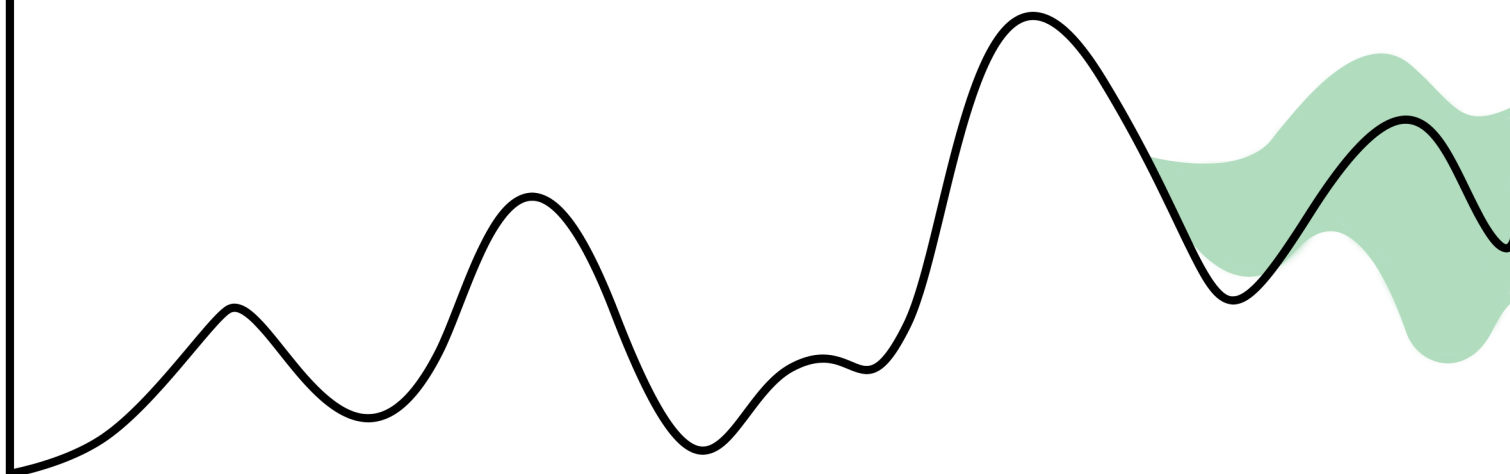


Comparación del desempeño de modelos estadísticos tradicionales, de aprendizaje automático y aprendizaje profundo para la predicción de series temporales

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	8
3.4 Modelos de aprendizaje profundo	9
3.4.1 Introducción a redes neuronales	9
3.4.2 <i>Long Short Term Memory (LSTM)</i>	10
3.4.2 Modelos transformadores	13
3.4.3 Diferencias entre TimeGPT y Chronos	17
3.5 Métricas de evaluación	18
3.6 Técnicas para la estimación de parámetros	19
4. Aplicación	20
4.1 Series a utilizar	20
Métodos Tradicionales	20
Métodos de Machine Learning	21
Deep Learning	22
Comparaciones	23
5. Conclusiones	24
6. Bibliografía	25
7. Anexo	27
NOTAS	27
Mejoras a la investigacion (en las que no me voy a centrar)	27

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 $\{z_1, z_2, \dots, z_t, \dots, z_n\}$ 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 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* (*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 generaliza en el sentido de los modelos diferenciados (I), en los que se resta a cada observación 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.
- $\psi_p(B) = 1 - \psi_1 B - \psi_2 B^2 - \dots - \psi_p B^p$ 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 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 aprendizaje 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 específicamente 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 cuáles determinan que tanto se ajusta el modelo a los datos con los cuáles 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.

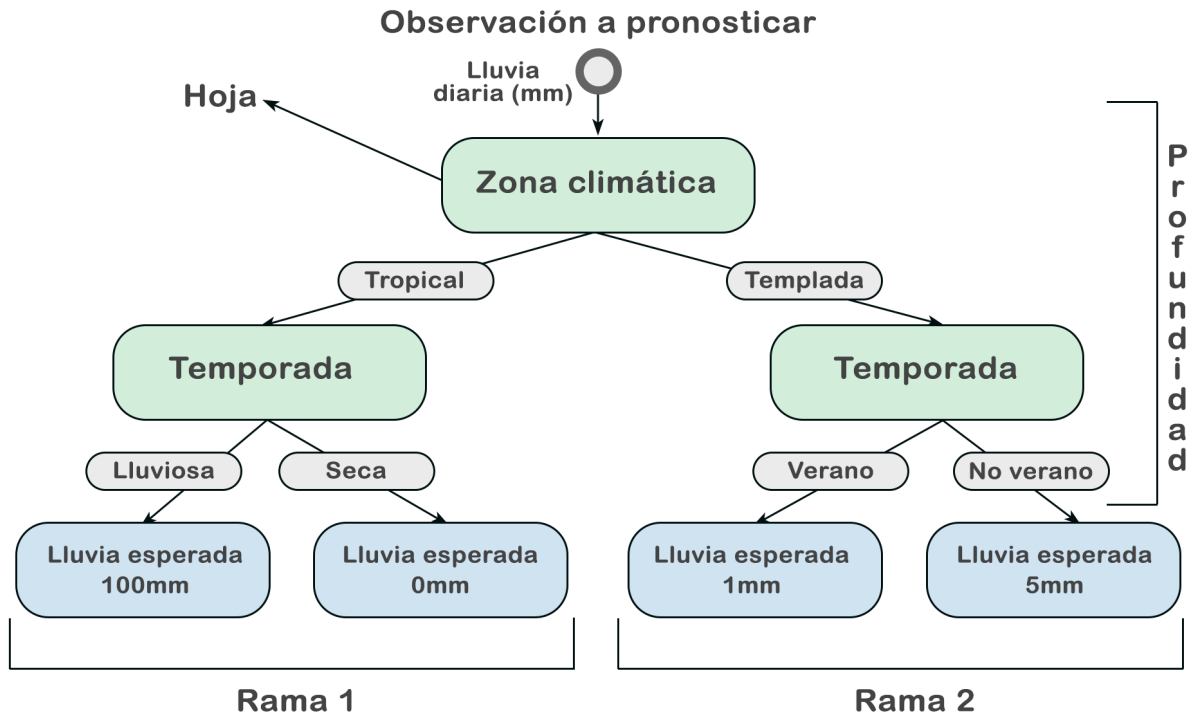


Figura 1: Ejemplo de árbol de decisión

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 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.

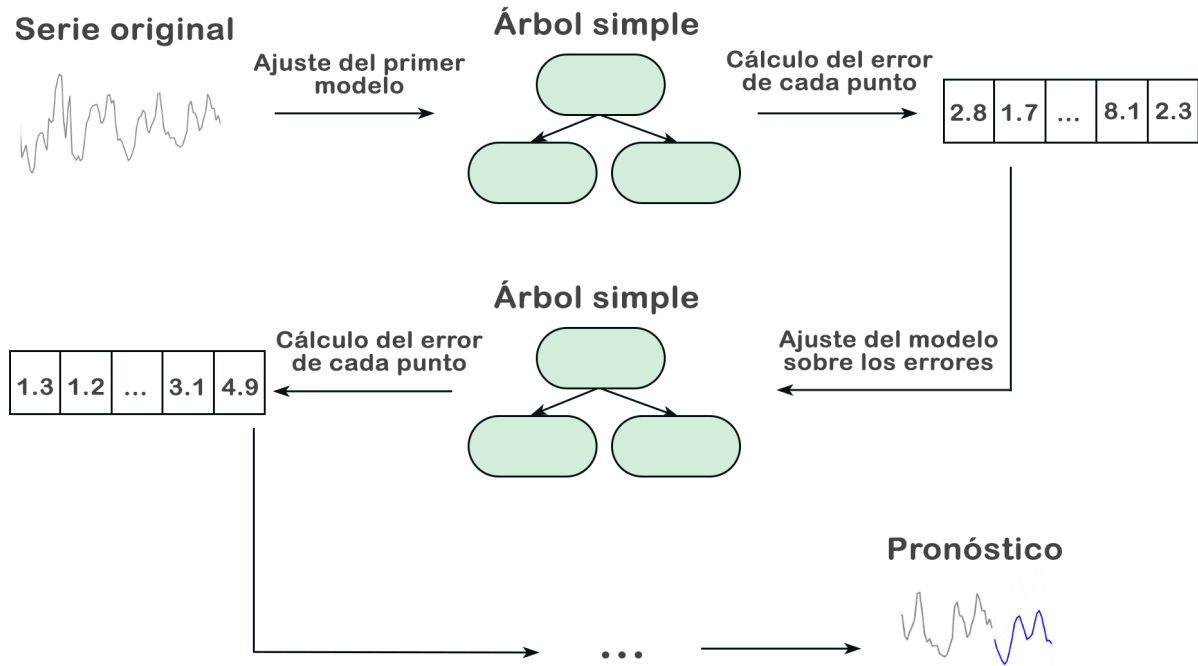


Figura 2: Proceso de ensamblado

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.

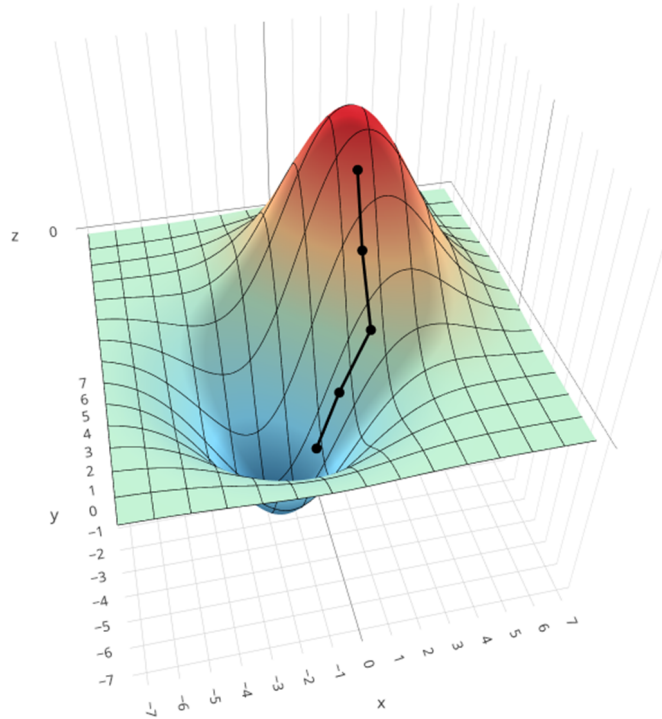


Figura 3: Ejemplo del descenso del gradiente en una función de pérdida

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 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. 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.

3.4 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 el conjunto de algoritmos que se menciona están las redes neuronales, las cuáles 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.

3.4.1 Introducción a redes neuronales

Una red neuronal esta 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.

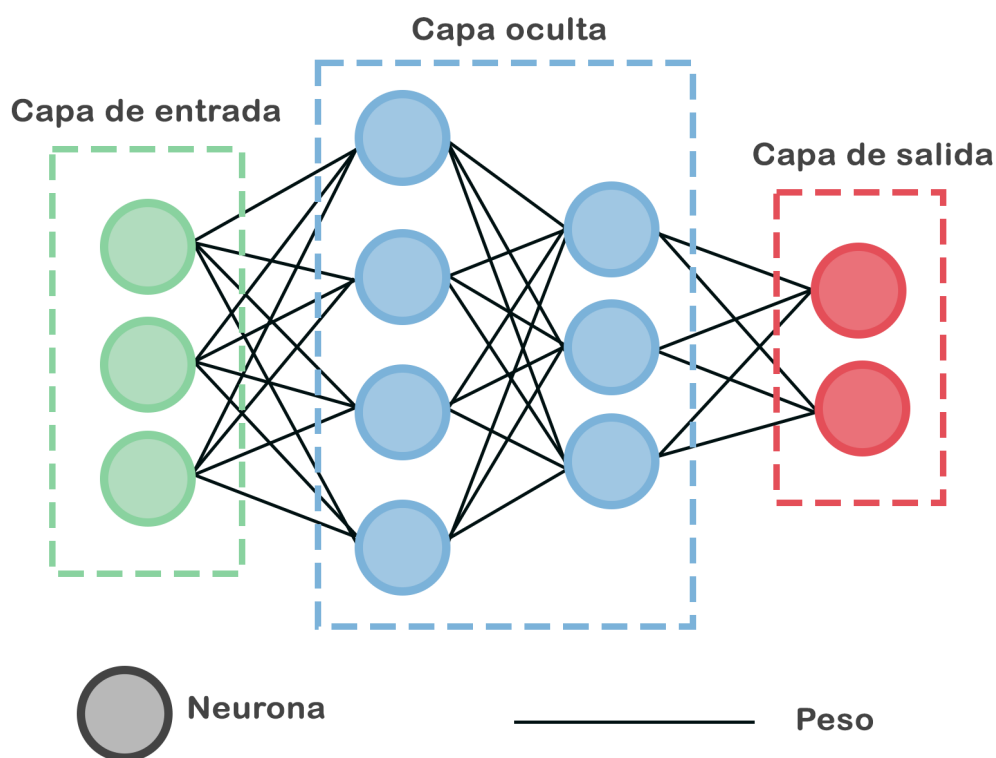


Figura 4: Red neuronal completamente conectada

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.

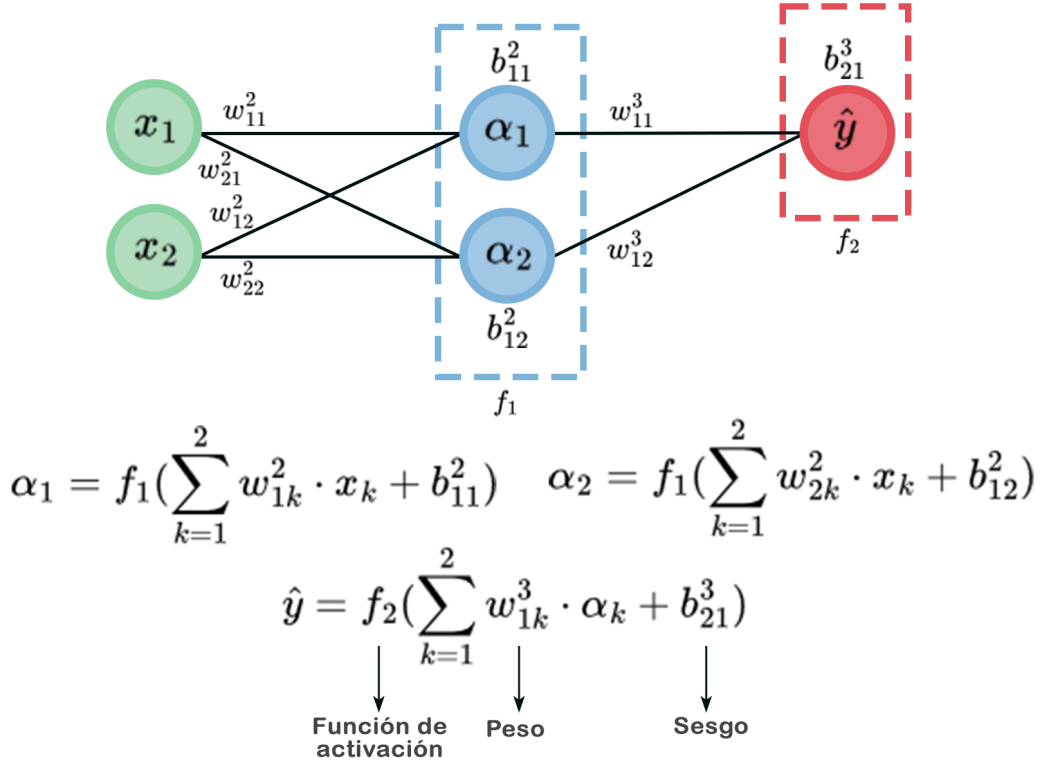


Figura 5: Red neuronal completamente conectada

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.

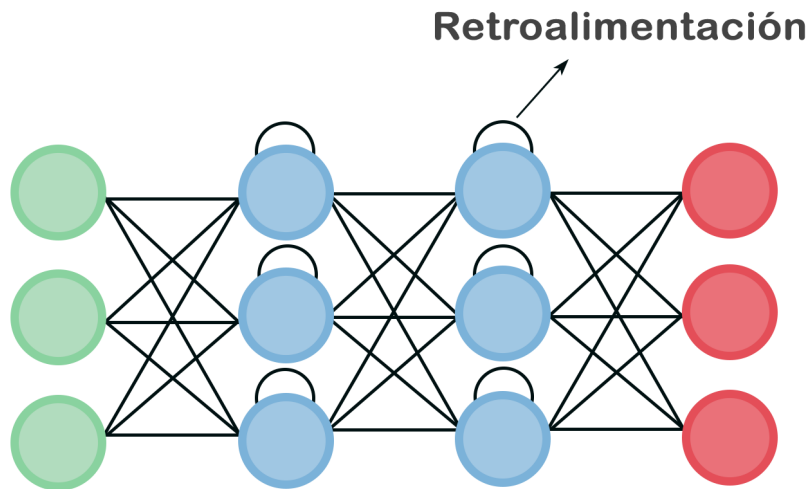


Figura 6: Ejemplo de RNN

Un problema común de las RNN es la dificultad que tienen para capturar dependencias a largo plazo en los datos. 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.

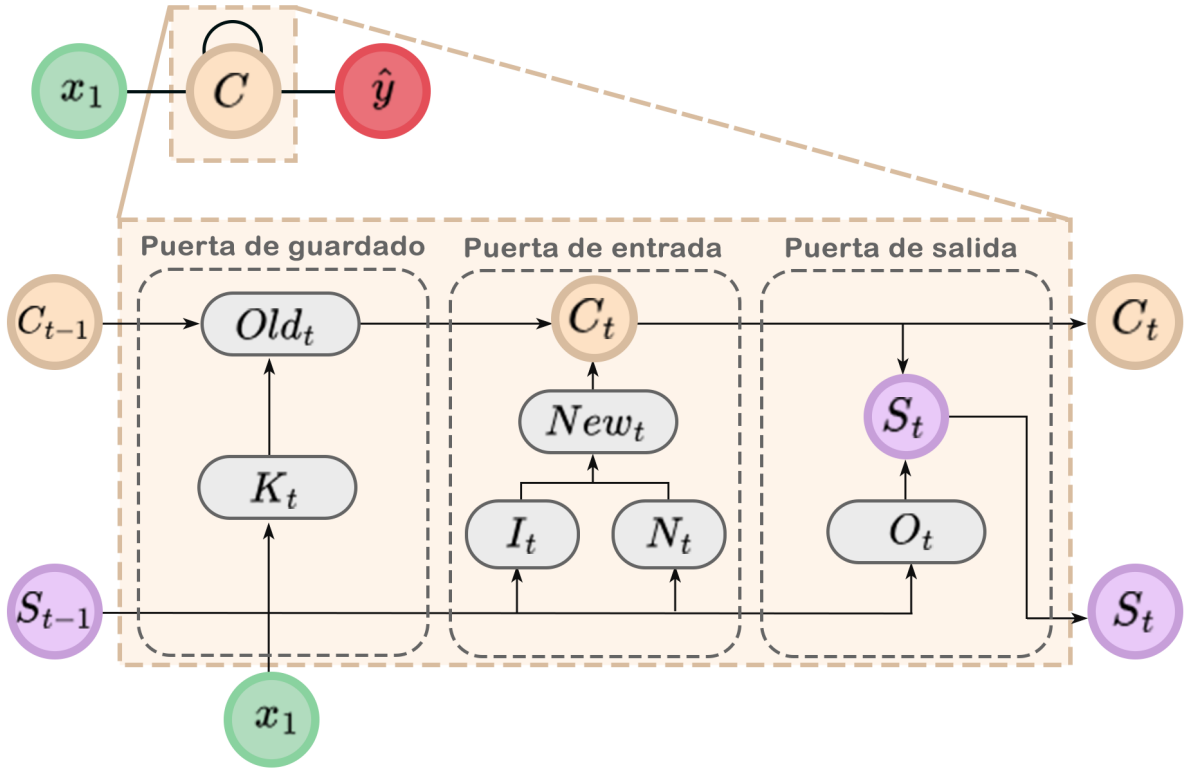


Figura 7: Estructura *Long Short Term Memory*

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.

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

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

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:

$$Old_t = K_t \times C_{t-1}$$

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:

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

$$New_t = I_t \times N_t$$

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:

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

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

Donde S_t es el nuevo estado de la red neuronal.

Las 3 puertas son logísticas 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 que información es relevante guardar.

3.4.2 Modelos transformadores

Otro tipo de modelo de aprendizaje profundo son los *transformer models* (modelos transformadores), los cuáles 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 cuáles pueden ser importante 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:

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

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

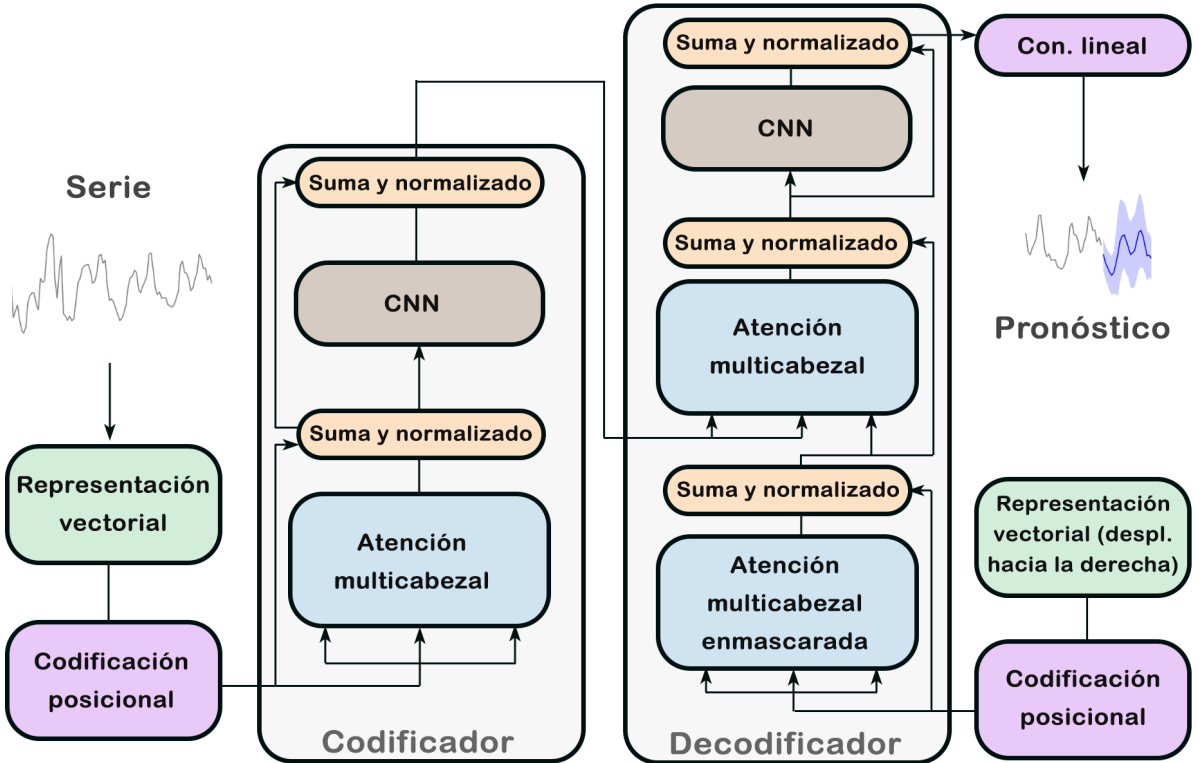


Figura 8: Diagrama de la estructura del modelo transformador de TimeGPT

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) esta 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	
	$\downarrow W_Q$	$\downarrow W_Q$	$\downarrow W_Q$	$\downarrow W_Q$	$\downarrow W_Q$	$\downarrow W_Q$	$\downarrow W_Q$	$\downarrow W_Q$	
	\vec{Q}_1	\vec{Q}_2	\vec{Q}_3	\vec{Q}_4	\vec{Q}_5	\vec{Q}_6	\vec{Q}_7	\vec{Q}_8	
$\boxed{\text{a}} \rightarrow \vec{E}_1 \xrightarrow{W_K} \vec{K}_1$	+0.7	-83.7	-24.7	-27.8	-5.2	-89.3	-45.2	-36.1	
$\boxed{\text{fluffy}} \rightarrow \vec{E}_2 \xrightarrow{W_K} \vec{K}_2$	-73.4	+2.9	-5.4	+93.0	-48.2	-87.3	-49.7	+7.8	
$\boxed{\text{blue}} \rightarrow \vec{E}_3 \xrightarrow{W_K} \vec{K}_3$	-53.4	-5.7	+1.8	+93.4	-55.6	-56.0	-26.1	-62.1	
$\boxed{\text{creature}} \rightarrow \vec{E}_4 \xrightarrow{W_K} \vec{K}_4$	-21.5	-29.7	-56.1	+4.9	-32.4	-92.3	-9.5	-28.1	
$\boxed{\text{roamed}} \rightarrow \vec{E}_5 \xrightarrow{W_K} \vec{K}_5$	-20.1	-40.9	-87.8	-55.4	+0.6	-64.7	-96.7	-18.9	
$\boxed{\text{the}} \rightarrow \vec{E}_6 \xrightarrow{W_K} \vec{K}_6$	-87.9	-33.3	-22.6	-31.4	+5.5	+0.6	-4.6	-96.8	
$\boxed{\text{verdant}} \rightarrow \vec{E}_7 \xrightarrow{W_K} \vec{K}_7$	-41.2	-55.5	-42.3	-59.8	-79.0	-97.9	+3.7	+93.8	
$\boxed{\text{forest}} \rightarrow \vec{E}_8 \xrightarrow{W_K} \vec{K}_8$	-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 W_V	$\begin{matrix} \boxed{\text{a}} \\ \downarrow \\ \vec{E}_1 \end{matrix}$	$\begin{matrix} \boxed{\text{fluffy}} \\ \downarrow \\ \vec{E}_2 \end{matrix}$	$\begin{matrix} \boxed{\text{blue}} \\ \downarrow \\ \vec{E}_3 \end{matrix}$	$\begin{matrix} \boxed{\text{creature}} \\ \downarrow \\ \vec{E}_4 \end{matrix}$	$\begin{matrix} \boxed{\text{roamed}} \\ \downarrow \\ \vec{E}_5 \end{matrix}$	$\begin{matrix} \boxed{\text{the}} \\ \downarrow \\ \vec{E}_6 \end{matrix}$	$\begin{matrix} \boxed{\text{verdant}} \\ \downarrow \\ \vec{E}_7 \end{matrix}$	$\begin{matrix} \boxed{\text{forest}} \\ \downarrow \\ \vec{E}_8 \end{matrix}$	
$\boxed{\text{a}} \rightarrow \vec{E}_1 \xrightarrow{W_V} \vec{V}_1$				0.00 \vec{V}_1					
$\boxed{\text{fluffy}} \rightarrow \vec{E}_2 \xrightarrow{W_V} \vec{V}_2$				0.42 \vec{V}_2					
$\boxed{\text{blue}} \rightarrow \vec{E}_3 \xrightarrow{W_V} \vec{V}_3$				0.58 \vec{V}_3					
$\boxed{\text{creature}} \rightarrow \vec{E}_4 \xrightarrow{W_V} \vec{V}_4$				0.00 \vec{V}_4					
$\boxed{\text{roamed}} \rightarrow \vec{E}_5 \xrightarrow{W_V} \vec{V}_5$				0.00 \vec{V}_5					
$\boxed{\text{the}} \rightarrow \vec{E}_6 \xrightarrow{W_V} \vec{V}_6$				0.00 \vec{V}_6					
$\boxed{\text{verdant}} \rightarrow \vec{E}_7 \xrightarrow{W_V} \vec{V}_7$				0.00 \vec{V}_7					
$\boxed{\text{forest}} \rightarrow \vec{E}_8 \xrightarrow{W_V} \vec{V}_8$				0.00 \vec{V}_8					

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

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.

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

De forma resumida

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

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.

3.4.3 Diferencias entre TimeGPT y Chronos

TimeGPT es un modelo transformer preentrenado (de aquí las siglas GPT, *generative pre-trained 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:

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

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 comunmente 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 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 h pasos hacia adelante son:

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

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

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

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

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

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

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 pronósticos probabilísticos 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*:

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

Donde:

$$W_l = IS_l - II_l$$

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

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

La correcta elección de los parámetros del modelo es la tarea más importante para llegar a un buen ajuste de los datos. Sin embargo, no es prudente utilizar todos los datos con los que se cuenta para este propósito, ya que se produciría un sobreajuste. Se habla de sobreajuste cuando el modelo aprende mucho de los datos de entrenamiento y se ajusta demasiado al patrón de estos, logrando que al querer estimar datos nuevos el modelo se desempeñe muy mal. Para evitar esto se utilizan distintas técnicas para la estimación de parámetros.

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.

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

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 del trabajo tiene como objetivo implementar, ajustar y comparar los modelos previamente detallados sobre un conjunto de series temporales seleccionadas, usando el software de programación de código abierto Python.

Se trabajará con series temporales reales obtenidas de bases de datos públicas y confiables. Para cada serie 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. Para cada modelo se explorarán distintas configuraciones de parámetros explicando sus significados y funciones.

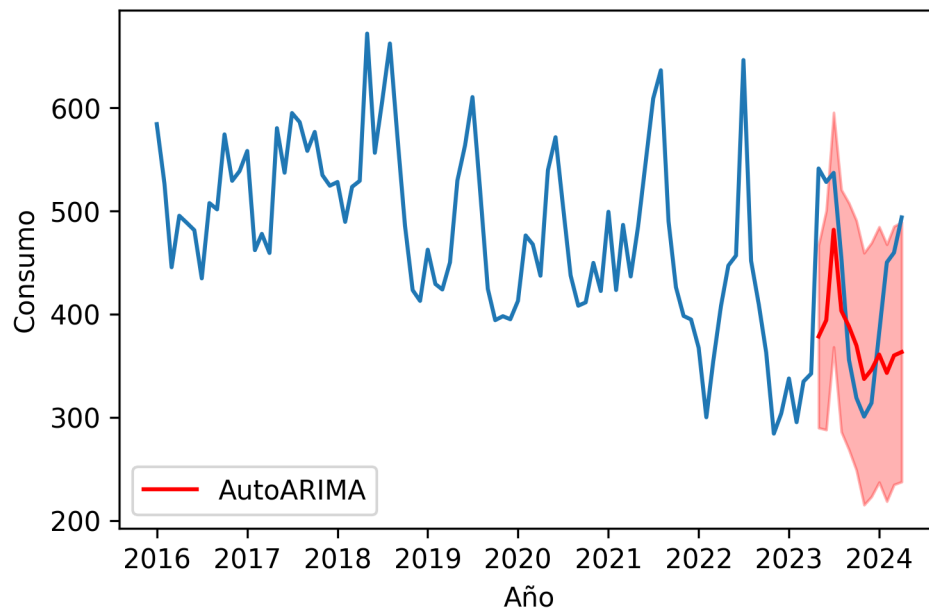
- Los modelos estadísticos clásicos, como ARIMA y SARIMA, serán ajustados utilizando la librería `pmdarima`. La selección de parámetros se realizará manualmente y con la librería mencionada, basandose en el criterio de información de Akaike (AIC).
- En el caso de los modelos de aprendizaje automático, se emplearán bosques aleatorios con *boosting* haciendo uso de los algoritmos XGBoost y LightGBM. Estos modelos serán implementados con las librerías `xgboost` y `lightgbm` respectivamente.
- Para los modelos de aprendizaje profundo, se desarrollarán redes neuronales del tipo LSTM, 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`.

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.

4.1 Series a utilizar

Métodos Tradicionales

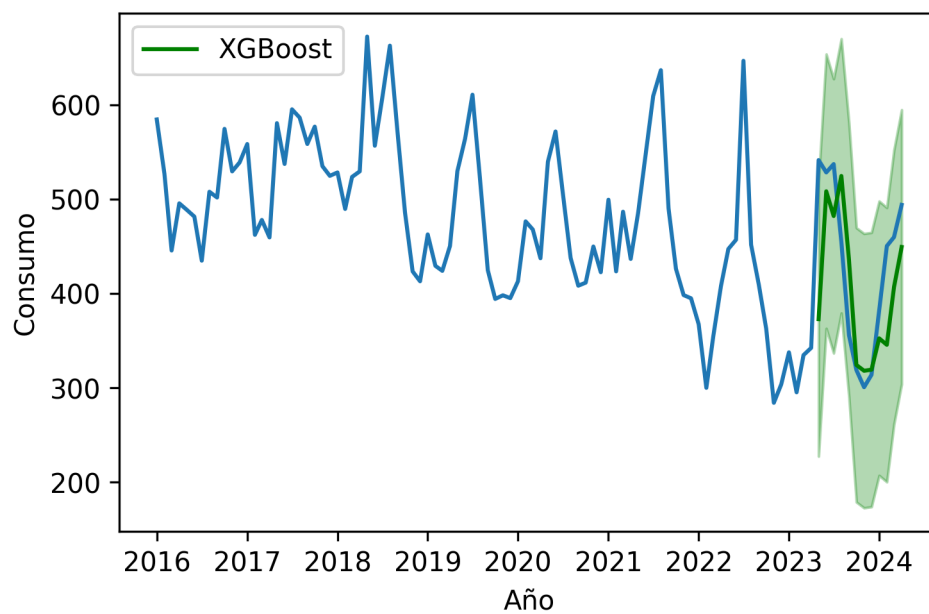
ARIMA



Métodos de Machine Learning

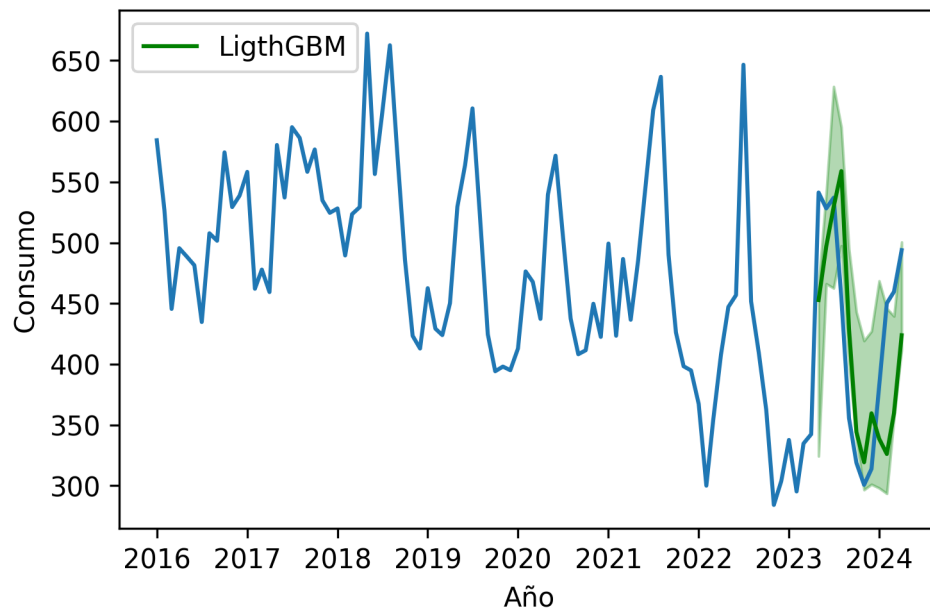
XGBoost

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



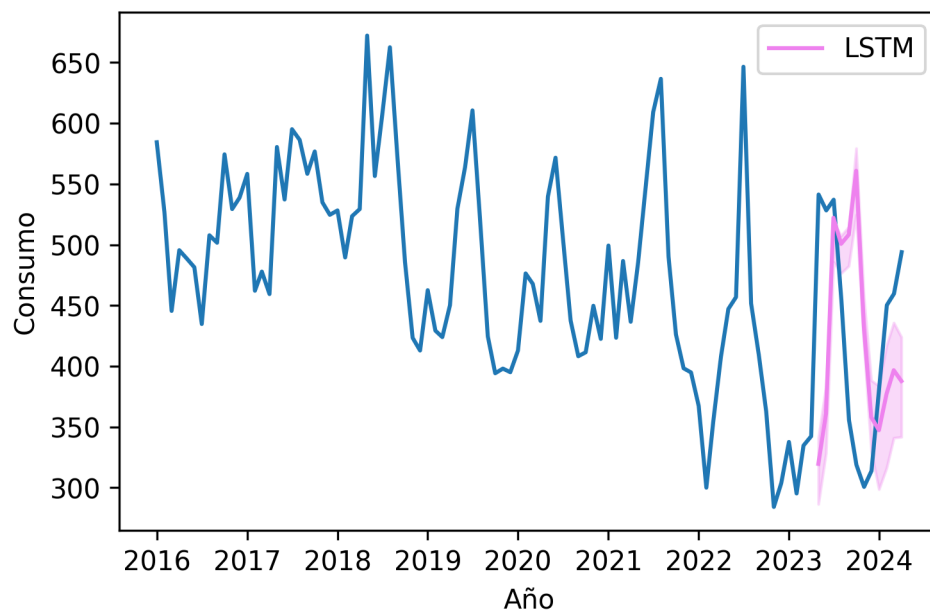
LightGBM

importante, los quantiles de estos pronosticos son independientes

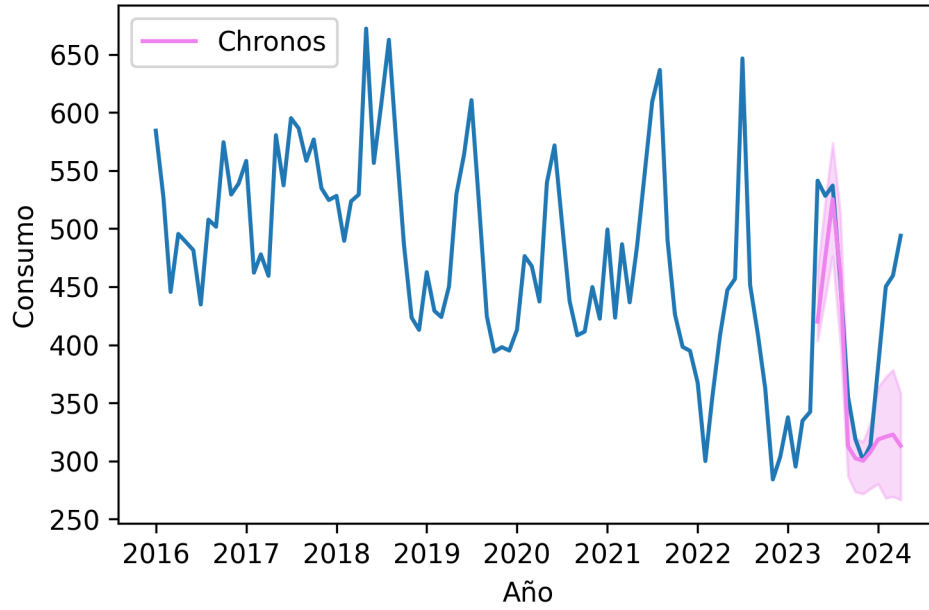


Deep Learning

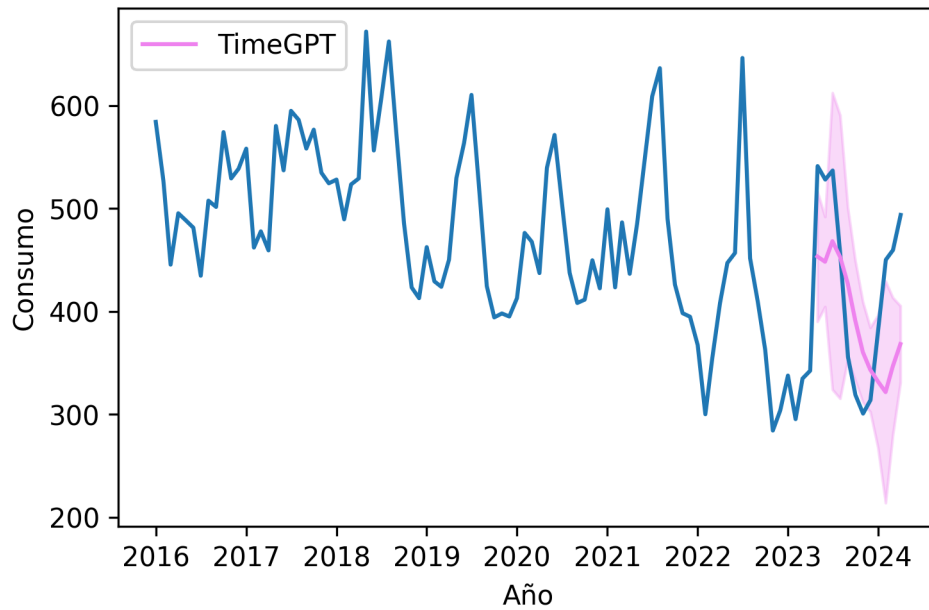
LSTM



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

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.** (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.
- 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 librerías, software y hardware en la que se corre el código
- Hacer equivalencia del documento en GitBook
- Datos: Graciela N. Klekailo Facultad de Ciencias Agrarias, Dirección general de estadística municipalidad
- Toda la modelización pasarla a .py y no .qmd
- leer fpp3 que tiene unas cosas que pueden servir
- Series a utilizar: Estacionalidad y tendencia > trabajadores registrados en educación en el ámbito privado (tmb puede ser aalim/tabaco, servicios de seg/lim/inv,) Estacionalidad > Demanda mensual residencial del consumo de luz o algún emae estacional, ENFERMEDADES RESPIRATORIAS variable exógena > Temperatura + Lluvia?
- TO DO: Aplicar a las distintas series Introducción a las series a utilizar Explicar Aplicación, parámetros usados Revisar Arima revisar transformers revisar medidas de evaluación
- Hacer dendograma de importancia de variables para los modelos de machine learning

Nombrar las 3 opciones de cálculo de intervalo de xgboost Tomar el tiempo como el último modelo PERO EXPLICARLO

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

Mejoras a la investigación (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 métodos
- 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