UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Pronósticos con Algoritmos de Aprendizaje Automático. Estudio Comparativo.

TESIS

Que para obtener el título de

Ingeniero Industrial

P R E S E N T A

Jorge Andrés Peña Araya

DIRECTORA DE TESIS

Dra. Esther Segura Perez

Ciudad Universitaria, Cd. Mx., 2020

All models are wrong, but some are useful

–George E. P. Box

Contenido

Introducción

Las decisiones en muchas áreas científicas, industriales y económicas, como la meteorología, los procesos productivos y las finanzas son guiadas por un pronóstico. Las estadísticas y la econometría han sido las principales áreas que se han encargado del estudio y desarrollo de métodos y modelos de pronósticos. La introducción de la metodología Box-Jenkins a los modelos ARIMA, la simplicidad, fácil implementación y los buenos resultados de los modelos estadísticos en general, los han mantenido por décadas como una herramienta indispensable para pronósticos de series de tiempo. Adicionalmente, los estudios empíricos dirigidos, tanto a la comunidad académica como a los profesionales interesados en utilizar métodos más precisos para sus diversas aplicaciones y reducir costos o maximizar beneficios, han destacado el enfoque de pronósticos con métodos estadísticos durante décadas [Makridakis et al., 2018b].

En las ultimas tres décadas, los algoritmos de aprendizaje automático -*Machine Learning Algorithms*- se han destacado como herramientas muy eficientes en la resolución de problemas en todas las áreas de ciencias, ingeniería y la industria. Han logrado exactitudes y desempeños únicos en problemas de traducción automática, reconocimiento de objetos, reconocimiento de voz, diagnóstico médico, automóviles autónomos, chatbots inteligentes y asistentes virtuales. La gran cantidad de datos disponibles, el poder de cómputo en la actualidad en conjunto con la gran capacidad de los algoritmos de aprendizaje automático para aprender y adaptarse a complejas relaciones que existen en los datos; además de ser robustos al ruido y trabajar con no linealidades, han posicionado al aprendizaje automático como la primera opción para abordar problemas de aprendizaje con datos, reconocimiento de patrones y modelado predictivo.

Como dato adicional, en [Columbus, 2019] se estima que el gasto global en sistemas de inteligencia cognitivos alcanzará los $77.6B de dólares en el 2022, creando un valor adicional de $2.6T en marketing y ventas, y hasta $ 2T en los rubros de manufactura y

planificación de cadenas de suministro. El mayor uso se espera en las áreas de investigación y descubrimiento farmacéutico, asesores de compras expertos y recomendaciones de productos, asistentes digitales para trabajadores de conocimiento empresarial y automatización de procesamiento inteligente.

Los algoritmos de aprendizaje automático -ML, para abreviar- también se han implementado en problemas de pronósticos. Existen numerosos estudios que se centran en *pronósticos de un paso adelante*. La mayoría de estos estudios proponen una implementación híbrida de modelos para obtener pronósticos más precisos. Por ejemplo, implementar un método estadístico en conjunto con un método basado en aprendizaje automático [Dorffner, 1996, Zhang, 2003, Gorr, 1994, Crone et al., 2011]. En el estudio de [Ahmed et al., 2010] se realizó un comparativo de los principales algoritmos de aprendizaje automático para pronósticos de un paso adelante con las series de tiempo de la competencia M3 [Makridakis and M., 2000]. La red neuronal *feed-forward* (perceptrón multicapa), Procesos Gaussianos y las Redes Neuronales Bayesianas obtuvieron los pronósticos más precisos, basados en el error sMAPE.

De mayor importancia y el gran desafío en problemas reales de pronósticos de series de tiempo es que los métodos de pronósticos tengan la capacidad y la precisión de predecir varias observaciones futuras. Los estudios y la literatura para este tipo de problemas denominados *pronósticos de múltiples pasos* con el enfoque de aprendizaje automático es escasa y los resultados reportados son diversos, no concluyentes y engañosos. Resultan engañosos porque aseguran obtener buenos resultados y pronósticos muy precisos, pero no existe un marco comparativo que los soporte. Makridakis et al. en un estudio del 2018, comparan los algoritmos de aprendizaje automático con los métodos estadísticos para pronósticos de un solo paso -considerando los resultados del estudio de [Ahmed et al., 2010] antes mencionado- y pronósticos de múltiples pasos con las series de tiempo de la competencia M3, implementando la estrategia Directa, Recursiva y MIMO -Multiples Input Multiples Outputs-. Concluyeron que los pronósticos más precisos y eficientes se

obtienen utilizando los métodos estadísticos, destacando el método ARIMA y el método Theta. El mencionado estudio destaca la eficiencia, rapidez de computo y las precisiones de los pronósticos que se obtienen con la mayoría de los métodos estadísticos utilizado y crítica a los especialistas de aprendizaje automático, con el fin de que las potencialidades que muestran los algoritmos inteligentes en otras áreas o aplicaciones, se reflejen en los problemas de pronósticos [Makridakis et al., 2018b].

Recientemente, Makridakis en [Makridakis et al., 2018a] presentó los resultados de la competencia M4. En este trabajo concluyó que, de los 17 métodos más precisos, 12 fueron *combinaciones* de enfoques mayormente estadísticos. La mayor sorpresa fue un enfoque *híbrido* que utilizaba características estadísticas y de aprendizaje automático. Adicional a esto, los seis métodos de aprendizaje automático *puros* funcionaron mal, ninguno de ellos fue más preciso que el punto de referencia y solo uno fue más preciso que el modelo ingenuo Naïve2.

La necesidad de probar y comparar las capacidades y acercar el enfoque de los algoritmos de aprendizaje automático para problemas de pronósticos de múltiples pasos utilizando series de tiempo en una aplicación de ingeniería industrial fue la motivación principal -hasta la publicación de Makridakis et al. y guardando las proporciones- para realizar esta tesis. En ese sentido se planteó responder lo siguiente: *¿por qué los sofisticados algoritmos de aprendizaje automático son tan exitosos para muchos problemas de ciencia e ingeniería y para el enfoque de pronósticos no hay buenos resultados o son dudosos y no existe una literatura seria que soporte y destaque sus capacidades?*.

Por otra parte, el estudio de Makridakis et al. esclarece las inquietudes acerca del aprendizaje automático aplicado a problemas de pronósticos multi-pasos. Prueba distintas técnicas de preprocesamiento y la validación *Kfold*. Sin embargo, y al mismo tiempo, los resultados que se estaban obteniendo en este estudio, no coincidían del todo con los resultados de Makridakis et al. La reacción inmediata fue comparar e igualar la metodología que implementaron en su estudio y encontrar los errores y diferencias en la

metodología que se presenta en este estudio.

De acuerdo con lo anterior, el presente estudio intenta igualar la metodología ocupada en el estudio de Makridakis para un problema particular de pronósticos de ventas. Presentamos pequeñas diferencias, pero importantes, que soportan la mejora de los algoritmos de aprendizaje automático como una herramienta útil para pronósticos de múltiples pasos con series de tiempo.

Objetivos

El objetivo principal de esta tesis es abordar el problema de pronósticos de múltiples pasos de series de tiempo desde el enfoque del aprendizaje automático. Se busca determinar y comparar las precisiones de los más importantes algoritmos de aprendizaje automático, las técnicas de validación de los modelos y las técnicas de pronósticos de múltiples pasos; incluyendo las precisiones de los modelos estadísticos más importantes.

Específicamente, se pronosticaron las ventas de un año de una compañía dedicada a la industria alimenticia con los algoritmos ML más importantes y utilizados, tomando como referencia los estudios de Ahmed et al. [Ahmed et al., 2010] y Makridakis et al. [Makridakis et al., 2018b]. Además, se agregaron los métodos de ensamble basados en árboles de regresión: bosques aleatorios (RF), árboles ensacados (BAGG), árboles extremadamente aleatorios (ExTree) y el algoritmo extreme gradient boosting (XGB o XGBoost) que ha ganado varias competencias Kaggle [1], la competencia KDD-Cup, Netflix Prize, entre otras.

Asimismo, la investigación se llevó a cabo dentro del enfoque de aprendizaje profundo, la Red Neuronal Convolucional (CNN o ConvNet), que destaca en el campo de visión por computadora y cuyo enfoque es aplicable a problemas de pronósticos de múltiples

[1]Kaggle, una subsidiaria de Google LLC; esta es una comunidad en línea de científicos de datos y profesionales del aprendizaje automático que permite a los usuarios encontrar y publicar conjuntos de datos, explorar y construir modelos, trabajar con otros profesionales y participar en concursos para resolver desafíos de ciencia de datos.

pasos. Como se informa en los resultados, las redes CNN son el método con el cual se obtienen los pronósticos más precisos. Hay que destacar que este estudio se viene realizando por más de un año, agregando los valores reales de las ventas proporcionados por la compañía mes por mes y repitiendo el procedimiento de obtener los pronósticos cada mes y por un año. La finalidad de realizar los pronósticos por un año permite confirmar la estabilidad y la robustez de los modelos al incluir *datos nuevos*.

En línea con lo anterior, Makridakis et al. en su estudio, utilizan la estrategia Directa, Recursiva y MIMO -Multiples Input Multiples Outputs- para obtener los pronósticos. En este estudio se incluye la estrategia híbrida que combina la técnica directa y recursiva. Además, se incluyen la técnicas de validación *Walk Forward*, que a diferencia de la técnica K-fold -utilizada por Makridakis- y la técnica Hold-out, permite entrenar y validar los modelos con los datos ordenados temporalmente. Se considera como premisa, que los modelos captarán mejor las características de estacionalidad o tendencia de una serie de tiempo, cuando se entrenan con los datos ordenados en el tiempo, garantizando mejores resultados en la generalización y la precisión de los pronósticos.

Con respecto a las redes neuronales artificiales, nos basamos en el enfoque actual de aprendizaje profundo. Puntualmente, se implementaron redes neuronales más profundas y se comprueba que se obtienen mejores resultados. Se utilizan las estrategias de *paradas tempranas*, para disminuir el tiempo de computo y regularizaciones dedicadas, como la regularización *Dropout* que garantizan mejores aprendizajes y pronósticos más precisos.

La exposición de la tesis esta dividida en tres partes: la primera describe los fundamentos del aprendizaje automático -ML para abreviar- y se describen brevemente los algoritmos utilizados. Los algoritmos de ML utilizados se dividieron en: i) Métodos no paramétricos: Vecino Más cercano (KNN), Regresión con vectores de soporte (SVR), Procesos gaussianos (GP), Árboles de clasificación y regresión (CART) e incluimos la Red neuronal con función de base radial (RBF), Red neuronal de regresión generalizada (GRNN) y Red neuronal regularizada bayesiana (BRNN) que se deriva del Perceptrón

multicapa, pero no comparten el enfoque de aprendizaje profundo. ii) Métodos de aprendizaje de ensamble: árboles embolsados (BAGG), bosque aleatorio (RF), árboles extremadamente aleatorios (ExT), Gradient Boosting Machine (GBM) y el algoritmo XGBoost y iii) Métodos de aprendizaje profundo: Perceptrón multicapa (MLP). Las redes neuronales recurrentes: unidad recurrente cerrada (GRU) y la red de memoria a largo plazo (LSTM) y por último la red neuronal Convolucional (ConvNet).

La segunda parte es una breve descripción de los métodos estadísticos utilizados para obtener los pronósticos: Método Holt, método de suavizado exponencial simple, método Holt-Winters aditivo y multiplicativo amortiguado, ARIMA y el método Theta. Incluyendo los métodos de referencia ingenuos no estacional Naïve y el método ingenuo estacional Naïve2.

Finalmente, se presenta el estudio de caso, cómo se obtuvieron los parámetros e hiperparámetros para los diferentes algoritmos. Las heurísticas para determinar, por ejemplo, las arquitecturas de los diferentes tipos de redes neuronales, las estructuras de los árboles de regresión, etc. Asimismo, se muestran las técnicas implementadas para reducir los recursos y tiempos de cómputo. Y cómo lograr un mejor rendimiento en los algoritmos de aprendizaje automático. Seguimos presentando los pronósticos obtenidos, informamos los resultados de las diferentes medidas de error y tiempos computacionales que cada uno de los algoritmos exige; para finalizar con las conclusiones y una sección de discusión y el futuro del estudio.

Respecto al caso de estudio, cabe recalcar que los datos de la empresa han sido tratados exclusivamente de forma académica y técnica, para aplicar los métodos y algoritmos del aprendizaje automático. Aunque el trabajo de esta tesis utiliza datos reales de ventas, se debe aclarar que los beneficios puntuales en cuanto a las ganancias financieras y/o ventajas competitivas que pueda obtener la compañía en cuestión, caen fuera del alcance de la presente investigación.

Capítulo 1

Aprendizaje Automático

Aprendizaje automático es una rama de la inteligencia artificial, que tiene como objetivo obtener modelos que generalicen datos *no vistos* -un conjunto de datos que no se considera en el modelo-. Debido a las diferentes características que tienen los datos, diferentes tipos de aprendizajes se han considerado. Aprendizaje no supervisado, aprendizaje de refuerzo, aprendizaje activo, aprendizaje en linea, entre otros -para más detalles sobre las características y enfoques de los diferentes tipos de aprendizaje, ver [Russell and Norvig, 2009]-.

En este trabajo y principalmente para obtener pronósticos de múltiples pasos con series de tiempo, se implementará el enfoque del *Aprendizaje supervisado*. La característica principal de los datos $\mathcal{D}$ en el aprendizaje supervisado es que a cada entrada o característica $\mathbf{x} \in \mathbb{R}^n$ le corresponde una salida u objetivo $y \in \mathbb{R}$. Y el objetivo principal es *aprender* la relación $f : \mathbf{x} \to y$. Dependiendo de las características de la salida, este problema se conoce como regresión, para salidas continuas, o clasificación, cuando las salidas son discretas.

De manera general, los algoritmos de aprendizaje automático minimizan la perdida o error en las etapas de entrenamiento y validación para obtener una hipótesis candidata f_d que mejor se aproxima a la relación $f : \mathbf{x} \to y$. Para garantizar el mínimo *error de generalización* se ha establecido un procedimiento y conjunto de técnicas que se describen brevemente a continuación.

1.1. entrenamiento con datos

El proceso de aprendizaje será satisfactorio cuando el error en las predicciones para un conjunto de datos *no vistos* sea mínimo. A este conjunto de datos se le llama *conjunto de prueba* y corresponde a una parte del conjunto de datos total $\mathcal{D}$, con la cual se evalúa el rendimiento del algoritmo ML. La parte restante de los datos, generalmente se divide en dos conjuntos: i) *conjunto de entrenamiento*, es donde se ejecuta el algoritmo de aprendizaje automático ii) *conjunto de validación* o también llamado conjunto de desarrollo -En ingles, *hold-out validation*-, se utiliza para ajustar la configuración, hiperparámetros, seleccionar las características y la toma de decisiones con respecto al algoritmo ML. El conjunto de prueba no se utiliza para la toma de decisiones con respecto a las características de los modelos o de los parámetros utilizados.

Específicamente, en la etapa de entrenamiento se minimiza una función objetivo -generalmente para problemas de regresión se utiliza la función raíz media cuadrada- que esta definida por:

$$E_t = \frac{1}{M}\sum_{i=1}^{M}(f_d(\mathbf{x}_i) - y_i)^2 \tag{1.1}$$

Obteniendo el error promedio E_t en el conjunto de entrenamiento M, o también llamado error en la muestra -*in sample*- que se minimiza para obtener una solución candidata f_d:

$$f_d = argmin_{f_d \in \mathcal{M}} \quad E_t \tag{1.2}$$

$f_d \in \mathcal{M}$, donde $\mathcal{M}$ corresponde al espacio de hipótesis, se evalúa en el conjunto de validación y se obtiene el error de validación E_v o error fuera de la muestra -*out of sample*-. Y se espera en el mejor de los casos la condición,

$$argmin_{f_d \in \mathcal{M}} \; E_t \approx argmin_{f_d \in \mathcal{M}} \; E_v \tag{1.3}$$

En la práctica, la condición anterior dada por la expresión 1.3 utilizando los algoritmos ML es sesgada. Los algoritmos ML se caracterizan por utilizar espacios de hipótesis $\mathcal{M}$ complejos garantizando un error de entrenamiento E_t mínimo -cercano a cero-. Sin embargo, un bajo E_t no garantiza un buen desempeño al utilizar un conjunto nuevo de datos -conjunto de validación-. El sesgo entre los errores de entrenamiento E_t y validación E_v se debe principalmente a dos factores. i) la complejidad del espacio de hipótesis $\mathcal{M}$ y ii) el tamaño del conjunto de datos $\mathcal{D}$, que comúnmente se conoce como problema de aprendizaje y generalización o problema de *sobreajuste* que se describe a continuación.

1.2. problema de sobreajuste (overfitting)

La comprensión del problema de aprendizaje y generalización o problema de *sobreajuste* puede entenderse con los conceptos estadísticos de sesgo y varianza de un modelo. El sesgo y la varianza son propiedades estadísticas importantes asociadas con cualquier modelo basado en datos [Geman et al., 1992]. El sesgo mide el desplazamiento de la media de las predicciones $\bar{f}_d(\mathbf{x})$ con respecto a $f(\mathbf{x})$ y la varianza del modelo se relaciona con la estabilidad de los modelos y representa las variación de las predicciones de $f_d(\mathbf{x})$ alrededor de su media $\bar{f}_d(\mathbf{x})$.

Los modelos ML al utilizar espacios de hipótesis o de funciones $\mathcal{M}$ complejos, por ejemplo, utilizar funciones de transferencia no lineales para obtener un modelo basado en redes neuronales artificiales, se consideran modelos con alta complejidad, modelos con varianza alta o modelos muy propensos al *sobreajuste*. En otras palabras, los modelos ML aprenden muy bien en el conjunto de entrenamiento -bajo error de entrenamiento- y no necesariamente generalizan bien en un conjunto de *datos no vistos*. Si, por el contrario,

el conjunto de hipótesis $\mathcal{M}$ es simple, como ocurre por ejemplo en los *modelos ingenuos*, como el modelo Naïve o Naïve estacional que se utilizan como base de comparación en el enfoque de pronósticos, la estimación f_d será deficiente y el error en el conjunto de datos de entrenamiento será alto. En otras palabras, el modelo *no aprende* en el conjunto de entrenamiento, lo que se conoce como bajoajuste -*underfitting*-.

Otro factor importante que causa sobreajuste son las características de los conjuntos de datos y en particular el tamaño de los conjuntos de datos. Es recurrente asociar los algoritmos de ML con conjuntos de datos *grandes*. Por ejemplo, la base de datos IMDB que se utiliza para problemas de clasificación y reconocimiento de patrones, posee 460.000 fotos de celebridades. Los conjuntos de datos grandes o *metadatos* garantizan una buena generalización y evitan sobreajuste en los modelos ML. Esta regla no se cumple en el enfoque de pronósticos de series de tiempo. En pronósticos, los datos representativos son los del *pasado cercano*, los últimos datos y generalmente la importancia va disminuyendo mientras más al pasado se encuentren. Bajo este argumento es erróneo pensar que las series de tiempo largas -o con muchos datos- garantizarán pronósticos precisos y sin duda provocará sobreajuste en un algoritmo ML.

Antes de presentar las soluciones al problema de sobreajuste, cuando se tienen conjunto de datos pequeños, se describen en la siguiente sección las *curvas de aprendizaje* como ayuda gráfica al diagnóstico del problema de sobreajuste.

1.3. curvas de aprendizaje

Los errores de entrenamiento y de validación se pueden graficar a través del tiempo o después de cada actualización o *época*, con el objetivo de monitorear el rendimiento del modelo ML -El número de épocas es un hiperparámetro que define el número de veces que el algoritmo ML funcionará en todo el conjunto de datos de entrenamiento-. Concretamente, las curvas de aprendizaje nos ayudarán a diagnosticar si el modelo tiene

problemas de sobrejuste. A continuación, se muestran los casos más comunes y sus probables causas.

- La figura 1.1 muestra el caso más común en los algoritmos ML, *sobreajuste*. Muy bajo error en el entrenamiento -curva roja- y mala capacidad de generalizar -curva azul-. Visualmente la brecha entre ambas curvas, nos indicará el grado de variabilidad o varianza del modelo, a mayor brecha mayor varianza. Por otra parte, el sesgo del modelo es bajo ya que el error de entrenamiento es bajo, es decir, el modelo se ajusto bien a los datos del conjunto de entrenamiento.

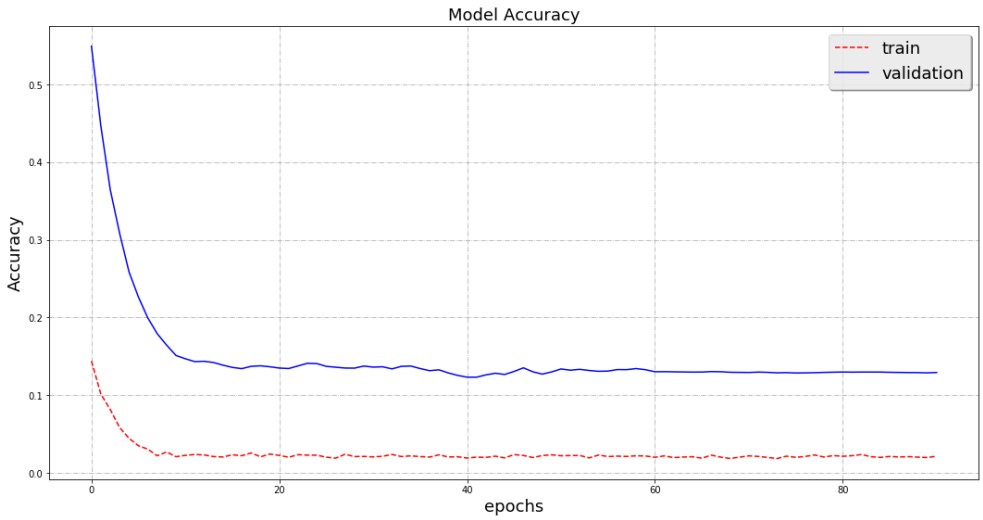


Figura 1.1. Curva de aprendizaje de una red neuronal LSTM. De color rojo, se muestra el error del conjunto de entrenamiento y en azul se muestra el error en el conjunto de validación después de cada época. La brecha entre ambas curvas determina un modelo con varianza o sobreajustado.

- La figura 1.2 muestra que el algoritmo ML no se ajusta bien al conjunto de entrenamiento -error de entrenamiento alto-, caso de *bajoajuste* o sesgo del modelo. También existe variabilidad (brecha de ambas curvas). La diferencia en este caso es que el modelo se desempeña mejor en el conjunto de validación y no se asocia a problemas de sobreajuste; sino que se asocian a un conjunto de validación pequeño, no representativos o que no proveen información suficiente del fenómeno.

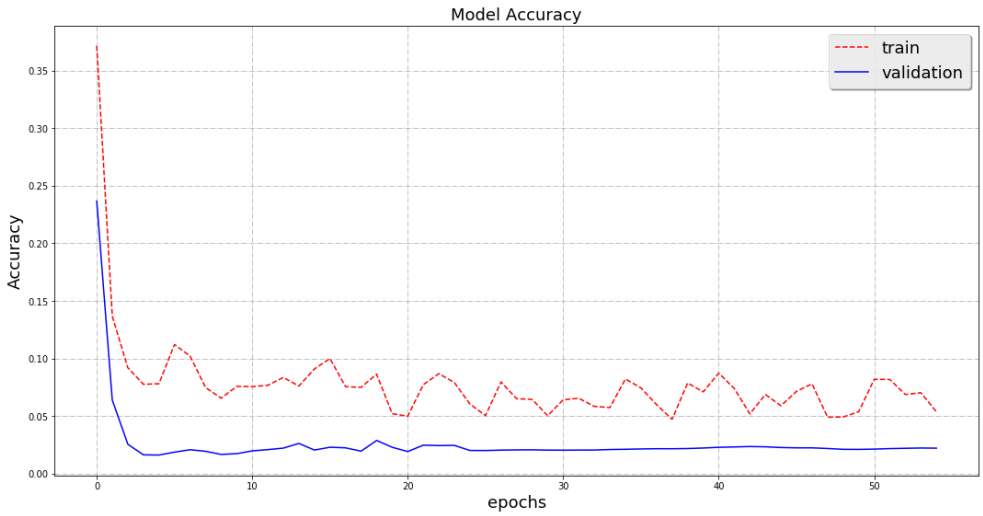


Figura 1.2. Curva de aprendizaje de una red neuronal LSTM. De color rojo, se muestra el error del conjunto de entrenamiento y en azul se muestra el error en el conjunto de validación después de cada época.

- La figura 1.3 muestra que después de unas épocas, los errores de entrenamiento y de validación disminuyen hasta que se estabilizan con una mínima diferencia. Visualmente, la brecha entre las curvas es mínima y el error de entrenamiento es bajo; lo que se asocia a una baja varianza y bajo sesgo del modelo.

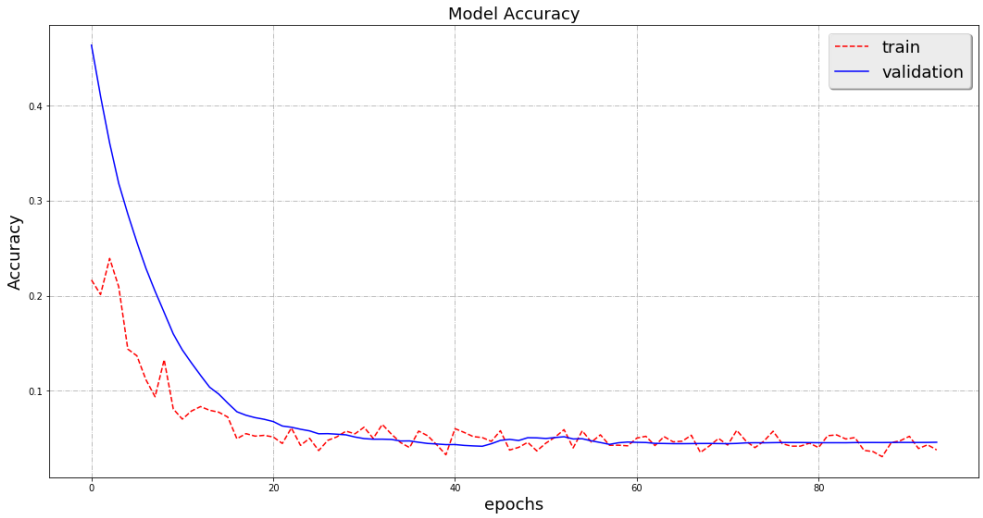


Figura 1.3. Curva de aprendizaje de una red neuronal LSTM. De color rojo, se muestra el error del conjunto de entrenamiento y en azul se muestra el error en el conjunto de validación después de cada época.

La figura 1.3 representa el objetivo central de los algoritmos de aprendizaje automático. Es decir, un buen entrenamiento y una buena generalización o en términos estadísticos,

una buena compensación entre la varianza y el sesgo del modelo -condición de la ec.1.3-. Para lograr esta condición, es necesario reducir el sobreajuste del modelo. A continuación, se describen las técnicas de regularización y validación cruzada, como ayuda para reducir el problema de sobreajuste.

1.4. reducción del sobreajuste

Para entender la reducción de sobreajuste, primero hay que enfatizar, que un modelo con alta capacidad o *complejidad* puede aprender *demasiado bien* -bajo error de entrenamiento- y sobreajustar el conjunto de entrenamiento. Y, por otra parte, un modelo con baja capacidad o complejidad no puede aprender las características del *fenómeno* -error de entrenamiento alto-, lo que se conoce como *bajoajuste*. Ambos casos no garantizan una buena generalización en el conjunto de datos *no vistos*; y en el caso particular de los métodos de aprendizaje automático, es común enfrentarse al problema de sobreajuste debido a la alta complejidad que tienen estos métodos. Un enfoque para reducir el sobreajuste es regularizar el modelo en la etapa de entrenamiento. A continuación, se describen las técnicas más utilizadas para reducir el problema de sobreajuste.

1.4.1. regularización

Una opción para solucionar el problema de sobreajuste es reducir el número de variables de entrada $\mathbf{x}$. Sin embargo, al eliminar algunas variables de entrada se elimina información que tenemos sobre el fenómeno, lo que puede perjudicar las capacidades de predicción del modelo. Otra opción, es *penalizar* la complejidad del modelo agregando un término en la función de perdida:

$$E_t = \frac{1}{M}[\sum_{i=1}^{M}(f_d(\mathbf{x}_i) - y_i)^2 + \lambda \sum_{j=1}^{M} w_j^2] \tag{1.4}$$

La constante λ restringe los valores de los parámetros $\mathbf{w}$ de los modelos. Si λ es pequeño, los valores de los parámetros serán pequeños y se obtendrán modelos más suaves y simples y, por lo tanto, menos propensas al sobreajuste. En términos estadísticos, esta regularización se llama *contracción* [Hastie et al., 2001] y en la literatura de ML se llama *regularización de pesos* y penaliza la norma de los parámetros. Se consideran dos enfoques:

- Calculando la suma de los valores absolutos de los pesos, llamada norma L1.
- Calculando la suma de los valores al cuadrado de los pesos, llamada norma L2.

En otras palabras, el hiperparámetro λ controlará la cantidad de penalización en función del tamaño de las normas L1 o L2 de los parámetros del modelo.

Existen diferentes métodos para calcular el valor de regularización como las penalizaciones que dependen del modelo: criterio de información Akaike (AIC) [Akaike, 1969] y el criterio de información bayesiano (BIC) [Schwarz, 1978], que se utilizan principalmente para la selección del modelo en los métodos estadísticos y que se describen en el capítulo 2. A continuación, se describe otra alternativa para la reducción de sobreajuste, los métodos de validación cruzada.

1.4.2. validación cruzada

En lugar de considerar un conjunto de entrenamiento y un conjunto de validación -como la validación Hold-out descrita en la sección 1.1-, los datos se pueden dividir en múltiples K divisiones independientes -o *folds*- para obtener varios modelos, que se promedian para obtener un modelo final. La idea principal de los métodos de validación cruzada es disminuir el sesgo por la diferencia en la cantidad de datos de los conjuntos de entrenamiento y validación, ya que se necesita una cantidad representativa de datos para encontrar el modelo f_d y al mismo tiempo una cantidad representativa de datos para evaluar el modelo. Consideramos para este estudio: la validación cruzada *Leave-*

One-Out-Validation (LOOV), la *validación cruzada K-fold* [Allen, 1974, Stone, 1977], Bootstrap y la validación walk forward.

validación LOOV

En LOOV, el conjunto de validación contiene un solo dato, y los $N-1$ datos restantes forman el conjunto de entrenamiento, obteniendo N conjuntos de datos diferentes. El error de validación -para cada dato-, esta dado por:

$$E_v(f_d(i)) = [f_d(x_i) - y_i)]^2 \tag{1.5}$$

los resultados se promedian para los N conjuntos de datos:

$$E_{LOOV} = \frac{1}{N}\sum_{i=1}^{N} E_v[f_d(i)] \tag{1.6}$$

Donde $f_d(i)$ es la relación aprendida del conjunto de datos $\mathcal{D}$ $\{(xi, yi)\}$ del cual eliminamos el i-ésimo dato y se evalúa en el único dato del conjunto de validación. Una de las ventajas de LOOV es que se usan $N-1$ datos para el entrenamiento, esperando un modelo muy cercano a f. Por otra parte, el error de validación E_v se basa en un ejemplo y, por lo tanto, no es confiable. Sin embargo, debido a que estamos aplicando el procedimiento N veces, esperamos que esté cerca del enfoque del conjunto de validación que contiene N observaciones. LOOV es un procedimiento de alto costo computacional, ya que ejecuta el procedimiento de aprendizaje N veces para un conjunto de datos con N ejemplos, lo que puede llevar mucho tiempo cuando N es grande.

validación K-fold

La validación cruzada $K-fold$ divide los datos en K secciones independientes o $folds$ $\mathcal{D}_1, ..., \mathcal{D}_K$ y a cada división $\mathcal{D}_K$ de tamaño N/K se le aplica el mismo procedimiento que en validación LOOV. Aprende la relación en $\mathcal{D}$ sin $\mathcal{D}_K$ y luego se obtiene el error

de validación en $\mathcal{D}_K$. Así sucesivamente para todas las divisiones K:

$$E_v(f_d) = \sum_{x_i \in \mathcal{D}_k} (f_d(x_i) - y_i)^2 \tag{1.7}$$

Donde f_d es la relación aprendida de los datos sin $\mathcal{D}k$, que se evalúa en los datos $\mathcal{D}k$. Al evaluar todos las K partes, se promedian los errores de validación obteniendo:

$$E_{Kfold} = \frac{1}{k} \sum_{k=1}^{k} E_v(f_d) \tag{1.8}$$

E_{Kfold} estima E_v y el sesgo y la varianza dependen del valor de k. Si k es grande como en la validación $LOOV$, la función se estimará con la mayoría de los datos de $\mathcal{D}$, por lo que el sesgo sera pequeño, de lo contrario, si k es pequeño el sesgo de la estimación f_d sera mayor. Makridakis en su estudio [Makridakis et al., 2018b] utiliza la validación $k - fold$ con $k = 10$ para obtener los pronósticos con los algoritmos ML. Este valor a demostrado empíricamente tener una buena compensación entre sesgo y varianza [Kohavi, 1995].

La principal ventaja de los métodos de validación cruzada es su generalidad. De hecho, no hacen suposiciones sobre el modelo subyacente y, por lo tanto, pueden aplicarse a cualquier modelo. Una desventaja de estos métodos es que los datos del conjunto de entrenamiento disminuyen y puede perjudicar la capacidad del modelo cuando se tienen conjunto de datos pequeños.

1.4.3. bootstrap

Bootstrap es otro método de *remuestreo* para estimar los errores *fuera de muestra* o error de validación. El método consiste en tomar una *muestra aleatoria* del conjunto de datos inicial, formando nuevos conjuntos de datos o *muestras bootstrap*. Con cada muestra bootstrap se entrena un modelo y con los datos restante se calcula el error fuera de muestra. Al final los modelos se combinan para generar el modelo final. Como

se verá en la sección 1.7, Bootstrap es una técnica importante para generar algoritmos ML basados en arboles de regresión. Además, los métodos estadísticos de pronósticos utilizan muestras bootstrap de los errores de los pronósticos o residuo para obtener varios posibles pronósticos de un modelo y calcular los intervalos de pronósticos. También se pueden obtener pronósticos con muestras bootstrap que se agregan a la serie de tiempo descompuesta como medida de incertidumbre y de mejora de los pronósticos [Hyndman and Athanasopoulos, 2018].

[Kohavi, 1995] proporciona una comparación entre validación cruzada y bootstrap y concluye que no existe un mejor enfoque y el rendimiento de cada enfoque dependerá de factores como el tamaño del conjunto de datos y la validez de los supuestos. Además, los dos enfoques se pueden combinar, por ejemplo, algunos métodos de regularización usan validación cruzada para encontrar la cantidad correcta de regularización. En algunos casos, se ha determinado que algunos métodos de regularización son equivalentes a los métodos de validación [Ben Taieb et al., 2010].Para más detalles de la técnica bootstrap ver: *Bootstrap Methods: Another Look at the Jackknife* [Efron, 1992].

1.4.4. validación Walk Forward

Un factor importante que no se considera en el estudio de [Makridakis et al., 2018b] cuando se entrenan y validan los modelos ML, es considerar el orden o dependencia temporal de los datos -el estudio de Makridakis solo considera la validación *K-fold*, que supone que no hay relación entre las observaciones y que cada observación es independiente-. Un enfoque basado en problemas secuenciales y pronósticos de series de tiempo, considera la importancia de entrenar y validar los modelos con datos ordenados temporalmente [1]. Tomando este enfoque, la validación *Hold-Out* es más probable funcione mejor que los enfoques *K-fold*, ya que existe un orden de los datos del pasado al presente

[1]En el pronóstico de series de tiempo, la evaluación de modelos en datos históricos se llama backtesting. En algunos dominios de series de tiempo, como la meteorología, esto se denomina predicción posterior - hindcasting -, en lugar de pronósticos.

y considera la mitad posterior como datos de validación. Sin embargo, la división de entrenamiento-prueba podría no ser un buen indicador de la relación temporal en los datos. En otras palabras, donde se elige dividir los datos juega un papel importante en el desempeño del modelo -principalmente en el conjunto de validación-. Con la finalidad de potenciar la etapa de entrenamiento y validación de datos ordenados temporalmente, se propone la validación de avance o *Walk Forward Validation*.

En la práctica, primero se selecciona el número mínimo de observaciones requeridas para entrenar el modelo -llamado ancho de la ventana-. A continuación, se debe decidir si el modelo se entrenara en todos los datos que tenga disponibles o solo en las observaciones más recientes -asumiendo que las observaciones mas recientes tienen mayor importancia-. Esto determina si se usará una ventana deslizante o desplegable. En el procedimiento, la primera ventana definida se usa para entrenar un modelo. Luego, se hace una predicción con el modelo entrenado para el próximo paso de tiempo. Esta predicción se almacena o evalúa con respecto al valor conocido y se obtiene el error de validación definido anteriormente. La ventana se expande para incluir el valor conocido y el proceso se repite para los N valores correspondientes de la serie de tiempo.

Adicionalmente a los métodos de regularización y validación presentados en esta sección, existen metodologías de regularización especificas para algunos algoritmos de aprendizaje automático. Puntualmente, es ampliamente citado y recomendado, utilizar la regularización *Dropout* en aprendizaje profundo. La regularización *Dropout* se incluye en este estudio y se describe en la sección 1.9. Por otra parte, en la sección 3.3, se describen los detalles de las configuraciones e implementación de cada una de las técnicas de entrenamiento y validación presentadas.

A continuación, se describen brevemente los diferentes métodos de aprendizaje automático utilizados para obtener los pronósticos de múltiples pasos.

1.5. modelos ML utilizados

En las siguientes secciones se describen brevemente los algoritmos de aprendizaje automático que utilizaron los estudios de [Makridakis et al., 2018b] y [Ahmed et al., 2010] para obtener los pronósticos de las series de tiempo de la competencia M3 y que se utilizan en este estudio. La primera categoría corresponde a los Métodos No paramétricos: Vecino más cercano (KNN), Regresión con Vectores de Soporte (SVR), Procesos gaussianos (GP) y Árboles de clasificación y regresión (CART).

1.6. modelos no paramétricos

Los modelos No paramétricos ML son en general modelos que no necesitan parámetros para obtener las predicciones y se basan en simples métodos que utilizan los datos cercanos al nuevo dato objetivo x' o una función kernel para realizar una predicción.

1.6.1. vecino más cercano (KNN)

El algoritmo Vecino más cercano o K-Nearest Neighbors (KNN) es un simple método no paramétrico utilizado también para problemas de regresión [2] asumiendo similitud en el espacio característico [Altman, 1992, Atkeson et al., 1997]. La predicción para x' esta dada por:

$$f_d(x') = \frac{1}{k} \sum_{x_i \in N_k(x')} y_i \tag{1.9}$$

Donde $Nk(x')$ es la proximidad a x' y contiene k datos que se promedian para calcular la predicción [Hastie et al., 2001]. Utilizamos la distancia euclidiana para encontrar los k puntos o vecinos más cercanos x_i en el conjunto de entrenamiento. El hiperparámetro

[2] los algoritmos de aprendizaje automático se han desarrollado primeramente para problemas de clasificación, reconocimiento de patrones y luego se derivaron para resolver problemas de regresión

k es definido por el usuario y determina la complejidad del modelo. Se implementó con la funciones *KNeighborsRegressor()* y *GridSearchCV ()* del API Scikit-learn [Pedregosa et al., 2011] para Python 3.7.

1.6.2. red neuronal de regresión generalizada (GRNN)

Al utilizar los promedios de los datos en el algoritmo KNN, se generan discontinuidades, ya que el promedio cambia de manera discreta a diferentes valores x'. La Red Neuronal de Regresión Generalizada GRNN utiliza una función kernel para que los promedios ponderados locales varíen suavemente, obteniendo funciones más suaves [Specht, 1991]. La estimación para x' esta dada por:

$$f_d(x') = \frac{\sum_{k=1}^{N_k} y_k \; K(x', x_k)}{\sum_{k=1}^{N_k} K(x', x_k)} \tag{1.10}$$

Donde y_k son los objetivos de x_k para los N_k valores en la proximidad de x' y K es una función kernel. La función kernel comúnmente usada es la función de base radial -kernel gaussiano- definida por:

$$K(x', x) = \exp\Big(- \frac{\|x - x'\|^2}{2\gamma}\Big) \tag{1.11}$$

La función kernel asigna pesos a los datos, que caen exponencialmente con el cuadrado de la distancia desde x. El hiperparámetro γ lo define el usuario y determina el ancho de la vecindad o tamaño de la ventana. Si γ es grande implica menor varianza ya que se promedia sobre más datos y un sesgo alto debido a la función constante en (x_{N_k}, y_{N_k}). El algoritmo GRNN se implementó con el paquete *Neupy* y la función GridSearchCV () de Scikit-learn para Python 3.7.

1.6.3. proceso gaussiano (GP)

El proceso gaussiano -*Gaussian Process*- es un modelo no paramétrico, donde a cada entrada $\mathbf{x}$ se le asigna una variable aleatorias $f(\mathbf{x})$ formando una distribución conjunta gaussiana multivariable y se define por su media, dada por:

$$m(\mathbf{x}) = \mathbb{E}[f(\mathbf{x})] \tag{1.12}$$

y su covarianza,

$$\mathrm{Cov}(f(\mathbf{x}), f(\mathbf{x}')) = K(\mathbf{x}, \mathbf{x}') = \mathbb{E}[(f(\mathbf{x}) - m(\mathbf{x}))(f(\mathbf{x}') - m(\mathbf{x}'))] \tag{1.13}$$

Así, un proceso gaussiano se puede representar por:

$$f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), K(\mathbf{x}, \mathbf{x}')) \tag{1.14}$$

Un proceso Gaussiano asume una distribución de probabilidad a priori normal para los datos de entrada:

$$f(\mathbf{x}) \sim \mathcal{N}(0, K(\mathbf{x}, \mathbf{x}') + \sigma_n^2) \tag{1.15}$$

y $K(\mathbf{x}, \mathbf{x}')$ es una función Kernel que se aplica para obtener la aproximación f_d. Usando las reglas de condicionamiento gaussianas sobre la distribución a priori, obtenemos las ecuaciones predictivas del proceso gaussiano:

$$f_d(\mathbf{x}')|\mathbf{x}, \mathbf{y}, \mathbf{x}' \sim \mathcal{N}(\bar{f}_d(\mathbf{x}'), \mathrm{Cov}[f_d(\mathbf{x}')]) \tag{1.16}$$

donde la media predictiva esta dada por:

$$\bar{f}_d(\mathbf{x}') = K(\mathbf{x}', \mathbf{x})[K(\mathbf{x}, \mathbf{x}) + \sigma_n^2\ \mathcal{I}]^{-1}\mathbf{y} \tag{1.17}$$

y la covarianza predictiva está dada por:

$$\text{Cov}[f_d(\mathbf{x}')] = K(\mathbf{x}', \mathbf{x}') - K(\mathbf{x}', \mathbf{x})[K(\mathbf{x}, \mathbf{x}) + \sigma_n^2 \, \mathcal{I}]^{-1}\mathbf{y} \; K(\mathbf{x}, \mathbf{x}') \tag{1.18}$$

La función Kernel de base radial:

$$\text{Cov}(f(\mathbf{x}), f(\mathbf{x}')) = \sigma_n^2 \; \exp\left(- \frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\gamma^2} \right) \tag{1.19}$$

El hiperparámetro σ_n^2 controla la varianza general de la función y γ la suavidad del ajuste. Los principales conceptos y el enfoque para el algoritmo de procesos gaussianos se tomaron del libro Gaussian Processes for Machine Learning [Rasmussen and Williams, 2005]), que puede detallarse en el sitio web del libro Gaussian Processes for Machine Learning. En este trabajo se implementó el algoritmo de procesos gaussianos con la función *GaussianProcessRegressor ()*, la función *GridSearchCV ()* para el proceso de entrenamiento y validación del API Scikit-learn para Python 3.7.

1.6.4. regresión con vectores de soporte (SVR)

Regresión con Vectores de Soporte -Support Vector Regression- deriva del algoritmo máquina de vectores de soporte -Support Vector Machine- que se utiliza para resolver problemas de clasificación y forman parte de los denominados *métodos kernel*. Fueron populares en la década de los noventas debido a su solida base en la teoría de aprendizaje estadístico o Teoría VC [Vapnik, 2000]. De manera general, el algoritmo SVR aprenderá la relación, dependencia, mapeo o función $f_d(\mathbf{x})$, que tiene la forma:

$$f_d(\mathbf{x}) = \mathbf{w}^T \Phi(\mathbf{x}) + b \tag{1.20}$$

La entrada $\mathbf{x} \in \mathbb{R}^n$ se mapea a un espacio característico de dimensión superior F, $\Phi(x) = [\phi_1(\mathbf{x}), \phi_2(\mathbf{x}), ..., \phi_n(\mathbf{x})]^T \in \mathbb{R}^f$, donde se resuelve un problema de regresión lineal.

$\Phi(\mathbf{x})$ es una función fija elegida, $\mathbf{w}$ es el vector de pesos o parámetros y b es el valor de sesgo asociado. La solución para 1.20 -llamado hiperplano de regresión- creará una hipersuperficie de regresión no lineal en el espacio de entrada original. Vapnik utiliza una función de perdida denominada función de perdida insensible ϵ, definida por:

$$E\Big[\mathbf{x}, y, f_d(\mathbf{x})\Big] = \begin{cases} 0 & \text{si } |y - f_d(\mathbf{x})| < \epsilon \ , \\ |y - f_d(\mathbf{x})| - \epsilon & \text{en otro caso.} \end{cases} \tag{1.21}$$

La particularidad de esta función de perdida es que el error en $[-\epsilon \ \epsilon]$ es cero, es decir, permite que $f_d(\mathbf{x})$ tenga desviaciones de valor 2 ϵ, también llamado tubo ϵ. Para obtener $\mathbf{w}$ y b se minimiza:

$$\arg\min_{\mathbf{w},b} \ C\sum_{i=1}^{M}|y_i - f_d(\mathbf{x}_i)|_\epsilon + \frac{1}{2}\|\mathbf{w}\|^2 \tag{1.22}$$

Como se puede ver en la ecuación 1.22, se agrega la constante C que compensa el error y el término de regularización $\|\mathbf{w}\|^2$. Además, para asegurar que el problema de optimización sea factible se agregan las variables de holgura ξ y ξ^* que representan la distancia desde los datos fuera del tubo al límite del tubo ϵ. Con estas variables de holgura, el problema de optimización -Ec. 1.22- se puede plantear en función de dichas variables de holgura [Vapnik, 2000]:

$$\text{mín}\, C\sum_{i=1}^{M}(\xi_i + \xi_i^*) + \frac{1}{2}\|\mathbf{w}\|^2 \tag{1.23}$$

Sujeto a

$$\begin{aligned} (\mathbf{w}^T\Phi(\mathbf{x}_i) + b) - y_i \leq \epsilon + \xi, \quad i = 1, \ldots, m. \\ y_i - (\mathbf{w}^T\Phi(\mathbf{x}_i) + b) \leq \epsilon + \xi^*, \quad i = 1, \ldots, m. \\ \xi \geq 0, \quad \xi^* \geq 0 \quad i = 1, \ldots, m. \end{aligned}$$

Se resuelve el problema de optimización 1.23 mediante el planteamiento de la función

de Lagrange:

$$\begin{aligned} L_p(\mathbf{w}, b, \xi_i, \xi_i^*, \alpha_i, \alpha_i^*, \beta_i, \beta_i^*) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{M}(\beta_i\xi_i + \beta_i^*\xi_i^*) \\ - \sum_{i=1}^{M}\alpha_i(\mathbf{w}^T\Phi(\mathbf{x}_i) + b) - y_i + \epsilon + \xi) \\ - \sum_{i=1}^{M}\alpha_i^*(y_i - (\mathbf{w}^T\Phi(\mathbf{x}_i) + b) + \epsilon + \xi^*) \end{aligned} \tag{1.24}$$

Donde $\alpha_i \geq 0, \alpha_i^* \geq 0, \beta_i \geq 0, \beta_i^* \geq 0$, son los multiplicadores de Lagrange. Se puede demostrar que esta función tiene un punto silla con respecto a las variables primal y dual en la solución -para más detalles [Mangasarian, 1969] y [Mc Cormick, 1983]-. La solución de la ecuación 1.24 se encuentra satisfaciendo las condiciones de Karush-Kuhn-Tucker -condiciones KKT-, donde las variables primales $\mathbf{w}, b, \xi_i, \xi_i^*$ deben desaparecer:

$$\begin{aligned} \frac{\partial L_p}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{M}(\alpha_i - \alpha_i^*)\Phi\mathbf{x}_i = 0 \\ \mathbf{w} = \sum_{i=1}^{M}(\alpha_i - \alpha_i^*)\Phi\mathbf{x}_i \end{aligned} \tag{1.25}$$

$$\frac{\partial L_p}{\partial b} = -\sum_{i=1}^{M}(\alpha_i - \alpha_i^*) = 0 \tag{1.26}$$

$$\begin{aligned} \frac{\partial L_p}{\partial \xi_i} = C - \alpha_i - \beta_i = 0 \\ \beta_i = C - \alpha_i, \quad \alpha_i \in [0 \quad C] \end{aligned} \tag{1.27}$$

$$\begin{aligned} \frac{\partial L_p}{\partial \xi_i^*} = C - \alpha_i^* - \beta_i^* = 0 \\ \beta_i^* = C - \alpha_i^*, \quad \alpha_i^* \in [0 \quad C] \end{aligned} \tag{1.28}$$

Las derivadas parciales con respecto a los multiplicadores de Lagrange devuelven las restricciones que tienen que ser menores o iguales a cero:

$$\frac{\partial L_p}{\partial \alpha_i} = \mathbf{w}^T \Phi(\mathbf{x}_i) + b - y_i + \epsilon + \xi_i \leq 0 \tag{1.29}$$

$$\frac{\partial L_p}{\partial \alpha_i^*} = y_i - \mathbf{w}^T \Phi(\mathbf{x}_i) - b + \epsilon + \xi_i^* \leq 0 \tag{1.30}$$

$$\frac{\partial L_p}{\partial \beta_i} = \sum_{i=1}^{M} \xi_i \leq 0 \tag{1.31}$$

$$\frac{\partial L_p}{\partial \beta_i^*} = \sum_{i=1}^{M} \xi_i^* \leq 0 \tag{1.32}$$

La condición final KKT establece que el producto de los multiplicadores de Lagrange y las restricciones son igual a cero:

$$\alpha_i(\mathbf{w}^T \Phi(\mathbf{x}_i) + b - y_i + \epsilon + \xi_i) = 0 \tag{1.33}$$

$$\alpha_i^*(-\mathbf{w}^T \Phi(\mathbf{x}_i) - b + y_i + \epsilon + \xi_i) = 0 \tag{1.34}$$

$$\beta_i \xi_i = (C - \alpha_i)\xi_i = 0 \tag{1.35}$$

$$\beta_i^* \xi_i^* = (C - \alpha_i^*)\xi_i^* = 0 \tag{1.36}$$

De 1.35 y 1.36 se deduce que solo los datos que les corresponde un valor de $\alpha_i, \alpha_i^* = C$ están fuera del tubo ϵ, o, dicho de otra forma, para todos los datos que están fuera del tubo ϵ, $|y - \mathbf{w}^T \mathbf{\Phi}(\mathbf{x_i}) - b| \geq \epsilon$, los valores de los multiplicadores de Lagrange son distintos de cero. A este conjunto de datos se les llama *Vectores de Soporte* y construyen la función de decisión $f_d(\mathbf{x})$. Por otra parte, con 1.33 y 1.33 se puede calcular el valor del término de sesgo b:

$$b = y_i - \mathbf{w}^T \Phi(\mathbf{x}_i) - \epsilon, \quad \text{para} \quad 0 \leq \alpha_1 \leq C \tag{1.37}$$

$$b = y_i - \mathbf{w}^T \Phi(\mathbf{x}_i) + \epsilon, \quad \text{para} \quad 0 \leq \alpha_1^* \leq C \tag{1.38}$$

El cálculo del término b es numéricamente muy sensible, y se recomienda calcular

b promediando sobre todos los vectores de soporte. Retomando el lagrangiano primal de la ecuación 1.24 y reemplazando las derivadas en este, se obtiene el problema de optimización dual:

$$L_d(\alpha_i, \alpha_i^*) = -\frac{1}{2}\sum_{i=1}^{M}\sum_{j=1}^{M}(\alpha_i - \alpha_i^*)(\alpha_j\alpha_j^*)\Phi(\mathbf{x}_i)\Phi(\mathbf{x}_j) - \epsilon\sum_{i=1}^{M}(\alpha_i - \alpha_i^*) + \sum_{i=1}^{M}(\alpha_i - \alpha_i^*)y_i \quad (1.39)$$

Sujeto a

$$\sum_{i=1}^{M}(\alpha_i - \alpha_i^*) = 0$$

$$0 \leq \alpha_i \leq C$$

$$0 \leq \alpha_i^* \leq C$$

Como podemos observar de la ecuación 1.39, el problema de optimización dual depende del producto punto $\Phi(\mathbf{x}_i)\Phi(\mathbf{x}_j)$ de las entradas $\mathbf{x}$ en el espacio de alta dimensión F, que calcularlo puede ser computacionalmente costoso si la dimensión de $\mathbf{x}$ es muy alta. Este problema está relacionado con un fenómeno conocido como la *maldición de la dimensionalidad* -para mas detalles, ver [Bishop, 2006]-. Para resolver este problema se utiliza una función kernel que cumple con $K(\mathbf{x_i}, \mathbf{x_j}) = \Phi(\mathbf{x}_i)^T\Phi(\mathbf{x}_j)$ y satisfacen las condiciones de Mercer [Mercer, 1909]. Así el problema de optimización dual 1.39 se representa:

$$L_d(\alpha_i, \alpha_i^*) = -\frac{1}{2}\sum_{i=1}^{M}\sum_{j=1}^{M}(\alpha_i - \alpha_i^*)(\alpha_j\alpha_j^*)K(\mathbf{x_i}, \mathbf{x_j}) - \epsilon\sum_{i=1}^{M}(\alpha_i - \alpha_i^*) + \sum_{i=1}^{M}(\alpha_i - \alpha_i^*)y_i \quad (1.40)$$

Al resolver el problema dual obtenemos los valores de los multiplicadores de Lagrange y reemplazando el valor de $\mathbf{w}$ dado por 1.25 en la función de aproximación $f_d(\mathbf{x})$ dada por 1.20 obtenemos:

$$f_d(x') = \sum_{i=1}^{M}(\alpha_i - \alpha_i^*)\Phi(x_i)\Phi(x') + b \quad (1.41)$$

o de manera similar, se puede sustituir por:

$$f_d(x') = \sum_{i=1}^{M}(\alpha_i - \alpha_i^*)K(x_i, x') + b \tag{1.42}$$

Esta es la llamada expansión SV, que indica que **w** se puede describir completamente como una combinación lineal de un subconjunto del conjunto de entrenamiento x_i que corresponden a los *vectores de soporte*. Así, el algoritmo completo se puede describir en términos de productos de puntos entre los datos e incluso cuando se evalúa $f_d(\mathbf{x})$ no es necesario calcular **w** de manera explícita.

Por otro lado, se debe considerar que la cantidad de vectores de soporte es importante para la precisión de la predicción. Al tener menos vectores de soporte, menor será la precisión requerida para aproximar los datos originales. Usualmente los kernels lineal, polinomial y de base radial son los más utilizados, para este estudio se utilizó la función kernel de base radial y en base a los estudios [Cherkassky and Ma, 2004, Ahmed et al., 2010] se propusieron los hiperparámetros ϵ, γ y C. En la practica, se utilizaron las funciones *GridSearchCV()* y *SVR()* del paquete Scikit-learn [Pedregosa et al., 2011] para el entrenamiento y validación del algoritmo SVR en Python 3.7.

1.6.5. árboles de clasificación y regresión (CART)

CART -*Classification and Regression Trees*- es un algoritmo de árbol de decisión y método no paramétrico, que particiona recursivamente el espacio característico según un conjunto de reglas de decisión simples -*aprendices base*- en estructura de árbol hasta que se alcanza el nodo terminal o la hoja j [Breiman, 1984]. El valor de cada hoja corresponde al promedio de los valores objetivos del conjunto de datos de entrenamiento para cada partición o región R_j, es decir, un valor constante por hoja c_j:

$$x \in R_j \Rightarrow f_d(x) = c_j \tag{1.43}$$

La estructura de las estimaciones de superficie de regresión es de la forma:

$$f_d(x) = T(x, \Theta) = \sum_{j=1}^{J} c_j \, \mathcal{I}(x \in R_j), \; T \in \mathcal{F} \tag{1.44}$$

Donde $\mathcal{F}$ es el espacio de árboles de regresión o espacio CART. Los hiperparámetros $\Theta = \{R_j, c_j\}_1^J$, se encuentran minimizando la función de perdida E:

$$\hat{\Theta} = \arg\min_{\Theta} \sum_{j=1}^{J} \sum_{x_i \in R_j} E \; (y_i, c_j) \tag{1.45}$$

El tamaño de árbol apropiado -número de divisiones o nodos en el árbol- evita el sobreajuste y determina la complejidad del modelo. Una práctica típica, una vez diseñado y alcanzado un tamaño de nodo inicial, es eliminar nodos ineficaces para mantener la complejidad del modelo y evitar el sobreajuste -*tree pruning*-. El algoritmo CART se implementó utilizando la función *GridSearchCV ()* y la función *DecisionTreeRegressor ()* del API Scikit-learn para Python 3.7. Solo el parámetro de profundidad de árbol -*max depth*- se estableció como una búsqueda de cuadrícula para evitar el ajuste excesivo y mantener la complejidad del modelo, el resto de los parámetros se dejaron por defecto. Los árboles de decisión destacan por ser fáciles de entender e interpretar y pueden manejar problemas de salida múltiple. Sin embargo, no son robustos, ya que pueden generar diferentes modelos para pequeñas variaciones en los datos. Utilizando un *método de ensamble* que se describe a continuación, se puede corregir este problema.

1.7. métodos de ensamble

El algoritmo CART visto en la sección anterior proporciona la base para los algoritmos esenciales en los métodos de ensamble. Un problema importante con el algoritmo CART

es su alta varianza, debido a que el error en una región R_j se propaga a todas las regiones posteriores [Hastie et al., 2001]. Una forma de reducir la varianza es combinar modelos básicos simples o *aprendices débiles* [3] para obtener un modelo más preciso -*aprendices fuertes*-. *Bagging*, *Boosting* y *Stacking* son métodos de ensamble que se han propuesto para mejorar la generalización de los modelos basados en árboles de regresión y en general se utiliza como mejora para todos los algoritmos de aprendizaje automático.

Bagging o Bootstrap AGGregatING: construye múltiples modelos a partir de diferentes submuestras del conjunto de datos de entrenamiento -cada árbol no debe estar correlacionado-. La predicción final se promedia a través de las predicciones de todos los submodelos [Breiman, 1996].

Boosting: agrega y entrena secuencialmente un modelo o aprendiz débil para corregir los errores de predicción del modelo anterior y se combinan para producir una regla de predicción más precisa [Bühlmann and Hothorn, 2007, Schapire and Freund, 2012].

Stacking -agregación apilada-: construye dos o más modelos independientes a partir del conjunto de datos. Las predicciones finales son las ponderaciones de las predicciones de cada modelo por separado.

Hay una gran cantidad de algoritmos que utilizan los métodos de ensamble -no necesariamente una estructura de árbol-. Es recurrente en la literatura mencionar que los mejores modelos se obtienen mediante la combinación de varios modelos y no usando solo un *modelo puro*. En las últimas décadas, se han destacado los métodos de ensamble con estructuras de árbol, debido a su fácil implementación, rápida ejecución, alta precisión y flexibles al control de sobreajuste. En este estudio, implementamos los métodos más reconocidos y exitosos: Bosque Aleatorio, Árboles Extremadamente Aleatorios, El método *Bagging*, Gradient Boosting Machine y el algoritmo XGBoost que describimos brevemente a continuación.

[3] un aprendiz débil, una hipótesis débil o una función débil se define como aquella cuyo desempeño es ligeramente mejor que el azar, tiene un alto sesgo y una baja varianza [Schapire, 1990])

1.7.1. bosque aleatorio (RF)

Si entrenamos un árbol de regresión CART, podemos reducir el sesgo de la estimación, pero no la varianza. El algoritmo Bosque Aleatorio -*Random Forest*- utiliza el método *Bagging* para construir M árboles y además un subconjunto aleatorio de las variables para determinar el modelo de árbol final. Esta aleatoriedad permite reducir aun más la varianza y el error de generalización del modelo final. El modelo está construido de manera *codiciosa*. En cada paso se crean dos nodos secundarios desde un nodo terminal mediante la definición de una división binaria utilizando una de las variables disponibles y en conjunto con un método de validación se determina la profundidad del árbol [Breiman, 1984, Breiman, 2001, Efron and Hastie, 2016]. La predicción final está definida por:

$$f_d(x) = \frac{1}{M} \sum_{m=1}^{M} T(x, \Theta_m) \tag{1.46}$$

Donde M corresponde al número de árboles y Θ_m corresponde a los hiperparámetros que caracterizan el número y los valores de los nodos terminales de cada árbol m. Específicamente para este estudio y después de realizar varias pruebas, se utilizó una búsqueda de cuadrícula para los hiperparámetros con la función *GridSearchCV ()* y la función del modelo RF *RandomForestRegressor ()* del API Scikit-learn para Python 3.7.

Además, se implementó el algoritmo *Bagging* en su forma *pura* y definición que se describió previamente, con la función *BaggingRegressor ()* y el algoritmo Árboles Extremadamente Aleatorios *Extremely Randomized Trees*- que, al igual que el algoritmo de bosque aleatorio, construye cada árbol a partir de un subconjunto aleatorio de características candidatas y además la elección del punto de corte del nodo al dividir un árbol es aleatoria, permitiendo una mayor disminución en la varianza del modelo final [Geurts et al., 2006]. El algoritmo Árboles Extremadamente Aleatorios se implementó

usando la función *GridSearchCV ()*, la función de pérdida de error cuadrático medio para el proceso de validación y la función *ExtraTreesRegressor ()* del API Scikit-learn para Python 3.7.

1.7.2. algoritmo XGBoost

Tomando un marco estadístico para el método de ensamble *Boosting*, se desarrolló *Gradient Boosting Machine (GBM)* como un problema de optimización en el que el objetivo es minimizar el error de un modelo de árbol mediante un procedimiento de gradiente descendiente [Friedman and Hastie, 2000]. Chen e Guestrin desarrollaron el algoritmo XGBoost basado en GBM que se ha convertido en la opción preferida para estimar funciones y que recientemente ha dominado el aprendizaje automático aplicado y las competencias KDD-Cup, Netflix Prize y Kaggle competitions entre otras. XGBoost es rápido, eficiente en memoria y de alta precisión, ya que combina un algoritmo para el aprendizaje de árboles paralelos y una estructura para el aprendizaje de árboles fuera del núcleo (out-of-core) [Chen and Guestrin, 2016]. La idea principal del modelo XGBoost es mejorar el desempeño de un aprendiz débil (árbol de decisión) adicionando arboles secuencialmente en una estrategia codiciosa para que al combinarse se logre un modelo más preciso [Friedman, 2001]:

$$f_d(\mathbf{x}) = \sum_t f_t(x_i) \tag{1.47}$$

donde f_t corresponde a un árbol independiente $T(x, \Theta_t)$ con $T \in \mathcal{F}$ en el tiempo t:

$$f_t(x) = f_{t-1}(x_i) + T(x_i, \Theta_t) \tag{1.48}$$

Al igual que los demás algoritmos basados en arboles, Θ_t corresponde a los hiperparámetros que caracterizan el número y los valores de los nodos terminales de cada árbol, que

se obtienen minimizando la función de perdida:

$$E_{(t)} = \sum_{i=1}^{N} E(y_i, f_{t-1}(x_i) + T(x_i, \Theta_t)) + \Omega(f_t) \tag{1.49}$$

$\Omega(f_t)$ es el termino de regularización. El algoritmo XGboost implementa una aproximación de segundo grado para optimizar la ecuación 1.49, obteniendo:

$$E_{(t)} = \sum_{i=1}^{N} [g_i \; T(x_i, \Theta_t) + \frac{1}{2} h_i \; T(x_i, \Theta_t)^2)] + \Omega(f_t) \tag{1.50}$$

donde g_i y h_i son los gradientes de primer y segundo orden de la función de pérdida respectivamente. El procedimiento continúa hasta obtener el predictor final f_d. Los detalles y las mejoras del algoritmo XGBoost se pueden encontrar en [Chen and Guestrin, 2016]. Implementamos ambos algoritmos, Gradient Boosting (Gradient boosting machine) con las funciones *GridSearchCV ()* y *GradientBoostingRegressor ()* del API Scikit-learn para Python 3.7. Y el modelo XGBoost se implementó con la función *GridSearchCV ()* del API Scikit-learn y la función *XGBRegressor ()* del paquete xgboost [Chen and Guestrin, 2016, Hofner et al., 2014] para Python 3.7.

1.8. aprendizaje profundo (Deep Learning)

En los últimos años las Redes Neuronales Artificiales[4] han cambiado radicalmente la forma de modelado predictivo en el aprendizaje automático y ciencias de la computación. En adición al creciente poderío de las capacidades de procesamiento de las computadoras y a la mayor disponibilidad de datos; se han implementado con éxito redes neuronales artificiales cada vez más grandes o *profundas*. Aprendizaje profundo es el nombre

[4]l Las primeras redes neuronales datan de los años 50 con el Perceptrón como su precursor. Debido a la poca capacidad de cálculo de las computadoras y a la dificultad de entrenar redes con múltiples capas, quedaron rezagadas hasta los años 80, cuando Geoffrey Hinton de la Universidad de Toronto, junto con David Rumelhart y Ronald Williams, resolvieron este problema de entrenamiento con la publicación del algoritmo de entrenamiento Backpropagation.

específico que se les da a redes neuronales artificiales que pueden tener de siete a diez o más de capas ocultas y que pueden sumar millones de neuronas. En esta sección, describimos brevemente los algoritmos más importantes del enfoque de Aprendizaje Profundo que se implementaron para pronósticos de ventas de varios pasos.

Comenzamos con el famoso Perceptrón multicapa -MLP- que históricamente ha tenido buenos resultados en aplicaciones de pronóstico de un paso adelante [Zhang, 2003, Ahmed et al., 2010]. Para problemas de procesamiento natural del lenguaje y reconocimiento de voz se propuso retroalimentar la salida de las capas del MLP, agregando memoria a la red neuronal artificial y captara las relaciones temporales que existe en los datos. A estas redes neuronales se les denomino Redes neuronales recurrentes y se consolidaron con éxito para procesar texto -entendido como secuencias de palabras o secuencias de caracteres-, series de tiempo y datos secuenciales en general [Chollet, 2017]. En esta sección se describen los algoritmos: la red GRU -Gated Recurrent Unit- y la Red de Memoria a Largo Plazo -Long Short Time Memory-. Y finalmente la red convolucional -Convnet- que es prioridad en los problemas de visión por computadora y su enfoque es compatible en problemas de pronósticos de series de tiempo.

1.8.1. perceptrón multicapa (MLP)

El tipo más común y pionero de red neuronal artificial es el Perceptrón multicapa -*Multilayer Perceptron*-. Como muestra la figura 1.4; está compuesto por una serie de elementos de procesamiento interconectados llamados *neuronas* o *nodos* que se agrupan en *capas*. En esta arquitectura la información se mueve en una sola dirección, *hacia adelante* -Es común llamar al Perceptrón multicapa, *Feed Forward Neural Network* ya que la información se mueve de las entradas hacia las salidas, pasando por las capas ocultas-. La predicción se obtiene en la capa de salida, dada por:

$$f_d(\mathbf{x}, \mathbf{w}) = h^{(l)}(...g(\underbrace{g(\mathbf{a}^{(1)}\mathbf{w}^{(1)} + \mathbf{b}^{(1)})}_{\mathbf{a}^{(2)}}\mathbf{w}^{(2)}))...) \tag{1.51}$$

Donde $\mathbf{a}$ se denomina *activación*, por ejemplo, $\mathbf{a}^{(1)} = \mathbf{x}$ es la activación en la capa uno -el superíndice l corresponde al número de la capa-, $\mathbf{w}$ son los parámetros o pesos, $\mathbf{b}$ el sesgo y $g()$ es la función de activación o de transferencia. En la actualidad, la función unidad lineal rectificada *Rectified Linear Unit function* ReLU es la que comúnmente se utiliza [5] y se define por:

$$g(\mathbf{x}) = max\ (0, \mathbf{x}) \tag{1.52}$$

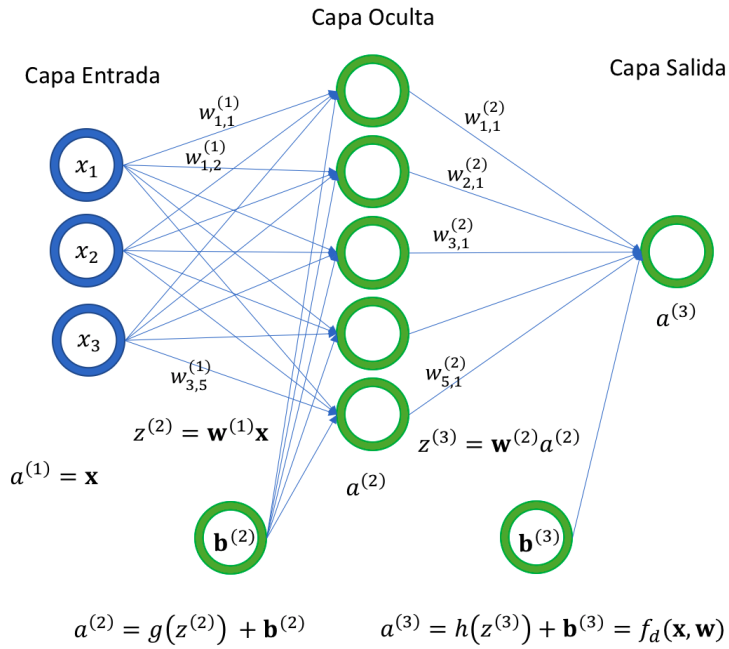


Figura 1.4. Paso hacia adelante -feed-forward- para un perceptrón multicapa. Las entradas se multiplican con los pesos $\mathbf{w}$ y se mapean por la función de transferencia $g()$ en la capa oculta. Luego se multiplican por los pesos de la capa de salida y finalmente se mapean por la función lineal $h()$.

En la primera etapa, se realiza una combinación lineal entre las entradas $\mathbf{x}$ y los pesos iniciales $\mathbf{w}$ -capa de entrada-, esta combinación se mapea por la función ReLU en la o las capas ocultas de la red neuronal. El resultado se vuelve a combinar con los pesos

[5] Existen varias funciones de activación, por ejemplo, la función sigmoide, la función tangente hiperbólica *tanh*, más detalles en [Géron, 2019].

de salida y se mapean por la función lineal $h(x)$, obteniendo la primera aproximación f_d. A la salida del perceptrón multicapa se calcula la perdida E y su gradiente $\nabla E(\mathbf{w})$ que, en un paso adicional, se propaga desde la capa de salida a la capa de entrada y actualiza los valores de los pesos $\mathbf{w}$ en cada capa hasta que se encuentre el valor óptimo -mínimos de la función E-. Este procedimiento es el famoso algoritmo *backpropagation*, que se muestra la figura 1.5. [Rumelhart et al., line].

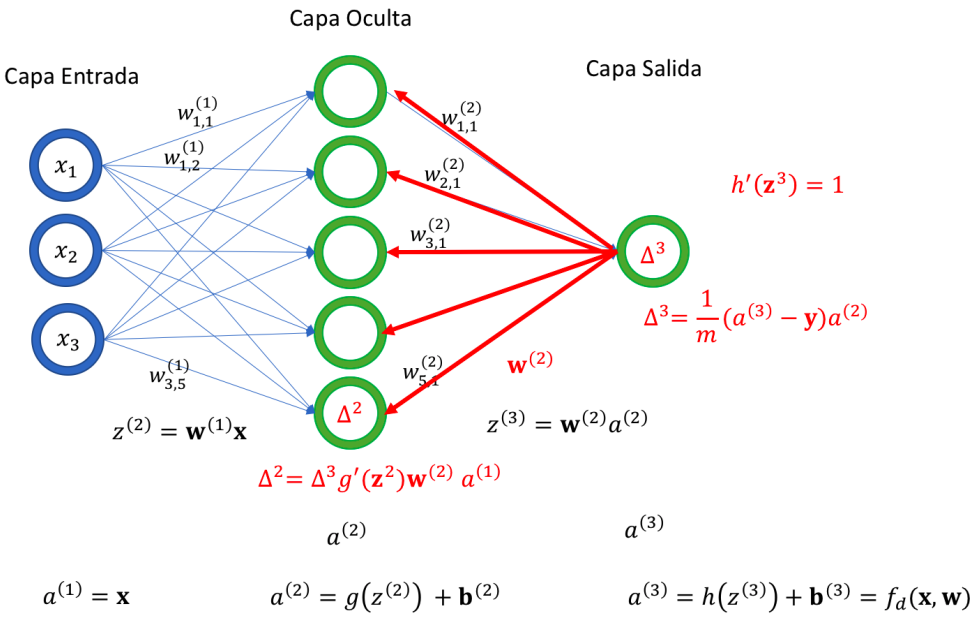


Figura 1.5. representación gráfica de la técnica de backpropagation. Se dan valores iniciales al vector $\mathbf{w}$ para realizar la propagación hacia adelante, para luego calcular Δ^l y calcular los errores en sentido contrario o hacia atrás de la red neuronal y obtener los valores actualizados de $\mathbf{w}$.

La actualización de los pesos de la red neuronal están dados por:

$$w_0 := w_0 - \alpha \nabla E(\mathbf{w}) \tag{1.53}$$

Donde α es la *razón de aprendizaje* y controla la magnitud con la que se actualizan los pesos $\mathbf{w}$. En cada actualización, el vector $\mathbf{w}$ se mueve en la dirección de la mayor tasa de disminución de la función de pérdida $E(\mathbf{w})$, esta técnica se utiliza como algoritmo de optimización genérico y se conoce como *gradiente descendiente* [Bishop, 2006]. El algoritmo perceptrón multicapa con función de activación ReLU fue entrenado y validado

con la función *layer_dense* del API *keras* para Python 3.7. [Chollet, 2015]. Después de varias pruebas, se estableció la siguiente arquitectura para MLP y todos los algoritmos de aprendizaje profundo utilizados en este estudio.

- capas ocultas: $\{1, 2\}$
- neuronas: $\{50, 100, 300, 400, 500, 1000\}$
- epochs: 500
- repeticiones: 10

Hay un par de algoritmos basados en el perceptrón multicapa que se utilizaron en los estudios de pronósticos mencionados - [Makridakis et al., 2018b, Hastie et al., 2001]- y que también se utilizaron en este estudio. Primero, el algoritmo **Red Neuronal con Función de Base Radial** se planteo como una alternativa a MLP. RBF utiliza n funciones de base radial -kernel Gaussiano- como función de activación [Moody and Darken, 1989], la predicción para datos nuevos x' viene dada por:

$$f_d(x') = \sum_{i=1}^{N} w_i \exp\Big(- \frac{\|x_i - c_i\|^2}{\gamma^2} \Big) \tag{1.54}$$

Donde w_i corresponden a los pesos de la red, γ corresponde al ancho de la función kernel que determina la suavidad del ajuste y c_i es el centro de la función kernel para la unidad i. La Red de función de base radial utiliza los mismos algoritmos de optimización que MLP y se programó con la función *layer_dense* del API *keras* para Python 3.7.

red neuronal bayesiana regularizada (BRNN)

El algoritmo Red Neuronal Bayesiana Regularizada o *Bayesian Regularized Neural Network (BRNN)* propuesto por David MacKay en [MacKay, 1992a, MacKay, 1992b] propone un enfoque bayesiano para la regularización y comparación de modelos basados en redes neuronales artificiales -puntualmente el perceptron multicapa o red neuronal *feedforward*-. Los parámetros **w** en un perceptrón multicapa -visto en la sección anterior-

se obtienen minimizando la función de perdida [6],

$$E_D(D|\mathbf{w},\mathcal{A}) = \sum_{i=1}^{N} \frac{1}{2}(f_d(x_i,\mathbf{w},\mathcal{A}) - y(x_i))^2 \tag{1.55}$$

Donde D es el conjunto de datos y $\mathcal{A}$ son un conjunto de arquitecturas plausibles de una red neuronal y utilizar un término de regularización, que evita el sobreajuste del modelo. Este se define por,

$$E_W(\mathbf{w}|\mathcal{A},R) = \frac{1}{2}\|\mathbf{w}\|^2 \tag{1.56}$$

Donde $\mathcal{R}$ son alternativas de regularizaciones. La función objetivo a optimizar se plantea como la suma de la Eq. 1.55 y Eq. 1.56,

$$M = \alpha\ E_W(\mathbf{w}|\mathcal{A},\mathcal{R}) + \beta\ E_D(D|\mathbf{w},\mathcal{A}) \tag{1.57}$$

α es una medida de cuan suave se espera sea el modelo f_d -constante de regularización- y β es una medida del ruido gaussiano aditivo con varianza $\sigma^2 = \frac{1}{\beta}$ incluido en el conjunto de datos. Cada modelo plausible $\mathcal{A}$ se define por su arquitectura y por la distribución de probabilidad apriori $P(\mathbf{w}|\alpha,\mathcal{A})$ que representa *nuestro conocimiento de los pesos* $\mathbf{w}$ *antes de que los datos sean obtenidos*. Usando la regla de bayes, la probabilidad posterior de los parámetros $\mathbf{w}$ es,

$$P(\mathbf{w}|\mathcal{D},\alpha,\beta,\mathcal{A},\mathcal{R}) = \frac{P(\mathcal{D}|\mathbf{w},\beta,\mathcal{A})\ P(\mathbf{w}|\mathcal{A},\mathcal{R},\alpha)}{P(\mathcal{D}|\alpha,\beta,\mathcal{A},\mathcal{R})} \tag{1.58}$$

En palabras,

$$\text{Posterior} = \frac{\text{Verosimilitud}\ \times\ \text{Priori}}{\text{Evidencia}} \tag{1.59}$$

[6]En esta sección se utilizará la notación matemática propuesta por el autor en [MacKay, 1992a, MacKay, 1992b]

$P(\mathcal{D}|\mathbf{w},\beta,\mathcal{A})$ es la verosimilitud -*likelihood*- o la probabilidad de la ocurrencia de los datos $\mathcal{D}$ dados los pesos $\mathbf{w}$. Se asume que el ruido en el conjunto de datos es gaussiano con media cero y desviación estándar σ^2, por lo tanto, la verosimilitud de $\mathcal{D}$ dados los parámetros $\mathbf{w}$ es,

$$P(\mathcal{D}|\mathbf{w},\beta,\mathcal{A}) = \frac{\exp\left(-\beta E_D\right)}{Z_D(\beta)} \tag{1.60}$$

Donde $\beta = \frac{1}{\sigma^2}$ y la distribución a priori para los parámetros $\mathbf{w}$ es,

$$P(\mathbf{w}|\mathcal{A},\mathcal{R},\alpha) = \frac{\exp\left(-\alpha E_W\right)}{Z_W(\alpha)} \tag{1.61}$$

Donde, $Z_D(\beta) = (\pi/\beta)^{N/2}$ y $Z_W(\alpha) = (\pi/\alpha)^{N/2}$. $P(\mathcal{D}|\alpha,\beta,\mathcal{A},\mathcal{R})$ es la evidencia o factor de normalización que en la elección de los parámetros $\mathbf{w}$, se ignora. Sustituyendo en la posterior -Ec.1.58-; obtenemos,

$$P(\mathbf{w}|\mathcal{D},\alpha,\beta,\mathcal{A},\mathcal{R}) = \frac{\exp\left(-(\beta E_D + \alpha E_W)\right)}{Z_W(\alpha)Z_D(\beta)} \tag{1.62}$$

$$P(\mathbf{w}|\mathcal{D},\alpha,\beta,\mathcal{A},\mathcal{R}) = \frac{\exp\left(-M(\mathbf{w})\right)}{Z_M(\alpha,\beta)} \tag{1.63}$$

Los pesos óptimos se obtienen maximizando la probabilidad a posterior $P(\mathbf{w}|\mathcal{D},\alpha,\beta,\mathcal{A},\mathcal{R})$.

Obtención de α y β

Ahora se considera la regla de Bayes para optimizar los parámetros α y β de la función objetivo,

$$P(\alpha,\beta|\mathcal{D},\mathcal{A},\mathcal{R}) = \frac{P(\mathcal{D}|\alpha,\beta,\mathcal{A},\mathcal{R})\ P(\alpha,\beta)}{P(\mathcal{D}|\mathcal{A},\mathcal{R})} \tag{1.64}$$

La maximización de la distribución posterior se obtiene maximizando la función de verosimilitud $P(\mathcal{D}|\alpha,\beta,\mathcal{A},\mathcal{R})$ -asumiendo una distribución a priori uniforme $P(\alpha,\beta)$

para los parámetros α y β-. Además, esta función de probabilidad es la evidencia de la Ec.1.58 y se puede resolver utilizando dicha ecuación,

$$P(\mathcal{D}|\alpha,\beta,\mathcal{A},\mathcal{R}) = \frac{P(\mathcal{D}|\mathbf{w},\beta,\mathcal{A},\mathcal{R})\ P(\mathbf{w}|\alpha,\mathcal{A},\mathcal{R})}{P(\mathbf{w}|\mathcal{D},\alpha,\beta,\mathcal{A},\mathcal{R})} \tag{1.65}$$

$$P(\mathcal{D}|\alpha,\beta,\mathcal{A},\mathcal{R}) = \frac{Z_M(\alpha,\beta)}{Z_D(\beta)Z_W(\alpha)}\frac{\exp\left(-\beta E_D - \alpha E_W\right)}{\exp\left(-M(\mathbf{w})\right)} = \frac{Z_M(\alpha,\beta)}{Z_D(\beta)Z_W(\alpha)} \tag{1.66}$$

Se puede estimar $Z_M(\alpha,\beta)$ por expansión en serie de Taylor, dado que la función objetivo tiene una forma cuadrática en un área que rodea el valor mínimo de la posterior $\mathbf{w}_{MP}$, donde el gradiente es cero. Así, $Z_M(\alpha,\beta)$ se aproxima de la forma,

$$Z_M \approx (2\pi)^{N/2}(det((\mathbf{H}_{MP})^{-1}))^{1/2}\exp\left(-M(\mathbf{w}_{MP})\right) \tag{1.67}$$

Donde $\mathbf{H} = \beta\nabla^2 E_D + \alpha\nabla^2 E_W$ es la matriz hessiana de la función objetivo. Reemplazando en la ecuación 1.67, se puede resolver para los valores óptimos -mínimos- de α y β calculando las derivadas de la ecuación 1.66 e igualando a cero. Obteniendo,

$$\alpha_{MP} = \frac{\gamma}{2E_W(\mathbf{w}_{MP})} \tag{1.68}$$

y

$$\beta_{MP} = \frac{n-\gamma}{2E_D(\mathbf{w}_{MP})} \tag{1.69}$$

Donde $\gamma = N - 2\alpha_{MP}tr(\mathbf{H}_{MP})^{-1}$ y N es el total numero de parámetros de la red neuronal. El parámetro γ es la medida de cuantos parámetros en la red neuronal son efectivamente usados para reducir la función de error. Este puede variar de cero a N. La optimización bayesiana de los parámetros de regularización requieren del cálculo de la matriz Hessiana de $M(\mathbf{w})$ en el mínimo $\mathbf{w}_{MP}$. El paquete *brnn* utiliza la aproximación

Gauss-Newton para obtener la matriz Hessiana $\mathbf{H} = \nabla^2 F(\mathbf{w}) \approx 2\beta\mathbf{J}^T\mathbf{J} + 2\alpha\mathbf{I}_N$, donde $\mathbf{J}$ es la matriz jacobiana del error del conjunto de entrenamiento. Para ver más en detalle los argumentos del modelo y la inferencia bayesiana, ver [MacKay, 1992a] y [MacKay, 1992b]. El modelo de red neuronal bayesiana implementado para los pronósticos de ventas fue entrenado y validado con la función *brnn* del paquete *brnn* en conjunto con el paquete *caret* para R.

1.8.2. redes neuronales recurrentes

Los problemas como traducción automática, reconocimiento de voz que en general se denominan problemas de *procesamiento natural del lenguaje*, necesitan que las redes neuronales puedan guardar la información importante a través del tiempo. Para esto le agregaron a las redes neuronales artificiales conexiones que incluyen *loops* o retroalimentaciones como muestra la figura 1.6. La retroalimentación permite que las redes neuronales artificiales *tengan memoria* y puedan capturar las relaciones temporales en los datos. Además de manejar y predecir secuencias con diferentes dimensiones -vectores de entrada y salida de diferentes tamaños- [Sutskever et al., 2014, Sun and Giles, 2001].

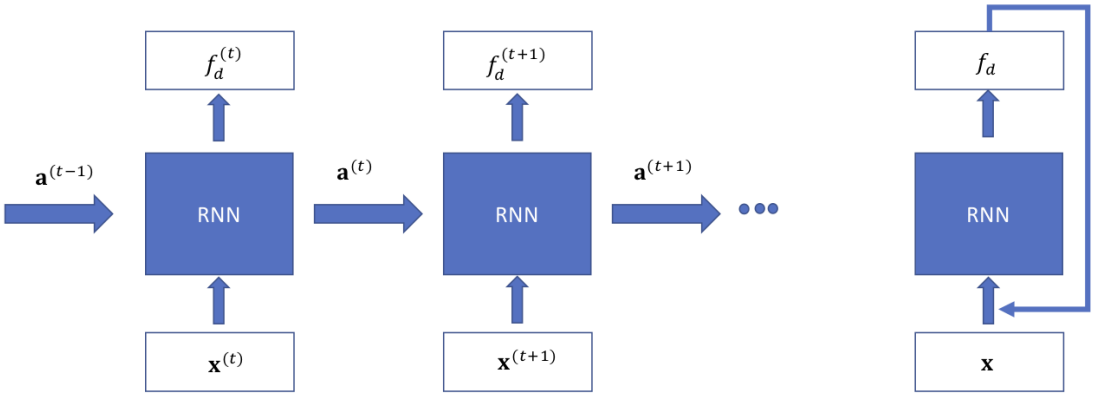


Figura 1.6. Representación gráfica de una Red Neuronal Recurrente

En el paso hacia adelante o propagación hacia adelante de una red neuronal recurrente, la activación o estado $\mathbf{a}^{(t)}$ en el momento t para una secuencia $\{\mathbf{x}^{(t-n)}, ..., \mathbf{x}^{(t-1)}, \mathbf{x}^{(t)}\}$ contiene información sobre los pasos de tiempo pasados -por ejemplo, desde 0 a t de la

secuencia de entrada-, que viene dada por,

$$\mathbf{a}^{(t)} = g(\mathbf{w}_{ax}[\mathbf{a}^{(t-1)}, \mathbf{x}^{(t)}] + b_a) \tag{1.70}$$

Donde $\mathbf{w}_{ax}$ son pesos de entrada, $\mathbf{a}^{(t-1)}$ es la activación en $t-1$ y $g()$ es la función de activación ReLU. La predicción en el tiempo t va a depender de la entrada $\mathbf{x}$ y de la activación del paso de tiempo anterior $\mathbf{a}^{(t-1)}$,

$$f_d^{(t)} = l(\mathbf{w}_{ya}\ \mathbf{a}^{(t)}) \tag{1.71}$$

Donde $\mathbf{w}_{ya}$ son pesos de salida, $l()$ es una función de salida lineal. Para cada paso de tiempo se realiza una copia de la red que requiere más memoria a medida que se agranda la red y se aumenta la cantidad de pasos de tiempo. La cantidad de pasos de tiempo en la red neuronal recurrente, corresponderá a la cantidad de derivadas que se calcularán para actualizar los parámetros. Para secuencias muy largas o redes neuronales muy profundas, los gradientes en la etapa de entrenamiento pueden alcanzar valores muy grandes -*explosión del gradiente*- o valores muy pequeños -*desvanecimiento del gradiente*- provocando entrenamientos inestables y lentos. Produciendo modelos poco fiables. Para resolver estos problemas denominados de *degradación del gradiente*, se puede establece un valor de umbral para el gradiente -*recorte del gradiente*- o se puede especificar una cantidad fija de pasos de tiempo para actualizar los pesos, evitando numerosos cálculos y la posibilidad de que el gradiente se desvanezca -Algoritmo de retro-propagación truncado a través del tiempo, consulte [Williams and Zipser, 1995] para obtener más información-. A continuación se describen las redes GRU y LSTM que se desarrollaron como mejoras de las redes neuronales recurrentes para abordar el problema de *explosión y fuga del gradiente* y se han destacado en el área de procesamiento natural del lenguaje -NLP-. Son redes neuronales con arquitecturas más complejas a comparación del MLP por lo que su costo computacional es mayor. Su enfoque del manejo de datos de forma

secuencial se ha aplicado para problemas de pronósticos de series de tiempo y en función de las diferentes configuraciones de los datos, se pueden obtener buenos resultados.

1.8.3. unidad recurrente con compuerta (GRU)

La Unidad Recurrente con Compuerta o *Gated Recurrent Unit* tiene una modificación que mejora la captura a largo plazo de secuencias y ayuda a resolver los problemas del gradiente [Chung et al., 2014]. En particular, la GRU tiene una celda $\mathbf{c}$[7] que le proporciona memoria a través de la compuerta Γ_u. De manera gráfica la celda $\mathbf{c}$ se puede representar por la figura 1.7,

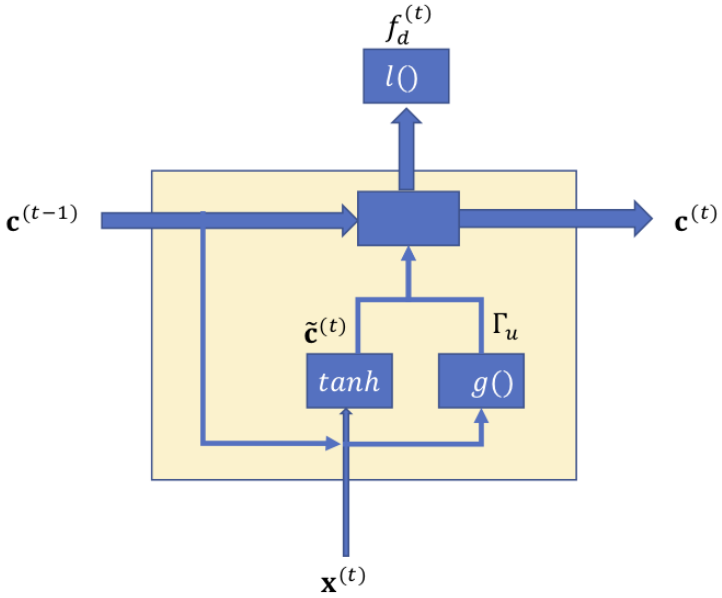


Figura 1.7. Unidad Recurrente con Compuerta

la celda $\mathbf{c}$ se define por,

$$\mathbf{c}^{(t)} = (1 - \Gamma_u)\mathbf{c}^{(t-1)} + \Gamma_u \tilde{\mathbf{c}}^{(t)} \tag{1.72}$$

[7]Considerar $\mathbf{c}^{(t)} = \mathbf{a}^{(t)}$, es decir, la celda de memoria $\mathbf{c}$ es igual al valor de activación $\mathbf{a}$. La celda o unidad de memoria empieza a realizar más funciones y cálculos.

Donde $\tilde{\mathbf{c}}^{(t)}$ será un candidato para actualizar el valor de la celda, dado por,

$$\tilde{\mathbf{c}}^{(t)} = \tanh(\mathbf{w}_c[\mathbf{c}^{(t-1)}, \mathbf{x}^{(t)}] + b_c) \tag{1.73}$$

Donde $\mathbf{w}_c$ son los pesos en la celda. $\mathbf{c}^{(t-1)}$ es el valor de la celda en el tiempo $t-1$. La compuerta de actualización Γ_u toma valores cero o uno y *decide* la actualización de la celda,

$$\Gamma_u = g(\mathbf{w}_u[\mathbf{c}^{(t-1)}, \mathbf{x}^{(t)}] + b_u) \tag{1.74}$$

Donde $\mathbf{w}_u$ son los pesos de la compuerta, g es la función de activación. Como se observa, si $\Gamma_u = 1$ el valor de $\mathbf{c}$ se actualizará por $\tilde{\mathbf{c}}$. Esto permitirá que la red neuronal retenga los valores pasados importantes de la secuencia y al mismo tiempo evita el problema de fuga de gradiente. La predicción final f_d en el momento t es

$$f_d^{(t)} = l(\mathbf{w}_{yc}\ \mathbf{c}^{(t)}) \tag{1.75}$$

Donde $\mathbf{w}_{yc}$ son pesos de salida, $l()$ es una función de salida lineal. Es común agregar otra compuerta que determina la relevancia de $\mathbf{c}^{(t)}$ para calcular el siguiente valor candidato de la celda (compuerta Γ_r). En resumen, podemos definir la red GRU por,

$$\begin{aligned}
\tilde{\mathbf{c}}^{(t)} &= \tanh(\mathbf{w}_c[\Gamma_r \mathbf{c}^{(t-1)}, \mathbf{x}^{(t)}] + b_c) \\
\Gamma_r &= \sigma(\mathbf{w}_r[\mathbf{c}^{(t-1)}, \mathbf{x}^{(t)}] + b_r) \\
\Gamma_u &= \sigma(\mathbf{w}_u[\mathbf{c}^{(t-1)}, \mathbf{x}^{(t)}] + b_u) \\
\mathbf{c}^{(t)} &= \Gamma_u\ \tilde{\mathbf{c}}^{(t)} + (1 + \Gamma_u) + \mathbf{c}^{(t-1)}
\end{aligned} \tag{1.76}$$

El modelo de Unidad Recurrente con Compuerta para el pronóstico de ventas se entrenó y validó con la función *layer_gru* del API *keras* para Python 3.7. [Chollet, 2015]. Un paso importante antes de utilizar la red GRU, y que se realiza también para la red neuronal LSTM y la red neuronal convolucional (ConvNet) - que se verán a continuación - es

que los datos de entrada deben ser tridimensional. Las tres dimensiones de los datos de entrada son:

- **Muestras**. Una secuencia es una muestra. Un lote se compone de una o más muestras.
- **Pasos de tiempo**. Un paso de tiempo es un punto de observación en la muestra. Una muestra es compuesta de múltiples pasos de tiempo.
- **Características**. Una característica es el conjunto de variables en el conjunto de datos. Un paso de tiempo se compone de una o más características.

Esta estructura tridimensional de los datos de entrada a menudo se resume utilizando la notación matricial: *[muestras, pasos de tiempo, características]*. A diferencia de los anteriores algoritmos ML vistos; los datos de entrada son bidimensional: *[muestras, características]*. Esto significa que estamos agregando la nueva dimensión de *pasos de tiempo*.

1.8.4. red con memoria a largo plazo (LSTM)

La Red de memoria a largo plazo o LSTM -*Long Short Term Memory*- es un tipo de red neuronal recurrente que tiene la capacidad de recordar y actualizar el estado de la celda -*c*- a través de las compuertas Γ; a diferencia de la Unidad Recurrente con Compuerta que se vio en la sección anterior, la red LSTM tiene tres compuertas para controlar el cambio de estado y la adición de información que fluye a través de la celda. LSTM puede aprender las dependencias a largo plazo entre los pasos de tiempo de la secuencia de manera más efectiva, incluso mejor que la red GRU [Hochreiter and Schmidhuber, 1997]. En LSTM se incluye la activación $\mathbf{a}()$ y el estado de la celda $\mathbf{c}()$, definidos por,

$$\mathbf{a}^{(t)} = \Gamma_o \ \tanh(\mathbf{c}^{(t)}) \tag{1.77}$$

El estado de la celda en el tiempo t viene dado por,

$$\mathbf{c}^{(t)} = \Gamma_f \ \mathbf{c}^{(t-1)} + \Gamma_u \ \tilde{\mathbf{c}}^{(t)} \tag{1.78}$$

Esto le da a la celda de memoria la opción de mantener el valor anterior de la celda -$\mathbf{c}^{(t-1)}$- y agregar el nuevo valor $\tilde{\mathbf{c}}^{(t)}$, dado por,

$$\tilde{\mathbf{c}}^{(t)} = \tanh(\mathbf{w}_c[\mathbf{a}^{(t-1)}, \mathbf{x}^{(t)}] + b_c) \tag{1.79}$$

Las compuertas están definidas por,

- Compuerta de Olvido (Γ_f)

$$\Gamma_f = \sigma(\mathbf{w}_f[\mathbf{a}^{(t-1)}, \mathbf{x}^{(t)}] + b_f) \tag{1.80}$$

- Compuerta de entrada o de actualización (Γ_u)

$$\Gamma_u = \sigma(\mathbf{w}_u[\mathbf{a}^{(t-1)}, \mathbf{x}^{(t)}] + b_u) \tag{1.81}$$

- Compuerta de salida (Γ_o)

$$\Gamma_o = \sigma(\mathbf{w}_o[\mathbf{a}^{(t-1)}, \mathbf{x}^{(t)}] + b_o) \tag{1.82}$$

Las compuertas cumplirán la función de actualizar, retener o memorizar información y determinar la salida de la celda con respecto a la activación $\mathbf{a}^{(t-1)}$ y la entrada $\mathbf{x}^{(t)}$. Es común en una red LSTM agregar las *conexiones de mirilla* en las compuertas permitiendo que las compuertas *miren* el estado $\mathbf{c}^{(t-1)}$ [Gers and Schmidhuber, 2000]. Además, las compuertas y el flujo de datos constante permite que la celda tenga un flujo constante de error (*el carrusel de error constante* o CEC) evitando los problemas de explosión o desvanecimiento del gradiente [Sutskever, 2013]. En la figura 1.8, se representa una

celda LSTM.

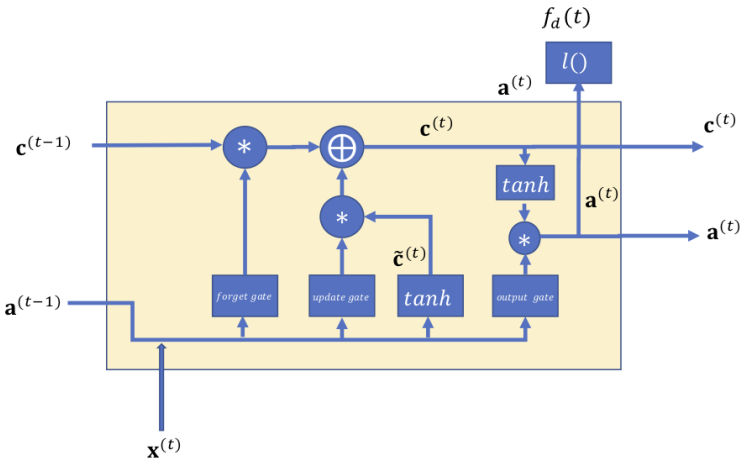


Figura 1.8. Celda LSTM

Al usar TBPTT -*Backpropagation Truncado a través del tiempo*- para la etapa de entrenamiento, se obtiene la predicción f_d,

$$f_d^{(t)} = l(\mathbf{w}_{ya}\ \mathbf{a}^{(t)}) \tag{1.83}$$

Se implementó la red LSTM para pronosticar las ventas de un año usando la función *layer_lstm* del API *keras* para Python 3.7. Los datos de entrada deben tener la forma tridimensional: *[muestras, pasos de tiempo, características].*

1.8.5. red neuronal convolucional (ConvNet)

La visión por computadora funciona al procesar imágenes o videos que pueden alcanzar dimensiones muy altas -millones de parámetros-, lo que hace que los recursos computacionales sean muy extensos y puedan causar los problemas del gradiente descritos en las secciones anteriores. Las redes neuronales convolucionales se propusieron como una alternativa de mejora, que ha tenido excelentes resultados en el reconocimiento de rostros, traducción de textos, reconocimiento de voz y automóviles autónomos [Rawat and Wang, 2017, Wolf et al., 2011, Sun et al., 2014, LeCun et al., 2010]. Puntualmente, las ConvNets utilizan la operación de *convolución* para capturar patrones espaciales invariantes en

una imagen, disminuyendo el número de parámetros en las capas ocultas, evitando los problemas del gradiente y disminuyendo el costo computacional del procesamiento [Gross et al., 2017]. Las convoluciones para un conjunto de imágenes, se definen por dos parámetros:

- *Tamaño de los parches extraídos de las entradas:* generalmente son 3×3 o 5×5.
- *Profundidad del mapa de características de salida:* la cantidad de filtros calculados por la convolución.

La convolución funciona deslizando estas ventanas o parches de tamaño 3×3 o 5×5 sobre el *mapa de características de entrada* 3D y extrayendo un parche de la forma *(altura, anchura, profundidad de entrada)*. Cada parche 3D se transforma en un vector 1D de la forma *(profundidad de salida,)* a través de un producto tensorial [8] con la matriz de pesos de los filtros -llamado kernel de convolución-. Todos estos vectores se vuelven a ensamblar espacialmente en un *mapa característico de salida* 3D de la forma *(altura, anchura, profundidad de salida)*. Cada ubicación espacial en el mapa característico de salida corresponde a la misma ubicación en el mapa de características de entrada. Tenga en cuenta que el ancho y la altura de salida pueden diferir del ancho y la altura de entrada. Difieren por la distancia que se mueve la ventana a través del conjunto de datos, denominada zancadas -*Strides*- $s^{(l)}$ y el Relleno -*Padding*- $p^{(l)}$ que agrega información, rellenando el mapa de características de entrada, ya que en el cálculo de la convolución, muchos datos se superponen en el área central de los datos y la información de los extremos se pierde.

Aprovechando la operación de convolución para obtener la relación temporal en series de tiempo, el enfoque de las ConvNets se puede implementar para el procesamiento

[8] Es común en la literatura de visión en computadora y en aprendizaje profundo referirse a los conjuntos de datos como tensores debido a sus dimensiones. Por ejemplo, las imágenes son tensores 3D, con dos ejes espaciales (altura y anchura) y un eje de profundidad -también llamado eje de canales-. Para una imagen RGB, la dimensión del eje de profundidad es 3, porque la imagen tiene tres canales de color: rojo, verde y azul. Para una imagen en blanco y negro, la profundidad es 1 (niveles de gris).

de secuencias y pronósticos [van den Oord et al., 2016, Borovykh et al., 2017]. Particularmente, el tiempo puede tratarse como una dimensión espacial, como la altura o el ancho de una imagen 2D. De la misma manera, se pueden utilizar convoluciones 1D, extrayendo parches locales 1D o subsecuencias de las series de tiempo, como muestra la figura 1.9.

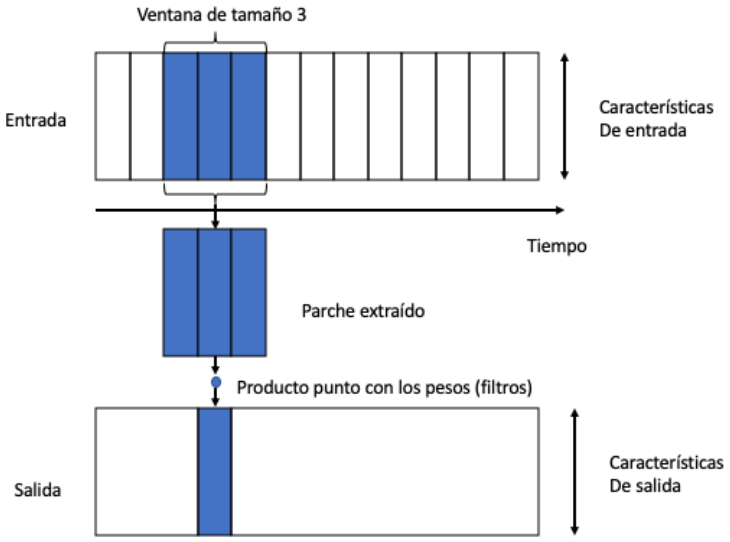


Figura 1.9. Convolución 1D, se obtiene un parche de la ventana de las entradas, en este caso, una ventana de tamaño 3

Las capas de convolución 1D toma como entrada los tensores 3D con forma *(muestras, tiempo, características)* y devuelve tensores 3D de forma similar. La ventana de convolución es una ventana 1D en el eje temporal -eje 1 en el tensor de entrada-. Para problemas de pronósticos de series de tiempo y en particular para este estudio de pronósticos de ventas se extrajeron secuencialmente los parches de las series de tiempo y se convolucionan con los filtros $f^{(l)}$, para obtener el mapa característico de salida o la activación de la capa l, dada por,

$$\mathbf{a}^{(l)} = f_n^{(l)}\ \mathbf{x} \tag{1.84}$$

Y la salida, esta dada por,

$$f_d^{(l)} = g(\mathbf{a}^{(l)}) \tag{1.85}$$

Se implementó una capa convolucional con la función *layers_Conv1D()*, una capa de agrupamiento *layers_MaxPooling1D* que extrae parches 1D -subsecuencias- de una entrada y generar el valor máximo -agrupación máxima- o el valor promedio -agrupación promedio-, esto se usa para reducir la longitud de las entradas 1D -submuestreo-. Se implementaron 500 épocas y la función de activación RElu del API *keras* para Python 3.7. Los datos de entrada deben tener la forma tridimensional: *[muestras, pasos de tiempo, características]*.

Las redes neuronales artificiales bajo el enfoque de aprendizaje profundo se caracterizan por ser fáciles de definir y ajustar, pero sigue siendo un reto lograr un buen rendimiento en el modelado predictivo. El principal problema, el sobreajuste. En los últimos años se han propuesto varios métodos de regularización, entre las que se destaca la regularización *Dropout* y que en conjunto con la *restricción de pesos*, proporcionan una mejora significativa en el desempeño de las redes neuronales profundas [Srivastava et al., 2014].

1.9. regularización para redes neuronales artificiales

La reducción de la capacidad de un modelo reduce la probabilidad de que el modelo se sobreajuste en el conjunto de entrenamiento, hasta el punto en que ya no se sobreajuste. En el caso de las redes neuronales artificiales, la complejidad se puede variar por: i) el cambio en el número de parámetros adaptativos en la red y ii) el uso de regularización, que implica la adición de un término que penaliza la función de error en proporción al tamaño de los pesos en el modelo. Los parámetros *pequeños* sugieren un modelo menos complejo y, a su vez, más estable y menos sensible a las fluctuaciones en los datos de

entrada -como se vio en la sección 1.4.1-.

En los últimos 10 años se ha visto el desarrollo y la adopción de configuraciones modernas de redes neuronales y técnicas de regularización que ofrecen beneficios en términos del rendimiento de generalización, configurabilidad y/o complejidad computacional. Entre estas se destacan la regularización usando *paradas tempranas*, *Dropout* y la *restricción de peso*, que describimos a continuación.

1.9.1. Paradas Tempranas

Un desafío importante en el entrenamiento de redes neuronales artificiales es cuánto tiempo entrenarlas. Demasiado poco entrenamiento significará que el modelo se ajustará *pobre* a los conjuntos de entrenamiento y validación. Demasiado tiempo de entrenamiento significará que el modelo se sobreajustará a los datos de entrenamiento y el rendimiento en el conjunto de validación se degradará después de unas pocas épocas.

La idea detrás de la técnica de *paradas tempranas*, es entrenar el modelo en una gran cantidad de épocas de entrenamiento y durante el este evaluar el modelo en el conjunto de validación después de cada época. Si el rendimiento del modelo en el conjunto de validación comienza a degradarse -por ejemplo, el error comienza a aumentar o la precisión comienza a disminuir-, entonces el proceso de entrenamiento se detiene [Prechelt, 1996]. Este enfoque simple y efectivo para entrenar redes neuronales es probablemente la forma de regularización más utilizada en el aprendizaje profundo [Chollet, 2017].

Hay tres elementos que se utilizan para implementar las paradas tempranas:

- Monitoreo del desempeño del modelo. El rendimiento del modelo se evalúa en el conjunto de validación al final de cada época.
- Disparador para dejar de entrenar. En el caso más simple, el entrenamiento se detiene tan pronto como el rendimiento en el conjunto de validación disminuye en comparación con el rendimiento en el conjunto de validación de la época de entrenamiento anterior.

- La elección del modelo a utilizar. Cada vez que disminuye el error en el conjunto de validación, se almacena una copia de los parámetros del modelo. Cuando termina el entrenamiento, se devuelven estos parámetros.

Se implemento la regularización *paradas temprana* o *early stopping* para los modelos de redes neuronales vistos en este capítulo, usando la función *Callbacks* [9] del API *keras* para Python 3.7.

1.9.2. Dropout

Se sabe que las redes neuronales artificiales con diferentes configuraciones reducen el sobreajuste, pero requiere costo computacional adicional para el entrenamiento y mantenimiento de múltiples modelos. Sin embargo, se puede usar un solo modelo y simular tener una gran cantidad de arquitecturas de red diferentes al ignorar o abandonar aleatoriamente neuronas -o pesos- durante el entrenamiento. Esto se denomina *Dropout* y ofrece un método de regularización computacionalmente económico y efectivo para reducir el sobreajuste y el error de generalización en redes neuronales profundas de todo tipo.

Dropout se implementa en cada capa de una red neuronal y se define por un nuevo hiperparámetro que especifica la probabilidad a la que se *abandonan* los pesos de salida de las capas, o inversamente, la probabilidad a la que se *retienen* los pesos de las salidas de la capa [Srivastava et al., 2014]. Por ejemplo, si se retiene con probabilidad p durante el entrenamiento, los pesos de salida de esa capa se multiplican por p en el momento de la validación, disminuyendo aleatoriamente la cantidad de pesos. Un valor común es una probabilidad de 0.5 para retener la salida de cada neurona en una capa oculta y un valor de 0.8 para retener las entradas de una capa visible [Hinton et al., 2012]. Se

[9] Callbacks o *devolución de llamada* es un fragmento de código que se puede ejecutar en un punto específico durante el entrenamiento, como antes o después del entrenamiento, una época o un lote. Proporcionan una forma de ejecutar un código e interactuar con el proceso de entrenamiento del modelo automáticamente

implemento la regularización *Dropout* para todas las capas ocultas de los modelos de redes neuronales vistos en este capítulo, usando la función *layer_dropout* del API *keras* para Python 3.7.

1.9.3. Restricción de pesos

Un problema en el uso de una penalización es que, aunque fomenta la red hacia pesos más pequeños, no fuerza pesos más pequeños. Una red neuronal entrenada con penalización por regularización de pesos puede permitir grandes pesos, en algunos casos pesos muy grandes. A diferencia de la regularización de pesos, la restricción de pesos verifica el tamaño o la magnitud de los pesos y los escala para que todos estén por debajo de un umbral predefinido. Esta restricción obliga a que los pesos sean pequeños y se puedan usar en conjunto con configuraciones más agresivas, como tasas de aprendizaje muy grandes. Esta restricción se implementa en cada nodo dentro de una capa. Todos los nodos dentro de la capa usan la misma restricción y, a menudo, múltiples capas ocultas dentro de la misma red usarán la misma restricción. Esta restricción se aplica a la norma de los vectores de pesos y, de forma predeterminada, se calcula como la norma L2 -la raíz cuadrada de la suma de los valores cuadrados del vector de pesos-. Algunos ejemplos de restricciones de pesos que se utilizan comúnmente incluyen:

- Forzar que la norma del vector de pesos sea 1. En el API Keras, *unit_norm()*
- Limita el tamaño máximo de la norma del vector de pesos. En el API keras, *max_norm()*.
- Limita el tamaño mínimo y máximo de la norma del vector de pesos. En el API Keras, *min_max_norm*.

Se implemento la restricción de peso para todas las capas ocultas de los modelos de redes neuronales vistos en este capítulo, usando las funciones anteriores para Python 3.7. En la literatura es común encontrar buenos resultados utilizando las regularizaciones

antes descritas en conjunto. Además de mejorar el desempeño de las redes neuronales en generales, también las hace más eficientes.

1.10. pronósticos de múltiples pasos con algoritmos ML

Hasta aquí se ha descrito el enfoque de los algoritmos de aprendizaje automático para obtener un modelo de regresión con el cual se puede obtener un pronóstico de *un paso adelante*. El pronóstico de un paso adelante $\hat{y}_{t+1}$ para una serie de tiempo se define por,

$$\hat{y}_{t+1} = f_d(y_t, ..., y_{t-n+1}) \tag{1.86}$$

$\hat{y}_{t+1}$ se obtiene con los n valores pasados de la serie de tiempo. En general, para pronósticos de múltiples pasos se han implementado métodos recursivos e iterativos. Y en específico para los métodos de aprendizaje automático se implementan los siguientes métodos:

- *Pronósticos recursivos* o iterativos: se obtienen al entrenar un modelo para un paso adelante dado por 1.86. Para el pronostico del siguiente *horizonte* ($h = 2$) se utiliza el mismo modelo y el pronóstico $\hat{y}_{t+1}$,

  $$\hat{y}_{t+2} = f_d(\hat{y}_{t+1}, y_t, ..., y_{t-n+1}) \tag{1.87}$$

  Consecutivamente se utiliza el valor pronosticado como parte de las variables de entrada para obtener los pronósticos de los siguientes h-pasos [Cheng et al., 2006],

  $$\hat{y}_{t+h} = f_d(\hat{y}_{t+h-1}, ..., \hat{y}_{t+1}, y_t, ..., y_{t-n+h}) \tag{1.88}$$

- *Pronósticos directo*: se entrenan h modelos, uno para cada pronostico con

$$\hat{y}_{t+h} = f_h(y_t, ..., y_{t-n+1}) \tag{1.89}$$

  La técnica *directa* no utiliza los pronósticos pasados para generar los nuevos pronósticos, no acumulando errores.

- La estrategia híbrida o *DirRec*: combina la técnica directa y recursiva. Puntualmente se calcula un modelo diferente incluyendo los pronósticos de pasos anteriores para cada horizonte h [Sorjamaa et al., 2007].

$$\hat{y}_{t+h} \begin{cases} f_d(y_t, ..., y_{t-n+1}) & \text{si} \quad h = 1 \\ f_h(\hat{y}_{t+h-1}, ..., \hat{y}_{t+1}, y_t, ..., y_{t-n+1}) & \text{si} \quad h \in (2, ..., h) \end{cases} \tag{1.90}$$

  Demanda el mayor costo computacional por la cantidad de modelos (h) que se entrenan [Cheng et al., 2006].

- Con el fin de mantener la dependencia estocástica entre los valores futuros, se creó la estrategia MIMO (*Multi-Inputs Multi-Outputs*). La estrategia MIMO, aprende un modelo con múltiples salidas a partir de la serie de tiempo,

$$\hat{y}_{t+h}, ...\hat{y}_{t+1} = F_d(y_t, ..., y_{t-n+1}) \tag{1.91}$$

Donde F_d corresponde al modelo propuesto con múltiples entradas y múltiples salidas. Generalmente, las múltiples salidas corresponderán al número de pronósticos futuros o el horizonte de pronósticos que se desean obtener. En el estudio de Makridakis, estos modelos de múltiples salidas se asocian solamente a redes neuronales con múltiples salidas. [Ben Taieb et al., 2010] argumentan que la estrategia MIMO, además de preservar las características estocásticas de la serie de tiempo, evita la condición de independencia de la estrategia directa y no acumula el error debido a la estrategia recursiva.

Los resultados que se obtienen dentro de la literatura de pronósticos con ML con las técnicas de pronósticos descritas anteriormente son diversos y no concluyentes. Se reportan pronósticos más precisos con la estrategia DirRec, cuando se han comparado con las técnicas Directa y Recursiva en [Sorjamaa et al., 2007]. En [Ji et al., 2005] se reporta que la técnica Directa es superior a la técnica recursiva. En [Makridakis et al., 2018b] se publicó que la técnica recursiva tiene mejores resultados para los diferentes horizontes de pronósticos cuando se utiliza el algoritmo MLP. En este estudio, además de obtener los pronósticos con la técnica directa, recursiva y MIMO para los algoritmos de aprendizaje profundo, se incluye la técnica DirRec. En el siguiente capítulo se describen brevemente la contraparte de los métodos de pronósticos con ML, los métodos estadísticos para pronósticos.

Capítulo 2

Modelos estadísticos para pronósticos

Pronósticos, por décadas, han sido una tarea estadística para la toma de decisiones sobre la programación de la producción, transporte y personal, y una guía para la planificación estratégica de una organización. Como se planteó, el desafío es obtener pronósticos precisos para horizontes h de corto, mediano y largo plazo, denominados pronósticos de múltiples pasos -*multi-step-ahead forecasting*-. Puntualmente, abordando pronósticos multi-pasos con series de tiempo.

En este capítulo se describirán brevemente los principales métodos estadísticos para problemas de pronósticos. Se expondrán las principales ideas detrás de los métodos de suavizado exponencial, la metodología Box-Jenkins aplicada al famoso modelo ARIMA para el ajuste y pronósticos de series de tiempo. Y finalmente se describe el método Theta que se impuso en varias categorías de la competencia M3 [Hyndman and Billah, 2003]. Se toma como referencia el trabajo de Hyndman y Athanasopoulos que se puede consultar para mayor detalle en: Forecating: Principles and Practice.

2.1. pronósticos de series de tiempo

Desde el enfoque estadístico, una serie de tiempo $[y_1, y_2, ..., y_t]$ se considera como una realización de un proceso estocástico; descrita como una familia de variables aleatorias indexadas en el tiempo y caracterizada por su distribución de probabilidad que se estima mediante su media y su varianza [Hyndman and Athanasopoulos, 2018]. El pronóstico $\hat{y}_t$ es el promedio de los posibles valores de la variable aleatoria y_t -o el valor promedio de la *distribución del pronóstico*- dado los datos previos de la serie de tiempo, por ejemplo,

$$\hat{y}_{t|t-1} = [y_t|y_{t-1}] \tag{2.1}$$

Significa, el pronóstico de y_t dado los valores previos $y_1, ...y_{t-1}$; que corresponden a la relación, hipótesis o función f_d encontrada para los problemas de regresión con los métodos de aprendizaje automático vistos en el capitulo anterior. Similarmente, los pronósticos de múltiples pasos adelante se escriben de la forma,

$$\hat{y}_{T+h|T} = [y_{T+h}|y_T] \tag{2.2}$$

Que significa, el pronóstico de h pasos por delante dada las observaciones hasta el tiempo T. Los métodos estadísticos de pronósticos basados en series de tiempo descomponen una serie de tiempo según su:

- *Tendencia (T)*, incremento o decrementos de los valores de los datos a largo plazo.
- *Estacionalidad (S)* o variaciones por periodos como meses o años. Es fija y de frecuencia conocida.
- *Ciclos(C)*, alzas o descensos en los datos con frecuencia desconocida. En [Hyndman and Athanasopoulos, 2018] la Tendencia y Ciclo se utilizan en conjunto y solo se denominan *tendencia*.
- Y una componente de *Residuo (R)* que contiene el resto de la información del

fenómeno.

Un ejemplo de las componentes de una serie de tiempo se muestra en la figura 2.1, utilizando la descomposición STL [1].

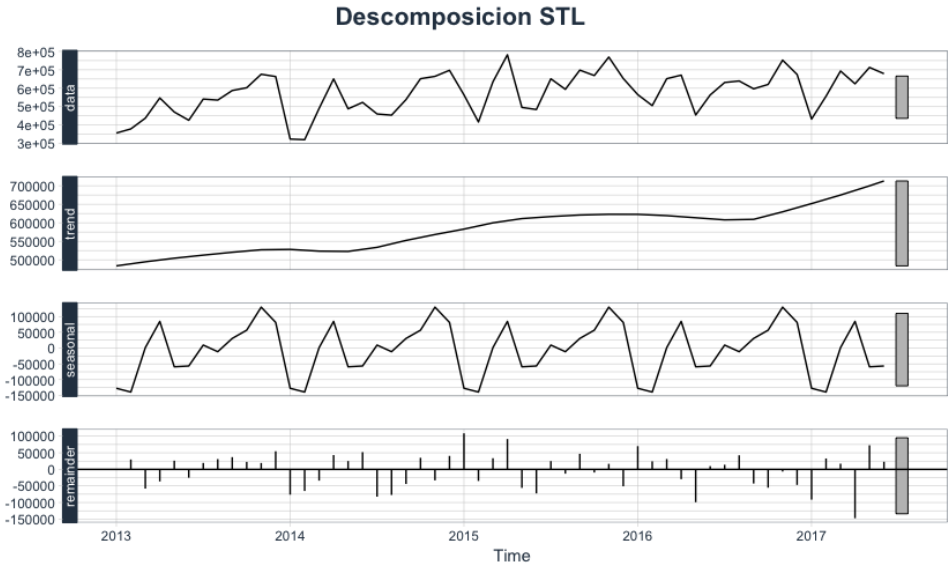


Figura 2.1. Un ejemplo de la descomposición de una serie de tiempo utilizando el método STL (Seasonal and Trend descomposition usando Loess) [Cleveland et al., 1990]. Desde arriba: serie original -*data*-, la componente de estacionalidad -*seasonal*-, tendencia -*trend*- y el residuo -*remainder*-). Ejemplo sacado de la librería stats de R.

Estas componentes se pueden combinar como un *modelo aditivo*,

$$y_t = T_t + S_t + R_t \tag{2.3}$$

Donde y_t es la serie de tiempo, T_t es la componente de tendencia, S_t es la componente de estacionalidad y R_t es la componente de residuo en el periodo t. Y estas componentes se pueden multiplicar, formando el modelo,

$$y_t = T_t * S_t * R_t \tag{2.4}$$

No solo la suma o la multiplicación de las componentes es posible, se pueden usar

[1] STL de -*Seasonal and Trend decomposition using Loess*-, se utiliza comúnmente en la descomposición de series de tiempo por ser más robusta y tener mayores ventajas con respecto a las descomposiciones SEATS, X11 y Promedio Móvil. Para detalles, consultar [Hyndman and Athanasopoulos, 2018].

combinaciones y obtener diferentes modelos, como los ampliamente utilizados modelos basados en *suavizado exponencial* [Makridakis et al., 1982] que se describen a continuación.

2.2. suavizado exponencial

Suavizado exponencial describe un método para hacer pronósticos, donde los pronósticos son combinaciones ponderadas de los datos. Puntualmente, a los datos se les asigna un peso, que disminuyen exponencialmente cuando estos son más *antiguos*. Considerando las variaciones en la combinación de la tendencia y los componentes estacionales e ignorando la componente de error, son posibles quince métodos de suavizado exponencial, listados en la tabla 2.1. Cada método es etiquetado por un par de letras (T, S) que definen el tipo de componentes de tendencia *Trend* y estacionalidad *Seasonal*. Por ejemplo, (A, M) es el método con una tendencia aditiva *A* y estacionalidad multiplicativa *M*. Este tipo de clasificación fue propuesta por [Pegels, 1969] y posteriormente extendido por [Gardner, 1985, Hyndman and Billah, 2003, Taylor, 2003] para incluir métodos con tendencia aditiva amortiguada.

Tendencia	**Estacionalidad**		
	N (Ninguna)	A (Aditiva)	M (Multiplicativa)
N(Ninguna)	(N,N)	(N,A)	(N,M)
A(Aditiva)	(A,N)	(A,A)	(A,M)
Ad(aditiva amortiguada)	(Ad,N)	(Ad,A)	(Ad,M)
M(multiplicativa)	(M,N)	(M,A)	(M,M)
Md(multiplicativa amortiguada)	(Md,N)	(Md,A)	(Md,M)

Tabla 2.1. 15 métodos de suavizado exponencial posibles

Algunos de estos métodos son más conocidos con otros nombres. Por ejemplo,

- (N, N) describe el método de suavizado exponencial simple -o SES- [Brown, 1959].

- (A, N) describe el método de tendencia lineal Holt [Holt, 1957].
- (Ad, N) describe el método de tendencia amortiguada.
- (A, A) describe el método Holt-Winters con estacionalidad aditiva [Winters, 1960].
- (A, M) describe el método Holt-Winters con estacionalidad multiplicativa.

Las ecuaciones para determinar el pronóstico para h pasos adelante, por ejemplo para el modelo Hot-Winters aditivo (A,A) están dadas por,

$$\text{Pronóstico} : \hat{y}_{t+h|t} = l_t + h\ b_t + s_{t+h-m(k+1)} \tag{2.5}$$

Donde l_t representa el nivel de la serie de tiempo, dado por,

$$\text{Nivel} : l_t = \alpha(y_t + s_{t-m}) + (1-\alpha)(l_{t-1} + b_{t-1}) \tag{2.6}$$

α es el parámetro de suavizado de nivel y controla la velocidad con que disminuyen los pesos. b_t denota la tendencia de la serie de tiempo, dada por,

$$\text{Tendencia} : b_t = \beta(l_t - l_{t-1}) + (1-\beta)b_{t-1} \tag{2.7}$$

β es el parámetro de suavizado de tendencia y s_t es el componente estacional, dada por,

$$\text{Estacionalidad} : s_t = \gamma(y_t - l_{t-1} - b_{t-1}) + (1-\gamma)s_{t-m} \tag{2.8}$$

γ es el parámetro de suavizado de la estacionalidad. m es la frecuencia de la estacionalidad -por ejemplo, $m = 12$ para número de meses en un año o $m = 4$ para los trimestres en un año-, k es la parte entera de $(h-1)/m$ y asegura que las estimaciones de los índices estacionales utilizados para la predicción provienen del último año de la muestra. Así con las ecuaciones 2.5 a 2.8 se pueden obtener los pronósticos para los quince modelos de la tabla 2.1. En [Hyndman and Koehler, 2006] se agrega una componente de error aditivo

o multiplicativo para obtener los intervalos de predicción -o intervalos de confianza de los pronósticos-. A la notación anterior, se le agrega una letra (E,T,S) para identificar las componentes de Error, Tendencia y Estacionalidad. A estos modelos se les denomina *modelos en el espacio de estado*. En [Hyndman and Athanasopoulos, 2018] se pueden encontrar las ecuaciones de todos los modelos incluyendo los modelos en el espacio de estado -ETS-.

2.2.1. selección del modelo de suavizado exponencial

La selección de un modelo se basa principalmente en determinar los parámetros *adecuados* que garanticen una buena generalización y un pronóstico preciso. Como se vio en el capítulo anterior, los modelos basados en aprendizaje automático minimizan una función de pérdida para obtener los parámetros del modelo y seleccionar el modelo final. En estadística y en el caso de los modelos de suavizado exponencial, los parámetros α, β y γ -que pueden tomar valores entre 0 y 1- y los valores iniciales l_0, b_0, s_{-1},...,s_{-m+1} se obtienen calculando la probabilidad del modelo con el *estimador de máxima verosimilitud*. Una alta probabilidad se asocia a un buen modelo; pero no exento de sobreajuste. A diferencia de los métodos de aprendizaje automático, que utilizan un conjunto de datos de reserva -conjunto de validación- para validar un modelo y los métodos de regularización y validación cruzada para evitar el sobreajuste. Los métodos estadísticos utilizan todo el conjunto de datos para el ajuste del modelo y los criterios de información para la selección del modelo y evitar el sobreajuste.

Criterio de Información Akaike AIC

En estadística, para aumentar la probabilidad de un modelo se tiende a aumentar la cantidad de parámetros de dicho modelo, pero como se vio en el capitulo anterior esto nos llevará a sobreajustar el modelo. Los criterios AIC y BIC miden por una parte la *bondad de ajuste* de un modelo y penalizan la verosimilitud del modelo a partir de la

cantidad de parámetros. El Criterio de Información Akaike (AIC) se define por,

$$AIC = 2k - 2\ln(\hat{L}) \tag{2.9}$$

Donde k es el número de parámetros independientes del modelo -incluyen α, β, γ y los valores iniciales de los modelos- y $\hat{L}$ es el estimador de máxima verosimilitud del modelo a elegir. Dentro de los posibles modelos, se elegirá el que tenga menor valor AIC [Akaike, 1969].

Criterio de Información Bayesiano de Schwarz's

Otro criterio utilizado para la selección de un modelo es el criterio bayesiano BIC [Schwarz, 1978] que se define por,

$$BIC = k\ln(N) - 2\ln(\hat{L}) \tag{2.10}$$

Al igual que en AIC, $\hat{L}$ es el estimador de máxima verosimilitud de los modelos a elegir, N son la cantidad de datos y k el número de parámetros del modelo. Al agregar el $\ln(N)$, la complejidad del modelo queda en función de la cantidad de datos, penalizando aun más los modelos en comparación al criterio Akaike. Asumiendo un error aditivo con distribución normal, se ha determinado que, para valores de N grandes, minimizar AIC es equivalente a minimizar el valor de del error cuadrático utilizando validación cruzada [Ord and Fildes, 2012].

2.2.2. pronósticos con suavizado exponencial

Los pronósticos para los modelos de suavizado exponencial se obtienen iterando la ecuación 2.5 para $t = T + 1, ..., T + h$. Por ejemplo, para obtener los pronósticos del modelo ETS (M,A,N), es decir, el modelo con error multiplicativo y tendencia aditiva. Primero se calcula el pronóstico *de un paso adelante*, dado por,

$$\hat{y}_{T+1} = (l_T + b_T)\ (1 + e_{T+1}) \tag{2.11}$$

ya que $e_t = 0$ para $t > T$ nos queda,

$$\hat{y}_{T+1|T} = l_T + b_T \tag{2.12}$$

y para $h = 2$ el pronóstico tomando la ecuación 2.5,

$$\hat{y}_{T+2|T} = l_T + 2b_T \tag{2.13}$$

En el caso particular de este estudio se utilizó el paquete *forecast* para R [Hyndman and Khandakar, 2008]. Con el cual se obtuvieron los pronósticos de ventas utilizando los siguientes modelos basados en suavizado exponencial:

- Suavizado exponencial simple (SES)
- Holt con tendencia lineal
- Holt winters con estacionalidad aditiva amortiguado (HWA)
- Holt Winters con estacionalidad multiplicativa amortiguado (HWM)
- ETS (ExponenTial Smoothing) o modelos de espacio de estados que incluye el error aditivo o multiplicativo.

2.3. modelos ARIMA

Otro modelo estadístico ampliamente utilizado para realizar pronósticos de series de tiempo son los basados en la metodología Box-Jenkins [Box et al., 1994]. Su enfoque es construir un modelo matemático con datos históricos de una serie de tiempo asumiendo propiedades de autocorrelación en estos. Un modelo ARIMA por su nombre en inglés -*AutoRegressive Integrated Moving Average*- se conforma por un modelo autorregresivo

(AR), un modelo de promedio móvil (MA) y un grado de diferenciación, que se describen a continuación.

modelos autorregresivos (AR)

En un modelo de regresión múltiple, la variable de interés y se modela utilizando una combinación de las variables *predictoras* $\mathbf{x}$,

$$y_t = f(x_1 + x_2 + ... + x_k) + e_t \tag{2.14}$$

y e_t es el error con media cero y varianza constante σ^2. En un modelo de autorregresión, la variable de interés y se modela utilizando una combinación de los valores pasados de la misma variable y.

$$y_t = f(y_{t-1} + y_{t-2} + ... + y_{t-p}) + e_t \tag{2.15}$$

Nos referimos a un modelo autorregresivo de orden p -**AR(p)**-, definido por,

$$y_t = \phi_0 + \phi_1 y_{t-1} + \phi_2 y_{t-2} + ... + \phi_p y_{t-p} + e_t \tag{2.16}$$

Donde se utilizan los p valores inmediatos pasados, un conjunto de parámetros ϕ y el error e_t -ruido blanco-. Los modelos autorregresivos permiten manejar diferentes patrones de series de tiempo cambiando los parámetros $\phi_1, ...\phi_p$.

modelos de medias móviles (MA)

En lugar de utilizar valores pasados de la variable de interés para realizar el modelo autorregresivo. Un modelo de media móvil o -*Moving Average*- utiliza los errores de los pronósticos anteriores como modelo. Un modelo de medias móviles de orden q -**MA(q)**-, se define por,

$$y_t = \theta_0 + e_t + \theta_1 e_{t-1} + \theta_2 e_{t-2} + ... + \theta_q e_{t-q} \tag{2.17}$$

Donde el error e, corresponde a un conjunto de variables aleatorias independientes idénticamente distribuidas tomadas de una muestra con distribución normal de media cero y varianza σ^2. θ es el conjunto de parámetros y q es son los valores de los errores utilizados por el modelo. Así, una serie de tiempo y_t se considera como una media móvil ponderada de los errores anteriores.

diferenciación

Para que los modelos ARIMA sean efectivos, las series de tiempo a estudiar deben considerarse *estacionarias*, es decir, que sus características estadísticas como la media y la estructura de autocorrelación sean constantes a lo largo del tiempo. Sin embargo, en la práctica ocurre lo contrario. Las series de tiempo generalmente presentan tendencia y heterocedasticidad, por lo tanto, las series de tiempo se deben transformar. Una forma de transformar una serie de tiempo *no estacionaria-estacionaria* es calculando las diferencias entre observaciones consecutivas. Esto se conoce como *diferenciación*. La diferenciación puede ayudar a estabilizar la media de la serie de tiempo al eliminar los cambios en el nivel de esta y eliminar -o reducir- la tendencia y la estacionalidad. Y se puede escribir como,

$$y'_t = y_t - y_{t-1} \tag{2.18}$$

y se refiere a las diferencias ordinarias o *primeras diferencias*, es decir, diferencias con el rezago 1. También se puede utilizar una diferenciación estacional, donde se resta la observación de m periodos anteriores,

$$y'_t = y_t - y_{t-m} \tag{2.19}$$

Donde, por ejemplo, $m = 12$ para una estacionalidad anual. A veces es necesario tomar una diferencia estacional y una primera diferencia para obtener datos estacionarios.

Esto se determinará con pruebas de estacionariedad, que se describen en la sección 2.3.3.

2.3.1. modelo ARIMA no estacional

Tomando el modelo de autorregresión -AR- , el modelo de media móvil -MA- y el proceso de *diferenciación* en la serie de tiempo, se obtiene el modelo ARIMA no estacional, que supone que el valor futuro de una variable de interés es una función lineal de las observaciones y los errores aleatorios pasados, y se define por,

$$y'_t = c + \phi_1 y'_{t-1} + \phi_2 y'_{t-2} + ... + \phi_p y'_{t-p} + \theta_1 e_{t-1} + \theta_2 e_{t-2} + ... + \theta_q e_{t-q} \tag{2.20}$$

y'_t es la serie de tiempo diferenciada, c es una constante que en conjunto con el grado de diferenciación de la series de tiempo determinan el comportamiento de los pronósticos a largo plazo -para mayor detalle ver [Hyndman and Athanasopoulos, 2018]- y e_t es el error aleatorio en el período de tiempo t. ϕ_i $(i = 1, 2, ..., p)$ y θ_j $(j = 1, 2, ...q)$ son los parámetros del modelo **ARIMA(p,d,q)** donde,

- *p = orden de la parte autorregresiva*
- *d = grado de diferenciación.*
- *q = orden del promedio móvil.*

Si es necesario, en una etapa inicial -antes de diferenciarla serie de tiempo- es común utilizar la transformación Box-Cox -sección 3.2.1- para estabilizar la varianza de la serie de tiempo.

2.3.2. modelos ARIMA con estacionalidad (SARIMA)

Un modelo estacional ARIMA se forma incluyendo los términos estacionales de la serie de tiempo y se representar de la siguiente manera,

$$\text{ARIMA}(p,d,q)\ (P,D,Q)_m \tag{2.21}$$

Donde,

- (p,d,q) parte no estacional del modelo
- $(P,D,Q)_m$ parte estacional del modelo

$m =$ representa el periodo de la estacionalidad. La parte estacional del modelo implican retrocesos del período estacional. Por ejemplo, un modelo $ARIMA(1,1,1)(1,1,1)_4$ se aplica a datos trimestrales -$m = 4$-. Los términos de la estacionalidad son multiplicados a los términos no estacionales para obtener el modelo final.

Otra clase de modelos llamados modelos autorregresivos no lineales -NAR- se han utilizado cuando la variable de interés se propone como una función no lineal. A pesar de las variantes que puedan existir para los modelos autorregresivos, es el modelo ARIMA el que mejores resultados y el que más referencias y aplicaciones posee en estudios de pronósticos en general. Particularmente, en el estudio [Makridakis et al., 2018b], el modelo ARIMA obtiene el menor error sMAPE para pronósticos a corto plazo y el menor error MASE promedio para los pronósticos de las series de tiempo de la competencia M3.

2.3.3. selección del modelo ARIMA

Determinar si la serie de tiempo es estacionaria es fundamental para implementar los modelos ARIMA. Un método simple para determinar si una serie de tiempo es estacionaria es utilizar las gráficas de autocorrelación ACF y PACF -ACF de Autocorrelation Function y PACF de Partial Autocorrelation Function- y evaluar el valor de los coeficientes de autocorrelación r. Además, estas funciones se utilizan dentro de la metodología Box-Jenkins para identificar la forma del modelo y se sabe que dichas funciones nos darán información sobre las características de una serie de tiempo [Box

et al., 1994]. Puntualmente y como se ve en la figura 2.2:

- La gráfica ACF de las series temporales con tendencia tiende a tener valores positivos que disminuyen lentamente a medida que aumentan los retrasos.
- Cuando los datos son estacionales, las autocorrelaciones serán mayores para *lag-values* o valores rezagados estacionales -en múltiplos de la frecuencia estacional- que para otros *lag-values* -PACF de la figura 2.2-.
- Cuando los datos tienen tendencia y son estacionales, se observará una combinación de los efectos antes mencionados [Hyndman and Athanasopoulos, 2018].

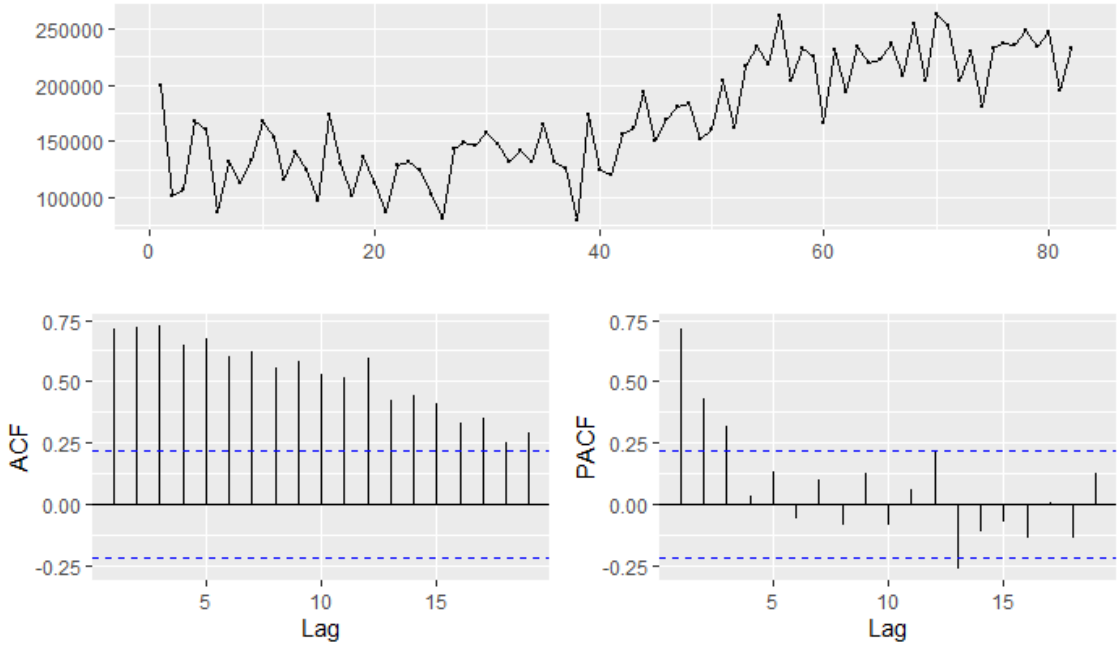


Figura 2.2. Serie de tiempo de ventas del estudio de caso que se presenta en el capitulo 3, y sus gráficas ACF y PACF.

Por lo tanto, las gráficas ACF y PACF de la serie de tiempo de la figura 2.3 no es estacionaria y se deben diferenciar para obtener la estacionariedad. Adicional a la inspección visual de las gráficas de autocorrelación, es común hacer pruebas de estacionariedad que determinarán de manera objetiva el grado de diferenciación necesario para lograr dicha estacionariedad de la serie de tiempo. La función *auto.arima()* del paquete *forecast* en R utiliza la prueba KPSS [Kwiatkowski et al., 1991] para estimar el

orden de la primera diferenciación d y una prueba de Canova-Hansen [Canova and Hansen, 1995] para estimar la diferenciación estacional D. Una vez obtenida la estacionariedad de la serie de tiempo, se plantean y prueban los modelos iniciales.

Con las curvas de ACF y PACF de los datos diferenciados de la serie de tiempo se pueden establecer los modelos ARIMA iniciales. Por ejemplo, en la figura 2.3 se muestra una serie de tiempo diferenciada -primera diferenciación- y se ve un valor alto en el rezago 1 y en el rezago 11 y 12; lo que sugiere implementar un modelo $ARIMA(0,1,1)(0,1,1)_{12}$ con las componentes $MA(1)$ no estacionales y estacionales, una primera diferenciación y una diferenciación estacional -de manera análoga se puede empezar con un modelo $ARIMA(1,1,0)(1,1,0)_{12}$ con las componentes $AR(1)$ no estacional y estacional-.

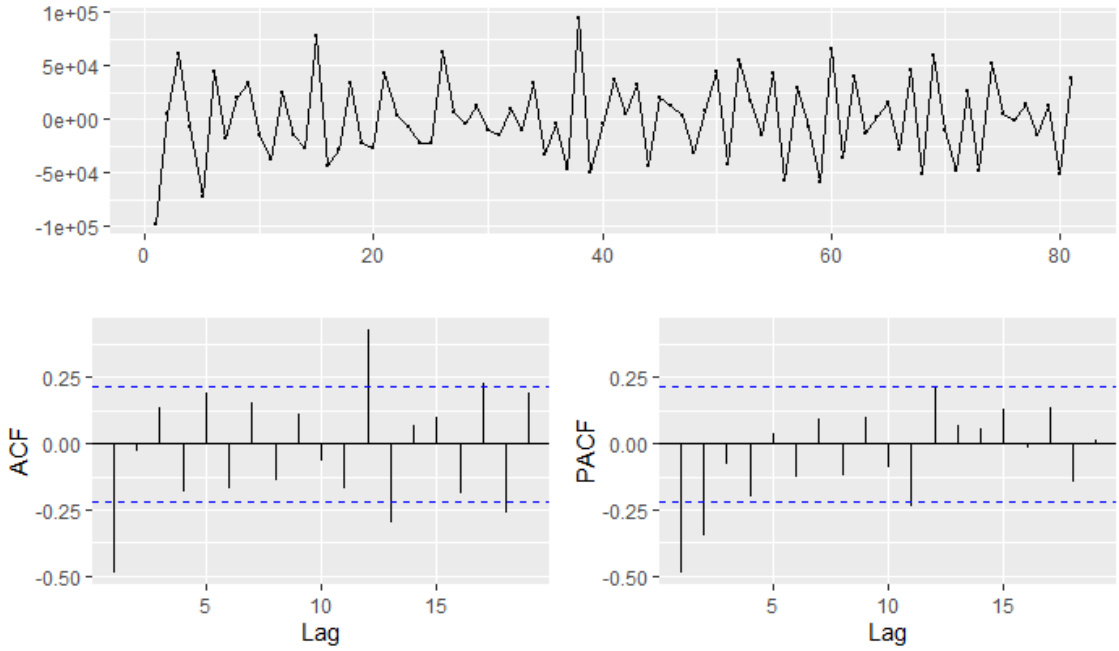


Figura 2.3. Serie de tiempo diferenciada -primera diferenciación- de las ventas del departamento Chocolate del estudio de caso presentado en el capitulo 3.

La función *auto.arima* considera variaciones de los modelos iniciales -variando p y q en factores de ± 1-. Los *buenos* modelos se obtienen minimizando el AIC, AICc o BIC. Como se señala en [Hyndman and Athanasopoulos, 2018] se prefiere utilizar el AICc. El AIC para un modelo ARIMA se define por,

$$AIC = -2\ log(L) + 2(p + q + k + 1) \tag{2.22}$$

Donde L es la *verosimilitud* de cada modelo con los diferentes valores de los parámetros p y q; $k = 1$ si $c \neq 0$ y $k = 0$ si $c = 0$. La función *auto.arima* también utiliza una versión corregida del criterio de información AIC, que se define por,

$$AIC_C = AIC + \frac{2(p + q + k + 1)(p + q + k + 2)}{N - p - q - k - 2} \tag{2.23}$$

y finalmente el criterio de información BIC, que se puede escribir como,

$$BIC = AIC + [log(N) - 2](p + q + k + 1) \tag{2.24}$$

Para nuestro ejemplo, la figura 2.4 muestra el residuo del modelo final seleccionado $ARIMA(0, 1, 1)(1, 0, 0)_{12}$ -que contiene una componente no estacional $MA(1)$, una componente estacional $AR(1)$ y una primera diferenciación-, con un AIC=1610.18, AICc=1610.55 y BIC=1616.88. El gráfico ACF del residuo muestra todas las autocorrelaciones están dentro del umbral, indicando que los residuos se comportan como ruido blanco.

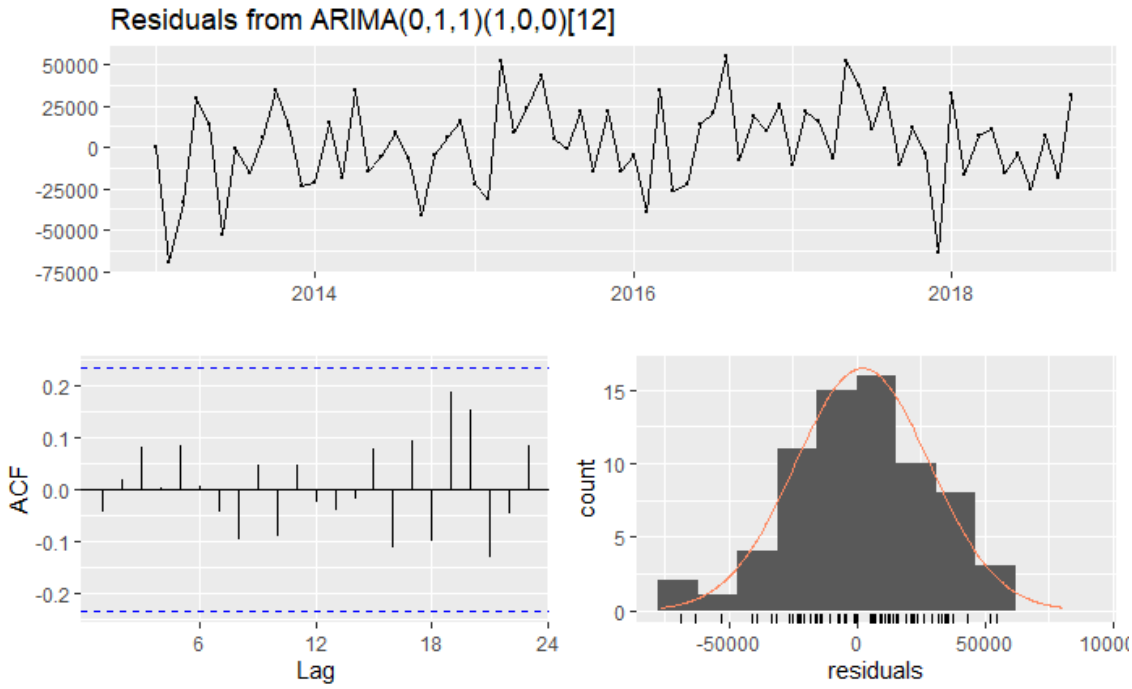


Figura 2.4. Residuo del modelo $ARIMA(0,1,1)(1,0,0)_{12}$ obtenido para la serie de tiempo del departamento Chocolate del estudio de caso presentado en el capitulo 3.

Hay que destacar que todo el procedimiento descrito en esta sección y el modelo para el ejemplo enunciado se obtuvieron de manera automática con la función *auto.arima* del paquete forecast de R. El siguiente y último paso, después de obtener el mejor modelo es calcular los pronósticos.

2.3.4. pronósticos con modelos ARIMA

En las secciones anteriores se estableció el modelo ARIMA mediante las componentes de diferenciación, la autorregresión -AR- y el promedio móvil de los errores de una serie de tiempo estacionaria -MA-. Por otra parte, el paradigma estadístico estándar asume que bajo las condiciones de estacionariedad, el mejor modelo ajustado a los datos históricos, será el modelo óptimo para pronosticar [Fildes and Makridakis, 1995].

Para describir como se obtienen los pronósticos con el modelo ARIMA, primero introduciremos el operador *backward shift* B que se utiliza cuando se pronostica una serie de tiempo con los valores rezagados. El operador B esta dado por,

$$B\ y_t = y_{t-1} \tag{2.25}$$

En este caso B desplaza los datos de la serie de tiempo en un periodo. Desplazando doce valores hacia atrás, se puede escribir como,

$$B^{12} y_t = y_{t-12} \tag{2.26}$$

El operador B también se utiliza para describir el grado de diferenciación de una serie de tiempo. Por ejemplo, la primera diferenciación se puede escribir,

$$y'_t = y_t - y_{t-1} = y_t - By_t = (1 - B)y_t \tag{2.27}$$

Por ejemplo, el modelo ARIMA (3,1,1) con $p = 3$, $q = 1$ y un grado de diferenciación, se puede escribir,

$$\begin{aligned} (1 - \hat{\phi}_1 B - \hat{\phi}_2 B^2 - \hat{\phi}_3 B^3)(1 - B)y_t &= (1 + \hat{\theta}_q B)e_t \\ [1 - (1 + \hat{\phi}_1)B + (\hat{\phi}_1 - \hat{\phi}_2)B^2 + (\hat{\phi}_2 - \hat{\phi}_3)B^3 + \hat{\phi}_3 B^4]y_t &= (1 + \hat{\theta}_q B)e_t \end{aligned} \tag{2.28}$$

y utilizando el operador B obtenemos,

$$y_t - (1 + \hat{\phi}_1)y_{t-1} + (\hat{\phi}_1 - \hat{\phi}_2)y_{t-2} + (\hat{\phi}_2 - \hat{\phi}_3)y_{t-3} + \hat{\phi}_3 y_{t-4} = e_t + \hat{\theta}_q e_{t-1} \tag{2.29}$$

Despejando y_t,

$$y_t = (1 + \hat{\phi}_1)y_{t-1} - (\hat{\phi}_1 - \hat{\phi}_2)y_{t-2} - (\hat{\phi}_2 - \hat{\phi}_3)y_{t-3} - \hat{\phi}_3 y_{t-4} + e_t + \hat{\theta}_q e_{t-1} \tag{2.30}$$

Se obtiene el modelo ARIMA (3,1,1) para la serie de tiempo y_t. El pronóstico para el

siguiente paso $T+1$ se obtiene,

$$y_{T+1} = (1+\hat{\phi}_1)y_T - (\hat{\phi}_1 - \hat{\phi}_2)y_{T-1} - (\hat{\phi}_2 - \hat{\phi}_3)y_{T-2} - \hat{\phi}_3 y_{T-3} + e_{T+1} + \hat{\theta}_q e_T \quad (2.31)$$

y dado que $e_{T+1} = 0$, el pronóstico para $T+1$ finalmente esta dado por,

$$\hat{y}_{T+1|T} = (1+\hat{\phi}_1)y_T - (\hat{\phi}_1 - \hat{\phi}_2)y_{T-1} - (\hat{\phi}_2 - \hat{\phi}_3)y_{T-2} - \hat{\phi}_3 y_{T-3} + \hat{\theta}_q e_T \quad (2.32)$$

Así mismo, para obtener el pronóstico del siguiente paso $T+2$,

$$\hat{y}_{T+2|T} = (1+\hat{\phi}_1)\hat{y}_{T+1|T} - (\hat{\phi}_1 - \hat{\phi}_2)y_T - (\hat{\phi}_2 - \hat{\phi}_3)y_{T-1} - \hat{\phi}_3 y_{T-2} \quad (2.33)$$

El procedimiento se repite hasta obtener el horizonte de pronósticos h deseado. Como se señalo en la sección anterior, se utilizó el paquete *forecast* para R y la función *auto.arima* para obtener los pronósticos de múltiples pasos con el método ARIMA. Para más detalles en el procedimiento de selección del modelo, optimización de los parámetros y la obtención del los pronósticos con el método ARIMA y en general para todos los métodos estadísticos utilizados en este estudio, consultar [Hyndman, 2008, Hyndman and Athanasopoulos, 2018].

2.4. método Theta

El método theta desarrollado por [Assimakopoulos and Nikolopoulos, 2000] está basado en descomponer una serie de tiempo en función de sus curvaturas locales; con este método se obtuvieron los mejores pronósticos de las series mensuales de la competencia M3. Puntualmente, este método descompone la serie de tiempo en dos nuevas líneas -llamadas *líneas theta*- a través de los *coeficientes theta* -$\theta \in \mathbb{R}$-, que se

aplican a la segunda diferenciación de los datos. Según el autor:

- Si $\theta < 1$, las segundas diferenciaciones se reducen resultando en la mejor aproximación del comportamiento a largo plazo de la serie de tiempo.
- Si $\theta = 0$, la línea descompuesta se transforma en una línea recta con pendiente constante.
- Si $\theta > 1$ la curvatura local aumenta, magnificando los movimientos a corto plazo de la serie temporal [Assimakopoulos and Nikolopoulos, 2000].

Las líneas thetas obtenidas, mantienen la media y la pendiente de la serie de tiempo original. Sin embargo, la curvatura local se filtra. Originalmente en [Assimakopoulos and Nikolopoulos, 2000], se proponen las lineas theta como una solución a la ecuación,

$$\nabla^2 Z_t(\theta) = \theta \nabla^2 y_t, \quad t = 3, .., n \tag{2.34}$$

Donde $y_1, ..., y_n$ es la serie de tiempo original -no estacional-, ∇ es el operador diferenciación y $Z_t(\theta)$ son las lineas theta. Dichas líneas se pueden calcular como una combinación lineal de los datos y una componente de tendencia, dada por,

$$Z_t(\theta) = \theta y_t + (1 - \theta)(\hat{a}_\theta + \hat{b}_\theta \, t), \quad t = 1, .., n \tag{2.35}$$

Donde y_t es la serie de tiempo, $\hat{a}_\theta$ y $\hat{b}_\theta$ son los coeficientes de una regresión lineal simple sobre $y_1, ...y_n$ y contra el tiempo $1, ...n$,

$$\hat{a}_\theta = \frac{1}{n} \sum_{t=1}^{n} y_t - \frac{n+1}{2} b_\theta \tag{2.36}$$

y

$$\hat{b}_\theta = \frac{6}{n^2 - 1} \left(\frac{2}{n} \sum_{t=1}^{n} t \; y_t - \frac{1+n}{n} \sum_{t=1}^{n} y_t \right) \tag{2.37}$$

Así, la serie de tiempo $y_1, ...y_n$, se puede representar como una combinación lineal de las dos lineas theta que se propusieron como modelo para la competencia M3,

$$\hat{y}_t = w\ Z_t(0) + (1-w)\ Z_t(2) \tag{2.38}$$

Donde $w \in [0,1]$ es el parámetro de pesos y en el caso del modelo utilizado en la competencia M3, estos pesos tienen igual valor.

2.4.1. pronósticos con el método Theta

Para obtener los pronósticos de h pasos con el método Theta, las dos líneas theta se extrapolan para obtener los pronósticos. La linea theta $Z_t(0)$ se extrapola por un modelo de regresión lineal y la linea theta $Z_t(2)$ se extrapola mediante el método de suavizado exponencial simple (SES). Los pronósticos finales se producen combinando los pronósticos de las dos líneas theta,

$$\hat{y}_{n+h|t} = \frac{1}{2}[Z_{n,0}(h) + Z_{n,2}(h)] \tag{2.39}$$

Donde $\hat{y}_{n+h|t}$ es el pronostico de n observaciones y $Z_{n,0}(h)$ esta dado por,

$$Z_{n,0}(h) = \hat{a}_0 + \hat{b}_0(n+h-1) \tag{2.40}$$

y a su vez $Z_{n,2}(h)$,

$$Z_{n,2}(h) = \alpha \sum_{i=1}^{n-1}(1-\alpha)^i\ Z_{n-1,2} + (1-\alpha)^n\ Z_{1,2} \tag{2.41}$$

Donde $\alpha \in (0,1)$ es el parámetro de suavizado para el modelo SES.

$$\begin{aligned} Z_{n,2}(h) &= \alpha \sum_{i=1}^{n-1}(1-\alpha)^i \left[\hat{a}_2 + \hat{b}_2(n-i-1) + 2y_{n-1}\right] + (1-\alpha)^n\ (\hat{a}_2 + 2y_1) \\ &= \hat{a}_2 + \hat{b}_2\left[n - \frac{1}{\alpha} + \frac{(1-\alpha)^n}{\alpha}\right] + 2\ \tilde{y}_n(h) \end{aligned} \tag{2.42}$$

Donde $\tilde{y}_n(h)$ es el pronóstico SES de la serie de tiempo, y dado que $\hat{a}_2 = -\hat{a}_0$ y $\hat{b}_2 = -\hat{b}_0$, se obtiene,

$$\hat{y}_n(h) = \tilde{y}_n(h) + \frac{1}{2}\hat{b}_0\left(h - 1 + \frac{1}{\alpha} + \frac{(1-\alpha)^n}{\alpha}\right) \tag{2.43}$$

y para n grande, se puede escribir,

$$\hat{y}_n(h) = \tilde{y}_n(h) + \frac{1}{2}\hat{b}_0\left(h - 1 + \frac{1}{\alpha}\right) \tag{2.44}$$

Con esta versión simplificada del método Theta, se obtuvieron los mejores resultados en las series de tiempo mensuales de la competencia M3 [Assimakopoulos and Nikolopoulos, 2000]. En [Hyndman and Billah, 2003], se demuestra que existe una relación entre el método Theta presentado y el modelo de suavizado exponencial simple con deriva -SES-d- y en [Nikolopoulos et al., 2011], se proponen usar más líneas theta -$\theta \in \{-1, 0, 1, 2, 3\}$- con el objetivo de extraer más información de los datos. Sin embargo, no se ha establecido un procedimiento formal para la selección de las líneas theta apropiadas [Fiorucci et al., 2016]. En este estudio se implemento el método Theta utilizando la función *theta* del paquete *forecast* para R, para obtener los pronósticos de múltiples pasos.

Recapitulando, en este apartado de la tesis se describieron los métodos estadísticos para pronósticos que se han utilizado comúnmente durante décadas y que destacan por su simplicidad y rapidez para obtener resultados. Makridakis en su estudio [Makridakis et al., 2018b] utiliza estos modelos que superan en precisión con respecto a los algoritmos de aprendizaje automático. Para el caso particular de este estudio, implementamos dichos métodos estadísticos que suman ocho métodos diferentes incluyendo los métodos ingenuo estacional y no estacional; y que se listan a continuación,

- Naïve o método ingenuo no estacional: Los pronósticos son el valor de la última observación. Es decir,

$$\hat{y}_{T+h|T} = y_T \tag{2.45}$$

- Naïve2 o método ingenuo estacional: Cada pronóstico es igual al último valor observado de la estación anterior -por ejemplo, el mismo mes del año anterior-,

$$\hat{y}_{T+h|T} = y_{T+h-m(k+1)} \tag{2.46}$$

  dónde m es el período estacional, y k es la parte entera de $(h-1)/m$ -es decir, el número de años completos en el período de pronóstico anterior al tiempo $T+h$-. A estos dos métodos ingenuos que se utilizan como métodos bases de comparación, se suman los siguientes métodos:
- Holt
- Holt winters aditivo amortiguado (HWA)
- Holt Winters multiplicativo amortiguado (HWM)
- Suavizado exponencial simple (SES)
- ETS (ExponenTial Smoothing) o modelos de espacio de estados que incluye el error aditivo o multiplicativo.
- ARIMA
- Theta

En el siguiente capítulo presentamos el estudio de caso, el procedimiento, algunas particularidades de los métodos de aprendizaje automático y las configuraciones finales de los modelos para obtener los pronósticos.

Capítulo 3

Pronósticos de ventas con algoritmos de aprendizaje automático (análisis de caso)

En los capítulos anteriores hemos introducido los fundamentos para abordar los problemas de regresión, las técnicas de pronósticos con el enfoque del aprendizaje automático y el enfoque de los métodos estadísticos para pronósticos de múltiples pasos de series de tiempo. En este capítulo, y para cumplir el objetivo de esta investigación, presentamos el estudio de caso y la metodología desarrollada para obtener los pronósticos de ventas de un año por delante de una organización dedicada a la industria alimenticia. La metodología se describe desde la adquisición, el preprocesamiento y la selección y transformación de las características o variables de entrada de los datos, en el marco del aprendizaje automático denominado *ingeniería de características*.

Establecida la primera etapa dedicada a la configuración de los conjuntos de datos - series de tiempo -, se describen las heurísticas realizadas para obtener las configuraciones de cada uno los algoritmos ML. Concretamente, se listan las arquitecturas, parámetros e hiperparámetros que se utilizaron en cada uno de los algoritmos presentados en el capítulo 1. Además, se describen algunas consideraciones al aplicar las diferentes técnicas de validación descritas en la sección 1.4 y para finalizar se describen algunas consideraciones al aplicar las técnicas de pronósticos - de la sección 1.10- utilizadas para el caso de estudio.

A pesar de que la metodología presentada en este capítulo se hace principalmente

desde el enfoque de aprendizaje automático, se consideran en simultaneo la metodología, consideraciones y detalles para la implementación de los métodos estadísticos de pronósticos.

3.1. presentación del estudio de caso

La figura 3.1 muestra las ventas mensuales en kilogramos de materia prima de una subsidiaria de un grupo internacional en los sectores de panadería, pastelería y chocolatería.

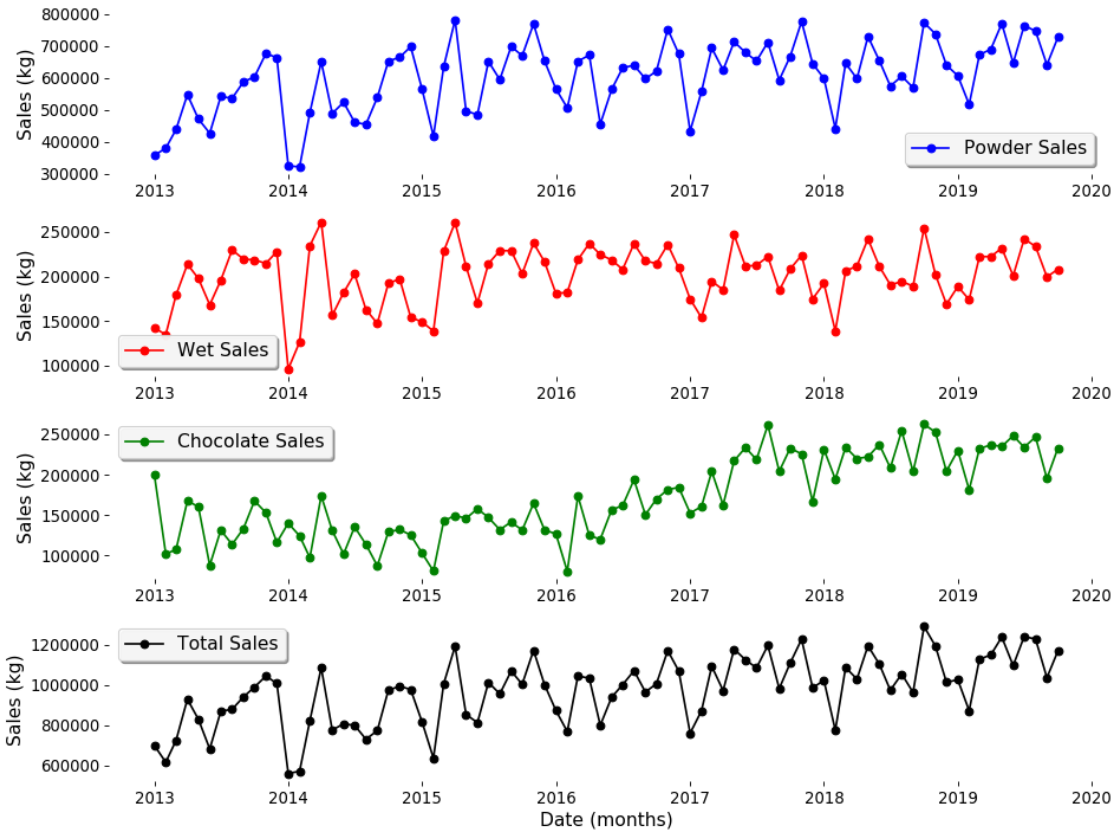


Figura 3.1. Series de tiempo de las ventas - desde enero del 2013 a octubre del 2019-. La compañía se divide en tres departamentos: Polvos, Wet y Chocolate, más la suma Total de las ventas (gráfica inferior).

Los datos obtenidos abarcan el período de ventas desde enero del 2013 a octubre del 2019 -$N = 82$- y corresponden a las series de tiempo de los tres departamentos que la conforman:

- Departamento de Polvos: produce principalmente productos de panadería -bases, premezclas- y representa el 62 % de las ventas de la compañía.
- Departamento Wet: produce productos como mermeladas, rellenos y cremas pasteleras. Representa el 21 % de las ventas de la compañía.
- Departamento Chocolate: son principalmente productos con base de chocolate -coberturas, rellenos, chocolate- y representa el 17 % de las ventas de la compañía.

AL estudio se agrego una cuarta serie de tiempo que corresponde a las ventas totales de la compañía - como se muestra en la figura 3.1-.

Como información adicional, la compañía produce un volumen aproximado de 20 toneladas diarias de materia prima y sus clientes son artesanos, compañías industriales, minoristas y empresas del sector. En el año 2018 las ventas fueron de 12,700 toneladas y en octubre del 2019 se vendieron 1,040 toneladas de materia prima total. Su presencia en el mercado es importante -abarca un 50 % aproximadamente- y por lo tanto la precisión en los pronósticos juegan un papel fundamental para las metas y el crecimiento de la compañía.

De las series de tiempo mensual de la compañía, se extrajeron los doce últimos datos -*conjunto de prueba*-, que corresponden al último año de ventas -desde noviembre del 2018 a octubre del 2019- y se utilizaron para cuantificar la precisión de los pronósticos de los diferentes modelos. Los $N - 12$ datos restantes *pasan* a la etapa que se denomina *ingeniería de características* y se describe a continuación.

3.2. ingeniería de características

La ingeniería de características suele ser la etapa más larga y difícil al crear proyectos de aprendizaje automático. Formalmente es el proceso de usar el conocimiento sobre los datos y sobre el algoritmo de aprendizaje automático para hacer que el algoritmo funcione mejor. Generalmente se aplican transformaciones a los datos -convertir los

datos en vectores de entrada o vectores de características útiles- antes de que entren a un modelo [Chollet, 2017]. El procedimiento inicia con el preprocesamiento de los datos, para luego identificar o crear las variables de entrada o características más importantes que se presentan en los datos.

3.2.1. preprocesamiento de las series de tiempo

Es común en los modelos estadísticos de pronósticos utilizar preprocesamiento para separar las componentes de tendencia, estacionalidad o *forzar* una serie de tiempo a ser estacionaria antes de realizar un pronóstico. En el estudio de [Ahmed et al., 2010] se utiliza diferenciación y promedios móviles como preprocesamiento y reporta que con diferenciación se obtienen mayores errores sMAPE, y en [Makridakis et al., 2018b] se reporta que el menor error sMAPE se obtiene cuando se utiliza desestacionalización y la transformación Box-Cox en conjunto.

Las transformaciones Box-cox, incluyen la transformación logarítmica y la transformación de potencia, y se definen por,

$$w_t = \begin{cases} log(y_t) & \text{si} \quad \lambda = 0 \\ \frac{(y_t^{\lambda}-1)}{\lambda} & \text{en otro caso} \end{cases} \tag{3.1}$$

Donde y_t es la serie de tiempo y λ determina que tipo de transformación se utilizará. Por ejemplo, si $\lambda = 0$ se aplica una transformación logarítmica o si λ es diferente de 0, se aplica la transformación de potencia. Un valor adecuado de λ hará que las variaciones estacionales sean homogéneas en toda la serie de tiempo. En particular, cuando se implementa el paquete *forecast* de R [Hyndman and Khandakar, 2008], se aplican automáticamente las transformaciones Box-Cox antes de descomponer la serie de tiempo [Hyndman and Athanasopoulos, 2018].

Con respecto a los algoritmos de aprendizaje automático, es común que las series de tiempo se normalicen entre los valores $(0 - 1)$. Esta normalización lineal garantiza

rapidez y eficiencia en la ejecución de los algoritmos de aprendizaje automático utilizados (se utilizó la función *MinMaxScaler* del API *Scikit-Learn* para realizar la normalización.)

3.2.2. selección de características

La etapa de selección de características o también denominada *selección de variables* o *selección de atributos* tiene como objetivo: identificar, crear y/o eliminar características innecesarias, irrelevantes y/o redundantes en los datos; permitiendo: i) mejorar la capacidad de generalización de los predictores, ii) proporcionar predictores más rápidos y eficientes, y iii) proporcionar una mejor comprensión del *fenómeno* subyacente que generó los datos.

Por otra parte, los métodos de pronósticos de series de tiempo en su mayoría son métodos autorregresivos y recursivos que asumen dependencia en las observaciones adyacentes de una serie de tiempo [Box et al., 1994] y las predicciones finales se obtienen de los valores pasados de la misma serie de tiempo.

Así, la selección de características para problemas de pronósticos de series de tiempo, se enfoca en determinar *qué* y *cuántos* valores rezagados *importantes* de la serie de tiempo se utilizarán como variables de entrada. Para este estudio, el grado de importancia de los datos rezagados de las series de tiempo se determinó con la función de autocorrelación ACF y la función de autocorrelación parcial PACF. La figura 3.2 muestra la función ACF y PACF de las series de tiempos de la compañía en estudio -Las gráficas de la autocorrelación y autocorrelación parcial mostradas se generaron usando R. Es muy útil trazar los límites de error estándar alrededor de cero para los coeficientes estimados. Los límites en R son aproximadamente dos límites de error estándar, $\pm\frac{2}{\sqrt{n}}$, donde n es el tamaño de la serie de tiempo-. Se puede observar que el lag 1, 11 y 12 tienen los mayores valores de ACF y PACF - que se pueden asociar a características estacionales - en los departamentos de Polvo, Wet y el Total de la compañía. Son estos valores -lag 1, 11 y 12- los que se utilizan como variables de entrada. Y para las ventas en el departamento

de Chocolate, son los primeros 3 lags con mayor valor ACF y PACF -que se asocian a una tendencia creciente en las ventas- los que se utilizan como variables de entrada para el proceso de entrenamiento y validación de los algoritmos ML.

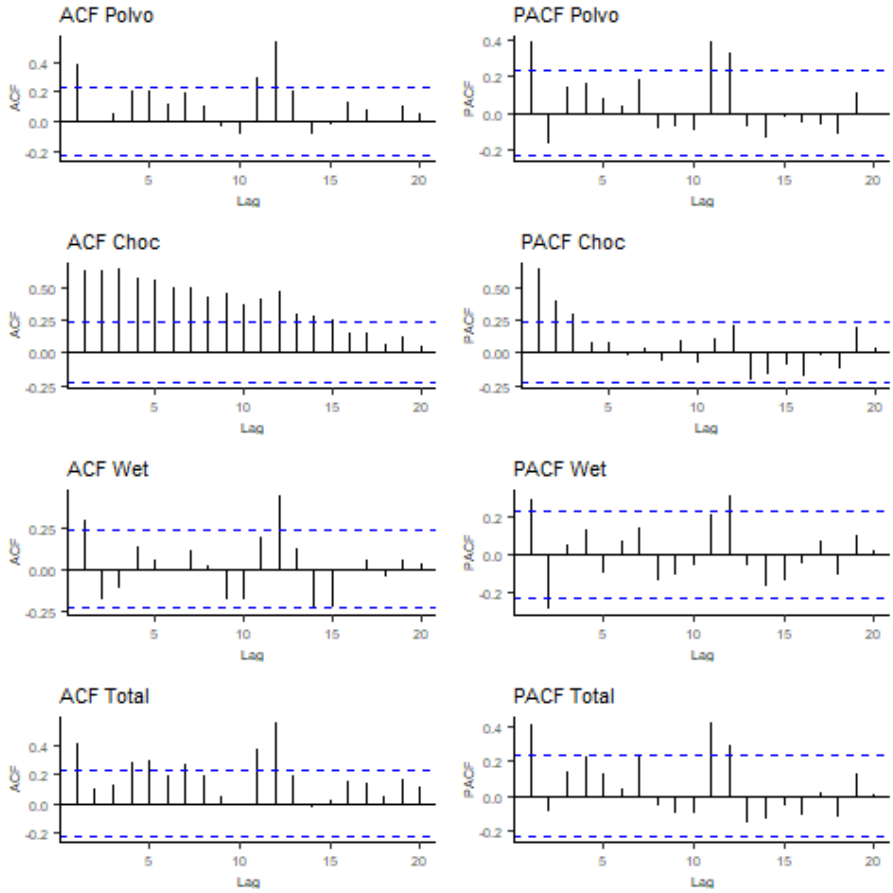


Figura 3.2. Acf y Pacf para las cuatro series de tiempo de los departamentos de la compañía en estudio, considerando las ventas hasta octubre del 2019.

En resumen, para este estudio, se utilizaron los lag-values de las series de tiempo correspondientes a los valores con mayor autocorrelación ACF y PACF y están dentro del límite de error estándar, $\pm\frac{2}{\sqrt{n}}$ y no mayor a tres lag-values. Establecidos estos valores, se conformaron tres matrices de variables de entradas $\mathbf{x}$. Una primera matriz se conformo con el lag de mayor autocorrelación. Una segunda matriz se conformo con los primeros dos lags y la última matriz se conformo con los tres lags de mayor autocorrelación. el objetivo de conformar tres matrices con los valores rezagados de las series de tiempo es determinar si a mayor números de lags utilizados, la precisión de los pronósticos

aumenta. Estas matrices en conjunto con los objetivos y, que corresponden a las ventas *originales* de cada departamento de la compañía, constituyeron los datos finales con los cuales se entrenaron y validaron todos los algoritmos de aprendizaje automático.

3.2.3. configuración e hiperparámetros

La siguiente etapa de la ingeniería de características es determinar las mejores configuraciones y el *mejor modelo* que se ajuste a las características de los datos. Las primeras configuraciones, hiperparámetros y arquitecturas de los algoritmos ML se obtuvieron con el objetivo de mantener una complejidad adecuada en los modelos y evitando el sobreajuste en la etapa de entrenamiento de los modelos; mediante la evaluación de los modelos en el conjunto de prueba y monitoreando las curvas de aprendizaje vista en la sección 1.3. Se debe destacar que esta etapa del procedimiento fue la más detallada y por ende la más tardada debido a su carácter heurístico. En una primera instancia, las configuraciones para cada algoritmo de aprendizaje automático se obtuvieron considerando las diferentes *sugerencias* que se establecen en artículos académicos[1] y bibliografías de cada algoritmo. A continuación, evaluando los resultados obtenidos con los modelos base y la experiencia obtenida en el proceso, se listan las configuraciones finales para todos los algoritmos ML utilizados.

Configuraciones para Modelos No Paramétricos:

Para los Modelos No Paramétricos se utilizó la API especializado para aprendizaje automático Scikit-Learn version 0.22.1. para Python versión 3.7. Los hiperparámetros que se utilizaron para cada uno de los modelos no parámetros se listan a continuación:

- KNN (Regresión con K vecinos cercanos):
  - hiperparametro K= $\{1, ..., 6\}$ en pasos de 1

[1]En la mayoría de los artículos académica citados y asociados a los diferentes algoritmos de aprendizaje automático, se sugieren configuraciones, metodologías de implementación y particularidades en la implementación informática

- GRNN (Red Neuronal de regresión generalizada):
  - en conjunto con la librería *NeuPY*
  - γ =$\{1e-2, ..., 1\}$ en intervalos de 0.01
- GP (Proceso Gausiano):
  - Función kernel de base radial y los demás hiperparámetros son ajustados automáticamente.
- SVR (Regresión con vectores de soporte):
  - función kernel de base radial
  - C =$\{1e-2, ..., 1\}$ en intervalos de 0.1
  - ϵ =$\{1e-4, 1e-3, 1e-2, 1e-1, 1\}$
  - γ =$\{1e-2, ..., 1\}$ en intervalos de 0.1
- CART (Árboles de clasificación y regresión):
  - Profundidad del árbol (tree depth): $\{1-3\}$ en pasos de 1

Configuraciones para Métodos de ensamble:

Para los métodos de ensamble basados en árboles de regresión, se utilizó el API especializado para aprendizaje automático Scikit-Learn versión 0.22.1. para Python versión 3.7. Los hiperparámetros que se utilizaron para cada uno de los métodos, fueron los siguientes:

- BAGG (Bagging):
  - cantidad de arboles (trees): $\{1-200\}$ en pasos de 1
- RF (Bosque aleatorio - Random Forest):
  - cantidad de arboles (trees): $\{1-200\}$ en pasos de 1
  - profundidad del árbol (tree depth): $\{1-3\}$ en pasos de 1
- ExT (Árboles extremadamente aleatorios):

  - cantidad de arboles (trees): $\{1 - 200\}$ en pasos de 1
  - profundidad del árbol (tree depth): $\{1 - 3\}$ en pasos de 1

- GBM (Gradient Boosting Machine):
  - cantidad de arboles (trees): $\{1 - 200\}$ en pasos de 1
  - razón de aprendizaje (learning rate or shrinkage): $\{1e-1, ..., 1\}$ en intervalos de 0.1
  - profundidad del árbol (tree depth): $\{1 - 3\}$ en pasos de 1

- XGBoost:
  - en conjunto con la librería *XGBoost*
  - cantidad de arboles (trees): $\{1 - 200\}$ en pasos de 1
  - razón de aprendizaje (learning rate or shrinkage): $\{0,1, ..., 1\}$ en pasos de 0.1
  - profundidad del árbol (tree depth): $\{1 - 3\}$ en pasos de 1

Configuraciones para Aprendizaje Profundo:

Para los algoritmos de aprendizaje profundo se utilizó la API tensorflow versión 2.0.0 y Keras versión 2.2.5. Los hiperparámetros que se utilizaron para cada una de las redes neuronales artificiales, fueron los siguientes:

- Neuronas o filtros: $\{50 - 1000\}$ en intervalos de 50
- Capas Ocultas: $\{1, 2, 3\}$
- 500 épocas
- 10 repeticiones[2]

Todas las configuraciones descritas anteriormente se integran en una parrilla de búsqueda -*Grid Search*- y en conjunto con las series de tiempo preprocesadas se ini-

[2]Debido a que en el proceso de optimización con redes neuronales artificiales se obtienen varias soluciones (mínimos locales); el procedimiento de entrenamiento y validación se repite y se promedian los resultados.

cia la etapa de entrenamiento y validación. Todos los modelos antes mencionados se programador en los libros de notas *Jupyter* versión 6.0 con lenguaje Python versión 3.7.

Con respecto a los métodos estadísticos, el único parámetro que se definió es el valor de la estacionalidad, que corresponde a 12 -12 ventas anuales-. El resto de los parámetros se obtuvieron automáticamente al utilizar el paquete *forecast* de R [Hyndman and Koehler, 2006].

3.3. entrenamiento y validación

El siguiente paso en el procedimiento es obtener el modelo final f_d para cada algoritmo de aprendizaje automático. En la práctica consiste en obtener la configuración que mejor representa al conjunto de datos preprocesados y el conjunto de configuraciones de cada algoritmo que se describieron en la sección anterior. Los modelos finales se obtuvieron con cada uno de los métodos de validación vistos en la sección 1.4. Considerando el conjunto de datos sin el conjunto de prueba - que corresponde a los doce últimos datos -, las configuraciones de las técnicas de validación aplicadas para obtener las ventas futuras fueron las siguientes:

- **Validación Hold-out**: Se programó para que los doce últimos datos de las series de tiempo correspondieran al conjunto de validación. El resto de los datos se utilizaron para el entrenamiento de los modelos.

Figura 3.3. Representación gráfica del proceso de validación cruzada Hold-out. Una parte de los datos corresponde al conjunto de entrenamiento y el resto corresponde al conjunto de validación.

- **Validación K-fold**: se utilizó la función *Kfold* del paquete Scikit-Learn versión 0.22.1. Se extrajeron diez subconjuntos consecutivos de las series de tiempo para el proceso de entrenamiento y validación de los modelos. En una primera iteración - como muestra la figura 3.4-, se toma el primer *fold* como conjunto de validación y los restantes nueve *fold* se utilizan para el entrenamiento del modelo. En una segunda iteración, el segundo *fold* se utiliza para la validación del modelo y los restantes nueve *folds*, incluyendo el primer *fold*, se utilizan para el entrenamiento del modelo. El procedimiento continuo hasta completar los diez *folds*. La validación *Kfold* con k=10 fue utilizada como base y única forma de validación en el estudio de [Makridakis et al., 2018b].

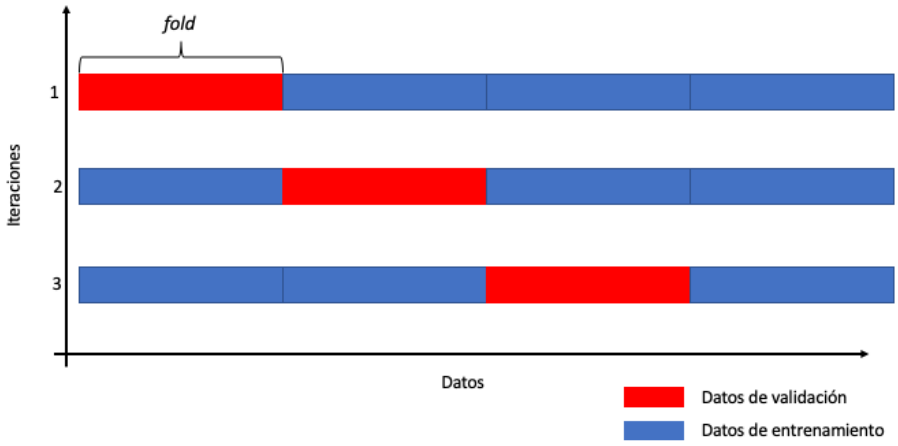


Figura 3.4. Representación gráfica del proceso de validación cruzada K-fold. El conjunto de datos se divide en K partes, que en sucesivas iteraciones se alternan como conjunto de entrenamiento y validación.

- **Validación Walk Forward**: Para este tipo de validación, se propone - después de realizar heurísticas diferentes, que consideraban varios tamaños de ventanas móviles - utilizar una ventana móvil mensual en el conjunto de entrenamiento y en los doce últimos datos correspondientes al conjunto de validación. Es decir, en la etapa de entrenamiento los datos de las series de tiempo se van desplazando mes por mes; lo mismo sucede para la etapa de validación - como lo representa La figura 3.5 -. Esta configuración se realizó a través de un *generador*[3] mediante la función *TimeseriesGenerator* del API Keras versión 2.2.5 para los algoritmos de aprendizaje profundo y programada para el resto de los algoritmos de aprendizaje automático.

[3]De manera simple, un generador es un objeto que actúa como un iterador: es un objeto que usa con el operador *for* y devuelve un vector

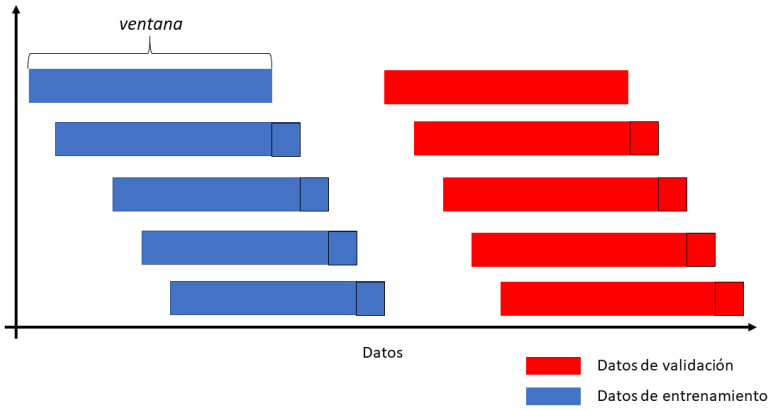


Figura 3.5. Representación gráfica del proceso de validación Walk forward. Esta representación corresponde a solo una época.

Con la validación Walk forward se quiere enfatizar y confirmar la importancia de realizar la etapa de entrenamiento y validación de los diferentes algoritmos con los datos ordenados en el tiempo.

Con respecto a la validación LOOV - que se describe en la sección 1.4 - se hicieron algunas pruebas, pero debido a las características de esta validación - el conjunto de validación es un dato y se conforman N conjunto de datos diferentes -, se decidió no implementarla, ya que el tiempo de computo se incrementa demasiado y los resultados obtenidos no superaban las técnicas de validación descritas anteriormente. El siguiente paso, una vez obtenidos los mejores modelos basados en las configuraciones y los procesos de entrenamiento y validación antes descritos, es obtener los pronósticos de ventas futuras.

3.4. pronósticos

En la sección 1.10, se describieron las cuatro técnicas para realizar pronósticos con los algoritmos de aprendizaje automático y que se utilizan en este estudio. En la implementación se consideran algunos detalles que se describen a continuación:

- Una vez que se tienen los modelos finales, se le entrega a cada algoritmo un vector, que corresponden a los doce últimos datos de las series de tiempo -no considerando el conjunto de prueba- con el cual se obtienen los pronósticos. Esto se realiza para todas las técnicas de pronósticos. En detalle:
- **Pronósticos Directos**: se entrena un modelo para cada horizonte h. Cada pronóstico corresponde al último valor que se obtiene de predecir el vector con los doce últimos datos de la serie de tiempo.
- **Pronósticos Recursivos**: se entrena un modelo. Con este modelo se obtiene todo el horizonte de pronósticos. Cada pronóstico corresponde al último valor que se obtiene de predecir el vector con los doce últimos datos de la serie de tiempo. Esta predicción o pronóstico obtenido se vuelve a concatenar a la matriz de datos para predecir los pronósticos sucesivos.
- **Pronósticos DirRec**: se entrena un modelo para cada horizonte h. Cada pronóstico corresponde al último valor que se obtiene de predecir el vector con los doce últimos datos de la serie de tiempo y los pronósticos obtenidos se vuelven a concatenar a la matriz de datos para predecir los pronósticos sucesivos.
- **Pronósticos MIMO (Multiples Inputs Multiples Outputs)**: Los múltiples pronósticos se obtienen al entregar al modelo ya entrenado y validado un vector de la longitud del horizonte de pronósticos, para este estudio, $h = 12$ -que corresponden a los doce últimos datos de las series de tiempo-.

Estas cuatro técnicas en conjunto con las técnicas de validación y las configuraciones e hiperparámetros señalados en las secciones anteriores entregan los pronósticos de las ventas de los doce meses siguientes. Una vez obtenidos los pronósticos, se evalúan las precisiones de los pronósticos considerando el conjunto de prueba - doce últimos valores de las series de tiempo -. Enfatizamos que este estudio se ha realizo mes por mes y durante un año -desde octubre del 2018 hasta octubre del 2019-. Es decir, cada mes se agrega el valor real de las ventas facilitado por la compañía y se vuelve a obtener los

pronósticos para un año. El propósito de repetir el procedimiento mes a mes, permitió identificar los algoritmos con mayor estabilidad y eficacia durante el tiempo. En la siguiente sección se describen los tipos de evaluación comúnmente utilizados para medir la precisión de los pronósticos.

3.5. evaluación de la precisión de pronósticos

Existen más de diez tipos de evaluación de modelos y medición de la precisión de los pronósticos. El enfoque de aprendizaje automático enfatiza la utilización de un solo tipo de evaluación y la más representativa al tipo de problema a resolver. Tomando este enfoque y considerando las medidas de evaluación que se utilizaron en los artículos que consideramos como base de este estudio [Makridakis et al., 2018b, Ahmed et al., 2010], se utilizaron las siguientes medidas de evaluación:

- Raíz del error cuadrático medio (RMSE)

$$\begin{aligned} RMSE =& \sqrt{MSE} \\ MSE =& \sum \frac{e_t^2}{N} \end{aligned} \tag{3.2}$$

- Error absoluto medio porcentual (MAPE)

$$MAPE = \frac{1}{N} \sum |\frac{e_t}{y_t}| 100 \tag{3.3}$$

- Error absoluto medio porcentual simétrico (sMAPE)

$$sMAPE = \frac{2}{h} \sum_{t=1}^{h} \frac{|e_i|}{|y_i| + |\hat{y}_i|} 100 \tag{3.4}$$

Donde N es la cantidad de datos, e_t es la diferencia entre el valor real de la serie de tiempo y_t y los pronósticos hasta t, dados por $\hat{y}$. h es el horizonte de pronósticos. El error

medio absoluto porcentual simétrico (sMAPE) se utilizó como base para este estudio, ya que es la medida de evaluación principal para evaluar los métodos participantes en la competencia M3 y M4 [Makridakis and M., 2000, Makridakis et al., 2018a].

La siguiente figura (3.6) resume el procedimiento completo para obtener los pronósticos de ventas de un año por delante utilizando los algoritmos de aprendizaje automático. En el siguiente capitulo, se presentan los resultados y las precisiones obtenidas.

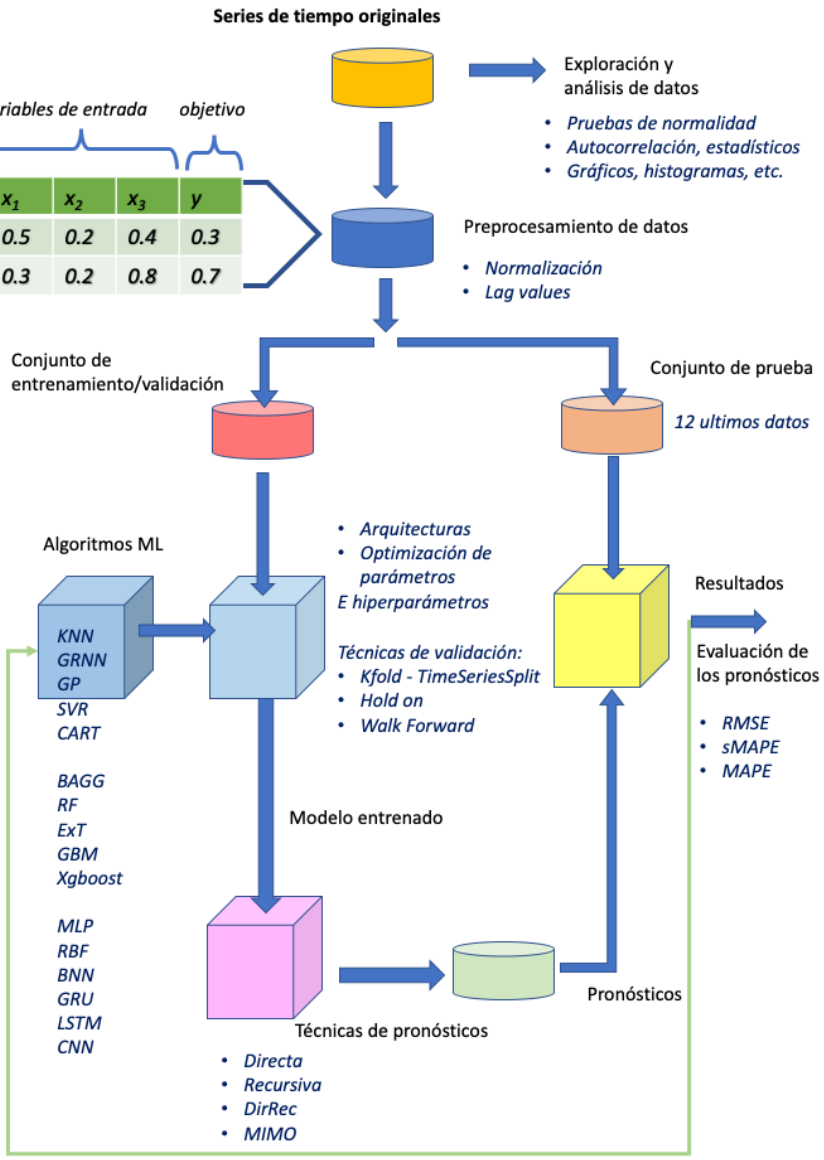


Figura 3.6. Diagrama del procedimiento para obtener los pronósticos de ventas para un año por delante con los algoritmos de aprendizaje automático. Este procedimiento se repite mes por mes.

Capítulo 4

Resultados

En este capítulo se presentan los resultados obtenidos para los pronósticos de ventas de un año de la compañía presentada en el capítulo anterior. Los resultados se presentan por mes o etapa. En la primera etapa se toman los datos mensuales desde enero del 2013 a octubre del 2017 como conjunto de entrenamiento y validación y se obtienen los pronósticos desde noviembre del 2017 a octubre del 2018 -conjunto de prueba -. En la segunda etapa se agrega el valor real de las ventas correspondiente al mes de noviembre del 2018 y se obtienen los pronósticos para los doce últimos meses -desde diciembre del 2017 a noviembre del 2018 -. Cada mes se agrega el valor real de ventas proporcionado por la compañía y se vuelven a obtener los pronósticos para $h = 12$. Este procedimiento duró un año, con la finalidad de medir y comprobar la capacidad de generalización de los diferentes algoritmos y metodologías cuando el fenómeno esta cambiando o se agregan datos nuevos. La última etapa contempla los pronósticos desde noviembre del 2018 a octubre del 2019.

Los resultados que se presentan en las siguientes secciones se clasificaron según:

- **Tipo de Preprocesamiento**: se obtienen los errores sMAPE de los pronósticos de un año, utilizando 1 lag , 2 lags y 3 lags o valores rezagados, la validación cruzada Kfold y la técnica de pronósticos DirRec; para todas las etapa. El objetivo inicial es determinar la cantidad de valores rezagados óptimos para el cálculo de los pronósticos.

- **Tipo de Validación**: una vez determinado la cantidad de valores rezados óptimos -en conjunto a las diferentes arquitecturas, hiperparámetros, parámetros y regularizaciones de cada algoritmo descrito en la sección 3.2.3-, se obtienen los pronósticos con los diferentes métodos de validación y regularización especificados en la sección 1.4 y 3.3 para la técnica de pronósticos DirRec. El objetivo es determinar la técnica de validación y regularización óptima para el caso de estudio.
- **Técnica de pronóstico**: Con los valores rezagados determinados anteriormente en conjunto con las técnicas de validación más importante para las ventas de la compañía, se obtienen los pronósticos con las técnicas de pronósticos restantes. A saber, la técnica Directa, Recursiva y MIMO -descritas en la sección 1.10-.

Todos los resultados y programas - en Python 3.7 y RStudio 1.2.5- se pueden encontrar en el enlace: TesisPronMultiVentasML. En este enlace se encuentran los directorios de cada mes o etapa del estudio.

4.1. resultados por preprocesamiento

En la primera etapa del procedimiento, se determinó la cantidad de datos rezagados *lag-values* necesarios y útiles para obtener los pronósticos más precisos -con base en las funciones ACF y PACF-. Además, se escogió -y con referencia al estudio de [Makridakis et al., 2018b]- la técnica de validación Kfold y la técnica DirRec para obtener los primeros pronósticos de ventas de un año por delante en todas las etapas. Las tablas 4.1, 4.2 y 4.3 que se presentan a continuación, muestran los errores sMAPE de los pronósticos obtenidos con cada algoritmo ML para los tres primeros meses o etapa y utilizando uno, dos y tres valores rezagados. Cabe señalar, que el error sMAPE presentado, corresponde al promedio del error de los pronósticos considerando las cuatro series de tiempo de las ventas de la compañía presentadas en la sección 3.1.

Modelo	Lag1	Lag2	Lag3
KNN	10.751	10.374	**9.62**
GP	11.606	**9.939**	10.886
SVR	11.391	10.413	**9.15**
GRNN	10.599	9.843	**9.124**
RBF	10.943	9.361	**8.711**
BRNN	11.596	10.443	**9.293**
CART	9.409	9.244	**9.022**
BAGG	11.532	10.142	**9.961**
RF	9.873	9.388	**8.840**
GBM	9.888	11.54	**9.634**
XGBoost	10.551	9.733	**9.423**
ExTree	10.527	**10.418**	11.276
MLP	12.215	**10.749**	10.987
GRU	11.694	**9.212**	9.772
LSTM	11.922	10.498	**9.062**
CNN	12.132	10.361	**9.494**

Los valores en negritas corresponden a los menores errores sMAPE.

Tabla 4.1. Errores sMAPE para los pronósticos de un año por delante $h = 12$ -desde noviembre del 2018 a octubre 2019-, considerando los tres primeros valores rezagados de cada serie de tiempo, la validación 10-fold y la técnica DirRec.

La tabla 4.1 muestra los errores sMAPE para los pronósticos desde noviembre del 2018 a octubre del 2019 y la siguiente tabla 4.2 muestra los errores sMAPE para los pronósticos desde octubre del 2018 a septiembre del 2019 -segunda etapa- utilizando los tres primeros valores rezagados como variables de entrada, la validación 10-fold y la técnica DirRec.

Modelo	Lag1	Lag2	Lag3
KNN	12.403	12.946	**12.533**
GP	11.177	11.186	**10.191**
SVR	11.479	11.672	**10.426**
GRNN	11.624	11.374	**11.264**
RBF	11.679	**10.847**	15.304
BRNN	13.659	12.882	**11.924**
CART	11.139	**10.929**	11.543
BAGG	**13.267**	13.542	13.313
RF	11.534	**11.357**	12.186
GBM	11.832	**11.477**	11.877
XGBoost	11.97	11.961	**11.737**
ExTree	**13.429**	15.536	15.929
MLP	12.344	11.599	**11.413**
GRU	**11.176**	11.559	11.642
LSTM	12.845	11.467	**11.138**
CNN	12.501	11.519	**11.171**

Tabla 4.2. Errores sMAPE para los pronósticos de un año por delante $h = 12$ -desde octubre del 2018 a septiembre 2019-, considerando los tres primeros *lag values* de cada serie de tiempo como variable de entrada, la validación 10-fold y la técnica DirRec.

Por último, la siguiente tabla muestra los errores sMAPE para los pronósticos desde los meses de septiembre del 2018 a agosto del 2019 utilizando los tres primeros lag values, la validación 10-fold y la técnica DirRec. Para ver los resultados de los demás meses o etapas, consultar el enlace: TesisPronMultiVentasML.

Modelo	Lag1	Lag2	Lag3
KNN	12.056	12.029	**10.765**
GP	12.29	**10.824**	11.402
SVR	12.479	12.222	**10.417**
GRNN	12.038	12.000	**11.216**
RBF	13.308	12.656	**11.069**
BRNN	12.973	12.558	**11.456**
CART	12.910	12.151	**12.141**
BAGG	12.561	12.587	**11.745**
RF	12.713	12.633	**11.581**
GBM	11.832	**11.477**	11.877
XGBoost	11.746	11.474	**10.222**
ExTree	13.829	12.426	**12.14**
MLP	13.301	14.030	**13.016**
GRU	13.465	13.175	**13.591**
LSTM	13.674	13.412	**13.250**
CNN	**13.040**	13.954	13.232

Tabla 4.3. Errores sMAPE para los pronósticos de un año por delante $h = 12$ -desde septiembre del 2018 a agosto 2019-, considerando los tres *lag values*, la validación 10-fold y la técnica DirRec.

La tabla 4.4 muestra los errores sMAPE promedio para todas las etapas -considerando las cuatro series de tiempo de las ventas de la compañía- utilizando la validación Kfold con $k = 10$, los tres valores rezagados de las series de tiempo con mayor valor de autocorrelación como variables de entrada y la técnica DirRec para obtener los doce pronósticos *futuros*.

Modelo	Lag1	Lag2	Lag3
KNN	11.836	11.415	**10.443**
GP	12.170	11.150	**10.894**
SVR	12.508	11.280	**10.118**
GRNN	11.779	10.849	**10.245**
RBF	12.362	11.990	**11.066**
BRNN	13.088	11.954	10.209
CART	11.576	10.892	**10.887**
BAGG	12.408	11.618	**11.262**
RF	11.894	11.022	**10.872**
GBM	11.449	11.580	**11.147**
XGBoost	11.644	11.084	**10.599**
ExTree	**12.622**	13.164	12.877
MLP	12.252	11.586	**11.412**
GRU	11.809	11.230	**11.100**
LSTM	12.421	11.709	**11.375**
CNN	12.792	11.611	**11.085**

Tabla 4.4. Errores sMAPE promedio para el proceso completo -pronósticos desde noviembre del 2017 a octubre del 2018 hasta los pronósticos desde noviembre del 2018 a octubre 2019- considerando los tres *lag values* de cada serie de tiempo, la validación Kfold y la técnica DirRec para pronosticar.

Como un primer acercamiento y analizando los resultados mostrados en esta sección, se puede apreciar que los mejores resultados -menores valores de sMAPE- se obtienen cuando se utilizan los tres valores rezagados como variables de entrada. Considerando este primer análisis, se determinó solo utilizar la matriz conformada con los tres valores rezagados -valores con mayor autocorrelación de las series de tiempo- para el resto del procedimiento y los resultados obtenidos se muestran en las siguientes secciones.

4.2. resultados por tipo de validación

En esta sección se muestran los resultados de los errores sMAPE promedio de los pronósticos de ventas para las tres últimas etapas considerando los tres valores rezagados, la técnica DirRec para obtener los pronósticos y las diferentes técnicas de validación consideradas y descritas en la sección 3.3. Además, se agregan los resultados obtenidos con los métodos ingenuos y los métodos estadísticos descritos en el capitulo 2. La tabla 4.5 muestra los errores sMAPE promedio para los pronósticos desde noviembre del 2018 a octubre 2019.

Modelo		Hold Out	10-fold	WF
Naïve2	8.835			
HWA	7.213			
HWM	7.385			
ARIMA	6.345			
THETA	8.234			
SES	8.778			
Holt	10.440			
ETS	7.449			
KNN		10.797	9.620	**9.394**
GP		15.915	10.886	**10.101**
SVR		10.075	**9.150**	10.126
GRNN		9.908	**9.124**	10.731
RBF		9.350	**8.540**	10.663
BRNN	9.293			
CART		15.238	**9.022**	9.582
BAGG		9.963	9.961	**9.756**
RF		9.091	**8.840**	9.485
GBM		15.868	**9.634**	10.716
XGBoost		11.258	**9.423**	10.014
ExTree		**9.529**	11.277	10.765
MLP		**8.810**	10.988	9.703
GRU		10.893	**9.771**	13.861
LSTM		9.993	**9.062**	9.606
CNN		9.513	9.494	**8.555**
Naïve	15.915			

Tabla 4.5. Errores sMAPE por tipo de validación para los pronósticos de un año $h = 12$ -desde noviembre del 2018 a octubre 2019-, considerando los tres *lag values* de cada serie de tiempo y la técnica DirRec para pronosticar.

La siguiente tabla -4.6- muestra los errores sMAPE promedio para los pronósticos de un año desde octubre del 2018 a septiembre 2019.

Modelo		Hold Out	10-fold	WF
Naïve2	9.118			
HWA	8.541			
HWM	8.018			
ARIMA	6.858			
THETA	9.791			
SES	11.103			
Holt	12.164			
ETS	9.539			
KNN		**11.216**	12.533	13.502
GP		12.984	**10.191**	11.957
SVR		12.139	**10.426**	12.633
GRNN		12.799	**11.264**	14.030
RBF		15.803	15.304	**12.961**
BRNN	11.924			
CART		13.805	15.929	**12.644**
BAGG		**12.736**	13.313	13.266
RF		12.232	12.186	**12.053**
GBM		12.080	**11.877**	13.250
XGBoost		14.321	**11.737**	13.616
ExTree		**11.643**	15.929	12.471
MLP		11.298	11.413	**10.058**
GRU		11.210	11.642	**9.319**
LSTM		10.914	11.138	**10.208**
CNN		14.656	11.172	**10.151**
Naïve	15.929			

Tabla 4.6. Errores sMAPE por tipo de validación para los pronósticos de un año $h = 12$ -desde octubre del 2018 a septiembre 2019-, considerando los tres *lag values* de cada serie de tiempo y la técnica DirRec para pronosticar.

Por último, se muestran los errores sMAPE promedio para los pronósticos desde septiembre del 2018 a agosto 2019.

Modelo		Hold Out	10-fold	WF
Naïve2	8.701			
HWA	8.468			
HWM	8.403			
ARIMA	7.061			
THETA	9.472			
SES	10.765			
Holt	11.193			
ETS	9.227			
KNN		**9.844**	10.765	11.415
GP		12.141	11.402	**11.114**
SVR		12.229	**10.417**	12.462
GRNN		**10.628**	11.216	12.570
RBF		12.126	11.070	11.473
BRNN	11.644			
CART		11.257	**10.148**	11.098
BAGG		**10.974**	11.745	11.006
RF		11.649	11.581	**11.064**
GBM		12.080	11.877	**11.286**
XGBoost		11.087	**10.222**	10.898
ExTree		**11.659**	12.141	12.318
MLP		11.566	13.649	**10.235**
GRU		11.268	13.591	**12.797**
LSTM		11.102	13.250	**11.015**
CNN		**9.736**	13.233	10.037
Naïve	12.141			

Tabla 4.7. Errores sMAPE por tipo de validación para los pronósticos de un año $h = 12$ (desde septiembre del 2018 a agosto 2019), considerando los tres *lag values* de cada serie de tiempo y la técnica DirRec para pronosticar

La tabla 4.8 muestra los promedios de los errores sMAPE para los pronósticos de las cuatro series de tiempo de la compañía; considerando el proceso completo - pronósticos desde noviembre del 2017 a octubre del 2018 hasta los pronósticos desde noviembre del 2018 a octubre 2019-. Los resultados corresponden a las diferentes técnicas de validación consideradas; utilizando los tres valores rezagados de las series de tiempo y la técnica DirRec para obtener los pronósticos.

Modelo		Hold Out	10-fold	WF
Naïve2	8.671			
HWA	9.771			
HWM	9.841			
ARIMA	8.295			
THETA	10.467			
SES	10.052			
Holt	12.884			
ETS	11.272			
KNN		10.474	**10.442**	11.102
GP		13.933	**10.894**	11.222
SVR		10.966	**10.118**	11.966
GRNN		10.872	**10.245**	12.539
RBF		13.018	13.366	**11.823**
BRNN	10.209			
CART		12.909	10.734	**10.657**
BAGG		11.052	11.262	**10.972**
RF		10.874	10.872	**10.595**
GBM		13.834	**11.147**	12.073
XGBoost		11.906	**10.599**	11.495
ExTree		**12.295**	12.877	12.402
MLP		12.789	11.412	**9.960**
GRU		11.131	11.100	**10.200**
LSTM		10.919	11.375	**10.064**
CNN		10.210	11.085	**9.660**
Naïve	14.172			

Tabla 4.8. Errores sMAPE promedio para los pronósticos de un año $h = 12$, considerando los *lag values* de cada serie de tiempo, las técnicas de validación y la técnica DiRec para pronosticar.

4.3. resultados promedio para la técnica DirRec

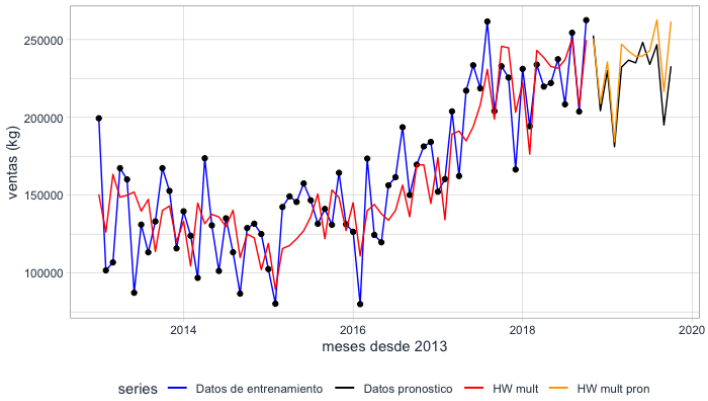
La tabla 4.9 muestra el error sMAPE promedio para los métodos estadísticos y los de aprendizaje automático utilizando la técnica DirRec, considerando los tres valores rezagados de las series de tiempo como variables de entrada. Los resultados mostrados, corresponden a los mejores pronósticos obtenidos para las diferentes técnicas de validación consideradas. Los errores sMAPE se muestran ordenados ascendentemente para todo el proceso.

Modelo	sMAPE	Validación
ARIMA	8.296	
Naïve2	8.671	
CNN	9.660	WF
HWA	9.771	
HWM	9.841	
MLP	9.960	WF
SES	10.051	
LSTM	10.064	WF
SVR	10.118	Kfold
BRNN	10.209	
GRNN	10.245	Kfold
KNN	10.442	Kfold
THETA	10.467	
RF	10.594	WF
XGBoost	10.599	Kfold
CART	10.657	WF
BAGG	10.972	WF
GP	10.894	Kfold
GRU	11.100	Kfold
GBM	12.074	Kfold
ETS	11.272	
RBF	11.823	WF
ExTree	12.452	WF
Holt	12.884	
Naïve	14.172	

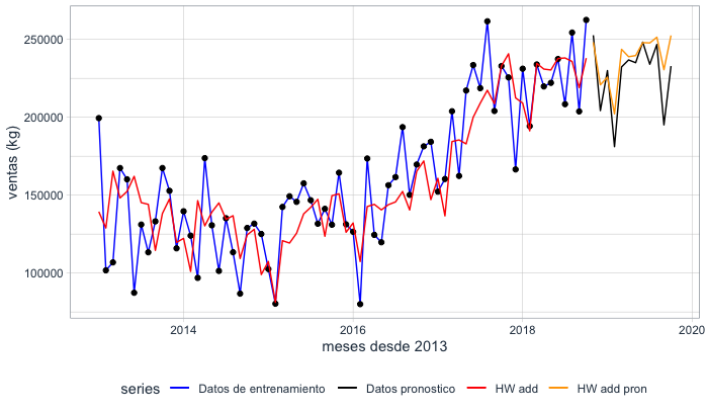
Tabla 4.9. Errores sMAPE promedio para todo el proceso, considerando los *lag values* de cada serie de tiempo, las técnicas de validación y la técnica DiRec para pronosticar.

A continuación, se muestran algunas gráficas con los modelos que alcanzaron las mejores precisiones en los pronósticos -desde noviembre del 2018 a octubre 2019- en el

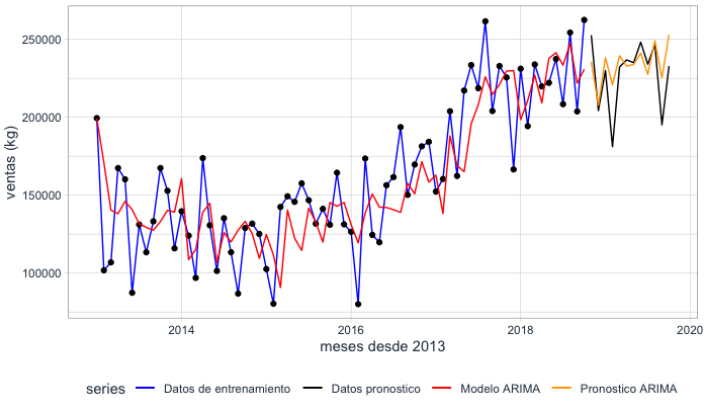
departamento de Chocolate .



(a) Pronósticos HoltWinters Multiplicativo para el departamento Chocolate -Nov 2018 a Oct19- sMAPE=4.45

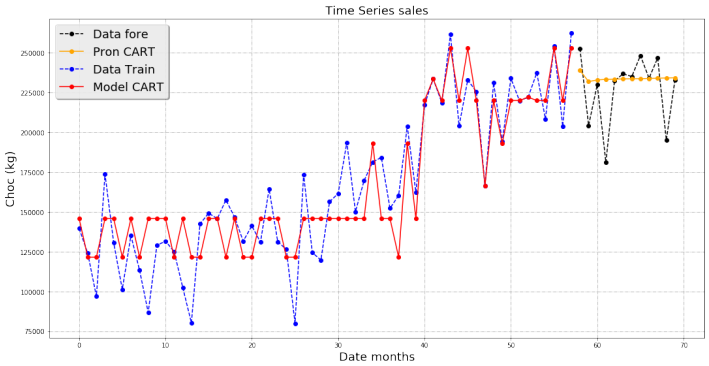


(b) Pronósticos HoltWinters Aditivo para el departamento Chocolate -Nov 2018 a Oct19- sMAPE=5.20

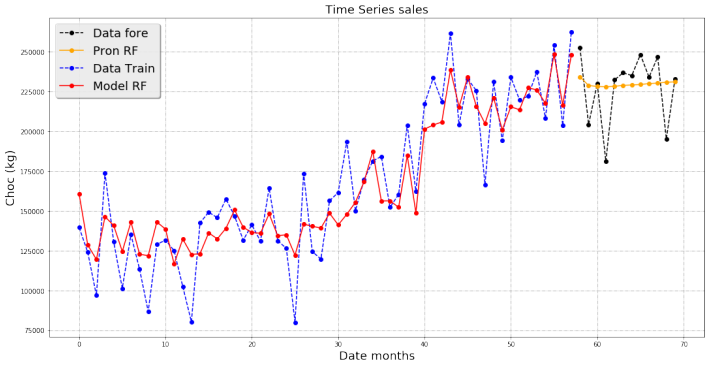


(c) Pronósticos ARIMA para el departamento Chocolate -Nov 2018 a Oct19- sMAPE=5.54

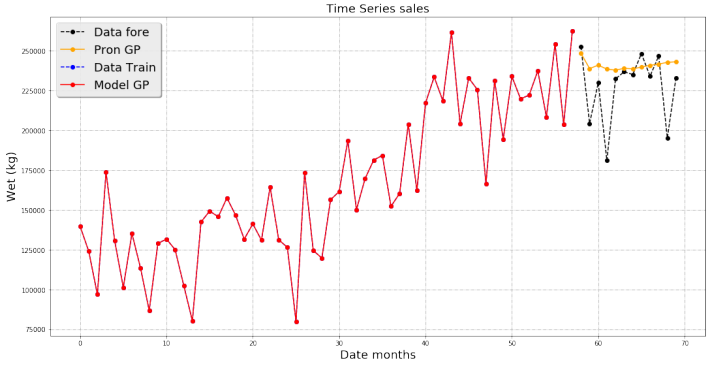
Figura 4.1. ejemplos de los mejores pronósticos -Nov2018 a Oct19- del departamento Chocolate



(a) Pronósticos CART para el departamento Chocolate -Nov 2018 a Oct19- validación WF, técnica DirRec, sMAPE=6.44



(b) Pronósticos RF para el departamento Chocolate -Nov 2018 a Oct19- validación WF, técnica DirRec, sMAPE=7.2



(c) Pronósticos GP para el departamento Chocolate -Nov 2018 a Oct19- validación WF, técnica DirRec, sMAPE=7.38

Figura 4.2. ejemplos de los mejores pronósticos -Nov2018 a Oct19- del departamento Chocolate

4.4. resultados por técnica de pronóstico

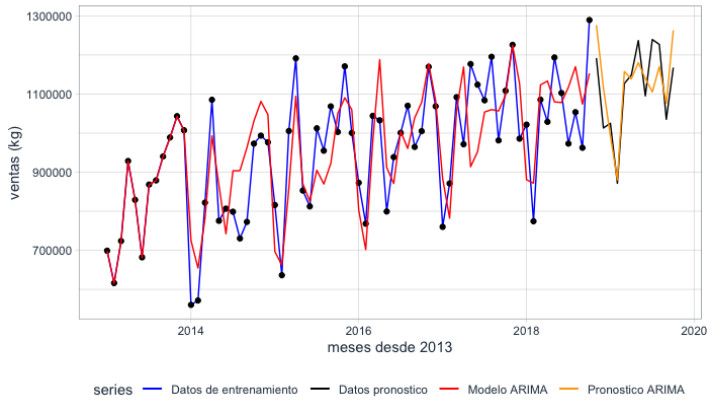
En esta sección se muestran las precisiones de los pronósticos utilizando las técnicas de pronósticos restantes que se describieron en la sección 1.10. A saber, la técnica Directa, Recursiva y MIMO. Para esta etapa y en base a los resultados obtenidos y mostrados en las secciones anteriores, se consideró utilizar la validación cruzada Kfold para los modelo no paramétricos y los algoritmos basados en árboles de regresión y la validación Walk forward para los algoritmos de aprendizaje profundo. La siguiente tabla -4.4.1- muestra los promedios de los errores sMAPE promedio ordenados ascendentemente incluyendo los errores RMSE y el error MAPE para la técnica recursiva. Seguido, se muestran algunas gráficas con los modelos que alcanzaron las mejores precisiones en los pronósticos -desde noviembre del 2018 a octubre 2019- para el total de las ventas de la compañía .

4.4.1. resultados recursiva

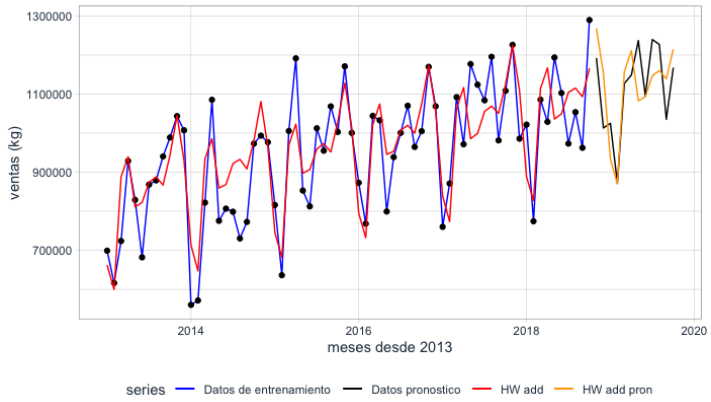
Modelo	sMAPE
ARIMA	8.296
Naïve2	8.671
HWA	9.771
HWM	9.841
CNN	10.049
SES	10.051
SVR	10.064
GP	10.262
GRNN	10.269
MLP	10.318
THETA	10.467
LSTM	10.491
RF	10.610
CART	10.646
KNN	10.743
GRU	10.829
XGB	10.996
ETS	11.272
BAGG	11.483
RBF	11.823
ExTree	11.982
BNN	12.277
GBM	12.393
Holt	12.884
Naïve	14.172

Modelo	RMSE
ARIMA	54864.595
Naïve2	56744.765
HWM	61067.318
HWA	61093.221
CNN	62665.960
SVR	63665.930
GRU	63990.999
LSTM	64456.015
MLP	64711.895
GP	64723.236
GRNN	65127.927
THETA	66095.532
SES	66552.214
ETS	67078.848
CART	68323.308
RF	69040.212
KNN	69257.924
XGB	69995.995
BAGG	71563.692
BNN	72821.323
RBF	72935.500
GBM	73403.043
ExTree	73576.803
Holt	77865.950
Naïve	88601.577

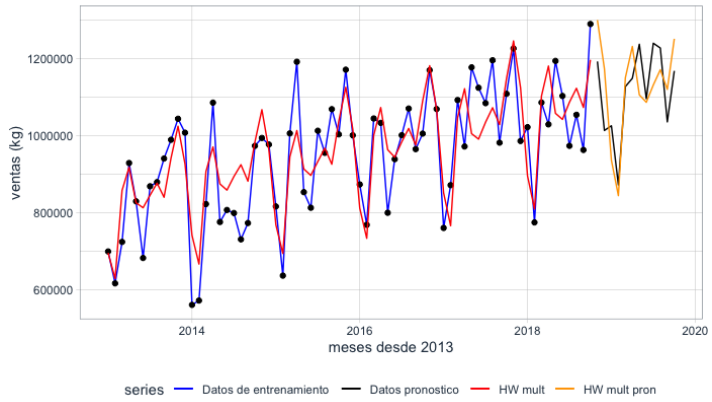
Modelo	MAPE
Naïve2	8.432
ARIMA	8.583
HWA	9.869
CNN	9.894
HWM	9.868
SVR	10.004
SES	10.060
GP	10.121
GRNN	10.171
MLP	10.188
LSTM	10.366
GRU	10.450
RF	10.509
CART	10.528
THETA	10.545
KNN	10.757
XGB	10.890
BAGG	11.070
RBF	11.537
ETS	11.648
ExTree	11.846
GBM	11.929
BNN	12.038
Holt	13.774
Naïve	14.114



(a) Pronósticos ARIMA para epara el Total de las ventas -Nov 2018 a Oct19- sMAPE=5.07

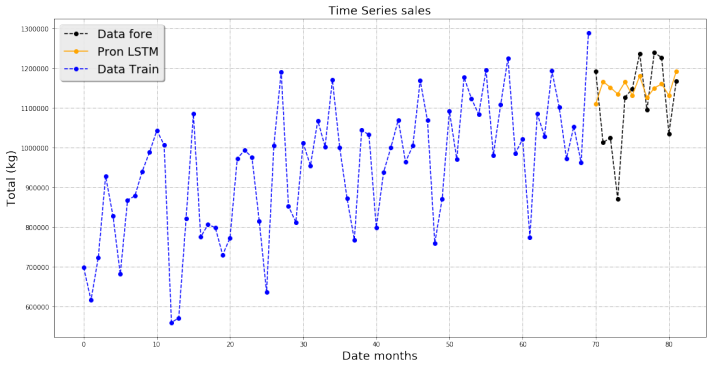


(b) Pronósticos HoltWinter aditivo para el Total de las ventas -Nov 2018 a Oct19- sMAPE=6.92

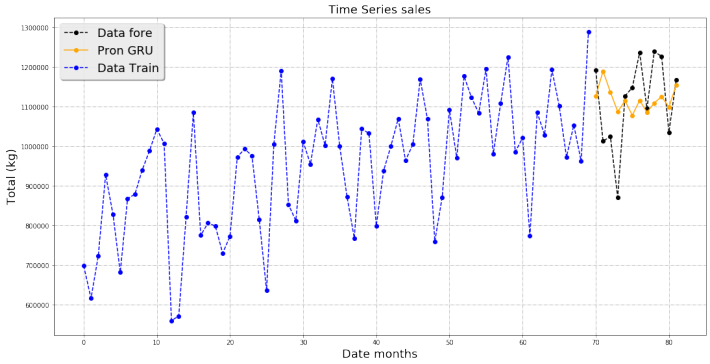


(c) Pronósticos HoltWinter multiplicvativo para el Total de las ventas -Nov 2018 a Oct19- sMAPE=7.77

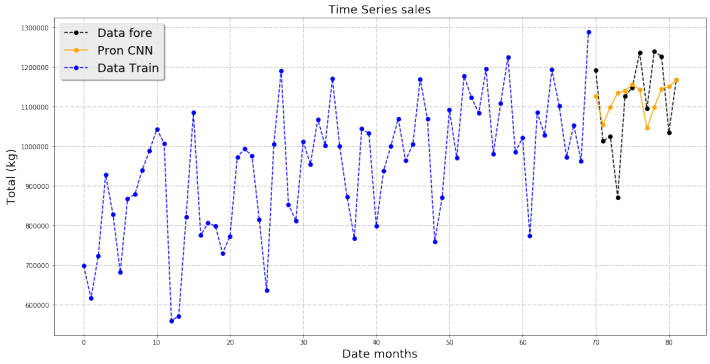
Figura 4.3. ejemplos de los mejores pronósticos -Nov2018 a Oct19- para el total de las ventas



(a) Pronósticos LSTM para el Total de las ventas -Nov 2018 a Oct19- validación WF, técnica Recursiva sMAPE=8.18



(b) Pronósticos GRU para el Total de las ventas -Nov 2018 a Oct19- validación WF, técnica Recursiva sMAPE=8.22



(c) Pronósticos CNN para el Total de las ventas -Nov 2018 a Oct19- validación WF, técnica Recursiva sMAPE=8.32

Figura 4.4. ejemplos de los mejores pronósticos -Nov2018 a Oct19- para el total de las ventas

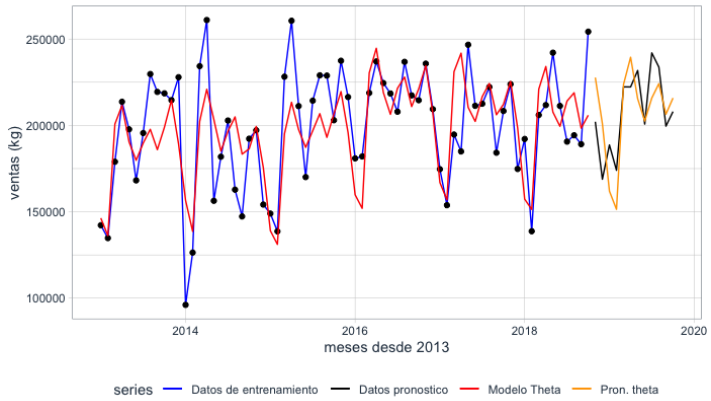
4.4.2. resultados directa

La tabla -4.4.2- muestra los promedios de los errores sMAPE ordenados ascendentemente incluyendo los errores RMSE y el error MAPE para la técnica directa.

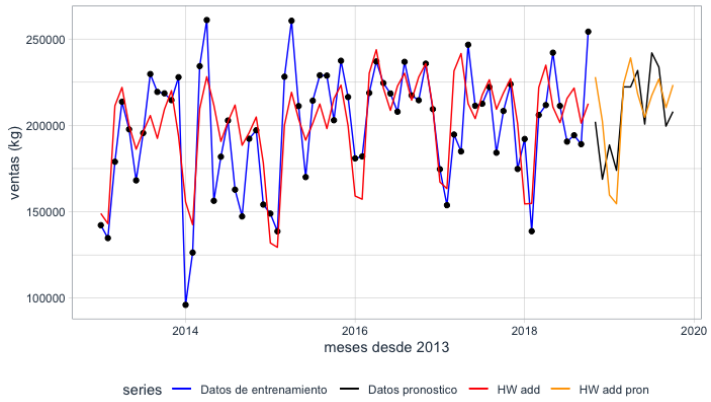
Modelo	sMAPE
ARIMA	8.296
Naïve2	8.671
HWA	9.771
HWM	9.841
SES	10.051
GRNN	10.317
THETA	10.467
SVR	10.599
GP	10.707
MLP	10.719
KNN	10.802
XGB	11.081
ETS	11.272
RF	11.335
GBM	11.465
LSTM	11.641
RBF	11.823
BAGG	11.983
BNN	12.277
CNN	12.439
GRU	12.594
Holt	12.884
CART	14.159
Naïve	14.172
ExTree	14.495

Modelo	RMSE
ARIMA	54864.595
Naïve2	56744.765
HWM	61067.318
HWA	61093.221
THETA	66095.532
SES	65499.495
ETS	67078.848
LSTM	71538.786
GRNN	71538.786
Holt	77865.950
BNN	72821.323
MLP	72902.562
RBF	72935.500
KNN	73179.623
GRU	73229.664
SVR	73229.664
GP	74060.010
XGB	77177.161
GBM	77741.575
RF	78234.925
BAGG	82163.449
CNN	86898.459
Naïve	88601.577
ExTree	91881.022
CART	92460.161

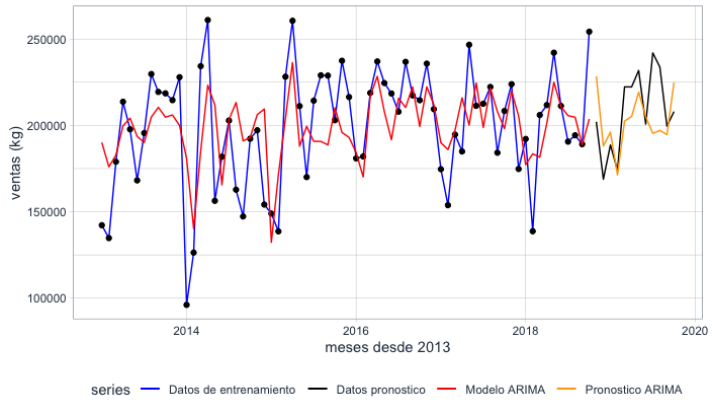
Modelo	MAPE
Naïve2	8.432
ARIMA	8.583
HWA	9.869
HWM	9.868
SES	10.060
GRNN	10.215
MLP	10.495
GRU	10.523
THETA	10.545
KNN	10.580
GP	10.762
GBM	10.799
SVR	10.820
XGB	10.916
RF	11.057
LSTM	11.062
CNN	11.503
RBF	11.537
ETS	11.648
BAGG	11.867
BNN	12.038
ExTree	13.440
CART	13.483
Holt	13.774
Naïve	14.114



(a) Pronósticos método Theta para el departamento Wet -Nov 2018 a Oct19- sMAPE=8.16

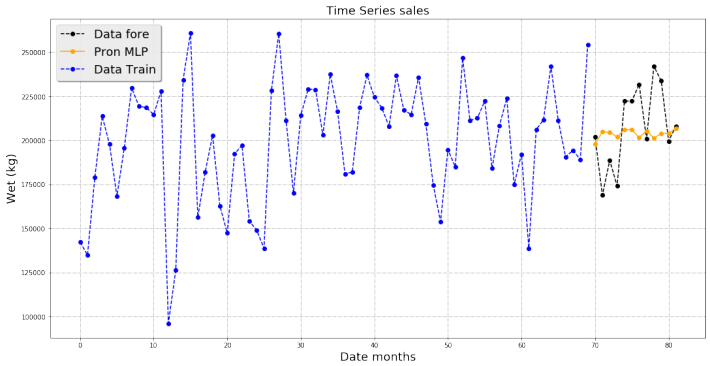


(b) Pronósticos HoltWinter aditivo para el departamento Wet -Nov 2018 a Oct19- sMAPE=8.35



(c) Pronósticos ARIMA para el departamento Wet -Nov 2018 a Oct19- sMAPE=8.49

Figura 4.5. ejemplos de los mejores pronósticos -Nov2018 a Oct19- para el departamento Wet



(a) Pronósticos MLP para el departamento Wet -Nov 2018 a Oct19- sMAPE=8.89

(b) Pronósticos GBM para el departamento Wet -Nov 2018 a Oct19- sMAPE=9.15

(c) Pronósticos GP para el departamento Wet -Nov 2018 a Oct19- sMAPE=9.15

Figura 4.6. ejemplos de los mejores pronósticos -Nov2018 a Oct19- para el departamento Wet

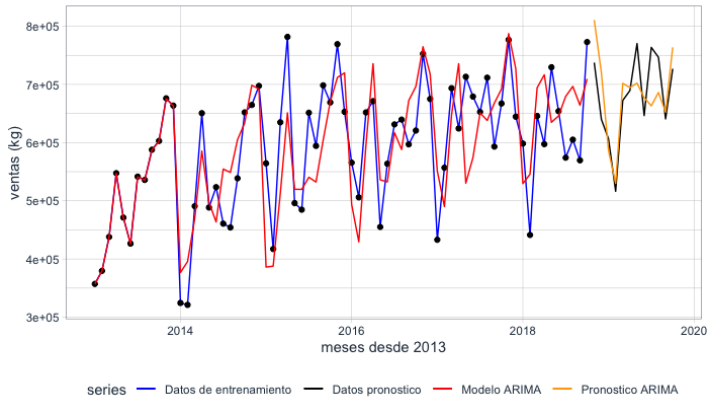
4.4.3. resultados MIMO

La siguiente tabla -4.4.3- muestra los promedios de los errores sMAPE ordenados ascendentemente incluyendo los errores RMSE y el error MAPE para la técnica MIMO.

Modelo	sMAPE
ARIMA	8.296
BAGG	8.564
GRNN	8.661
Naïve2	8.671
KNN	9.032
GBM	9.547
HWA	9.771
HWM	9.841
XGB	9.916
SES	9.938
RF	10.035
GP	10.091
THETA	10.467
CART	10.597
CNN	10.692
MLP	10.814
RBF	10.861
ExTree	10.914
GRU	11.165
BNN	11.235
ETS	11.272
LSTM	11.688
SVR	11.749
Holt	12.884
Naïve	14.172

Modelo	RMSE
BAGG	52212.999
KNN	54513.437
GRNN	54531.987
ARIMA	54864.595
Naïve2	56744.765
GBM	57522.401
GP	58953.528
RF	60817.227
HWM	61067.318
HWA	61093.221
XGB	61236.554
CNN	62221.121
MLP	63830.040
LSTM	64539.022
RBF	64598.890
GRU	64748.151
SVR	65490.910
SES	65499.495
ExTree	65529.501
THETA	66095.532
CART	66495.514
ETS	67078.848
BNN	72821.323
Holt	77865.950
Naïve	88601.577

Modelo	MAPE
BAGG	8.384
Naïve2	8.432
GRNN	8.443
ARIMA	8.583
KNN	8.789
GBM	9.171
XGB	9.429
GP	9.679
RF	9.744
HWA	9.869
HWM	9.868
SES	9.969
CART	10.200
CNN	10.431
THETA	10.545
MLP	10.618
RBF	10.629
ExTree	10.702
LSTM	10.746
GRU	10.869
SVR	11.164
ETS	11.648
BNN	12.038
Holt	13.774
Naïve	14.114



(a) Pronósticos Arima para el departamento Polvo -Nov 2018 a Oct19- sMAPE=6.28

(b) Pronósticos BAGG para el departamento Polvo -Nov 2018 a Oct19- validación WF, técnica MIMO, sMAPE=6.45

(c) Pronósticos CART para el departamento Polvo -Nov 2018 a Oct19- validación WF, técnica MIMO, sMAPE=6.74

Figura 4.7. ejemplos de los mejores pronósticos -Nov2018 a Oct19- para el departamento Polvo

(a) Pronósticos RF para el departamento Polvo -Nov 2018 a Oct19- validación WF, técnica MIMO sMAPE=6.99

(b) Pronósticos KNN para el departamento Polvo -Nov 2018 a Oct19- validación WF, técnica MIMO, sMAPE=7.42

(c) Pronósticos GP para el departamento Polvo -Nov 2018 a Oct19- validación WF, técnica MIMO, sMAPE=7.51

Figura 4.8. ejemplos de los mejores pronósticos -Nov2018 a Oct19- para el departamento Polvo

4.5. tiempo computacional

Makridakis en su estudio reporta el tiempo de computo -*Computational Complexity*- que requieren los diferentes métodos y algoritmos para obtener los pronósticos. Argumenta, que para aplicaciones donde se requiere determinar, por ejemplo, la demanda de cientos de productos de un inventario es poco práctico utilizar métodos de pronósticos que requieren mayor tiempo computacional. De la misma forma, el citado autor reporta que los métodos estadísticos necesitan menores requerimientos y tiempo computacional a pesar de las continuas mejoras del proceso de optimización en los métodos de aprendizaje automático [Makridakis et al., 2018b]. En esta sección mostramos el tiempo computacional requerido para hacer este estudio. Comenzamos mostrando el tiempo de computo requerido al implementar las diferentes técnicas de validación y la técnica DirRec para hacer los pronósticos. Es importante destacar que se utilizó un sistema con un procesador Intel Core i7 de 3.1 GHz y 16 GB 2133 MHz de memoria.

4.5.1. tiempo computacional técnica DirRec

La figura 4.9, muestra el tiempo promedio en segundos empleado -para todas las etapas- por los diferentes métodos utilizando la validación Hold Out y la técnica DirRec para obtener los pronósticos de un año por delante.

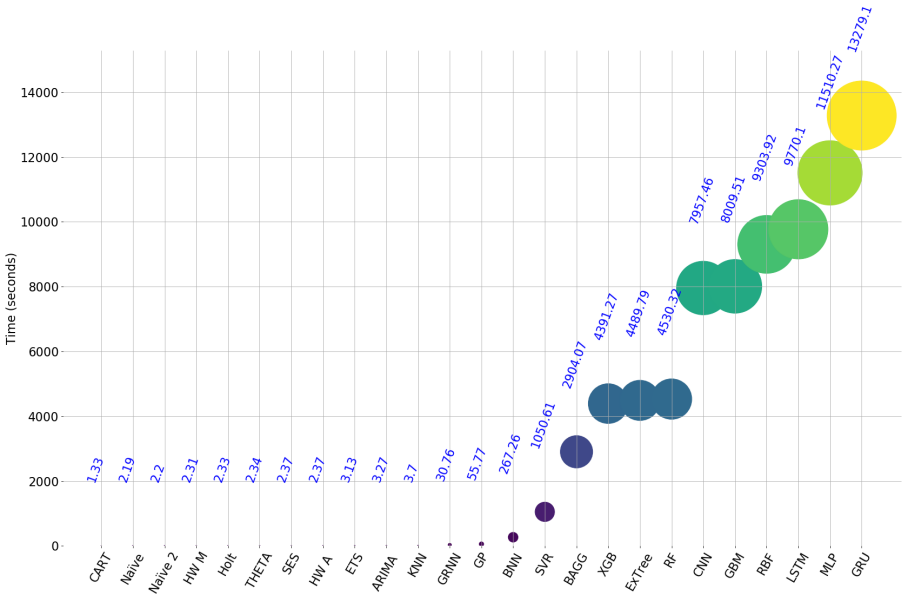


Figura 4.9. Tiempo computacional promedio para todas las etapas empleado por cada modelo para obtener los pronósticos de ventas utilizando la validación Hold Out y la técnica DirRec)

La figura 4.10, muestra el tiempo promedio en segundos empleado -para todas las etapas- por los diferentes métodos utilizando la validación Kfold y la técnica DirRec para obtener los pronósticos de un año por delante.

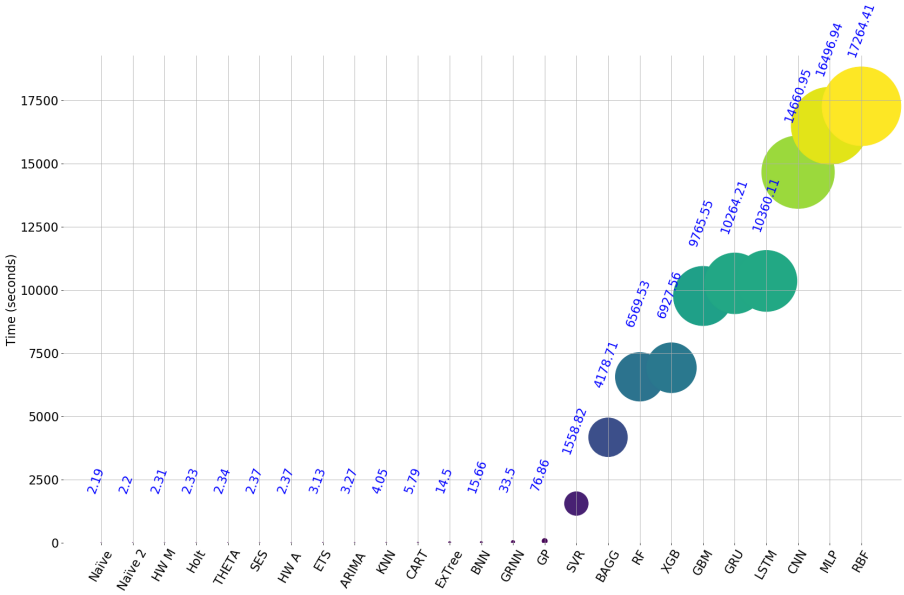


Figura 4.10. Tiempo computacional promedio para todas las etapas empleado por cada modelo para obtener los pronósticos de ventas utilizando la validación Kfold y la técnica DirRec)

La figura 4.11, muestra el tiempo promedio en segundos empleado -para todas las etapas- por los diferentes métodos utilizando la validación Walk Forward y la técnica DirRec para obtener los pronósticos de un año por delante.

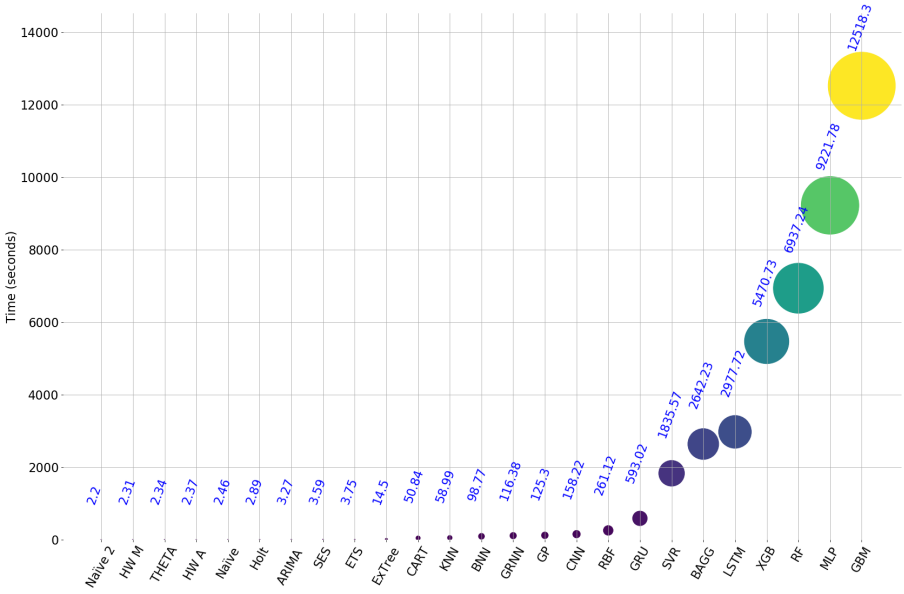


Figura 4.11. Tiempo computacional promedio para todas las etapas empleado por cada modelo para obtener los pronósticos de ventas utilizando la validación Walk Forward y la técnica DirRec

4.5.2. tiempo computacional técnica Directa

La figura 4.12, muestra el tiempo promedio en segundos empleado -para todas las etapas- por los diferentes métodos utilizando la técnica directa para obtener los pronósticos de un año por delante.

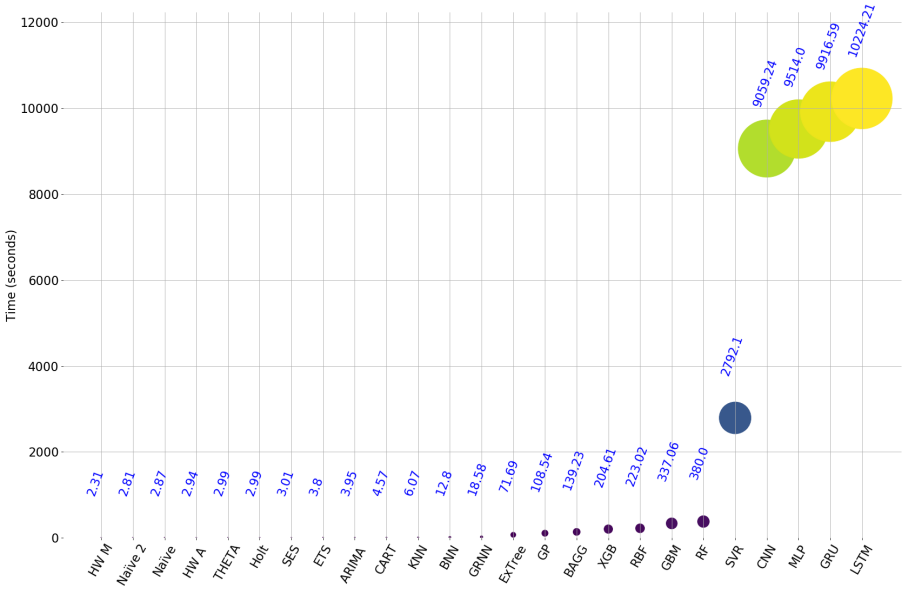


Figura 4.12. Tiempo computacional promedio para todas las etapas empleado por cada modelo para obtener los pronósticos de ventas utilizando técnica Directa

4.5.3. tiempo computacional técnica Recursiva

La figura 4.13, muestra el tiempo promedio en segundos empleado -para todas las etapas- por los diferentes métodos utilizando la técnica recursiva para obtener los pronósticos de un año por delante.

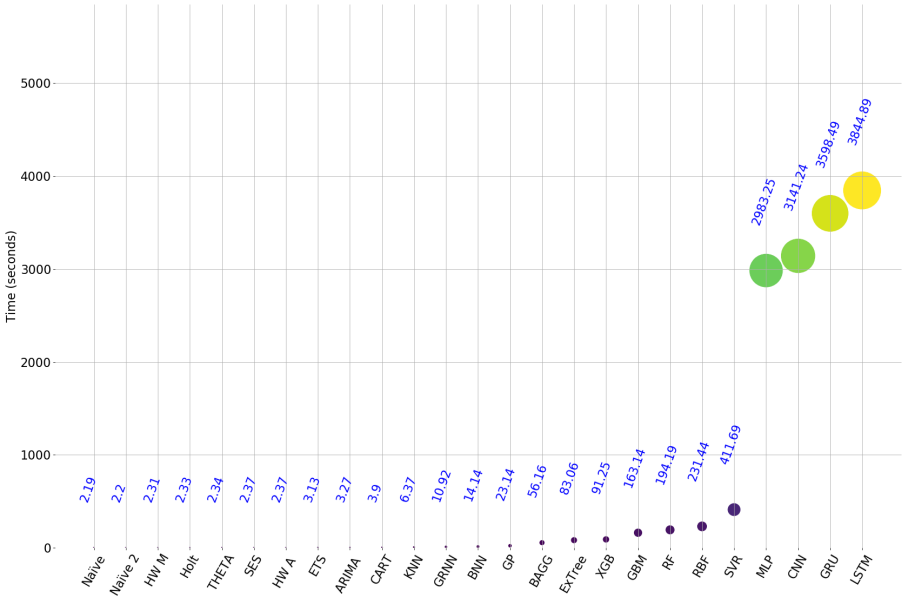


Figura 4.13. Tiempo computacional promedio para todas las etapas empleado por cada modelo para obtener los pronósticos de ventas utilizando técnica Recursiva

4.5.4. tiempo computacional técnica MIMO

La figura 4.14, muestra el tiempo promedio en segundos empleado -para todas las etapas- por los diferentes métodos utilizando la técnica MIMO -multiples inputs multiples outputs- para obtener los pronósticos de un año por delante.

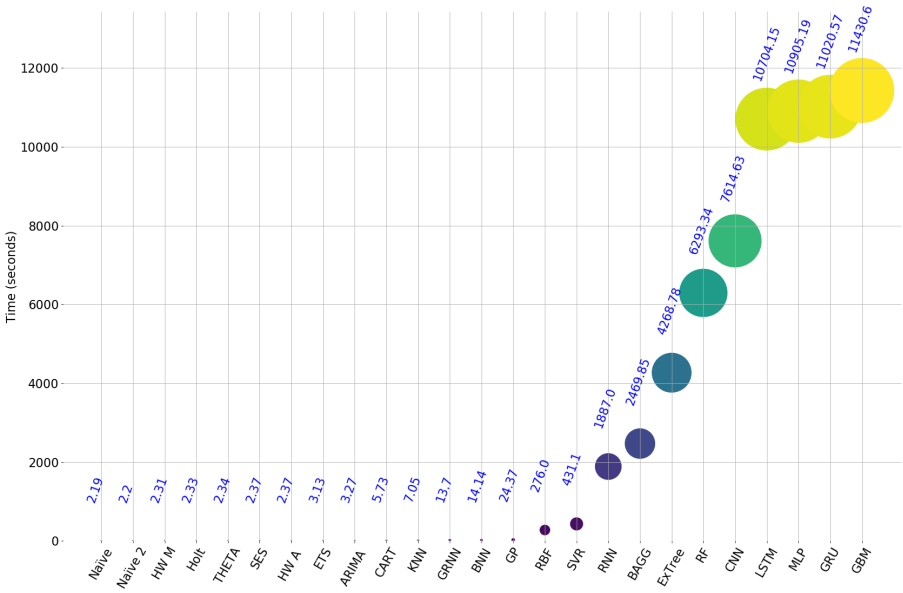


Figura 4.14. Tiempo computacional promedio para todas las etapas empleado por cada modelo para obtener los pronósticos de ventas utilizando técnica MIMO

4.6. discusión

La tabla 4.4 muestra que las mejores precisiones en los pronósticos se obtienen cuando se utilizan los tres primeros valores rezagados de cada serie de tiempo. Esto fue concluyente para todas las series de tiempo, para todos los tipos de validación y para todos los algoritmos de aprendizaje automático utilizados.

La tabla 4.8 muestra los resultados sMAPE promedio para todos los métodos de pronósticos con respecto a los diferentes métodos de validación propuestos. No se puede afirmar que una técnica de validación es superior a otra cuando se utiliza un modelo no parámetros o un método de ensamble basado en arboles de regresión. Sin embargo, se puede afirmar para este estudio de caso, que los mejores pronósticos con los métodos de aprendizaje profundo se obtienen cuando se utilizan las técnicas de entrenamiento y validación donde los datos se procesan de manera secuencial -validación Walk Forward-.

Con respecto a la técnica de pronósticos utilizada con los algoritmos de aprendizaje profundo. No se ven resultados que confirmen la eficacia de una técnica sobre otra. Cuantificando los resultados por tipo de técnica de pronósticos se destaca:

- 5 de los 10 mejores pronósticos utilizando la técnica DirRec corresponden a algoritmos ML. Destacando ConvNet y MLP.
- 5 de los 10 mejores pronósticos utilizando la técnica Recursiva corresponden a algoritmos ML. Destacando ConvNet y SVR.
- 4 de los 10 mejores pronósticos utilizando la técnica Directa corresponden a algoritmos ML. Destacando GRNN y SVR.
- 5 de los 10 mejores pronósticos utilizando la técnica MIMO corresponden a algoritmos ML. Destacando BAGG y GRNN.

Y como se muestra la sección 4.5, la técnica recursiva es la que menos tiempo computacional emplea. El caso puntual de la red neuronal LSTM, con la técnica DirRec demandan el mayor tiempo computacional para obtener los pronósticos. Un aproximado

de 4 horas en obtener los 12 pronósticos de una serie de tiempo.

Con respecto a los métodos estadísticos, se confirma la supremacía del método ARIMA, como el mejor método de pronósticos y de los más rápidos; alcanzando en el mes de octubre del 2019 el menor error sMAPE promedio de 6.86 %. Al igual, los métodos Holt Winters con estacionalidad aditiva y multiplicativa amortiguada, destacan entre los cinco métodos con los mejores pronósticos y en la rapidez de ejecución. Se debe considerar y destacar las precisiones utilizando el método ingenuo estacional *Naive2*, que en el proceso alcanzó un error sMAPE del 8.671 %, confirmando la estacionalidad anual de las ventas.

La red neuronal convolucional -ConvNet- se destaca como el mejor algoritmo de aprendizaje automático -utilizando la técnica DirRec- con un sMAPE promedio del 9.66 %. El algoritmo GRNN es el mejor algoritmo dentro de la categoría de los modelos no perimétricos con un sMAPE promedio del 8.661 % utilizando la técnica MIMO y el algoritmo *Bagging* con un 8.564 % -utilizando la técnica MIMO- es el mejor algoritmo basado en los métodos de ensamble utilizando arboles de regresión. Incluso es el único que supera al método ARIMA en el error promedio RMSE para la técnica MIMO.

Dentro de las propuestas iniciales de este estudio, se consideró probar redes neuronales, bajo el enfoque actual de aprendizaje profundo, es decir, redes neuronales con mayor número de capas ocultas y neuronas. La tabla 4.10, muestra los resultados obtenidos para los pronósticos de 12 meses con la red neuronal ConvNet utilizando diferentes configuraciones de capas ocultas y número de neuronas. En este caso particular, se obtiene la mejor precisión con dos capas ocultas y quinientas neuronas en cada capa oculta, lo cual sugiere -y considerando los resultados obtenidos en todo el estudio- que el uso de redes neuronales más *profundas* no necesariamente sobreajustarán los datos y como se sugiere en el estudio de [Makridakis et al., 2018b]; el uso de redes neuronales con una capa oculta y con $2N + 1$ neuronas no necesariamente arrojarán los mejores resultados. En la práctica, el número de capas oculta y neuronas -y por ende

los resultados- van a estar limitados a las capacidades computacionales disponible y el tiempo que los algoritmos requieran.

Capas	Neuronas	RMSE	MAPE	sMAPE
1	10	58386.327	22.52	25.845
1	50	38251.843	14.407	15.703
1	100	34064.643	13.033	13.989
1	300	22748.473	9.345	9.342
1	500	25340.230	9.415	8.994
1	1000	25703.163	9.291	9.286
1	2000	28871.826	10.936	11.373
2	10	74804.777	30.132	35.945
2	50	46402.983	17.460	19.49
‘2	100	40610.983	15.160	16.651
2	300	26052.221	10.398	10.666
2	500	24606.815	9.041	8.596
2	1000	42912.964	16.096	17.783
2	2000	30028.231	10.756	11.153

Tabla 4.10. precisiones para los pronósticos -octubre 2018 a septiembre 2019- para el departamento wet utilizando la red MLP y diferentes configuraciones de capas ocultas y número de neuronas.

Por último, se consideró utilizar solo regularización en los modelos de aprendizaje profundo. Es decir, no utilizar ninguna técnica de validación descrita en los capítulos anteriores. La intención, es ajustar y pronosticar los modelos considerando todos los datos disponibles de las series de tiempo y no interrumpir la dependencia temporal ni la importancia de los datos cuando se divide una serie de tiempo. La tabla 4.11 muestra los resultados de los pronósticos para todo el proceso utilizando solo regularización *Dropout* y la técnica recursiva en la red ConvNet y la red MLP[1]. Además, se utilizó -como *prueba*- la transformación de BoxCox para estabilizar la varianza en los datos.

[1]Solo se consideraron estas redes neuronales y la técnica recursiva, por la rapidez de ejecución. Esta implementación queda pendiente en las redes como GRU y las demás técnicas de pronósticos debido al tiempo que requieren.

Modelo	Validación	Técnica	sMAPE
ConvNet	WF	Recursiva	10.049
ConvNet	WF + BoxCox	Recursiva	9.919
ConvNet	No	Recursiva	9.760
ConvNet	No + BoxCox	Recursiva	**9.668**
MLP	WF	Recursiva	**10.318**
MLP	WF + BoxCox	Recursiva	10.488
MLP	No	Recursiva	10.601
MLP	No + BoxCox	Recursiva	10.463
LSTM	WF	Recursiva	10.491
LSTM	WF + BoxCox	Recursiva	10.696
LSTM	No	Recursiva	**9.653**
LSTM	No + BoxCox	Recursiva	10.521

Tabla 4.11. precisiones utilizando la técnica recursiva, regularización Dropout -no utilizando técnica de validación- y la transformación BoxCox para la red neuronal ConvNet, MLP y LSTM

Como se muestra en la tabla 4.11, el utilizar solo regularización -Dropout- y la transformación Boxcox aumentan la precisión de los pronósticos en una red ConvNet. Estos resultados sugieren que las transformaciones como BoxCox o el uso de la diferenciación en las series de tiempo para reducir la tendencia o estacionalidad, podrían ayudar a mejorar las precisiones de los pronósticos. Hay que enfatizar que la motivación inicial de este estudio es probar el desempeño de los algoritmos de aprendizaje automático en sus configuraciones y enfoques *más puros* y estas últimas pruebas consideradas se presentan como un adelanto a estudios futuros.

Conclusiones

Los buenos resultados y la eficacia que se les atribuye a los algoritmos de aprendizaje automático se asocian principalmente a dos factores: i) la complejidad de los espacios de hipótesis que manejan y ii) el manejo de grandes conjuntos de datos. La complejidad del espacio de hipótesis provocará sobreajuste, que se puede evitar o reducir con las diferentes técnicas de regularización y validación. Con respecto al tamaño del conjunto de datos, se puede establecer una evidente contradicción. Primero, las series de tiempo utilizadas en este estudio contienen 84 datos, lo cual se considera un conjunto de datos pequeño. Lo mismo se puede afirmar de las series de tiempo de la competencia M3 -aquí las series de tiempo *más largas* tienen 150 datos en promedio-. Tomando estas consideraciones, se puede establecer que las series de tiempo que se utilizan para el enfoque de pronósticos son *conjuntos de datos pequeños* comparados con los conjuntos de datos que se utilizan, por ejemplo, para problemas de visión por computadora, procesamiento natural del lenguaje o reconocimiento de voz. Segundo, en el enfoque de pronósticos para series de tiempo de ventas -como es en este estudio- o series de tiempo del mundo industrial o económico, no necesariamente las series de tiempo largas son las más útiles o serán las que arrojen mejores pronósticos. Por el contrario, se privilegia la importancia de los datos -que pueden ser los datos más reciente si tomamos el enfoque estadístico- ante la cantidad de datos. Lo anterior nos lleva a concluir, después de analizar los resultados que se mostraron en el capitulo anterior; que la *importancia de los datos* es el factor predominante sobre la *cantidad de datos* cuando se realizan pronósticos con series de tiempo utilizando algoritmos de aprendizaje automático. En particular, plantearse una

metodología eficiente bajo el marco de ingeniería de características es el camino a futuro para mejorar las precisiones de los pronósticos con algoritmos de aprendizaje automático.

Los métodos estadísticos de pronósticos han demostrado -una vez más- su superioridad en la precisión y la rapidez al obtener pronósticos de múltiples pasos adelante. Asociado a un enfoque que lleva décadas de mejoras y el respaldo de un marco teórico solido -destacando la metodología Box-Jenkins- que abarca todas las etapas del modelado predictivo, desde la identificación y selección de parámetros del modelo hasta la selección y comparación de modelos.

Por último, tomando los argumentos anteriores, se concluye que tanto el tamaño o complejidad del espacio de hipótesis, las grandes arquitecturas -por ejemplo, gran cantidad de neuronas y capas ocultas en una red neuronal artificial- y el manejo de funciones no lineales en los algoritmos de aprendizaje automático, no son los factores primordiales que determinen un buen desempeño de dichos algoritmos en el enfoque de pronósticos de múltiples pasos. El buen desempeño se logra con una buena estrategia para determinar la importancia de los datos y utilizar las técnicas de entrenamiento y validación, que mantengan el orden o dependencia temporal de los datos.

Como se advirtió al inicio del documento, las decisiones que puedan tomar la compañía en cuanto a los pronósticos derivados del método aplicado, quedan a criterio de los respectivos responsables; los beneficios que la empresa puede obtener al aplicar los resultados dependen de varios factores financieros, y por lo tanto, es necesaria más información para emitir recomendaciones en la toma de medidas a nivel organizacional.

trabajo futuro

Dentro de los trabajos a futuro, que pueden derivar de la presente investigación, se sugiere utilizar una técnica de validación y selección del modelo que no dividan los datos. Esto mantendrá la importancia y las características temporales de los datos. Cómo se mostró, todas las técnicas de validación o validación cruzada crean subconjuntos de los datos perdiendo la importancia en el tiempo de estos.

Se puede emplear el aprendizaje de transferencia para optimizar el procesamiento y el cómputo de los pronósticos. También sería útil utilizar unidades de procesamiento externó o GPU; esto ayudaría a implementar heurísticas más complejas en los diferentes modelos con bajo costo computacional y de tiempo.

En el marco de ingeniería de características, se sugiere implementar como mejoras algunos tipos de preprocesamiento, como la transformación BoxCox para estabilizar la varianza o la diferenciación para estabilizar la tendencia y estacionalidad de las series de tiempo. Como lo muestra la tabla 4.11, existe una mejora cunado se aplica la transformación BoxCox en una red neuronal ConvNet.

Referencias

[Ahmed et al., 2010] Ahmed, N., Atiya, A., Gayar, N., and El-Shishiny, H. (2010). An empirical comparison of machine learning models for time series forecasting. *Econometric Reviews. 2010; 29(5–6):594–621.*

[Akaike, 1969] Akaike, H. (1969). Fitting autoregressive models for prediction. *Annals of the institute of statistical mathematics 21(1).*

[Allen, 1974] Allen, D. (1974). The relationship between variable selection and data augmentation and a method for prediction. *Technometrics 16(1), 125–127.*

[Altman, 1992] Altman, N. (1992). An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician 46(3), 175–185.*

[Assimakopoulos and Nikolopoulos, 2000] Assimakopoulos, V. and Nikolopoulos, K. (2000). The theta model: a decomposition approach to forecasting. *International Journal of Forecasting 16 (2000) 521–530.*

[Atkeson et al., 1997] Atkeson, C., Moore, A., and Schaal, S. (1997). Locally weighted learning. *Artificial intelligence review 11(1), 11–73.*

[Ben Taieb et al., 2010] Ben Taieb, S., Sorjamaa, A., and Bontempi, G. (2010). Multiple-output modeling for multi-step-ahead time series forecasting. *Neurocomput.*, 73(10-12):1950–1957.

[Bishop, 2006] Bishop, M. C. (2006). *Pattern Recognition and Machine Learning.* Springer.

[Borovykh et al., 2017] Borovykh, A., Bohte, S., and Oosterlee, K. (2017). Conditional time series forecasting with convolutional neural networks. In *Lecture Notes in Computer Science/Lecture Notes in Artificial Intelligence*, pages 729–730.

[Box et al., 1994] Box, G. E. P., Jenkins, G. M., and Reinsel, G. C. (1994). *Time Series Analysis, Forecasting and Control*. Hoboken:Wiley.

[Breiman, 1984] Breiman, L. (1984). *Classification and Regression Trees.* Routledge, New York, 1 edition.

[Breiman, 1996] Breiman, L. (1996). Bagging predictors. *Machine Learning*, 24(2):123–140.

[Breiman, 2001] Breiman, L. (2001). Random forests. *Machine Learning*, 45(1):5–32.

[Brown, 1959] Brown, R. G. (1959). *Statistical Forecasting for Inventory Control.* McGraw/Hill.

[Bühlmann and Hothorn, 2007] Bühlmann, P. and Hothorn, T. (2007). Boosting algorithms: Regularization, prediction and model fitting. *Statistical Science 22(4), 477–505.*

[Canova and Hansen, 1995] Canova, F. and Hansen, B. E. (1995). Are seasonal patterns constant over time? a test for seasonal stability. *Journal of Business and Economic Statistics*, 13(3):237–252.

[Chen and Guestrin, 2016] Chen, T. and Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA. ACM.

[Cheng et al., 2006] Cheng, H., Tan, P.-N., Gao, J., and Scripps, J. (2006). Multistep-ahead time series prediction. In Ng, W.-K., Kitsuregawa, M., Li, J., and Chang, K.,

editors, *Advances in Knowledge Discovery and Data Mining*, pages 765–774, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Cherkassky and Ma, 2004] Cherkassky, V. and Ma, Y. (2004). Practical selection of svm parameters and noise estimation for svm regression. *Neural Networks*, 17(1):113–126.

[Chollet, 2015] Chollet, F. (2015). Keras. https://github.com/fchollet/keras.

[Chollet, 2017] Chollet, F. (2017). *Deep Learning with Python*. Manning Publications Co., USA, 1st edition.

[Chung et al., 2014] Chung, J., Gülçehre, Ç., Cho, K., and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555.

[Cleveland et al., 1990] Cleveland, R., Cleveland, W., McRae, J., and Terpenning, I. (1990). Stl: A seasonal-trend decomposition procedure based on loess. *Journal of Official Statistics,6(1):3-73*.

[Columbus, 2019] Columbus, L. (2019). Round up of machine learning forecasts and market estimates for 2019.

[Crone et al., 2011] Crone, S. F., Hibon, M., and Nikolopoulos, K. (2011). Advances in forecasting with neural networks? empirical evidence from the nn3 competition on time series prediction. *International Journal of Forecasting*, 27(3):635 – 660. Special Section 1: Forecasting with Artificial Neural Networks and Computational Intelligence Special Section 2: Tourism Forecasting.

[Dorffner, 1996] Dorffner, G. (1996). Neural networks for time series processing. *Neural Network World*, 6:447–468.

[Efron, 1992] Efron, B. (1992). *Bootstrap Methods: Another Look at the Jackknife*, pages 569–593. Springer New York, New York, NY.

[Efron and Hastie, 2016] Efron, B. and Hastie, T. (2016). *Computer Age Statistical Inference: Algorithms, Evidence, and Data Science*. Cambridge University Press, New York, NY, USA, 1st edition.

[Fildes and Makridakis, 1995] Fildes, R. and Makridakis, S. (1995). The impact of empirical accuracy studies on time series analysis and forecasting. *Int. Statist. Rev. 63*.

[Fiorucci et al., 2016] Fiorucci, J., Pellegrini, T., Louzada, F., Petropoulos, F., and Koehler, A. (2016). Models for optimising the theta method and their relationship to state space models. *International Journal of Forecasting*, 32.

[Friedman, 2001] Friedman, J. (2001). Greedy function approximation: A gradient boosting machine. *The Annals of Statistics 29(5), 1189–1232.*

[Friedman and Hastie, 2000] Friedman, J. and Hastie, T. (2000). Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). *The annals of statistics 28(2), 337–407.*

[Gardner, 1985] Gardner, J. E. S. (1985). Exponential smoothing: The state of the art. *Journal of Forecasting, 4(1).*

[Geman et al., 1992] Geman, S., Bienenstock, E., and Doursat, T. (1992). Neural networks and the bias/variance dilemma. *Neural Computation, 5.*

[Gers and Schmidhuber, 2000] Gers, F. A. and Schmidhuber, J. (2000). Recurrent nets that time and count. In *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium*, pages 189–194 vol.3.

[Geurts et al., 2006] Geurts, P., Ernst, D., and Wehenkel, L. (2006). Extremely randomized trees. *Machine Learning*, 63(1):3–42.

[Gorr, 1994] Gorr, W. (1994). Research prospective on neural network forecasting. *International Journal of Forecasting. 1994; 10(1):1–4.*

[Gross et al., 2017] Gross, W., Lange, S., Boedecker, J., and Blum, M. (2017). Predicting time series with space-time convolutional and recurrent neural networks. In *Proceeding of european Symposium on Artificial Neural Network.*

[Géron, 2019] Géron, A. (2019). *Hands-on machine learning with Scikit-Learn and TensorFlow : concepts, tools, and techniques to build intelligent systems.* O'Reilly Media, Sebastopol, CA.

[Hastie et al., 2001] Hastie, T., Tibshirani, R., and Friedman, J. (2001). *The Elements of Statistical Learning.* Springer Series in Statistics. Springer New York Inc., New York, NY, USA.

[Hinton et al., 2012] Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580.

[Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation. 1997; 9(8):1735–1780.*

[Hofner et al., 2014] Hofner, B., Mayr, A., Robinzonov, N., and Schmid, M. (2014). Model-based boosting in r: A hands-on tutorial using the r package mboost. *Comput. Stat.*, 29(1-2):3–35.

[Holt, 1957] Holt, C. E. (1957). Forecasting seasonals and trends by exponentially weighted averages. *O.N.R. Memorandum 52.*

[Hyndman and Billah, 2003] Hyndman, R. and Billah, B. (2003). Unmasking the theta method. *International Journal of Forecasting 19 (2003) 278–290.*

[Hyndman and Khandakar, 2008] Hyndman, R. and Khandakar, Y. (2008). Automatic time series forecasting: The forecast package for r. *Journal of Statistical Software, Articles*, 27(3):1–22.

[Hyndman and Koehler, 2006] Hyndman, R. and Koehler, A. (2006). Another look at measures of forecast accuracy. *International Journal of Forecasting. 2006 22(4).*

[Hyndman, 2008] Hyndman, R. J. (2008). *Forecasting with exponential smoothing.* Springer.

[Hyndman and Athanasopoulos, 2018] Hyndman, R. J. and Athanasopoulos, G. (2018). *Forecasting: principles and practice.* Heathmont: OTexts.

[Ji et al., 2005] Ji, Y., Hao, J., Reyhani, N., and Lendasse, A. (2005). Direct and recursive prediction of time series using mutual information selection. In *Proceedings of the 8th International Conference on Artificial Neural Networks: Computational Intelligence and Bioinspired Systems*, IWANN'05, page 1010–1017, Berlin, Heidelberg. Springer-Verlag.

[Kohavi, 1995] Kohavi, R. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. *Proceedings of the 14th International Joint Conference on Artificial Intelligence -Volume 2.*

[Kwiatkowski et al., 1991] Kwiatkowski, D., Phillips, P. C., and Schmidt, P. (1991). Testing the Null Hypothesis of Stationarity Against the Alternative of a Unit Root: How Sure Are We That Economic Time Series Have a Unit Root? Cowles Foundation Discussion Papers 979, Cowles Foundation for Research in Economics, Yale University.

[LeCun et al., 2010] LeCun, Y., Kavukcuoglu, K., and Farabet, C. (2010). Convolutional networks and applications in vision. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, pages 253–256.

[MacKay, 1992a] MacKay, D. J. C. (1992a). *Bayesian Interpolation*, pages 39–66. Springer Netherlands, Dordrecht.

[MacKay, 1992b] MacKay, D. J. C. (1992b). A practical bayesian framework for backpropagation networks. *Neural Computation*, 4(3):448–472.

[Makridakis et al., 1982] Makridakis, S., Anderson, A., Carbone, R., Fildes, R., Hibdon, M., Lewandowski, R., Newton, J., Parzen, E., and Winkler, R. (1982). The accuracy of extrapolation (time series) methods: Results of a forecasting competition. *Journal of Forecasting*.

[Makridakis and M., 2000] Makridakis, S. and M., H. (2000). The m3-competition: results, conclusions and implications. *International Journal of Forecasting 16(4)*.

[Makridakis et al., 2018a] Makridakis, S., Spiliotis, E., and Assimakopoulos, V. (2018a). The m4 competition: Results, findings, conclusion and way forward. *International Journal of Forecasting*, 34(4):802 – 808.

[Makridakis et al., 2018b] Makridakis, S., Spiliotis, E., and Assimakopoulos, V. (2018b). Statistical and machine learning forecasting methods: Concerns and ways forward. *PLoS ONE 13(3)*.

[Mangasarian, 1969] Mangasarian, O. (1969). *Nonlinear Programming.* Mc Graw-Hill.

[Mc Cormick, 1983] Mc Cormick, G. P. (1983). *Nonlinear Programming: Theory, Algorithms, and Applications.* John Wiley and Sons, New York.

[Mercer, 1909] Mercer, J. (1909). Xvi. functions of positive and negative type, and their connection the theory of integral equations. *Philosophical Transactions of*

the Royal Society of London A: Mathematical, Physical and Engineering Sciences, 209(441-458):415–446.

[Moody and Darken, 1989] Moody, J. and Darken, C. J. (1989). Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1(2):281–294.

[Nikolopoulos et al., 2011] Nikolopoulos, K., Assimakopoulos, V., Bougioukos, N., Litsa, A., and Petropoulos, F. (2011). The theta model: An essential forecasting tool for supply chain planning. *Lecture Notes in Electrical Engineering*, 123:431–437.

[Ord and Fildes, 2012] Ord, J. K. and Fildes, R. (2012). *Principles of business forecasting.* South-Western College Pub.

[Pedregosa et al., 2011] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

[Pegels, 1969] Pegels, C. C. (1969). Exponential forecasting: Some new variations. *Management Science, 15.*

[Prechelt, 1996] Prechelt, L. (1996). Early stopping-but when? In Orr, G. B. and Müller, K.-R., editors, *Neural Networks: Tricks of the Trade*, volume 1524 of *Lecture Notes in Computer Science*, pages 55–69. Springer.

[Rasmussen and Williams, 2005] Rasmussen, C. E. and Williams, C. K. I. (2005). *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning).* The MIT Press.

[Rawat and Wang, 2017] Rawat, W. and Wang, Z. (2017). Deep convolutional neural networks for image classification: A comprehensive review. *Neural Computation*, 29(9):2352–2449. PMID: 28599112.

[Rumelhart et al., line] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986/10/09/online). Learning representations by back-propagating errors. *Nature*, 323(7):533–536.

[Russell and Norvig, 2009] Russell, S. J. and Norvig, P. (2009). *Artificial Intelligence: a modern approach*. Pearson, 3 edition.

[Schapire, 1990] Schapire, R. (1990). The strength of weak learnability. *Machine learning 5(2), 197–227.*

[Schapire and Freund, 2012] Schapire, R. and Freund, Y. (2012). Boosting: Foundations and algorithms. *The MIT Press.*

[Schwarz, 1978] Schwarz, G. (1978). Estimating the dimension of a model. *The annals of statistics 6(2).*

[Sorjamaa et al., 2007] Sorjamaa, A., Hao, J., Reyhani, N., Ji, Y., and Lendasse, A. (2007). Methodology for long-term prediction of time series. *Neurocomput.*, 70(16-18):2861–2869.

[Specht, 1991] Specht, D. F. (1991). A general regression neural network. *IEEE Transactions on Neural Networks*, 2(6):568–576.

[Srivastava et al., 2014] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958.

[Stone, 1977] Stone, M. (1977). An asymptotic equivalence of choice of model by cross-validation and akaike's information. *Journal of the Royal Statistical Society. Series B 39(1), 44–47.*

[Sun and Giles, 2001] Sun, R. and Giles, C. L. (2001). Sequence learning: from recognition and prediction to sequential decision making. *IEEE Intelligent Systems*, 16(4):67–70.

[Sun et al., 2014] Sun, Y., Wang, X., and Tang, X. (2014). Deep learning face representation from predicting 10,000 classes. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1891–1898.

[Sutskever, 2013] Sutskever, I. (2013). *Training Recurrent Neural Networks*. PhD thesis, University of Toronto, Toronto, Ont., Canada, Canada. AAINS22066.

[Sutskever et al., 2014] Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. *CoRR*, abs/1409.3215.

[Taylor, 2003] Taylor, J. W. (2003). Exponential smoothing with a damped multiplicative trend. *International Journal of Forecasting, 19*.

[van den Oord et al., 2016] van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A. W., and Kavukcuoglu, K. (2016). Wavenet: A generative model for raw audio. *CoRR*, abs/1609.03499.

[Vapnik, 2000] Vapnik, V. N. (2000). *The Nature of Statistical Learning Theory*. Springer.

[Williams and Zipser, 1995] Williams, R. J. and Zipser, D. (1995). Gradient-based learning algorithms for recurrent networks and their computational complexity. In Chauvin, Y. and Rumelhart, D. E., editors, *Backpropagation: theory, architectures, and applications*, chapter 13, pages 433–486. L. Erlbaum Associates Inc., Hillsdale, NJ, USA.

[Winters, 1960] Winters, P. R. (1960). Forecasting sales by exponentially weighted moving averages. *Management Science 6:324–42.*

[Wolf et al., 2011] Wolf, L., Hassner, T., and Maoz, I. (2011). Face recognition in unconstrained videos with matched background similarity. In *CVPR 2011*, pages 529–534.

[Zhang, 2003] Zhang, G. P. (2003). Time series forecasting using a hybrid arima and neural network model. *Neurocomputing 50*.