



Facultad de Ciencias Económicas y Empresariales

ICADE

COMPARACIÓN ENTRE ALGORITMOS CLÁSICOS DE MACHINE LEARNING Y REDES NEURONALES EN PROBLEMAS DE CLASIFICACIÓN: RENDIMIENTO, INTERPRETABILIDAD Y COSTE COMPUTACIONAL

Autor: Guillermo González García

Director: Carlos Martínez de Ibarreta Zorita

MADRID | Junio 2026

Resumen

Este trabajo compara el rendimiento entre diferentes modelos clásicos de *Machine Learning* y redes neuronales multicapa (*Deep Learning*) en diferentes conjuntos de datos para problemas de clasificación, analizando la capacidad predictiva, la interpretabilidad, el coste computacional y la robustez ante conjuntos de datos heterogéneos. Para ello, los *datasets* escogidos incluyen problemas tabulares, datos simulados, imágenes en formato tabular y un caso aplicado de riesgo de crédito. Los resultados siguen la línea de la literatura existente, demostrando que no existe un modelo universalmente superior, ya que los métodos ensemble, como Random Forest y XGBoost, ofrecen un rendimiento muy competitivo en datos tabulares, mientras que las redes neuronales destacan especialmente en escenarios de mayor dimensionalidad y complejidad. En conjunto, este trabajo evidencia que la elección del algoritmo predictivo debe depender del equilibrio entre precisión, interpretabilidad y eficiencia computacional, especialmente en contextos empresariales donde la justificación de las decisiones resulta fundamental.

Palabras clave: *Machine Learning*, clasificación; redes neuronales, interpretabilidad, predicción y coste computacional.

Abstract

This project compares the performance of different classical Machine Learning models and multilayer neural networks (Deep Learning) across several classification datasets, analysing predictive performance, interpretability, computational cost, and robustness to different data characteristics. For this purpose, the selected datasets include tabular problems, simulated data, images represented in tabular format, and an applied credit risk case study. The results are consistent with the existing literature, showing that there is no universally superior model: ensemble methods, such as Random Forest and XGBoost, offer highly competitive performance on tabular data, while neural networks stand out especially in scenarios with higher dimensionality and complexity. Overall, this work shows that the choice of predictive algorithm should depend on the balance between Accuracy, interpretability, and computational efficiency, particularly in business contexts where the justification of decisions is essential.

Keywords: Machine Learning, classification, neural networks, interpretability, prediction and computational cost.

ÍNDICE

1. INTRODUCCIÓN	8
1.1 CONTEXTO Y MOTIVACIÓN	8
1.2 JUSTIFICACIÓN DEL ESTUDIO	9
1.3 PROBLEMA DE INVESTIGACIÓN Y OBJETIVO GENERAL	9
1.4 METODOLOGÍA E HIPÓTESIS	9
2. MARCO TEÓRICO	11
2.1 APRENDIZAJE SUPERVISADO Y PROBLEMAS DE CLASIFICACIÓN....	11
2.2 ALGORITMOS CLÁSICOS DE MACHINE LEARNING	11
2.2.1 Regresión logística	11
2.2.2 k-Nearest Neighbors (k-NN)	12
2.2.3 Árboles de decisión	12
2.2.4 Random Forest	12
2.2.5 XGBoost	13
2.3 REDES NEURONALES MULTICAPA (MLP).....	14
2.3.1 Funcionamiento de las neuronas	15
2.3.2 Inicialización de pesos.....	16
2.3.3 Funciones de activación	16
2.3.4 Función de coste	17
2.3.5 Cálculo del gradiente	18
2.3.6 Desvanecimiento de gradiente	19
2.3.7 Épocas y proceso de entrenamiento	19
2.3.8 Optimización y descenso por gradiente	20
2.3.9 Bias y varianza en redes neuronales.....	21
2.3.10 Regularización	21
2.3.11 Ajuste de hiperparámetros.....	22
2.4 INTERPRETABILIDAD, COMPLEJIDAD Y COSTE COMPUTACIONAL.	23
3. DISEÑO EXPERIMENTAL	25
3.1 ALGORITMOS SELECCIONADOS	25
3.2 CONJUNTO DE DATOS	26
3.2.1 Dataset simulado no lineal.....	28
3.2.2 Breast Cancer Wisconsin Dataset.....	29
3.2.3 Adult Income Dataset	30
3.2.4 MNIST (versión tabular).....	30
3.2.5 Dataset simulado con ruido.....	31
3.2.6 Credit Default Dataset.....	32
3.3 PREPROCESAMIENTO DE DATOS	33
3.4 VALIDACIÓN DE LOS MODELOS.....	34

3.5	EVALUACIÓN DE LOS MODELOS	34
3.5.1	Métricas de evaluación	34
3.5.2	Tiempo de entrenamiento e inferencia.....	36
3.5.3	Herramientas de visualización.....	37
B)	Matrices de confusión.....	38
3.6	DIAGRAMA GENERAL DE LA METODOLOGÍA.....	39
3.7	REPRODUCIBILIDAD DEL EXPERIMENTO	40
4.	RESULTADOS Y DISCUSIÓN	41
4.1	OPTIMIZACIÓN DE HIPERPARÁMETROS	41
4.1.1	Regresión logística	41
4.1.2	k-Nearest Neighbors	42
4.1.3	Árbol de Decisión	42
4.1.4	Random Forest.....	42
4.1.5	XGBoost.....	42
4.1.6	Redes Neuronales	43
4.2	RESULTADOS GENERALES POR CONJUNTO DE DATOS	45
4.3	DATASET SIMULADO NO LINEAL.....	46
4.4	BREAST CANCER WISCONSIN DATASET.....	47
4.5	ADULT INCOME DATASET	48
4.6	MNIST	49
4.7	DATASET SIMULADO CON RUIDO	51
4.8	CREDIT DEFAULT DATASET.....	53
4.9	APLICACIÓN EMPRESARIAL DEL CASO DE USO.....	54
4.10	COMPARACIÓN GLOBAL ENTRE MODELOS CLÁSICOS Y REDES NEURONALES	55
5.	CONCLUSIONES	57
6.	DECLARACIÓN DE USO DE HERRAMIENTAS DE INTELIGENCIA ARTIFICIAL GENERATIVA EN TRABAJOS FIN DE GRADO.....	58
7.	BIBLIOGRAFÍA	60

ÍNDICE DE TABLAS

Tabla 1. Comparación de modelos clásicos de Machine Learning	13
Tabla 2. Comparación entre modelos clásicos y redes neuronales.....	23
Tabla 3. Características de los conjuntos de datos utilizados.....	27
Tabla 4. Modelo con mejor rendimiento por conjunto de datos.....	45
Tabla 5. Resultados Dataset Simulado no lineal	46
Tabla 6. Resultados Dataset Breast Cancer Wisconsin	47
Tabla 7. Resultados Dataset Adult Income	48
Tabla 8. Resultados Dataset MNIST	49
Tabla 9. Resultados Dataset Simulado con Ruido.....	51
Tabla 10. Resultados Dataset Credit Default.....	53
Tabla 11. Conclusiones finales	56

ÍNDICE DE ILUSTRACIONES

Ilustración 1: Ejemplo de red neuronal.....	15
Ilustración 2. Comparación de las funciones de activación Sigmoid, Tanh y ReLU.....	17
Ilustración 3. Dataset simulado no lineal.....	28
Ilustración 4. Dataset simulado con ruido	32
Ilustración 5. Ejemplo de Curva ROC.....	37
Ilustración 6. Ejemplo de Matriz de Confusión.....	38
Ilustración 7. Diagrama general de la metodología.....	39
Ilustración 8: Optimización en XGBoost	43
Ilustración 9. Optimización en redes neuronales.....	44
Ilustración 10. Comparación de red neuronal con Numpy vs Keras	50

ÍNDICE DE FÓRMULAS

Fórmula 1. Función sigmoide	11
Fórmula 2. Función de activación de la neurona	14
Fórmula 3. Fórmula de combinación lineal de la neurona	15
Fórmula 4. Función sigmoide	16
Fórmula 5. Función Tangente Hiperbólica (tanh)	16
Fórmula 6. Función ReLU	17
Fórmula 7. Función de coste: entropía cruzada	17
Fórmula 8. Función de descenso de gradiente	20
Fórmula 9. Función de coste con regularización L2	21
<i>Fórmula 10. Accuracy</i>	<i>34</i>
<i>Fórmula 11. Precision</i>	<i>35</i>
<i>Fórmula 12. Recall</i>	<i>35</i>
<i>Fórmula 13. F1-Score</i>	<i>35</i>
<i>Fórmula 14. F2-Score</i>	<i>36</i>

1. INTRODUCCIÓN

1.1 CONTEXTO Y MOTIVACIÓN

Durante los últimos años el aprendizaje automático ha experimentado un crecimiento importante tanto en el ámbito empresarial como académico, ya que nuevas técnicas como el Aprendizaje Profundo (*Deep Learning*) han cobrado especial relevancia debido a su capacidad para modelar relaciones complejas y no lineales a partir de grandes volúmenes de datos. Esta evolución en los modelos predictivos ha generado un gran cambio en la forma de abordar los problemas de clasificación, impulsando la adopción de redes neuronales en múltiples sectores. Aunque, los algoritmos tradicionales continúan con un papel clave, en especial en contextos donde la interpretabilidad de variables, el coste computacional o la disponibilidad limitada de datos son factores determinantes.

La elección de comparar los algoritmos tradicionales con el aprendizaje profundo para la realización de este Trabajo de Fin de Grado, surge tras la realización de varios cursos de un programa de *Deep Learning* en el que se abordan diversos temas y características de estos nuevos modelos. Los cursos que he realizado y que han servido de base han sido:

- Redes Neuronales y Aprendizaje Profundo
- Mejora de Redes Neuronales Profundas; Ajuste de Hiperparámetros, Regularización y Optimización
- Estructuración de Proyectos de Aprendizaje Automático.

Estos tres cursos, me han dado una sólida base teórica y metodológica que me ha permitido comprender y demostrar que el *Deep Learning*, ha transformado la manera en la que se abordaban los problemas de clasificación en múltiples sectores, como la banca, la salud o la industria, ya que el desarrollo de modelos capaces de aprender patrones complejos a partir de grandes volúmenes de datos, permite los modelos alcancen niveles de precisión muy elevados a los anteriores, en casos especialmente relevantes como el diagnóstico médico, el reconocimiento de imágenes o la detección de fraude.

Durante el desarrollo de este trabajo, se analizarán diversos *datasets* de diferentes características, para intentar mejorar su precisión, intentándolo con algunos utilizados en la carrera universitaria sobre los que ya he realizado modelos predictivos.

Sin embargo, pese a ser las redes neuronales los modelos con mejores rendimientos en la mayoría de problemas, la falta de interpretabilidad de las variables implicadas en el modelo y su alto coste computacional, hacen que se siga debatiendo los contextos en los que es relevante usar *Deep Learning* frente a algoritmos tradicionales de *Machine Learning*.

1.2 JUSTIFICACIÓN DEL ESTUDIO

En aplicaciones empresariales, especialmente en el ámbito bancario y financiero, la selección del modelo no solo depende de su capacidad predictiva, ya que hay diversos factores como la interpretabilidad, el coste computacional o la robustez de datos ruidosos que desempeñan un papel clave para elegir modelo.

El desarrollo de este trabajo tiene una proyección aplicada, ya que uno de los modelos desarrollados podrá presentarse en el área de *Business Analytics* de banca de NTT Data, la empresa donde estoy desarrollando las prácticas, para aportar de esta forma una dimensión orientada al negocio y transferencia real del conocimiento.

1.3 PROBLEMA DE INVESTIGACIÓN Y OBJETIVO GENERAL

Principalmente, el problema que se aborda en este Trabajo Fin de Grado es el siguiente:

Analizar y comparar el rendimiento de algoritmos clásicos de *Machine Learning* con algoritmos de aprendizaje profundo como redes neuronales en problemas de clasificación, comparando conjuntos de datos con diferentes características para analizar tanto criterios técnicos (rendimiento o capacidad predictiva) como implicaciones prácticas (coste computacional, reproducibilidad o interpretabilidad), todo esto con el fin de extraer conclusiones prácticas sobre su aplicabilidad en contextos reales.

1.4 METODOLOGÍA E HIPÓTESIS

La metodología adoptada aporta un enfoque experimental comparativo, que evalúa el rendimiento de diferentes algoritmos bajo conjuntos de datos con diferentes características:

- Tamaño muestral
- Tipo de variables
- Dimensionalidad
- Presencia de ruido

Esta diversidad permite analizar los modelos en diferentes escenarios.

Los algoritmos seleccionados para comparar son:

- Regresión logística
- *K-Nearest Neighbors*
- Árboles de decisión
- Métodos ensemble
- Redes neuronales multicapa

Todos ellos con diferentes parámetros a optimizar, siendo las redes neuronales el modelo en el que más importancia cobran.

El procedimiento experimental seguido en este trabajo tiene una estructura clara y sistemática que se repite en todos los modelos:

- 1) División en conjuntos de entrenamiento y prueba, incorporando un conjunto de validación adicional en el caso de las redes neuronales y en los procesos de optimización de hiperparámetros cuando fue necesario.
- 2) Ajuste de parámetros mediante técnicas de búsqueda, como *grid search* o *random search*.
- 3) Evaluación mediante métricas como *Accuracy*, *F1-Score* y AUC-ROC.
- 4) Análisis del coste computacional.
- 5) Estudio de la interpretabilidad a través de la importancia de variables.

También se van a analizar el rendimiento de los modelos en función del tamaño del conjunto de datos, con el objetivo de observar el comportamiento de cada modelo ante escenarios de datos limitados.

Este procedimiento busca garantizar rigor estadístico, aplicabilidad práctica y reproducibilidad a partir de los resultados obtenidos.

Este trabajo surge de varias hipótesis principales:

- Las redes neuronales muestran un rendimiento superior especialmente en conjuntos de grandes volúmenes de datos y con relaciones no lineales complejas.
- Los algoritmos clásicos de *Machine Learning* presentan un rendimiento competitivo o incluso superior, en conjuntos de datos estructurados y de menor volumen, con menor coste computacional y mayor interpretabilidad de variables.
- En contextos empresariales, la elección del modelo depende de un equilibrio entre precisión, interpretabilidad y eficiencia computacional.

Este conjunto de hipótesis permite estructurar el análisis y orientar la investigación hacia resultados y conclusiones aplicables a entornos reales.

2. MARCO TEÓRICO

2.1 APRENDIZAJE SUPERVISADO Y PROBLEMAS DE CLASIFICACIÓN

El aprendizaje supervisado es uno de los principales enfoques dentro del *Machine Learning*, donde el objetivo consiste en aprender una función que sea capaz de relacionar un conjunto de variables de entrada con una variable de salida a partir de ejemplos etiquetados, minimizando el error entre las predicciones y los valores reales.

Como señala Ng (DeepLearning.AI), “*el aprendizaje supervisado puede aplicarse a múltiples problemas reales, incluyendo clasificación binaria, reconocimiento de imágenes o detección de fraude, donde input (x) → output (y) representa la relación fundamental que el modelo debe aprender*”.

Dentro del aprendizaje supervisado, aparecen los problemas de clasificación, caracterizados porque la variable de salida u objetivo es categórica, lo que implica que el modelo tenga que asignar una etiqueta a cada observación, estos tipos de problemas son especialmente importantes en ámbitos como el diagnóstico médico o la evaluación de riesgo crediticio.

2.2 ALGORITMOS CLÁSICOS DE MACHINE LEARNING

2.2.1 Regresión logística

La regresión logística es uno de los modelos más simples y utilizados en problemas de clasificación binaria, ya que se trata de un modelo lineal que estima la probabilidad de pertenecer a una clase mediante una función sigmoide, que transforma una combinación lineal de las variables de entradas en un valor comprendido entre 0 y 1.

El objetivo es encontrar los parámetros que minimicen la función de pérdida definida sobre el conjunto de entrenamiento.

Fórmula 1. Función sigmoide

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

, donde: $z = w^T x + b$, y (z, b) son los parámetros a optimizar

Fuente: Hastie, Tibshirani y Friedman (2009)

El entrenamiento de estos modelos consiste en minimizar la función de coste, siendo especialmente eficaz en datos estructurados, con relaciones lineales entre variables de entrada y salida, es por ello que la principal ventaja de este modelado, es la interpretabilidad, convirtiéndolo en una herramienta muy útil en contextos empresariales y financieros.

2.2.2 k-Nearest Neighbors (k-NN)

El algoritmo k-NN es un método basado en instancias que permite clasificar nuevas observaciones en función de las etiquetas de sus k observaciones (vecinos) más cercanas, según una métrica de distancia.

La característica diferencial de este modelo es que no realiza una fase de aprendizaje en el entrenamiento, sino que almacena los datos y realiza la clasificación en la fase de predicción.

Pese que su principal ventaja sea la simplicidad conceptual del modelo, presenta varias limitaciones entre las que destacan:

- Sensibilidad a la escala de las variables y a la dimensionalidad de espacio.
- Falta de rendimiento en dimensiones altas.

2.2.3 Árboles de decisión

Los árboles de decisión son modelos que dividen el espacio de variables en nodos, cada uno de ellos selecciona las variables y el punto de corte que mejor separa las clases, para poder encontrar el más homogéneo respecto a la variable objetivo.

Cada división de nodos se selecciona maximizando la reducción de la impureza, medida mediante el índice de Gini o la entropía.

La principal ventaja de este modelo es la interpretabilidad, ya que las decisiones son representadas de forma estructurada, sin embargo, su alta varianza hace que el modelo se sobreajuste en muchas ocasiones sino se aplican técnicas de regularización, que serán explicadas más adelante.

2.2.4 Random Forest

Random Forest es un método *ensemble* basado en la combinación de múltiples árboles de decisión, propuesto por *Breiman (2001)*. Su funcionamiento se basa en la técnica del *bagging*, que consiste en construir varios modelos independientes a partir de subconjuntos aleatorios, para combinar sus predicciones y reducir la varianza y mejorar el rendimiento.

El Random Forest combina múltiples árboles de decisión entrenados sobre subconjuntos aleatorios de datos de las variables explicativas, obteniendo la variable de salida mediante agregación, lo que permite reducir la varianza y soluciona uno de los principales problemas de los árboles de decisión.

Esta reducción de varianza permite mejorar la capacidad de generalización, siendo aparte una técnica con una alta interpretabilidad que permite estimar la importancia relativa de las variables.

Esta técnica, destaca en conjunto de datos tabulares ofreciendo una alta interpretabilidad de variables.

2.2.5 XGBoost

XGBoost es un algoritmo ensemble desarrollado por *Chen y Guestrin (2016)* y destaca por su elevada capacidad predictiva y eficiencia computacional. Este modelo pertenece a la familia del *boosting*, una técnica de aprendizaje secuencial, que construye modelos a través de árboles que corrigen los errores cometidos por los anteriores, para mejorar así de forma progresiva el rendimiento global del modelo.

El funcionamiento de este algoritmo se basa en optimizar una función objetivo con dos principales componentes:

- Una función de pérdida para medir el error del modelo.
- Un término de regularización, que evita el *overfitting*.

El XGBoost destaca también por su buen rendimiento de conjunto de datos tabulares (con forma de tabla organizados en filas y columnas), sin embargo, ofrece una menor interpretabilidad comparado con la regresión logística o los árboles de decisión.

2.2.6 Comparación de algoritmos clásicos de Machine Learning

Tabla 1. Comparación de modelos clásicos de Machine Learning

Modelo	Tipo de modelo	Ventajas	Desventajas	Interpretabilidad
Regresión Logística	Lineal	Simple e interpretable	Limitada a relaciones lineales	Alta
k-NN	Basado en instancias	Fácil de implementar	Coste alto y sensible a escala	Media
Árbol de decisión	No lineal	Muy interpretable	Alta varianza, sobreajuste.	Muy alta
Random Forest	<i>Ensemble (bagging)</i>	Reduce varianza	Menos interpretabilidad	Media
XGBoost	<i>Ensemble (boosting)</i>	Datos tabulares	Mayor complejidad	Baja-media

Fuente: Elaboración propia basada en la revisión bibliográfica realizada.

2.3 REDES NEURONALES MULTICAPA (MLP)

Las redes neuronales artificiales son modelos inspirados en el funcionamiento del cerebro humano, formados por capas de neuronas conectadas entre sí que permiten aprender relaciones complejas a partir de los datos. Según *Goodfellow et al. (2016)*, constituyen la base de los sistemas modernos de *Deep Learning* debido a su capacidad para aprender representaciones jerárquicas de la información.

Estos modelos multicapa son una extensión no lineal de los modelos lineales tradicionales, permitiendo de esta forma explicar relaciones complejas entre variables de entrada y salida. Las redes neuronales siguen una estructura organizada en capas:

- Capa de entrada: recibe los datos. El número de neuronas de esta capa depende del número de variables del conjunto de datos.
- Una o varias capas ocultas: procesamiento intermedio
- Una capa de salida: genera la predicción final.

Dentro de cada capa aparecen las neuronas (representadas por las bolas en la Ilustración 1), estas son las unidades básicas de procesamiento, cada una de ellas realiza una transformación de la información recibida a través de dos pasos:

- 1) Combinación lineal de las entradas (función z)
- 2) Función de activación no lineal (función a)

Fórmula 2. Función de activación de la neurona

$$a^{[l]} = g(z^{[l]})$$

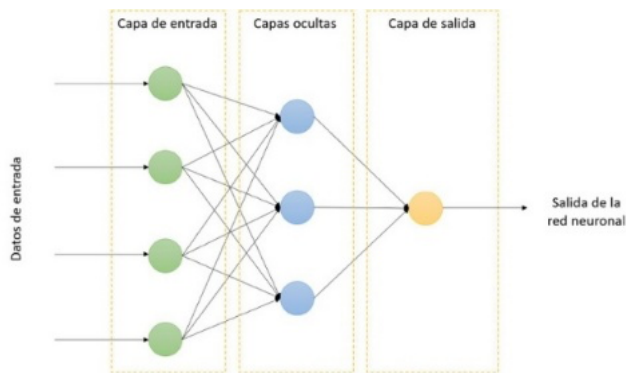
Fuente: Goodfellow, Bengio y Courville (2016)

En las redes neuronales es importante diferenciar entre estos dos conceptos claves:

- Neurona: unidad individual de procesamiento que recibe la combinación de entradas, y a través de una ponderación de pesos genera una salida transformada mediante la función de activación.
- Capa: conjunto de neuronas que operan conjuntamente y cuyas salidas sirven como entrada para la siguiente capa.

La capacidad predictiva del modelo depende tanto del número de capas (profundidad de la red) como del número de neuronas (anchura de cada capa), ya que conceptualmente, una red neuronal sin capas ocultas sería equivalente a un modelo predictivo lineal, siendo las capas ocultas y el número de neuronas lo que convierte las redes en herramientas complejas capaces de capturar relaciones no lineales, aunque aparentemente las redes neuronales parecen modelos perfectos, su coste computacional y su falta de interpretabilidad dificultan su aplicación en muchos problemas.

Ilustración 1: Ejemplo de red neuronal



Fuente: Interactive Chaos. (s.f.)

2.3.1 Funcionamiento de las neuronas

Estos sistemas se basan en el procesamiento realizado por cada neurona, como se puede observar en la Ilustración 1:

- 1) Recibe información de la capa anterior.
- 2) Realiza una transformación sobre la entrada recibida.
- 3) Genera una salida que servirá como entrada para la capa posterior.

Las diferentes transformaciones que realizan a través de las capas es lo que permite explicar relaciones complejas.

En un primer lugar, antes de la función de activación definida previamente, la neurona realiza una combinación lineal de las variables de entrada, a través de pesos que determinan la importancia de cada variable, y de un término independiente que ajusta la salida independientemente de las entradas, al igual que pasa en la regresión logística.

Fórmula 3. Fórmula de combinación lineal de la neurona

$$z^{[l]} = W^{[l]}a^{[l-1]} + b^{[l]}$$

Fuente: Goodfellow, Bengio y Courville (2016)

donde:

- $W^{[l]}$ representa la matriz de pesos de la capa.
- $a^{[l-1]}$ representa las salidas de la capa anterior.
- $b^{[l]}$ representa el sesgo (*bias*).
- $z^{[l]}$ representa el resultado de la transformación lineal.

Después de la transformación lineal, se introduce una función de activación, que introduce no linealidad en el modelo y permite aprender relaciones complejas entre variables y modelar patrones que para los modelos lineales tradicionales era imposible. La salida obtenida, es la que usan las neuronas de la siguiente capa como entrada, haciendo que la información progrese desde la capa de entrada hasta la de salida.

2.3.2 Inicialización de pesos

Antes de iniciar la fase de entrenamiento, los pesos de la red neuronal deben inicializarse con los valores adecuados, ya que una mala inicialización puede ralentizar el aprendizaje o provocar problemas en la propagación de información entre capas.

En redes neuronales que utilizan la función de activación ReLU, una de las estrategias más habituales es la inicialización He, que ajusta la escala inicial de los pesos para mantener estable la varianza de las activaciones entre capas.

2.3.3 Funciones de activación

Las funciones de activación son uno de los elementos fundamentales de las redes neuronales, ya que son las encargadas de introducir no linealidad en el modelo, entre las más utilizadas destacan las siguientes:

Fórmula 4. Función sigmoide

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

Fuente: Goodfellow, Bengio y Courville (2016)

Transforma las variables de entrada en probabilidades entre 0 y 1, por lo que se usa principalmente en problemas de clasificación binaria y para la capa de salida, ya que se interpreta como una probabilidad. Sin embargo, aquí aparece el problema del desvanecimiento de gradiente, que hace que el aprendizaje del modelo se ralentice.

Fórmula 5. Función Tangente Hiperbólica (tanh)

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

Fuente: Goodfellow, Bengio y Courville (2016)

Similar a la sigmoide, la tangente transforma los valores entre -1 y 1, generando salidas centradas en cero, lo que permite una convergencia más estable en el entrenamiento, aunque esta función, también se ve afectada por el desvanecimiento de gradiente.

Al generar salidas centradas en cero, suele favorecer una convergencia más estable durante el entrenamiento. No obstante, también puede verse afectada por el problema del desvanecimiento del gradiente.

Fórmula 6. Función ReLU

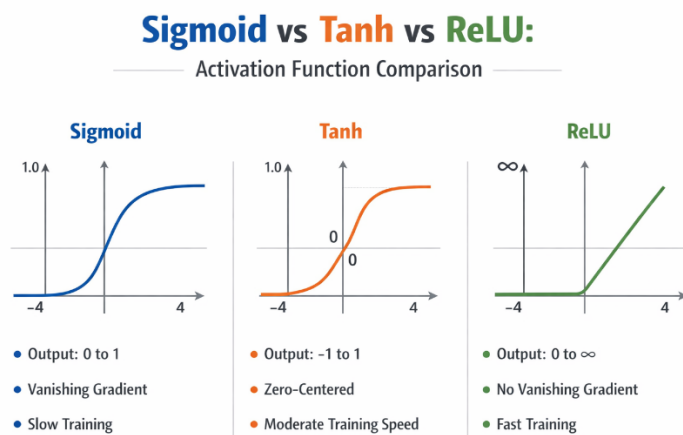
$$g(z) = \max(0, z)$$

Fuente: Goodfellow, Bengio y Courville (2016)

Es la función más utilizada en capas ocultas, debido a su simplicidad y eficiencia, ya que transforma los valores negativos en 0, y los positivos los mantiene en su valor original, reduciendo el problema del descenso del gradiente y acelerando el entrenamiento.

La elección de las distintas funciones de activación impacta de una manera directa en la capacidad de aprendizaje, la velocidad de convergencia y el rendimiento final del modelo.

Ilustración 2. Comparación de las funciones de activación Sigmoid, Tanh y ReLU.



Fuente: Aitude (2026).

2.3.4 Función de coste

Esta función es la herramienta utilizada para medir el error cometido por la red neuronal durante el entrenamiento, para poder cuantificar la diferencia entre las predicciones generadas y los valores reales del conjunto de datos.

A través del entrenamiento, la red neuronal ajusta sus parámetros intentando minimizar dicha función, ya que a menor valor de la función mejor será su capacidad predictiva.

En problemas de clasificación binaria, como los que se van a estudiar en este trabajo, la función más utilizada es la entropía cruzada, ya que penaliza las predicciones incorrectas que tienen un alto grado de confianza.

Fórmula 7. Función de coste: entropía cruzada

$$J = -\frac{1}{m} \sum_{i=1}^m [y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})]$$

Fuente: Goodfellow, Bengio y Courville (2016)

Al igual que sucedía con la función de activación, la elección de la función de coste depende en su mayoría del tipo de problema, ya que es la encargada de guiar el proceso de aprendizaje en las redes neuronales. Por ejemplo, para problemas de clasificación multiclase, la función de coste es la *softmax* combinada con entropía cruzada, ya que permite estimar probabilidades para múltiples clases.

2.3.5 Cálculo del gradiente

El gradiente indica la variación de la función de coste respecto a los parámetros (pesos y término independiente), representando cómo deben de modificarse para reducir el error del modelo en la fase de entrenamiento.

El cálculo de gradiente se realiza mediante un algoritmo, *backpropagation*, que es el encargado de permitir el aprendizaje en redes neuronales. Este algoritmo se divide a su vez en dos etapas:

- **Propagación hacia delante (*forward propagation*)**

En esta etapa la información fluye desde la capa de entrada a la de salida, donde cada neurona recibe las salidas de la capa anterior, las transforma y genera una salida aplicando la función de activación correspondiente.

A través del paso por las distintas capas, se obtiene una salida con la que se calcula el error del modelo comparándolo con el valor real esperado.

Este proceso permite transformar progresivamente los datos de entrada hasta obtener una predicción final, siendo uno de los mecanismos fundamentales del aprendizaje en redes neuronales (*Goodfellow et al., 2016*).

- **Propagación hacia detrás (*backpropagation*)**

Una vez calculado el error, comienza la segunda etapa, en la que el error fluye desde la capa de salida hasta la de entrada, con el objetivo de calcular el peso que tiene cada parámetro en el error final.

Es a través de esta segunda etapa, donde se obtienen los gradientes de la función de coste respecto a cada parámetro, todo ello calculado mediante derivadas parciales. Estos gradientes son lo que usarán los algoritmos de optimización para actualizar los parámetros y reducir progresivamente el error del modelo.

2.3.6 Desvanecimiento de gradiente

Uno de los principales problemas que surgen en la fase de entrenamiento, es el del desvanecimiento de gradiente o *vanishing gradient problema*. Este problema surge cuando los gradientes se vuelven muy pequeños en las primeras capas de la red durante la *backpropagation*.

Como resultado a este problema:

- Los pesos apenas se actualizan
- El aprendizaje se ralentiza
- Algunas capas dejan de aprender.

Como se explicó anteriormente, la función ReLU reduce significativamente este problema, es por esto que es una de las más utilizadas en las redes neuronales.

2.3.7 Épocas y proceso de entrenamiento

El entrenamiento en una red neuronal es un proceso iterativo a través del que el modelo intenta ajustar continuamente los parámetros para reducir el error de predicción.

Durante la etapa de entrenamiento, los datos se suelen dividir en pequeños lotes (*mini-batches*), esta división permite actualizar parámetros de forma más eficiente que haciéndolo sobre todo el conjunto de datos.

Dado que el proceso de entrenamiento es iterativo, se define como épocas la iteración completa sobre todo el conjunto de entrenamiento, por lo que en toda la fase de entrenamiento se realizan múltiples épocas, ya que solo una pasada es insuficiente para que el modelo aprenda los patrones ocultos en los datos.

Siendo la elección del número de épocas algo fundamental, que influye directamente en el aprendizaje del modelo, permitiendo ajustar el modelo en función de la complejidad del conjunto de datos.

Para evitar que el proceso de entrenamiento sea excesivo surge una herramienta conocida como *Early Stopping*, que detiene de forma automática el entrenamiento cuando el error del conjunto de validación deja de mejorar durante varias épocas consecutivas. Esta herramienta permite reducir el riesgo de sobreajuste, ya que conserva la mejor configuración sobre datos no vistos.

2.3.8 Optimización y descenso por gradiente

El entrenamiento de las redes neuronales se realiza a través de algoritmos de optimización que se basan en descenso por gradiente (*gradient descent*), cuyo objetivo es actualizar los parámetros iterativamente para conseguir minimizar la función de coste.

En cada iteración de este algoritmo, los pesos se actualizan en dirección opuesta al gradiente de la función de coste (es decir en la dirección que la función de coste decrece), para reducir el error cometido en las predicciones y mejorar la capacidad de aprendizaje progresivamente.

Fórmula 8. Función de descenso de gradiente

$$w := w - \alpha \frac{\partial J}{\partial w}$$

Fuente: Goodfellow, Bengio y Courville (2016)

donde:

- w representa los pesos,
- α la tasa de aprendizaje (*learning rate*),
- $y \frac{\partial J}{\partial w}$ el gradiente de la función de coste.

Entre los algoritmos de optimización más utilizados en las redes neuronales, aparecen:

- Mini-batch Gradient Descent: usa subconjuntos de datos en cada iteración, lo que simplifica la eficiencia computacional
- Momentum: cada iteración incorpora información de las iteraciones anteriores para poder acelerar la convergencia hacia el mínimo de la función de coste.
- RMSprop: ajusta en cada iteración la tasa de aprendizaje automáticamente en función de la variación de la función de coste (gradiente)
- Adam: combina dos algoritmos, *Momentum* y *RMSprop*, siendo uno de los más utilizados por su estabilidad y rapidez de convergencia.
- Reducción adaptativa de la tasa de aprendizaje: disminuye el valor de la tasa de aprendizaje cuando la función de coste deja de mejorar, permitiendo realizar más ajustes en las fases finales, lo que favorece una convergencia más estable y directa al mínimo.

Al igual que pasaba con la elección de la función de activación, la selección del algoritmo de optimización influye en la velocidad de entrenamiento y el rendimiento final.

2.3.9 Bias y varianza en redes neuronales

Para poder evaluar la capacidad de generalización de las redes neuronales, es muy útil analizar la varianza y el sesgo del modelo.

- **Alto sesgo:** indica que el modelo es demasiado simple y no captura la estructura de datos, lo que produce problemas de subajuste (*underfitting*)
- **Alta varianza:** indica que el modelo se ajusta en exceso a los datos del entrenamiento, produciendo problemas de sobreajuste (*overfitting*)

Poder conocer si el error en nuestro modelo viene por un subajuste o sobreajuste a los datos de entrenamiento, aporta la capacidad de ajustar de una manera más sencilla la complejidad de la red neuronal.

2.3.10 Regularización

Para poder resolver problemas de subajuste, solo sirve con hacer la red neuronal algo más complejas, ya sea aumentando el número de neuronas o el número de capas ocultas; pero los problemas de sobreajuste, requiere el uso de técnicas de regularización que permiten mejorar la capacidad de generalización del modelo.

Entre las técnicas más utilizadas, destacan las siguientes:

- **Regularización L2**

Esta técnica penaliza los pesos de gran magnitud, es decir, aquellos que hacen que el modelo dependa en exceso de alguna variable para realizar las predicciones. Esta penalización permite una distribución más equilibrada en los pesos a través de un término adicional en la función de coste, lo que favorece modelos más simples que reducen el riesgo de sobreajuste.

Fórmula 9. Función de coste con regularización L2

$$J(w) = Loss + \frac{\lambda}{2m} ||w||^2$$

Fuente: Goodfellow, Bengio y Courville (2016)

- **Dropout**

Esta técnica desactiva ciertas neuronas de forma aleatoria durante la fase de entrenamiento, permitiendo así al modelo mejorar su robustez y no depender en exceso de determinadas variables.

Estas técnicas tienen como objetivo reducir el sobreajuste y mejorar la capacidad de generalización de la red neuronal (*Goodfellow et al., 2016*), la L2 lo controla reduciendo la magnitud de los pesos y el *dropout* lo controla añadiendo aleatoriedad; en la construcción de redes neuronales es muy común el uso de ambas técnicas, ya que su combinación mejora ampliamente la generalización del modelo

- **Batch Normalization**

Esta técnica de regularización conocida como normalización por lotes, permite normalizar las salidas de una capa de la red neuronal durante el entrenamiento, para lograr de esta forma un aprendizaje más estable y rápido.

A través del paso de la información por las distintas capas, la distribución de las activaciones varía continuamente, dificultando así el entrenamiento, ya que esto obliga a las capas posteriores a adaptarse a nuevos rangos de valores. Esta técnica reduce el problema normalizando las activaciones de cada *mini-batch*, lo que permite acelerar el entrenamiento y mejorar la estabilidad de la red neuronal al reducir los cambios en la distribución de las activaciones entre capas (*Ioffe y Szegedy, 2015*).

2.3.11 Ajuste de hiperparámetros

Cómo se ha ido observando a través de los diferentes términos explicados, el rendimiento de una red neuronal depende de la correcta selección de sus hiperparámetros, es decir aquellos elementos cuya configuración no se ajusta automáticamente en el entrenamiento, sino que se define anteriormente.

Entre los más importantes, aparecen:

- tasa de aprendizaje
- número de capas ocultas
- número de neuronas por capa
- tamaño del mini-batch
- número de épocas del entrenamiento
- parámetros de regularización.

La correcta selección de estos parámetros repercute directamente en el rendimiento del modelo, ya que una selección inadecuada provoca entrenamientos inestables; por ejemplo, una tasa de aprendizaje demasiado alta impide converger al mínimo, mientras que una tasa baja ralentiza en exceso la convergencia.

El ajuste de estos hiperparámetros se realiza mediante técnicas de búsqueda como *Grid Search* o *Random Search*, que evalúan el rendimiento del modelo con las diferentes configuraciones de parámetros, la primera analiza todas las combinaciones posibles de hiperparámetros definidas, mientras que la segunda evalúa solo un número aleatorio de combinaciones. Esta búsqueda es por tanto un proceso clave para optimizar el rendimiento final de la red neuronal.

2.4 INTERPRETABILIDAD, COMPLEJIDAD Y COSTE COMPUTACIONAL

La creciente complejidad de los conjuntos de datos, hacen que se haya generado un debate importante sobre los modelos de *Machine Learning* entre:

- Precisión predictiva.
- Interpretabilidad.
- Coste computacional

La interpretabilidad en los modelos predictivos es fundamental, ya que permite comprender qué variables influyen en las predicciones realizadas por un modelo y constituye un aspecto especialmente relevante en ámbitos donde las decisiones deben ser justificadas y explicadas (Molnar, 2022).

Ya que mientras algunos algoritmos priorizan la capacidad de aprendizaje (redes neuronales), otros destacan por su capacidad y facilidad de interpretación (modelos clásicos), siendo ambos aspectos relevantes en aplicaciones reales. La siguiente tabla recoge una comparación general entre todos los modelos vistos hasta el momento, para poder visualizar de forma resumida las ventajas y limitaciones de cada uno.

Tabla 2. Comparación entre modelos clásicos y redes neuronales.

Aspecto	Modelos clásicos de <i>Machine Learning</i>	Redes neuronales profundas
Capacidad predictiva	Buena en datos estructurados y problemas simples	Muy alta en problemas complejos y relaciones no lineales
Interpretabilidad	Alta o media	Baja
Coste computacional	Bajo o moderado	Elevado
Tiempo de entrenamiento	Reducido	Alto
Necesidad de datos	Menor volumen de datos	Grandes volúmenes de datos
Robustez ante alta dimensionalidad	Limitada en algunos modelos	Muy elevada
Facilidad de implementación	Alta	Media-Alta
Riesgo de sobreajuste	Moderado	Alto sin regularización
Aplicaciones típicas	Negocio, banca, análisis estructurado	Visión artificial, NLP, reconocimiento complejo

Fuente: Elaboración propia con fines comparativos

La Tabla 2 indica que no existe un modelo claramente superior al resto, sino que al igual que en las redes neuronales se eligen los hiperparámetros, a la hora de seleccionar modelo hay que observar cuál se ajuste mejor en función de la complejidad y el volumen de las bases de datos.

Mientras que los modelos clásicos ofrecen soluciones más eficientes e interpretables computacionalmente, las redes neuronales ofrecen una alta capacidad para modelar relaciones complejas y poder así obtener una alta precisión en dichos modelos.

Siendo el equilibrio entre interpretabilidad, rendimiento y coste computacional uno de los principales ejes de este trabajo, y justifica la necesidad de realizar análisis experimentales comparando ambos enfoques.

3. DISEÑO EXPERIMENTAL

El principal objetivo de este trabajo consiste en estudiar cómo afectan las características del conjunto de datos en el comportamiento de los diferentes modelos explicados, a través de una metodología experimental que compara el rendimiento entre los diferentes algoritmos de clasificación supervisada.

A parte de la capacidad predictiva que obtenga cada modelo, se pretende evaluar más aspectos como la capacidad de generalización, la interpretabilidad o el coste computacional. Todos los modelos van a ser entrenados y evaluados de forma homogénea, para poder compararlos de una forma sencilla e intuitiva.

3.1 ALGORITMOS SELECCIONADOS

Para poder representar los diferentes enfoques de algoritmos de clasificación, se han seleccionado modelos tanto de *Machine Learning* clásico como de Aprendizaje Profundo.

Los modelos clásicos incluidos en el estudio son:

- Regresión Logística
- k-Nearest Neighbors (k-NN)
- Árboles de decisión
- Random Forest
- XGBoost

Por otro lado, el Aprendizaje Profundo está representado a través de las redes neuronales, cuya estructura será ajustada a través de los hiperparámetros vistos anteriormente.

El estudio y desarrollo de estos seis algoritmos para cada uno de los diferentes conjuntos de datos, permitirá observar que algoritmo se ajusta mejor, en función si se opta por interpretabilidad (*Machine Learning* clásico) o por rendimiento (Aprendizaje Profundo)

3.2 CONJUNTO DE DATOS

Con el objetivo de recoger una muestra representativa de conjunto de datos de clasificación, se han seleccionado varios con características heterogéneas que permiten analizar el comportamiento de los algoritmos bajo escenarios variados.

Esta selección busca cubrir diferentes circunstancias reales:

- Conjuntos de datos pequeños y grandes
- Clasificación binaria y multiclase,
- Relaciones lineales y no lineales
- Presencia de ruido
- Alta dimensionalidad
- Variables categóricas
- Problemas desbalanceados

En la siguiente tabla se recogen las principales características de los conjuntos de datos utilizados para el estudio.

Tabla 3. Características de los conjuntos de datos utilizados

Dataset	Tipo de clasificación	Observaciones	Variables	Tipo de variables	Particularidad	Fuente
Simulado no lineal	Binaria	10.000	3	Numéricas	Relaciones no lineales	Elaboración propia
Breast Cancer Wisconsin	Binaria	569	31	Numéricas	<i>Dataset</i> pequeño y estructurado	<i>Kaggle</i> (origen: <i>UCI Repository</i>)
Adult Income	Binaria	48.842 / 45.222 tras limpieza	15 / 96 tras One-hot Encoding	Mixto	Variables categóricas y desbalanceo	<i>Kaggle</i> (origen: <i>UCI Repository</i>)
MNIST (tabular)	Multiclase	70.000	784	Numéricas	Alta dimensionalidad	OpenML
Simulado con ruido	Binaria	5.000	31	Numéricas	Variables irrelevantes	Elaboración propia
Credit Default	Binaria	3.641	85	Numéricas	Riesgo financiero y desequilibrio	Elaboración propia, a través de <i>dataset</i> cedido por Álvaro Caballo en la asignatura de Simulación Financiera

Fuente: Elaboración propia a modo de síntesis.

La Tabla 3 resume las principales características de los conjuntos de datos utilizados en la experimentación. Se incluyen tanto *datasets* reales obtenidos a través de repositorios ampliamente utilizados en la comunidad científica, como *datasets* simulados generados específicamente para este trabajo con el objetivo de analizar el comportamiento de los modelos en escenarios controlados. Para cada conjunto de datos se indica el número de observaciones, el tipo de variables presentes y la principal característica que justifica su inclusión dentro del estudio comparativo.

3.2.1 Dataset simulado no lineal

Es un *dataset* artificial generado a través de la librería *scikit-learn* de Python, ya que de esta forma es posible controlar la complejidad de las relaciones entre variables y la separación entre las diferentes clases, facilitando de esta manera el análisis comparativo entre modelos.

Este conjunto de datos tendrá los objetivos principales de estudiar:

- La capacidad de los algoritmos para aprender relaciones no lineales entre variables.
- El comportamiento de algoritmos lineales en relaciones complejas.
- La capacidad de generalización de los algoritmos frente a este tipo de relaciones complejas.

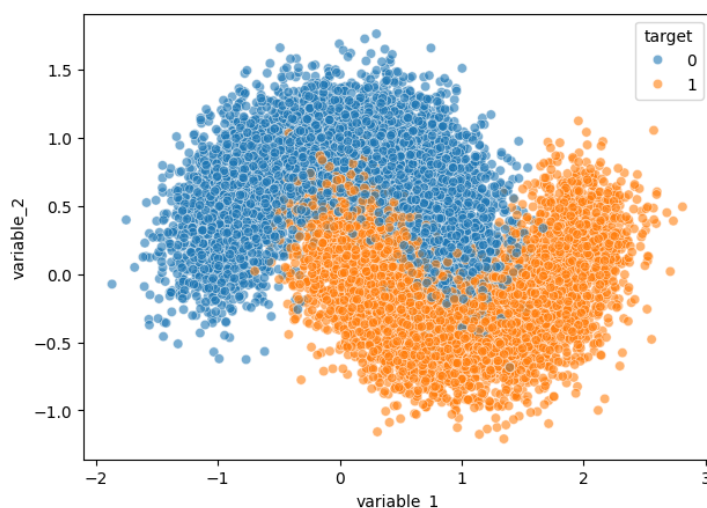
Este conjunto de datos ha sido generado artificialmente mediante la función *make_moons()* de la librería *scikit-learn*, para ello se crearon 10.000 observaciones repartidas en dos clases binarias, fijando una semilla aleatoria para garantizar la reproducibilidad.

Esta función, genera dos grupos de observaciones en forma de semicírculos entrelazados, creando de esta forma una frontera no lineal, siendo una figura muy útil para evaluar la capacidad de los algoritmos para clasificar relaciones no lineales.

Pese a ser un problema sin excesivo ruido, se añadió un nivel de ruido (*noise* = 0,25), para desplazar las observaciones respecto a su posición inicial y provocar un leve solapamiento.

Como resultado, se genera un conjunto de datos con dos variables predictoras y una variable objetivo. Esta combinación de frontera no lineal con ruido moderado permite analizar la capacidad de los diferentes algoritmos para aprender relaciones no lineales y generalizar correctamente sobre datos no perfectamente separables.

Ilustración 3. Dataset simulado no lineal



Fuente: distribución de las variables en dataset simulado no lineal.

3.2.2 Breast Cancer Wisconsin Dataset

Este conjunto de datos consiste en clasificar tumores mamarios como benignos o malignos en función de características extraídas de imágenes mamarias; siendo un ejemplo ampliamente conocido en el estudio de algoritmos de clasificación binaria por su sencillez de formato, ya que presenta:

- Tamaño reducido (cuenta con 569 registros)
- Variables numéricas correctamente definidas
- Baja presencia de ruido que alteren los resultados
- Estructura relativamente limpia en comparación con otros conjuntos de datos.

Estas características permiten analizar:

- El comportamiento de los algoritmos con pocos registros.
- La capacidad de generalización.
- Riesgo de sobreajuste especialmente en modelos complejos.

Aunque el conjunto de datos fue obtenido a través de *Kaggle* para facilitar su descarga y tratamiento, su origen corresponde al repositorio *UCI Machine Learning Repository*, una de las principales fuentes de referencia para la investigación (*Dua y Graff, 2019*).

3.2.3 Adult Income Dataset

Este conjunto de datos, también muy utilizado en el estudio de algoritmos de clasificación, tiene como objetivo predecir, en función de diferentes variables demográficas y laborales, si los ingresos anuales de una persona superan o no los 50.000\$ anuales.

Este conjunto de datos es muy usado en entornos académicos por las siguientes características:

- La combinación de variables numéricas y categóricas.
- El desbalanceo entre clases.
- El elevado número de dimensiones tras la codificación de las variables categóricas.

La inclusión de este conjunto de datos en el trabajo permite extraer varios aspectos importantes:

- Sensibilidad de los modelos ante variables categóricas
- Comportamiento de los algoritmos clásicos frente a las redes neuronales en problemas tabulares.

Al igual que ocurría con el *dataset* anterior, este también fue descargado desde la plataforma *Kaggle*, si bien su fuente original es el repositorio *UCI Machine Learning Repository* (Dua y Graff, 2019), donde se encuentra disponible como uno de los conjuntos de datos de referencia para la evaluación de algoritmos de clasificación.

3.2.4 MNIST (versión tabular)

Este es uno de los conjuntos de datos más utilizados para elaborar modelos de clasificación de imágenes manuscritas. Para este trabajo se va a usar una versión tabular en la que cada imagen va a ser transformada en un vector numérico de 784 variables correspondientes a los píxeles que contiene la imagen.

La utilidad de este conjunto de datos reside en:

- Problema multiclase
- Alta dimensionalidad
- Relaciones complejas entre variables

La inclusión en el trabajo permite estudiar:

- Capacidad de aprendizaje, sobre todo en redes neuronales.
- Comportamiento de modelos clásicos en problemas complejos con un gran número de variables.
- Impacto de la dimensionalidad sobre el rendimiento.

Este *dataset* fue obtenido a través de OpenML, un repositorio público de conjuntos de datos para *Machine Learning*, mediante la función `fetch_openml` de *scikit-learn*.

3.2.5 Dataset simulado con ruido

Al igual que se hizo en el primero caso, se generará un segundo *dataset* sintético que a mayores incorpora:

- Variables irrelevantes
- Ruido aleatorio
- Relaciones parcialmente no lineales.

El objetivo principal de incluir este conjunto de datos en el trabajo es analizar:

- La robustez y sensibilidad de los algoritmos frente al ruido
- Comportamiento frente al posible sobreajuste al ruido.

Siendo especialmente útil para estudiar la capacidad de generalización de modelos complejos.

Este segundo conjunto de datos sintético fue generado mediante la función `make_classification()` de *scikit-learn*, con el objetivo de simular un problema que incorpora ruido y variables irrelevantes,

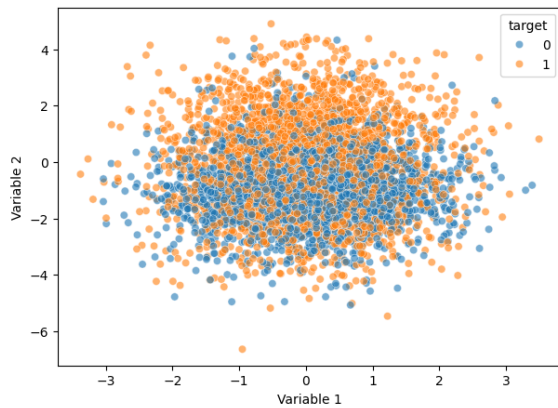
Para la construcción se utilizaron los siguientes parámetros:

- 5.000 observaciones.
- 30 variables predictoras en total.
- 5 variables informativas ($n_informative = 5$), que contienen información útil para determinar la clase.
- 5 variables redundantes ($n_redundant = 5$), generadas a partir de combinaciones lineales de las variables informativas.
- 0 variables repetidas ($n_repeated = 0$).
- 2 clases ($n_classes = 2$).
- Separación entre clases = 1,0 ($class_sep = 1.0$).
- Semilla aleatoria = 42 ($random_state = 42$).

A mayores, en este *dataset* se incorpora el ruido a través de $flip_y = 0,1$, lo que significa que el 10% de las observaciones tienen su clase modificada aleatoriamente, simulando errores de medición, registros incorrectos o inconsistencias que aparecen con conjuntos de datos reales.

A diferencia del simulado no lineal, en este caso las relaciones se generan mediante una combinación de variables informativas y redundantes, lo que produce fronteras de decisión mas complejas y menos evidentes, lo que obliga a los algoritmos a distinguir variables con información y variables con ruido o redundancia,

Ilustración 4. Dataset simulado con ruido



Fuente: distribución de las variables en dataset simulado con ruido.

3.2.6 Credit Default Dataset

Este conjunto de datos consiste en un problema de clasificación binaria orientado a predecir el impago de clientes bancarios en función de su balance.

Me gustaría agradecer a Álvaro Caballo, profesor de la asignatura Simulación Financiera, dónde utilice este *dataset*, su disposición para facilitarme toda la información necesaria y permitir el uso de este conjunto de datos para su estudio en este trabajo.

Esta base de datos se construye a partir de información financiera extraída del balance de empresas. A partir de estos estados financieros se generan distintas métricas e indicadores económico-financieros, como ratios de endeudamiento, liquidez, rentabilidad, fondo de maniobra, *cash flow* o Z-score, que permiten resumir la situación financiera de cada empresa. La variable objetivo indica si la empresa incurre o no en una situación de impago, convirtiendo el problema en una tarea de clasificación binaria orientada a la predicción del riesgo crediticio.

Dicho *dataset* presenta un especial interés debido a:

- Relación directa con problemas reales de riesgo de crédito, aportando una dimensión empresarial al trabajo.
- Presencia de variables financieras reales
- Desbalanceo entre clases

3.3 PREPROCESAMIENTO DE DATOS

Previo al entrenamiento de los modelos, todos los conjuntos de datos han sido preprocesados con el fin de garantizar su correcta implementación y evitar problemas de escalas, variables categóricas o formatos.

Entre las principales técnicas aplicadas, destacan:

- Tratamiento de valores nulos
- Eliminación de variables que no aportan información
- Codificación de variables categóricas a través de *One-Hot-Encoding*.
- Normalización y estandarización de variables numéricas, siendo especialmente importante en algoritmos sensibles a las escalas, como es el caso de k-NN o las redes neuronales.
- Balanceo de clases en el caso de ser necesario
- División del conjunto de datos en entrenamiento y test.

La técnica conocida como *One-Hot Encoding*, consiste en transformar cada categoría de la variable, en variables binarias independientes, es decir, si una variable categórica tiene seis categorías posibles, esta técnica la transforma en seis variables binarias, permitiendo que los algoritmos puedan trabajar sin introducir relaciones ordinales artificiales.

En el caso de normalización y estandarización, se aplicó en los modelos de k-NN o de redes neuronales, ya que son sensibles a la escala de los datos. En el caso concreto del conjunto de datos MNIST, ya que está compuesto por imágenes de dígitos manuscritos, los valores fueron normalizados de una manera alternativa, normalizando entre 0 y 1 a través de la división entre 255, que representa el valor máximo de intensidad de un píxel.

La división entre conjuntos de entrenamiento y prueba se realizó manteniendo la proporción de clases original y garantizando de esta manera una evaluación más representativa del rendimiento de los modelos.

Todo el proceso de preprocesamiento es implementado a través de Python garantizando reproducibilidad, automatización y ausencia de escapes de información.

3.4 VALIDACIÓN DE LOS MODELOS

En algunos de los modelos se utilizó una técnica de validación cruzada estratificada, lo que permite dividir el conjunto de entrenamiento en diferentes subconjuntos que entrenan y evalúan al modelo varias veces, consiguiendo:

- Reducir la dependencia respecto a una única división
- Obtener resultados más estables y robustos.

Además, en varios algoritmos se realizó búsqueda de hiperparámetros mediante técnicas de optimización como:

- Exploración manual.
- Búsqueda iterativa de parámetros, parecido a un *Grid Search* manual.
- *Randomized Search CV* en modelos más complejos como XGBoost.

Esta búsqueda de hiperparámetros permite encontrar las configuraciones que mejor se adapten a cada conjunto de datos y mejorar así la capacidad predictiva de los diferentes algoritmos

3.5 EVALUACIÓN DE LOS MODELOS

3.5.1 Métricas de evaluación

Las métricas de evaluación usadas son compartidas por todos los modelos, garantizando así su sencilla comparación. Dichas métricas no se limitan únicamente a la precisión global, sino que evalúan múltiples aspectos del rendimiento del modelo.

- Accuracy

Representa el porcentaje total de observaciones clasificadas correctamente:

Fórmula 10. Accuracy

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

Fuente: Powers (2011)

donde:

- TP = verdaderos positivos (predicción = +, real = +)
- TN = verdaderos negativos (predicción = -, real = -)
- FP = falsos positivos (predicción = +, real = -)
- FN = falsos negativos (predicción = -, real = +)

Pese a ser ampliamente utilizada, puede resultar insuficiente en algunos problemas desbalanceados, ya que un modelo podría obtener un gran *Accuracy* prediciendo siempre la clase mayoritaria.

- Precision

Mide la proporción de predicciones positivas correctas respecto al total de positivos predichos:

Fórmula 11. Precision

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

Fuente: Powers (2011)

Esta métrica resulta especialmente importante en problemas donde los falsos positivos tienen un coste elevado, por ejemplo, para problemas financieros donde clasificar de manera errónea una operación correcta puede generar graves problemas para el cliente.

- Recall

Representa la capacidad del modelo para detectar correctamente los casos positivos:

Fórmula 12. Recall

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

Fuente: Powers (2011)

Es especialmente relevante en contextos como diagnóstico médico, donde no detectar un caso que es realmente positivo puede tener graves consecuencias.

- F1-Score

Combina *Precision* y *Recall* mediante su media armónica:

Fórmula 13. F1-Score

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

Fuente: Powers (2011)

Esta métrica es muy útil en conjuntos de datos desbalanceados, donde la *Accuracy* puede ser engañosa, en este trabajo, *F1-Score* es el encargado de elegir la mejor combinación de hiperparámetros, ya que permite evaluar simultáneamente la capacidad predictiva y la estabilidad de clasificación.

- F2-Score

Es una variante al *F1-Score*, ya que combina *Precision* y *Recall*, pero en este caso da más peso al *Recall*, siendo una métrica especialmente útil en problemas de clasificación donde la importancia reside en detectar correctamente los casos positivos y no tanto evitar falsos positivos. La siguiente fórmula es para $\beta = 2$:

Fórmula 14. F2-Score

$$F2 = 5 \cdot \frac{Precision \cdot Recall}{4 * Precision + Recall}$$

Fuente: Van Rijsbergen (1979)

Esta métrica resulta especialmente útil en la base de datos de *Credit Default*, donde el objetivo es evitar que una empresa que entre en *default* sea clasificada como no *default*.

- AUC-ROC

La métrica AUC-ROC (*Area Under the ROC Curve*) mide la capacidad del modelo para separar correctamente las clases usando distintos umbrales de decisión. La curva ROC representa la relación entre la tasa de verdaderos positivos y la tasa de falsos positivos para distintos umbrales de clasificación. *Fawcett, T. (2006)*

Valores cercanos a 1 representan una alta capacidad discriminativa del modelo, mientras que cercanos a 0,5 indican un comportamiento aleatorio. Esta métrica es especialmente relevante en los problemas de clasificación binarios.

3.5.2 Tiempo de entrenamiento e inferencia

Para realizar una correcta comparación entre modelos, además de las métricas predictivas, se analiza también el coste computacional asociado a cada algoritmo teniendo en cuenta tanto el tiempo de entrenamiento, como el tiempo requerido para realizar las predicciones.

Este análisis permite analizar la eficiencia computacional y su posible aplicación en entornos reales, ya que modelos más complejos como XGBoost o redes neuronales, suelen incrementar exponencialmente el tiempo de entrenamiento respecto modelos clásicos como regresiones logísticas o árboles de decisión.

Por lo que el estudio del coste computacional permite observar el equilibrio entre:

- Precisión
- Complejidad
- Interpretabilidad
- Eficiencia temporal
- Eficiencia computacional

3.5.3 Herramientas de visualización

A mayores de las métricas tradicionales y el coste computacional, se usan herramientas visuales que facilitan la interpretación del comportamiento de los modelos y la detección de posibles errores en el entrenamiento. Entre las principales herramientas destacan:

- Curvas ROC
- Matrices de confusión
- Curvas de Aprendizaje

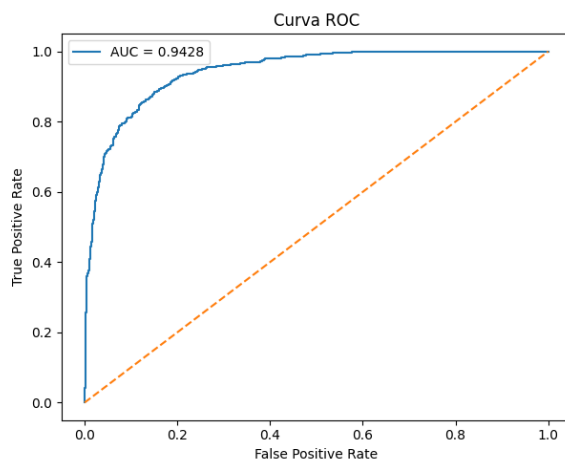
Estas tres herramientas facilitan el análisis de:

- Comportamiento frente al sobreajuste
- Capacidad de generalización
- Tipos de errores cometidos

A) Curvas ROC

Las curvas ROC representan la relación entre la tasa de verdaderos positivos (*Recall*) y la tasa de falsos positivos (porcentaje de negativos reales son clasificados como positivos) para diferentes niveles de clasificación, permitiendo analizar como varía el rendimiento del modelo en función del umbral seleccionado. Es decir, cuanto más próxima esta la curva a la esquina izquierda mejor será la capacidad del modelo para separar entre las clases.

Ilustración 5. Ejemplo de Curva ROC



Fuente: Elaboración propia con fines ilustrativos basada en el proyecto.

B) Matrices de confusión

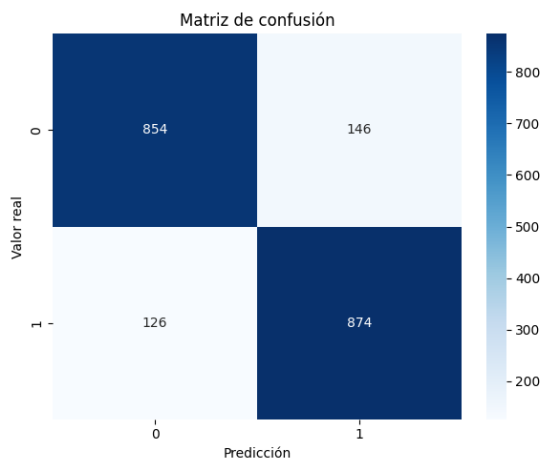
Permiten visualizar de una forma clara y detallada el número de:

- verdaderos positivos,
- verdaderos negativos,
- falsos positivos,
- y falsos negativos.

Esta representación facilita identificar qué tipos de errores comete cada modelo y analizar si existen clases especialmente difíciles de predecir.

Las matrices de confusión son herramientas con una gran utilidad, salvo para los problemas de clasificación multiclase, como MNIST, donde pese a conservar su utilidad pierden algo de interpretabilidad.

Ilustración 6. Ejemplo de Matriz de Confusión



Fuente: Elaboración propia con fines ilustrativos basada en el proyecto.

3.6 DIAGRAMA GENERAL DE LA METODOLOGÍA

Ilustración 7. Diagrama general de la metodología

DATASETS	Adult Income, Breast Cancer, MNIST, Simulados, Credit Default
PREPROCESAMIENTO	• Escalado, codificación y tratamiento de variables
DIVISIÓN TRAIN/TEST	División en conjuntos de entrenamiento y prueba, incorporando el de validación en algunos casos
OPTIMIZACIÓN DE HIPERPARÁMETROS	• Grid Search manual y Randomized Search CV
ENTRENAMIENTO DE LOS MODELOS	• RL, k-NN, Árbol de decisión, Random Forest, XGBoost y Redes neuronales
EVALUACIÓN	• Accuracy, Precision, Recall, F1-Score, F2-Score y AUC
COMPARACIÓN DE RESULTADOS	• Coste computacional, interpretabilidad y capacidad predictiva

Fuente: Elaboración propia con fines explicativos

La Figura 7 resume la metodología aplicada durante el desarrollo de este trabajo de una forma visual, ya que resume de manera global y estructurada todas las fases que componen el desarrollo.

3.7 REPRODUCIBILIDAD DEL EXPERIMENTO

Con el fin de garantizar la reproducibilidad de este estudio, se han adoptado diferentes decisiones comunes en la literatura de *Machine Learning*.

La primera decisión fue fijar la semilla aleatoria (*random_state* = 42), para que todos aquellos algoritmos y procedimientos que contienen elementos aleatorios, generen resultados consistentes entre todas las ejecuciones.

A mayores, todos los modelos han sido entrenados usando la misma división entre entrenamiento y prueba, para garantizar que las comparaciones se realicen bajo las mismas condiciones experimentales.

Por último, la experimentación ha sido completamente desarrollada en Python usando biblioteca ampliamente conocidas y usadas por la comunidad científica, para facilitar la replicación de los experimentos, todo el código desarrollado ha sido publicado en un repositorio público en *GitHub*, permitiendo así a cualquier investigador revisar o profundizar la implementación realizada en este trabajo.

La combinación de estas decisiones metodológicas contribuye a garantizar la transparencia y reproducibilidad del estudio, favoreciendo la validación de los resultados obtenidos y su posible aplicación en futuras investigaciones.

Repositorio disponible en: <https://github.com/guillermogonzalezgar/TFG-Machine-Learning-vs-Deep-Learning>

4. RESULTADOS Y DISCUSIÓN

Tras la aplicación y el análisis de los distintos algoritmos sobre los diferentes conjuntos de datos seleccionados, se ha alcanzado no solo el objetivo principal de identificar el modelo con mayor precisión global, sino que se ha evaluado el comportamiento en función de las características de cada base datos:

- tipo de variables,
- presencia de ruido,
- complejidad de las relaciones entre variables, y
- coste computacional.

Todos los diferentes modelos comparados han sido evaluados con las mismas métricas, para poder realizar de esa manera comparaciones homogéneas.

Los resultados obtenidos en este trabajo siguen la línea de la literatura existente, ya que tal y como indican *Hastie, Tibshirani y Friedman (2009)*, los algoritmos clásicos de *Machine Learning* siguen siendo altamente competitivos en problemas tabulares y estructurados. Por otro lado, *Goodfellow, Bengio y Courville (2016)* reivindican el uso de las redes neuronales en entornos de alta dimensionalidad, demostrando su gran utilidad en grandes volúmenes de datos con relaciones complejas. Es por esto que los resultados explicados a continuación, refuerzan la literatura existente: el *Deep Learning* no sustituye a los modelos clásicos, si no que refuerzan contextos determinados con una complejidad elevada.

4.1 OPTIMIZACIÓN DE HIPERPARÁMETROS

Antes de analizar los resultados obtenidos en cada conjunto de datos, se describe el proceso de optimización aplicado a los distintos algoritmos con el objetivo de encontrar la combinación de hiperparámetros que proporcionara el mejor rendimiento posible. Mientras que los parámetros son aprendidos automáticamente por el modelo durante el entrenamiento, los hiperparámetros deben ser definidos previamente por el investigador y condicionan el proceso de aprendizaje. Por ello, la correcta selección de estos valores resulta fundamental para maximizar la capacidad predictiva de cada algoritmo.

Debido a las grandes diferencias entre los algoritmos estudiados, se han usado diferentes estrategias en función de la complejidad.

4.1.1 Regresión logística

Este modelo ha requerido la utilización de un único hiperparámetro, el *threshold* de clasificación, es decir el valor a partir del cual una observación es asignada a la clase positiva. Aunque en la mayoría de los casos se utiliza el 0,5, en algunos *datasets* como el de Credit Default se usaron otros valores con el objetivo de priorizar la detección de clientes con riesgo de impago,

4.1.2 k-Nearest Neighbors

En este modelo se optimizó el hiperparámetro k , que determina el número de vecinos usados para clasificar una nueva observación. Valores bajos de este hiperparámetro permiten encontrar patrones locales lo que puede favorecer un sobre ajuste, mientras que valores elevados generan modelos estables, pero con menor capacidad de captar relaciones complejas.

Durante el entrenamiento, a través de un *loop* (*for k in [3, 5, 7, 8, 9]*), se han evaluado distintos valores de k hasta encontrar el valor que mejor se ajusta a cada conjunto de datos.

4.1.3 Árbol de Decisión

En este algoritmo se ajustaron hiperparámetros relacionados con la complejidad del árbol, entre los que destacan:

- *max_depth*: profundidad máxima permitida.
- *min_samples_split*: número mínimo de observaciones necesarias para dividir un nodo.
- *min_samples_leaf*: número mínimo de observaciones por hoja.

La optimización de estos parámetros controlar el sobreajuste, ya que un árbol excesivamente profundo puede memorizar de forma literal los datos de entrenamiento y perder totalmente la capacidad de generalización.

4.1.4 Random Forest

Para este algoritmo se optimizaron tanto hiperparámetros relacionados con la complejidad individual de los árboles, como el número de árboles utilizados:

- *n_estimators*: número de árboles
- *max_depth*: profundidad máxima permitida.
- *min_samples_split*: número mínimo de observaciones necesarias para dividir un nodo.
- *min_samples_leaf*: número mínimo de observaciones por hoja.

Un aumento en el número de árboles empleados, suelen mejorar la estabilidad del modelo a la vez que aumenta el coste computacional, es por ello que hay que buscar un equilibrio entre rendimiento y tiempo de entrenamiento.

4.1.5 XGBoost

Este algoritmo ha implementado un proceso de optimización muy exhaustivo, debido al alto número de hiperparámetros que influyen en el rendimiento de este modelo, es por esto que se usó *Randomized Search CV* junto con validación cruzada estratificada, para para explorar distintas combinaciones de hiperparámetros de forma eficiente.

La validación cruzada estratificada, consiste en dividir el conjunto de entrenamiento en varios bloques, manteniendo en cada subconjunto una distribución similar de clases a la del conjunto original. Esta división, permite entrenar y evaluar diferentes particiones, para obtener una estimación mas robusta de su rendimiento y reduciendo la dependencia de una única división.

Los principales hiperparámetros optimizados fueron los siguientes:

- *n_estimators*: Número de árboles utilizados por el modelo
- *max_depth*: Profundidad máxima de cada árbol
- *learning_rate*: Velocidad con la que el modelo corrige los errores cometidos
- *subsample*: Porcentaje de observaciones utilizado en cada iteración
- *colsample_bytree*: Porcentaje de variables utilizadas para construir cada árbol
- *gamma*: Controla la reducción mínima de pérdida necesaria para dividir un nodo.
- *reg_alpha*: Regularización L1 para reducir sobreajuste
- *reg_lambda*: Regularización L2 para reducir sobreajuste

Ilustración 8: Optimización en XGBoost

```
param_grid_xgb = {
    'n_estimators': [50, 100, 200, 300],
    'learning_rate': [0.01, 0.05, 0.1, 0.2],
    'max_depth': [2, 3, 4, 5, 6],
    'subsample': [0.7, 0.8, 0.9, 1.0],
    'colsample_bytree': [0.7, 0.8, 0.9, 1.0],
    'gamma': [0, 0.1, 0.2, 0.5],
    'reg_alpha': [0, 0.01, 0.1, 1],
    'reg_lambda': [0.5, 1, 2, 5]
}
random_search = RandomizedSearchCV(
    estimator=xgb_model,
    param_distributions=param_grid_xgb,
    n_iter=50,
    scoring='f1',
    cv=5,
    random_state=42,
    n_jobs=-1
)
```

Fuente: Elaboración propia a partir del código de optimización implementado en XGBoost.

La utilización de esta optimización permite obtener configuraciones optimizadas para cada conjunto de datos, sin la necesidad de evaluar todas las combinaciones posibles como en los anteriores casos, ya que al implementar *Randomized Search*, probamos solo 50 combinaciones (*n_iter=50*), lo que reduce significativamente el tiempo de entrenamiento, ya que una búsqueda completa (*Grid Search*) con tantos parámetros conllevaría un alto coste computacional.

4.1.6 Redes Neuronales

La red neuronal fue el algoritmo que requirió una mayor optimización de hiperparámetros, en este caso se realizó una búsqueda iterativa sobre las distintas combinaciones posibles, con el objetivo de encontrar la configuración que maximizase las métricas.

Los hiperparámetros optimizados fueron:

- *Arquitectura*: Define el número de capas y neuronas de la red
- *learning_rate*: Controla la velocidad de actualización de los pesos
- *Iteraciones / épocas*: Número máximo de ciclos de entrenamiento
- *Regularización L2*: Penaliza pesos demasiado grandes y reduce sobreajuste
- *Dropout*: Desactiva neuronas aleatoriamente durante el entrenamiento
- *Batch Size*: Número de observaciones usadas en cada actualización
- *Threshold*: Umbral para convertir probabilidades en clases

Ilustración 9. Optimización en redes neuronales

```
arquitecturas = [  
    [16, 8],  
    [32, 16],  
    [64, 32]  
]  
learning_rates = [0.001, 0.0005]  
dropouts = [0.2, 0.3]  
l2_rates = [0.001, 0.01]  
batch_sizes = [16, 32]  
thresholds = np.arange(0.01, 0.99, 0.01)  
for arquitectura in arquitecturas:  
    for lr in learning_rates:  
        for dropout in dropouts:  
            for l2 in l2_rates:  
                for batch_size in batch_sizes:  
                    model = crear_modelo(  
                        arquitectura,  
                        lr,  
                        dropout,  
                        l2  
                    )  
                    history = model.fit(...)  
                    y_prob = model.predict(...)  
                    for threshold in thresholds:  
                        y_pred = (y_prob >= threshold).astype(int)
```

Fuente: Elaboración propia a partir del código de optimización implementado en Redes neuronales.

Para la optimización de las redes neuronales implementadas, se ha desarrollado el anterior procedimiento de búsqueda exhaustivas, en función de la complejidad del conjunto de datos, los valores probados de arquitecturas eran diferentes, siendo valores más grandes para conjuntos más complejos.

En el *dataset* de *Credit Default*, se evaluaron más de 4.700 de combinaciones con el objetivo de seleccionar aquella que maximice la métrica objetivo, F2-Score para este problema

4.2 RESULTADOS GENERALES POR CONJUNTO DE DATOS

La siguiente tabla resume el algoritmo con mejor comportamiento global en cada conjunto de datos, tomando referencia el *F1-Score* ya que es una métrica que mide el rendimiento global del modelo, salvo algunas bases de datos que requieren priorizar una métrica concreta.

Tabla 4. Modelo con mejor rendimiento por conjunto de datos

Conjunto de datos	Mejor modelo	<i>Accuracy</i>	<i>F1-Score</i>	AUC-ROC
Dataset simulado no lineal	k-NN / XGBoost	0,9485	0,9488	0,9874
Breast Cancer Wisconsin	Red Neuronal MLP	0,9912	0,9880	0,9980
Adult Income	XGBoost	0,8676	0,7112	0,9265
MNIST	Red Neuronal MLP	0,9771	0,9771	—
Dataset simulado con ruido	XGBoost	0,8770	0,8746	0,9361
Credit Default	Random Forest	0,7517	0,8232	0,7837

Fuente: extracción de resultados de los modelos de las implementaciones en Python

Los resultados extraídos de las diferentes bases de datos demuestran que no hay un algoritmo dominante sobre el resto. Es decir, la elección del algoritmo de clasificación depende de las características de la base de datos y del objetivo concreto de clasificación. Por ejemplo, para problemas de mayor dimensionalidad como MNIST, la red neuronal obtiene los mejores resultados.

A continuación, se procede a realizar un análisis detallado de cómo se adapta cada modelo en función de las características de las bases de datos.

4.3 DATASET SIMULADO NO LINEAL

Tabla 5. Resultados Dataset Simulado no lineal

Modelo	Accuracy	Precision	Recall	F1-score	AUC-ROC	Tiempo (s)	Interpretabilidad
k-NN	0.9485	0.9436	0.954	0.9488	0.9845	2.6069	No
XGBoost	0.9460	0.9398	0.953	0.9464	0.9874	40.7228	Sí
Árbol de Decisión	0.9420	0.9325	0.953	0.9426	0.9781	0.5189	Sí
Random Forest	0.9415	0.9341	0.950	0.9420	0.9839	20.3806	Sí
Red Neuronal MLP	0.9020	0.8836	0.926	0.9043	0.9481	493.7352	No
Regresión Logística	0.8640	0.8569	0.874	0.8653	0.9428	0.0828	Sí

Fuente: extracción de resultados de los modelos en el dataset simulado no lineal

Los resultados obtenidos en el *dataset* simulado no lineal indican el rendimiento que tienen los diferentes algoritmos en conjuntos de datos con fronteras de decisión no lineales y con un nivel de ruido muy bajo,

Debido a que se trata una base de datos con relaciones poco complejas, los mejores resultados fueron obtenidos por k-NN y *XGBoost*, destacando ambos en *F1-Score* (combina *Precision* y *Recall*). Los resultados de ambos modelos en esta métrica indican un buen equilibrio entre falsos positivos y falsos negativos, lo que garantiza una correcta capacidad para identificar las diferentes clases.

Pese a ser un modelo simple, el buen rendimiento en k-NN se debe a la capacidad de adaptarse a fronteras irregulares, siendo especialmente un algoritmo útil en problemas con ruido moderado, como es este caso. Este comportamiento se debe a que el conjunto de datos únicamente cuenta con dos variables predictoras y una frontera de decisión claramente definida, permitiendo que algoritmos basados en proximidad puedan separar correctamente ambas clases sin necesidad de modelos más complejos.

Por otro lado, *XGBoost* obtuvo grandes resultados por su naturaleza de método *ensemble*, es decir combina múltiples modelos simples para construir uno más complejo y preciso, a través de un aprendizaje secuencial y progresivo. Estas características hacen que tenga una buena capacidad predictiva y una gran capacidad de generalización.

Sin embargo, la regresión logística lineal obtuvo un rendimiento bastante pobre en comparación al resto, ya que este algoritmo solo establece clases de forma lineal, por lo su capacidad para encontrar una frontera de decisión se reduce significativamente.

Por último, la red neuronal pese a lograr resultados satisfactorios, no superó a los modelos clásicos, estos resultados se deben a que el conjunto de datos pese a no ser lineal es muy sencillo y sin presencia de ruido significativo ni alta dimensionalidad.

A modo de conclusión, se demuestra que, para relaciones no lineales sin alta dimensionalidad ni ruido, técnicas clásicas como k-NN o métodos *ensemble* son técnicas altamente competitivas que mejoran a los modelos complejos, y como se observa en la Tabla 5, necesitando un menor tiempo de ejecución y por lo tanto un coste computacional.

4.4 BREAST CANCER WISCONSIN DATASET

Tabla 6. Resultados Dataset Breast Cancer Wisconsin

Modelo	Accuracy	Precision	Recall	F1-score	F2-score	AUC-ROC	Tiempo (s)	Interpretabilidad
Red Neuronal MLP	0.9912	1.0000	0.9762	0.9880	0.9809	0.9977	628.9998	No
Random Forest	0.9737	1.0000	0.9286	0.9630	0.9420	0.9931	10.0445	Sí
XGBoost	0.9737	1.0000	0.9286	0.9630	0.9420	0.9980	51.3091	Sí
Regresión Logística	0.9649	0.9750	0.9286	0.9512	0.9375	0.9960	0.0310	Sí
k-NN	0.9561	0.9744	0.9048	0.9383	0.9179	0.9828	1.0149	No
Árbol de Decisión	0.9211	0.9024	0.8810	0.8916	0.8852	0.9127	0.1440	Sí

Fuente: extracción de resultados de los modelos en el Dataset Breast Cancer Wisconsin

En este caso, debido a la naturaleza médica del problema, los modelos han sido ordenados en función del F2-Score. Esta métrica otorga mayor peso al *Recall*, lo que resulta adecuado en un contexto donde el principal objetivo es minimizar los falsos negativos, es decir, evitar clasificar como benigno un tumor que realmente es maligno. Esta métrica analiza el *Recall* (mayor peso) junto con la *Precision*, evitando favorecer modelos que aumenten la detección de positivos a costa de generar un número excesivo de falsos positivos.

Los resultados muestran un alto rendimiento en todos los algoritmos analizados, debido al reducido número de observaciones y la naturaleza estructurada de las variables, que favorecen que incluso modelos relativamente simples como la regresión logística puedan alcanzar rendimientos muy elevados.

En este caso la red neuronal alcanzó los mejores resultados globales y el mayor valor de F2-Score, reduciendo el riesgo de que un caso maligno sea calificado como benigno. Pese a no ser el factor determinante en detección médica, este modelo indica que los tumores diagnosticados como malignos son efectivamente malignos (*Precision* = 1).

Pese a que la red neuronal obtuvo los mejores resultados, otros modelos como Random Forest y XGBoost obtuvieron valores también muy elevados, demostrando la eficacia de los métodos *ensemble* en problemas de clasificación tabular, ya que al combinar modelos simples mejoran la capacidad de generalización del modelo.

A pesar de la sencillez del modelo, la regresión logística mostró un rendimiento considerablemente alto, por lo que se concluye que las imágenes médicas presentan una clara separación entre clases, es por esto que también k-NN consiguió resultados competitivos.

El modelo con peor rendimiento fue el árbol de decisión, ya que la utilización de un único árbol limita la capacidad de generalización del modelo respecto a los métodos *ensemble*.

En resumen, todos los modelos analizados muestran un alto rendimiento, indicando que las variables disponibles poseen una alta capacidad discriminatoria. Pese a que la red neuronal fue el modelo con mejor rendimiento global, también es el modelo con mayor tiempo de entrenamiento y por lo tanto con mayor coste computacional. Por lo tanto, los métodos *ensemble* son una correcta alternativa, ya que aunque tengan peores

rendimientos, cuentan con una mayor interpretabilidad y un coste computacional más reducido.

4.5 ADULT INCOME DATASET

Tabla 7. Resultados Dataset Adult Income

Modelo	Accuracy	Precision	Recall	F1-score	AUC-ROC	Tiempo (s)	Interpretabilidad
XGBoost	0.8676	0.7739	0.6579	0.7112	0.9265	358.0717	Sí
Red Neuronal MLP	0.8318	0.6472	0.7070	0.6758	0.8924	16299.6961	No
Random Forest	0.8576	0.7836	0.5879	0.6718	0.9146	51.1144	Sí
Árbol de Decisión	0.8505	0.7489	0.5972	0.6645	0.8990	0.3871	Sí
Regresión Logística	0.8458	0.7305	0.5986	0.6580	0.9046	0.8976	Sí
k-NN	0.8258	0.6762	0.5700	0.6186	0.8550	313.9741	No

Fuente: extracción de resultados de los modelos en el Dataset Adult Income

En esta base de datos los algoritmos han sido ordenados usando el *F1-Score*, ya que combina *Precision* y *Recall* en una única métrica, permitiendo evaluar de forma equilibrada la capacidad de identificar correctamente a los individuos con ingresos superiores a 50.000 dólares anuales, es decir un equilibrio entre evitar clasificaciones incorrectas de individuos con altos ingresos (*Precision*), como la pérdida de individuos de dicha categoría (*Recall*).

En la tabla 7 se observa que el mejor modelo fue el XGBoost, ya que además de obtener el mejor valor en *F1-Score*, obtiene el mejor valor en el resto de las métricas salvo en *Recall*. Este resultado era esperable, ya que se trata de un problema tabular con variables tanto numéricas como categóricas y relaciones parcialmente no lineales, un contexto en el que los métodos *ensemble* suelen destacar frente a otros enfoques.

Estos resultados confirman la idea anterior de que los métodos *ensemble* son muy adecuados para datos tabulares. Es por esto que Random Forest también obtuvo un alto rendimiento para esta base de datos.

Para este conjunto de datos tanto la regresión logística como el árbol de decisión obtuvieron resultados cercanos a los mejores modelos, lo que indica que la mayor parte de la información se puede capturar mediante relaciones sencillas, y esa pequeña diferencia de resultados proviene de las ventajas que aportan la combinación de múltiples modelos que capturan las relaciones más complejas.

El modelo con peor rendimiento global fue el k-NN, mostrando dificultades para generalizar debido a que cuenta con un alto número de observaciones y variables, por lo que al ser un modelo basado en la proximidad pierde competitividad frente a otros modelos.

A modo de conclusión, los resultados demuestran que XGBoost fue el modelo más equilibrado obteniendo los mejores valores en *Accuracy*, *F1-Score* y *AUC-ROC*. Aunque la red neuronal alcanzó unos buenos resultados con el mejor *Recall*, el incremento del tiempo de entrenamiento y por lo tanto del coste computacional, junto con la falta de interpretabilidad lo convierten en una alternativa menos atractiva respecto a los métodos *ensemble*.

4.6 MNIST

Tabla 8. Resultados Dataset MNIST

Modelo	Accuracy	Precision	Recall	F1-score	Tiempo (s)	Interpretabilidad
Red Neuronal MLP	0.9771	0.9772	0.9771	0.9771	678.3392	No
XGBoost	0.9761	0.9761	0.9761	0.9761	2412.6643	Sí
k-NN	0.9714	0.9716	0.9714	0.9714	1166.1389	No
Random Forest	0.9687	0.9687	0.9687	0.9687	355.0607	Sí
Regresión Logística	0.9208	0.9205	0.9208	0.9206	104.7097	Sí
Árbol de Decisión	0.8738	0.8736	0.8738	0.8736	64.0628	Sí

Fuente: extracción de resultados de los modelos en el Dataset MNIST

Los resultados obtenidos muestran un gran rendimiento para todos los algoritmos analizados, lo que indica que la representación tabular de las imágenes aporta información suficiente para diferenciar los diez dígitos manuscritos. Sin embargo, las diferencias en cuanto a coste computacional e interpretabilidad son significativas.

En este caso, la red neuronal obtuvo los mejores resultados, demostrando su alta eficacia en entornos con alta dimensionalidad y patrones complejos, como en este conjunto de datos, donde cada observación está formado por cientos de píxeles que generan relaciones complejas difíciles de capturar, que la arquitectura multicapa de la red sí consigue aprender.

XGBoost y Random Forest en menor medida, consiguieron resultados equivalentes a los de la red neuronal, demostrando de nuevo la capacidad predictiva de los métodos *ensemble*, y esta vez no solo en datos tabulares, sino también en problemas complejos siempre que las características relevantes estén correctamente representadas.

Pese a que k-NN es un modelo simple, ha obtenido resultados competitivos, lo que es algo lógico observando la naturaleza del problema, ya que las imágenes de un mismo dígito se tienden a agruparse en regiones próximas.

Por último, la regresión logística obtuvo resultados inferiores lo que demuestra la gran limitación de los modelos lineales en entornos con estructuras tan complejas. Lo mismo sucede con el árbol de decisión, que obtuvo los peores resultados globales, evidenciando las dificultades de un único árbol para modelizar relaciones complejas.

La importancia de este modelo radica en el punto de vista computacional, destacando que en este caso la red neuronal presentó un tiempo de ejecución significativamente inferior al que habría requerido una implementación manual mediante *Numpy*. Para ello se utilizó la biblioteca *Keras*, que incluye optimizaciones internas y automáticas que simplifican muchos procesos como la propagación hacia delante, hacia atrás o la actualización de parámetros. En este trabajo, el uso en los anteriores conjuntos de datos de *Numpy*, se justifica a través de fines didácticos y explicativos, ya que la definición manual de las funciones permite entender el funcionamiento interno de una red neuronal.

Sin embargo, en entornos más complejos, como es el caso de *MNIST*, se empleó *Keras* ya que se trata de una herramienta más eficiente, reproducible y adecuada para entrenar redes neuronales en conjuntos de datos de mayor tamaño y dimensionalidad.

Además, para limitar el tiempo de entrenamiento se aplicaron tres técnicas que ayudaron a reducirlo:

- Mini-batches
- Optimizador Adam
- Early stopping

Estas tres técnicas permiten acelerar el entrenamiento, a la vez que reducen el coste computacional sin disminuir el rendimiento. Gracias a estas estrategias y a la librería *Keras*, es posible una red neuronal con métricas elevadas manteniendo tiempos de ejecución razonables.

En cuanto a la interpretabilidad, la importancia es mucho menor en este conjunto de datos que en los anteriores, ya que a diferencia de otros *datasets* donde resulta clave entender que variables condicionan la predicción, en MNIST al tratarse de un problema de reconocimiento de imágenes con elevado número de variables, la pérdida de interpretabilidad de las redes neuronales resulta menos importante, y el principal objetivo radica en maximizar la capacidad de clasificación.

Ilustración 10. Comparación de red neuronal con Numpy vs Keras

```
def relu(Z):
    return np.maximum(0, Z)

def softmax(Z):
    expZ = np.exp(Z - np.max(Z, axis=0, keepdims=True))
    return expZ / np.sum(expZ, axis=0, keepdims=True)

def inicializar_parametros(input_dim, hidden1, hidden2, output_dim):
    params = {
        "w1": np.random.randn(hidden1, input_dim) * np.sqrt(2 / input_dim),
        "b1": np.zeros((hidden1, 1)),
        "w2": np.random.randn(hidden2, hidden1) * np.sqrt(2 / hidden1),
        "b2": np.zeros((hidden2, 1)),
        "w3": np.random.randn(output_dim, hidden2) * np.sqrt(2 / hidden2),
        "b3": np.zeros((output_dim, 1))
    }
    return params

def forward(X, params):
    Z1 = np.dot(params["w1"], X) + params["b1"]
    A1 = relu(Z1)
    Z2 = np.dot(params["w2"], A1) + params["b2"]
    A2 = relu(Z2)
    Z3 = np.dot(params["w3"], A2) + params["b3"]
    A3 = softmax(Z3)
    cache = (Z1, A1, Z2, A2, Z3, A3)
    return A3, cache

def calcular_coste(A3, Y):
    m = Y.shape[1]
    return -np.sum(Y * np.log(A3 + 1e-8)) / m

def backward(X, Y, params, cache):
    Z1, A1, Z2, A2, Z3, A3 = cache
    m = X.shape[1]
    dZ3 = A3 - Y
    dw3 = np.dot(dZ3, A2.T) / m
    db3 = np.sum(dZ3, axis=1, keepdims=True) / m
    dA2 = np.dot(params["w3"].T, dZ3)
    dZ2 = dA2 * (Z2 > 0)
    dw2 = np.dot(dZ2, A1.T) / m
    db2 = np.sum(dZ2, axis=1, keepdims=True) / m
    dA1 = np.dot(params["w2"].T, dZ2)
    dZ1 = dA1 * (Z1 > 0)
    dw1 = np.dot(dZ1, X.T) / m
    db1 = np.sum(dZ1, axis=1, keepdims=True) / m
    return {
        "dw1": dw1, "db1": db1,
        "dw2": dw2, "db2": db2,
        "dw3": dw3, "db3": db3
    }

def actualizar_parametros(params, grads, lr):
    for i in range(1, 4):
        params["w" + str(i)] -= lr * grads["dw" + str(i)]
        params["b" + str(i)] -= lr * grads["db" + str(i)]
    return params

model = Sequential([
    Input(shape=(X_train_scaled.shape[1],)),
    Dense(128, activation="relu"),
    BatchNormalization(),
    Dropout(0.2),
    Dense(64, activation="relu"),
    BatchNormalization(),
    Dropout(0.2),
    Dense(10, activation="softmax")
])

model.compile(
    optimizer=Adam(learning_rate=0.001),
    loss="sparse_categorical_crossentropy",
    metrics=["accuracy"]
)

early_stop = EarlyStopping(
    monitor="val_loss",
    patience=3,
    restore_best_weights=True
)

history = model.fit(
    X_train_scaled,
    y_train,
    validation_split=0.2,
    epochs=10,
    batch_size=128,
    callbacks=[early_stop],
    verbose=1
)
```

Fuente: elaboración propia para uso explicativo

A través de este ejemplo de una red neuronal simple, se puede observar las diferencias en cuanto a líneas de código que conlleva la implementación con una librería respecto a la otra. Mientras que *Numpy* requiere una implementación manual que requiere programar todas las funciones de la red, *Keras* tiene funciones propias que permite implementarlo de una forma automática. Sin embargo, *Numpy* tiene una gran ventaja, su valor didáctico, ya que permite comprender en detalle el funcionamiento interno de una red neuronal.

4.7 DATASET SIMULADO CON RUIDO

Tabla 9. Resultados Dataset Simulado con Ruido

Modelo	Accuracy	Precision	Recall	F1-score	AUC-ROC	Tiempo (s)	Interpretabilidad
XGBoost	0.877	0.8956	0.8546	0.8746	0.9361	152.6257	Sí
Random Forest	0.875	0.8935	0.8526	0.8726	0.9334	52.1425	Sí
Red Neuronal MLP	0.844	0.8145	0.8924	0.8517	0.9137	1256.7488	No
Árbol de Decisión	0.830	0.8487	0.8048	0.8262	0.8593	5.8831	Sí
k-NN	0.833	0.8781	0.7749	0.8233	0.9080	4.5345	No
Regresión Logística	0.775	0.7821	0.7649	0.7734	0.8399	0.0219	Sí

Fuente: extracción de resultados de los modelos en el Dataset Simulado con Ruido

Este *dataset* permite analizar el comportamiento de los diferentes algoritmos a un problema de clasificación se le incorpora un nivel adicional de dificultad mediante ruido y variables poco informativas. La presencia de ruido en este conjunto de datos permite aproximar a los algoritmos a situaciones reales.

El algoritmo con mejores resultados globales es el XGBoost, seguido del Random Forest, lo que confirma de nuevo la alta capacidad de los métodos *ensemble*, ya que son capaces de combinar múltiples modelos de forma secuencial que permiten adaptarse a relaciones complejas a la vez que mantienen una buena capacidad de generalización frente al ruido o a una baja calidad en los datos.

La red neuronal también es un modelo que obtiene muy buenos resultados, siendo capaz de detectar un porcentaje mayor de observaciones pertenecientes a la clase positiva. Aunque las redes neuronales suelen funcionar mejor en entornos complejos, en este caso no fue así, esto se debe a que la base de datos presenta una dimensionalidad algo baja y un tamaño moderado, características en las que los métodos *ensemble* se desenvuelven mejor. A mayores, el coste computacional de las redes es mayor, lo que las convierte en una herramienta con poca utilidad para estos problemas.

Pese a que en el simulado no lineal sin ruido k-NN fue uno de los modelos que mejor funcionó, se observa que al introducir ruido su rendimiento disminuyó de manera notable, demostrado que este algoritmo es muy sensible a valores atípicos al estar basado en la proximidad de las observaciones. Este demuestra que la robustez frente al ruido constituye una característica tan importante como la propia capacidad predictiva, especialmente cuando se trabaja con datos reales donde es habitual encontrar observaciones erróneas o variables poco informativas.

En resumen, la comparación entre el dataset con ruido y sin ruido evidencia la importancia que tiene la elección del algoritmo en función de la naturaleza del problema. Ya que la presencia de ruido penaliza el rendimiento en algoritmos como k-NN, mientras que favorece a algoritmos *ensemble* más robustos (XGBoost y Random Forest). Con esta comparación la red neuronal mejora su rendimiento al añadir ruido, pero sigue sin superar a los métodos *ensemble* al tratarse de un problema con poca dimensionalidad de variables y pocas observaciones, por lo que no compensa su uso por su alto coste computacional.

4.8 CREDIT DEFAULT DATASET

Tabla 10. Resultados Dataset Credit Default

	Modelo	Accuracy	Precision	Recall	F1-score	F2-score	AUC-ROC	Tiempo (s)	Interpretabilidad
0	Random Forest	0.7517	0.7397	0.9513	0.8323	0.8998	0.7837	240.8275	Sí
1	XGBoost	0.7627	0.7530	0.9428	0.8373	0.8975	0.7885	152.3561	Sí
2	Red Neuronal MLP	0.7407	0.7386	0.9280	0.8225	0.8827	0.7592	1551.0276	No
3	k-NN	0.7572	0.7639	0.9047	0.8283	0.8725	0.7370	16.3251	No
4	Árbol de Decisión	0.7599	0.7676	0.9025	0.8296	0.8719	0.7318	98.2394	Sí
5	Regresión Logística	0.7257	0.7353	0.9004	0.8095	0.8617	0.7146	1.3507	Sí

Fuente: extracción de resultados de los modelos en el Dataset Credit Default

En esta base de datos, los modelos han sido ordenados en función de una nueva métrica, F2-Score, esto se debe a la naturaleza del problema, ya que el objetivo principal, al igual que pasaba en el de *Breast Cancer Wisconsin*, consiste en identificar a los clientes que pueden incurrir en impago, ya que un falso negativo puede acarrear grandes pérdidas para la entidad financiera que incorpore el modelo.

Además, durante la optimización de los modelos en *Python* se exigió un *Recall* mínimo de 0,90. Esta restricción se incorporó con el objetivo de garantizar que los modelos seleccionados fueran capaces de detectar al menos el 90% de los casos de impago, priorizando así la reducción de falsos negativos. No obstante, en el entorno financiero también es importante controlar los falsos positivos, ya que clasificar como empresa de riesgo a una compañía solvente puede provocar la denegación injustificada de crédito o la pérdida de oportunidades comerciales. Por este motivo, se ha optado por el F2-Score, ya que otorga una mayor importancia al *Recall*, pero sin ignorar completamente la *Precision*, proporcionando una evaluación más equilibrada y representativa del rendimiento real de los modelos.

Los resultados demuestran de nuevo que los métodos *ensemble* son los mejores algoritmos para este conjunto de datos, para este caso concreto, el Random Forest alcanzó el mejor F2-Score junto con el mejor *Recall*, indicando que ha conseguido identificar el mayor número de empresas con impago, minimizando los falsos negativos. Por su parte, *XGBoost* consiguió resultados muy parecidos, confirmando la capacidad de estos modelos para capturar relaciones complejas y para generalizar observaciones no vistas.

Este comportamiento puede explicarse por la naturaleza del conjunto de datos, compuesto por indicadores financieros derivados de balances empresariales. Las relaciones entre variables como liquidez, endeudamiento, rentabilidad o capacidad de generación de caja suelen ser complejas y no lineales, favoreciendo el uso de modelos basados en árboles capaces de capturar interacciones entre variables sin necesidad de asumir relaciones lineales. Además, al tratarse de un problema tabular con un número moderado de observaciones y variables, los métodos *ensemble* suelen aprovechar mejor la información disponible que las redes neuronales.

Por otro lado, la regresión logística obtuvo el peor rendimiento global, ya que, pese a que consiguió detectar una gran proporción de impagos, el resto de las métricas fueron bastante inferiores al resto. Este comportamiento demuestra la limitación de este modelo para capturar relaciones complejas entre variables.

En conclusión, los resultados de este *dataset* vuelven a evidenciar que los métodos *ensemble* son la opción más adecuada para problemas tabulares de riesgo crediticio como este caso. Además de obtener los mejores resultados en las métricas de evaluación, presentan una mayor interpretabilidad y un coste computacional inferior al de las redes neuronales. Desde una perspectiva aplicada, estos resultados refuerzan la utilidad de modelos como Random Forest o XGBoost en sistemas de evaluación del riesgo financiero, donde resulta fundamental detectar el mayor número posible de impagos sin generar un volumen excesivo de falsas alarmas y manteniendo la capacidad de justificar las decisiones tomadas por el modelo.

4.9 APLICACIÓN EMPRESARIAL DEL CASO DE USO

Con el objetivo de evaluar la utilidad práctica de los resultados obtenidos, el caso de uso desarrollado sobre el *Credit Default Dataset* fue presentado y discutido en el entorno del área de *Business Analytics* de banca de NTT Data, empresa en la que he desarrollado las prácticas académicas. La propuesta consistía en utilizar información procedente de balances empresariales para generar indicadores financieros relevantes y, a partir de ellos, estimar la probabilidad de impago mediante técnicas de *Machine Learning*.

Desde una perspectiva de negocio, se destacó la utilidad potencial de este tipo de soluciones para apoyar procesos de concesión de crédito, seguimiento de clientes empresariales y detección temprana de situaciones de deterioro financiero. La posibilidad de identificar de forma anticipada empresas con mayor probabilidad de incumplimiento permitiría optimizar los procesos de análisis de riesgo y priorizar la revisión de aquellos casos con mayor impacto potencial para la entidad financiera.

Asimismo, se valoró positivamente la utilización de modelos interpretables como Random Forest o XGBoost, ya que permiten identificar qué indicadores financieros influyen en mayor medida en las predicciones realizadas. Este aspecto resulta especialmente relevante en el sector bancario, donde las decisiones deben estar justificadas y alineadas con criterios regulatorios, de auditoría y de gestión del riesgo.

La valoración obtenida puso de manifiesto que soluciones de este tipo presentan una aplicación directa en el ámbito financiero, actuando como herramientas de apoyo a la toma de decisiones y contribuyendo a mejorar la eficiencia operativa, la homogeneidad en los criterios de evaluación y la capacidad de anticipación ante posibles situaciones de impago.

4.10 COMPARACIÓN GLOBAL ENTRE MODELOS CLÁSICOS Y REDES NEURONALES

Tras analizar las diferentes bases de datos disponibles para este trabajo, se deduce que no existe un algoritmo universal superior al resto, sino que como era de prever, el rendimiento depende en gran medida de las características del problema:

- Dimensionalidad
- Presencia de ruido
- Tamaño de la muestra
- Complejidad de las relaciones

Todas estas características condicionan la capacidad predictiva de los algoritmos analizados.

La primera gran conclusión obtenida de los resultados es que los métodos *ensemble* (XGBoost y Random Forest) son los que han alcanzado las mejores métricas en gran parte de los datasets, especialmente en entornos con datos tabulares y tamaño y dimensionalidad limitada, ya que su capacidad para combinar múltiples árboles de decisión les permite capturar relaciones complejas, a la vez que mantienen la capacidad de generalización evitando así sobreajuste en exceso a los datos de entrenamiento.

Por otro lado, las redes neuronales mostraron un buen comportamiento en problemas con alta complejidad y dimensionalidad, como fue el caso de Breast Cancer Wisconsin o MNIST, sin embargo, en problemas de tamaño moderado, como son el resto de los estudiados, las redes neuronales no traducen sus ventajas teóricas en mejores resultados a los modelos clásicos, añadiendo también el problema de la interpretabilidad y el coste computacional.

Los algoritmos más simples presentan diferentes comportamientos en función de la base de datos estudiada:

- La regresión logística destaca por su rapidez e interpretabilidad, sirviendo muchas veces como base para observar el peso de cada variable en la variable objetivo.
- El k-NN muestra un buen rendimiento en los *datasets* con ruido moderado, al basar sus predicciones en la proximidad de las observaciones.
- Los árboles de decisión sirven para ganar interpretabilidad y observar la importancia de las variables, pero fueron siempre superados por los métodos *ensemble* con mayor capacidad de generalización

En todos los casos estudiados, se observa que las diferencias que resultaron no fueron especialmente significativas, y que estas surgen desde el punto de vista computacional. Las redes neuronales, pese a ser implementadas con la librería *Keras*, requirieron generalmente los mayores tiempos de entrenamiento, evidenciando que las mejoras de rendimiento deben evaluarse juntamente con el coste computacional.

Al igual que hay que evaluar el coste computacional, también hay que evaluar la interpretabilidad, ya que, en ámbitos como la banca o la salud, encontrar las variables que más influyen en las decisiones del modelo es algo clave, casi tan importante como la predicción, es por eso que la supuesta mejora que puedan implementar las redes neuronales está siempre condicionada por su ausencia de interpretabilidad.

Con el objetivo de resumir los principales resultados obtenidos, la tabla 11 recoge los objetivos estudiados y analizados en la experimentación, para así poder identificar los modelos que mejor se adaptan a las situaciones y facilitar una interpretación global.

Tabla 11. Conclusiones finales

Objetivo analizado	Conclusión obtenida
Datos tabulares	XGBoost y Random Forest fueron los modelos más consistentes
Datos de alta dimensionalidad	Las redes neuronales mostraron mejores resultados
Ruido	Los métodos <i>ensemble</i> fueron más robustos
Interpretabilidad	Ventaja para los modelos clásicos
Coste computacional	Las redes neuronales requirieron más recursos

Fuente: Elaboración propia con fines comparativos y didácticos

5. CONCLUSIONES

El objetivo principal de este trabajo consistía en comparar el comportamiento de algoritmos clásicos de *Machine Learning* con redes neuronales multicapa en diferentes problemas de clasificación, analizando su capacidad predictiva, su interpretabilidad, su coste computacional y su robustez frente al ruido.

Los resultados obtenidos a través del modelado con *Python* permiten extraer varias conclusiones.

En primer lugar, se concluye que no hay un modelo óptimo para todos los tipos de problemas, sino que la elección del algoritmo depende tanto de las características de los datos como de los requisitos de la aplicación. Por lo que se puede confirmar una de las hipótesis planteadas: el rendimiento de los modelos está condicionado por factores como la dimensionalidad, el volumen de datos o la complejidad de las relaciones entre variables. Esta hipótesis confirmada pone de manifiesto la importancia que tiene comprender el problema antes de seleccionar una técnica.

Asimismo, los resultados contradicen la idea de que las redes neuronales constituyen siempre la mejor opción para los problemas de clasificación, ya que los métodos *ensemble* obtuvieron los mejores resultados en la mayoría de los conjuntos de datos analizados, lo que demuestra que los algoritmos clásicos continúan siendo una alternativa plenamente vigente.

Por otro lado, los resultados demuestran que, en entornos con complejidad y dimensionalidad elevada, las redes neuronales superan los rendimientos de otros modelos, demostrando la utilidad de las redes en estos entornos. También es importante destacar que, en datos con un número alta de variables, la interpretabilidad se convierte en un factor secundario que no condiciona la elección de un modelo u otro.

Otro aspecto especialmente relevante es alcanzar un equilibrio entre rendimiento e interpretabilidad. Los resultados muestran que, en la mayoría de los casos analizados, es posible obtener un elevado nivel de rendimiento utilizando modelos más interpretables que las redes neuronales. Esta característica resulta especialmente importante en ámbitos regulados como la banca o la salud, donde la capacidad de comprender y justificar las decisiones tomadas por un modelo constituye un requisito fundamental (*Molnar, 2022*).

Por lo tanto, las principales conclusiones de este trabajo son:

- La elección entre algoritmos clásicos de Machine Learning y Deep Learning no puede plantearse como una sustitución de unos por otros. Los resultados demuestran que ambos enfoques son complementarios y que el uso de uno u otro depende del problema a resolver: mientras que las redes neuronales ofrecen ventajas en escenarios complejos y de alta dimensionalidad, los métodos clásicos, especialmente los *ensemble*, continúan siendo altamente competitivos en problemas tabulares.
- Los métodos *ensemble*, considerados conjuntamente debido a las reducidas diferencias de rendimiento observadas entre Random Forest y XGBoost a lo largo de los distintos conjuntos de datos, continúan siendo una solución altamente competitiva en la mayoría de los problemas tabulares analizados, ya que combinan una elevada capacidad predictiva, una buena interpretabilidad —especialmente en el caso de Random Forest— y un coste computacional moderado.

6. DECLARACIÓN DE USO DE HERRAMIENTAS DE INTELIGENCIA ARTIFICIAL GENERATIVA EN TRABAJOS FIN DE GRADO

ADVERTENCIA: Desde la Universidad consideramos que ChatGPT u otras herramientas similares son herramientas muy útiles en la vida académica, aunque su uso queda siempre bajo la responsabilidad del alumno, puesto que las respuestas que proporciona pueden no ser veraces. En este sentido, NO está permitido su uso en la elaboración del Trabajo fin de Grado para generar código porque estas herramientas no son fiables en esa tarea. Aunque el código funcione, no hay garantías de que metodológicamente sea correcto, y es altamente probable que no lo sea.

Por la presente, yo, Guillermo González García, estudiante de ADE + Business Analytics de la Universidad Pontificia Comillas al presentar mi Trabajo Fin de Grado titulado "Comparación entre algoritmos clásicos de Machine Learning y redes neuronales en problemas de clasificación: rendimiento, interpretabilidad y coste computacional", declaro que he utilizado la herramienta de Inteligencia Artificial Generativa ChatGPT u otras similares de IAG de código sólo en el contexto de las actividades descritas a continuación:

1. **Brainstorming de ideas de investigación:** Utilizado para idear y esbozar posibles áreas de investigación.
2. **Referencias:** Usado conjuntamente con otras herramientas, como Science, para identificar referencias preliminares que luego he contrastado y validado.
3. **Metodólogo:** Para descubrir métodos aplicables a problemas específicos de investigación.
4. **Interpretador de código:** Para realizar análisis de datos preliminares.
5. **Estudios multidisciplinares:** Para comprender perspectivas de otras comunidades sobre temas de naturaleza multidisciplinar.
6. **Corrector de estilo literario y de lenguaje:** Para mejorar la calidad lingüística y estilística del texto.
7. **Generador previo de diagramas de flujo y contenido:** Para esbozar diagramas iniciales.
8. **Sintetizador y divulgador de libros complicados:** Para resumir y comprender literatura compleja.
9. **Revisor:** Para recibir sugerencias sobre cómo mejorar y perfeccionar el trabajo con diferentes niveles de exigencia.

Afirmo que toda la información y contenido presentados en este trabajo son producto de mi investigación y esfuerzo individual, excepto donde se ha indicado lo contrario y se han dado los créditos correspondientes (he incluido las referencias adecuadas en el TFG y he explicitado para que se ha usado ChatGPT u otras herramientas similares). Soy consciente de las implicaciones académicas y éticas de presentar un trabajo no original y acepto las consecuencias de cualquier violación a esta declaración.

Fecha: 15 de junio de 2026

A stylized, handwritten signature in black ink, consisting of a large, sweeping 'A' shape followed by a smaller, more complex mark.

Firma: _____

7. BIBLIOGRAFÍA

- Aitude. (2026). *Sigmoid vs Tanh vs ReLU activation function graph* [Imagen]. <https://www.aitude.com/wp-content/uploads/2026/02/sigmoid-vs-tanh-vs-relu-activation-function-graph.png>
- Breiman, L. (2001). *Random Forests*. *Machine Learning*, 45(1), 5–32.
- Bruce, P., Bruce, A., & Gedeck, P. (2020). *Practical Statistics for Data Scientists*. O'Reilly.
- Chen, T., & Guestrin, C. (2016). *XGBoost: A Scalable Tree Boosting System*. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.
- Chollet, F. (2015). Keras. <https://keras.io>
- Dua, D., & Graff, C. (2019). *UCI Machine Learning Repository*. University of California, Irvine.
- Fawcett, T. (2006). An introduction to ROC analysis. *Pattern Recognition Letters*, 27(8), 861–874.
- Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.
- Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning*. Springer.
- Interactive Chaos. (s.f.). *Estructura de una red neuronal*. <https://interactivechaos.com/es/manual/tutorial-de-machine-learning/estructura-de-una-red-neuronal>
- Ioffe, S., & Szegedy, C. (2015). *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. *Proceedings of the 32nd International Conference on Machine Learning*.
- Kaggle. (s.f.). *Adult Income Dataset*. <https://www.kaggle.com/datasets/wenruiiu/adult-income-dataset>
- Kaggle. (s.f.). *Breast Cancer Wisconsin (Diagnostic) Data Set*. <https://www.kaggle.com/datasets/uciml/breast-cancer-wisconsin-data>
- Molnar, C. (2022). *Interpretable Machine Learning* (2nd ed.).
- Ng, A. (DeepLearning.AI). *Improving Deep Neural Networks: Hyperparameter Tuning, Regularization and Optimization*. Coursera.
- Ng, A. (DeepLearning.AI). *Neural Networks and Deep Learning*. Coursera.

- Ng, A. (DeepLearning.AI). *Structuring Machine Learning Projects*. Coursera.
- OpenML. (s.f.). *MNIST dataset*. <https://www.openml.org/>
- Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., et al. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12, 2825–2830.
- Powers, D. M. W. (2011). *Evaluation: From Precision, Recall and F-Measure to ROC, Informedness, Markedness and Correlation*. *Journal of Machine Learning Technologies*, 2(1), 37–63.
- Van Rijsbergen, C. J. (1979). *Information Retrieval* (2nd ed.). Butterworths.