Métodos de ML para pronóstico de series de tiempo

Lic. Francisco Javier Hernández Velasco
Centro de Investigación en Matemáticas. Unidad Monterrey
Email: francisco.velasco@cimat.mx

Lic. Oscar Uriel Pérez Salazar
Centro de Investigación en Matemáticas. Unidad Monterrey
Email: oscar.salazar@cimat.mx

***Abstract*— En el presente trabajo se muestran alternativas para realizar pronósticos de valores futuros de series de tiempo, así como diversas arquitecturas de redes neuronales empleadas para cumplir con dicha tarea. Además, se muestra dos implementaciones de estos modelos.**

I. Introducción

Una serie de tiempo es una secuencia de valores ordenados que se obtuvieron cada cierto intervalo de tiempo de un fenómeno en particular. Algunas aplicaciones del análisis de este tipo de datos es la detección de anomalías, pronóstico de valores, entre otros. Algunas técnicas que se emplean para el análisis de estos datos son las siguientes:

- **Métricas estadísticas**: se extraen características tales como la periodicidad de los datos, la varianza, la mediana, etc. Estas características se pueden extraer de la serie de tiempo como tal, también conocido como *Time-domain analysis* ya que ya son datos que dependen del tiempo, o se pueden aplicar transformaciones como la transformada rápida de Fourier y obtener dichas características, haciendo con ello el cambio a *Frecuency-domain analysis*.
- **Técnicas de regresión**: se usa la dependencia que puede existir entre los datos que conforman la serie de tiempo con ellos mismos o con otras variables independientes. Algunos ejemplos de las técnicas de regresión utilizadas son la familia Autoregressive Moving Average (ARMA) o la regresi'on Bayesiana.
- **Modelos de Machine Learning**: se adaptaron los modelos de Machine Learning para ser usados adecuadamente con series de tiempo.
- **Modelos de Aprendizaje profundo** (Deep learning): se adaptaron y mejoraron arquitecturas de redes neuronales para analizar las series de tiempo.

Por lo anterior, se mostrarán 6 alternativas basadas en modelos de Machine Learning y Deep Learning para el pronóstico de valores, los cuales son: *Support Vector Regression*,*Multilayer Perceptron*, *Long Short-Term Memory Networks*, Gated Recurrent Units y *Emsamble learning*. Además, se mostrara el ajuste de estos modelos a dos series de tiempo en particular.

II. Características relativas a la temporalidad de las series de tiempo

Las series de tiempo tienen las siguientes características relacionadas con su temporalidad:

- **Tendencia** (Trend): describe el cambio que presentan los valores conforme el tiempo pasa, sin considerar ciclos.
- **Estacionalidad** (Seasonality): describe patrones recurrentes y persistentes que se presentan en los datos durante un periodo de tiempo fijo.
- **Ciclicidad** (Cyclic): similar a estacionalidad, describe patrones recurrentes y persistentes que se presentan en los datos pero no tienen un periodo de tiempo fijo.
- **Ruido** (Noise): describe el comportamiento irregular que se puede presentar en los datos.

El conocer estas características pueden ser el objetivo de analizar las series de tiempo. Además, también ayuda en el ajuste de un modelo que pueda explicar adecuadamente la serie de tiempo.

III. Modelos para pronóstico tradicionales

Algunos modelos de regresión tradicionales para realizar pronósticos de series de tiempo son contemplados en dos familias de modelos:

- **Modelos de suavizamiento exponencial**: las técnicas de suavizamiento se pueden ver como técnicas para separar la señal y el ruido lo más que se pueda, siendo la señal cualquier patrón causado por la dinámica natural del fenómeno estudiado y el ruido el comportamiento irregular o aleatorio afecta a la señal. Lo anterior trae consigo el supuesto de que cualquier serie de tiempo se puede explicar como la combinación de estos dos elementos, uno determinista el cual es la señal y uno aleatorio el cual es el ruido.
- **Modelos Autoregresivos de Media Móvil Integrados (ARIMA)**: también conocidos como modelos Box-Jenkins, están conformados por tres elementos: un elemento auto regresivo (AR), un elemento de media movil (MA) y un elemento que indica la diferenciación que debe aplicarse a la serie de tiempo para que esta tenga un comportamiento estacionario, también llamado elemento integrado (I). Este último elemento se agrega para que se cumpla un supuesto que requiere esta familia de modelos, un comportamiento estacionario en la serie de tiempo, es decir, que su media, su varianza y su estructura de correlación no cambie conforme pasa el tiempo.

Se puede notar que estas dos familias de modelos requieren que se cumplan sus supuestos para que el modelo ajustado al fenómeno estudiado genere pronósticos precisos. No obstante, estas supuestos no siempre se pueden asegurar y por ello han surgido otras alternativas tales como los modelos GARCH o

los modelos de Machine Learning que pueden no requerir que se cumplan tantos supuestos como estas familias de modelos.

IV. Tratamiento de los datos

Dada la característica temporal de las series de tiempo, los datos que se espera analizar como tal deben pasar por un tratamiento el cual se explica de forma general en los siguientes pasos:

Algorithm 1 Data Preparation

1: Dado un conjunto de datos del fenómeno a analizar.
2: Se indexan los datos a una variable de tipo fecha la cual a su vez es una secuencia ordenada de fechas con la misma temporalidad
3: **if** Se desea cambiar la temporalidad de los datos **then**
4: **if** Es una temporalidad más fina **then**
5: Se distribuye el valor en la temporalidad original de tal manera que al agrupar los valores en la temporalidad más fina se coincida con el valor en la temporalidad original.
6: **end if**
7: **if** Es una temporalidad menos fina **then**
8: Se agrupan los valores en las temporalidades originales que estén incluidas en la nueva temporalidad menos fina. La función de agrupación puede ser suma, promedio, etc.
9: **end if**
10: **end if**
11: **if** Existen valores nulos o desconocidos en los datos **then**
12: Se aplican técnicas de imputación de valores
13: **end if**
14: Se transforman los datos para disminuir el impacto que el ruido puede tener, ya sea con la transformada rápida de Fourier o una media móvil.
15: Se construyen variables que faciliten el ajuste del modelo, como las variables lagueadas o estadísticos de ventanas de tiempo móviles.
16: Se reescalan los valores normalizándolos o estandarizándolos, depende del modelo de interés a ajustar a los datos.
17: Se eliminan registros que presenten valores nulos o missing por la construcciones de variables del paso anterior.

Una vez aplicado este tratamiento a los datos ya se considera adecuado el proseguir con el ajuste de un modelo a las series de tiempo para generar pronósticos.

V. Métricas de evaluación

De forma similar a los modelos que generan estimaciones, existen métricas para evaluar los pronósticos de un modelo ajustado a una serie de tiempo. Algunas métricas popularmente empleadas son las siguientes:

- **Raíz del Error Cuadrático Medio** (RMSE)

$$RMSE = \sqrt{\frac{1}{n}\sum_{t=1}^{n}(A_t - F_t)^2}$$

- **Error Porcentual Absoluto Medio** (MAPE)

$$MAPE = \frac{1}{n}\sum_{t=1}^{n}\left|\frac{A_t - F_t}{A_t}\right| \times 100$$

- **Media del error absoluto** (MAE)

$$MAE = \frac{1}{n}\sum_{t=1}^{n}|A_t - F_t|$$

Donde n es el número de observaciones, A_t es el valor real en el tiempo t y F_t es el valor predicho en el tiempo.

Dado que el MAPE proporciona un error absoluto medio en porcentajes entre los valores reales y los valores predichos, lo que permite que un valor se interprete de manera comprensible en comparación, se escogió como metrica de evaluación de los modelos del presente trabajo. Un valor MAPE más bajo implica que el modelo es más preciso. Además, se graficará el pronóstico del modelo junto con los datos reales con el propósito adicional de obtener un método claro de visualización.

VI. Datos de prueba

Para probar los modelos, utilizaremos los datos de AirPassengers que contienen información mensual sobre el número de pasajeros aéreos de 1949 a 1960:

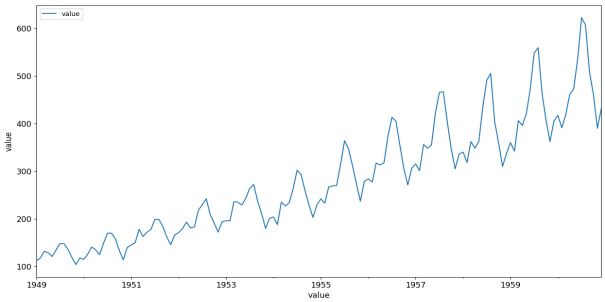

Fig. 1. Serie de tiempo de los datos AirPassengers.

Se puede observar que estos datos tienen características temporales visualmente notorias. Por otro lado, realizaremos las mismas pruebas con la información mensual sobre los precios de cierre de las acciones de Nvidia (NVDA) desde julio de 2018 hasta junio de 2024:

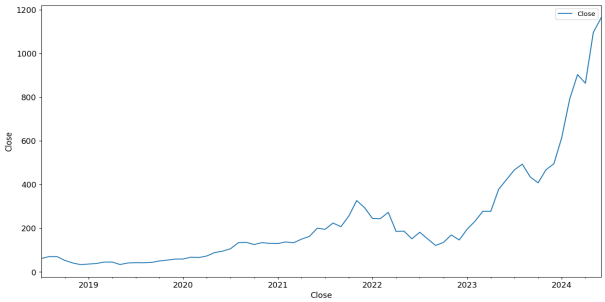

Fig. 2. Serie de tiempo de los datos de NVIDIA.

Estos Datos representan una gran dificultad de predecir, debido a ciertas características inherentes de las acciones y de la misma empresa como el alza del valor de NVIDIA por el creciente uso y desarrollo de sus productos, su volatilidad y su sensibilidad a eventos económicos.

VII. Modelos Autoregresivos de Media Móviles Integrados (ARIMA)

Como se mencionó anteriormente, los modelos de la familia ARIMA son un enfoque tradicional para ajustar modelos de

pronóstico los cuales generaron los siguientes resultados con los datos de prueba.

Análisis de resultados.

Predicción de ARIMA del número de vuelos para 1960:

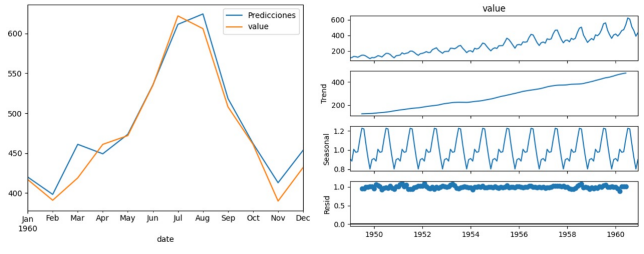

Fig. 3. Serie de tiempo de los datos de NVIDIA.

Se puede observar un muy buen ajuste a la predicción, casi alcanzando al resultado obtenido en MLP, un modelo que se explicará más adelante, con un MAPE de 2,8 %. ARIMA logra capturar bien la forma y tendencia de la serie de tiempo. Este desempeño es un tanto esperado ya que esta serie si puede transformarse para que sea estacionaria. Por otro lado, se tienen las predicciones de ARIMA para NVIDIA:

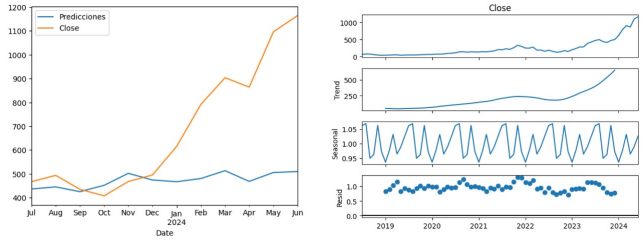

Fig. 4. Serie de tiempo de los datos de NVIDIA.

Para el caso de NVIDIA, ARIMA presenta las mismas dificultades que los demás modelos con una puntuación MAPE de 25,3 %. Pese a la mejoría en el MAPE no logra predecir correctamente ni la tendencia ni la forma de la explosión de NVIDIA a inicios de año como si lo logra GRU, un modelo que se mencionará más adelante. Contrario a los datos de Airpassengers, los datos de NVIDIA no pueden hacerse estacionarios ya que características como su varianza no es constante en el tiempo. Esto último se puede notar en el último año donde se presentan cambios drásticos en los valores.

VIII. Support Vector Regression (SVR)

Es una extensión de **Support Vector Machine (SVM)**, su objetivo es encontrar una función lineal, $f(x) = \langle w, x\rangle + b$ con $w \in X$(el espacio de datos de input), $b \in \mathbb{R}$, que pueda aproximar con cierta tolerancia ϵ a un conjunto de datos de entrenamiento. Dado que dicha función lineal puede no existir se contemplan variables de holgura, ξ y ξ^*, que denotan las desviaciones respecto a la tolerancia deseada, similarmente a como se hace en SVM al considerar un margen suave en donde se permiten ciertos errores. Con esto se tiene el siguiente problema a optimizar:

$$\text{mín } \frac{1}{2}||w|| + C\sum_{i=1}^{l}(\xi + \xi^*) \tag{1}$$

$$\text{subject to } y_i - \langle w, x_i\rangle - b \leq \epsilon + \xi_i \tag{2}$$

$$\langle w, x_i\rangle + b - y_i \leq \epsilon + \xi_i^* \tag{3}$$

$$\xi_i, \xi_i^* \geq 0 \tag{4}$$

Donde C determina el trade off entre lo aplanada (rígida) que puede ser la función f y las desviaciones respecto a la tolerancia deseada. Esto corresponde con lidiar con las denominadas funciones de perdida ϵ-insensitive, descritas como:

$$|\xi|_\epsilon = \begin{cases} 0 & \text{si } |\xi| \leq \epsilon \\ |\xi| - \epsilon & \text{otro caso} \end{cases} \tag{5}$$

Este planteamiento se puede visualizar en la siguiente imagen que se presenta en el articulo "A tutorial on support vector regression" creado por Smola y Schölkopf[1]:

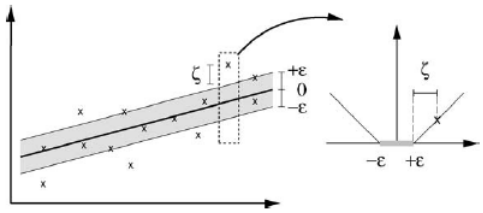

Fig. 5. Analogía a los margenes suaves que se usan en SVM.

Lo anterior se puede resolver abordando su respectivo problema dual, tal como lo señalan Smola y Schölkopf, lo cual consiste en aplicar multiplicadores de Lagrange para construir una nueva función la cual al aplicar condiciones de optimalidad de los puntos silla se puede llegar a lo siguiente:

$$f(x) = \sum_{i=1}^{l}(\alpha_i - \alpha_i^*)\langle x_i, x\rangle + b \tag{6}$$

Donde $\sum_{i=1}^{l}(\alpha_i - \alpha_i^*) = 0$ y $\alpha, \alpha^* \in [0, C]$. Lo anterior se conoce como Support Vector Expansion lo cual indica que w se puede describir como una combinación lineal de los datos de entrenamiento x_i. Por otro lado, para obtener a b se utilizan las condiciones Karush-Kuhn-Tucker (KKT) con las cuales se obtiene la siguiente expresión:

$$max\{-\epsilon + y_i - \langle w, x_i\rangle | \alpha_i < Co\alpha_i^* > 0\} \leq b \leq \tag{7}$$

$$min\{-\epsilon + y_i - \langle w, x_i\rangle | \alpha_i > 0o\alpha_i^* < C\} \tag{8}$$

Por otro lado, si se quiere que SVM sea un algoritmo no lineal entonces se aplican Kernels. El algoritmo 1 ilustra el esquema para optimizar el modelo.

Algorithm 2 SVR

1: Dado un conjunto de datos del fenómeno a analizar.
2: Se construye el problema primal a optimizar
3: Se usa el método de multiplicadores de Lagrange para obtener un nuevo problema a optimizar.
4: Se resuelve el nuevo problema utilizando condiciones de optimalidad de puntos silla.
5: Se obtiene b con ayuda de las condiciones KKT.
6: **if** Se quiere un modelo no lineal **then**
7: Se aplican kernels y se optimiza el nuevo modelo.
8: **end if**
9: Se regresa el valor de dicha función ajustada como el valor pronosticado.

Análisis de resultados

Empezando con los datos "AirPassengers", se obtuvieron los pronósticos que se muestran en la siguiente figura:

Fig. 6. Predicción del modelo SVR del número de vuelos para 1960.

Visualmente se aprecia que este modelo no ajusta particularmente bien los datos observados, sobre todo en los meses donde se presentan cambios más notorios. Por ello no sorprende que su puntuación MAPE sea de $41{,}29\,\%$. No obstante, características como la tendencia la logra identificar así como los comportamientos cíclicos que tiene esta serie de tiempo en particular.

Por otro lado, el pronóstico para los datos de NVIDIA se pueden observar en la figura 7 en la cual uno puede notar que fueron considerablemente peor dado que el valor de MAPE excede el $100\,\%$ y contrario a los datos anteriores, el efecto de la tendencia no logra ser identificado. Esto ultimo se podría explicar por el repentino aumento en el valor de las acciones de esta empresa.

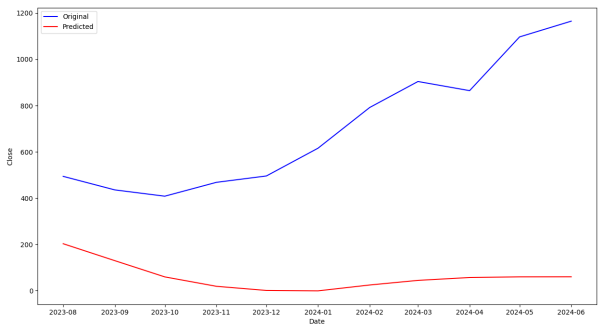

Fig. 7. Predicción del modelo SVR para NVIDIA.

IX. Perceptrón Multicapa (MLP)

El Perceptrón Multicapa es una red neuronal usada en clasificación y regresión. Se caracteriza por su estructura en capas:

- **Capa de Entrada:** Recibe los datos de entrada.
- **Capas Ocultas:** Procesan la información mediante funciones de activación (Sigmoide, Tangente Hiperbólica, ReLU, etc.). Puede haber varias capas ocultas.
- **Capa de Salida:** Produce el resultado final predicho.

El entrenamiento consiste en cuatro etapas:

- **Propagación Hacia Adelante:** Los datos se propagan a través de la red hasta obtener la salida.
- **Cálculo del Error:** Se compara la salida predicha con la real y se calcula el error (usualmente usando MSE).
- **Retropropagación:** El error se propaga hacia atrás, ajustando los pesos de las conexiones para minimizarlo.
- **Algoritmos de Optimización:** Se utilizan algoritmos como SGD, ADAM o RMSProp para optimizar los pesos y sesgos de la red.

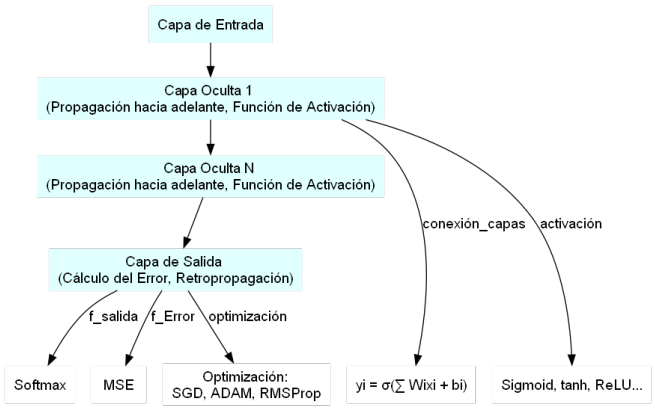

Fig. 8. Estructura de una red MLP

El MLP se adapta a la predicción de series de tiempo mediante técnicas que buscan capturar las dependencias temporales:

- **Ventanas Deslizantes:** Agrupan observaciones consecutivas como entrada para capturar la dependencia local.
- **Extracción de Características:** Métodos estadísticos y transformaciones (como la FFT) se usan para extraer información temporal relevante.

Además, con frecuencia es necesario la Normalización y Escalamiento para estabilizar el aprendizaje y mejorar la convergencia.

Algorithm 3 MLP para Pronóstico de Series de Tiempo

```
Inicializar pesos y sesgos aleatoriamente
Definir tamaño de ventana de tiempo
Dividir serie de tiempo en ventanas de tamaño fijo
Normalizar datos si es necesario
for cada época do
    for cada ventana de tiempo en datos de entrenamiento do
        Propagación hacia adelante
        Cálculo del error
        Retropropagación
    end for
end for
Evaluación del MLP:
Dividir y normalizar datos de prueba
Propagación hacia adelante y evaluar rendimiento
Predicción de nuevos datos:
Dividir y normalizar nuevos datos
Propagación hacia adelante para generar predicciones
```

- **Ventanas Deslizantes:** Agrupan observaciones consecutivas como entrada para capturar la dependencia local.
- **Extracción de Características:** Métodos estadísticos y transformaciones (como la FFT) se usan para extraer información temporal relevante.

En la figura 9 se ilustra la estructura y flujo de un MLP usado para series de tiempo.

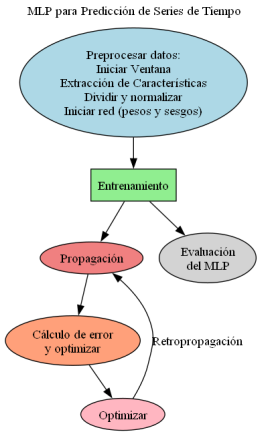

Fig. 9. Flujo de MLP en Series de tiempo.

Análisis de resultados

Para los datos “AirPassengers” obtenemos la predicción que se muestra en la figura 10.

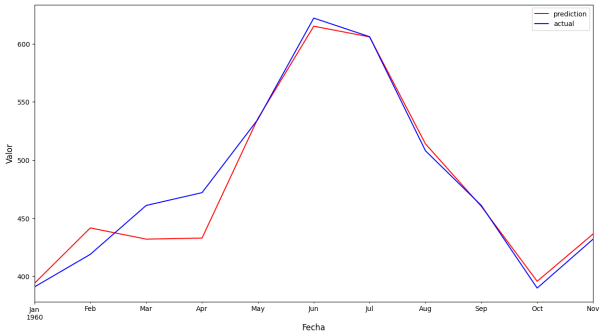

Fig. 10. Predicción del modelo MLP del número de vuelos para 1960.

Como podemos observar, el modelo se ajusta bien, sobre todo en la segunda parte del año. El modelo obtiene una puntuación MAPE de $2{,}3\,\%$, lo cual es notable.

Para el caso de los datos de NVIDIA los resultados se ven en la figura 11. El modelo es bastante peor obteniendo una puntuación MAPE de $55\,\%$, y una aproximación que se puede ver en la figura siguiente.

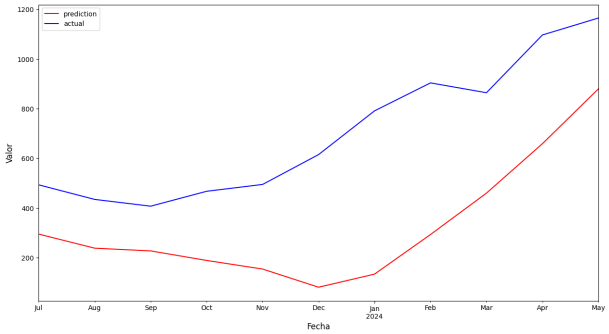

Fig. 11. Predicción del modelo MLP para NVIDIA.

Se observa un rendimiento pobre en la predicción, pese a que parece captar en general el movimiento, prediciendo un crecimiento, sin embargo, no logra capturar la magnitud de este crecimiento. Esto parece ser congruente con la explosión del valor de NVIDIA en 2024.

X. Redes Neuronales de Memoria de Corto y Largo Plazo (LSTM)

Antes de abordar las redes LTSM es necesario conocer las redes RNN. Las Redes Neuronales Recurrentes (RNN) se especializan en procesar datos secuenciales, se caracterizan por “recordar” información de entradas previas. Las RNN se componen por las siguientes componentes:

- **Unidades Recurrentes:** Procesan la entrada actual y la memoria previa (estado oculto) para generar una salida y actualizar la memoria.
- **Estado Oculto (h_t):** La ”memoria”de la red, almacena información de entradas pasadas para el contexto actual.
- **Pesos Compartidos:** Aprende patrones generales, no memoriza detalles, adaptándose a secuencias de diferentes longitudes.

En la figura 12 se observa la estructura base de una red con su arquitectura en el tiempo.

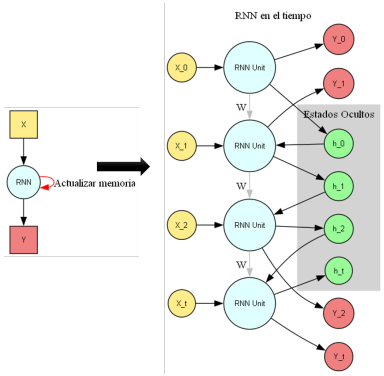

Fig. 12. Estructura de una Red RNN

Las RNN son buenas para predecir a corto plazo, a largo plazo presentan dificultades debido al desvanecimiento del gradiente. Los gradientes para actualizar los pesos se vuelven muy pequeños durante la retropropagación, mitigando su influencia en la predicción. Las Memorias a Largo y Corto Plazo (LSTM) son una extensión de las RNN diseñadas para abordar estas limitaciones.

Las LSTM se componen de células de memoria y múltiples puertas que regulan el flujo de información:

- **Puerta de Olvido:** Decide qué información descartar de la célula de memoria.

$$f_t = \sigma_g \left(W_f x_t + U_f h_{t-1} + b_f \right)$$

- **Puerta de Entrada:** Decide qué nueva información almacenar en la célula de memoria.

$$i_t = \sigma_g \left(W_i x_t + U_i h_{t-1} + b_i \right)$$

- **Célula de Memoria:** Actualiza su estado basado en la información nueva y la retenida.

$$c_t = f_t * c_{t-1} + i_t * \sigma_c \left(W_c x_t + U_c h_{t-1} + b_c \right)$$

- **Puerta de Salida:** Decide qué parte de la célula de memoria se utiliza para generar la salida.

$$o_t = \sigma_g \left(W_o x_t + U_o h_{t-1} + b_o \right)$$

- **Estado Oculto:** La salida de la célula de memoria, modulada por la puerta de salida.

$$h_t = o_t * \sigma_h \left(c_t \right)$$

Cabe señalar que σ_g es la función de activación sigmoide mientras que σ_c y σ_h es la tangente hiperbólica.

La célula de memoria se estructura con la información recopilada y la nueva, regulada por las puertas de entrada y olvido a través de funciones sigmoideas. La función tangente hiperbólica modula esta información, introduciendo no linealidades. La puerta de salida, también modulada por la tangente hiperbólica, determina la porción de la célula de memoria que se utiliza para generar el estado oculto final. Este método ayuda a mantener la memoria a largo plazo y a superar los problemas de las RNN.

Para entender mejor cómo funcionan las LSTM, podemos usar la analogía de un chef preparando una receta compleja. La célula de memoria es la receta completa que el chef sigue, mientras que la puerta de olvido decide qué ingredientes descartar, y la puerta de entrada controla qué nuevos ingredientes añadir. El estado del plato se actualiza combinando ingredientes antiguos y nuevos. La puerta de salida permite al chef probar una parte del plato para ajustar el sabor, y el estado oculto es el sabor actual del plato, utilizado para guiar los próximos pasos. Esta analogía ilustra cómo las LSTM retienen y ajustan información relevante a lo largo del tiempo.

El entrenamiento de la red se puede observar en el siguiente algoritmo:

Algorithm 4 LSTM para Pronóstico de Series de Tiempo

Inicialización:
Inicializar pesos y sesgos aleatoriamente: $W_f, U_f, b_f, W_i, U_i, b_i, W_c, U_c, b_c, W_o, U_o, b_o$
Inicializar estado oculto h_t y célula de memoria c_t
Entrenamiento:
for cada época **do**
  for cada entrada x_t en la secuencia de entrenamiento **do**
    Puerta de Olvido:
    $f_t = \sigma_g(W_f \cdot x_t + U_f \cdot h_{t-1} + b_f)$
    Puerta de Entrada:
    $i_t = \sigma_g(W_i \cdot x_t + U_i \cdot h_{t-1} + b_i)$
    Generar Candidatos de Memoria:
    $\tilde{c}_t = \tanh(W_c \cdot x_t + U_c \cdot h_{t-1} + b_c)$
    Actualizar la Célula de Memoria:
    $c_t = f_t * c_{t-1} + i_t * \tilde{c}_t$
    Puerta de Salida:
    $o_t = \sigma_g(W_o \cdot x_t + U_o \cdot h_{t-1} + b_o)$
    Actualizar el Estado Oculto:
    $h_t = o_t * \tanh(c_t)$
  end for
end for
Evaluación de la LSTM:
Dividir y normalizar datos de prueba
Propagación hacia adelante y evaluar rendimiento
Predicción de nuevos datos:
Dividir y normalizar nuevos datos
Propagación hacia adelante para generar predicciones

Análisis de resultados

En “AirPassengers” obtenemos la predicción que se muestra en la figura 13.

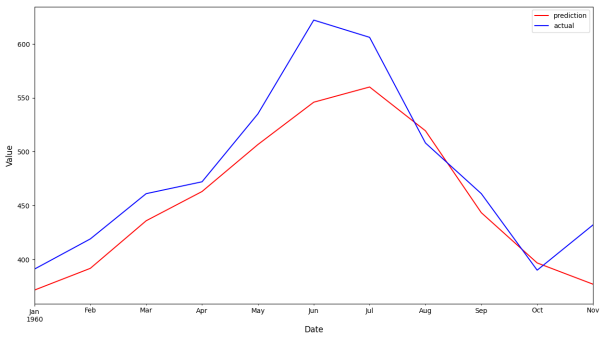

Fig. 13. Predicción del modelo LTSM del número de vuelos para 1960.

En este caso, aunque no es tan buena como la obtenida con MLP se obtiene una puntuación MAPE de 5.8 %, por lo que el modelo parece estar capturando bien la estructura de la serie.

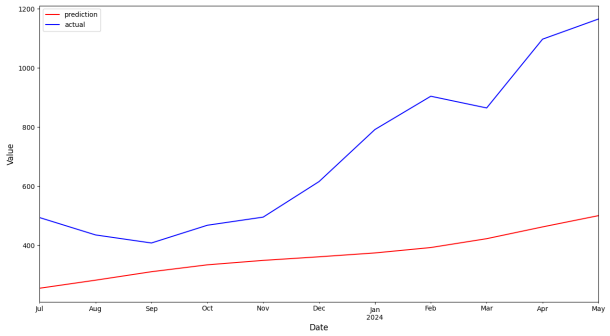

Fig. 14. Predicción del modelo MLP para NVIDIA.

En cuanto a NVIDIA, el modelo mejora la puntuación MAPE de MLP al obtener 43.8 %, sin embargo, sigue siendo un mal desempeño. A diferencia de MLP, LTSM logra capturar la tendencia la mayor parte del tiempo, no obstante, igualmente no logra replicar ni la "forma" ni la magnitud. Las prediciones se puden ver en la figura 14.

XI. Echo State Network (ESN)

Echo State Network (ESN) es un tipo de red RNN. Se caracteriza por su facilidad de entrenamiento, ya que solo se ajustan los pesos de salida, mientras que el resto de la red (llamado reservorio) se genera aleatoriamente y se mantiene fijo.

El reservorio ESN cumple dos funciones: transforma la entrada de forma no lineal y actúa como memoria para la entrada anterior. Al igual que los métodos del kernel, los reservorios proyectan datos en un espacio de dimensiones superiores, lo que facilita su separación. Cuando se aplica una entrada, la memoria genera un conjunto de señales internas que se combinan linealmente para producir una salida. El reservorio se genera aleatoriamente antes del entrenamiento y permanece fijo durante el proceso, solo se ajustan los pesos de las conexiones de salida.

Imaginemos un equipo de expertos en una biblioteca, cada uno con especialidades diferentes y una memoria única, asignados aleatoriamente a diferentes áreas. El reservorio en las ESNs funciona de manera similar. Cada vez que entra una nueva pregunta, cada experto la analiza de acuerdo con su propio conocimiento y evalúa las respuestas recientes de los demás expertos (memoria, actualización del estado), no solo la pregunta actual, sino también su contexto histórico y las tendencias más recientes. Análogamente, el reservorio transforma y retiene la información de las entradas pasadas. Esto permite una predicción adecuada capturando patrones y conexiones en los datos.

El objetivo de la ESN es minimizar la medida de error entre la salida de la red $y(n)$ y el valor de referencia $y^{\text{target}}(n)$. La medida de error típica es el RMSE:

$$E(y, y^{\text{target}}) = \frac{1}{N_y} \sum_{i=1}^{N_y} \sqrt{\frac{1}{T} \sum_{n=1}^{T} \left(y_i(n) - y_i^{\text{target}}(n)\right)^2}$$

La actualización del estado del reservorio se realiza mediante las siguientes ecuaciones:

$$\widetilde{x}(n) = \tanh\left(W^{\text{in}} \begin{bmatrix} 1 \\ u(n) \end{bmatrix} + W x(n-1)\right)$$

$$x(n) = (1-\alpha)x(n-1) + \alpha\widetilde{x}(n)$$

Donde W^{in} es la matriz de pesos de entrada, W es la matriz de pesos recurrentes, α es la tasa de filtración y $\tanh$ es la función de activación no lineal aplicada elemento a elemento.

La salida de la red se calcula mediante:

$$y(n) = W^{\text{out}} \begin{bmatrix} 1 \\ u(n) \\ \mathbf{x}(n) \end{bmatrix}$$

El pseudocódigo de la red ESN se observa en el algoritmo 5.

Algorithm 5 ESN básico

Inicialización:
Definir N_x, α, ρ, sparsity, Win_scale
Inicializar $\mathbf{W}^{\text{in}}$ y $\mathbf{W}$ aleatoriamente
Escalar $\mathbf{W}$ con ρ
Inicializar estado del reservorio $\mathbf{x}$ en cero
Entrenamiento:
for cada par $(\mathbf{u}(n), \mathbf{y}^{\text{target}}(n))$ en el conjunto de entrenamiento **do**
    $\tilde{\mathbf{x}}(n) = \tanh(\mathbf{W}^{\text{in}} \begin{bmatrix} 1 \\ \mathbf{u}(n) \end{bmatrix} + \mathbf{W}\mathbf{x}(n-1))$
    $\mathbf{x}(n) = (1-\alpha)\mathbf{x}(n-1) + \alpha\tilde{\mathbf{x}}(n)$
    Almacenar $\mathbf{x}(n)$ y $\mathbf{y}^{\text{target}}(n)$
end for
Calcular $\mathbf{W}^{\text{out}}$ usando regresión lineal minimizando $E(\mathbf{y}, \mathbf{y}^{\text{target}})$
Predicción:
for cada nueva entrada $\mathbf{u}(n)$ **do**
    $\tilde{\mathbf{x}}(n) = \tanh(\mathbf{W}^{\text{in}} \begin{bmatrix} 1 \\ \mathbf{u}(n) \end{bmatrix} + \mathbf{W}\mathbf{x}(n-1))$
    $\mathbf{x}(n) = (1-\alpha)\mathbf{x}(n-1) + \alpha\tilde{\mathbf{x}}(n)$
    $\mathbf{y}(n) = \mathbf{W}^{\text{out}} \begin{bmatrix} 1 \\ \mathbf{u}(n) \\ \mathbf{x}(n) \end{bmatrix}$
end for
Evaluar ESN

Los hiperparámetros de las redes ESN son los siguientes:

- **Tamaño del reservorio (N):** Número de neuronas en el reservorio. Mayor tamaño mejora el rendimiento (evitando el sobreajuste).
- **Esparsidad:** Proporción de elementos no nulos en las matrices de pesos. Mejora ligeramente el rendimiento y reduce costos computacionales.
- **Distribución de elementos no nulos:** Afecta la riqueza del espacio de señales (uniforme, bi-valuada, normal).

- **Radio espectral (ρ):** Valor absoluto máximo de los autovalores de la matriz del reservorio. Crucial para la propiedad de estado de eco.
- **Escala de entrada (Win_scale):** Determina la magnitud de la señal de entrada al reservorio. Afecta la no linealidad de las respuestas.
- **Tasa de filtración (α):** Controla la velocidad de actualización del estado del reservorio. Se ajusta a la dinámica temporal de las señales de entrada y salida.

Análisis de resultados

Para "AirPassengers" las predicciones se muestran en la figura 15. Se puede observar un rendimiento inferior a los otros modelos, con una puntuación MAPE de 14.6 %.

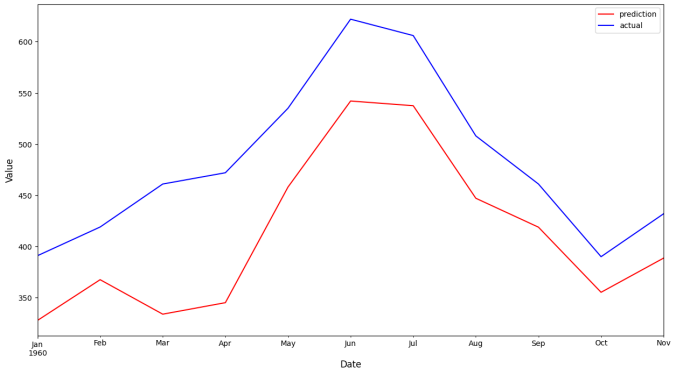

Fig. 15. Predicción del modelo ESN del número de vuelos para 1960.

En cuanto a NVIDIA, el modelo presenta las mismas o más dificultades que los modelos anteriores, con una puntuación MAPE de 58,7 %. Claramente, el modelo es más preciso a corto plazo.

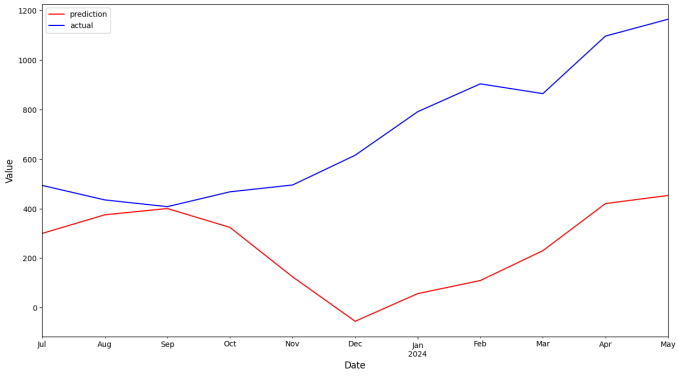

Fig. 16. Predicción del modelo ESN para NVIDIA.

Cabe resaltar que, en ambas pruebas, pero sobre todo en esta última de NVIDIA, se realizaron distintas ejecuciones y se quedó con el mejor modelo. Dada la naturaleza aleatoria de este modelo, los resultados varían ampliamente, obteniendo modelos bastante más deficientes que el actual.

XII. Gated Recurrent Units Network (GRU)

Es otro tipo de red RNN la cual se puede ver como una alternativa a ESN o una simplificación al modelo LSTM ya que su estructura se caracteriza por la existencia de las siguientes dos puertas:

- **Puerta de Reinicio**: determina qué tanta información de los estados ocultos anteriores se olvida.

$$r_t = \sigma_g \left(W_r * [h_{t-1}, x_t] \right)$$

- **Vector de activación de candidatos**: combina la información de los datos de entrada y del estado oculto anterior.

$$h_t \sim= \tanh(W_h * [r_t * h_{t-1}, x_t])$$

- **Puerta de Actualización**: determina cuanta información del vector de candidatos se incorpora al nuevo estado oculto.

$$z_t = \sigma_g \left(W_z * [h_{t-1}, x_t] \right)$$

Lo anterior implica que ya no existen las puertas de olvido, entrada y salida que si existen en LSTM. Además, la célula de memoria de LSTM es sustituida por el "vector de activación de candidatos". Lo anterior permite mejoras en el tiempo de ejecución sin perder la capacidad de generar pronósticos relativamente confiables.

El entrenamiento de la red se puede observar en el siguiente algoritmo:

Algorithm 6 GRU para Pronóstico de Series de Tiempo

Inicialización:
Inicializar pesos aleatoriamente: W_r, W_z, W_h
Inicializar estado oculto h_t y el vector de activación de candidatos c_t
Entrenamiento:
for cada época **do**
  for cada entrada x_t en la secuencia de entrenamiento **do**
    Puerta de Reinicio:
    $r_t = \sigma_g \left(W_r * [h_{t-1}, x_t] \right)$
    Puerta de Actualización:
    $z_t = \sigma_g \left(W_z * [h_{t-1}, x_t] \right)$
    Cálculo del vector de activación de candidatos :
    $h_t \sim= \tanh(W_h * [r_t * h_{t-1}, x_t])$
    Actualizar el Estado Oculto:
    $h_t = (1 - z_t) * h_{t-1} + z_t * h_t \sim$
  end for
end for
Evaluación del GRU:
Dividir y normalizar datos de prueba
Propagación hacia adelante y evaluar rendimiento
Predicción de nuevos datos:
Dividir y normalizar nuevos datos
Propagación hacia adelante para generar predicciones

Esta relación entre el modelo GRU con los modelos LSTM y RNN se puede apreciar visualmente en la siguiente imagen:

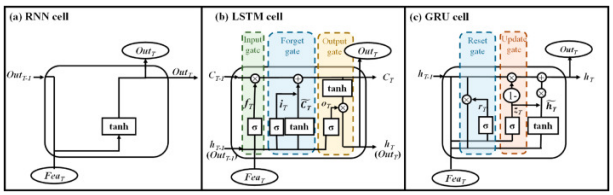

Fig. 17. Imagen comparativa del articulo de Li et al[2].

Análisis de resultados

Los pronósticos obtenidos para los datos “AirPassengers” se pueden observar en la figura 18.

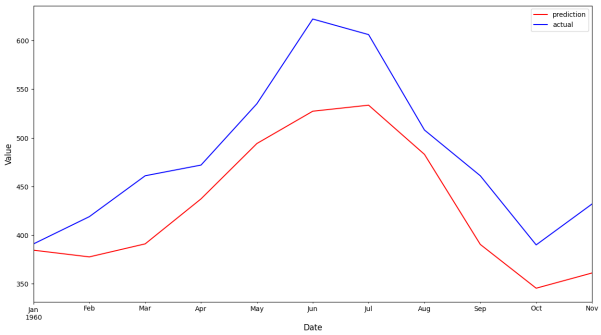

Fig. 18. Predicción del modelo GRU del número de vuelos para 1960.

Se puede notar que el modelo logra identificar los efectos de tendencia y estacionalidad de la serie de tiempo, siendo las fechas con mayor crecimiento o mayor decremento las que tienen una mayor diferencia respecto del valor original.

Si bien su valor MAPE de $10{,}63\,\%$ es mayor que el de LTSM, se puede considerar que es un modelo relativamente bueno pese a tener una estructura más sencilla que LTSM. Por otro lado, logra mejorar el pronóstico obtenido con ESN.

Respecto a los pronósticos de NVIDIA, se pueden apreciar en la figura 18.

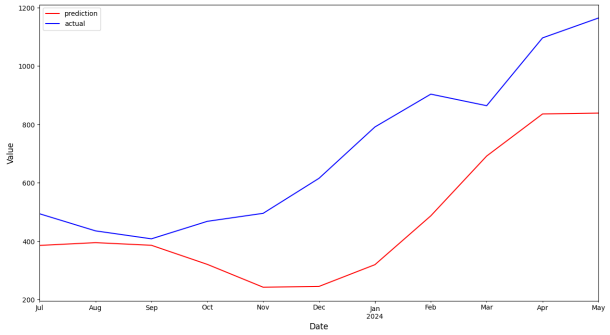

Fig. 19. Predicción del modelo GRU para NVIDIA.

Se obtuvo un valor MAPE de $32{,}46\,\%$ siendo el mejor valor obtenido con arquitecturas de redes neuronales, lo cual se puede notar visualmente ya que se el modelo logra identificar el comportamiento de la tendencia aunque con cierto desfase y no con la intensidad real.

Otra vez sobresale que el repentino incremento de los valores de NVIDIA es complicado de detectar para el modelo y por ello se observa ese desface en el comportamiento del pronóstico y del valor real.

XIII. Métodos de Ensamble.

Los métodos de ensamble combinan las predicciones de múltiples redes neuronales para mejorar la precisión y robustez de los pronósticos. Al agrupar las predicciones de diversos modelos se busca reducir la incertidumbre asociada a cada uno.

Media:

El método de ensamble con media es uno de los más simples y comunes. Calcula el promedio de los pronósticos de varios modelos, ponderado o no ponderado, para obtener el pronóstico final.

Fórmula de ensamble:

$$\widetilde{y}_{\text{Mean}} = \sum_{m=1}^{M} w_m y_{m_t}$$

Donde w_m son los pesos de cada modelo m de los M modelos totales. y_{m_t} es la predicción del modelo m para el periodo t.

Algorithm 7 Método de Ensamble con Media

Inicialización:
Definir M (número de modelos)
Inicializar lista vacía para los pronósticos combinados: *lista_combinada*
Cálculo del Ensamble:
for cada periodo t en $\{1, \ldots, \text{T}\}$ **do**
    $suma\_ponderada \leftarrow \sum_{m=1}^{M} w_m \cdot y_{m,t}$
    Agregar $suma\_ponderada$ a *lista_combinada*
end for
Retorno: *lista_combinada*

Mediana:

El método de ensamble con mediana utiliza la mediana de los pronósticos ordenados. La mediana se determina ordenando los pronósticos y_{m_t} de cada modelo de menor a mayor y seleccionando el valor medio si M es impar o el promedio de los dos valores medios si M es par.

Algorithm 8 Método de Ensamble con Mediana

Inicialización:
Definir M (número de modelos)
Inicializar lista vacía para los pronósticos combinados: *lista_combinada*
Cálculo del Ensamble:
for cada periodo t en $\{1, \ldots, \text{T}\}$ **do**
    Ordenar los pronósticos $y_{1,t}, \ldots, y_{M,t}$ en orden ascendente
    if M es impar **then**
        $mediana \leftarrow y_{\lceil M/2 \rceil, t}$
    else
        $mediana \leftarrow (y_{M/2,t} + y_{M/2+1,t})/2$
    end if
    Agregar $mediana$ a *lista_combinada*
end for
Retorno: *lista_combinada*

Moda:

El operador de ensamble con moda se basa en la estimación de la densidad de kernel para calcular la moda de los pronósticos (Kourentzes et al., 2014).

Fórmula de ensamble:

$$\hat{f}_{th}(x) = (Mh)^{-1} \sum_{m=1}^{M} K\left(\frac{x - y_{mt}}{h}\right)$$

Donde $K(\cdot)$ es una función kernel que satisface $\int K(x)\,dx = 1$. Muchas veces se usa el Kernel Gaussiano:

$$\phi_h(x) = \frac{1}{\sqrt{2\pi h}} e^{-\frac{x^2}{2h^2}}$$

Finalmente, se puede aproximar el ancho de banda h bajo la regla de Silverman:

$$h = \left(\frac{4\hat{\sigma}^5}{3M}\right)^{\frac{1}{5}}$$

Algorithm 9 Método de Ensamble con Moda

Inicialización:
Calcular h usando $\hat{\sigma}$ de los pronósticos $y_{m,t}$
Inicializar lista vacía para los pronósticos combinados: *lista_combinada*
Cálculo del Ensamble:
for cada periodo t en {1, ..., T} **do**
  $densidad\_max \leftarrow 0$
  for cada x en rango de valores posibles **do**
    $densidad \leftarrow (Mh)^{-1} \sum_{m=1}^{M} K\left(\frac{x - y_{m,t}}{h}\right)$
    if $densidad > densidad\_max$ **then**
      $densidad\_max \leftarrow densidad$
      $x\_moda \leftarrow x$
    end if
  end for
  Agregar x_moda a *lista_combinada*
end for
Retorno: *lista_combinada*

Análisis de resultados

Para "AirPassengers" los pronósticos para 1960 usando distintos operadores de ensamble se pueden observar en la figura 20, 21 y 22 usando los modelos GRU, MLP, LTSM y ENS. El ensamble con mediana obtuvo una puntuación MAPE de 4.9 %, el hecho con la media resultó con una puntuación de 5.8 % mientras que el de moda obtuvo un 6.2 %. Estos resultados evidencian que, aunque los ensambles pueden mejorar un modelo, su rendimiento está directamente relacionado a los modelos propuestos. Eliminar los peores modelos o los que no aportan algo significativo con respecto a otros es un paso clave.

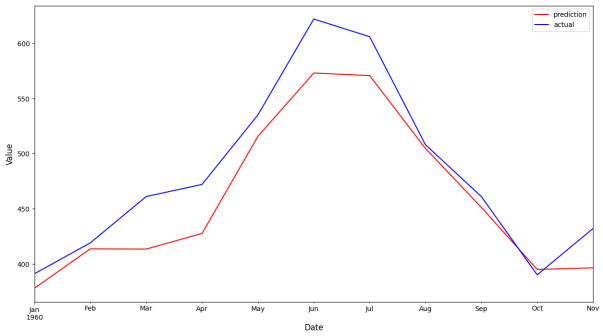

Fig. 20. Pronóstico con ensamble de media.

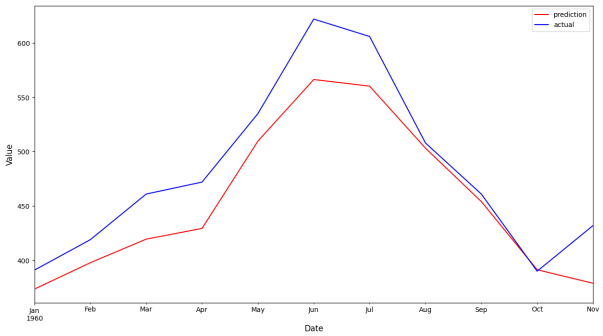

Fig. 21. Pronóstico con ensamble de mediana.

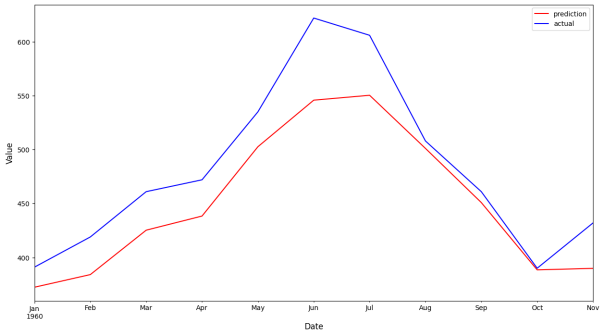

Fig. 22. Pronóstico con ensamble de moda.

En cuanto a NVIDIA, los modelos con ensamble GRU, MLP y LTSM obtuvieron puntuaciones MAPE de 46.5 % para la media, 45.8 % para la mediana y la moda. Aunque no mejoran en cuanto a las puntuaciones, si observamos en las figuras 23, 24 y 25, una mejora en cuanto a la "forma" o tendencia.

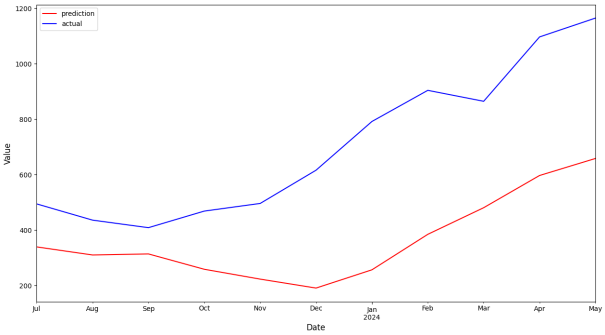

Fig. 23. Predicción con ensamble de media para NVIDIA.

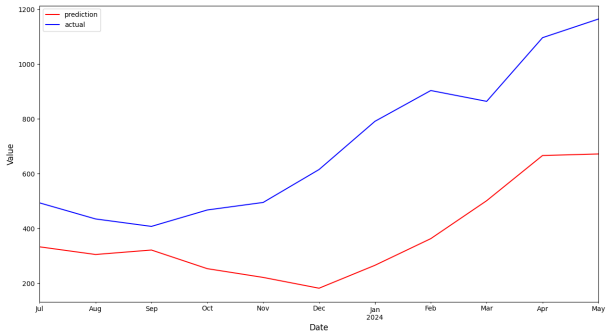

Fig. 24. Predicción con ensamble de mediana para NVIDIA.

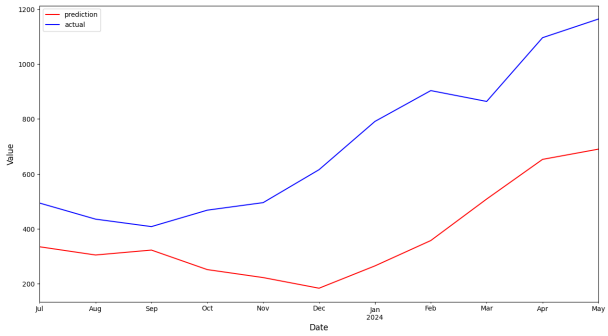

Fig. 25. Predicción con ensamble de moda para NVIDIA.

Otro detalle que sobresale de usar modelos de ensamble es que se debe tener en cuenta el tiempo de ejecución ya que así como hereda ciertas características de los pronósticos de los modelos que lo conforma también hereda tiempos de ejecución.

XIV. Conclusiones

Al igual que con otra clase de modelos, el tratamiento de los datos sigue siendo una parte esencial para poder ajustar modelos de pronóstico a series de tiempo ya que si no se realizan las transformaciones o generación de variables necesarias estos modelos pueden no detectar características temporales de las series de tiempo que permitan mejorar su pronóstico.

En cuanto al desempeño de los modelos, se observó que SVR fue el peor modelo evaluado pero con series de tiempo como la de "AirPassengers" logra capturar relativamente bien características como la tendencia y la ciclicidad de los datos. No obstante, para series de tiempo que pueden sufrir grandes cambios repentinos como la serie de tiempo de NVIDIA no logra detectar dichas características. Por su parte, los modelos que son distintas arquitecturas de Deep Learning mostraron tener un mejor desempeño para detectar las características de las series de tiempo con un comportamiento tan marcado como la de "AirPassengers". No obstante, respecto a las series de tiempo como la de NVIDIA se comparte el problema de no poder replicar adecuadamente los cambios inesperados y drásticos. Estos detalles se heredaron al modelo de ensamble que se hizo con base a las distintas arquitecturas de redes neuronales que se implementaron en el presente trabajo.

El problema de no poder replicar adecuadamente los cambios inesperados en parte se explica porque sólo se implementaron versiones univariadas de los modelos. Por ello, se considera que los modelos pueden mejorar su capacidad para poder pronosticar las series de tiempo si se añaden variables que permitan anticipar los cambios repentinos, ya sea de tendencia o de la magnitud de esta, ya que algunas series de tiempo son más susceptibles a sufrir cambios de este tipo que otras, tal como es el caso de la serie de tiempo de NVIDIA.

Por otro lado, aún con las series de tiempo relativamente pequeñas que se emplearon en las pruebas se pudieron detectar los beneficios que en el tiempo de ejecución presentan algunas arquitecturas de redes neuronales respecto de otras. Otro beneficio es que los modelos de ML contemplados no requieren de un supuesto fuerte como los modelos tradicionales de pronóstico de series de tiempo. Si bien se pueden beneficiar de los supuestos de estos modelos tradicionales, no son un supuesto con el que se hayan construido los modelos de ML lo cual se convierte en una ventaja por no depender de dichos supuestos.

Finalmente, se considera que al momento de escoger el modelo que se quiera ajustar a los datos del fenómeno de interés se debe tener consciencia de las restricciones que puedan existir, ya que restricciones de tiempo o de recursos pueden evitar que se pueda usar algún modelo en particular aunque este sea el mejor evaluado en las métricas de desempeño.

Bibliografía

[1] A. Smola y B. Schölkopf, "A tutorial on support vector regression," *Statistics and Computing*, vol. 14, págs. 199-222, ago. de 2004. DOI: 10.1023/B%3ASTCO.0000035301.49549.88.

[2] X. Li, X. Ma, F. Xiao, C. Xiao, F. Wang y S. Zhang, "Time-series production forecasting method based on the integration of Bidirectional Gated Recurrent Unit (Bi-GRU) network and Sparrow Search Algorithm (SSA)," *Journal of Petroleum Science and Engineering*, vol. 208, pág. 109 309, ene. de 2022, ISSN: 09204105. DOI: 10.1016/j.petrol.2021.109309.

[3] O. B. Sezer, M. U. Gudelek y A. M. Ozbayoglu, "Financial Time Series Forecasting with Deep Learning : A Systematic Literature Review: 2005-2019," nov. de 2019.

[4] P. Lara-Benítez, M. Carranza-García y J. C. Riquelme, "An Experimental Review on Deep Learning Architectures for Time Series Forecasting," *CoRR*, vol. abs/2103.12057, 2021. arXiv: 2103.12057. dirección: https://arxiv.org/abs/2103.12057.

[5] F. Lazzeri, *Machine Learning for Time Series Forecasting with Python®*. Wiley, dic. de 2020, ISBN: 9781119682363. DOI: 10.1002/9781119682394.

[6] A. Dhavlle y S. M. Pudukotai Dinakarrao, "A Comprehensive Review of ML-based Time-Series and Signal Processing Techniques and their Hardware Implementations," en *2020 11th International Green and Sustai-*

nable Computing Workshops (IGSC), 2020, págs. 1-8. DOI: 10.1109/IGSC51522.2020.9291173.

[7] M. Lukoševičius, "A Practical Guide to Applying Echo State Networks," en *Neural Networks: Tricks of the Trade: Second Edition*, G. Montavon, G. B. Orr y K.-R. Müller, eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, págs. 659-686, ISBN: 978-3-642-35289-8. DOI: 10.1007/978-3-642-35289-8_36. dirección: https://doi.org/10.1007/978-3-642-35289-8_36.

[8] N. Kourentzes, D. K. Barrow y S. F. Crone, "Neural network ensemble operators for time series forecasting," *Expert Systems with Applications*, vol. 41, págs. 4235-4244, 9 jul. de 2014, ISSN: 09574174. DOI: 10.1016/j.eswa.2013.12.011.

[9] G. T. Wilson, "Time Series Analysis: Forecasting and Control, 5th Edition, by George E. P. Box, Gwilym M. Jenkins, Gregory C. Reinsel and Greta M. Ljung, 2015. Published by John Wiley and Sons Inc., Hoboken, New Jersey, pp. 712. ISBN: 978-1-118-67502-1," *Journal of Time Series Analysis*, vol. 37, págs. 709-711, 5 sep. de 2016, ISSN: 0143-9782. DOI: 10.1111/jtsa.12194.

[10] D. Montgomery, C. Jennings y M. Kulahci, *Introduction to Time Series Analysis and Forecasting* (Wiley Series in Probability and Statistics). Wiley, 2008, ISBN: 9780471653974. dirección: https://books.google.com.mx/books?id=IY6OQgAACAAJ.