Universidad Nacional de Córdoba
Facultad de Matemática, Astronomía, Física y Computación

Modelos de puntuación basados en aprendizaje automático para calcular el *scoring* de empresas

TRABAJO ESPECIAL

Licenciatura en Física

Autor:
Martín G. Trucco

Directores:
Mgter. David A. Giuliodori
Dr. Francisco A. Tamarit

Mayo 2024

Agradecimientos

A mi familia que me acompañó y apoyó en este largo camino, sin interponerse en ninguna de las decisiones que tomé.

A mi papá, Sergio, y su hermana, Noris, que las circunstancias de la vida llevaron a que hoy no estén acá. ¡Lo logré!

A Pancho y David que aceptaron el desafío y la responsabilidad de guiarme en esta última etapa de la carrera.

A Juan, Gabriela y Sergio por ser parte del tribunal.

A mis amigos de toda la vida, a los que muchas veces tuve que renunciar porque no tenía tiempo (sí, en esta facultad todos los días son lunes). Y a los que me dio la facultad porque no hubiese continuado sin su sostén.

A la Facultad por haberme brindado una educación pública y de calidad y a sus profesores que, en mayor o menor medida, todos contribuyeron a mi formación.

Gracias a todos, un abrazo.

A mi viejo

Resumen

El *scoring* bancario es una herramienta esencial para facilitar la toma de decisiones en instituciones financieras, ya que permite determinar si es factible hacer entrega del crédito o financiamiento solicitado por parte de individuos o personas jurídicas. En consecuencia, este trabajo tiene como objetivo predecir la capacidad de impago de empresas nacionales, analizando bases de datos públicas y anonimizadas correspondientes a 51.141 firmas, con datos recopilados durante 49 meses consecutivos. Debido a la importancia de la dimensión temporal en este tipo de problemas, se adopta un enfoque basado en series temporales y se aplican tres arquitecturas de aprendizaje automático: regresión logística, Random Forest de clasificación y redes neuronales recurrentes LSTM.

La evaluación del rendimiento de los modelos se realiza mediante diversas métricas e indicadores como la sensibilidad, exactitud o curvas ROC, que proporcionan información valiosa para identificar el método con mejor desempeño. Los resultados reflejan una alta efectividad en general, siendo el clasificador Random Forest predominante, lo que destaca la relevancia actual del aprendizaje automático en el ámbito económico.

***D*escriptores:** inteligencia artificial, redes neuronales, series temporales, análisis de datos, scoring bancario, econometría

Abstract

Bank scoring is an essential tool to facilitate the decision-making in financial institutions, as it allows determining whether it is feasible to grant the requested credit or loan to individuals or legal entities. Consequently, this work aims to predict the default capacity of national companies, analyzing public and anonymized databases corresponding to 51.141 firms, with data collected over 49 consecutive months. Due to the importance of the time dimension in this type of problems, a time series-based approach is adopted, and three machine learning architectures are applied: logistic regression, Random Forest classifier and LSTM recurrent neural networks.

The performance evaluation of the models is carried out using various metrics and indicators such as recall, accuracy or ROC curves, which provide valuable information to identify the best method. The results reflect a high effectiveness overall, with the Random Forest classifier being predominant, highlighting the current relevance of machine learning in the economic domain.

Keywords: machine learning, neural networks, time series, data analysis, credit scoring, credit risk

Contenido

1 Introducción

La previsión y clasificación de series temporales (especialmente en el ámbito financiero) han sido abordadas históricamente mediante métodos tradicionales (lineales y no lineales) [1]. No obstante, estos métodos suelen tener deficiencias cuando se analizan series de tiempo complejas, como en el caso del precio o rendimiento futuro de un activo en el mercado financiero [2]. Frente a estas limitaciones y desafíos, durante los últimos años se ha hecho cada vez más extensivo el uso de técnicas de aprendizaje automático.

El uso de técnicas de aprendizaje automático a la evaluación del riesgo crediticio de una empresa es un claro e importante ejemplo de la aplicación de inteligencia artificial a problemas de naturaleza económica, los cuales, a su vez, tienen un fuerte impacto en las sociedades contemporáneas. Esta valoración es esencial para tomar decisiones fundamentadas y basadas en datos a la hora de otorgar financiamiento bancario, tanto público como privado. Sin embargo, debido a la existencia de múltiples factores que pueden influir en la capacidad de la empresa para saldar la deuda contraída y dado que muchos de estos no tienen una expresión clara a la hora de modelar el riesgo, puede ser altamente complicado llevar a cabo esta tarea sin la ayuda de métodos propios de la ciencia de datos. Es en este contexto donde se presentan los modelos de puntuación (*scoring*) como una solución prometedora, ya que permiten procesar grandes volúmenes de datos y establecer patrones de comportamiento útiles para hacer predicciones precisas sobre el riesgo de crédito de una empresa.

En general, se dispone de series temporales con un gran número de variables ligadas al desempeño de la empresa durante largos periodos de tiempo. Algunas variables son, por ejemplo, el nivel de deuda, las entidades financieras a las que se adeuda, la ubicación geográfica de la empresa, la rama de la actividad, la cantidad de empleados o el salario promedio de estos. El nivel de deuda es un factor crítico a tener en cuenta, pues indica la cantidad de dinero que la empresa ha pedido prestado y que debe ser devuelto con intereses. Las entidades financieras a las que se debe es otro factor determinante, dado que cada una tiene sus propios criterios de evaluación. La ubicación geográfica es importante ya que el nivel de actividad,

de consumo y las regulaciones correspondientes impuestas por los estados locales cambian de región a región. La rama de la actividad puede ser relevante puesto que algunos sectores suelen ser más volátiles y otros más robustos, algunos dependen principalmente de condiciones políticas y económicas regionales y otros están sujetos a las reglas del mercado internacional.

En el presente trabajo se implementan técnicas de aprendizaje automático para generar modelos capaces de evaluar y predecir la capacidad de pago de empresas nacionales, siendo clasificadas según su situación de deuda como **0** ("no morosas") o **1** ("morosas").

Utilizando un conjunto de datos públicos y anonimizados extraídos de fuentes administrativas, se construye un panel único de datos para el periodo de tiempo comprendido entre abril de 2018 y julio de 2022. Se rezagan las variables de la base y posteriormente se aplican tres tipos de métodos: regresión logística (LogReg), bosques aleatorios (RanFor) y redes neuronales recurrentes de tipo Long Short-Term Memory (LSTM). Así, el propósito de este trabajo consiste en calcular la situación de deuda de cada empresa para el mes siguiente al cual se está analizando, en función de las variables rezagadas correspondientes al mes actual y a los 11 meses anteriores, y definir finalmente si la entidad solicitante será capaz de devolver el crédito en las formas pactadas o si, por el contrario, el banco rechaza dicha solicitud.

El documento se organiza de la siguiente manera: en el capítulo 2 se abordan conceptos básicos relacionados al aprendizaje automático y las redes neuronales, se especifican los algoritmos empleados y criterios para evaluar su rendimiento, se comenta sobre los sistemas de puntuación y el acceso al financiamiento de las empresas y se describe brevemente el problema a resolver. El capítulo 3 detalla el procedimiento de preparación de la base de datos y la aplicación de los modelos. En el capítulo 4 se presentan y discuten los resultados obtenidos. Por último, en el capítulo 5 se desarrollan las conclusiones y se sugieren posibles mejoras y/o recomendaciones para estudios futuros.

2 Marco Teórico

2.1. Aprendizaje Automático

El aprendizaje automático es un campo dentro de la inteligencia artificial centrado en desarrollar algoritmos con la capacidad de aprender a resolver problemas y mejorar su rendimiento automáticamente, sin la necesidad de ser programarlos explícitamente para ello. Estas habilidades se logran mediante la obtención de conocimientos, los cuales se adquieren a partir de la experiencia (el entrenamiento).

Se diferencian tres categorías principales, según la metodología de entrenamiento:

* Aprendizaje supervisado: el algoritmo se entrena usando un conjunto de datos previamente etiquetados, es decir, datos con pares de entrada y salida. Cuando el sistema realiza una predicción, la misma se compara con la salida correcta (etiqueta) y a través de la comparación el modelo aprende.

* Aprendizaje no supervisado: a diferencia del caso anterior, solo se cuenta con los datos de entrada, por lo que no es posible el aprendizaje por comparación. El algoritmo, en este caso, debe extraer patrones y detectar las características relevantes por sí mismo.

* Aprendizaje por refuerzos: el sistema interactúa con el entorno y la información obtenida es si se equivoca o no, recibiendo beneficios o sanciones por sus acciones. Estos estímulos logran que el algoritmo aprenda al crear una estrategia que maximice las recompensas.

Para el caso particular del aprendizaje supervisado, la idea es predecir un resultado en base a los datos existentes. Esto involucra dos tipos de tareas posibles: regresión y clasificación.

En los problemas de regresión, el objetivo es asignar un valor numérico continuo a cada nueva instancia de entrada. Por otro lado, en los problemas de clasificación la predicción es una categoría discreta o clase.

Esta tesina se focaliza en esta última tarea, ya que se dispone de un conjunto completo de entrada y salida de datos y además se requiere que el resultado sea binario, es decir, dos clases posibles.

2.2. Modelos "lineales"

Los **modelos lineales** son métodos estadísticos utilizados para predecir el resultados de la variable dependiente (respuesta) en función de una o más variables independientes (predictoras), por lo que reciben el nombre de regresión lineal simple y múltiple respectivamente. La idea subyacente a estos modelos es asumir una relación lineal entre las variables, de manera que la respuesta se expresa como combinación lineal de las predictoras.

Matemáticamente su expresión es

$$\begin{aligned} y &= w_1 x_1 + w_2 x_2 + \ldots + w_n x_n + b \\ &= \left(\sum_{i=1}^{n} w_i x_i \right) + b = \mathbf{w} \cdot \mathbf{x} + b \end{aligned} \tag{2-1}$$

donde y es la variable dependiente, x_i son las variables independientes o características de entrada, w_i son coeficientes que representan la relación entre las variables (pesos) y b es el término de intersección o sesgo (*bias*).

El objetivo principal es encontrar el hiperplano que minimiza la diferencia entre el valor real de y con el predicho por la ecuación (2-1), lo que hace a estos métodos útiles en problemas de regresión pero ineficaces para clasificación.

Una extensión a estos son los **modelos lineales generalizados (GLM)**, que generalizan la regresión lineal al establecer una función de vínculo (*link*) entre las variables dependiente e independientes. Existen diferentes funciones de vínculo que derivan en distintos algoritmos similares, pero a los efectos de este trabajo es necesario introducir uno de ellos: la regresión logística binaria.

2.2.1. Regresión Logística Binaria (LogReg)

Cuando a tareas de clasificación se refiere, existen muchas situaciones en la que no conviene etiquetar directamente a la instancia de salida en categorías sino más bien asignarle una probabilidad de pertenecer a alguna de ellas. Esta condición probabilística no es aplicable a la regresión lineal pues dicho modelo admite valores de salida desde $-\infty$ a ∞, es decir, fuera del intervalo [0,1]. Para solucionar este problema se mantiene la linealidad en las predictoras pero, en vez de predecir una probabilidad, se predice el logaritmo del cociente

de probabilidades[1]. Este es, para el caso de dos clases (0 y 1 por ejemplo), el modelo de **regresión logística binaria**.

Su ecuación se traduce en

$$\ln\left(\frac{p(y=1|\mathbf{x})}{1-p(y=1|\mathbf{x})}\right) = \mathbf{w}\cdot\mathbf{x}+b \quad \Rightarrow \quad \frac{p(y=1|\mathbf{x})}{1-p(y=1|\mathbf{x})} = e^{\mathbf{w}\cdot\mathbf{x}+b} \tag{2-2}$$

con $p(y=1|\mathbf{x})$ la probabilidad de pertenecer a la clase 1 y $1-p(y=1|\mathbf{x}) = p(y=0|\mathbf{x})$ la probabilidad de pertenecer a la clase 0. Además, el cociente de probabilidades se denomina odds, log-odds se conoce como función logit y $e^{\mathbf{w}\cdot\mathbf{x}+b}$ es la función de vínculo.

Aplicando un poco de álgebra se puede expresar la probabilidad de pertenencia a cada clase en función de las variables exógenas

$$p(y=1|\mathbf{x}) = \frac{1}{1+e^{-(\mathbf{w}\cdot\mathbf{x}+b)}} \quad ; \quad p(y=0|\mathbf{x}) = \frac{e^{-(\mathbf{w}\cdot\mathbf{x}+b)}}{1+e^{-(\mathbf{w}\cdot\mathbf{x}+b)}} \tag{2-3}$$

donde $(1+e^{-x})^{-1} \equiv \sigma(x)$ es la llamada sigmoide o función logística, que da nombre al modelo.

A la hora de identificar a qué categoría pertenece una dada observación, el umbral de decisión tiene un rol preponderante y se sigue la regla

$$\text{clase}(x) = \begin{cases} 1 & \text{si } p(y=1|\mathbf{x}) > \text{umbral} \\ 0 & \text{en otro caso} \end{cases}$$

El umbral predeterminado es 0,5 aunque puede ser ajustado según criterios como la curva PRC o la curva ROC. En la sección 2.6 se describe esta última ya que es la empleada para elegir el valor óptimo.

2.3. Modelos basados en árboles

Una alternativa para modelar la respuesta en función de las variables de entrada pero con un enfoque radicalmente diferente es el **árbol de decisión (DTs)**.

Ampliamente utilizado en problemas tanto de clasificación como de regresión, es un algoritmo jerárquico de toma de decisiones con estructura de árbol, con una raíz como elemento principal que se divide binariamente en nodos internos o ramas y hojas (ver figura **2-1**).

La raíz hace referencia a la variable más relevante y se elige a partir de algún criterio estadístico como la ganancia de información o la impureza de Gini, que mide la importancia de las características para separar los datos de manera efectiva. Luego se formula una

[1]Una primera propuesta podría ser el ratio de probabilidades, sin embargo, esta opción no es adecuada porque la imagen de las funciones a cada lado de la igualdad difiere.

pregunta sobre un atributo específico representado por las ramas, dividiendo al conjunto en subconjuntos más pequeños. La elección de dichos atributos se realiza, como en el caso de la raíz, mediante la medida estadística definida. Este proceso de selección se repite recursivamente, tal que cada nodo interno conduce a nuevas ramificaciones basadas en la respuesta a la pregunta planteada, hasta llegar a las hojas del árbol. Estas últimas son las etiquetas y constituyen las predicciones finales del modelo.

A pesar de que los árboles de decisión son herramientas muy poderosas, la mayor desventaja es su poca sensibilidad ante cambios en el conjunto de datos inicial (por ejemplo al agregar información extra). Esta carencia es solventada por los **Random Forests**.

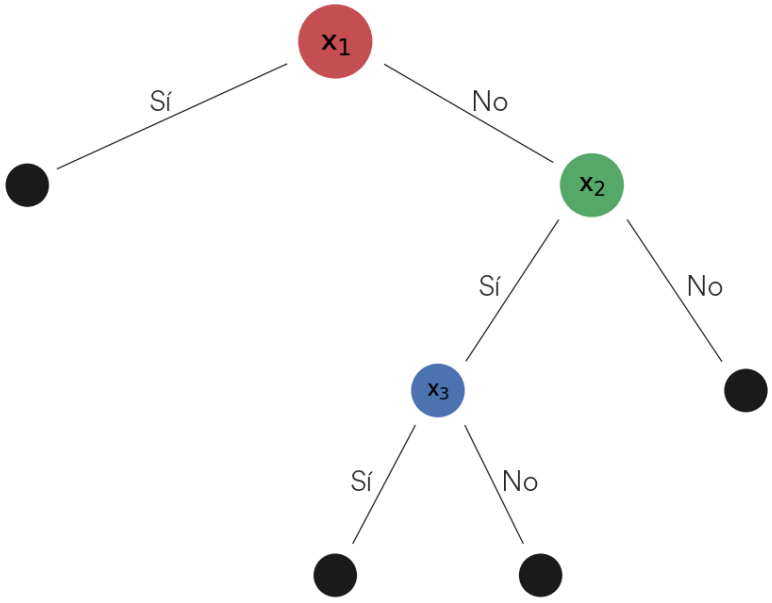
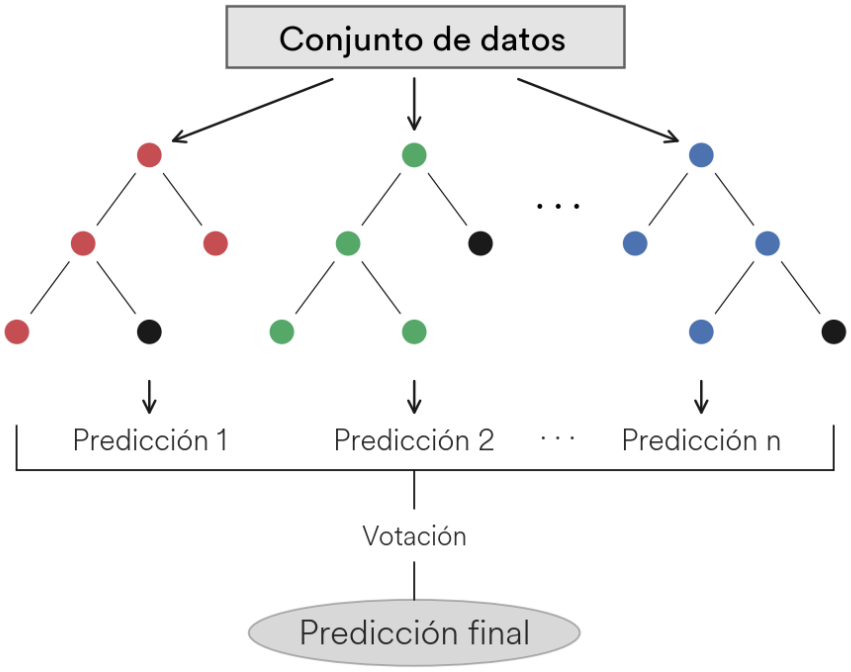


Figura 2-1.: Esquema de un árbol de decisión. x_1 es el nodo raíz y contiene la característica más relevante. Los nodos internos x_2 y x_3 son las ramificaciones del árbol. Los círculos negros son las hojas y representan las predicciones del modelo.

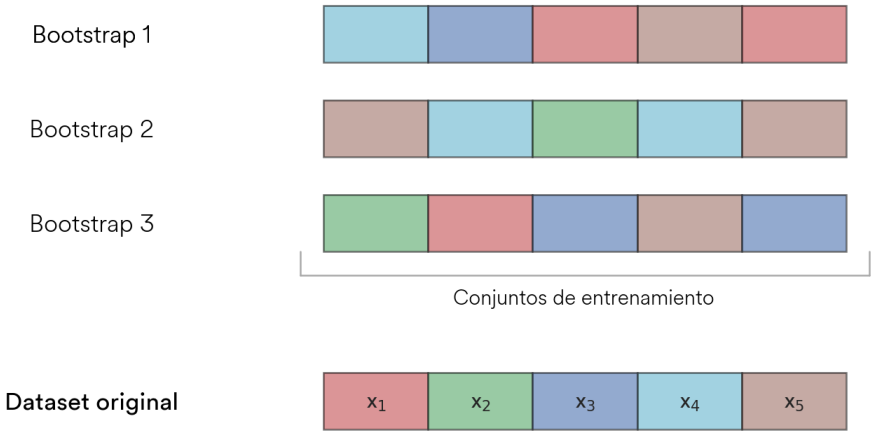
2.3.1. Random Forest Clasificador (RanFor)

Un bosque aleatorio de clasificación[2] [3], representado por $\{h(\mathbf{x}, \Theta_k),\ k = 1, 2, ..., n\}$, es una colección de n árboles de decisión individuales $h(\mathbf{x}, \Theta_k)$, donde $\mathbf{x}$ es el vector de características de entrada y $\{\Theta_k\}$ es un conjunto de vectores aleatorios independientes e idénticamente distribuidos (i.i.d) relativos al k-ésimo árbol (ver figura **2-2**a). Estos Θ_k dependen de dos fuentes de aleatoriedad:

[2]Como señala [3], estas ideas también pueden ser aplicadas a problemas de regresión.



(a) Cuando se introduce una nueva observación del conjunto de testeo, los DTs realizan una predicción (en negro). Luego se somete a votación y la clase más frecuente se cataloga como la predicción final.



(b) Se generan nuevos conjuntos de entrenamiento mediante la extracción aleatoria de instancias del dataset original. El reemplazo asegura que los conjuntos adicionales tengan la misma dimensión que el principal, diversificando las muestras.

Figura 2-2.: Ilustración de (a) Random Forest y (b) técnica de bootstrapping.

1. Muestreo con reemplazo (*bootstrapping*): para la construcción de cada árbol se crea un nuevo conjunto de datos (muestra bootstrap) tomando aleatoriamente observaciones del conjunto original $\mathbf{x}$, permitiendo que cada instancia sea devuelta y pueda seleccionarse nuevamente (reemplazo). Este proceso implica que ciertas filas del conjunto se repetirán, mientras que otras podrían no aparecer en la muestra (ver figura **2-2**b).

2. Selección aleatoria de características (*random feature selection*): una vez realizado el bootstrapping, se escoge de forma aleatoria un subconjunto de variables que se utilizarán para la división en cada nodo de decisión del árbol.

Luego, cada árbol emite un voto y la predicción final se realiza eligiendo la clase más popular de $\mathbf{x}$, ya sea mediante *hard voting* (la clase con más votos es la ganadora) o *soft voting* (se promedian las probabilidades de cada clase y gana la que obtiene el mayor resultado).

La ventaja de los bosques aleatorios, como ya se mencionó previamente, reside en su capacidad de generalización. Por un lado, el muestreo con reemplazo asegura la diversidad de árboles ya que cada uno se construye con datos distintos. Por otro, la selección aleatoria de características reduce la correlación entre ellos pues, si las variables usadas fueran idénticas, la mayoría tendrían nodos de decisión similares y clasificarían de manera análoga.

Se podría pensar que la aleatoriedad de esta última pueda generar, en ciertas circunstancias, muchos árboles con una tasa de clasificación baja. Para evitar esto, la cantidad mínima de árboles en un Random Forest suele ser del orden de 100. De esta manera, al promediar los resultados, se suaviza el efecto, al mismo tiempo que se reduce la varianza del modelo.

2.4. Redes Neuronales Artificiales

Las redes neuronales son una de las herramientas más poderosas y extendidas actualmente en el área del aprendizaje automático por su amplio campo de aplicaciones y eficacia. La capacidad de manipular grandes volúmenes de datos, reconocer patrones complejos y tomar decisiones las hace un elemento decisivo en problemas hasta el momento irresolubles. Tomando los procesos del cerebro humano como inspiración, las redes neuronales no solo son una innovación tecnológica, sino también un cambio en el paradigma sobre cómo abordar problemáticas en diferentes ámbitos de la sociedad.

2.4.1. Neurona artificial

Las redes neuronales tratan de imitar, parcialmente, el funcionamiento de las neuronas biológicas, centrándose en modelar las componentes esenciales para almacenar y transmitir la información entre estas unidades. De forma general, una neurona natural es estimulada por múltiples señales de entrada proveniente de otras neuronas o de células receptoras de

estímulos. Estas señales son ponderadas al viajar de una neurona a otra a través de las sinapsis, puntos de conexión que conforman una red compleja entre las neuronas. Si como resultante de todos los estímulos que una neurona recibe de sus vecinas pre-sinápticas (aquellas que le envían señales) se produce una alteración en el potencial de membrana (la diferencia de potencial entre el interior y el exterior de la neurona) que supera un cierto umbral, la neurona relajará a su estado de equilibro emitiendo una señal electroquímica que viajará a través de su axón para alcanzar muchas otras neuronas post-sinápticas o células receptoras.

Para modelar una red neuronal se utilizan neuronas artificiales de la forma más simple posible, con el fin de optimizar la eficiencia computacional al integrarlas en grandes conglomerados o redes neuronales.

Matemáticamente, este proceso se puede describir como

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right) \tag{2-4}$$

donde y es la salida de la red, f es la función de activación, w son los pesos asociados a las sinapsis, x_i son las entradas de la red y b el umbral de activación o sesgo (bias).

Esta idea, propuesta originalmente por Warren McCulloch y Walter Pitt [4], impulsó el desarrollo de la primera red neuronal programada: el Perceptrón Mark I, diseñado por Frank Rosenblatt [5] (ver figura **2-3**a). En su publicación, Rosenblatt acuña el término Perceptrón para referirse a la neurona de McCulloch-Pitts, que hoy en día es más conocido como **perceptrón simple**.

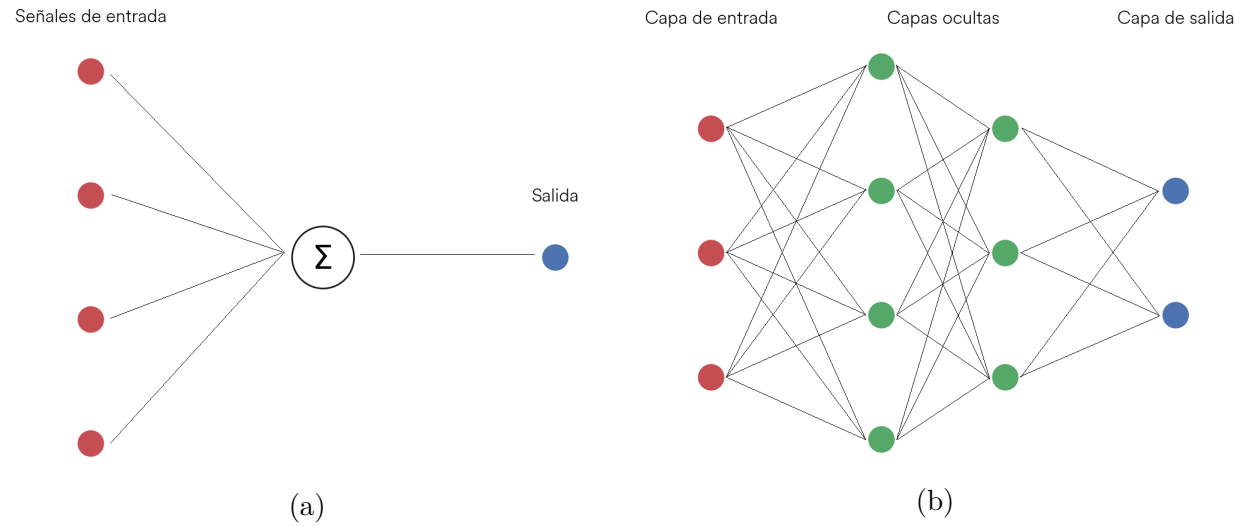


Figura 2-3.: Esquema de un Perceptrón (a) simple y (b) multicapa (red feedforward).

La incapacidad de este método para resolver problemas no separables linealmente (entre otras limitaciones [6]) condujeron al surgimiento del **perceptrón multicapa (MLP)** o **red**

feedforward (FFNN), una extensión del perceptrón simple donde la información fluye hacia adelante a través de capas (ver figura **2-3**b).

Estas capas se clasifican en tres tipos:

- Capa de entrada: consiste en nodos que reciben las características de entrada, las cuales representan el estímulo externo.
- Capas ocultas (una o varias): los nodos de estas capas se encargan de procesar la información recibida en la capa de entrada mediante los pesos, sesgos y funciones de activación.
- Capa de salida: la unidad o unidades de salida presentan las predicciones o inferencias finales.

2.4.1.1. Función de activación

La función de activación mide la respuesta de las neuronas artificiales, actuando como un interruptor que decide si se emite (o no) la señal de salida. Si bien existen funciones de tipo lineales, en el aprendizaje automático se suele optar por funciones no lineales, cuya salida usualmente está confinada en un intervalo finito (por ejemplo [0,1] o [-1,1]) y que permiten controlar e interpretar mejor los resultados del problema estudiado. Además, es fundamental que satisfaga las condiciones de continuidad y diferenciabilidad para poder aplicar el algoritmo de *backpropagation*, el mecanismo que posibilita al modelo aprender (en la sección 2.4.1.3 se detalla el mismo).

Algunos ejemplos típicos de funciones no lineales son (ver figura **2-4**):

- Sigmoide o función logística (logit): es la misma a la utilizada en 2.2.1. El rango de salida es [0,1] y se expresa matemáticamente como

$$f_{\text{logit}}(x) = \frac{1}{1+e^{-x}} \equiv \sigma(x) \tag{2-5}$$

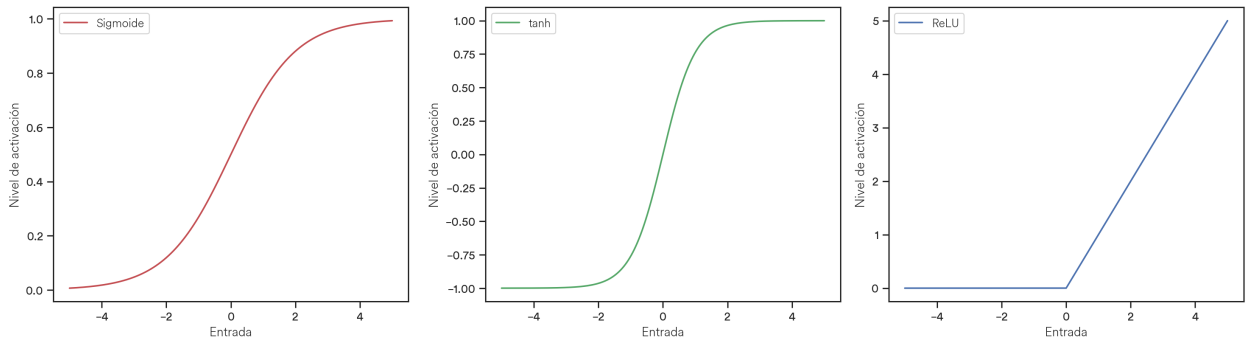
- Tangente hiperbólica: la salida está contenida en [-1,1] y está descrita por

$$f_{\tanh}(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \equiv \tanh(x) \tag{2-6}$$

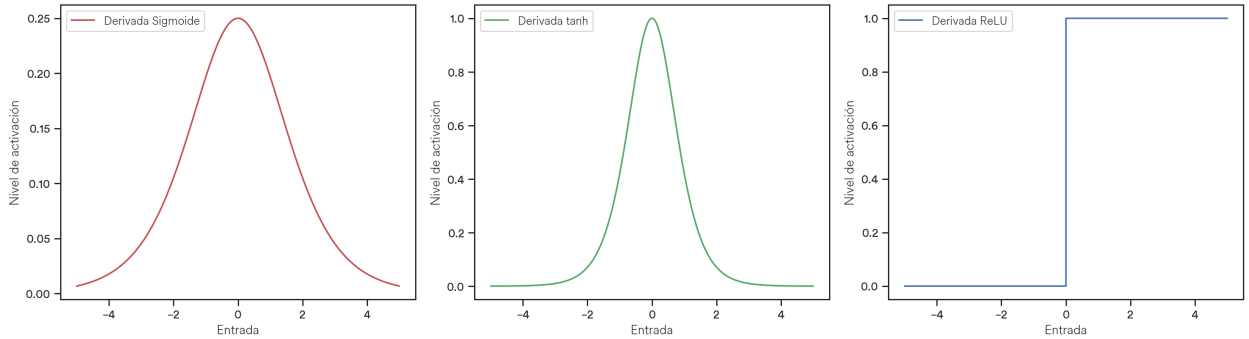
- ReLU (*Rectified Linear Units*): pese a que su salida no es acotada, su derivada se computa rápido, a diferencia de las funciones anteriores, lo que es crucial en el aprendizaje profundo. Además, evita un fenómeno muy perjudicial denominado desvanecimiento/explosión del gradiente, el cual se volverá a mencionar más adelante aunque está fuera del alcance de esta tesis. La ReLU se define como

$$f_{\text{ReLU}}(x) = \text{máx}(0,x) = \begin{cases} x & \text{si } x \geq 0 \\ 0 & \text{si } x < 0 \end{cases} \tag{2-7}$$

Aunque existen muchas otras variantes, para este trabajo se tienen en cuenta las dos primeras opciones, ya que los modelos aplicados las incorporan como parte integral de su funcionamiento.



(a) Izquierda: sigmoide. Centro: tangente hiperbólica. Derecha: ReLU.



(b) Izquierda: sigmoide. Centro: tangente hiperbólica. Derecha: ReLU.

Figura 2-4.: (a) Funciones de activación. (b) Derivadas de las funciones de activación.

Como se mencionó con anterioridad, el aprendizaje de una red neuronal se produce durante su entrenamiento. Para describir tal proceso, un paso previo y necesario es introducir la noción de pérdida o función de costo.

2.4.1.2. Función de costo

Para verificar si la red está aprendiendo se debe comparar el valor de la predicción con el valor real. Esa comparación se efectúa a través de la evaluación de la función error, también llamada función de costo, función de pérdida o función transferencia. Esta función cuantifica, a la salida de la red, el error entre dicha salida y el resultado correcto (etiqueta). Si la función de costo es cero, entonces el modelo reproduce exactamente la instancia verdadera, que es el resultado deseado.

En este trabajo se emplean dos tipos de funciones error:

- Error cuadrático medio (*MSE*): se define como el promedio (sobre n muestras) de las diferencias al cuadrado entre las predicciones $\hat{y}_i$ hechas por la red y las etiquetas y_i, lo que permite penalizar a los errores más grandes sobre los más pequeños. Normalmente se aplica en tareas de regresión y su representación algebraica es

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2-8}$$

- Entropía cruzada binaria (*BCE*): mide las diferencias entre distribuciones de probabilidad, penalizando severamente al modelo si está muy seguro de su predicción pero es errónea[3]. Se emplea en problemas de clasificación y, para el caso específico de clasificación binaria, adopta la forma

$$BCE = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)] \tag{2-9}$$

tal que n es el número de muestras, y_i son las etiquetas (0 o 1) e $\hat{y}_i$ son las predicciones (probabilidad de la clase 0 o 1).

Teniendo en cuenta estos conceptos, se presenta a continuación el método por el cual la red aprende.

2.4.1.3. Entrenamiento de la red

Para garantizar el correcto funcionamiento de una FFNN, el error entre la salida predicha y la etiqueta debe ser mínimo, es decir, la función de pérdida debe anularse. Dada la complejidad matemática que esto implica, se buscan valores de mínimos locales ajustando repetidamente los pesos sinápticos y sesgos a través del algoritmo conocido como **retropropagación del error** (*backpropagation*).

Este algoritmo se divide en dos etapas: la propagación hacia adelante y la propagación hacia atrás, de donde proviene el nombre.

En la primera fase se inicializan los pesos con valores aleatorios extraídos de una distribución normal y media nula, y se realiza la predicción. En la segunda fase se evalúa el error de la red usando una función de costo predefinida y se propaga dicho error hacia atrás, desde las últimas capas hacia las primeras. Durante este proceso se calculan las derivadas parciales de la pérdida con respecto a los parámetros[4], lo que permite ajustar iterativamente cada uno de ellos mediante alguna técnica de descenso por el gradiente. En particular, en este estudio se

[3] Una forma alternativa es pensar a la *BCE* como una medida de la "sorpresa" del modelo: si la predicción coincide con el valor real, la sorpresa es nula. No obstante, si la predicción difiere, el modelo experimenta una "gran sorpresa" ante el inesperado resultado, lo que se traduce en un alto valor de entropía.

[4] Ahora es claro por qué se exige diferenciabilidad en la función de costo.

aplican dos variantes muy conocidas: SGD (Descenso por el Gradiente Estocástico) y ADAM (Estimación Adaptativa de Momentos) [7].

El ciclo se repite a lo largo de múltiples épocas, siendo la época definida como la aplicación de este doble proceso para todo el conjunto de datos de entrenamiento disponibles. El algoritmo se detiene cuando la red converge a un mínimo o se alcanza el criterio de detención establecido.

Considerando la l-ésima capa y a $\mathcal{P}$ como la función de pérdida, los i-ésimos parámetros actualizados se determinan a partir de los antiguos según

$$\tilde{w}_i^{(l)} = w_i^{(l)} - \eta \frac{\partial \mathcal{P}}{\partial w_i^{(l)}} \quad ; \quad \tilde{b}_i^{(l)} = b_i^{(l)} - \eta \frac{\partial \mathcal{P}}{\partial b_i^{(l)}} \tag{2-10}$$

La constante η es la tasa de aprendizaje y determina la velocidad con la que el gradiente se aproxima al mínimo. Además cabe destacar que, debido a la naturaleza de las FFNNs, tanto los pesos como los sesgos de una capa dependen de los parámetros de las capas posteriores, lo que implica la existencia implícita de la regla de la cadena del cálculo diferencial en las derivadas de la ecuación (2-10).

2.4.1.4. Entrenamiento, validación y prueba

Para asegurar que los modelos son capaces de realizar predicciones precisas en situaciones del mundo real se suele fraccionar al conjunto de datos en dos subconjuntos. Por un lado, el conjunto de entrenamiento se utiliza para entrenar con ejemplos etiquetados. Por otro, el conjunto de validación proporciona nuevos datos que sirven para hacer predicciones, las cuales se comparan luego con las etiquetas. Esto contribuye a la optimización del modelo mediante la evaluación del rendimiento y el ajuste de sus hiperparámetros.

En ciertas ocasiones los datos no se separan en dos partes sino en tres. En ese caso, el tercer subconjunto es el conjunto de testeo[5] y se utiliza para poner a prueba al modelo en situaciones prácticas. Las observaciones del mismo permanecen ocultas durante todo el proceso y solo se presentan al final para una evaluación imparcial.

Es importante resaltar que en el manejo de datos reales es frecuente encontrarse con desequilibrios entre clases, donde las instancias de una categoría son considerablemente más numerosas que otras. Para evitar el sobreajuste de la clase mayoritaria y desestimar a las demás es común el uso de técnicas de balanceo de datos. Estos métodos consisten en generar nuevas observaciones de las clases minoritarias que promueven una representación equitativa de las mismas, mejorando el rendimiento y la capacidad de generalización del modelo.

[5] Cuando los datos solo se dividen en dos es habitual referirse indistintamente como conjunto de validación o conjunto de testeo, aunque en términos estrictos no son iguales.

2.5. Redes recurrentes y series de tiempo

A diferencia de las redes feedforward, existe un tipo de red neuronal donde la información de entrada no se procesa de forma independiente sino que tiene en cuenta el orden en el que se presentan los datos. Estas son las **redes recurrentes (RNNs)**. Su diseño en bucle permite generar la salida de la red empleando conjuntamente la entrada actual y la salida de la iteración previa como *inputs* (ver figura **2-5**). En otras palabras, las RNNs usan una "memoria" de los estados anteriores para identificar patrones ocultos y capturar dependencias temporales presentes en los datos.

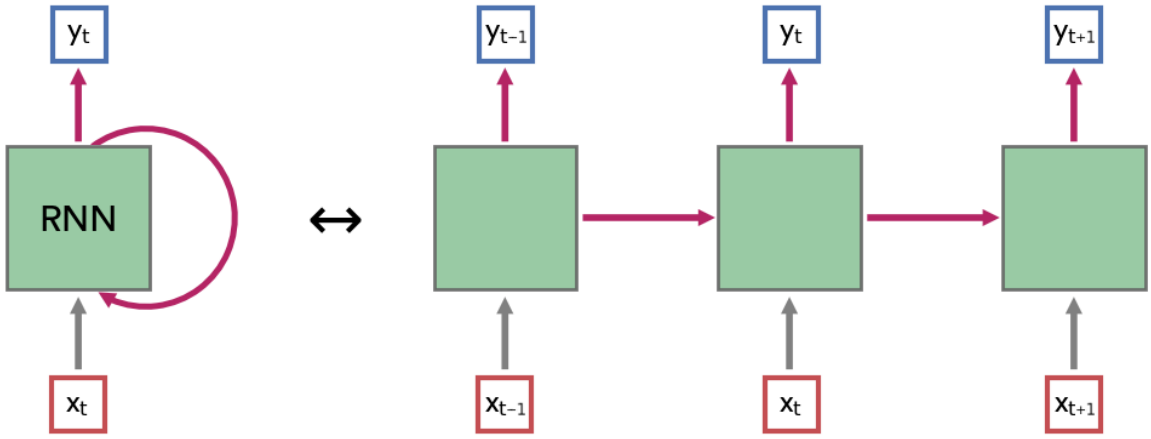


Figura 2-5.: Representación de una RNN "plegada" (a izquierda) y "extendida" (a derecha). La entrada actual y la salida del bucle anterior son los *inputs* en la nueva iteración.

El entrenamiento de estas redes se realiza mediante la **retropropagación a través del tiempo (BPTT)** [8], una versión extendida de la retropropagación básica que se aplica a sistemas dinámicos. Los parámetros correspondientes a las conexiones de la red se actualizan en un proceso iterativo de optimización, facilitando el aprendizaje y la adaptación de la red a lo largo del tiempo.

Una de las primeras aplicaciones propuestas y que continúa siendo relevante en la actualidad es el análisis de series temporales. Sin embargo, aunque en principio pueda pasar inadvertido, estos modelos enfrentan un desafío mayúsculo cuando se abordan series de tiempo prolongadas y multidimensionales.

De manera simplificada e ilustrativa, se puede pensar que en cada iteración de la RNN el resultado es multiplicado por un peso $\tilde{w}$ y se introduce nuevamente en la red. Como el proceso es cíclico, este factor se multiplica repetidas veces, aumentando exponencialmente. Si $\tilde{w} < 1$, la salida tiende a desaparecer ya que el gradiente se acerca al mínimo de la función de costo en pasos infinitesimales y lo alcanza en un tiempo infinito. En cambio, si $\tilde{w} > 1$ la salida tiende a divergir dado que el gradiente se dirige al mínimo con pasos demasiado

grandes y no puede converger (lo "saltea" sistemáticamente). A este fenómeno se lo conoce como desvanecimiento/explosión del gradiente [9].

Para mitigar dicho efecto se han desarrollado variantes de las RNNs, entre las que se destacan las **redes neuronales Long Short-Term Memory (LSTM)**, y que son las escogidas para este estudio.

2.5.1. Long Short-Term Memory (LSTM)

Las redes neuronales con memoria a corto-largo plazo [10] son una extensión de las redes recurrentes que solventan la problemática del desvanecimiento/explosión del gradiente. La diferencia entre ambas reside en que las LSTM pueden aprender dependencias a corto pero también largo plazo, lo que las hace muy útiles en problemas con datos secuenciales como el procesamiento de lenguaje natural (NLP), la generación de texto, la traducción automática o el reconocimiento de voz.

Está conformada por unidades de memoria llamadas celdas de estado, c_t, responsables de la memoria a largo plazo y estados ocultos, h_t, que se encargan de la memoria a corto plazo. Ambos están interconectados por puertas que permiten añadir o eliminar datos de la memoria de la red, regulando el flujo de información. Además, el estado oculto se bifurca para producir la salida actual del modelo, pasando previamente por una capa totalmente conectada (*fully connected*) y utilizando alguna función de activación. Una representación esquemática se presenta en la figura **2-6**.

La arquitectura se divide en cuatro partes [11]:

1. Puerta de olvido (*forget gate*): decide qué información de la etapa anterior debe olvidarse al tiempo t y cuál pasará a la celda de estado. Para ello se toma el estado oculto del paso de tiempo anterior h_{t-1} y la entrada actual x_t, y se concatenan para formar la matriz $[h_{t-1}, x_t]$. Esta matriz se introduce en una red *fully connected* que ajusta las dimensiones y luego se aplica una función de activación sigmoide, lo que resulta en un vector de salida f_t con números entre 0 y 1. Valores próximos a 0 implican omitir la información y valores cercanos a 1 la retienen.

   Matemáticamente el proceso se describe como

   $$f_t = \sigma\Big([h_{t-1}, x_t]W_f + b_f\Big) \tag{2-11}$$

   donde W_f es la matriz de pesos y b_f el vector de sesgo asociados a la puerta de olvido.

2. Puerta de entrada (*input gate*): se utiliza para agregar nuevos elementos a la memoria y en ella se ejecutan dos tareas en paralelo:

- Se repite el procedimiento de f_t para identificar qué valores serán actualizados pero con nuevos parámetros W_i y b_i, de modo que el vector de salida es

$$i_t = \sigma\Big([h_{t-1}, x_t]W_i + b_i\Big)$$

- Similar al anterior solo que se cambia la función de activación por una tangente hiperbólica para crear un vector de valores candidatos a formar parte de la nueva memoria $\tilde{c}_t$, con pesos W_c y sesgo b_c tal que

$$\tilde{c}_t = \tanh\Big([h_{t-1}, x_t]W_c + b_c\Big)$$

A continuación se multiplican ambas salidas, siendo la σ correspondiente a i_t la que determina qué información de la salida $\tilde{c}_t$ es importante mantener

$$u_t = i_t \times \tilde{c}_t \tag{2-12}$$

3. Actualización de la celda de estado: se realiza multiplicando f_t con la celda de estado previa c_{t-1} para despreciar la información irrelevante y se le suma u_t. Así, la nueva celda c_t está dada por

$$c_t = f_t \times c_{t-1} + u_t \tag{2-13}$$

4. Puerta de salida (*output gate*): se emplea para calcular el nuevo estado oculto h_t, que es una versión filtrada de la celda de estado del ítem 3. y se efectúa también en dos etapas conjuntas:

   - Se reitera el proceso de f_t con pesos W_o y sesgo b_o tal que

$$o_t = \sigma\Big([h_{t-1}, x_t]W_o + b_o\Big)$$

   - Se redimensiona c_t al intervalo

$$[-1, 1]$$

     por medio de $\tanh(c_t)$.

Por último, el estado oculto final al tiempo t es el vector generado por la multiplicación de estas salidas

$$h_t = o_t \times \tanh(c_t) \tag{2-14}$$

y se transfiere al paso de tiempo siguiente $t + 1$.

NOTA: los W_j y b_j $(j = f, i, c, o)$ se aprenden durante el entrenamiento a través de las conexiones neuronales. Además, $x_t \in \mathbb{R}^{B\times m}$, $h \wedge c \in \mathbb{R}^{B\times n}$, $W_j \in \mathbb{R}^{(n+m)\times n}$ y $b_j \in \mathbb{R}^{1\times n}$, con B el tamaño de lote (*batch*), m el número de variables de entrada al tiempo t y n el número de neuronas en la capa LSTM. Con esto se puede comprobar que $[h_{t-1}, x_t] \in \mathbb{R}^{B\times(n+m)}$ y es factible multiplicar por la matriz W_j. El resultado de ese producto $\in \mathbb{R}^{B\times n}$, coincidente con las dimensiones de c. Por último, a pesar de la incompatibilidad entre $\mathbb{R}^{B\times n}$ y $\mathbb{R}^{1\times n}$, la adición del sesgo es posible sumando b a cada una de las filas de $[h_{t-1}, x_t]W_j$.

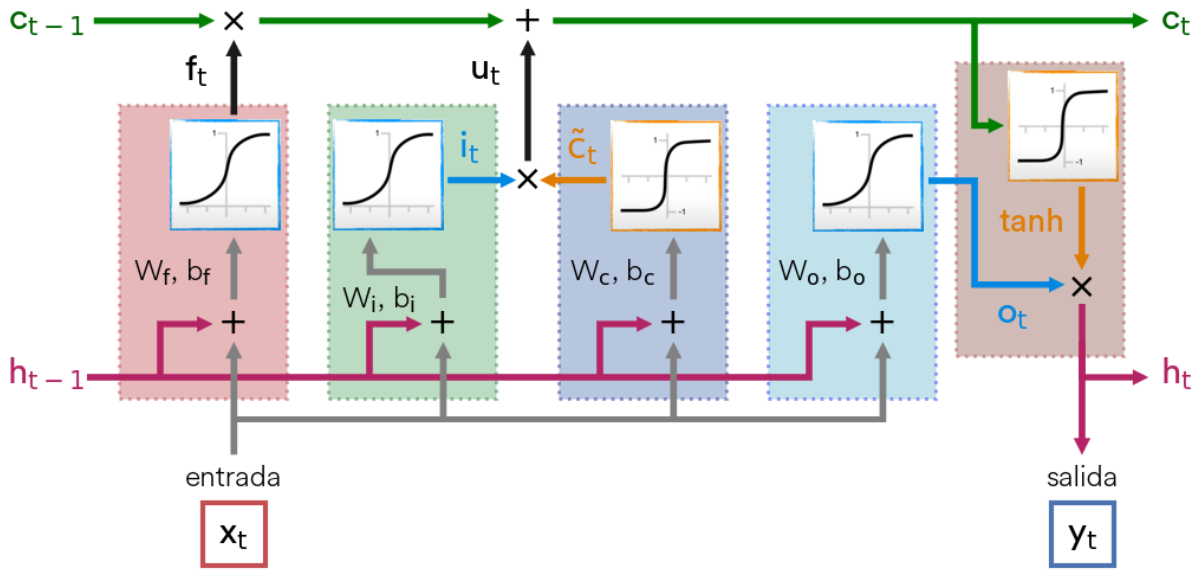


Figura 2-6.: Unidad LSTM al tiempo t. Sobre las flechas verdes superiores se traslada la celda de estado c y por las púrpuras inferiores el estado oculto h. El rectángulo azul representa la puerta de olvido (salida f_t), el verde y anaranjado a la puerta de entrada (salida u_t) y el violeta y rojo a la puerta de salida (salida h_t). Los símbolos '+' y '×' indican la operación aplicada. Imagen adaptada de [12].

2.6. Criterios para evaluar el desempeño de los modelos

La evaluación del rendimiento de los modelos es esencial para comprobar su correcta operatividad, verificar la capacidad predictiva e interpretar apropiadamente los resultados. En esta sección se examinan diferentes métricas e indicadores que proporcionan una medida de la eficacia de las arquitecturas implantadas.

2.6.1. Matriz de confusión

La matriz de confusión [13] se trata de una matriz de tamaño $c \times c$ (c el número de diferentes clases) que muestra las clasificaciones reales y predicciones de un modelo (figura **2-7**). Para el caso binario, $c = 2$, y se definen los siguientes términos:

- Verdaderos negativos (TN): la predicción del modelo coincide con el valor real, que es negativo.
- Falsos positivos (FP): la predicción del modelo es positiva pero el valor real es negativo (error Tipo I).
- Falsos negativos (FN): la predicción del modelo es negativa pero el valor real es positivo (error Tipo II).
- Verdaderos positivos (TP): la predicción del modelo coincide con el valor real, que es positivo.

2.6.2. Métricas

A partir de los datos obtenidos de la matriz se derivan las métricas:

- Exactitud (*A*ccuracy): es una medida de cuántas predicciones correctas hizo el modelo en el conjunto total y está dada por

$$A = \frac{TN + TP}{TN + FP + FN + TP}$$

- Sensibilidad (*R*ecall): indica la tasa de éxito en la detección de la clase positiva sobre el total de casos positivos y se expresa como

$$TPR = \frac{TP}{FN + TP}$$

- Precisión: mide el número de predicciones positivas correctas que hizo el modelo sobre el total de elementos positivos y está descrito por

$$P = \frac{TP}{FP + TP}$$

- Especificidad: refleja la proporción de aciertos al clasificar correctamente la clase negativa y se representa mediante

$$TNR = \frac{TN}{TN + FP}$$

2.6.3. Umbral de clasificación y curvas ROC

En la figura **2-7** se observa un umbral que delimita la distribución entre las clases negativa (a izquierda) y positiva (a derecha). Cuando la barrera se desplaza hacia la derecha, los FP (rojo) disminuyen pero aumentan los FN (celeste), y recíprocamente si se mueve hacia la izquierda. El problema consiste entonces en escoger el punto de corte que minimice ambos FP y FN, lo que se logra analizando la curva ROC. Dicha curva es un diagrama que exhibe la relación entre $1 - TNR \equiv FPR$, en el eje de las abscisas, y TPR, en el eje de las ordenadas (ver figura **2-8**). En ella, el umbral se representa con un punto donde la sensibilidad y especificidad son máximos, etiquetando negativamente a los valores inferiores a este y positivamente a los superiores.

El área debajo de la curva, AUC, también proporciona información útil sobre el desempeño general de un modelo: si AUC = 1 discrimina las clases perfectamente (comportamiento ideal); si AUC = 0,5 es incapaz de hacer distinciones (resultado aleatorio). Por tanto, una curva cuya pendiente se aproxime al ángulo superior izquierdo de la figura **2-8**, esto es, al punto (0,1), tendrá mejor rendimiento. En contraste, una curva que se acerque a la diagonal resultará en un mal estimador.

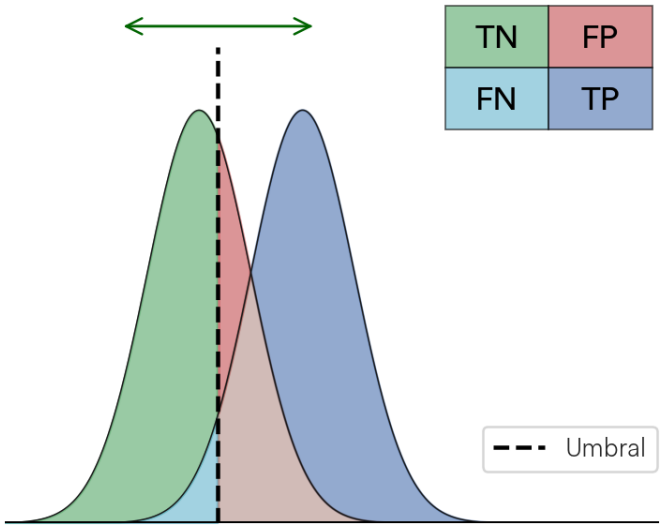


Figura 2-7.: Distribución de clases divididas por el umbral y (arriba) matriz de confusión. A izquierda se encuentran las predicciones negativas (TN y FN) y a derecha las positivas (FP y TP). En la región marrón se solapan predicciones de ambas clases.

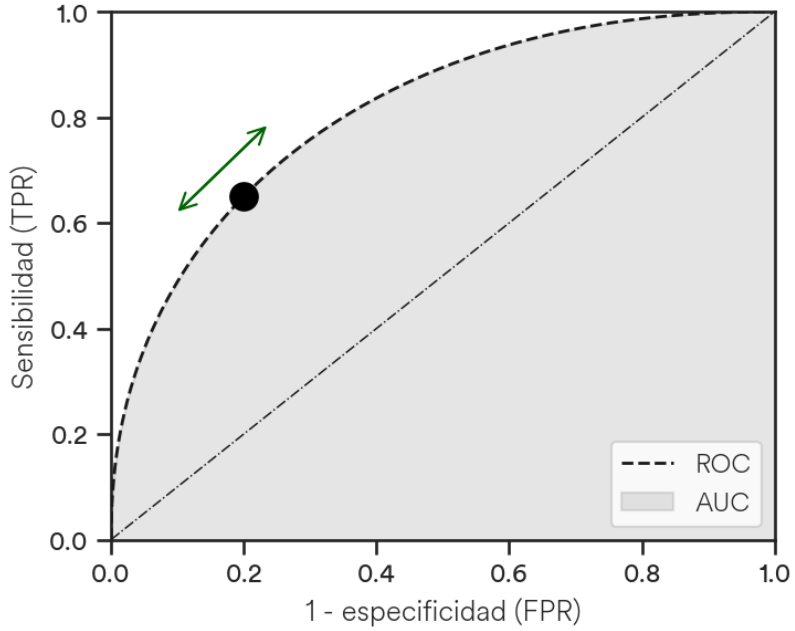


Figura 2-8.: Curva ROC y su correspondiente AUC. El punto de corte es aquel con mayor sensibilidad y especificidad, y varía conforme lo hace el umbral en la figura **2-7**.

2.7. Sistema de puntuación y políticas de acceso al crédito

El *scoring* bancario es un método utilizado por las entidades financieras para estimar la capacidad de impago de un individuo o persona jurídica que solicita el acceso a un préstamo, crédito o financiación, en base a múltiples factores como el nivel de deuda, el historial crediticio, los ingresos, entre otros. En el caso puntual de las empresas, la disponibilidad de financiamiento es un elemento central para impulsar la innovación, el crecimiento económico, aumentar la productividad y mejorar la igualdad de oportunidades para emprender [14, 15]. Sin embargo, existen fallas de mercado que obstaculizan el financiamiento, lo que provoca un aumento en las tasas de interés, créditos con vencimientos más cortos, mayores exigencias en las garantías, reducidos montos de los préstamos o incluso la denegación de los mismos [16, 17].

La principal falla es la información asimétrica [18] producida por la disparidad de información entre el prestamista y el prestatario: los acreedores tienen información incompleta acerca de la solvencia de los solicitantes, mientras que estos últimos tienen un conocimiento más detallado de su situación financiera. Como resultado, los proyectos de inversión presentados por las empresas (generalmente las más pequeñas) suelen ser vistos por los eventuales inversores como de alto riesgo, sin tener en cuenta la calidad del proyecto presentado [16].

Otro fallo del mercado se vincula a las economías de escala donde, a medida que el tamaño del préstamo crece, el costo medio de la evaluación del proyecto[6] es cada vez menor. De esta manera, los prestamistas tienen un mayor incentivo para concentrarse en empresas más grandes, afectando negativamente a las de menor tamaño [19–21].

Estos aspectos obligan a la creación de métodos de *scoring* que evalúen y gestionen apropiadamente el incumplimiento del pago para determinar si se otorgan finalmente los créditos solicitados. En base a esto, en la siguiente sección se describe la idea general de la problemática a resolver.

2.8. Descripción del problema

Como se ha señalado ya, en este trabajo se desea crear y comparar, con diversas técnicas, modelos capaces de puntar a las empresas en base a su situación de deuda. En particular, se modela la morosidad a futuro de las compañías en términos de un conjunto de variables independientes según la ecuación

$$y_{t+1} = f\Big(y_t, \ldots, y_{t-k}, x_t^1, \ldots, x_{t-k}^1, x_t^2, \ldots, x_{t-k}^2, \ldots, x_t^n, \ldots, x_{t-k}^n\Big) \tag{2-15}$$

donde y_t es la situación de deuda de la empresa a tiempo t y x^i son las $i = 1, \ldots, n$ características independientes. Como se puede observar, el valor futuro de y_{t+1} depende tanto de atributos asociados al instante actual t y a tiempos pasados $t - k$, como así también de los propios valores de y pero en momentos previos. Queda claro entonces que se trata de un problema complejo de predicción de valores en una serie temporal multidimensional.

Para hacer frente a este problema se dispone de $n = 10$ variables, cada una con mediciones en 12 periodos de tiempo diferentes ($k = 11$) y se define la morosidad como una variable dicotómica, donde **0** implica ausencia de mora y **1** indica mora. Centrándose en tales consideraciones y utilizando los modelos LogReg, RanFor y LSTM examinados anteriormente se estima el valor de y_{t+1}.

De esta manera, el objetivo último del presente trabajo es seleccionar el modelo que minimice los riesgos a la hora de decidir si se otorga (o no) el crédito por parte de la institución acreedora. Para ello, no solo es importante predecir la situación de la deuda a futuro sino que debe complementarse con la valoración de dos situaciones concretas: (i) rechazar la solicitud de una empresa financieramente sólida pues el modelo la calificó como no apta para su devolución (error tipo I) y (ii) conceder el crédito a una entidad con alto riesgo de impago debido a que el modelo la catalogó como solvente (error tipo II). A este respecto, el tratamiento de los datos, la metodología empleada y el análisis de los resultados son parte esencial y

[6] El costo de evaluación de un proyecto hace referencia a los gastos iniciales necesarios para determinar si el mismo será factible y rentable. Este costo incluye los recursos económicos, humanos y el tiempo requerido para realizar el análisis.

se tratan con mayor profundidad en los próximos capítulos, donde se desarrolla la temática central del documento.

3 Datos y metodología

En el análisis de datos es frecuente encontrarse con datos incompletos o inconsistentes, por lo que es necesario prepararlos adecuadamente. Por tal motivo, su preprocesamiento es importante y constituye la primera sección de este capítulo. En la segunda se discute la optimización de los hiperparámetros y en la tercera se aplican los modelos.

3.1. El conjunto de datos

El conjunto de datos surge de la selección de ciertas características relativas a las bases *Deudores* y *Cheques*, extraídas del sitio oficial de la Administración Federal de Ingresos Públicos (AFIP) [22] y que corresponden a empresas argentinas. La primera de ellas posee información sobre la deuda de las entidades con diferentes instituciones financieras mientras que la segunda hace referencia a la relación entre empresas y cheques. Abarcan un lapso de tiempo de aproximadamente cuatro años, desde abril de 2018 hasta julio de 2022, aunque se suprimen los meses de enero de 2019 y abril de 2020 por falta de datos y debido al inicio de la pandemia respectivamente. Esto resulta en un total de $t = 49$ meses o periodos, que se ordenan de forma cronológica. Además, aquellas entidades que no aportaron datos durante todos los periodos de estudio se eximen del análisis, reduciendo la muestra a 51.141 firmas.

Así pues, las características más relevantes de cada una se unifican para generar el nuevo dataset, compuesto por las siguientes 11 variables, todas al tiempo t:

- $y_t \equiv SD_t$: es la situación de deuda máxima alcanzada por una empresa. Si bien *Deudores* contempla inicialmente siete categorías (de 1 a 6 y 11) de acuerdo con la disposición del Banco Central de la República Argentina (BCRA) [23], se eliminan 6 y 11, de manera que SD se transforma en una variable dicotómica: la clase **0** representa las entidades cuya categoría máxima es 1 (no morosas) y la clase **1** a aquellas con categoría máxima 2, 3, 4 o 5 (morosas).

- $\diamond$ $x_t^1 \equiv B_t$: número de bancos adeudados por la empresa.
- $\diamond$ $x_t^2 \equiv BPu_t$: del total de bancos B_t, cuántos de estos son públicos.
- $\diamond$ $x_t^3 \equiv F_t$: cantidad máxima de fideicomisos con los que la empresa tiene deuda.
- $\diamond$ $x_t^4 \equiv CF_t$: cantidad máxima de compañías financieras con los que la empresa tiene deuda.
- $\diamond$ $x_t^5 \equiv TM_t$: cantidad máxima de tarjetas, mutuales u otros con los que la empresa tiene deuda.
- $\diamond$ $x_t^6 \equiv P_t$: monto total de crédito adquirido por la empresa. Dado que esta variable es continua, se normaliza entre [0,1] para aplicar correctamente los modelos.
- $\diamond$ $x_t^7 \equiv CR1_t$: cantidad de cheques rechazados por causa 1 (vicios formales).
- $\diamond$ $x_t^8 \equiv MR1_t$: monto total de los $CR1_t$. También se normaliza al [0,1].
- $\diamond$ $x_t^9 \equiv CR2_t$: cantidad de cheques rechazados por causa 2 (falta de fondos).
- $\diamond$ $x_t^{10} \equiv MR2_t$: monto total de los $CR2_t$. Normalizada al [0,1].

3.2. Optimización de hiperparámetros

Dado el significativo número de hiperparámetros que pueden influir en el funcionamiento de los modelos, se implementan diversas estrategias para elegir un conjunto reducido de los más apropiados en cada arquitectura, manteniendo los restantes sus configuraciones por defecto.

Para LogReg se efectúa una búsqueda exhaustiva sobre una grilla de hiperparámetros previamente establecidos, examinando a través de alguna métrica la eficiencia del método para cada combinación de valores. El arreglo que maximiza el desempeño general se designa como el conjunto óptimo.

En el caso de RanFor se apela a un enfoque similar pero donde las combinaciones se seleccionan de manera aleatoria. En ambas búsquedas la métrica utilizada es *recall*.

En contrapartida, en la LSTM no se aplica ninguna de estas técnicas debido al extenso tiempo computacional. En cambio, se exploran manualmente ciertos hiperparámetros, así como el número de capas, y estos se fijan para todas las camadas. No obstante, para contar con una cierta variedad de modelos entre los que se pueda realizar una elección, la exploración se ejecuta ocho veces: cuatro con SGD como optimizador y otras cuatro con ADAM, optando por el que mejor resultado presente de cada tipo. En estas redes también se usa el criterio de *Early Stopping*, que consiste en interrumpir el entrenamiento si el rendimiento en el conjunto

de validación deja de mejorar, evitando el sobreajuste. Por último, la capa de salida es una *fully connected* con función de activación sigmoide.

3.3. Implementación de los modelos

La ecuación (2-15) se formula en función de las variables del problema como

$$SD_{t+1} = f\Big(SD_t, \ldots, SD_{t-11}, \ldots, B_t, \ldots, B_{t-11}, \ldots, CR1_t, \ldots, CR1_{t-11}\Big) \tag{3-1}$$

donde SD_{t+1} es la **variable objetivo** y representa la situación de deuda al periodo siguiente al que se está analizando. $B_t, \ldots, B_{t-11}$ son el número de bancos en los 12 meses de estudio, $CR1_t, \ldots, CR1_{t-11}$ son la cantidad de cheques rechazados por causa 1 en esos 12 meses y sucesivamente con las restantes variables. Notar nuevamente que SD se incluye, a su vez, entre las características de entrada.

El cálculo de la variable objetivo se realiza utilizando el mecanismo de ventanas deslizantes (también denominadas camadas): primero se elige un tamaño de ventana inicial, que equivale a cuánto hacia atrás en el tiempo se está considerando (en este caso son 12 periodos) y se hace la estimación futura SD_{t+1}. Luego de la predicción, se incorpora una nueva observación al modelo y se descarta el primer punto. Así, el tamaño de la ventana permanece constante en el tiempo a la vez que se adquieren nuevos datos (ver figura **3-1**). El proceso se replica hasta completar la muestra, lo cual supone un total de 38 ventanas para el problema actual.

Teniendo en cuenta lo anterior y como $t = 1, \ldots, 49$, la primera predicción posible será SD_{13} ($t = 12$; camada $= 1$) y la última, SD_{49} ($t = 48$; camada $= 37$).

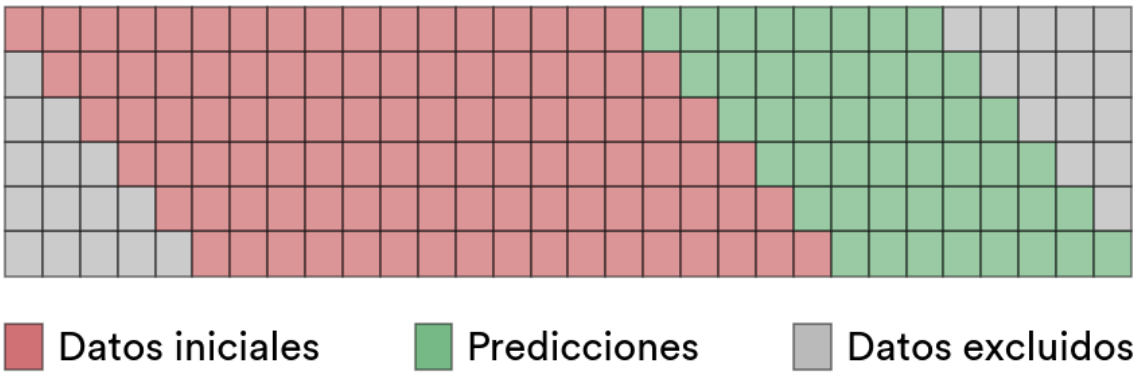


Figura 3-1.: Esquema de ventana deslizante para una muestra de 30 datos. El tamaño de ventana es 17 y se hacen 8 estimaciones a futuro, completando la muestra mediante 6 ventanas. Figura adaptada de [24].

Los códigos utilizados se pueden consultar en https://github.com/MartinTrucco22/tesis.

Para entrenar los modelos es necesario hacer una serie de transformaciones preliminares al conjunto de datos:

En primer lugar se separan las características ($\mathbf{X}$) asociadas a la i-ésima camada de la etiqueta ($\mathbf{y}$), referente a la $(i+1)$-ésima camada. Posteriormente se hace un balanceo entre las clases usando SMOTE[1], un algoritmo que genera muestras sintéticas de la clase minoritaria bajo el supuesto de que instancias próximas entre sí tienen mayor probabilidad de pertenecer a una misma clase [26]. En particular, la proporción original entre **0** y **1** es aproximadamente 90 % vs 10 % mientras que la relación posterior al remuestreo es 60 % vs 40 %.

Luego se fraccionan $\mathbf{X}$ e $\mathbf{y}$, asignando el 80 % de la muestra al conjunto de entrenamiento y el 20 % restante al conjunto de testeo.

Con estas modificaciones, es oportuno comenzar con el entrenamiento:

En la camada inicial se aplica la búsqueda de hiperparámetros a los datos de entrada (o se determinan manualmente) y se entrena el modelo. Dicho estimador, con sus pesos y sesgos, se establece como base para la siguiente iteración, aprovechando el preentrenamiento y generando una transferencia de aprendizaje [27, 28] entre las sucesivas camadas. Tal procedimiento se replica durante 37 camadas, donde se predice por última vez, y da como resultado el modelo final de cada arquitectura junto a las "probabilidades"[2] de pertenencia de una dada observación a la clase **1** (mora)[3].

Para comprobar la eficacia de los diseños estudiados, con el fin de concluir si el banco adjudica el crédito al solicitante sin comprometer el capital del prestamista, es necesario minimizar los errores de tipo I y II. Por este motivo, se determinan la precisión y la sensibilidad de la clase **1**, la exactitud de los estimadores y su pérdida, la curva ROC, el AUC y el umbral que optimiza la separación de clases. Adicionalmente, se grafica la curva ROC, la matriz de confusión de cada método y, para las LSTM, el error de entrenamiento y testeo en función de las épocas en las camadas comprendidas.

[1] Para una lectura adicional ver [25].

[2] Es posible asignar probabilidades reales a las predicciones de LogReg y RanFor aunque no ocurre lo mismo con las LSTM. Esto es así porque, si bien la función sigmoide mapea la salida al intervalo [0,1] y mantiene el orden inicial, las redes también dependen de pesos y sesgos que se inicializan aleatoriamente. Cada configuración resulta en una red diferente, que puede clasificar igualmente bien pero con valores de salida distintos. En otras palabras, para un mismo conjunto de datos de entrada, la probabilidad predicha varía.

[3] Es conveniente estudiar la clase **1** porque se prefiere identificar correctamente a las empresas propensas a incumplir con el pago.

4 Resultados y discusión

Siguiendo la metodología planteada en el capítulo 3, este capítulo se compone de dos secciones: en la primera, relacionada con la sección 3.1, se hace un análisis exploratorio del conjunto de datos y en la segunda, vinculada a las secciones 3.2 y 3.3, se presentan los modelos finales a utilizar y los resultados obtenidos de su aplicación.

4.1. Análisis exploratorio del dataset

Con el fin de destacar algunas características propias al dataset empleado, se exponen las tablas **4-1**, **4-2** y **4-3**.

La tabla **4-1** muestra el porcentaje de cheques rechazados por vicios formales, $\%CR1$, y por falta de fondos, $\%CR2$, para cada año evaluado. Asimismo, se incluyen los porcentajes correspondientes al monto total de estos rechazos, donde $\%MR1$ refiere a la proporción debida a $CR1$ y $\%MR2$ a $CR2$. Se observa que la razón principal del rechazo de los cheques se debe a la falta de fondos, siendo el importe de estos mayoritario. No obstante se puede notar que, pese al reducido número de rechazos por vicios formales, el monto es significativamente alto en el primero, segundo y cuarto año.

Año[1]	Cantidad		Monto	
	$\%CR1$	$\%CR2$	$\%MR1$	$\%MR2$
Abril 2018 – Abril 2019	0,60	99,40	6,09	93,91
Mayo 2019 – Mayo 2020	1,04	98,96	7,48	92,52
Junio 2020 – Mayo 2021	2,75	97,25	1,30	98,70
Junio 2021 – Julio 2022	2,99	97,01	9,37	90,63

Tabla 4-1.: Porcentaje de cheques rechazados (y monto que representan) por vicios formales (causa 1) y por falta de fondos (causa 2), en cada año de estudio.

Año	Situación de deuda				
	$\%SD1$	$\%SD2$	$\%SD3$	$\%SD4$	$\%SD5$
Abril 2018 − Abril 2019	92,39	2,01	1,58	2,08	1,94
Mayo 2019 − Mayo 2020	90,06	1,70	1,30	2,00	4,94
Junio 2020 − Mayo 2021	89,53	0,98	1,01	1,26	7,22
Junio 2021 − Julio 2022	87,01	1,56	1,36	1,83	8,24

Tabla 4-2.: Proporción de empresas que alcanzaron la máxima categoría de situación de deuda $(1-5)$ en un año.

Año	Bancos		Otras entidades		
	$\%BPu$	$\%BPr$	$\%F$	$\%CF$	$\%TM$
Abril 2018 − Abril 2019	21,24	78,76	2,27	44,86	52,87
Mayo 2019 − Mayo 2020	19,61	80,39	0,73	24,51	74,76
Junio 2020 − Mayo 2021	19,25	80,75	1,33	24,06	74,61
Junio 2021 − Julio 2022	20,18	79,82	2,53	19,59	77,88

Tabla 4-3.: Porcentaje de empresas con deuda a bancos públicos y privados, fideicomisos, compañías financieras y tarjetas, mutuales, otros en cada año de estudio.

En la tabla **4-2** se representa el porcentaje de firmas que alcanzaron la máxima categoría de situación de deuda[2] en un año, $\%SD$. Por otro lado, en la tabla **4-3** se expone, del total de bancos a los que se pidió un crédito en todo el año, la deuda de las empresas con los bancos públicos, $\%BPu$, y con los bancos privados, $\%BPr$. Con respecto a las últimas tres columnas, estas indican, porcentualmente, el número máximo de fideicomisos, $\%F$, compañías financieras, $\%CF$, y tarjetas, mutuales u otros, $\%TM$, a los que recurrieron las firmas para adquirir deuda en un año.

Se puede observar que, en general, la cantidad de bancos públicos adeudados se mantiene constante entre el 19 % y 21 %, señalando una clara inclinación hacia los bancos privados. En cuanto a la situación de deuda, existe una marcada diferencia entre las empresas consideradas no morosas ($SD = 1$, clase **0**) y las morosas ($SD \neq 1$, clase **1**), promediando las primeras un 90 % durante los cuatro años. Esto motiva la necesidad de equilibrar las clases como se mencionó en el capítulo anterior.

Finalmente, al comparar las últimas tres instituciones financieras de la tabla **4-3**, se refleja un bajo porcentaje de fideicomisos adeudados, lo cual sugiere que las compañías financieras, tarjetas, mutuales u otros rubros se reparten la mayor cuota del mercado crediticio. En particular, en el primer año las compañías financieras poseen el 45 % de la deuda total,

[1]Si bien el inicio y final de cada año varía, todos contienen 12 meses salvo el último, que es de 14 meses.
[2]Conforme a la normativa del BCRA.

aunque dicha cantidad se reduce al $20\,\% - 25\,\%$ en los años siguientes, siendo esa diferencia absorbida por las tarjetas y mutuales.

4.2. Modelos y resultados

La presentación de los hiperparámetros utilizados varía pues, en el caso de LogReg y RanFor, las búsquedas en grilla y aleatoria se realizan en cada una de las camadas, a diferencia de las LSTM donde los parámetros se fijan para todas las iteraciones[3]. En consecuencia, la tabla **4-4** muestra los hiperparámetros óptimos para los modelos LogReg y RanFor en la última iteración, así como los seleccionados manualmente para las redes LSTM y que son válidos para todo el procedimiento.

Estimador	Hiperparámetro	Valor
LogReg	`C`	`1.0`
	`max_iter`	`10000`
	`solver`	`liblinear`
RanFor	`n_estimators`	`110`
	`min_samples_split`	`5`
	`min_samples_leaf`	`1`
	`max_depth`	`25`
	`criterion`	`entropy`
LSTM	`optimizer`	`SGD` `Adam`
	`learning_rate`	`0.5` `0.001`
	`loss`	`mean_squared_error`
	`batch_size`	`64`
	`units`	`50`
	`activation`	`sigmoid`
	`Dropout`	`0.2`
	# capas[4]	2

Tabla 4-4.: Hiperparámetros resultantes del cálculo de la última camada.

Por otra parte, las curvas ROC se muestran en la figura **4-1** donde visualmente se puede advertir que la curva del RanFor es la que más se acerca a la esquina superior izquierda. Sin embargo, un análisis más detallado puede hacerse basado en la tabla **4-5**, que evalúa la

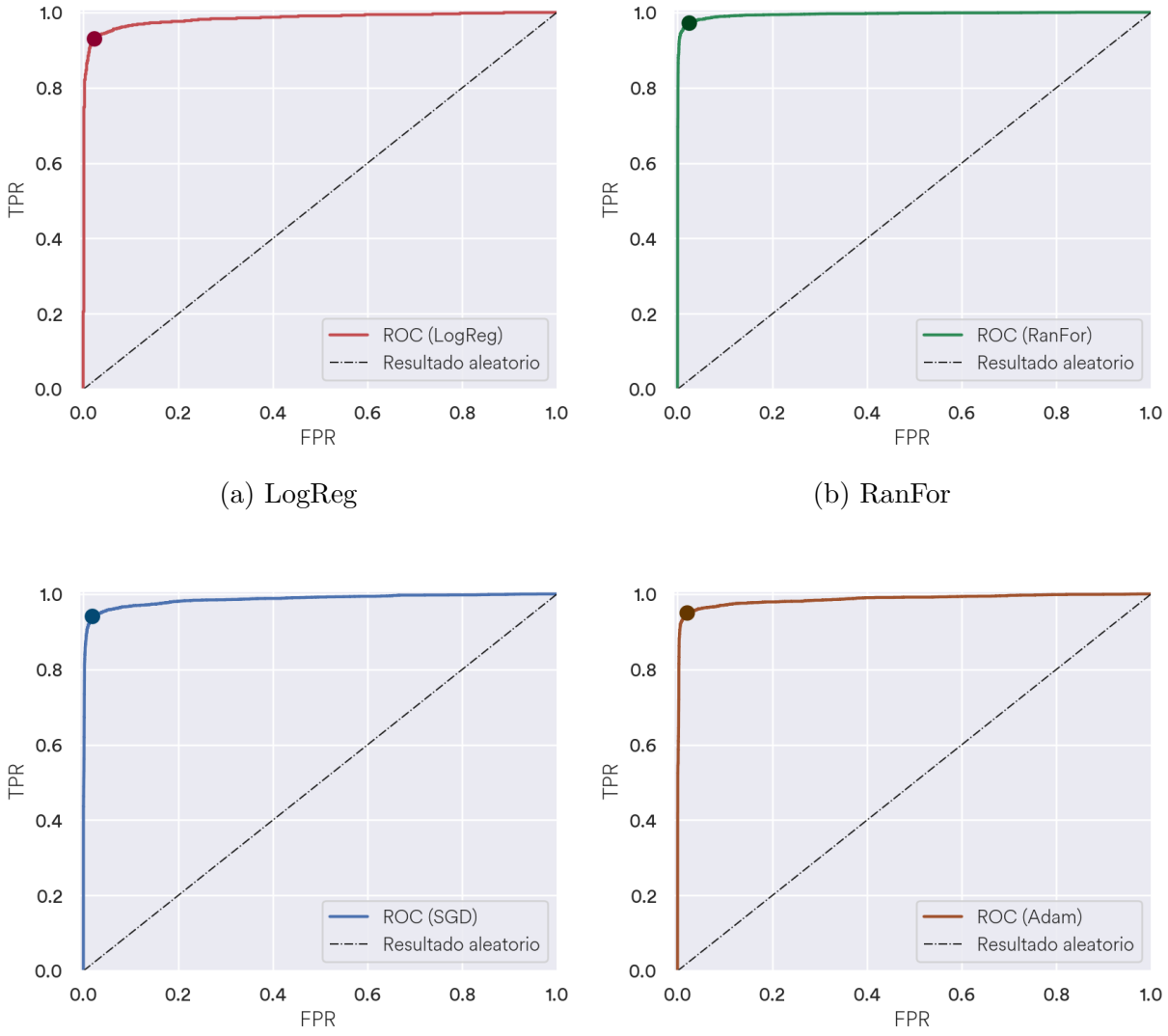
[3]Las pruebas alternativas se pueden ver en el Apéndice A.

[4]Pese a que el # capas no es un hiperparámetro, se lo incluye en la tabla por practicidad.

pérdida[5], AUC y el umbral óptimo que maximiza la clasificación de clases, y en la tabla **4-6**, donde se especifican las métricas de sensibilidad, precisión y exactitud.

Notar que el menor error es el de RanFor, seguido por LSTM (Adam) con un aumento del 31,71 %, LSTM (SGD) con un incremento del 47,96 % y LogReg cuyo error difiere un 77,54 % del primero.

De igual manera, RanFor es quien tiene mayor AUC, un 0,87 % ligeramente por delante de LSTM (Adam) y 0,9 % de LSTM (SGD), finalizando con LogReg que es 1,13 % superior.



(a) LogReg

(b) RanFor

(c) LSTM (SGD)

(d) LSTM (Adam)

Figura 4-1.: Curvas ROC.

[5]Como la pérdida de ambas LSTM es MSE, se comparan todos los estimadores en torno a ese indicador.

Estimador	Pérdida	AUC	Umbral óptimo
LogReg	0,030085	0,983535	0,231872
RanFor	**0,016945**	**0,994803**	0,221456
LSTM (SGD)	0,025072	0,985842	0,247458
LSTM (Adam)	0,022318	0,986196	0,186512

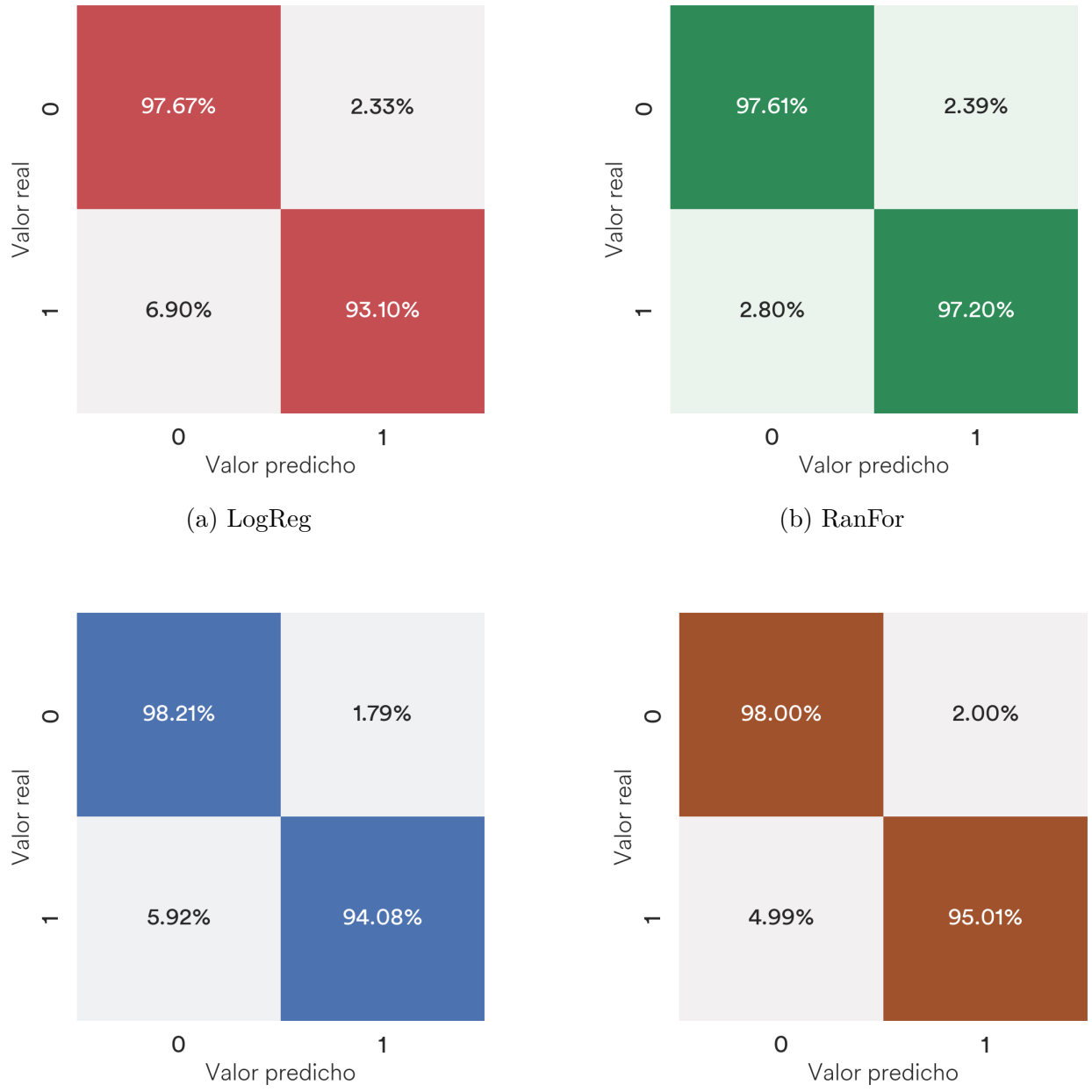
Tabla 4-5.: Valores de error, AUC y umbral óptimo de cada estimador.

También se puede observar a partir de las matrices de confusión de la figura **4-2** y de la tabla **4-6** que las dos redes son las que mejor clasifican a la clase 0 pero tienen más inconvenientes con la clase 1, por lo que disminuyen los FP y aumentan los FN. En contraste, RanFor tiene menor tasa de clasificación de la clase 0 a cambio de una mejora sustancial en la distinción de la clase 1. Esto último es de suma importancia ya que el interés principal reside en identificar de manera rigurosa a las empresas morosas. Y es ahí donde radica la diferencia: al ser RanFor un método más equilibrado, compensa el mayor error de tipo I (menor precisión) reduciendo el error de tipo II (mayor sensibilidad). Este enfoque tiende a ser más adecuado puesto que las escasas empresas ubicadas en el límite entre la aprobación o negación de un préstamo, no lo obtienen. Así, el banco pierde, tal vez, a un potencial cliente pero a cambio no pone en peligro su integridad financiera. Sumado a esto, RanFor es, a la vez, el modelo con rendimiento general (exactitud) superior y por consecuencia se erige como el método más robusto y práctico para abordar de manera efectiva los casos críticos analizados.

Por último, mencionar que LogReg es el diseño con menor eficacia de los cuatro, lo que es esperable debido a su menor complejidad, aunque es capaz de lograr resultados sorprendentemente altos.

Estimador	Sensibilidad	Precisión	Exactitud
LogReg	0,931034	0,940793	0,963667
RanFor	**0,971965**	0,941864	**0,974926**
LSTM (SGD)	0,940847	**0,954494**	0,970375
LSTM (Adam)	0,950098	0,949832	0,971492

Tabla 4-6.: Sensibilidad, precisión y exactitud de cada estimador.



(a) LogReg

(b) RanFor

(c) LSTM (SGD)

(d) LSTM (Adam)

Figura 4-2.: Matrices de confusión.

4.2.1. Comparación de las redes LSTM

Para concluir, se ilustran en las figuras **4-3**a y **4-3**b los errores de entrenamiento y testeo de las redes en función de las épocas en las sucesivas camadas, es decir, las épocas de cada iteración se grafican acumulativamente una tras otra. La variación de este número en las figuras se debe al uso del *EarlyStopping*, cuyo criterio de corte no depende de un número fijo sino que tiene en cuenta el rendimiento del modelo.

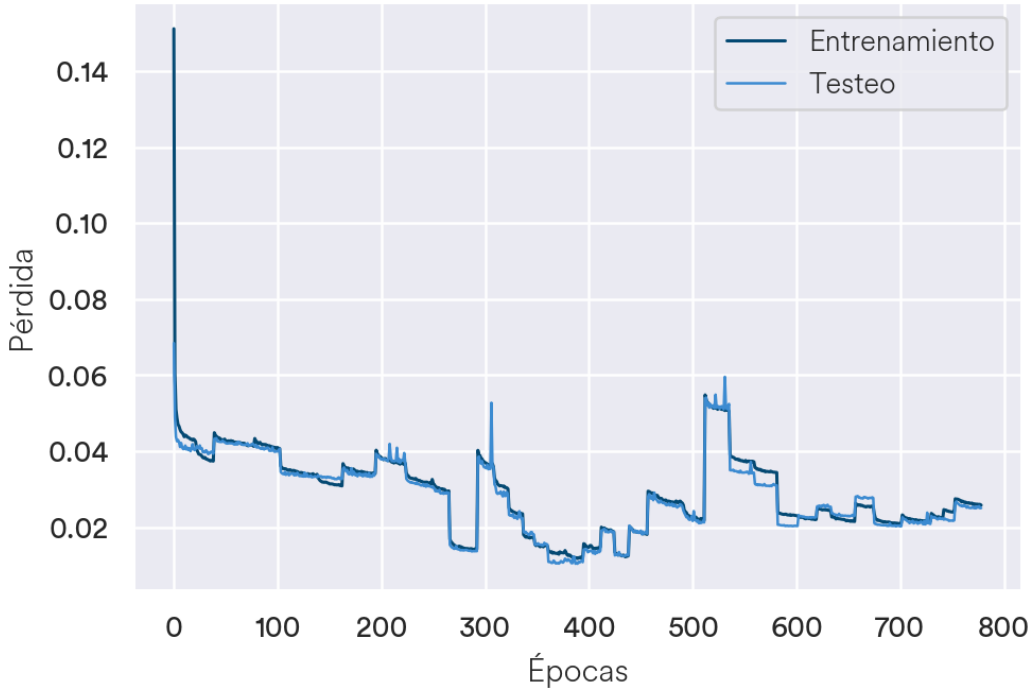
Centrándose en el contenido de las figuras, se observa el mismo gráfico solo que es más acentuado en LSTM (Adam): la curva de aprendizaje desciende para la primera camada como es previsible y luego tiene un pico, que indica el incio de la siguiente camada. Este comportamiento se repite continuamente con cada cambio de camada. La razón de dicha conducta es que, al emplear la transferencia de aprendizaje, el modelo se inicializa preentrenado, de modo que tiene que reajustar los parámetros anteriores a los nuevos datos, ocasionando los picos. A medida que avanzan las épocas, estos parámetros se van adaptando al problema actual y, por tanto, la curva de aprendizaje disminuye. Sin embargo, hay camadas (aproximadamente en las épocas 280 y 520 para SGD, 480 y 900 para Adam) donde la pérdida se incrementa sustancialmente, lo que refleja una anomalía en ciertos conjuntos de datos.

Los errores obtenidos en el entrenamiento y testeo al completar el proceso se muestran en la tabla **4-7**, mientras que los resultantes al finalizar cada camada se pueden encontrar en el Apéndice B.

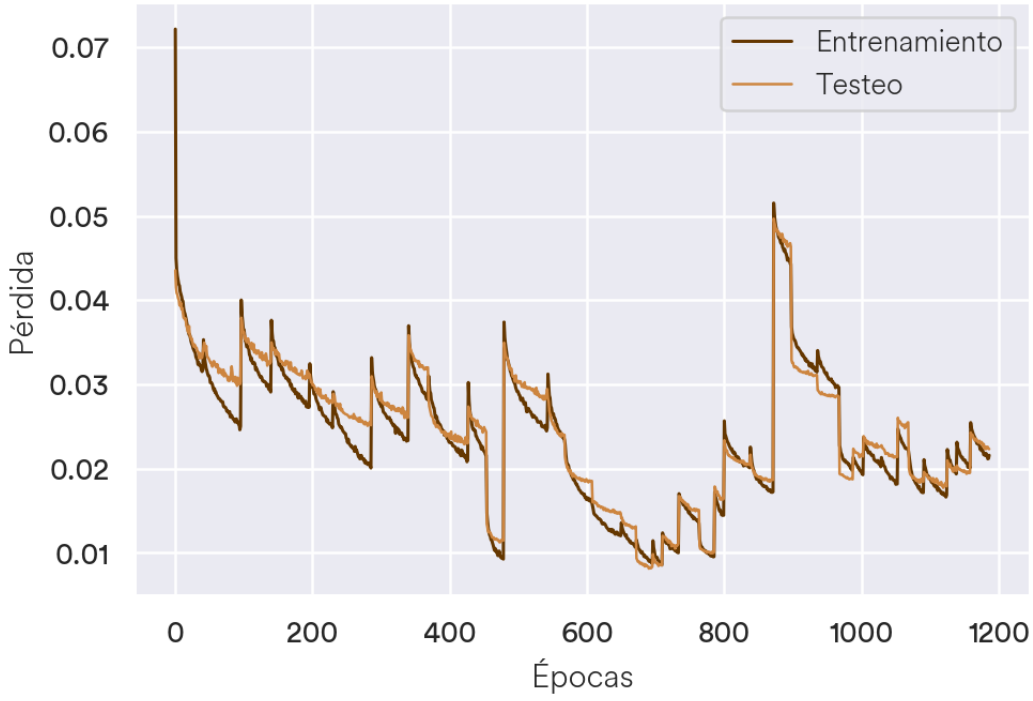
Con todo esto es posible afirmar que el optimizador Adam presenta resultados más homogéneos y contundentes que SGD (quien solo es capaz de superarlo en la métrica precisión por un escaso 0,5 %), evidenciando un mejor funcionamiento como se comprobó anteriormente.

Estimador	Pérdida entrenamiento	Pérdida testeo
LSTM (SGD)	0,024335	0,028050
LSTM (Adam)	**0,021587**	**0,022318**

Tabla 4-7.: Error de entrenamiento y testeo de las LSTM para la última época de la camada 37.



(a)



(b)

Figura 4-3.: Error de entrenamiento y validación de la red con (a) SGD y (b) Adam en función de las épocas.

5 Conclusiones

En este trabajo se exploraron diversas arquitecturas con el propósito de predecir qué empresas son más proclives al impago de un préstamo, lo que permite a la entidad prestadora determinar si concede o no dicho financiamiento. En particular, se decidió trabajar con un regresor logístico (LogReg), árboles de decisión (RanFor) y una red neuronal recurrente LSTM, la cual está específicamente diseñada para analizar series temporales, y a la que se aplicaron dos estrategias de descenso por el gradiente diferentes: uno aleatorio (SGD) y otro adaptativo (ADAM). Se abordaron aspectos como la selección de hiperparámetros que optimicen a los estimadores y criterios para verificar su desempeño. Además se corrigió el desequilibrio entre clases en el conjunto de entrenamiento, una problemática que interfiere en el proceso de clasificación debido al reducido número de muestras en la clase minoritaria.

Todos los modelos obtuvieron un elevado rendimiento, destacando a Random Forest como el modelo más eficiente, seguido de la red LSTM (Adam) con una eficacia algo menor. En tercer lugar se posicionó la red LSTM (SGD) y por último la regresión logística, como cabía esperar debido a su menor complejidad en el diseño.

Estos notables resultados pueden atribuirse, en parte, a que la base de datos utilizada está compuesta por empresas que proporcionaron datos durante todos los meses de estudio. En consecuencia, es razonable suponer que estas sean más confiables en comparación con aquellas que podrían haber quebrado, cerrado o motivos similares y que, por tanto, no fueron consideradas en el análisis. Otro de los motivos puede ser la técnica de transferencia de aprendizaje empleada, donde cada camada se inicializa con información derivada de las camadas precedentes, lo cual facilita y agiliza el proceso de aprendizaje.

Asimismo, es curioso el hecho de que ambas redes presenten mejores resultados con la función de perdida MSE, que suele utilizarse en regresión, siendo que la BCE está concebida precisamente para problemas como el de este trabajo.

En base a estas observaciones, futuros trabajos podrían centrarse en camadas variables,

donde el tamaño de cada una fluctúe conforme a la cantidad de empresas que aportaron datos en ese periodo de tiempo particular. Esto permitiría incluir a todas las compañías presentes en la base, incrementando el volumen de datos y aproximando el problema a un escenario más realista. Otras posibles mejoras consisten en aumentar el número de características agregando nuevas bases, ajustar variables a la inflación, hacer una revisión del tamaño óptimo de las ventanas o introducir un conjunto de testeo con muestras adicionales y desconocidas para los modelos. Por último, se podría profundizar en el estudio de los hiperparámetros, en especial los referidos a las LSTM, puesto que su inspección se realizó de manera manual.

A

Búsqueda de hiperparámetros y pruebas alternativas

En la tabla **A-1** se exhiben los conjuntos de datos utilizados para la búsqueda en grilla y la búsqueda aleatoria respectivamente. Seguidamente se exponen las pruebas adicionales de las redes: tres para LSTM (SGD) y otras tres para LSTM (Adam), las cuales resultaron de la elección manual de los hiperparámetros y que, debido a un rendimiento inferior respecto a las expuestas en el trabajo, no fueron incluidas. En tales pruebas se presentan los hiperparámetros elegidos, las métricas de sensibilidad, precisión y exactitud obtenidas y el error de entrenamiento y validación del modelo final.

Estimador	Hiperparámetro	Valores
LogReg	`max_iter`	`10000, 100000`
	`solver`	`'liblinear', 'newton-cg'`
	`C`	`100, 10, 1.0, 0.1, 0.01`
RanFor	`n_estimators`	`np.arange(10, 201, 10)`
	`criterion`	`'gini', 'entropy'`
	`max_depth`	`np.arange(5, 31, 5)`
	`min_samples_split`	`np.arange(2, 11)`
	`min_samples_leaf`	`np.arange(1, 5)`

Tabla A-1.: Conjunto de hiperparámetros usado para la búsqueda en grilla y búsqueda aleatoria.

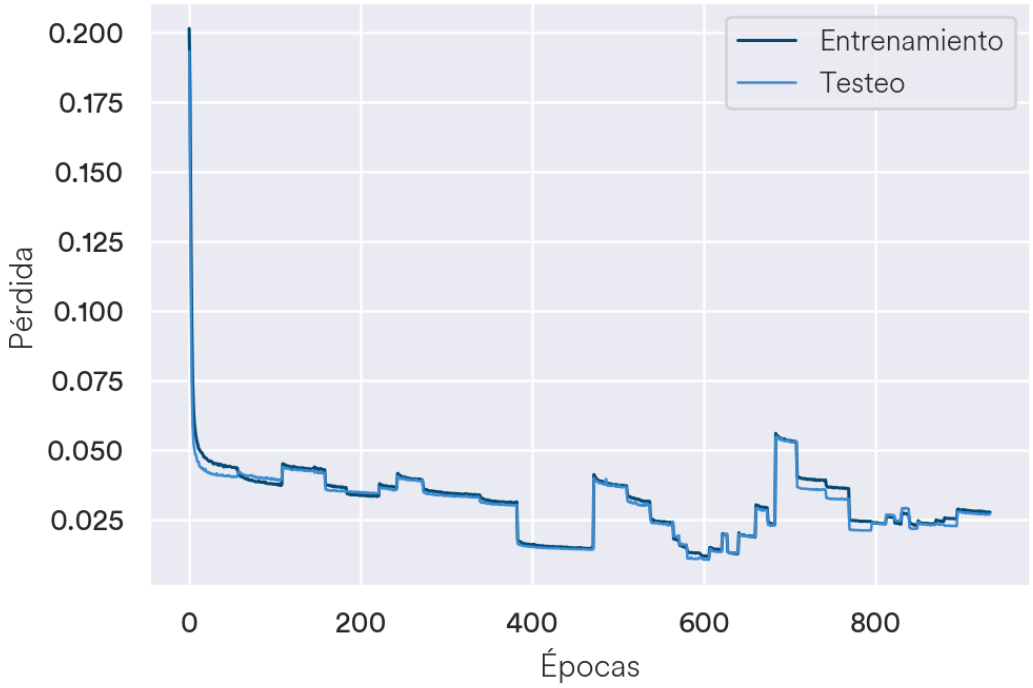


Figura A-1.: Error de entrenamiento y testeo en función de las épocas (de cada camada). En la tabla **A-2** se presentan los hiperparámetros empleados y los correspondientes resultados.

Hiperparámetro	Valor
`optimizer`	`SGD`
`learning_rate`	`0.1`
`loss`	`mean_squared_error`
`units`	`50`
`activation`	`sigmoid`
`batch_size`	`64`
`Dropout`	`0.2`
# capas	2

(a)

Sensibilidad	Precisión	Exactitud
0,940566	0,941887	0,966542

Error entrenamiento	Error testeo
0,027691	0,027158

(b)

Tabla A-2.: (a) Hiperparámetros utilizados en la red y (b) resultados finales obtenidos, referentes a la figura **A-1**.

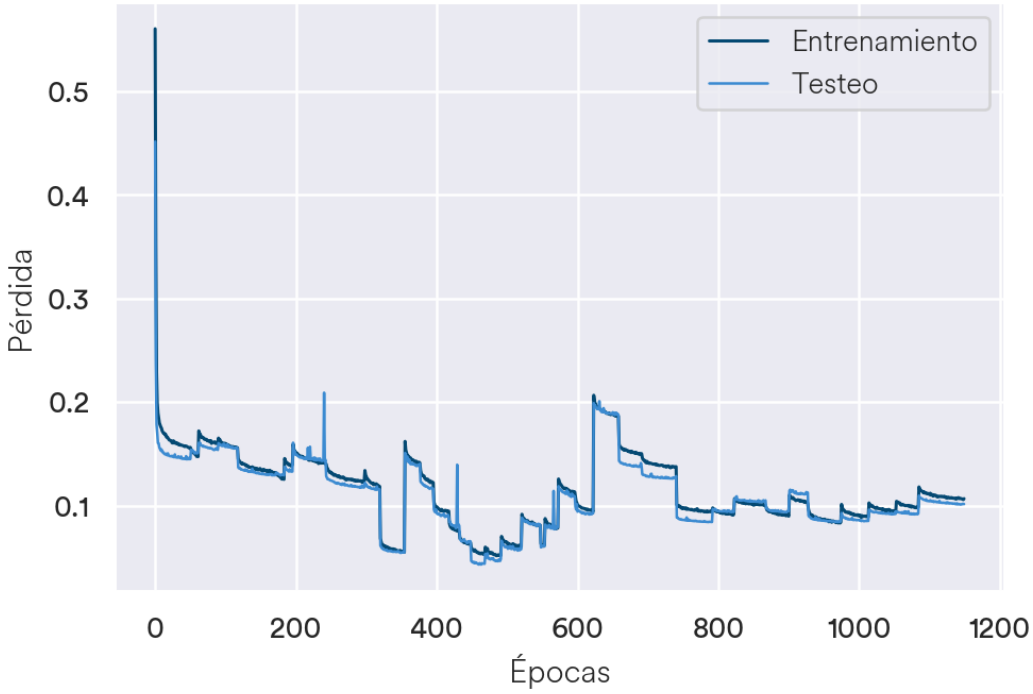


Figura A-2.: Error de entrenamiento y testeo en función de las épocas (de cada camada). En la tabla **A-3** se presentan los hiperparámetros empleados y los correspondientes resultados.

Hiperparámetro	Valor
`optimizer`	`SGD`
`learning_rate`	`0.1`
`loss`	`binary_cross_entropy`
`units`	`50`
`activation`	`sigmoid`
`batch_size`	`64`
`Dropout`	`0.2`
# capas	2

(a)

Sensibilidad	Precisión	Exactitud
0,935800	0,956721	0,969656

Error entrenamiento	Error testeo
0,106710	0,101450

(b)

Tabla A-3.: (a) Hiperparámetros utilizados en la red y (b) resultados finales obtenidos, referentes a la figura **A-2**.

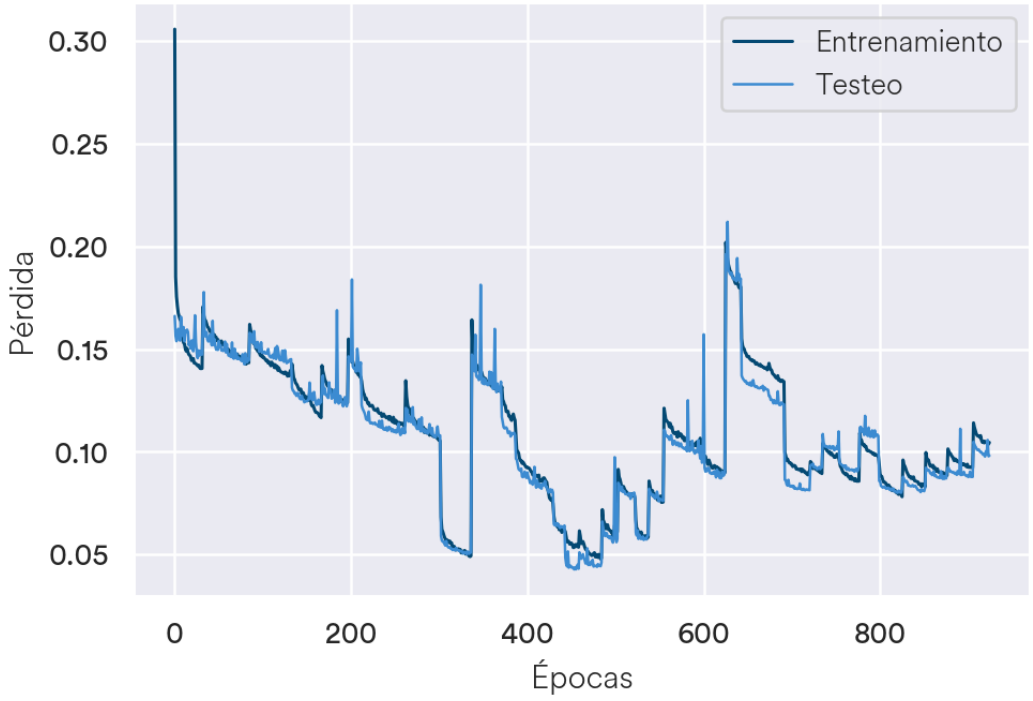


Figura A-3.: Error de entrenamiento y testeo en función de las épocas (de cada camada). En la tabla **A-4** se presentan los hiperparámetros empleados y los correspondientes resultados.

Hiperparámetro	Valor
`optimizer`	`SGD`
`learning_rate`	`0.1`
`loss`	`binary_cross_entropy`
`units`	`50`
`activation`	`sigmoid`
`batch_size`	`64`
`Dropout`	`0.2`
# capas	2

(a)

Sensibilidad	Precisión	Exactitud
0,939725	0,954170	0,969975

Error entrenamiento	Error testeo
0,104584	0,098151

(b)

Tabla A-4.: (a) Hiperparámetros utilizados en la red y (b) resultados finales obtenidos, referentes a la figura **A-3**.

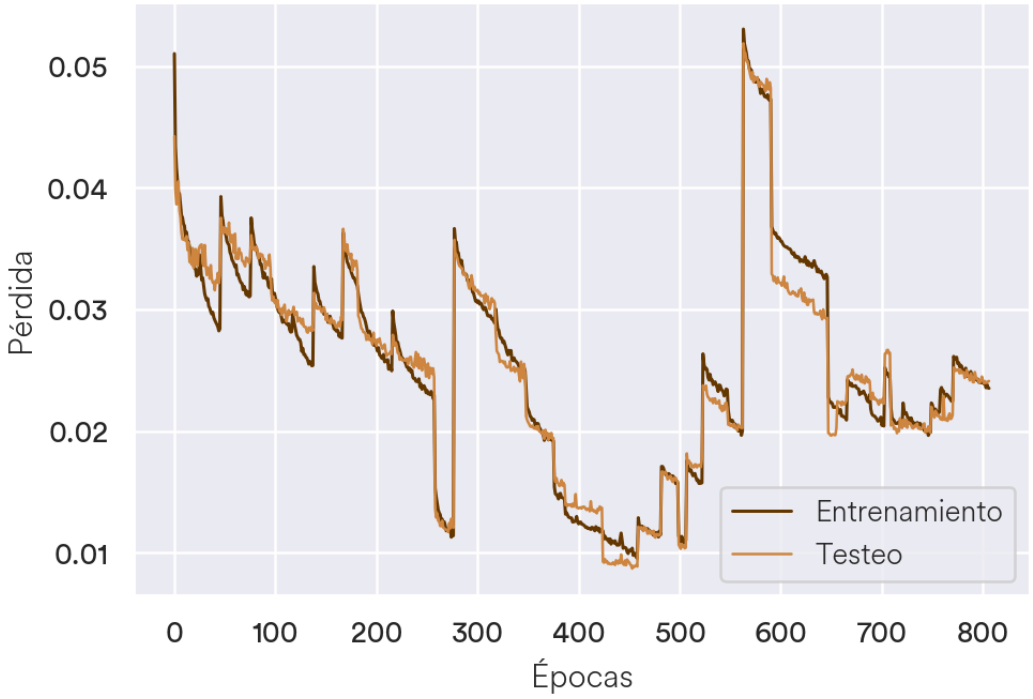


Figura A-4.: Error de entrenamiento y testeo en función de las épocas (de cada camada). En la tabla **A-5** se presentan los hiperparámetros empleados y los correspondientes resultados.

Hiperparámetro	Valor
`optimizer`	`Adam`
`learning_rate`	`0.01`
`loss`	`mean_squared_error`
`units`	`50`
`activation`	`sigmoid`
`batch_size`	`64`
`Dropout`	`0.2`
# capas	2

(a)

Sensibilidad	Precisión	Exactitud
0,945613	0,954443	0,971652

Error entrenamiento	Error testeo
0,023524	0,024119

(b)

Tabla A-5.: (a) Hiperparámetros utilizados en la red y (b) resultados finales obtenidos, referentes a la figura **A-4**.

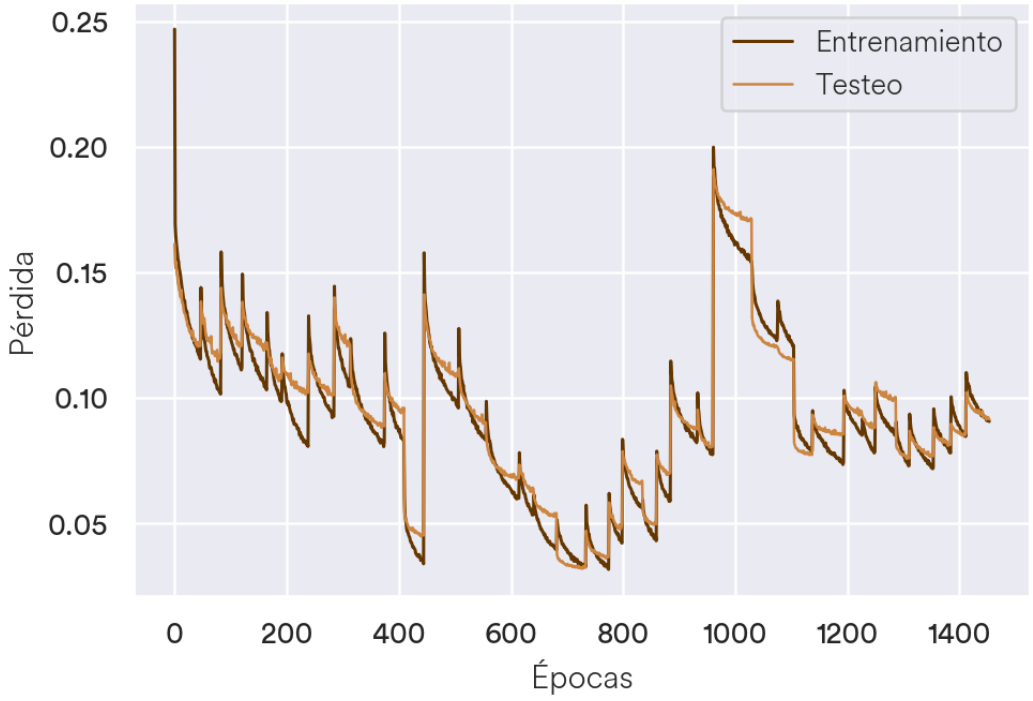


Figura A-5.: Error de entrenamiento y testeo en función de las épocas (de cada camada). En la tabla **A-6** se presentan los hiperparámetros empleados y los correspondientes resultados.

Hiperparámetro	Valor
`optimizer`	`Adam`
`learning_rate`	`0.001`
`loss`	`binary_cross_entropy`
`units`	`50`
`activation`	`sigmoid`
`batch_size`	`64`
`Dropout`	`0.2`
# capas	2

(a)

Sensibilidad	Precisión	Exactitud
0,943650	0,957066	0,971892

Error entrenamiento	Error testeo
0,090650	0,091523

(b)

Tabla A-6.: (a) Hiperparámetros utilizados en la red y (b) resultados finales obtenidos, referentes a la figura **A-5**.

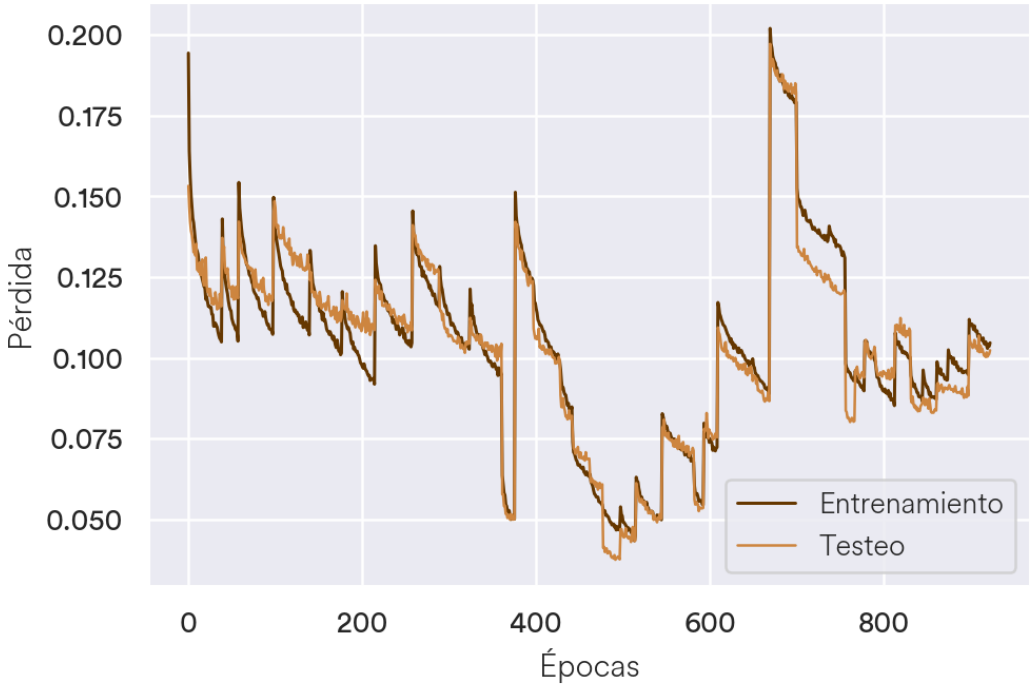


Figura A-6.: Error de entrenamiento y testeo en función de las épocas (de cada camada). En la tabla **A-7** se presentan los hiperparámetros empleados y los correspondientes resultados.

Hiperparámetro	Valor
`optimizer`	`Adam`
`learning_rate`	`0.01`
`loss`	`binary_cross_entropy`
`units`	`50`
`activation`	`sigmoid`
`batch_size`	`64`
`Dropout`	`0.2`
# capas	2

(a)

Sensibilidad	Precisión	Exactitud
0,941688	0,956163	0,971093

Error entrenamiento	Error testeo
0,104699	0,102392

(b)

Tabla A-7.: (a) Hiperparámetros utilizados en la red y (b) resultados finales obtenidos, referentes a la figura **A-6**.

B

Tabla de errores de las redes LSTM

Las tablas **B-1** y **B-2** muestran la pérdida de entrenamiento y testeo de las redes LSTM implementadas en el trabajo.

Camada	Entrenamiento	Testeo
1	0,042507	0,040639
2	0,037400	0,039358
3	0,041279	0,041677
4	0,040700	0,040666
5	0,033929	0,033166
6	0,030875	0,032682
7	0,034112	0,034441
8	0,036855	0,037062
9	0,031447	0,030745
10	0,029559	0,029051
11	0,014212	0,013705
12	0,035909	0,035760
13	0,030101	0,028410
14	0,023556	0,022389
15	0,017168	0,018084
16	0,015059	0,015437
17	0,013248	0,011045
18	0,011919	0,010809
19	0,014812	0,013805
20	0,018796	0,018795
21	0,012546	0,012619
22	0,018108	0,018307
23	0,026367	0,025457
24	0,021783	0,021257
25	0,050560	0,052440
26	0,037208	0,034367
27	0,034416	0,031031
28	0,022935	0,020339
29	0,022064	0,022737
30	0,024363	0,025397
31	0,021516	0,023051
32	0,025374	0,027696
33	0,021135	0,020287
34	0,021533	0,020873
35	0,022498	0,022327
36	0,024075	0,021830
37	0,025804	0,025072

Tabla B-1.: Pérdida de entrenamiento y testeo al final de cada camada en LSTM (SGD).

Camada	Entrenamiento	Testeo	Camada	Entrenamiento	Testeo
1	0,031717	0,033712	20	0,013509	0,015088
2	0,025038	0,030315	21	0,009552	0,010077
3	0,029092	0,032666	22	0,014463	0,016441
4	0,027291	0,031136	23	0,020096	0,020431
5	0,024850	0,027807	24	0,017255	0,018660
6	0,020073	0,025150	25	0,044293	0,046344
7	0,023333	0,026049	26	0,031473	0,031110
8	0,028160	0,032014	27	0,029648	0,028545
9	0,021096	0,022990	28	0,020076	0,018735
10	0,021582	0,024325	29	0,019276	0,021448
11	0,009262	0,011471	30	0,020219	0,022609
12	0,024844	0,029442	31	0,018174	0,021390
13	0,023428	0,024016	32	0,022330	0,025493
14	0,015989	0,018538	33	0,017151	0,018457
15	0,012026	0,014707	34	0,016874	0,017899
16	0,010677	0,013140	35	0,019370	0,019720
17	0,008761	0,008358	36	0,020062	0,019504
18	0,008553	0,008527	37	0,021587	0,022318
19	0,010011	0,011099			

Tabla B-2.: Pérdida de entrenamiento y testeo al final de cada camada en LSTM (Adam).

Bibliografía

[1] Ghosh, K., Time Series Analysis: A Brief History and Its Future Challenges, *Indian Science Cruiser* **34**, 5, 22-27 (2020), DOI: 10.24906/isc/2020/v34/i5/206994.

[2] Shi, J. ; Jain, M. ; Narasimhan, G., Time Series Forecasting Using Various Deep Learning Models, *International Journal of Computer and Systems Engineering* **16**, 6, 224-232 (2022). Disponible en: https://publications.waset.org/abstracts/146879/time-series-forecasting-tsf-using-various-deep-learning-models.

[3] Breiman, L., Random Forests, *Machine Learning* **45**, 5-32 (2001), DOI: 10.1023/A:1010950718922.

[4] McCulloch, W. S. ; Pitts, W., A logical calculus of the ideas immanent in nervous activity, *Bulletin of Mathematical Biophysics* **5**, 4, 115-133 (1943), DOI: 10.1007/BF02478259.

[5] Rosenblatt, F., The Perceptron - A perceiving and recognizing automaton, Technical Report 85-460-1, Cornell Aeronautical Laboratory, 1957. Disponible en: https://link.springer.com/article/10.1007/BF02478259.

[6] Minsky, M. ; Papert, S. A., *Perceptrons: An Introduction to Computational Geometry*, The MIT Press, 1st edition, 1969, ISBN: 9780262630221.

[7] Kingma, D. P. ; Ba, J. L., Adam: a Method for Stochastic Optimization, *arXiv preprint arXiv:1412.6980* (2014), DOI: 10.48550/arXiv.1412.6980. Disponible en: https://arxiv.org/pdf/1412.6980.pdf%5D.

[8] Werbos, P., Backpropagation through time: what it does and how to do it, *Proceedings of the IEEE* **78**, 10, 1550-1560 (1990), DOI: 10.1109/5.58337.

[9] Glorot, X. ; Bengio, Y., Understanding the difficulty of training deep feed forward neural networks, *International Conference on Artificial Intelligence and Statistics* , 249-256 (2010). Disponible en: https://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf.

[10] Hochreiter, S. ; Schmidhuber, J., Long Short-Term Memory, *Neural computation* **9**, 8, 1735-1780 (1997), ISSN: 0899–7667, DOI: 10.1162/neco.1997.9.8.1735.

[11] Chunduri, Understanding LSTM Internal blocks and Intuition. Disponible en: https://medium.com/@chunduri11/understanding-lstm-plain-and-simple-96026b4468c6, 2020.

[12] Starmer, J., Long Short-Term Memory (LSTM), Clearly Explained. Disponible en: https://www.youtube.com/watch?v=YCzL96nL7j0, 2022.

[13] Kohavi, R. ; Provost, F., Glossary of terms, *Machine Learning* **30**, 271-274 (1998), DOI: 10.1023/A:1017181826899.

[14] Schumpeter, J. A., *The Theory of Economic Development; An Inquiry Into Profits, Capital, Credit, Interest, and the Business Cycle. Translated From the German by Redvers Opie*, Harvard University Press, 1st edition, 1949.

[15] Butler, I. ; Guiñazú, S. ; Giuliodori, D. ; Martinez-Correa, J. ; Rodriguez, A. ; Tacsir, E., Programas de Financiamiento Productivo a PYMES, acceso al crédito y desempeño de las firmas: Evidencia de Argentina, (2017). Disponible en: https://scioteca.caf.com/handle/123456789/1151.

[16] Bueso-Merriam, J. ; Demichelis, F. ; Díez, M. C. F. ; Giuliodori, D. ; Rodríguez, A. ; Stucchi, R., El impacto del Programa de Crédito para el Desarrollo de la Producción y el Empleo en la Provincia de San Juan, *Banco Interamericano de Desarrollo* (2016). Disponible en: https://publications.iadb.org/es/publications/spanish/viewer/El-impacto-del-Programa-de-Cr%C3%A9dito-para-el-Desarrollo-de-la-Producci%C3%B3n-y-el-Empleo-en-la-Provincia-de-San-Juan.pdf.

[17] Gorman, L., Lenders Treat Large and Small Firms Differently. Disponible en: https://www.nber.org/digest/202102/lenders-treat-large-and-small-firms-differently, 2021.

[18] Stiglitz, J. ; Weiss, A., Credit Rationing in Markets With Imperfect Information, *The American Economic Review* **71**, 3, 393-410 (1981), ISSN: 0002–8282. Disponible en: https://www.jstor.org/stable/1802787.

[19] Bebczuk, R., Acceso al financiamiento de las PYMES en Argentina: Estado de situación y propuestas de política, Technical Report 0104, CEDLAS, Universidad Nacional de La Plata, 2010.

[20] Ibarrarán, P. ; Maffioli, A. ; Stucchi, R., *Grandes interrogantes sobre pequeñas empresas* 245–264, Banco Interamericano de Desarrollo, 2010. Disponible en: https://publications.iadb.org/es/publications/spanish/viewer/La-era-de-la-productividad-C%C3%B3mo-transformar-las-econom%C3%ADas-desde-sus-cimientos.pdf.

[21] Goldstein, E., El crédito a las PYMES en la Argentina: evolución reciente y estudio de un caso innovador, Technical Report 3895, Naciones Unidas Comisión Económica para América Latina y el Caribe (CEPAL), 2011. Disponible en: https://repositorio.cepal.org/server/api/core/bitstreams/01353d08-431c-471c-be97-87f657bb8f94/content.

[22] Administración Federal de Ingresos Públicos, AFIP — Portal principal. Disponible en: https://www.afip.gob.ar/landing/default.asp.

[23] Banco Central de la República Argentina, BCRA — Portal principal. Disponible en: https://www.bcra.gob.ar/.

[24] Chan, F. ; Mátyás, L., *Econometrics with Machine Learning*, Springer, 1st edition, 2022, ISBN: 978–3–031–15148–4, DOI: 10.1007/978-3-031-15149-1.

[25] Chawla, N. V. ; Bowyer, K. W. ; Hall, L. O. ; Kegelmeyer, W. P., SMOTE: Synthetic Minority Over-sampling Technique, *Journal of Artificial Intelligence Research* **16**, 321–357 (2002), ISSN: 1076–9757, DOI: 10.1613/jair.953.

[26] Spositto, O. M. ; Blanco, G. E. ; Matteo, L. ; Levi, M. ; Bossero, J., SMOTE, algoritmo para balanceo de clases en un estudio aplicado a la ganadería, in *XXVI Congreso Argentino de Ciencias de la Computación (CACIC)* 289–298, Universidad Nacional de La Matanza, 2020. Disponible en: https://sedici.unlp.edu.ar/bitstream/handle/10915/114339/Documento_completo.pdf-PDFA.pdf?sequence=1&isAllowed=y.

[27] Hosna, A. ; Merry, E. ; Gyalmo, J. ; Alom, Z. ; Aung, Z. ; Azim, M., Transfer learning: a friendly introduction, *Journal of Big Data* **9**, 1, 102 (2022), ISSN: 2196–1115, DOI: 10.1186/s40537-022-00652-w.

[28] Zhuang, F. ; Qi, Z. ; Duan, K. ; Xi, D. ; Zhu, Y. ; Zhu, H. ; Xiong, H. ; He, Q., A Comprehensive Survey on Transfer Learning, *Proceedings of the IEEE* **109**, 1, 43-76 (2021), DOI: 10.1109/JPROC.2020.3004555.