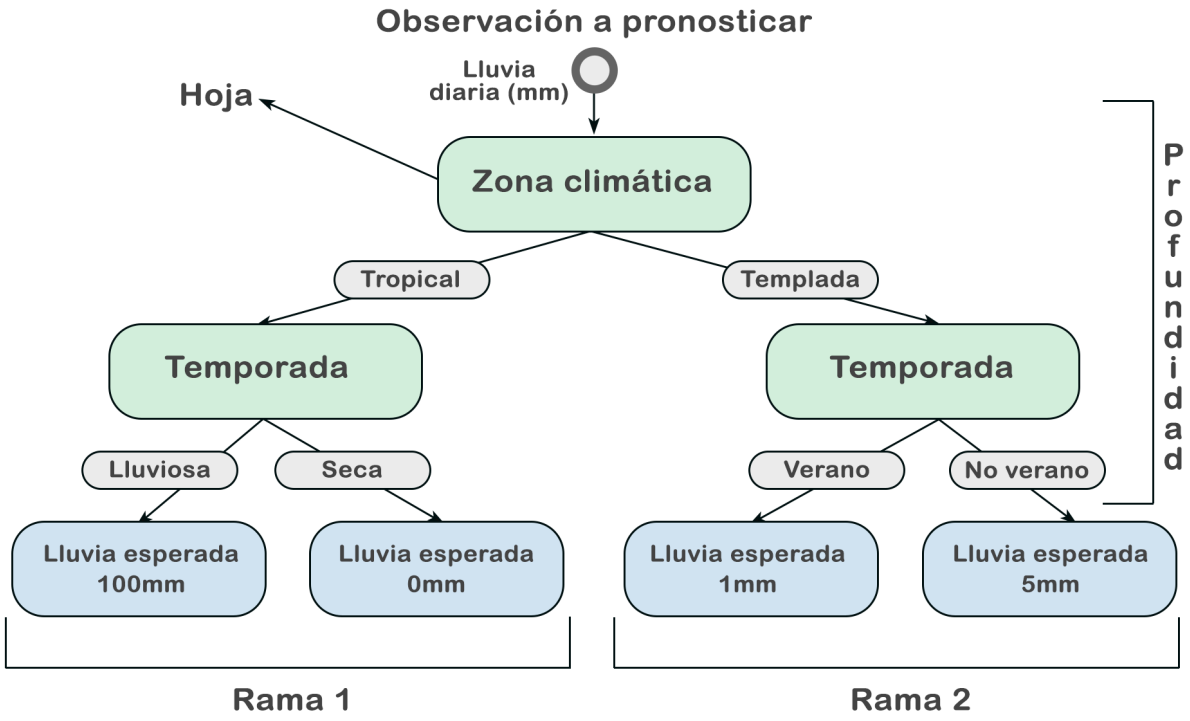


Figura 1: Ejemplo de árbol de decisión

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 cúales 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 el método del gradiente. **(prfundizar sobre el gradiente???)**

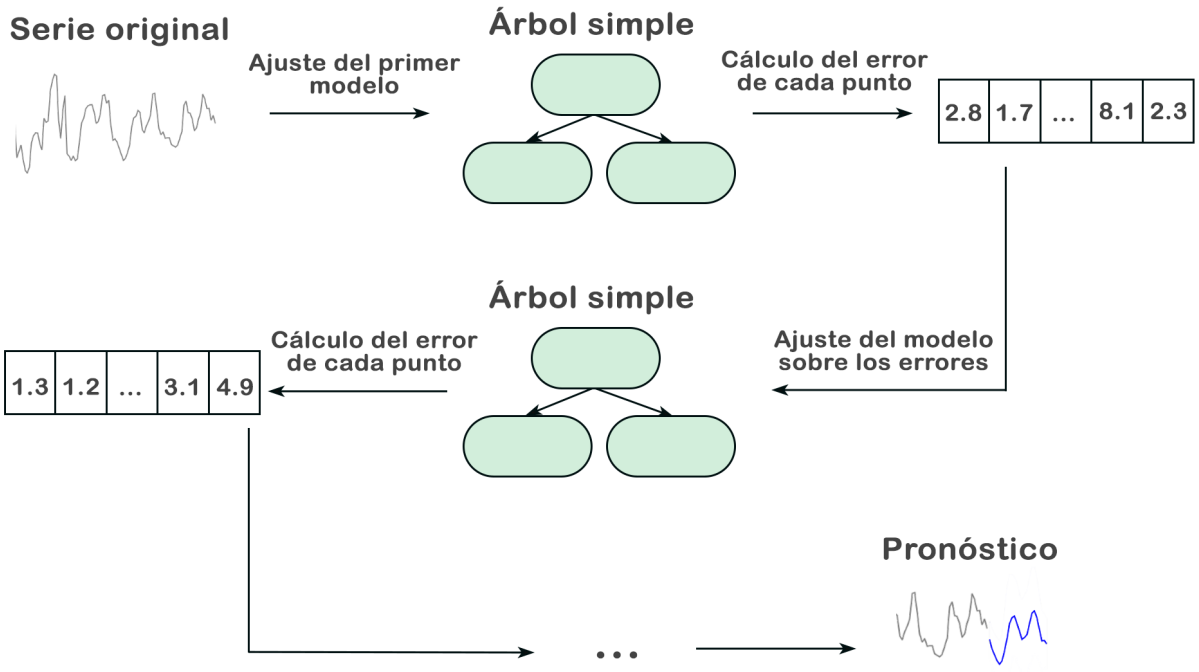


Figura 2: Proceso de ensamblado

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 XGBoost y LightGBM.

3.3.2 Diferencias entre XGBoost y LightGBM

Las diferencias entre XGBoost y LightGBM radican en la forma en 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 cuáles 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. LigthGBM, 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.

3.4 Modelos de apredizaje 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 el conjunto de algoritmos que se menciona están las redes neuronales, las cúales son un tipo de modelo que toma decisiones de la misma forma que las personas, usando procesos que simulan la forma biológica en la que trabajan las neuronas para identificar fenómenos, evaluar opciones y llegar a conclusiones. Una red neuronal funciona con varias neuronas y pesos. 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.

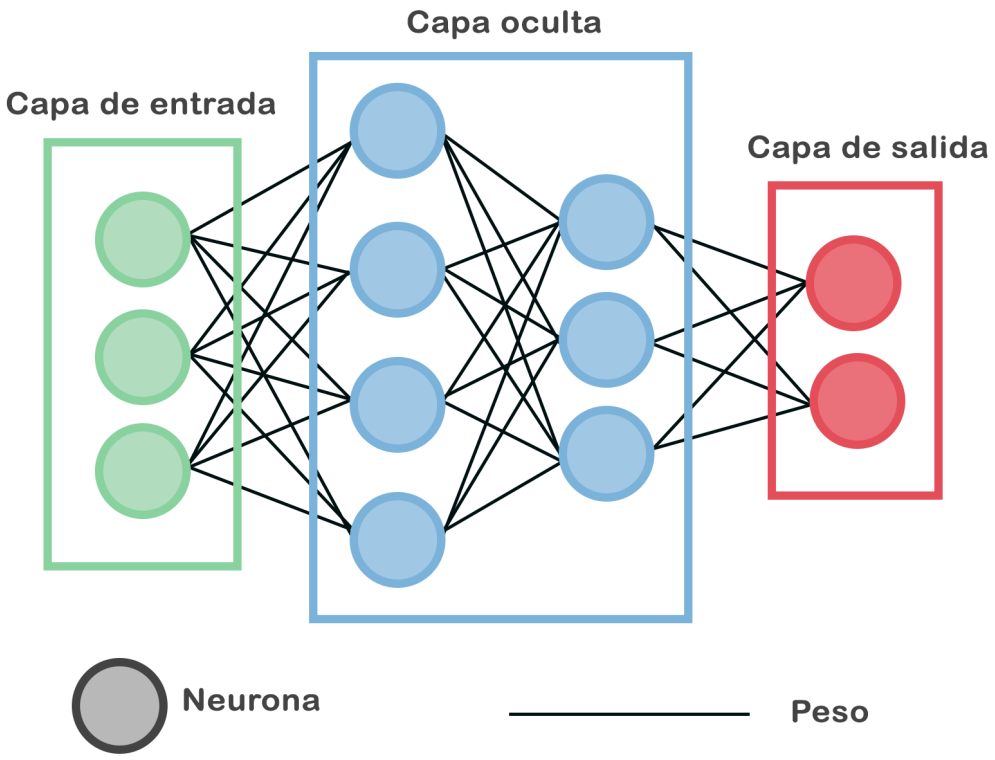


Figura 3: Red neuronal completamente conectada

Hay 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), 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.

Otro tipo de modelo de aprendizaje profundo son los *transformer models* (modelos transformadores), los cúales son significativamente más eficientes al entrenar y realizar inferencias que las CNNs y las RNNs gracias al uso de mecanismos de atención, presentados en la publicación '*Attention is all you need*' de Google.

3.4.1 *Long Short Term Memory (LSTM)*

Lo que caracterizan a las redes neuronales recurrentes son los bucles de retroalimentación que se presentan en la Figura 4. 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.

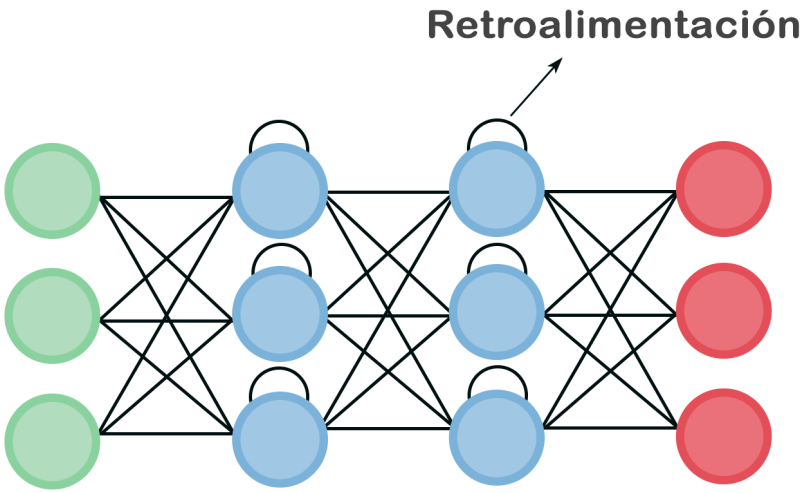


Figura 4: Ejemplo de RNN

Las redes neuronales con memoria a corto y largo plazo (LSTM) son un tipo de RNN que, para evitar que los gradientes se vayan a cero o al infinito muy rápidamente al actualizar los parámetros, usa un algoritmo logístico de 3 puertas:

- Puerta de guardado: Guarda la información relevante del estado actual de la red neuronal.
- Puerta de entrada: Guarda la información que debe ser actualizada en la red neuronal.
- Puerta de salida: Combina la información de las puertas de guardado y entrada, y decide que tanto de esa información usar para seguir utilizando en la red neuronal.

3.4.2 TimeGPT

TimeGPT es un modelo fundacional preentrenado para el pronóstico de series de tiempo que puede producir predicciones en diversas áreas y aplicaciones con gran precisión y sin entrenamiento adicional. Los modelos preentrenados constituyen una gran innovación, lo que mejora la accesibilidad, precisión, eficiencia computacional y velocidad del pronóstico.

TimeGPT es un modelo de tipo transformer con mecanismos de autoatención que no es de código abierto. La autoatención captura dependencias y relaciónes en la secuencias de valores que se alimentan al modelo, logrando poner en contexto a cada observación.

La arquitectura del modelo consiste en una estructura con codificador y decodificador de múltiples capas, cada una con conexiones residuales y normalización de capas. Por último, contiene una capa lineal que mapea la salida del decodificador a la dimensión del pronóstico.

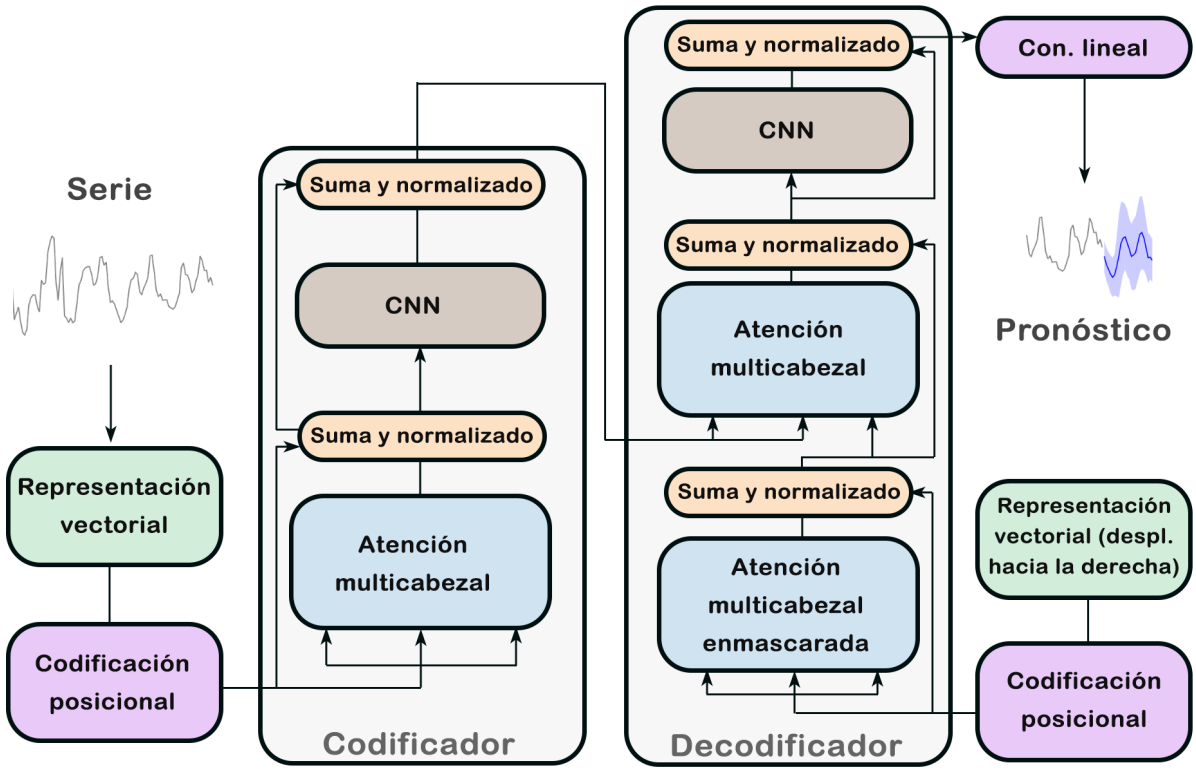


Figura 5: Diagrama de la estructura del modelo

Paso a paso del funcionamiento del modelo

Codificador

1. *Input embedding*: Cada *token* (observaciones en el caso de TimeGPT) es transformado en un vector (vector de entrada) con muchas dimensiones ($\vec{E}$). Las dimensiones corresponden a diferentes caracteristicas que el modelo definió en el pre-entrenado con una numerosa cantidad de parámetros.
2. Codificación posicional: Típicamente se introduce como un set de valores adicionales o vectores que 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. *Multi-Head Attention* (Atención multi-cabezal): 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 el token está y manipula los valores del vector que lo representa para añadir esta información contextual.

La verificación del contexto funciona gracias a una matriz la cúal se llamará *Query* (W_Q) que examina ciertas características, y es multiplicada por el vector de entrada ($\vec{E}$) resultando en un vector de consultas ($\vec{Q}$) para cada token. Una matriz de claves (*Key matrix*) (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 vectores[1] ($\sqrt{d_k}$) y se normaliza con *softmax*[2] por columna[3] ($\vec{S}$), valores más altos indican que un token (de las columnas) esta siendo influenciado por el comportamiento de otro token (de las filas).

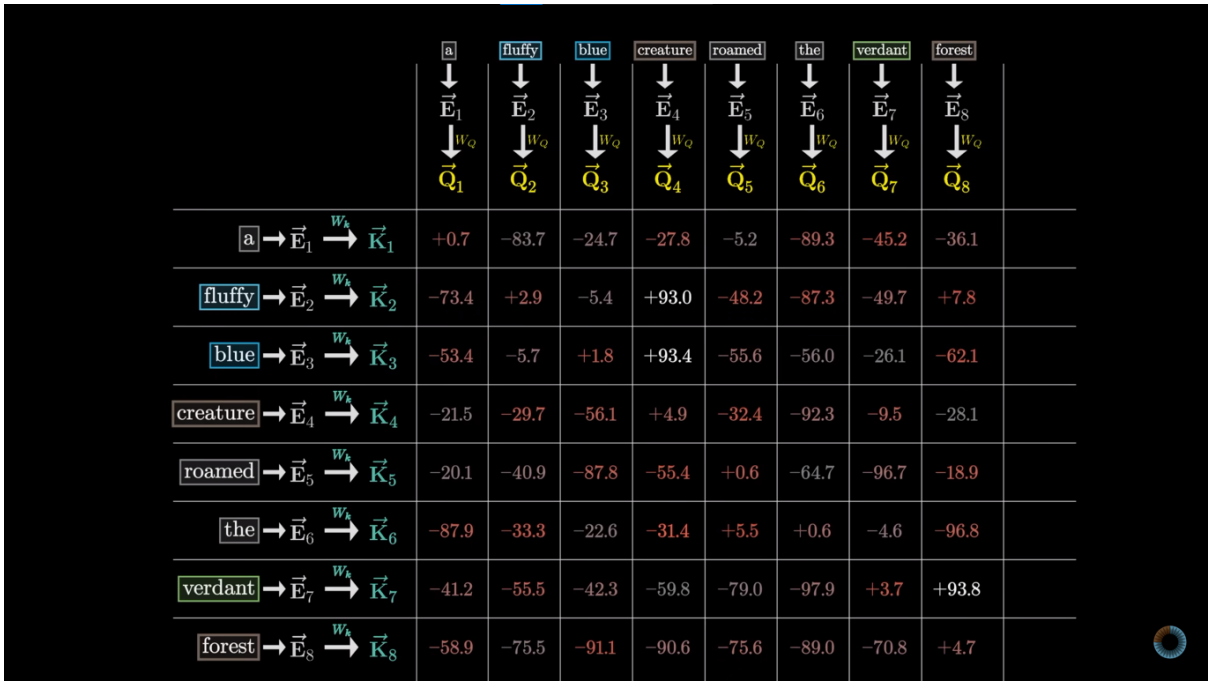


Figura 6: (before applying softmax) for example: Fluffy and Blue contribute to creature

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 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.

[1] Es 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] $softmax(x) = \frac{e^{x_i/t}}{\sum_j e^{x_j/t}}$

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

$$\Delta\vec{E}_j = \sum_i S_j \cdot \vec{V}_i$$

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.

$$\vec{E} \Leftarrow \vec{E} + \sum_h \Delta\vec{E}_h$$

En resumen, y haciendo referencia la publicación Attention is all you need:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

Es importante notar que todas las matrices Q, K y V son pre-entrenadas.

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 *CNN*s 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 hacerce 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.

3.4.3 Chronos

Chronos es una familia de modelos fundacionales preentrenados para series de tiempo basados en arquitecturas de modelos de lenguaje. Chronos trabaja transformando la serie de tiempo en una secuencia de *tokens* por medio de escalado y cuantificación, para luego entrenar un modelo de lenguaje con estos, usando entropía cruzada. Una vez entrenado el modelo, se generan múltiples pronósticos probabilísticos a partir del contexto historico.

3.5 Métricas de evaluación

Para comparar el rendimiento de los modelos, se utilizarán 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. Las comparaciones permitirán evaluar precisión, robustez y eficiencia de cada enfoque.

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

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

$$MSE = \frac{1}{L}\sum_{l=1}^{L} e_l^2$$

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

$$MAE = \frac{1}{L}\sum_{l=1}^{L} |e_l|$$

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

$$MAPE = (\frac{1}{L}\sum_{l=1}^{L}|\frac{e_l}{Z_{n+l}}|) \cdot 100\%$$

Aquel modelo que tenga el menor valor en estas medidas será considerado el mejor.

El problema de estos errores es que solo tienen en cuenta la estimación puntual, y por lo general es buena idea trabajar con pronosticos probabilisticos para cuantificar la incertidumbre de los valores futuros de la variable. Gneiting & 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, llamandola *Interval Score*:

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

Donde:

$$W_l = IS_l - II_l$$

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

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.

3.6 Técnicas para la estimación de parámetros

https://www.researchgate.net/publication/330742498_Evaluation_of_statistical_and_machine_learning_m
of-the-art_and_the_best_conditions_for_the_use_of_each_model

3.6.1 Holdout

3.6.2 Box-Jenkins

4. Aplicación

La aplicación empírica del trabajo se desarrollará en varias etapas, con el objetivo de implementar, ajustar y comparar los modelos previamente detallados sobre un conjunto de series temporales seleccionadas, usando el software de programación de código abierto Python.

4.1 Etapa 1: Selección y preparación de los datos

Se trabajará con series temporales reales obtenidas de bases de datos públicas y confiables. Las series seleccionadas presentarán diferentes características, incluyendo, por ejemplo, estacionalidad, tendencia o diferentes periodicidades, con el fin de evaluar el comportamiento de los modelos bajo condiciones variadas.

Una vez seleccionadas, las series serán sometidas a un proceso de limpieza y transformación, que incluirá la conversión de fechas a formatos estándar, el tratamiento de valores faltantes (si los hubiera) y la división de los datos en subconjuntos de entrenamiento y validación. Se explorará también la opción de aplicar una normalización o estandarización de las series para facilitar el entrenamiento de los modelos de aprendizaje profundo.

4.2 Etapa 2: Implementación de los modelos

En esta etapa se implementarán los distintos modelos de pronóstico contemplados en el trabajo, agrupados en función de su enfoque metodológico: estadístico tradicional, aprendizaje automático y modelos de aprendizaje profundo.

Los modelos estadísticos clásicos, como ARIMA y SARIMA, serán ajustados utilizando la librería `pmdarima`, basando la selección de parámetros en el criterio de información de Akaike (AIC) y usando validación cruzada temporal.

En el caso de los modelos de aprendizaje automático, se emplearán bosques aleatorios con *boosting* haciendo uso de los algoritmos XGBoost y LightGBM. Estos modelos serán implementados con las librerías `xgboost` y `lightgbm`.

Para los modelos de aprendizaje profundo, se desarrollarán redes neuronales del tipo LSTM, con diferentes configuraciones de capas y funciones de activación, usando en este caso la librería `neuralforecast`.

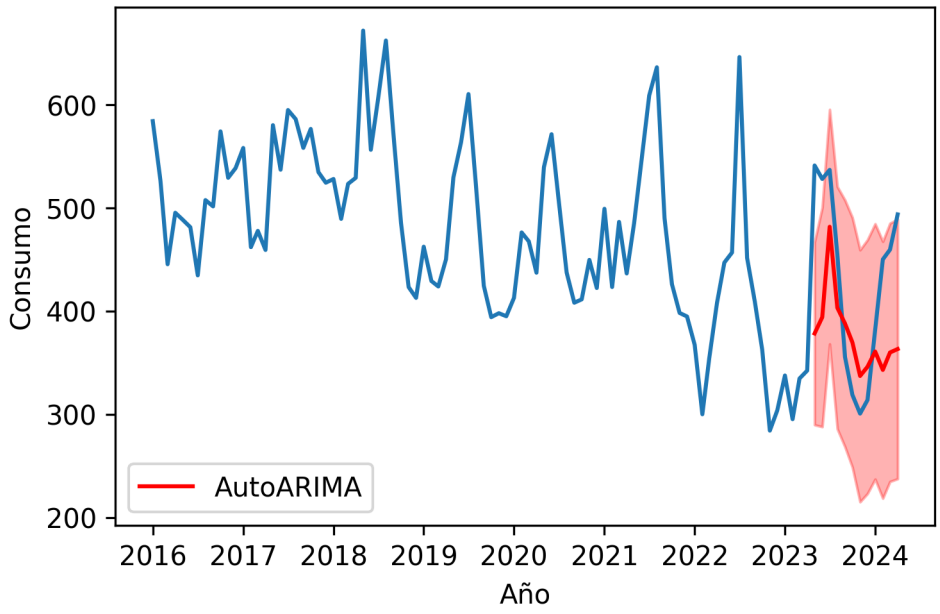
Por último, se explorarán dos modelos fundacionales preentrenados. El primero es TimeGPT, que será accedido a través de la API provista por Nixtla y haciendo uso de la librería `nixtla`. El segundo es Chronos, una familia de modelos preentrenados desarrollada por *Amazon Web Services* cuya implementación será llevada a cabo con la librería `autogluon`.

4.3 Etapa 3: Evaluación y comparación

Cada modelo será evaluado en función de su desempeño predictivo, utilizando métricas como MAPE e *Interval Score*, junto con observaciones sobre el tiempo de cómputo, la facilidad de implementación e interpretabilidad de cada uno. Los resultados se sistematizarán en tablas comparativas y visualizaciones gráficas, acompañadas de un análisis crítico.

Métodos Tradicionales

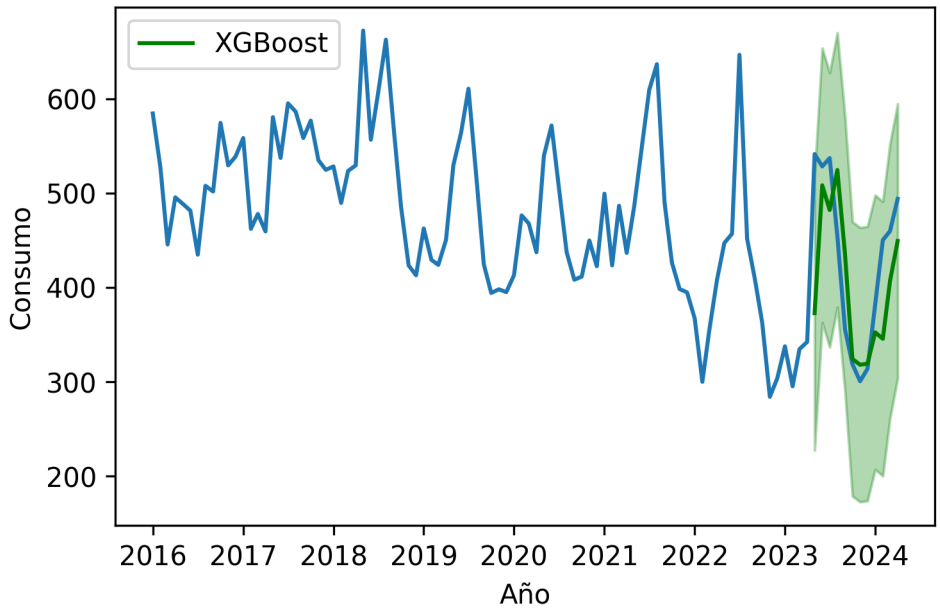
ARIMA



Métodos de Machine Learning

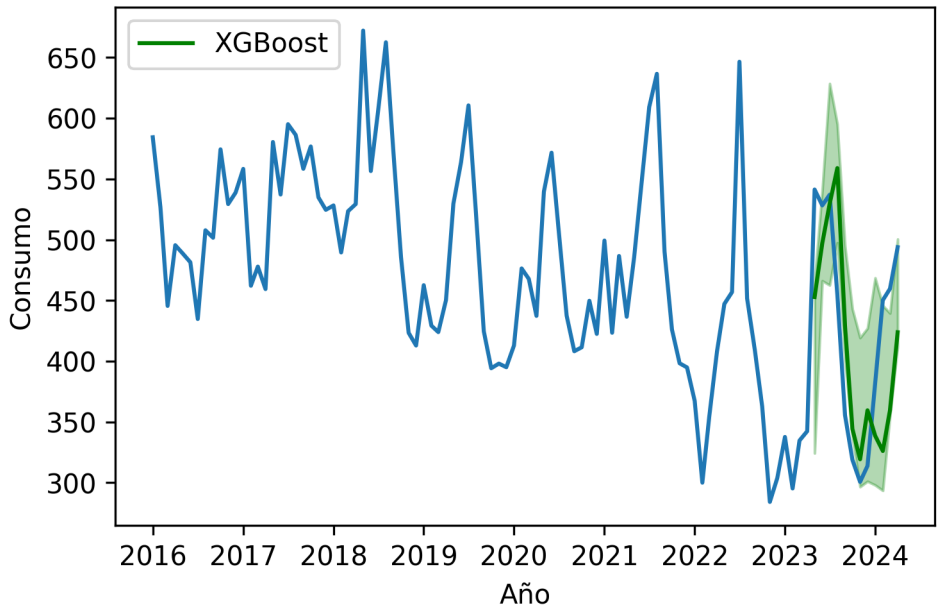
XGBoost

los intervalos de estos pronosticos no son probabilisticos, estoy usando conformal predictions



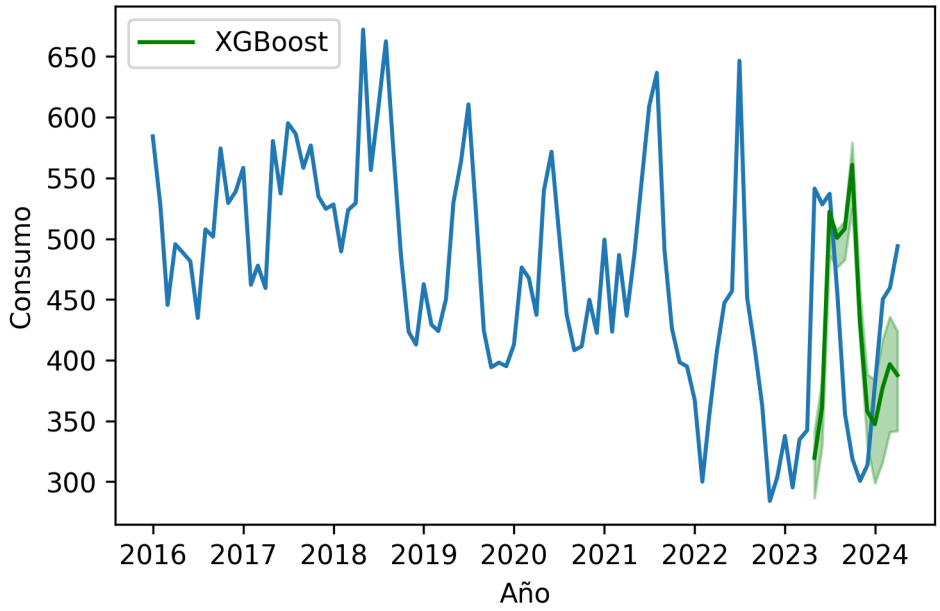
LightGBM

importante, los quantiles de estos pronosticos son independientes

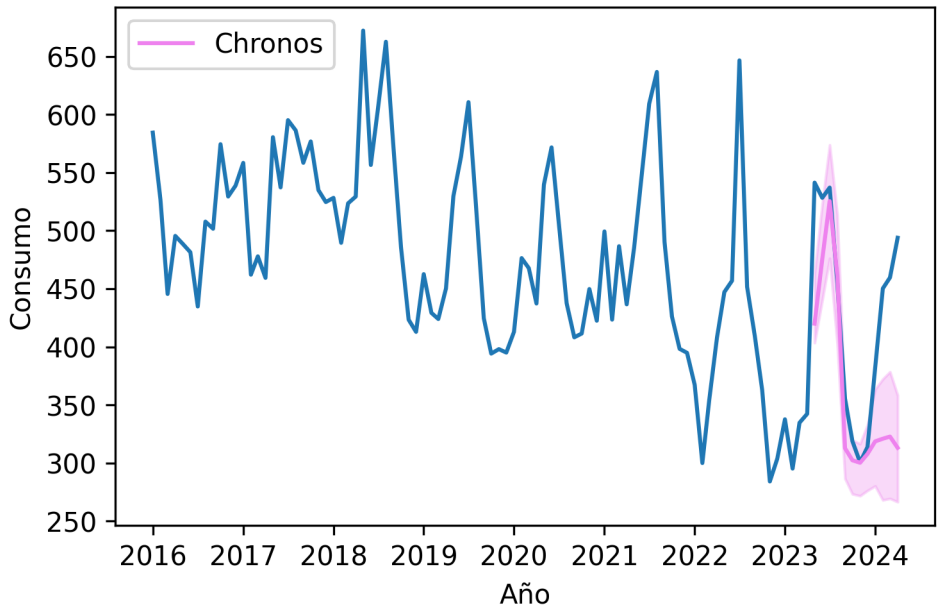


Deep Learning

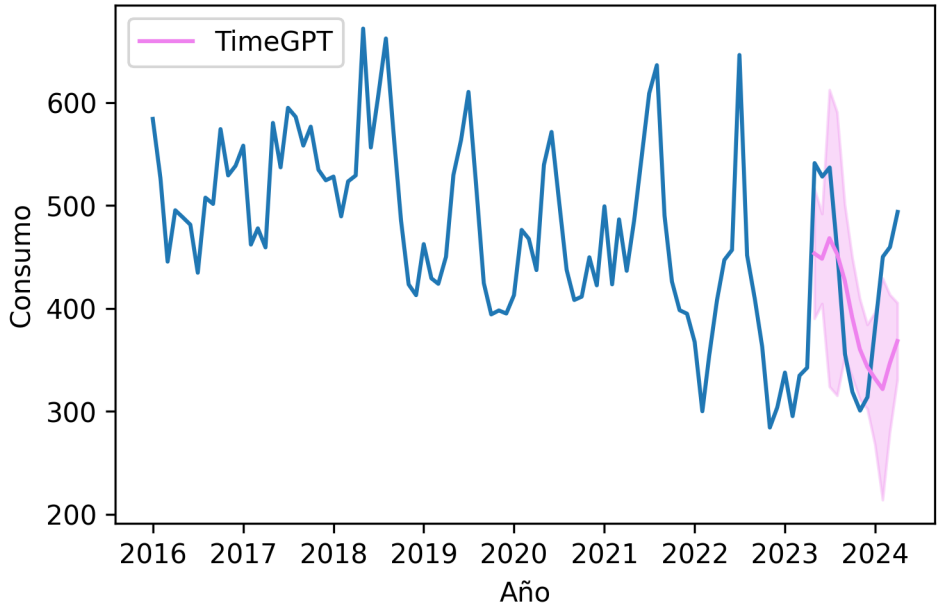
LSTM



Chronos



TimeGPT



Comparaciones

	Modelo	MAPE	Interval Score	Tiempo
0	AutoARIMA	0.168367	324.286537	15.071169
1	XGBoost	0.119703	310.010545	0.051024
2	LightGBM	0.142309	290.805218	2.732818
3	LSTM	0.268863	1355.535530	34.157234
4	TimeGPT	0.173721	349.266613	2.896710
5	Chronos	0.139081	415.731862	624.001588

5. Conclusiones

- Resume las principales conclusiones del estudio.
- Destaca las contribuciones del trabajo y su relevancia para el campo de la estadística y el análisis de datos.
- Fortalezas y limitaciones de TimeGPT en comparación con otros modelos.
- Proporciona recomendaciones finales y reflexiones sobre posibles direcciones futuras de investigación.

6. Bibliografía

Ansari et al. (4 de noviembre de 2024). Chronos: Learning the Language of Time Series. Transactions on Machine Learning Research. https://arxiv.org/abs/2403.07815

Awan, A. A. (2 de septiembre de 2024). Time Series Forecasting With TimeGPT. Datacamp. https://www.datacamp.com/tutorial/time-series-forecasting-with-time-gpt

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.

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-llms-intuition

Silberstein, E. (7 de noviembre de 2024). Tracing the Transformer in Diagrams. Medium. https://medium.com/data-science/tracing-the-transformer-in-diagrams-95dbeb68160c

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

7. Anexo

Incluye cualquier material adicional, como código fuente, datos adicionales, detalles sobre la implementación en R o Python, entre otros.

NOTAS

- Guardar las versiones de librerias, software y hardwate en la que se corre el codigo
- Hacer equivalencia del documento en GitBook
- Datos: Graciela N. Klekailo Facultad de Ciencias Agrarias, Direccion general de estadistica municipalidad
- Investigar mas sobre medidas de error (weighted quantile loss)
- Toda la modelización pasarla a .py y no .qmd
- leer fpp3 que tiene unas cosas que pueden servir
- incluir varios tipos de series? con estacionalidad, con tendencia, con ambas, con variables exogenas?
- Series a utilizar: Estacionalidad y tendencia > trabajadores registrados en educacion en el ambito privado Estacionalidad > Demanda mensual residencial del consumo de luz o algun emae estacional variable exogena >
- Orden: Redes neuronales LSTM Modelos transformer y fundacionales Chronos Revisar TimeGPT Revisar Arima
- Hacer dendograma de importancia de variables para los modelos de machine learning

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 mape, como el interval score
- explorar otras medidas del error
- implementar boosting y ensamblaje con otros metodos
- medir el tiempo de ejecucion de los algoritmos de otra manera