GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

Análisis de los métodos de clasificación en Machine Learning

Autor: Lucía Álvarez Castro

Director: Francisco Martín Martínez

Madrid

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

Análisis de los métodos de clasificación en ML

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el curso académico 2024/2025 es de mi autoría, original e inédito y no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido tomada de otros documentos está debidamente referenciada.

Fdo.: Lucía Álvarez Castro Fecha: 29/06/2025

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

MARTIN MARTINEZ FRANCISCO MARIA - ***7410**
29/06/2025
AC FNMT Usuarios

Fdo.: Francisco Martín Martínez Fecha: 29/06/2025

GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

Análisis de los métodos de clasificación en Machine Learning

Autor: Lucía Álvarez Castro

Director: Francisco Martín Martínez

Madrid

Agradecimientos

Quiero agradecer, en primer lugar, a mi familia, especialmente a mis padres, por apoyarme en todo momento, confiar en mí y no dejarme nunca sola en este proceso. Sin su paciencia, este camino no habría sido posible.

A mis amigos, por estar siempre ahí, sobre todo en los malos momentos, y por recordarme la importancia de disfrutar del proceso.

Por último, quiero agradecer especialmente a mi tutor, Francisco, por su orientación, disponibilidad y apoyo durante todo el desarrollo de este proyecto. Su acompañamiento ha sido clave para avanzar en cada etapa del trabajo.

ANÁLISIS DE LOS MÉTODOS DE CLASIFICACIÓN EN MACHINE LEARNING.

Autor: Álvarez Castro, Lucía.
Director: Martín Martínez, Francisco.
Entidad Colaboradora: ICAI – Universidad Pontificia Comillas

RESUMEN DEL PROYECTO

Este Trabajo de Fin de Grado analiza y compara seis métodos de clasificación en Machine Learning, entre los que se encuentran los Árboles de Decisión, Random Forest, Support Vector Machines (SVM), Redes Neuronales Multicapa (MLP), Regresión Logística y Kolmogorov-Arnold Networks (KAN). Dichas técnicas se aplican a dos conjuntos de datos reales de diferentes dimensiones y características para evaluar aspectos como la precisión, la eficiencia computacional y la interpretabilidad. El primer conjunto es una base de datos meteorológica de menor tamaño, mientras que el segundo dataset está vinculado al bosón de Higgs y es de mayor tamaño.

Los resultados evidencian que las ventajas y desventajas teóricas de las diferentes técnicas se cumplen en la práctica y permiten entender cuándo conviene utilizar cada enfoque, en lugar de encontrar un único modelo óptimo.

Palabras clave: Clasificación, Machine Learning, KAN, Árboles de Decisión, Random Forest, SVM, MLP, Regresión Logística

1. Introducción

El aprendizaje automático supervisado ha ganado protagonismo en los últimos años, especialmente en tareas de clasificación, donde se busca predecir una categoría o clase a partir de variables explicativas que permiten construir un modelo predictivo. En este contexto, existen distintos métodos con ventajas y limitaciones en términos de precisión, eficiencia, interpretabilidad, capacidad de generalización y casos de uso.

Este trabajo tiene el objetivo de analizar seis métodos de clasificación, observando cómo se comportan en la práctica y qué características destacan en función del tipo de conjunto de datos utilizado.

2. Definición del proyecto

El objetivo de este proyecto es demostrar en la práctica las propiedades teóricas de cada modelo. Las técnicas que se evalúan y comparan a lo largo del trabajo son: Regresión Logística, Árboles de Decisión, Random Forest, SVM, MLP y KAN. Se presta especial atención a las KAN ya que surgieron en el año 2024 y no había comparativas con los métodos tradicionales. Se aplican a dos conjuntos de datos reales:

1. weatherAUS: un conjunto de datos meteorológico cuyo objetivo es predecir si lloverá en alguna región australiana al día siguiente [1].

2. **Bosón de Higgs**: un conjunto de datos extenso y complejo cuyo objetivo es predecir la naturaleza de un evento, ideal para evaluar la escalabilidad y capacidad de generalización de los modelos [2].

El enfoque puede dividirse en tres fases:

1. **Preparación del entorno y familiarización con las técnicas:** En esta primera etapa se llevaron a cabo diferentes tutoriales individuales con cada algoritmo para comprender mejor su funcionamiento, sintaxis, parámetros clave y maneras de implementarlo.
2. **Aplicación sobre conjuntos de datos reales:** Se aplicaron distintas técnicas de pre-procesado de datos, tras lo cual se ajustaron los hiperparámetros y se entrenaron los modelos para evaluarlos sobre los dos datasets ya mencionados.
3. **Comparación y análisis de resultados:** A continuación, se elaboraron distintas tablas para comparar las métricas más importantes: *accuracy*, *recall*, F1-Score, AUC-ROC, tiempo de ejecución y entrenamiento, y uso energético. Se analizó también el comportamiento de cada modelo en función de sus ventajas y limitaciones teóricas.

En la Ilustración se muestra un esquema visual de este proceso.

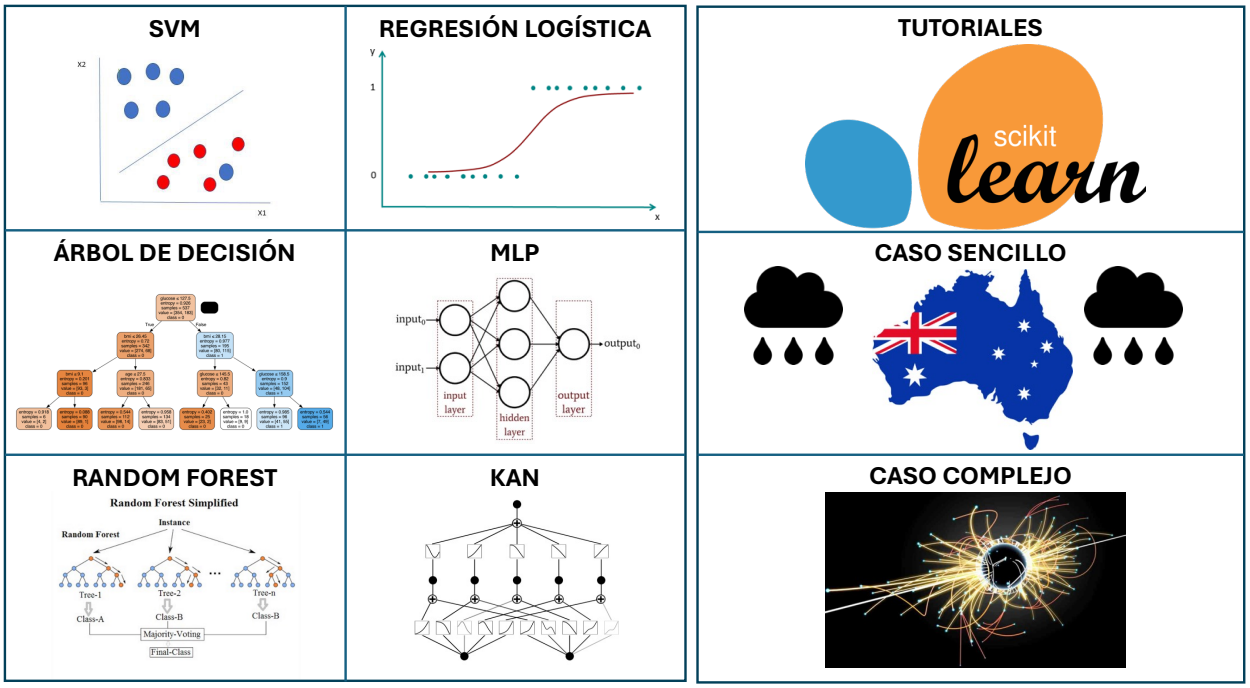

Ilustración I: Esquema visual de la metodología del proyecto.

3. Descripción del sistema y herramientas utilizadas

El proyecto se ha desarrollado íntegramente en Python, con el uso de librerías estándar como `scikit-learn` para la implementación de modelos clásicos, `pyKAN` para el entrenamiento de los modelos KAN y `pandas` o `matplotlib` para la manipulación y visualización de datos.

Todos los modelos se entrenaron bajo las mismas condiciones, aplicando preprocesamiento de datos (escalado, codificación de variables categóricas, selección de variables relevantes), validación cruzada estratificada y análisis de métricas. Para evaluar el consumo energético estimado, se utilizó la herramienta `codecarbon`.

4. Resultados

Los resultados muestran cómo cada modelo responde a las características del dataset. A modo de resumen, se muestran en la Tabla 1 los resultados obtenidos durante el entrenamiento del conjunto de datos más complejo, el del Bosón de Higgs. Estos resultados validan tanto el comportamiento esperado de los modelos clásicos como el rendimiento prometedor de KAN. La tabla se organiza en tres bloques de métricas: las primeras reflejan la precisión del modelo, seguidas por los tiempos de ajuste y entrenamiento, y finalmente el consumo energético estimado.

Tabla 1: Resultados de los modelos aplicados al conjunto de datos del bosón de Higgs.

Modelo	*Accuracy*	*Sensibilidad*	*F1 Score*	*AUC*	*Tiempo de ajuste*	*Tiempo de entrenamiento*	*Uso energético*
SVM	83,38%	70,36%	74,37%	89,54%	1416,3 s.	9746,9 s.	26,118 Wh
Árboles de decisión	80,45%	73,94%	71,93%	87,27%	10,5 s.	253,8 s.	0,587 Wh
Random forest	83,88%	71,38%	75,22%	90,61%	4262 s.	4250,4 s.	9,428 Wh
Regresión logística	73,69%	76,00%	66,00%	81,5%	27 s.	331 s.	1,058 Wh
MLP	83,89%	73,00%	75,53%	90,80%	1159,8 s.	1147,9 s.	2,554 Wh
KAN	81,15%	76,30%	73,50%	88,32%	12,2 s.	16239,3 s.	45,321 Wh

5. Conclusiones

A lo largo de este trabajo se ha demostrado que las propiedades teóricas de los métodos de clasificación se reflejan en su rendimiento práctico, especialmente al aplicarlos en diversos contextos.

Las técnicas clásicas siguen siendo muy competitivas, pero su desempeño depende en gran medida del tipo de datos y del preprocesamiento aplicado. Las redes KAN, a pesar de su novedad y menor documentación, han demostrado un rendimiento robusto en tareas complejas, con potencial para convertirse en una alternativa válida a modelos más tradicionales.

Este estudio proporciona una guía comparativa para elegir la mejor técnica en función de las características del problema. Como trabajo futuro, se propone profundizar en la optimización de las KAN para alcanzar su máximo potencial y ampliar su aplicabilidad a otros casos de uso.

6. Referencias

[1] Anonymous . *Rain in Australia*. Available: https://www.kaggle.com/datasets/jsphyg/weather-dataset-rattle-package/data.

[2] Joycenv. *Higgs Boson Machine Learning Challenge*. Available: https://www.kaggle.com/competitions/higgs-boson/data.

ANALYSIS OF CLASSIFICATION METHODS IN MACHINE LEARNING

Author: Álvarez Castro, Lucía
Supervisor: Martín Martínez, Francisco
Collaborating Entity: ICAI – Universidad Pontificia Comillas

ABSTRACT

This Bachelor's Thesis analyzes and compares six classification methods in Machine Learning: Decision Trees, Random Forest, Support Vector Machines (SVM), Multilayer Perceptrons (MLP), Logistic Regression, and Kolmogorov–Arnold Networks (KAN). These techniques are applied to two real-world datasets of different sizes and characteristics to evaluate aspects such as accuracy, computational efficiency, and interpretability. The first dataset is a smaller meteorological database, while the second is related to the Higgs boson and is considerably larger.

The results show that the theoretical strengths and weaknesses of each method are reflected in practice, helping to identify which approach is more suitable depending on the context, rather than aiming to find a single optimal model.

Keywords: classification, Machine Learning, KAN, Decision Trees, Random Forest, SVM, MLP, Logistic Regression

1. Introduction

Supervised machine learning has gained prominence in recent years, especially in classification tasks, where the goal is to predict a category or class based on explanatory variables that allow the construction of a predictive model. In this context, various methods exist, each with its advantages and limitations in terms of accuracy, efficiency, interpretability, generalization capability, and typical use cases.

This work aims to analyze six classification methods, examining their performance in practice and identifying which characteristics stand out depending on the type of dataset used.

2. Project Definition

The objective of this project is to demonstrate in practice the theoretical properties of each model. The techniques evaluated and compared throughout the work include:

Logistic Regression, Decision Trees, Random Forest, SVM, MLP, and KAN. Special attention is given to KANs, as they emerged in 2024 and had not yet been compared with traditional methods. They are applied to two real datasets:

1. **weatherAUS**: a meteorological dataset aimed at predicting whether it will rain in an Australian region the next day [1].

2. **Higgs Boson**: a large and complex dataset aimed at predicting the nature of an event, ideal for assessing the scalability and generalization capacity of the models [2].

The approach is divided into three phases:

1. **Environment setup and familiarization with techniques**: In this initial stage, individual tutorials were carried out for each algorithm to better understand their operation, syntax, key parameters, and implementation.

2. **Application to real datasets**: Various data preprocessing techniques were applied. Then, hyperparameters were tuned, and the models were trained and evaluated on the previously mentioned datasets.

3. **Comparison and result analysis**: Several tables were created to compare key metrics such as accuracy, recall, F1-score, AUC-ROC, training and inference times, and energy consumption. The behavior of each model was also analyzed considering its theoretical strengths and limitations.

Figure shows a visual diagram of this process.

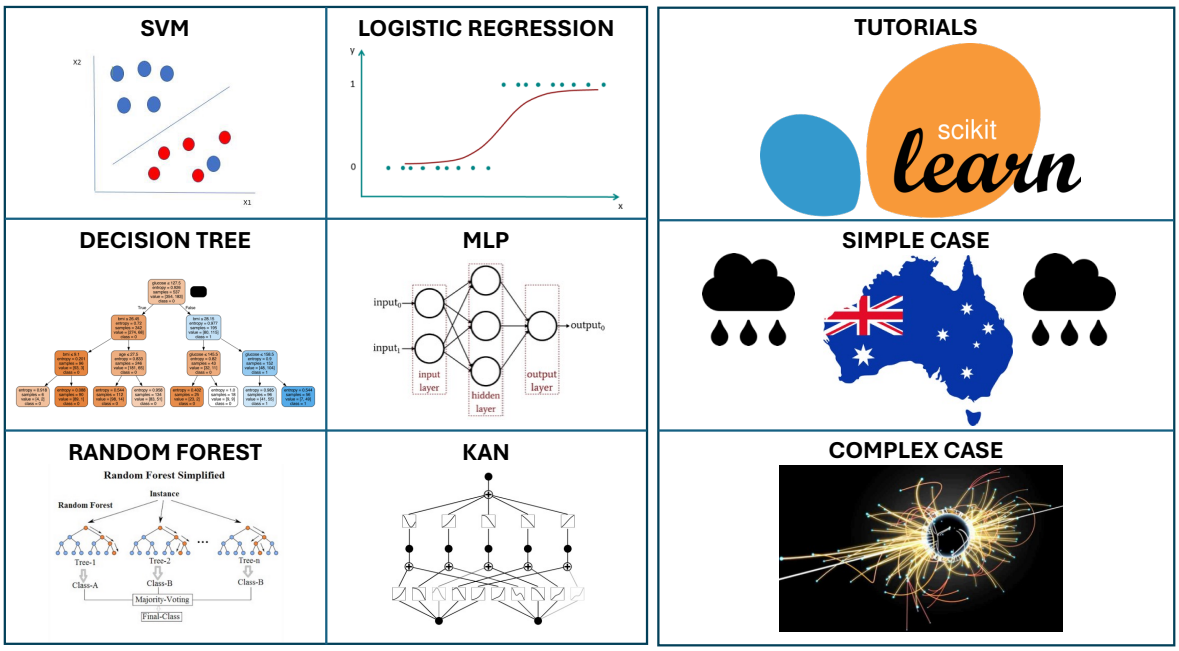

Figure I: Visual diagram of the project methodology.

3. System Description and Tools Used

The entire project was developed in Python using standard libraries such as `scikit-learn` for classical model implementation, `pyKAN` for training KAN models, and `pandas` and `matplotlib` for data manipulation and visualization.

All models were trained under the same conditions, applying data preprocessing (scaling, categorical variable encoding, relevant feature selection), stratified cross-validation, and metric analysis. To estimate energy consumption, the `codecarbon` tool was used.

4. Results

The results show how each model responds to the characteristics of the dataset. As a summary, Table 1 presents the results obtained during the training phase using the more complex dataset — the Higgs boson dataset. These results validate both the expected behavior of traditional models and the promising performance of KAN. The table is organized into three metric blocks: the first reflect model accuracy, followed by tuning and training times, and finally the estimated energy consumption.

Table 1: Results of the models applied to the Higgs boson dataset.

Model	*Accuracy*	*Recall*	*F1 Score*	*AUC*	*Tuning Time*	*Training Time*	*Energy Consumption*
SVM	83,38%	70,36%	74,37%	89,54%	1416,3 s.	9746,9 s.	26,118 Wh
Decision Trees	80,45%	73,94%	71,93%	87,27%	10,5 s.	253,8 s.	0,587 Wh
Random Forest	83,88%	71,38%	75,22%	90,61%	4262 s.	4250,4 s.	9,428 Wh
Logistic Regression	73,69%	76,00%	66,00%	81,5%	27 s.	331 s.	1,058 Wh
MLP	83,89%	73,00%	75,53%	90,80%	1159,8 s.	1147,9 s.	2,554 Wh
KAN	81,15%	76,30%	73,50%	88,32%	12,2 s.	16239,3 s.	45,321 Wh

5. Conclusions

This work has demonstrated that the theoretical properties of classification methods are reflected in their practical performance, especially when applied in varied contexts.

Traditional techniques remain highly competitive, though their performance depends heavily on the data type and preprocessing applied. Despite their novelty and limited documentation, KANs have shown robust performance in complex tasks, with potential to become a valid alternative to more established models.

This study provides a comparative guide to choosing the most suitable technique based on the characteristics of each problem. As future work, it is proposed to further optimize KANs to reach their full potential and expand their applicability to other use cases.

6. References

[1] Anonymous . *Rain in Australia*. Available: https://www.kaggle.com/datasets/jsphyg/weather-dataset-rattle-package/data.

[2] Joycenv. *Higgs Boson Machine Learning Challenge*. Available: https://www.kaggle.com/competitions/higgs-boson/data.

Índice de la memoria

Índice de figuras

Índice de tablas

Capítulo 1. INTRODUCCIÓN

El ***Machine Learning* (*ML*)** es una rama de la ciencia que se encarga de programar sistemas y desarrollar algoritmos que puedan aprender patrones a partir de una base de datos denominada *training set*, que contiene muestras utilizadas para entrenar los modelos. Existen numerosos y diversos ejemplos de aplicaciones de ML, abarcando desde filtros de spam en el correo electrónico hasta sistemas más avanzados, como inteligencias artificiales [1]. El uso generalizado de estas técnicas puede verse reflejado en diversos campos y disciplinas como la salud, la educación, las finanzas, el marketing o la política.

Gracias a los continuos avances en el campo del *Machine Learning*, se han desarrollado métodos más sofisticados, eficientes y precisos, especialmente durante las dos últimas décadas. Esta rápida evolución ha sido impulsada por la necesidad de resolver problemas cada vez más complejos y el acceso a grandes volúmenes de datos. Estas mejoras han permitido la investigación de nuevas tecnologías que amplían aún más el potencial de estos sistemas [2]. El reciente desarrollo y el rápido crecimiento de la inteligencia artificial han contribuido también a la creación de nuevos métodos y algoritmos.

Uno de los casos de uso sobre los que se profundizará a lo largo de este trabajo es la clasificación, que consiste en utilizar un sistema capaz de predecir una clase. Por ejemplo, el filtro de spam mencionado anteriormente clasifica un correo electrónico en “spam” o “no spam” en base a distintas características de entrada. Esto puede llevarse a cabo utilizando uno de los múltiples de algoritmos disponibles, eligiendo aquellos que más se ajusten a los objetivos buscados [3].

Capítulo 2. DESCRIPCIÓN DE LAS TECNOLOGÍAS

El presente capítulo tiene como objetivo introducir los conceptos fundamentales relacionados con la clasificación en ML, así como presentar las tecnologías y herramientas empleadas a lo largo del trabajo. Se comienza explicando qué es un método de clasificación supervisada y algunos casos de uso. A continuación, se expondrá la metodología general seguida durante el desarrollo del trabajo, para posteriormente introducir con brevedad los seis modelos de clasificación analizados en este proyecto. Finalmente, se justifica la elección de *Python* como entorno de desarrollo frente a otras alternativas.

2.1 LOS MÉTODOS DE CLASIFICACIÓN

La clasificación y regresión son los dos tipos de tareas más comunes dentro del aprendizaje supervisado en ML. La primera consiste en separar los datos y la segunda en ajustarlos. El aprendizaje supervisado utiliza ejemplos y muestras con datos de entrada y de salida para entrenar un modelo que posteriormente sea capaz de clasificar una salida en función de nuevas entradas que reciba. Un ejemplo típico es la categorización de textos, como la detección de la emoción en un tweet o una reseña de producto.

La clasificación convierte matemáticamente datos de entrada a los que se les llama variable X en una variable de salida Y, que puede ser una etiqueta o una categoría. Puede aplicarse a datos estructurados o no estructurados para predecir la clase de los puntos de datos proporcionados. Existen diferentes tipos de clasificación supervisada en función de la cantidad y naturaleza de las etiquetas a predecir. Los tres enfoques principales son la clasificación binaria, la multiclase y la multietiqueta, tal y como se muestra en la Figura 1.

- **Clasificación binaria**: Se debe categorizar entre dos clases. En este tipo de tareas, la condición normal suele pertenecer a una clase mientras que el estado patológico pertenece a otra distinta. El ejemplo más típico de clasificación binaria es el detector de spam en correo electrónico ya mencionado en el capítulo de Introducción.

- **Clasificación multiclase**: Se utiliza para tareas en las que existen más de dos etiquetas de clasificación. Al contrario que la clasificación binaria, no se utiliza el concepto de resultado normal o patológico. Se categoriza en función de un rango de clases definido. Por ejemplo, el set de datos Iris clasifica las flores en tres tipos: *Setosa*, *Versicolor* o *Virginica*.
- **Clasificación multietiqueta:** Este tipo de clasificación en ML es crucial cuando un ejemplo está conectado a varias clases o etiquetas. Esta modalidad es, en realidad, una extensión del caso multiclase, y requiere utilizar técnicas más sofisticadas. Un ejemplo de esta clasificación son los artículos de *Google News*, donde un mismo artículo puede pertenecer a la sección "tecnología" y "última hora" [4].

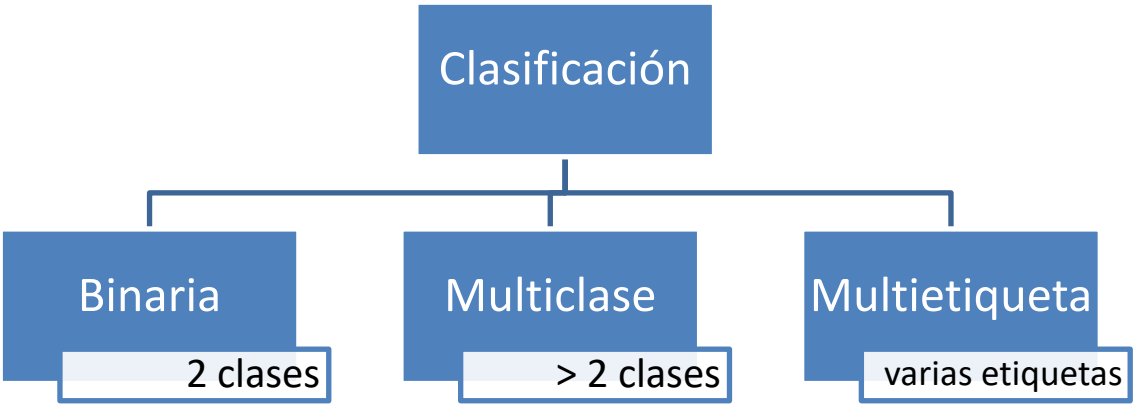

Figura 1: Tipos de Clasificación en Machine Learning

2.2 TÉCNICAS DE CLASIFICACIÓN UTILIZADAS

Para el desarrollo de este trabajo se han seleccionado seis técnicas de clasificación supervisada, cinco de uso extendido en el ámbito del aprendizaje automático y las recientes

KAN. Estas técnicas han sido elegidas con el fin de representar diversos enfoques, desde métodos más simples y lineales hasta los modelos más complejos y no lineales. El objetivo principal es realizar una comparativa representativa de su comportamiento y características. A continuación, se presenta una breve descripción de cada una de ellas:

- **Support Vector Machines (SVM):** Las máquinas de vectores soporte tienen su origen en los años 90. Inicialmente fueron pensadas para resolver problemas de clasificación binaria, pero actualmente se utilizan para resolver problemas más allá de la clasificación binaria, como tareas de regresión y clasificación multiclase en datos complejos [5]. Las *SVMs* pertenecen a la categoría de los clasificadores lineales, ya que inducen separadores lineales o hiperplanos [6]. La idea es que estos hiperplanos separen los datos en diferentes clases con el máximo margen posible. Las *SVMs* destacan por su gran capacidad de generalización y eficiente gestión de memoria, ya que trabajan con un subconjunto de puntos de entrenamiento [7]. Son precisas, robustas frente a valores atípicos y flexibles, dado que pueden adaptarse a fronteras no lineales complejas. Sin embargo, son modelos de caja negra, lo que dificulta su interpretación, y requieren un preprocesamiento de los datos para la clasificación [8]. Esta diferencia entre problemas linealmente separables y no separables se ilustra en la Figura 2, donde se aprecia que, en algunos casos, no es posible separar las clases con una línea recta en el espacio original, lo que justifica el uso de funciones *kernel*.

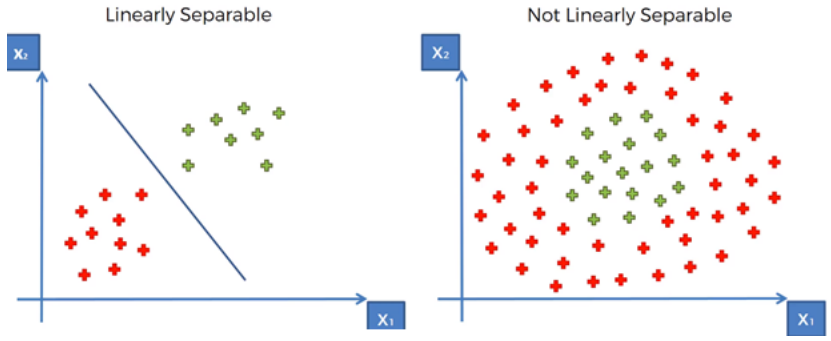

Figura 2: Ejemplo de conjuntos de datos linealmente separables (izquierda) y no separables (derecha)[9]

- **Árboles de Decisión (Decision Trees):** Son algoritmos que utilizan una estructura de árbol para modelar decisiones basadas en condiciones para obtener un resultado. Son muy útiles para problemas en los que las decisiones han de tomarse secuencialmente. Es uno de los algoritmos con mayor interpretabilidad, ya que es fácil de visualizar y entender cómo se toman las decisiones dentro del modelo. Además, requiere un preprocesamiento de los datos menor al de otros algoritmos, ya que no necesita normalizar los datos. Son robustos frente a valores atípicos. Por otro lado, son modelos propensos al *overfitting*, especialmente en árboles profundos, con un gran coste computacional en conjuntos de datos grandes [10], [11]. Un ejemplo práctico puede verse en la Figura 3, donde se muestra un árbol de decisión entrenado para predecir la presencia de diabetes a partir de variables como el índice de masa corporal (BMI), los niveles de glucosa o la edad. Este árbol de decisión se desarrolló en uno de los tutoriales iniciales de este trabajo.

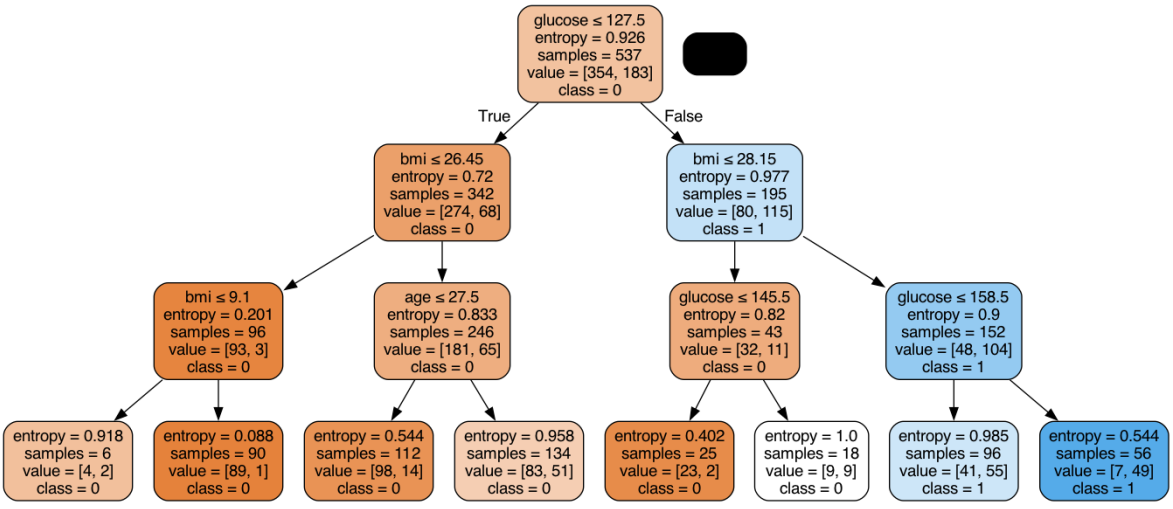

Figura 3: Árbol de decisión entrenado sobre un conjunto de datos de diabetes

- **Random Forest**: Es un método de *ensemble* basado en árboles de decisión que se combinan para mejorar la precisión y reducir el riesgo de *overfitting*. Cada árbol del conjunto tiene un voto, asignando cada entrada a la etiqueta de clase que considera más probable, como puede observarse en la Figura 4 [12]. Este algoritmo puede utilizarse también en problemas de regresión. Ofrece una mayor precisión que los

árboles de decisión, con una mayor robustez frente a los valores atípicos. Sin embargo, tienen un gran coste computacional, son lentos y poco interpretables [13].

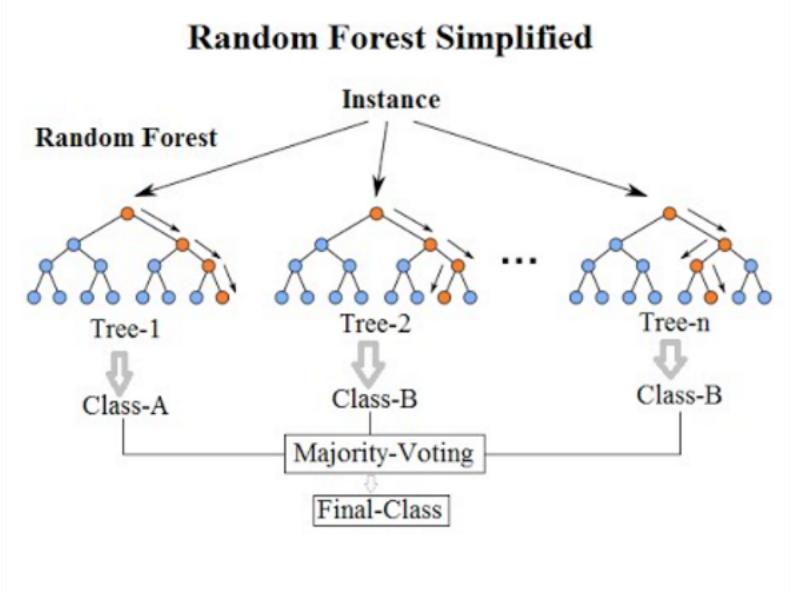

Figura 4: Esquema simplificado del funcionamiento del algoritmo Random Forest [14]

- **Regresión logística:** Este método determina qué variables tienen más importancia para aumentar o disminuir la probabilidad de pertenecer a una clase en concreto [15]. Se utiliza sobre todo para clasificación binaria. Modela la probabilidad de que un dato pertenezca a una clase específica. Es un algoritmo fácil de implementar, entender y entrenar, rápido al trabajar con bases de datos pequeñas y con poca tendencia al *overfitting*. Sin embargo, su simpleza hace que no sea adecuado para encontrar relaciones complejas entre datos, ya que asume una relación lineal entre variables dependientes e independientes. Su principal limitación es que requiere que las variables independientes no presenten colinealidad [16].
- **Redes Neuronales Multicapa (MLP):** Consisten en una red neuronal artificial de varias capas: una de entrada, una o más capas ocultas y una capa de salida. Se puede observar un ejemplo de dicha estructura en la Figura 5 . Cada capa está formada por neuronas que aplican funciones de activación no lineales, lo que permite modelar relaciones complejas entre variables de entrada y salida. Tiene un buen rendimiento

en conjuntos de datos variados y complejos, ofreciendo una predicción rápida y eficiente tras el entrenamiento. Sin embargo, tiene un gran coste computacional, el entrenamiento es lento, y puede ser propenso al overfitting cuando se entrenan con bases de datos pequeñas. Presentan también dificultades de interpretabilidad, ya que funcionan como cajas negras donde no es fácil entender cómo se toman las decisiones [17], [18].

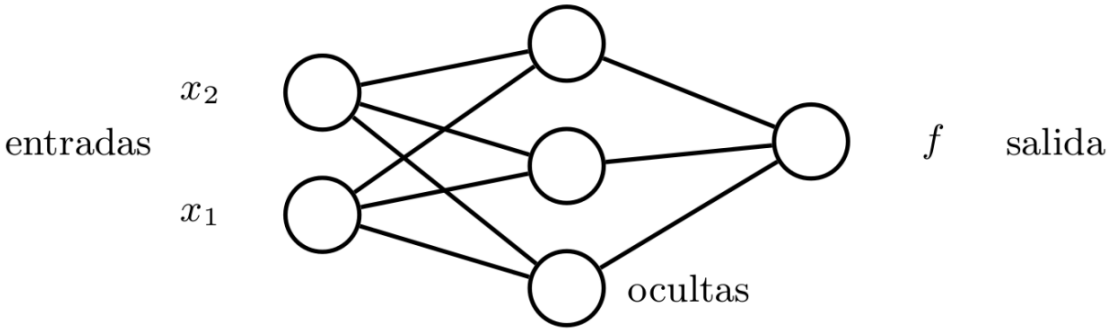

Figura 5: Estructura de una red neuronal multicapa (MLP)[19]

- **Kolmogorov-Arnold Networks (KAN):** Se trata de una novedosa alternativa a las redes neuronales convencionales, propuesta como una forma innovadora de abordar tareas de clasificación y regresión. En lugar de basarse en el teorema de aproximación universal, como los modelos tradicionales, las KAN se basan en el teorema de representación de Kolmogorov-Arnold, que garantiza que cualquier función multivariable continua puede representarse como una suma de funciones univariables compuestas [20]-[23]. Ofrece una gran interpretabilidad, ya que su funcionamiento interno puede analizarse mediante funciones univariables visuales. Una representación esquemática de su arquitectura puede observarse en la Figura 6.

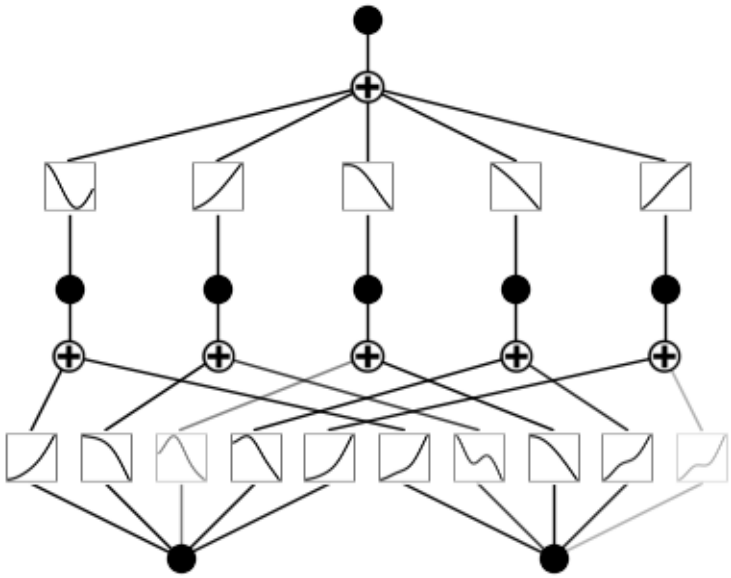

Figura 6: Arquitectura interna de una KAN[24]

Tras trabajar con cada uno de estos métodos sobre el mismo conjunto de datos y evaluar su comportamiento bajo las mismas métricas, se podrá realizar una comparativa rigurosa, comparando sus fortalezas y debilidades.

2.3 ESTRUCTURA DEL TRABAJO

En la Figura 7 se muestra un resumen visual de la estructura del trabajo, incluyéndose las seis técnicas de clasificación usadas y los conjuntos de datos elegidos. Este documento está estructurado de forma que en el Capítulo 2. y en el Capítulo 3. se describen las técnicas utilizadas y el estado de la cuestión y la metodología en el Capítulo 4. En el Capítulo 5. se explica detalladamente la estructura, el funcionamiento y el proceso de entrenamiento de las KAN, por su novedad. En el Capítulo 6. se detallan los modelos desarrollados trabajando con el caso sencillo de la lluvia en Australia y en el Capítulo 7. se detallan los modelos desarrollados trabajando con el caso complejo del Bosón de Higgs.

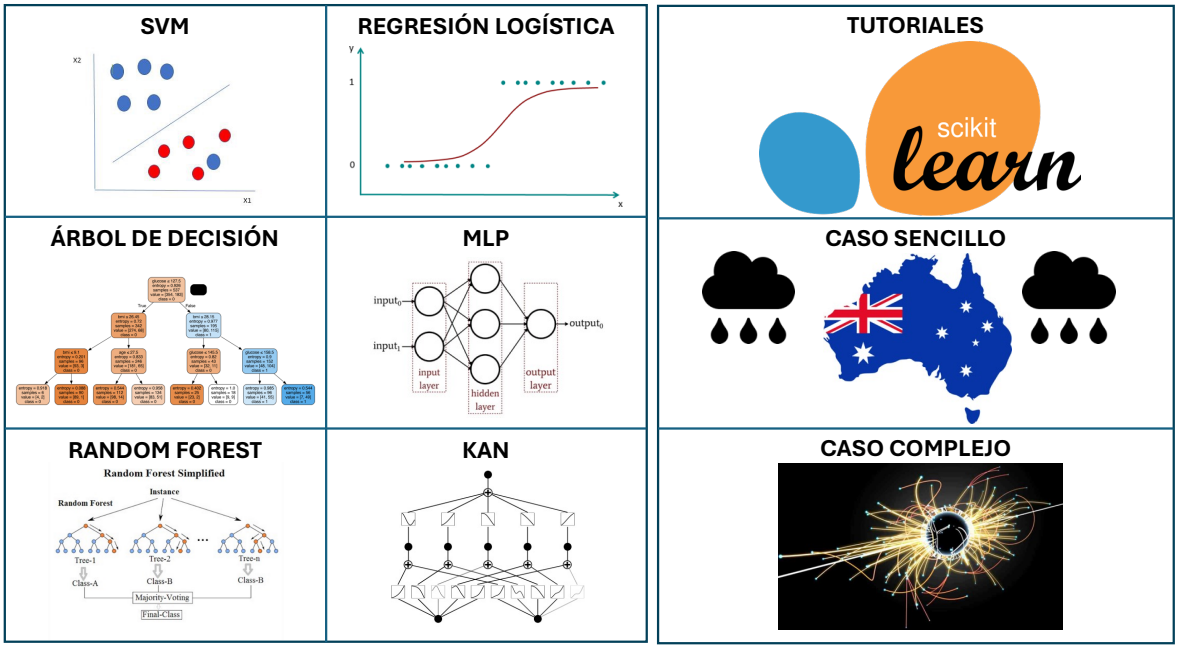

Figura 7: Resumen visual de la estructura del trabajo.

Capítulo 3. ESTADO DE LA CUESTIÓN

Este capítulo presenta el estado de la cuestión en relación con los enfoques y técnicas de clasificación más relevantes en el ámbito del *Machine Learning*. Se incluyen tanto métodos tradicionales de aprendizaje supervisado como aproximaciones más recientes basadas en el *Deep Learning*. Asimismo, se detallan las librerías utilizadas en el desarrollo del trabajo y los indicadores empleados para evaluar el rendimiento de los modelos.

3.1 ENFOQUES DE MACHINE LEARNING EN CLASIFICACIÓN

La clasificación es una de las tareas más comunes dentro del *Machine Learning*, y puede abordarse mediante distintos enfoques. En este apartado se presentan las dos categorías principales que se han utilizado en este trabajo: el aprendizaje supervisado, que incluye técnicas clásicas como árboles de decisión o máquinas de vectores soporte (SVM), y el *Deep Learning*, representado por modelos como las redes neuronales o las redes Kolmogorov-Arnold KAN. Ambos enfoques presentan características particulares en cuanto a complejidad, interpretabilidad y capacidad de generalización.

Los sistemas de ML pueden clasificarse en distintas categorías en función del tipo de aprendizaje. En la Figura 8 se presenta una visión general de esta clasificación.

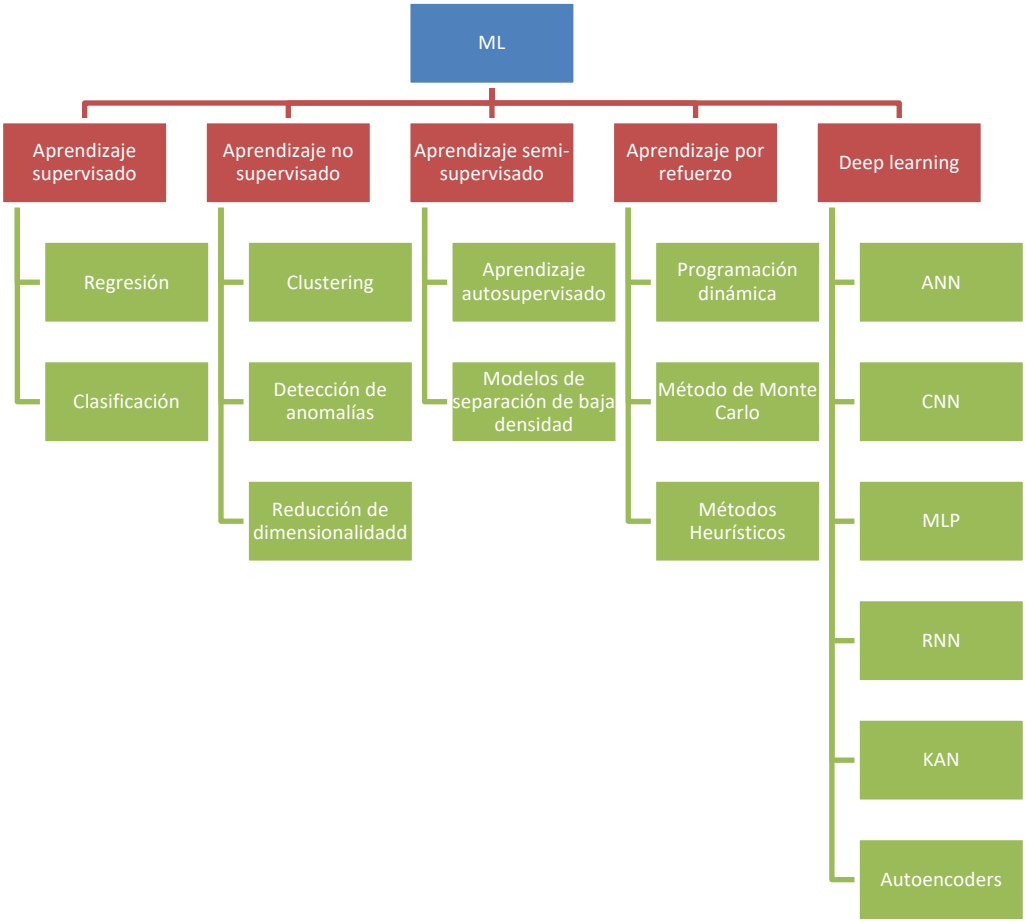

Figura 8: Taxonomía de enfoques de ML. Fuente: elaboración propia a partir de [25]

3.1.1 APRENDIZAJE SUPERVISADO

El aprendizaje supervisado consiste en que el algoritmo se entrena con un *training set* que incluye las respuestas deseadas o datos etiquetados [1]. Algunos de los algoritmos de aprendizaje supervisado más importantes son:

- **Regresión**: predice un valor numérico. Son algoritmos utilizados para establecer una relación lineal entre variables que pueden ser o no dependientes entre sí [16].
- **Clasificación**: predice una etiqueta o clase a partir de un conjunto de características. Dentro de esta categoría pueden encontrarse los siguientes algoritmos [26]:

- ***K-Nearest Neighbors* (KNN):** algoritmo en el que los puntos de datos similares forman grupos utilizados para identificar nuevos objetos sin etiquetar [16].
- **Máquinas de Vectores de Soporte (SVM):** algoritmos útiles para la regresión y clasificación de conjuntos de datos complejos de un tamaño pequeño o medio.
- **Árboles de Decisión:** algoritmo que utiliza una estructura de árbol para modelar decisiones basadas en condiciones para obtener un resultado.
- ***Random Forest*:** conjunto de múltiples árboles de decisión que se combinan para mejorar la precisión y reducir el riesgo de *overfitting*.
- ***Naïve Bayes*:** algoritmo basado en la probabilidad condicional, es decir, existe un evento A cuya probabilidad depende de otro evento B que ya ha ocurrido. Estos algoritmos son utilizados en casos en los que la clasificación ha de hacerse rápidamente sobre una gran cantidad de datos [16].
- **Regresión logística:** modela la probabilidad de que un dato pertenezca a una clase específica.

Nota: algunos de estos algoritmos ya fueron explicados en detalle en el Capítulo 2. , por lo que aquí se presentan de forma resumida.

3.1.2 DEEP LEARNING

El *Deep Learning* es un subconjunto de técnicas de *Machine Learning* que emplea redes neuronales profundas, compuestas por múltiples capas, para procesar grandes volúmenes de datos. Estas redes están inspiradas en el funcionamiento del cerebro humano y son capaces de aprender tanto con supervisión como sin ella. Se aplican en una amplia variedad de campos, como el procesamiento de imágenes, el análisis financiero o el reconocimiento de voz. Dentro de esta rama, los métodos pueden clasificarse según el tipo de aprendizaje en supervisados y no supervisados [27]:

- **Redes Neuronales Artificiales (ANN):** Son el bloque principal del *Deep Learning* y permiten resolver problemas altamente complejos. Identifican patrones, comparan

sus predicciones con los resultados reales y ajustan sus parámetros para minimizar el error.

- **Redes Neuronales Convolucionales (CNN):** Redes neuronales muy utilizadas en el campo del reconocimiento de imágenes por su capacidad para capturar relaciones espaciales entre píxeles. Estas redes están compuestas por capas convolucionales y de submuestreo que permiten extraer características espaciales de las imágenes, seguidas de capas densas que realizan la clasificación (véase Figura 9).

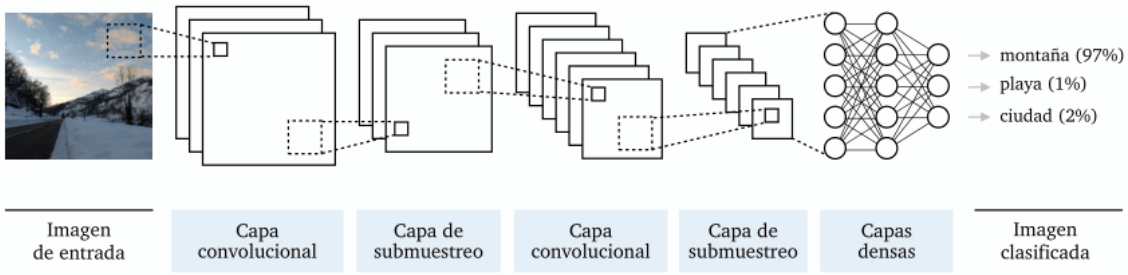

Figura 9: Funcionamiento típico de una red neuronal convolucional (CNN)[28]

- **Redes Neuronales Multicapa (MLP):** Red neuronal de varias capas que se utiliza principalmente en problemas de clasificación.
- **Redes Neuronales Recurrentes (RNN):** Son redes que alimentan la salida de un paso como entrada del siguiente. Este algoritmo es muy utilizado para secuencias de datos, como la captura de imágenes, procesamiento del lenguaje natural o reconocimiento de escrituras a mano.
- **Redes Kolmogorov-Arnold (KAN):** Se trata de una novedosa alternativa a las redes neuronales convencionales. Su principal ventaja es que puede lograr una precisión comparable o superior a la de las redes neuronales convencionales gracias a su estructura altamente flexible.
- **Autoencoders:** Modelos que trabajan automáticamente sobre grandes muestras de datos útiles para la reducción de dimensionalidad y la detección de anomalías c[29], [30].

Nota: algunos de estos algoritmos ya fueron explicados en detalle en el Capítulo 2. , por lo que aquí se presentan de forma resumida.

3.2 TÉCNICAS DE CLASIFICACIÓN UTILIZADAS

El objetivo principal de este trabajo es analizar diferentes técnicas de clasificación en *Machine Learning* para comprobar si, en la práctica, se cumplen las ventajas e inconvenientes teóricos que habitualmente se asocian a cada una de ellas. Para ello, se han seleccionado seis algoritmos representativos que abarcan desde enfoques clásicos hasta propuestas más recientes.

A continuación, en la Tabla 2, se resumen sus principales características, ventajas y limitaciones.

Tabla 2: Comparativa de ventajas e inconvenientes de las técnicas de clasificación utilizadas

Técnicas	***Ventajas***	***Desventajas***
SVM	- Alta capacidad de generalización - Aplicables a clasificación y regresión - Buena precisión - Flexibles y robustas frente a valores atípicos	- Poco eficientes en conjuntos de datos grandes - Difíciles de interpretar - Requiere preprocesamiento
Árboles de decisión	- Preprocesado de datos sencillo - Muy interpretables - Robustos frente a valores atípicos	- Propensos al *overfitting* - Alto coste computacional
Random Forest	- Alta precisión - Robustas frente a valores atípicos - Menos riesgo de *overfitting* - Útiles tanto para clasificación como regresión	- Coste computacional alto - Lentos - Menos interpretables
Regresión logística	- Fácil de implementar y entrenar - Fácil de interpretar - Rápida - Baja tendencia al *overfitting*	- Requiere linealidad entre variables dependientes e independientes - Dificulta la captura de relaciones complejas
MLP	- Buen rendimiento con datos complejos - Rápida en la predicción - Capacidad para aprender relaciones no lineales	- Alto coste computacional - Entrenamiento lento - Sensible al *overfitting*
KAN	- Alta precisión - Buena interpretabilidad - Bajo coste computacional - Aplicables a clasificación y regresión	- Entrenamiento lento - Requiere ajuste de muchos hiperparámetros - Sensibles al *overfitting* - Aún poco extendidas en aplicaciones generales

3.3 *TECNOLOGÍAS Y HERRAMIENTAS UTILIZADAS*

Para el desarrollo de este trabajo se ha empleado *Python* como lenguaje de programación, debido a su amplia adopción en la comunidad científica, la gran disponibilidad de recursos educativos y documentación online, la experiencia anterior con este lenguaje, y el gran número de librerías especializadas en ML disponibles, entre las que se encuentran scikit-learn, pandas, matplotlib o seaborn. Estas herramientas han facilitado el análisis y la visualización de datos, el desarrollo e implementación de modelos y la evaluación comparativa de los resultados obtenidos.

Python se caracteriza por ser un lenguaje eficiente, multiplataforma y de sintaxis sencilla, lo que lo convierte en una opción especialmente adecuada para proyectos académicos y prototipos experimentales. En este caso, el entorno de desarrollo elegido ha sido *Visual Studio Code*, tal y como se muestra en la *Figura 10*. Su sintaxis básica similar a la del inglés hace que sea un lenguaje fácilmente comprensible.mPor todo ello, se convierte en una opción especialmente adecuada para proyectos académicos y prototipos experimentales [31].

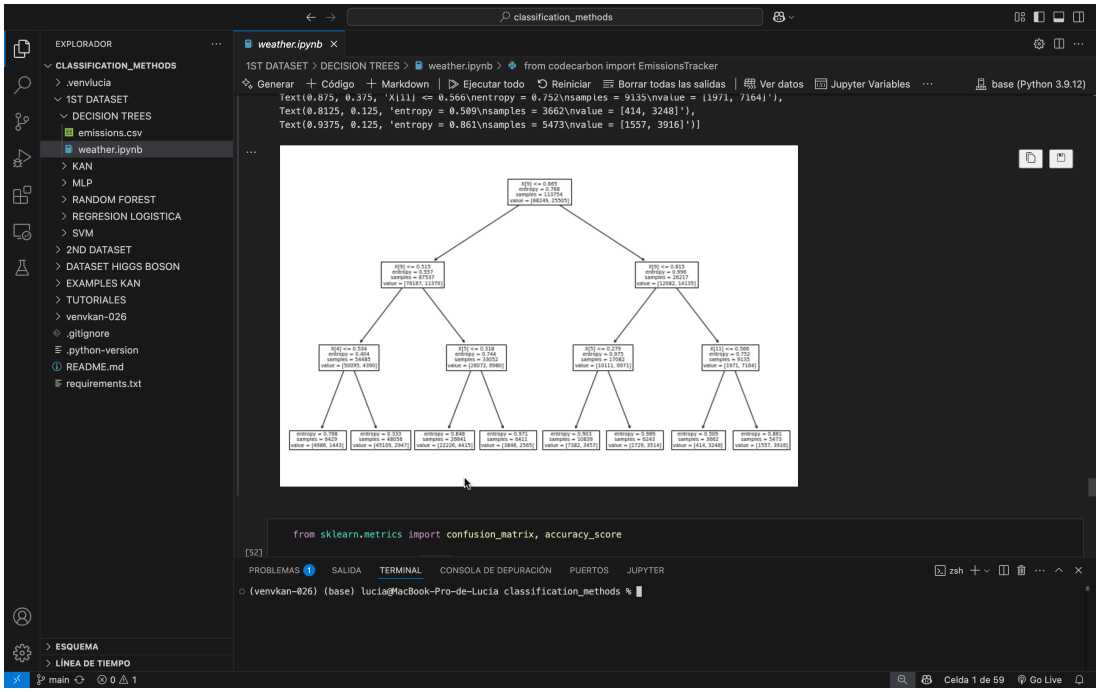

Figura 10: Entorno de desarrollo en Visual Studio Code utilizado durante el proyecto

Existen otras tecnologías que también permiten trabajar con algoritmos de ML como alternativas a *Python*. Una comparativa entre ellas puede observarse en la *Tabla 3*. Entre ellas, destacan:

- ***MATLAB***: Es una plataforma de programación y cálculo numérico utilizada por millones de ingenieros y científicos para analizar datos, desarrollar algoritmos y crear modelos. Es una herramienta fácil de utilizar, pero con limitaciones en cuanto a código abierto, flexibilidad y comunidad en comparación con *Python* [32].
- ***R***: Se trata de un lenguaje muy popular para realizar cálculos estadísticos y analizar grandes volúmenes de datos. Cuenta con numerosos paquetes para clasificación, aunque está menos orientado a tareas de ML [33].

Tabla 3: Comparativa entre tecnologías utilizadas para ML

Tecnología	***Código abierto***	***Facilidad de uso***	***Capacidades en ML***	***Comunidad***	***Capacidades de visualización***
Python	Sí	Alta	Sí	Muy activa	Sí
MATLAB	No	Alta	Limitado	Moderada	Sí
R	Sí	Media	Parcial	Activa	Sí

Además, se ha utilizado la web *Kaggle* para acceder a numerosas bases de datos con las que se ha trabajado a lo largo de todo el trabajo.

3.4 LIBRERÍAS EMPLEADAS EN PYTHON

Para implementar y comparar las distintas técnicas de clasificación, se ha utilizado Python como lenguaje de programación principal (véase Figura 11), dada su versatilidad y amplia adopción en la comunidad científica. Entre las librerías disponibles, se han seleccionado dos que permiten abordar tanto algoritmos tradicionales como propuestas más recientes: `scikit-learn`, para la mayoría de los métodos clásicos, y `pyKAN`, para las redes Kolmogorov–Arnold (KAN).

Figura 11: Logo oficial del lenguaje de programación Python

Dado que el objetivo principal de este trabajo es comparar distintas técnicas de clasificación poniendo especial énfasis en las KAN, se ha considerado oportuno describir en mayor detalle únicamente las librerías `scikit-learn` y `pyKAN`. La primera, por haber sido utilizada como base común para la implementación del resto de modelos, y la segunda, por su relevancia central en la parte experimental y por tratarse de una propuesta reciente con escasa documentación formal. El resto de librerías empleadas, aunque relevantes, se han usado de forma complementaria y se mencionan únicamente a nivel de soporte en la última sección.

3.4.1 SCIKIT-LEARN

`scikit-learn` (véase Figura 12) es una de las librerías más utilizadas en el ámbito del ML en Python. Proporciona herramientas eficientes y fáciles de usar para el preprocesamiento de datos, selección de modelos, evaluación y validación cruzada. En este trabajo, `scikit-learn` se ha utilizado para implementar los modelos de regresión logística, SVM, árboles de decisión, Random Forest y MLP, así como para el cálculo de métricas y el manejo de escalado o codificación de variables [34].

Figura 12: Logo oficial de la librería scikit-learn utilizada para implementar algoritmos clásicos de ML.

3.4.2 PYKAN

`pyKAN` es una librería desarrollada recientemente para entrenar redes Kolmogorov–Arnold (KAN). Esta librería no está incluida en los repositorios estándar de Python, por lo que fue necesario instalarla desde su repositorio oficial. A lo largo del proyecto, se han probado diferentes versiones de `pyKAN`, y se ha adaptado el código para mejorar la estabilidad y optimizar el entrenamiento de modelos. Su uso ha permitido explorar la capacidad de las KAN frente a otras técnicas más consolidadas [24].

3.4.3 RELACIÓN ENTRE TÉCNICAS Y LIBRERÍAS

Todas las técnicas de clasificación implementadas en este trabajo han sido desarrolladas utilizando la librería `scikit-learn`, especialmente en las fases iniciales y durante el trabajo con el conjunto de datos más simple. Sin embargo, al trabajar con un conjunto de datos más avanzado y de mayor volumen, se han incorporado otras librerías especializadas que permiten optimizar el rendimiento o disponer de funcionalidades más específicas.

En la Tabla 4 se muestra la relación entre cada técnica utilizada y las librerías de Python con las que se han implementado o podrían haberse implementado. A continuación, se describe brevemente cada una de estas librerías complementarias:

- `libsvm`: Es un software integrado para clasificación, regresión y estimación de distribuciones mediante máquinas de vectores soporte (SVM). Soporta clasificación multiclase y ofrece una interfaz simple. Incluye funcionalidades esenciales como

valiación cruzada (*cross-validation)* [35]. Especialmente eficiente para conjuntos de datos medianos y grandes.

- `XGBoost`***:*** `XGBoost` es una librería diseñada para ser altamente eficiente, flexible y portable al trabajar con árboles de decisión. Implementa algoritmos de ML dentro del marco del *gradient boosting* [36]. Esta técnica potencia el aprendizaje combinando múltiples modelos débiles para formar uno fuerte, mejorando progresivamente el rendimiento y reduciendo los errores [37].
- `LightGBM`***:*** Framework basado también en *gradient boosting* que utiliza árboles de decisión. Fue desarrollado por Microsoft y está diseñado para ser más rápido y con menor uso de memoria que `XGBoost`. Es capaz de manejar grandes volúmenes de datos y promete resultados más precisos [38].
- `CatBoost`***:*** Es una librería de código abierto desarrollada por *Yandex* que también se basa en *gradient boosting*. Su principal ventaja es que maneja de forma nativa variables categóricas, lo que la hace especialmente útil para conjuntos de datos con este tipo de variables [37].
- `statsmodels`***:*** Librería que proporciona clases y funciones para la estimación de diferentes modelos estadísticos, así como para realizar pruebas y análisis exploratorios. Cada estimador ofrece una amplia gama de resultados estadísticos que se han validado frente a paquetes estadísticos consolidados para garantizar su fiabilidad [39].
- `TensorFlow` ***y*** `Keras`***:*** Son bibliotecas de código abierto de Google. Están orientadas al desarrollo de redes neuronales profundas como las MLP, y son ampliamente utilizadas en aplicaciones de *Deep Learning* [40].
- `PyTorch`***:*** Es un framework de *Deep Learning* de código abierto que combina la biblioteca `Torch` con una API de alto nivel en Python. Destaca por su flexibilidad, facilidad de uso, y capacidad para ejecutar fragmentos de código en tiempo real, lo que permite realizar pruebas parciales sin necesidad de compilar toda la red [41].

Tabla 4: Relación entre técnicas de clasificación y librerías de Python comúnmente utilizadas.

Técnica	***Librería***
SVM	`scikit-learn`, `libsvm`
Árboles de decisión	`scikit-learn`, `XGBoost`, `LightGBM`, `CatBoost`
Random Forest	`scikit-learn`, `XGBoost`, `LightGBM`, `CatBoost`
Regresión logística	`scikit-learn`, `statsmodels`, `TensorFlow`
MLP	`scikit-learn`, `TensorFlow`, `Keras`, `PyTorch`
KAN	`scikit-learn`, `pyKAN`

3.5 INDICADORES DE EVALUACIÓN DEL RENDIMIENTO

Las diferentes técnicas de clasificación pueden evaluarse en base a varios criterios. No existe un único algoritmo superior en todos los contextos, ya que su rendimiento depende del problema específico que se desea resolver. A continuación, se nombran algunos de los criterios más relevantes para evaluar el rendimiento de los clasificadores [42]:

- **Exactitud general del modelo:** Es una de las métricas más utilizadas. Cuantifica la exactitud del modelo. Sin embargo, la precisión por sí sola ignora otros criterios, como la comprensibilidad o complejidad del modelo, por lo que puede no ser suficiente.
- **Comprensibilidad e interpretabilidad:** Es la combinación de varias métricas que describen la naturalidad de los clasificadores desde la perspectiva del usuario. Esto incluye qué tan fácil es entender los resultados del modelo y utilizar dicha información para la toma de decisiones. Por ejemplo, los árboles de decisión son métodos muy interpretables.

- **Complejidad:** Esto puede medirse en términos de tiempo necesario para entrenar el modelo, la complejidad computacional o el espacio de memoria para almacenar y operar con el modelo. Se centra en el modelo y sus características, como el número de parámetros o las operaciones.
- **Tiempo de ajuste de hiperparámetros y de entrenamiento**: Aquí se evalúa el tiempo en segundos para cada fase, importante en aplicaciones en tiempo real.
- **Consistencia:** Evalúa la capacidad del modelo para mantener un rendimiento constante en diferentes subconjuntos pertenecientes al conjunto completo de datos. Esto puede medirse analizando la desviación estándar de las métricas de precisión en las distintas divisiones de los datos.
- **Confianza:** Mide qué tan confiable es un modelo en términos de la calidad de sus predicciones. Esto puede medirse utilizando métricas como el error medio absoluto (MAE) o el error cuadrático medio (MSE).
- **Recursos utilizados:** Se refiere a la cantidad de recursos computacionales requeridos para entrenar y ejecutar el modelo, como la memoria RAM o el uso de CPU. Este criterio está estrechamente relacionado con la complejidad de un modelo, pero se centra más en el hadware que en el modelo.
- **Casos de uso:** Es importante situar el contexto del proyecto y el objetivo que se quiere conseguir, ya que los algoritmos tienen fortalezas y debilidades que deben ser tenidas en cuenta.

Estos criterios pueden evaluarse en base a algunos indicadores numéricos [43]:

- **Exactitud o *accuracy*:** Es la proporción de casos correctamente clasificados frente al total de los casos. Útil cuando las clases están balanceadas. Su fórmula es:

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

  Donde:

  - **TP (*True Positives*):** Casos positivos correctamente clasificados por el modelo.
  - **TN (*True Negatives*):** Casos negativos correctamente clasificados.

- **FP (*False Positives*):** Casos negativos clasificados incorrectamente como positivos.
- **FN (*False Negatives*):** Casos positivos clasificados incorrectamente como negativos.

La Tabla 5 muestra cómo se distribuyen las predicciones realizadas por un modelo clasificatorio según los valores reales y predichos, lo que da lugar a la conocida matriz de confusión.

Tabla 5: Distribución de predicciones en la matriz de confusión.

Predicción \ Realidad	***Positivo (1)***	***Negativo (0)***
Positivo (1)	TP	FP
Negativo (0)	FN	TN

- **Precisión**: Mide qué porcentaje de las predicciones positivas hechas por el modelo son correctas. Es útil cuando el coste de los falsos positivos es alto. Su fórmula es:

$$\text{Precisión} = \frac{TP}{TP + FP}$$

- **Sensibilidad o *recall*:** Mide qué porcentaje de los casos realmente positivos han sido identificados por el modelo. Es útil cuando el coste de los falsos negativos es alto. Su fórmula es:

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

- **F1-Score:** Es la media armónica de la precisión y la sensibilidad, varía entre 0 y 1, y es muy útil cuando se busca un balance entre ambas métricas. Es de gran ayuda cuando se trabaja con bases de datos desbalanceadas. Su fórmula es:

$$F_1 = \frac{2 \cdot \text{Precisión} \cdot Recall}{\text{Precisión} + Recall}$$

- **Curva ROC:** Es una herramienta gráfica que relaciona la sensibilidad con la tasa de falsos positivos en función de varios umbrales. La AUC (área bajo la curva ROC) es un valor entre 0 y 1 que evalúa el rendimiento general del modelo. En la Figura 13

se muestra un ejemplo de curva ROC obtenida a partir del conjunto de datos reducido utilizado en este trabajo. Su fórmula es:

$$\text{AUC} = \int_0^1 \text{TPR}(FPR)\ d(\text{FPR})$$

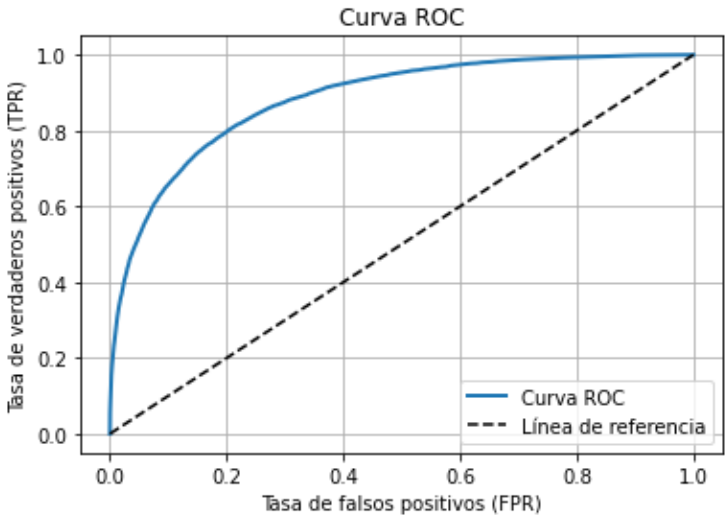

Figura 13: Curva ROC del modelo entrenado con el conjunto de datos reducido.

- **Matriz de confusión:** Se trata de una tabla que resume el número de predicciones correctas e incorrectas y proporciona información sobre el desempeño del modelo en la clasificación. Su estructura puede consultarse en la Tabla 5. Un ejemplo de matriz de confusión generado a partir del conjunto reducido utilizado en este trabajo puede verse en la Figura 14.
- **Uso energético:** Hace referencia a la cantidad de energía eléctrica consumida durante el proceso de entrenamiento del modelo. Este valor, generalmente expresado en vatios-hora (Wh), permite evaluar el impacto ambiental y la eficiencia computacional del algoritmo más allá del rendimiento predictivo. Para calcularlo, se ha utilizado la herramienta `codecarbon`, que estima el consumo energético en función del uso de CPU, memoria y duración del proceso.

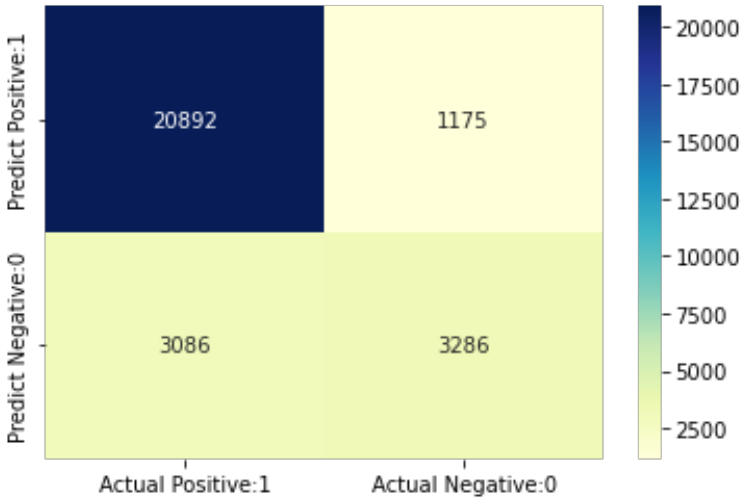

Figura 14: Matriz de confusión del modelo aplicado al conjunto de test (caso reducido).

Para el desarrollo de este trabajo se han seleccionado las siguientes métricas con el fin de comparar el rendimiento de los distintos métodos de *Machine Learning* entre sí:

- **Exactitud (*Accuracy*)**
- **Sensibilidad (*Recall*)**
- **F1-score**
- **Área bajo la curva ROC (AUC)**
- **Tiempo de ajuste de hiperparámetros**
- **Tiempo de ejecución**
- **Consumo energético**

Estas métricas permitirán evaluar tanto la calidad predictiva de los modelos como su eficiencia computacional, facilitando una comparación justa entre los diferentes enfoques implementados.

Capítulo 4. DEFINICIÓN DEL TRABAJO

4.1 JUSTIFICACIÓN

Las Kolmogorov-Arnold Networks (KAN) representan una novedosa y prometedora alternativa a las redes neuronales y otros métodos de clasificación tradicionales. A pesar de su reciente aparición, los primeros estudios y experimentos sugieren que las KAN podrían superar en diversos aspectos a los algoritmos convencionales, destacando especialmente por su interpretabilidad, capacidad de aproximación, eficiencia computacional y adaptabilidad.
Sin embargo, al tratarse de una tecnología en desarrollo, aún existen muchas incógnitas sobre su verdadero potencial y sus limitaciones en la práctica. Por ello, es necesario realizar un análisis sistemático que permita contrastar las ventajas teóricas que se les atribuyen frente a métodos consolidados como las Support Vector Machines (SVM), los árboles de decisión, Random Forest, la regresión logística o las redes neuronales multicapa (MLP). Dado que tanto las KAN como las MLP pertenecen a la familia de modelos neuronales, resulta especialmente interesante observar sus diferencias en la práctica.
Este proyecto surge con la intención de explorar e investigar el comportamiento de las KAN en tareas de clasificación, en comparación con otros algoritmos ampliamente utilizados en el ámbito del Machine Learning. Se pretende no solo comprobar si las ventajas teóricas de las KAN se reflejan en la práctica, sino también identificar en qué contextos pueden resultar más eficaces que otras técnicas, considerando tanto el rendimiento como los recursos computacionales empleados.

4.2 OBJETIVOS

Se pretende evaluar el rendimiento de las Kolmogorov-Arnold Networks (KAN) como alternativa a los algoritmos de clasificación convencionales. Se pretende evaluar si las ventajas teóricas atribuidas a las KAN se cumplen también en entornos prácticos.
Para alcanzar este objetivo general, se plantean los siguientes objetivos específicos.

- Estudiar, implementar y aplicar los principales algoritmos de clasificación supervisada, incluyendo Support Vector Machines (SVM), árboles de decisión, Random Forest, regresión logística, redes neuronales multicapa (MLP) y Kolmogorov–Arnold Networks (KAN), comenzando por un conjunto de datos sencillo que permita una primera comparación entre modelos en términos de precisión, sensibilidad, F1-score, AUC y eficiencia computacional.
- Analizar el comportamiento de los modelos en un conjunto de datos más complejo, que simule situaciones reales y exija una mayor capacidad de aprendizaje y generalización por parte de los clasificadores.
- Extraer conclusiones relevantes sobre el desempeño relativo de cada algoritmo, considerando tanto métricas de evaluación como aspectos prácticos.

4.3 METODOLOGÍA

La metodología seguida en este trabajo se resume en la *Figura 15*, que muestra las tres fases principales del proceso. En primer lugar, se realizó una investigación inicial sobre las *Kolmogorov-Arnold Networks* (KAN), motivada por su reciente aparición como técnica novedosa en el campo del aprendizaje automático, desarrolladas por el MIT en 2024 [26]. A partir de esta base, se amplió el estudio a otras técnicas de clasificación ampliamente utilizadas para comparar su comportamiento con respecto a las KAN.

En segundo lugar, se llevaron a cabo pequeños tutoriales de familiarización en *Python* para cada una de las técnicas consideradas. Esta primera toma de contacto permitió comprender su funcionamiento interno, los hiperparámetros a configurar y los requisitos previos de preprocesado para su correcta aplicación. Las técnicas exploradas en esta fase incluyen:

- Support Vector Machines (SVM)
- *Random Forest*
- *Árboles de Decisión (Decision Trees)*
- Regresión logística
- Redes neuronales multicapa (MLP)
- Kolmogorov-Arnold *Networks* (KAN)

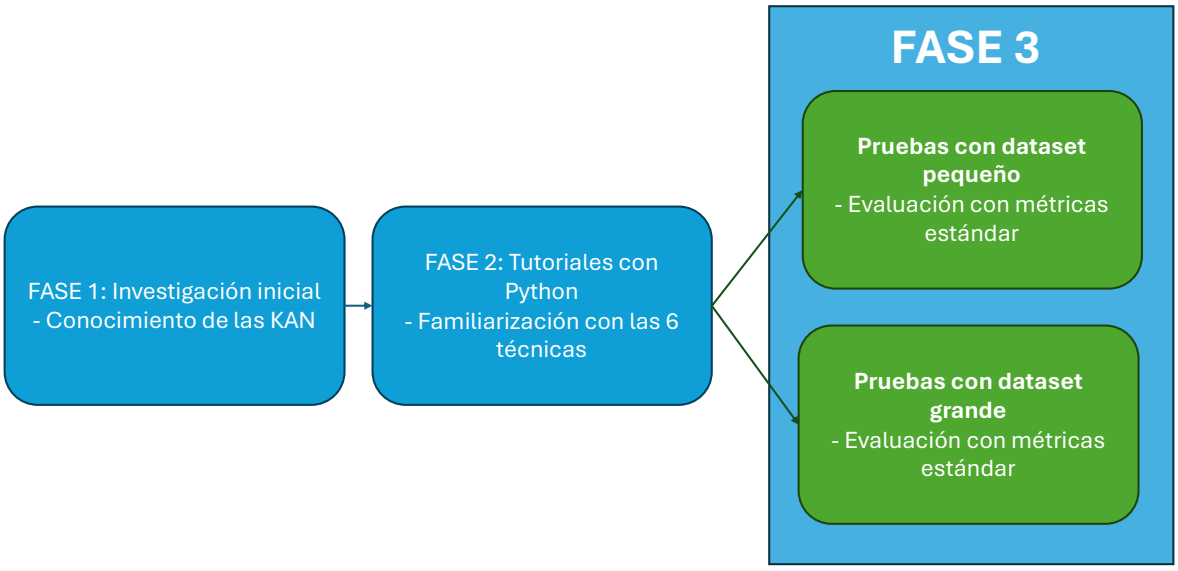

Figura 15: Esquema Metodológico del Trabajo

Para finalizar, se llevó a cabo una fase de pruebas compuesta por dos etapas diferenciadas. En la primera, se entrenaron y evaluaron todos los modelos sobre un conjunto de datos más pequeño y sencillo, con el fin de realizar un análisis preliminar. En la segunda, se repitió dicho proceso con un dataset más complejo y de mayor dimensión, lo que permitió evaluar el rendimiento de los algoritmos en un entorno más exigente y que refleja mejor la complejidad de los casos de uso reales. En ambas pruebas se utilizaron métricas estándar como la precisión o la F1-score para comparar los resultados con el objetivo de comprobar si las ventajas y desventajas teóricas de cada método se cumplían también en la práctica.

Para la implementación de los modelos y el análisis de resultados se han utilizado principalmente las siguientes **librerías** de Python:

- `pandas` y `numpy` para la manipulación y análisis de datos.
- `scikit-learn` para la implementación de modelos tradicionales (SVM, Árbol de Decisión, Random Forest, MLP, regresión logística) y métricas de evaluación.
- `matplotlib` y `seaborn` para la visualización de datos y resultados.
- `pyKAN` para la implementación de Kolmogorov–Arnold Networks.
- `sklearn.model_selection` para la validación cruzada y optimización mediante *GridSearchCV*.

Todas las pruebas se han realizado en Visual Studio Code como entorno de desarrollo.

4.4 PLANIFICACIÓN Y ESTIMACIÓN ECONÓMICA

El desarrollo del proyecto se ha planificado de forma escalonada a lo largo de varios meses, siguiendo una distribución temporal coherente con la carga académica y los hitos marcados. Esta planificación ha permitido avanzar de manera progresiva desde la fase de documentación hasta el análisis final de los modelos.

En las primeras etapas, se ha dedicado tiempo tanto al estado del arte como a la realización de diversos tutoriales para familiarizarse con las distintas técnicas de clasificación, el uso de librerías específicas y el preprocesamiento de datos. Posteriormente, se ha trabajado con algoritmos clásicos y con KAN sobre conjuntos de datos sencillos, y finalmente con datasets más complejos para evaluar el comportamiento de los modelos en escenarios exigentes.

Las principales tareas del proyecto y su distribución temporal se recogen en la Tabla 6:

Tabla 6: Planificación temporal del proyecto (diagrama de Gantt)

Tarea	*Sept*	*Oct*	*Nov*	*Dic*	*Ene*	*Feb*	*Mar*	*Abr*	*May*
Estado del arte, documentación inicial y realización de tutoriales prácticos									
Aplicación de modelos clásicos y KAN a conjuntos de datos sencillos									
Aplicación de los algoritmos a datasets más complejos y comparación de resultados									

Desde el punto de vista económico, el proyecto no ha requerido ninguna inversión monetaria directa, al haberse desarrollado exclusivamente con herramientas de software libre y equipos personales. El trabajo se ha llevado a cabo en un entorno de desarrollo local, sin depender de recursos externos de pago.

Capítulo 5. DESCRIPCIÓN TÉCNICA DE LOS MODELOS KAN

En este capítulo se presenta una descripción detallada de las Kolmogorov-Arnold Networks (KAN). Para comprender el funcionamiento y el potencial de esta arquitectura, se abordarán tanto sus fundamentos teóricos como su estructura técnica, incluyendo su implementación práctica en Python mediante la librería `pyKAN`. Además, se explorarán sus características diferenciales, como la interpretabilidad.

5.1 ¿QUÉ SON LAS KOLMOGOROV-ARNOLD NETWORKS?

Las Kolmogorov-Arnold Networks (KAN) son una novedosa y prometedora arquitectura de red neuronal desarrollada en el MIT, inspirada por el teorema de representación de Kolmogorov-Arnold, en lugar del teorema universal de aproximación que utilizan las redes neuronales multicapa (MLP). Una de sus principales innovaciones es la introducción de funciones de activación en los bordes entre neuronas, en lugar de dentro de las propias neuronas [44].

Las KAN pretenden ayudar a los científicos en campos como la física y la inteligencia artificial, y su nombre rinde homenaje a los matemáticos soviéticos Andrey Kolmogorov y Vladimir Arnold (véase Figura 16).

A diferencia de las redes neuronales, las KAN prometen una mayor transparencia en la forma en que alcanzan sus resultados. Esto resuelve uno de los principales problemas de los modelos actuales: su escasa interpretabilidad y su naturaleza de "caja negra" [44]. Las KAN pueden visualizarse con facilidad y su estructura permite la interacción con usuarios humanos.

Otro aspecto destacado es su precisión. Se ha demostrado que redes KAN más pequeñas pueden alcanzar una exactitud comparable o incluso superior a la de MLP más grandes en

tareas de ajuste de funciones. Además, tanto desde el punto de vista teórico como empírico, las KAN presentan leyes de escalado neuronal más rápidas que las MLP [20].

Más adelante, en la sección 5.6, se compararán en mayor profundidad las KAN y las MLP.

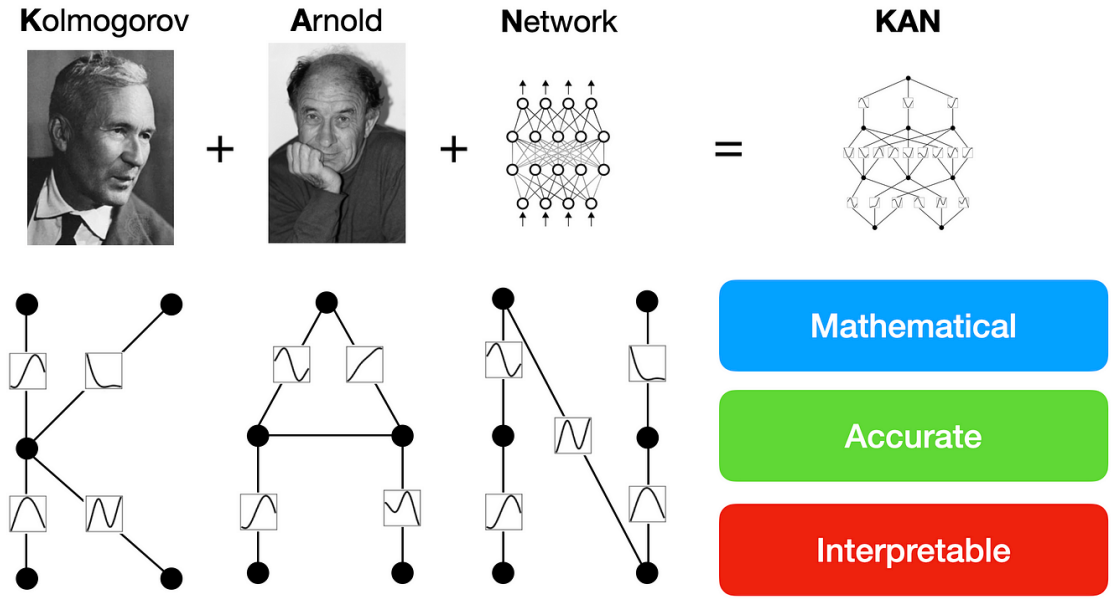

Figura 16: Origen y Principios de las Kolmogorov-Arnold Networks (KAN)[20]

5.2 FUNDAMENTOS MATEMÁTICOS

5.2.1 TEOREMA DE REPRESENTACIÓN DE KOLMOGOROV-ARNOLD

Como se explicó anteriormente, las KAN utilizan el **teorema de representación de Kolmogorov-Arnold**. De acuerdo con este teorema, cualquier función multivariable f se puede expresar como una suma finita de funciones continuas de una sola variable. Este teorema puede expresarse de la siguiente forma:

$$f(x_1, \dots, x_n) = \sum_{q=1}^{2n+1} \Phi_q \left(\sum_{p=1}^{n} \phi_{q,p}(x_p) \right)$$

Donde:

- $f(x_1, \dots, x_n)$ es la función multivariable.

- $\phi_{q,p}(x_p)$ son funciones univariables, con dominio en $[0,1]$, es decir: $(\varphi_{q,p}: [0,1] \rightarrow R)$.
- $(\Phi_q: R \rightarrow R)$ es la función que combina las salidas univariables

De esta forma, puede entenderse cómo problemas más complejos pueden afrontarse en problemas más pequeños, haciendo el proceso más fácil de gestionar [21]. Modelos actuales como las MLP suelen dar problemas cuando se trabaja con datos con muchas dimensiones. Este problema se conoce como la maldición de la dimensionalidad.

En lugar de aproximar directamente toda una función compleja, como hacen la mayoría de los demás modelos, las KAN se centran en aprender estas funciones univariables más sencillas. Este enfoque da como resultado un modelo flexible, interpretable, y más sencillo al reducirse el número de parámetros, sobre todo cuando las relaciones en los datos son no lineales [44]. Además, las KAN se diferencian de las MLP porque reemplazan las funciones de activación fijas por funciones que se pueden entrenar, eliminando la necesidad de matrices de pesos lineares [45].

5.2.2 FUNCIONES B-SPLINE

Formalmente, las ***B-splines*** son un método avanzado de ajuste de curvas y un tipo específico de *spline*, un término matemático que hace referencia a una función flexible y polinómica por tramos, utilizada para definir una curva suave a través de una serie de puntos. De manera informal, podría explicarse con el siguiente ejemplo. Si se tuviera una serie de puntos en un gráfico para representar cómo han fluctuado los gastos en los últimos 10 meses y se quisiera tener una línea que mostrara cómo dichos gastos han ido variando, podría utilizarse un ajuste polinomial, como puede verse en la Figura 17.

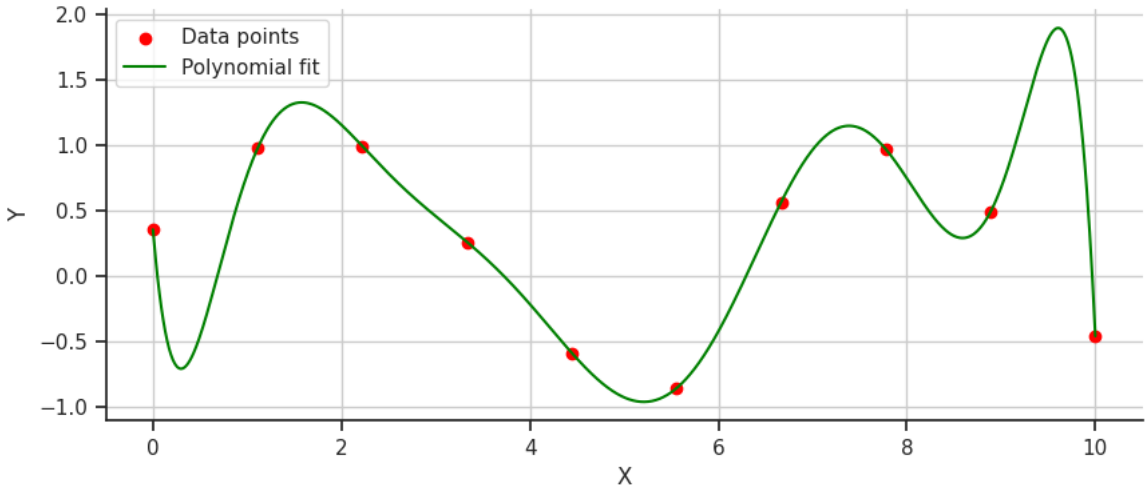

Figura 17: Ajuste polinomial de datos [21].

Si se observa con atención la figura, puede comprobarse que después del primer punto ocurre una bajada drástica de la línea antes de llegar al segundo punto. Este es uno de los problemas del ajuste polinomial, su tendencia a las oscilaciones bruscas. Dicho problema se conoce como el fenómeno de Runge.

Utilizando *splines*, se puede conseguir un mejor ajuste de esta línea. Un *spline* divide los datos en segmentos y ajusta polinomios individuales a cada uno, como se muestra en la Figura 18.

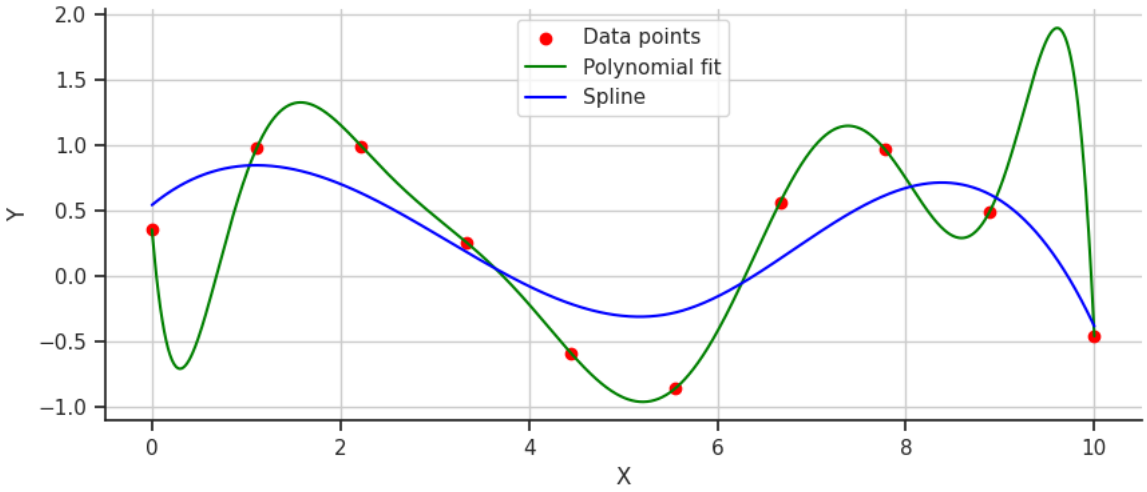

Figura 18: Ajuste polinomial y con spline de datos [21].

Esta aproximación es mucho más suave, pero quizás no se ajusta lo suficiente a los datos (*underfitting*). Las *B-splines* pueden solucionar este problema. Las *B-splines* son un tipo de *spline* que usa puntos de control para controlar la forma de la curva y guiar a los polinomios hacia un mejor ajuste, ofreciendo una solución más precisa, como se aprecia en la Figura 19.

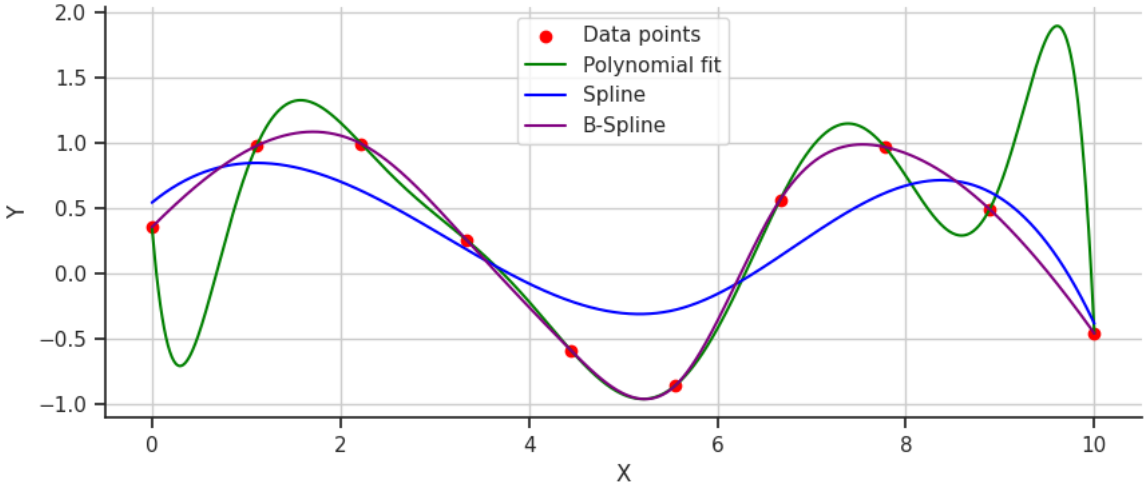

Figura 19: Ajuste polinomial, con spline y con B-spline de datos [21].

La *B-spline* no presenta oscilaciones bruscas ni *underfitting*. Al contrario, captura los datos a la perfección. Las *B-splines* proporcionan una suavidad superior y una exactitud crucial para modelar funciones complejas. Su flexibilidad les permite adaptarse y modelar

relaciones complejas en los datos ajustando su forma para minimizar el error de aproximación, mejorando la capacidad de la red para aprender patrones sutiles en conjuntos de datos de muchas dimensiones. Pueden adaptarse a cambios en patrones de datos fácilmente sin reentrenar completamente el modelo, lo que las hace una herramienta versátil y robusta para el ajuste de datos [21].

Un resumen de las características de las tres técnicas de ajuste mencionadas en este apartado puede verse en la Tabla 7.

Tabla 7: Comparativa entre técnicas de ajuste de curvas en términos de suavidad, precisión e interpretabilidad.

Técnica de ajuste	***Suavidad***	***Precisión***	***Oscilaciones***	***Interpretabilidad***
Ajuste polinomial	Baja	Baja	Alta	Alta
Spline	Media	Media	Media	Media
B-spline	Alta	Alta	Baja	Alta

Matemáticamente, una *B-spline* puede definirse como:

$$\mathcal{C}(t) = \sum_{i=0}^{n} P_i N_{i,k}(t)$$

Donde:

- P_i son los puntos de control: los coeficientes que se optimizan durante el entrenamiento para minimizar la función de perdida que mide la diferencia entre las predicciones y los valores reales.
- $N_{i,k}(t)$ son las funciones base *B-spline* de grado (k), definidas sobre una cuadrícula.
- t representa el vector de nodos (o "*knots*") que define los intervalos del dominio.

En conclusión, debido a que las KAN tienen funciones de activación en los extremos de la red, cada parámetro de peso en una KAN se reemplaza por una función univariable parametrizada como una *B-spline*, convirtiéndose en una red flexible y capaz de modelar

funciones complejas con potencialmente menos parámetros y una interpretabilidad mejorada (véase Figura 20) [45].

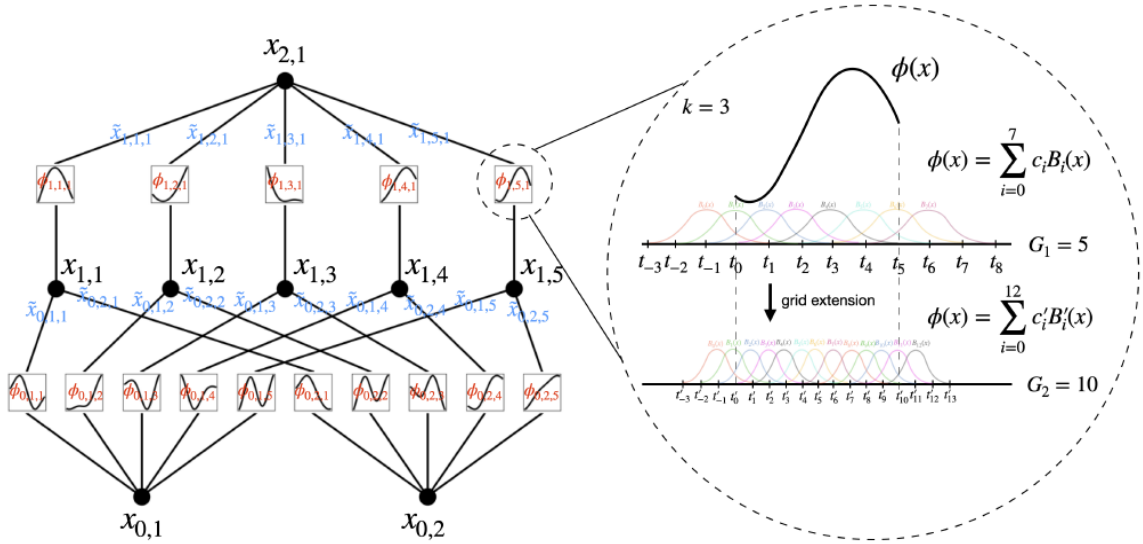

Figura 20: Las KAN se aprovechan de la estructura de las MLP mientras se benefician de las B-splines [45].

5.3 ARQUITECTURA Y FUNCIONAMIENTO INTERNO

Las Kolmogorov-Arnold Networks (KAN) suponen un avance significativo en el diseño de redes neuronales al combinar el teorema de representación de Kolmogorov-Arnold (KAR) con funciones *B-spline*, dando lugar a un modelo dinámico y potente. El teorema KAR permite descomponer funciones complejas en otras más simples. Las KAN aplican este principio en cada uno de los bordes de la red, de modo que cada conexión entre neuronas funcione como una función de activación *B-spline* entrenable. Esto permite que cada borde aprenda con precisión la parte específica de los datos que le corresponde. Un ejemplo de la arquitectura de las KAN puede observarse en la Figura 21.

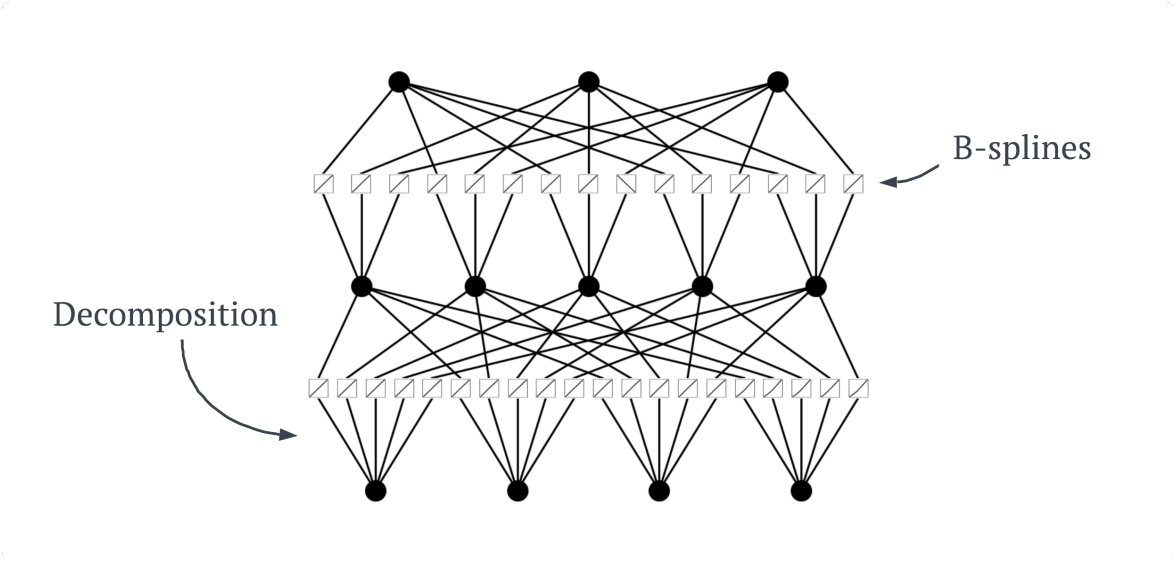

Figura 21: Ejemplo de arquitectura KAN [21].

A medida que las KAN se entrenan, cada *B-spline* ajusta sus puntos de control P_i mediante un proceso conocido como *backpropagation*, común en el entrenamiento de redes neuronales, pero que en este contexto adquiere una nueva dimensión. Este proceso adaptativo permite que las KAN refinen progresivamente su forma de interpretar los datos en cada iteración de entrenamiento, mejorando continuamente su precisión y eficiencia [21]. Esta mejora se lleva a cabo mediante técnicas como el descenso de gradiente, actualizando los parámetros de cada *spline* en cada iteración para reducir el error de predicción [45].

La arquitectura de las KAN se basa en un concepto novedoso: sustituir los pesos tradicionales por parámetros de funciones univariables en los bordes de la red. Cada nodo en una KAN suma las salidas de estas funciones sin aplicar ninguna transformación no lineal, a diferencia de las MLP, que combinan transformaciones lineales seguidas de funciones de activación no lineales. La diferencia se muestra en la siguiente expresión:

$$\text{KAN}(x) = (\Phi_{L-1} \circ \Phi_{L-2} \circ \cdots \circ \Phi_1 \circ \Phi_0)x$$

$$\text{MLP}(x) = (W_{L-1} \circ \sigma \circ W_{L-2} \circ \sigma \circ \cdots \circ W_1 \circ \sigma \circ W_0)x$$

Los pesos lineales $\boldsymbol{W}$ son sustituidos gracias al uso de funciones *B-spline*, como se explicó en la sección 5.2.2. La Figura 22 muestra la diferencia entre ambas arquitecturas.

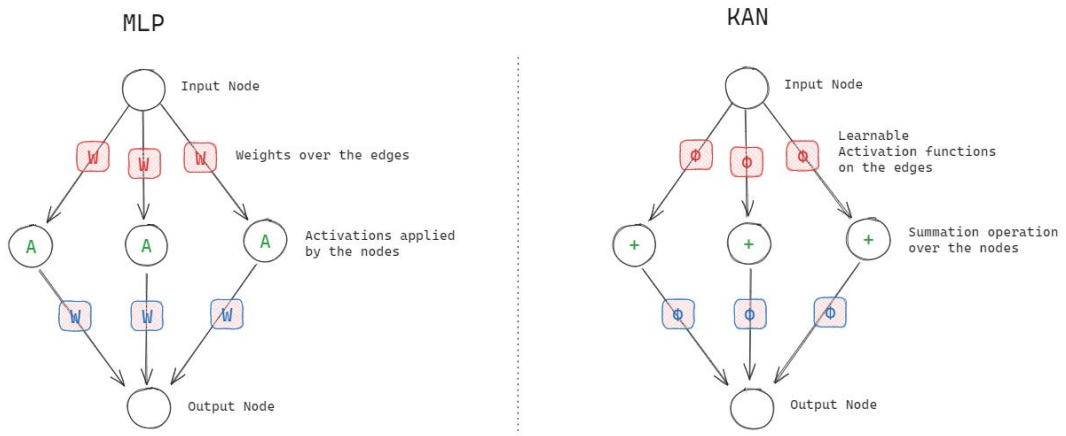

Figura 22: Comparación estructural entre una red MLP y una red KAN [46].

La arquitectura de una KAN también puede escribirse de forma matricial como:

$$f(x) = \Phi_{\text{out}} \circ \Phi_{\text{in}} \circ x$$

Donde:

- $\Phi_{\text{in}} = \begin{pmatrix} \varphi_{1,1}(\cdot) & \cdots & \varphi_{1,n}(\cdot) \\ \vdots & \ddots & \vdots \\ \varphi_{2n+1,1}(\cdot) & \cdots & \varphi_{2n+1,n}(\cdot) \end{pmatrix}$
- $\Phi_{\text{out}} = \begin{pmatrix} \Phi_1(\cdot) & \cdots & \Phi_{2n+1}(\cdot) \end{pmatrix}$

Tanto Φ_{in} y Φ_{out} son casos particulares de una capa de Kolmogorov-Arnold, definida como:

$$\Phi = \begin{pmatrix} \varphi_{1,1}(\cdot) & \cdots & \varphi_{1,n_{\text{in}}}(\cdot) \\ \vdots & \ddots & \vdots \\ \varphi_{n_{\text{out}},1}(\cdot) & \cdots & \varphi_{n_{\text{out}},n_{\text{in}}}(\cdot) \end{pmatrix}$$

Donde:

- Φ_{in} corresponde a $n_{\text{in}} = n, n_{\text{out}} = 2n + 1$
- Φ_{out} corresponde a $n_{\text{in}} = 2n + 1, n_{\text{out}} = 1$

Una KAN completa se construye simplemente apilando capas como las anteriores. Esto lleva de nuevo a la formulación general mostrada anteriormente. Además, las KAN pueden visualizarse fácilmente, ya que cada capa equivale a una capa completamente conectada en la que cada conexión contiene una función univariable entrenable en su borde [47].

5.4 IMPLEMENTACIÓN EN PYKAN

`pyKAN` es la librería de Python que permite trabajar con Kolmogorov-Arnold Networks (KAN), desarrollada por Ziming Liu. En esta sección se describen en profundidad las principales opciones, parámetros e hiperparámetros que pueden configurarse al entrenar un modelo KAN utilizando dicha librería.

5.4.1 CARACTERÍSTICAS DEL MODELO

Para construir una KAN, se deben definir primero sus características estructurales. A continuación, se muestra un ejemplo:

```
model = KAN(width=[4,5,3], grid=5, k=3, seed=0, device=cpu)
```

En primer lugar, se especifica la estructura de la red mediante el parámetro `width`. En este caso, se ha definido una KAN con 4 nodos de entrada, 5 nodos en una capa oculta, y 3 nodos de salida. En la Figura 23 se muestra una representación de una KAN con esta configuración. Se trata de un modelo de pequeño tamaño, en el que cada borde entre nodos contiene una *B-spline* inicializada. En este ejemplo, las cuatro variables situadas en la parte inferior representan las entradas del conjunto de datos, mientras que las tres variables superiores representan las posibles clases de salida [21].

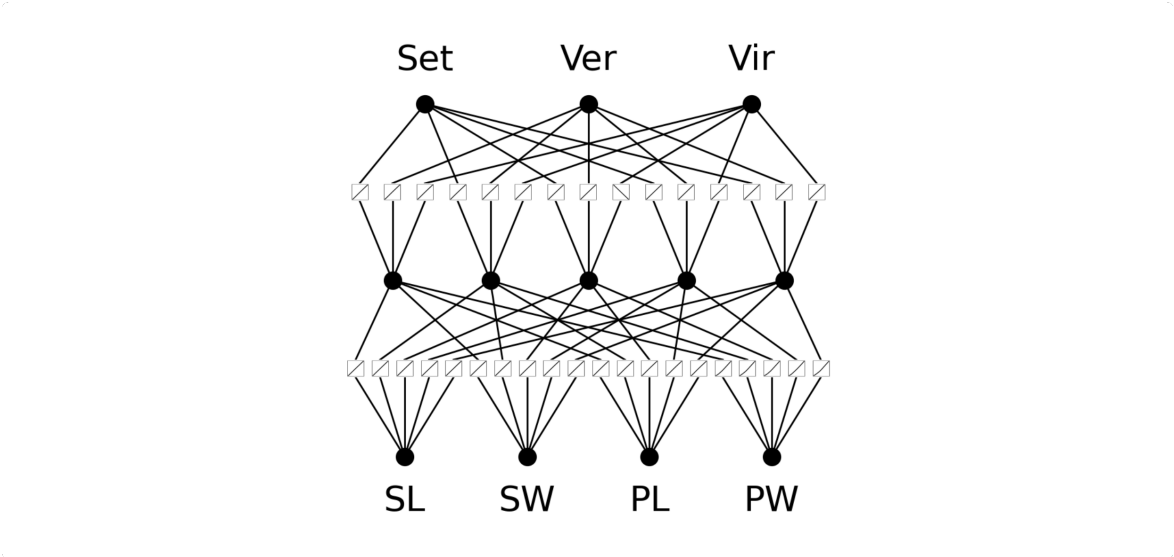

Figura 23: Ejemplo de KAN con 4 entradas, 3 salidas y una capa oculta con 5 nodos [21].

El siguiente parámetro es `grid`. Este parámetro es especialmente relevante, ya que afecta directamente a la capacidad de ajuste del modelo. Las funciones base *B-spline* se definen sobre una cuadrícula (`grid`). Los puntos de esta cuadrícula determinan los intervalos en los que cada función base está activa, influyendo así en la forma y suavidad de la *spline*. Cuanto mayor sea el número de intervalos definidos por el `grid`, mayor será la capacidad del modelo para capturar detalles en los datos. En este ejemplo, se han definido 5 intervalos de cuadrícula [45].

El parámetro `k` indica el orden de los polinomios que conforman las *B-splines*. En este caso, se ha utilizado el orden 3, un valor común en modelos KAN. No obstante, este parámetro puede aumentarse para incrementar la suavidad de las funciones. Sin embargo, valores de `k` demasiado altos pueden provocar oscilaciones excesivas, dificultando el proceso de optimización. Este parámetro no afecta significativamente a la interpretabilidad del modelo [20].

El parámetro `seed` es un hiperparámetro de control de aleatoriedad. En el entrenamiento de modelos de Machine Learning, procesos como la división en conjuntos de *train* y *test* o la inicialización de parámetros suelen incluir elementos aleatorios. Establecer una semilla (`seed`) permite obtener resultados reproducibles, lo que resulta fundamental para poder comparar modelos de forma justa. En este ejemplo, se ha utilizado una semilla con valor 0.

Es importante tener en cuenta que los resultados pueden variar significativamente al modificar este parámetro [20].

En la Figura 24 se observa la diferencia en los resultados obtenidos al entrenar el mismo modelo con `seed=1` y con `seed=42`.

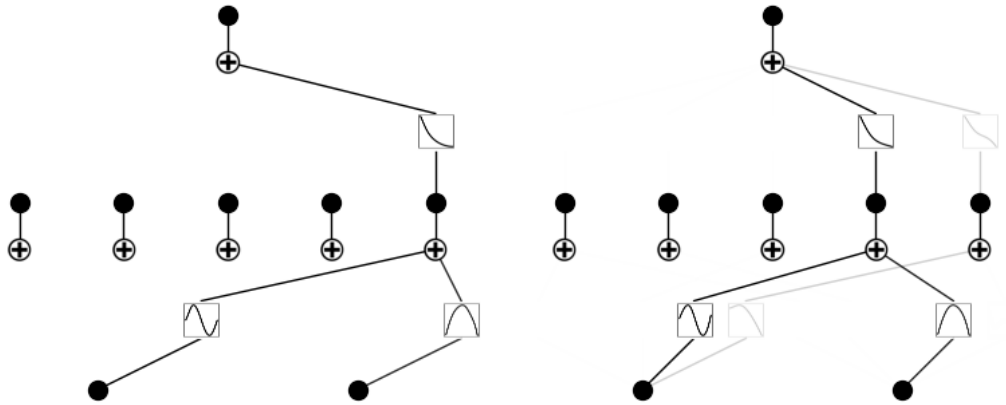

Figura 24: Diferencia en los resultados obtenidos utilizando `seed=1` y `seed=42` en una KAN [48].

Por último, el parámetro `device` indica el entorno en el que se ejecutará el modelo. Puede establecerse como `'cpu'` para entrenar en el procesador (más lento, pero siempre disponible), o como `'cuda'` para utilizar una GPU compatible con CUDA, lo que permite acelerar significativamente el entrenamiento en modelos de mayor tamaño [48].

Un resumen de los parámetros utilizados para construir el modelo KAN presentado en esta sección se recoge en la Tabla 8.

Tabla 8: Resumen de parámetros definidos para la construcción del modelo KAN.

Parámetro	***Descripción***	***Valor usado en el ejemplo***
`width`	Estructura de la red (entradas, ocultas, salidas)	`[4,5,3]`
`Grid`	Número de intervalos en la cuadrícula spline	`5`
`k`	Orden del polinomio de las B-splines	`3`
`seed`	Control de aleatoriedad	`0`
`device`	Entorno de ejecución	`'cpu'`

5.4.2 ENTRENAMIENTO

Una vez construido el modelo, el siguiente paso es entrenarlo. Este proceso se asemeja al entrenamiento de una red neuronal estándar, ya que también se emplean un optimizador, una función de pérdida y un número de pasos de entrenamiento (*"epochs"*). No obstante, las KAN incorporan además unos parámetros de penalización únicos. Estos parámetros e hiperparámetros se explican detalladamente en esta sección. Para ello, se va a utilizar como ejemplo el código de entrenamiento del modelo de la sección 5.4.1 [21]:

```
results = model.fit(iris_dataset, opt="Adam",
      metrics=(train_acc, test_acc),
      loss_fn=torch.nn.CrossEntropyLoss(),
      steps=100, lamb=0.01, lamb_entropy=10)
```

En primer lugar, una vez se ha definido el conjunto de datos con el que se va a trabajar, se debe seleccionar el optimizador del modelo. Existen dos opciones [48]:

- `"LBFGS"`: Implementa el método de Memoria Limitada de Broyden, Fletcher, Goldfarb y Shanno. Se trata de un algoritmo de optimización de segundo orden perteneciente a la clase de métodos Quasi-Newton. Este método aproxima la segunda derivada en aquellos problemas en los que no se puede calcular directamente. Su

principal ventaja es que evita el cálculo explícito del Hessiano completo, utilizando menos memoria y siendo más eficiente que otros métodos de segundo orden [49].

- `"Adam"`: Es la opción escogida en este ejemplo. Implementa Adam, un algoritmo para la optimización de funciones objetivo-estocásticas basado en el gradiente de primer orden, que utiliza estimaciones adaptativas de momentos de orden inferior. El método es sencillo de implementar, computacionalmente eficiente, requiere poca memoria y es especialmente adecuado para problemas de gran escala, tanto en datos como en número de parámetros [50].

A continuación, se debe seleccionar una lista de métricas (`metrics`) a calcular en el entrenamiento definidas como funciones. Lo más común es definir una función para calcular la *accuracy* en el entrenamiento y en el *test*, como se muestra en el ejemplo [48].

El parámetro `loss_fn` corresponde a la función pérdida utilizada en el entrenamiento del modelo. En este caso, se ha seleccionado `CrossEntropyLoss`, ya que se está llevando a cabo una clasificación multiclase. Existe una amplia variedad de funciones pérdida ofrecidas en la clase `torch.nn.Module` (véase Tabla 9), aunque también puede definirse una función de pérdida personalizada, implementándola como una subclase una clase de `torch.nn.Module` o como una función que retorne un `torch.Tensor`.

Tabla 9: Funciones de pérdida disponibles en PyTorch [51].

Función pérdida	***Descripción***
`nn.L1Loss`	Crea un criterio que mide el error absoluto medio (MAE) entre cada elemento de la entrada `x` y el objetivo `y`.
`nn.MSELoss`	Crea un criterio que mide el error cuadrático medio entre cada elemento de la entrada `x` y el objetivo `y`.
`nn.CrossEntropyLoss`	Calcula la pérdida de entropía cruzada entre los *logits* de entrada y las etiquetas objetivo.
`nn.CTCLoss`	Pérdida de Clasificación Temporal Conectiva (*Connectionist Temporal Classification*).
`nn.NLLLoss`	Pérdida de log-verosimilitud negativa (*Negative Log Likelihood*).
`nn.PoissonNLLLoss`	Pérdida de log-verosimilitud negativa bajo la suposición de que el objetivo sigue una distribución de Poisson.
`nn.GaussianNLLLoss`	Pérdida de log-verosimilitud negativa con distribución gaussiana.
`nn.KLDivLoss`	Pérdida por divergencia de Kullback-Leibler.
`nn.BCELoss`	Crea un criterio que mide la entropía cruzada binaria entre el objetivo y las probabilidades de entrada.
`nn.BCEWithLogitsLoss`	Combina una capa sigmoide y la BCELoss en una sola clase.
`nn.MarginRankingLoss`	Crea un criterio que mide la pérdida dada dos tensores de entrada `x1`, `x2` y un tensor de etiquetas `y` (conteniendo 1 o -1).
`nn.HingeEmbeddingLoss`	Mide la pérdida dada una entrada `x` y un tensor de etiquetas `y` (conteniendo 1 o -1).
`nn.MultiLabelMarginLoss`	Optimiza una pérdida tipo *hinge* para clasificación multiclase y *multilabel*, entre una entrada `x` (tensor 2D) y una salida `y` (tensor 2D con los índices de clase objetivo).

`nn.HuberLoss`	Usa un término cuadrático si el error absoluto está por debajo de un umbral `delta`, y un término lineal escalado por `delta` en caso contrario.
`nn.SmoothL1Loss`	Usa un término cuadrático si el error absoluto está por debajo de `beta`, y un término L1 en caso contrario.
`nn.SoftMarginLoss`	Optimiza una pérdida logística para clasificación binaria entre una entrada `x` y un objetivo `y` (conteniendo 1 o -1).
`nn.MultiLabelSoftMarginLoss`	Optimiza una pérdida tipo "uno contra todos" basada en entropía máxima, entre `x` y `y` de tamaño `(N, C)`.
`nn.CosineEmbeddingLoss`	Mide la pérdida dada dos tensores de entrada `x1`, `x2`, y un tensor de etiquetas `y` con valores 1 o -1, utilizando la similitud del coseno.
`nn.MultiMarginLoss`	Optimiza una pérdida tipo *hinge* multiclase (basada en márgenes) entre una entrada `x` (tensor 2D) y una salida `y` (tensor 1D con los índices de clase).
`nn.TripletMarginLoss`	Mide la pérdida tripleta dada los tensores de entrada `x1`, `x2`, `x3` y un margen mayor que 0.
`nn.TripletMarginWithDistanceLoss`	Mide la pérdida tripleta dada los tensores de entrada `a`, `p` y `n` (ancla, positivo y negativo, respectivamente), utilizando una función de distancia no negativa y real.

El parámetro `steps` se refiere al número de iteraciones de entrenamiento o pasos de optimización que se van a realizar sobre los datos. Es equivalente al número de *epochs* en otros *frameworks*. Controla cuántas veces se actualizan los parámetros del modelo durante el entrenamiento. Cada paso implica calcular la función de pérdida, obtener los gradientes y ajustar las funciones *B-spline* en consecuencia. A mayor número de pasos, el modelo puede aprender más patrones de los datos, pero también existe el riesgo de *overfitting* si se excede. En este caso, la KAN se entrena en 100 pasos.

El parámetro `lamb` (λ) es la intensidad global de la penalización. Controla la *sparsity* y la regularización. La *sparsity* hace referencia a la capacidad del modelo para mantener la

mayoría de sus funciones base sin una contribución significativa a la salida del modelo, lo que reduce la complejidad del modelo y mejora su interpretabilidad, al centrarse en las funciones más relevantes. Este parámetro ayuda a evitar la redundancia funcional, es decir, que varias funciones base aporten la misma información. En este caso, la regularización es de 0.01. Utilizar un modelo sin regularización tiene el riesgo de ofrecer una KAN con poca interpretabilidad, por lo que es recomendable aumentar gradualmente este parámetro. En la Figura 25 se observa la diferencia entre el mismo modelo KAN sin regularización (`lamb=0`) y con un valor de `lamb=0.01`. Del mismo modo, en la Figura 26 se aprecia la diferencia entre el mismo modelo KAN sin regularización (`lamb=0`) y con un valor de `lamb=1` [24], [48].

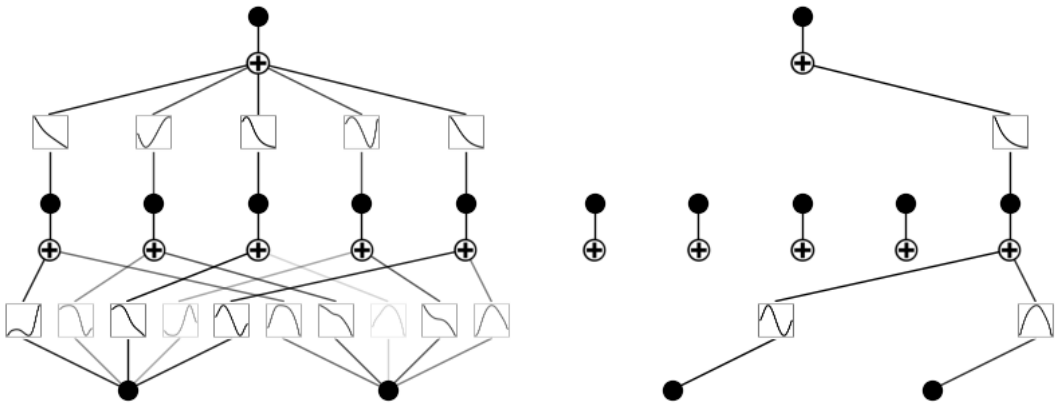

Figura 25: Diferencia entre un modelo KAN sin regularización (izquierda) y con un valor de `lamb=0.01` (derecha) [48].

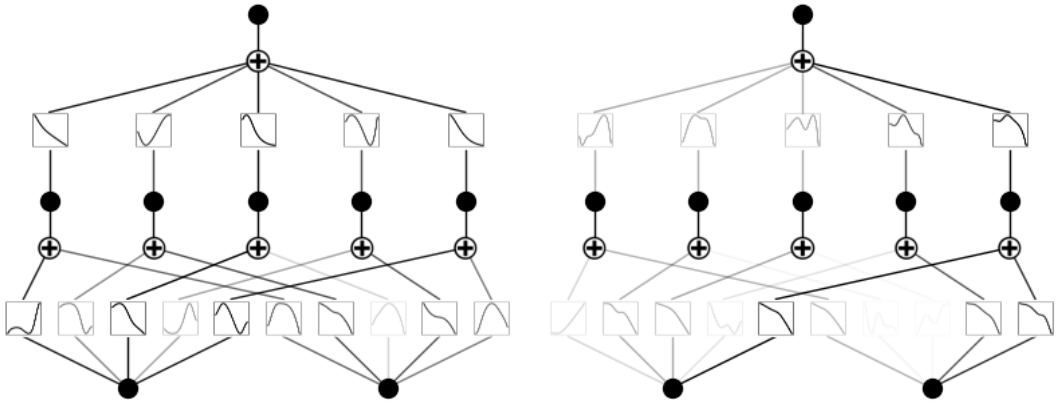

Figura 26: Diferencia entre un modelo KAN sin regularización (izquierda) y un valor de lamb=1 (derecha) [48].

Finalmente, el parámetro `lamb_entropy` se refiere a la penalización relativa por entropía. En este caso, el valor escogido ha sido 10. Este parámetro regula cuánto se penaliza que las funciones sean muy impredecibles o desordenadas. Una entropía muy alta puede indicar que las funciones están oscilando en exceso o que presentan una complejidad innecesaria. Esto puede comprometer tanto la estabilidad como la interpretabilidad del modelo. Al introducir esta penalización, se incentiva que el modelo mantenga funciones más suaves y estables. Un valor alto de este parámetro fuerza al modelo a aprender funciones más simples, mientras que un valor bajo permite mayor libertad y complejidad en las funciones. En la Figura 27 puede observarse la diferencia entre entrenar el mismo modelo con un valor `lamb_entropy=0` y un valor `lamb_entropy=10`.

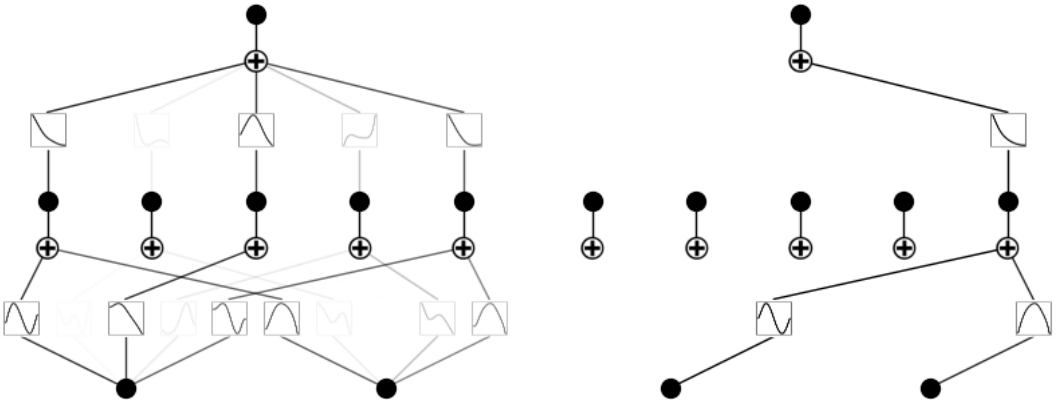

Figura 27: Diferencia entre un modelo KAN entrenado con un valor `lamb_entropy=0` *(izquierda) y un valor* `lamb_entropy=10` *(derecha) [48].*

Durante las primeras iteraciones del entrenamiento, las *B-splines* experimentan transformaciones importantes, hasta converger en una configuración estable. El modelo KAN tras el entrenamiento puede observarse en la Figura 28. Algunos de los extremos han desaparecido completamente. Esto se debe a que las neuronas que tienen una actividad por debajo de cierto umbral se apagan por completo para hacer la red más eficiente [21].

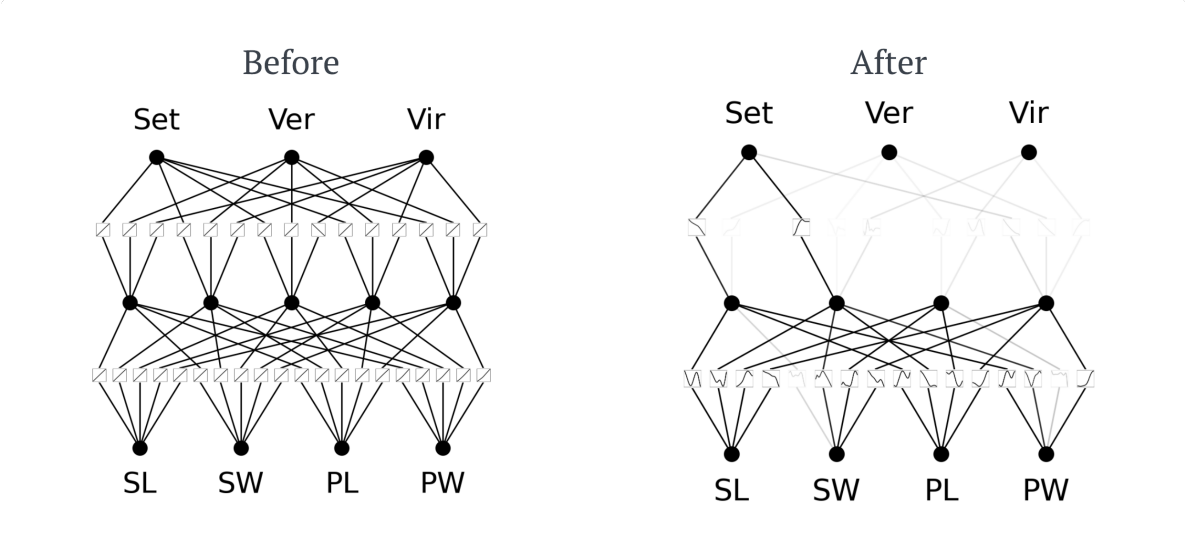

Figura 28: Representación del modelo KAN de la Figura 23 una vez se ha completado el entrenamiento [21].

Un resumen de los parámetros utilizados para construir el modelo KAN presentado en esta sección se recoge en la Tabla 10.

Tabla 10: Resumen de parámetros definidos para el entrenamiento del modelo KAN.

Parámetro	***Descripción***	***Valor usado en el ejemplo***
`opt`	Optimizador utilizado para ajustar los parámetros del modelo	`"Adam"`
`metrics`	Métricas evaluadas durante el entrenamiento	`(train_acc, test_acc)`
`loss_fn`	Función de pérdida utilizada para calcular el error	`torch.nn.CrossEntropyLoss()`
`steps`	Número de pasos o iteraciones de entrenamiento	`100`
`lamb`	Penalización global que controla la *sparsity* y la regularización	`0.01`
`Lamb_entropy`	Penalización relativa que controla la entropía de las funciones aprendidas	`10`

Por último, se incluyen las funcionalidades más comunes y útiles de la librería `pyKAN` en la Tabla 11.

Tabla 11: Funcionalidades más comunes de `pyKAN` [20]

Funcionalidad	***Descripción***
`model.train(dataset)`	Entrenar el modelo con un conjunto de datos
`model.plot()`	Visualización (gráficas)
`model.prune()`	Poda del modelo
`model.fix_symbolic(l,i,j,fun)`	Fija la función de activación $\phi_{l,i,j}$ a una función simbólica `fun`
`model.suggest_symbolic(l,i,j)`	Sugiere funciones simbólicas que se aproximen al valor numérico de $\phi_{l,i,j}$
`model.auto_symbolic()`	Usa la mejor sugerencia simbólica de `suggest_symbolic` para reemplazar todas las funciones de activación
`model.symbolic_formula()`	Devuelve la formula simbólica del modelo

5.4.3 SIMPLIFICACIÓN DE LAS KAN

En esta sección se describen una serie de técnicas que mejoran la interpretabilidad de las KAN [20]:

- ***Sparsification*:** En las MLP, se emplea la regularización L1 de los pesos lineales para favorecer la *sparsity*. Las KAN pueden adaptar esta idea general, pero requieren dos modificaciones:
    - En las KAN no hay pesos lineales, ya que estos son reemplazados por funciones de activación aprendibles. Por tanto, debe definirse la norma L1 sobre estas funciones.

  - La norma L1 por sí sola no resulta suficiente para lograr la *sparsification* en las KAN, por lo que se complementa con una regularización adicional basada en la entropía.

- **Visualización:** Al visualizar una KAN, las funciones con magnitud pequeña aparecen atenuadas visualmente. Esto permite centrar la atención en las más importantes durante el análisis.
- **Poda (*Pruning*):** Consiste en eliminar nodos y conexiones inactivos de la red, con el fin de reducir su complejidad y tamaño. `pyKAN` permite realizar la poda tanto de forma automática como manual, y ofrece la opción de podar nodos, bordes, o ambos simultáneamente. Dado que esta última opción es la más habitual, en la Figura 29 se muestra un ejemplo de una KAN antes y después del proceso de poda. Reentrenar una KAN tras la poda puede mejorar su rendimiento.

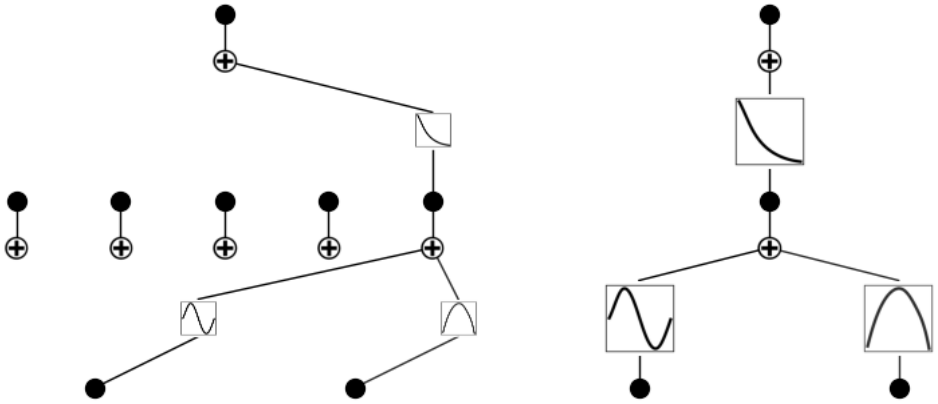

Figura 29: Diferencia entre una KAN antes (izquierda) y después (derecha) de ser podada [48].

- **Obtención de la fórmula simbólica:** Una de las características más destacadas de las KAN es su capacidad para extraer la fórmula simbólica que representa el modelo. Esto resulta extremadamente útil, ya que permite realizar inferencias directamente sobre dicha fórmula, reduciendo considerablemente el coste computacional. Al tratarse de la misma fórmula que utiliza la red internamente, no se pierde precisión al inferir sobre ella [21]. Este proceso se resume en la Figura 30.

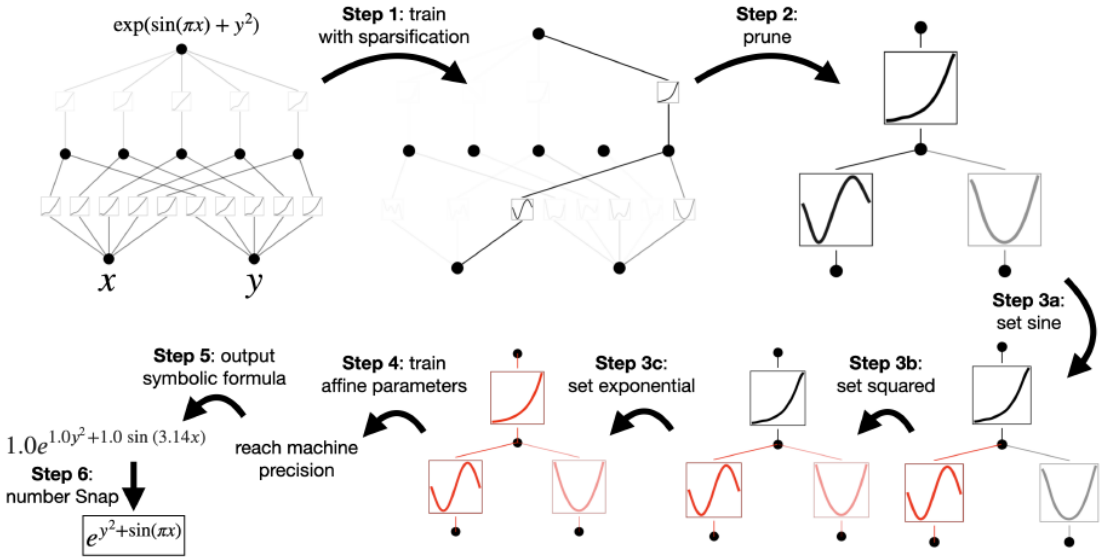

Figura 30: Proceso de la obtención de la fórmula simbólica en una KAN [20].

Como resumen, en la Tabla 12 se recogen las principales técnicas de simplificación explicadas en esta sección, junto con su propósito y utilidad en el entrenamiento y análisis de las KAN.

Tabla 12: Técnicas de simplificación aplicadas a las KAN y su utilidad principal.

Técnica	***Descripción breve***	***Ventaja principal***
Sparsification	Aplica penalización L1 y entropía	Reduce funciones irrelevantes
Visualización	Atenúa funciones poco relevantes	Mejora interpretabilidad
Poda	Elimina nodos y bordes inactivos	Reduce complejidad y tamaño
Simbologización	Extrae fórmula analítica de la red	Permite inferencia simbólica

En la Figura 31 se muestra un esquema que resume los pasos esenciales para ajustar, entrenar y simplificar las KAN.

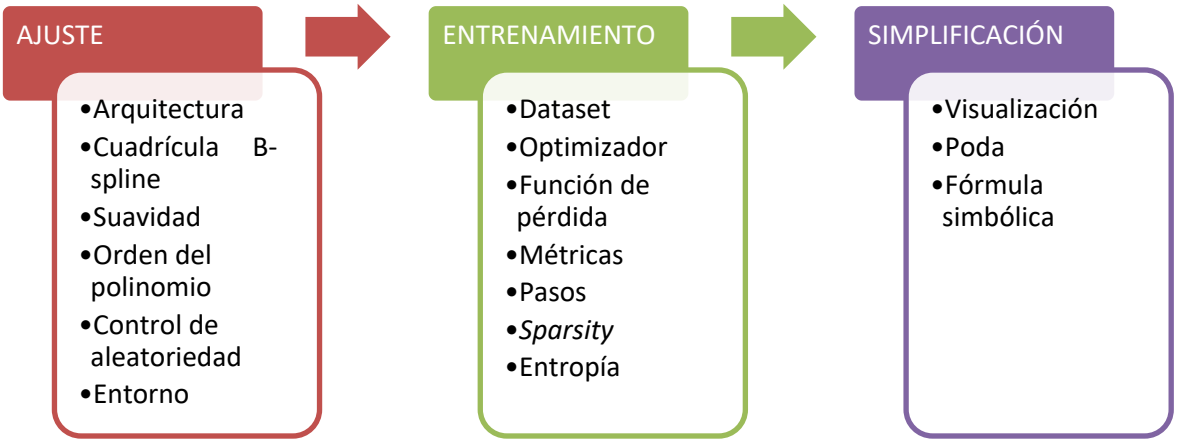

Figura 31: Esquema visual para ajustar, entrenar y simplificar las KAN.

5.5 *VENTAJAS Y DESVENTAJAS*

Una vez explicado en detalle el funcionamiento de las KAN, esta sección presenta sus principales fortalezas y limitaciones.

Ventajas [44]:

- **Interpretabilidad**: Como se ha demostrado a lo largo de este trabajo, las KAN proporcionan una estructura más interpretable que la mayoría de los modelos clásicos. Las funciones aprendidas pueden visualizarse y analizarse, lo cual es especialmente valioso en disciplinas científicas, donde comprender el funcionamiento del modelo es tan crucial como lograr una gran precisión.
- **Flexibilidad**: Aunque comúnmente emplean B-splines, las KAN no se limitan únicamente a este tipo de función base. También pueden utilizar polinomios de Chebyshev u otras funciones, lo que las hace altamente adaptables a distintos tipos de problemas.

- **Exactitud**: En algunos casos, las KAN han demostrado ofrecer mejores resultados y precisión en el modelo que modelos como las MLP.

Desventajas [44]:

- **Complejidad computacional**: Las funciones de activación aprendibles en los bordes de la red introducen una mayor carga computacional, especialmente en tareas que requieren funciones muy detalladas o redes de gran tamaño.
- **Riesgo de *overfitting***: La capacidad de las KAN para modelar relaciones complejas puede hacerlas propensas a sobreajustar el modelo cuando se dispone de pocos datos, capturando ruido como si se tratase de patrones significativos.
- **Necesidad de experiencia**: El diseño y ajuste de una KAN exige conocimientos matemáticos avanzados. Seleccionar las funciones base apropiadas y configurar correctamente el modelo requiere una mayor interacción y experiencia, lo que puede limitar su adopción por parte de usuarios no especializados.
- **Casos de uso limitados**: La relativa lentitud del entrenamiento y cierta inestabilidad en la optimización hacen que, actualmente, los casos de uso de las KAN sean aún limitados.

Para facilitar la comparación, la Tabla 13 presenta un resumen visual de las fortalezas y limitaciones de las KAN, según lo expuesto en esta sección.

Tabla 13: Comparativa entre las ventajas y desventajas principales de las KAN.

Ventajas	***Desventajas***
Interpretabilidad	Complejidad computacional
Flexibilidad en funciones base	Riesgo de overfitting
Mejor precisión en algunos casos	Requiere experiencia técnica
	Casos de uso aún limitados

5.6 COMPARACIÓN CON MLP

En esta sección se presenta una comparación visual entre las KAN y las MLP, recogida en la Figura 32. Aunque la mayoría de las diferencias entre ambas arquitecturas ya han sido tratadas a lo largo del trabajo, esta figura resume de manera clara sus principales contrastes en cuanto a teoremas fundamentales, fórmulas matemáticas, estructura y naturaleza de los parámetros.

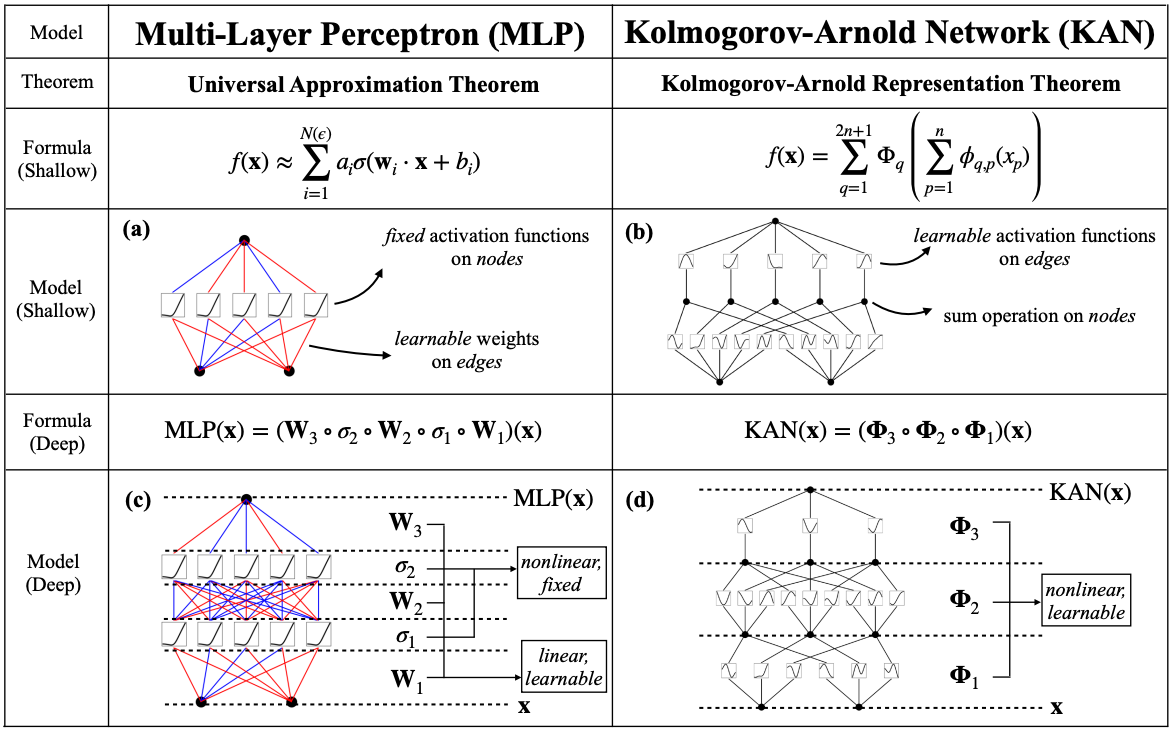

Figura 32: Comparación entre las KAN y las MLP [20].

Para concluir, la Figura 33 ofrece un árbol de decisión sobre cuándo resulta más conveniente utilizar KAN frente a MLP, en función de distintos criterios como la precisión, interpretabilidad o eficiencia. Como puede observarse, las KAN resultan especialmente recomendables cuando se busca interpretar el modelo, trabajar con estructuras composicionales complejas o llevar a cabo aprendizaje continuo. En cambio, si el objetivo principal es la rapidez de entrenamiento, las MLP siguen siendo una opción preferente.

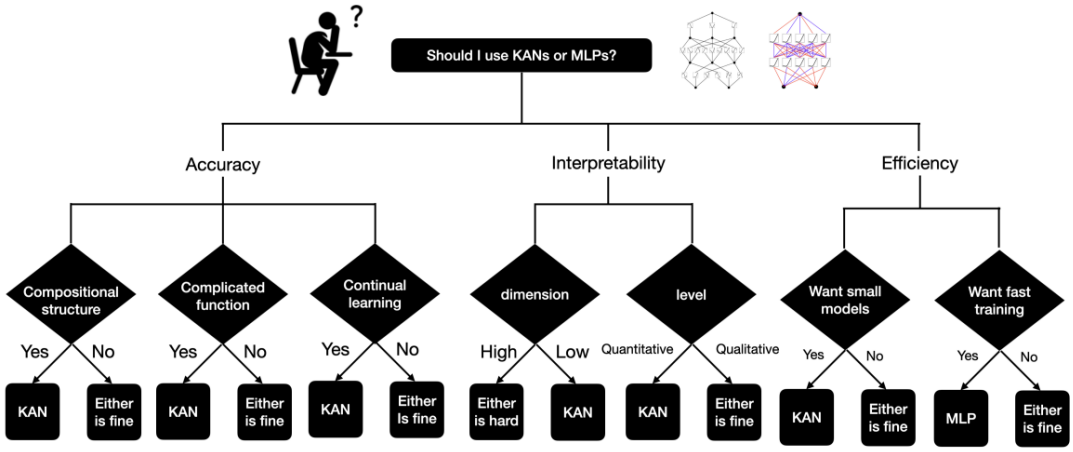

Figura 33: Diagrama de decisión para seleccionar entre KAN o MLP en función de la tarea y los objetivos del usuario [20].

5.7 EJEMPLOS Y CASOS DE USO

Para cerrar este capítulo, se describen en esta última sección algunos de los principales ámbitos donde las Kolmogorov–Arnold Networks (KAN) han demostrado un gran potencial [44]:

- **Modelización científica y ajuste de datos**: Las KAN son especialmente eficaces en entornos científicos que requieren el modelado preciso de funciones complejas.
- **Resolución de ecuaciones diferenciales parciales (EDP):** Las KAN han demostrado un gran potencial en la resolución de EDP, que se utilizan habitualmente en física e ingeniería para modelizar procesos como la transferencia de calor y la dinámica de fluidos.
- **Regresión simbólica**: Las KAN destacan en la regresión simbólica, donde el objetivo es descubrir las expresiones matemáticas que mejor describen un conjunto de datos.
- **Aplicaciones móviles**: En dispositivos con recursos computacionales limitados, como teléfonos móviles, las KAN resultan especialmente útiles. Tras entrenar el modelo, es posible convertirlo en una fórmula simbólica interpretable, permitiendo

realizar inferencias sin ejecutar la red, lo que reduce notablemente el coste computacional [21].

- **Aprendizaje continuo (*Continual Learning*)**: Las KAN muestran una notable capacidad de adaptación a nuevos datos sin olvidar la información previamente aprendida. Esto se debe a la naturaleza local de sus funciones *spline*, que permite actualizar solo una parte del modelo sin afectar al resto. Así, se reduce el riesgo de *olvido catastrófico*, un problema habitual en otros modelos como las MLP. Esta capacidad se ilustra en la Figura 34, donde se comparan los resultados obtenidos por una KAN y una MLP en un entorno de aprendizaje por fases sucesivas. Se observa cómo la MLP olvida información aprendida en fases anteriores (catastrophic forgetting), mientras que la KAN conserva los patrones aprendidos y los integra progresivamente.

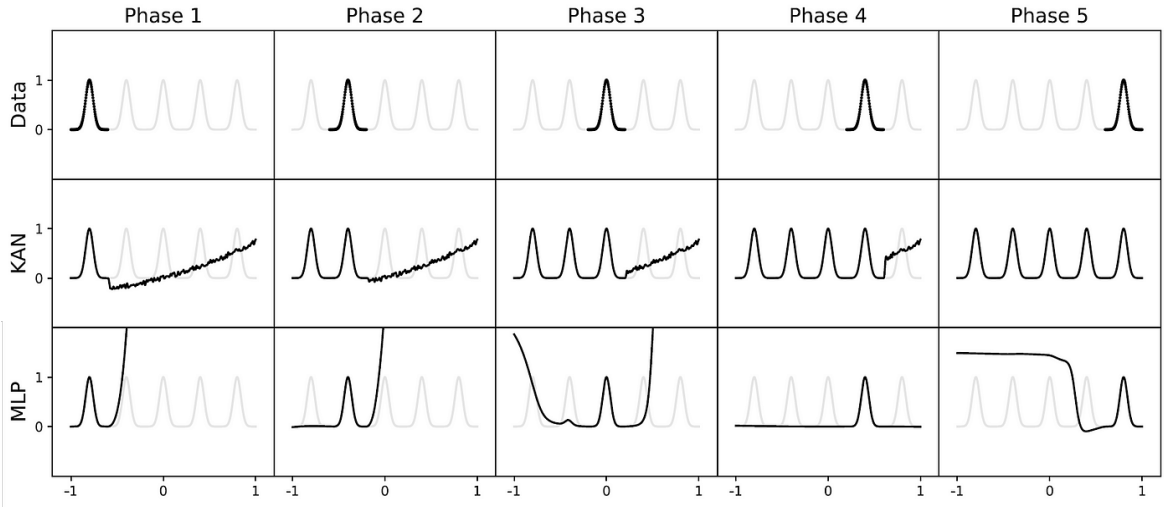

Figura 34: Comparación del comportamiento de una KAN y una MLP en un escenario de aprendizaje continuo a lo largo de cinco fases [20].

Capítulo 6. EVALUACIÓN DE MODELOS SOBRE UN CASO SIMPLE

El presente capítulo tiene como objetivo aplicar y comparar seis técnicas de clasificación sobre un conjunto de datos sencillo, con el fin de observar en la práctica sus principales ventajas y limitaciones. Este análisis sirve como punto de partida para entender el rendimiento de cada algoritmo antes de abordar un conjunto de datos más complejo.

Para ello, se ha utilizado el dataset `weatherAUS.csv`, un conjunto de datos que está orientado a la predicción de lluvia que incluye aproximadamente 10 años de observaciones meteorológicas diarias tomadas en múltiples ubicaciones de Australia [52]. A partir de él, se han entrenado seis modelos distintos: Support Vector Machine (SVM), Árboles de decisión, Random Forest, Regresión logística, Redes Neuronales Multicapa (MLP) y Kolmogorov-Arnold Networks (KAN).

En las secciones siguientes se describirá en primer lugar el conjunto de datos. A continuación, se detallará el ajuste de cada uno de los modelos. Finalmente, se analizarán y compararán los resultados obtenidos.

6.1 DESCRIPCIÓN DEL CONJUNTO DE DATOS

En esta primera sección se describe el origen, finalidad y estructura del dataset, y a continuación se detalla el preprocesamiento de los datos que se ha llevado a cabo para posteriormente construir los modelos. Por último, se ofrece un análisis exploratorio de los datos.

6.1.1 ORIGEN Y FINALIDAD DEL DATASET

Como se ha descrito en la introducción de este capítulo, el dataset `weatherAUS.csv` tiene como objetivo la predicción de lluvia. Este conjunto de datos es de dominio público y se ha descargado a través de la página web de *Kaggle* (véase Figura 35), una plataforma online

que es, entre otras cosas, un repositorio de datos. Los ejemplos que forman parte de este dataset fueron recopilados por numerosas estaciones meteorológicas.

Figura 35: Logo oficial de la plataforma Kaggle [53].

Al trabajar con este dataset, la finalidad es predecir la clase binaria `'RainTomorrow'`, cuyas categorías son únicamente "Sí" o "No". En otras palabras, se debe construir un modelo que prediga si lloverá mañana en alguna localidad australiana en base a los datos de 23 columnas y sus aproximadamente 142.000 ejemplos.

6.1.2 ESTRUCTURA DEL DATASET

La Tabla 14 muestra un resumen de las columnas que forman parte de este conjunto de datos [52].

Tabla 14: Descripción de las variables del dataset `weatherAUS.csv`.

Variable	***Tipo de dato***	***Descripción breve***
`Date`	Fecha	Fecha de la observación
`Location`	Categórica	Nombre de la estación meteorológica
`MinTemp`	Numérica (°C)	Temperatura mínima del día
`MaxTemp`	Numérica (°C)	Temperatura máxima del día
`Rainfall`	Numérica (mm)	Precipitación acumulada durante el día
`Evaporation`	Numérica (mm)	Evaporación tipo A en las 24 h previas a las 9:00
`Sunshine`	Numérica (horas)	Horas de sol directo durante el día
`WindGustDir`	Categórica	Dirección de la racha de viento más fuerte hasta la medianoche
`WindGustSpeed`	Numérica (km/h)	Velocidad de la racha de viento más fuerte
`WindDir9am`	Categórica	Dirección del viento a las 9:00
`WindDir3pm`	Categórica	Dirección del viento a las 15:00
`WindSpeed9am`	Numérica (km/h)	Velocidad media del viento antes de las 9:00
`WindSpeed3pm`	Numérica (km/h)	Velocidad media del viento antes de las 15:00
`Humidity9am`	Numérica (%)	Humedad relativa a las 9:00
`Humidity3pm`	Numérica (%)	Humedad relativa a las 15:00
`Pressure9am`	Numérica (hPa)	Presión atmosférica reducida al nivel del mar a las 9:00
`Pressure3pm`	Numérica (hPa)	Presión atmosférica reducida al nivel del mar a las 15:00
`Cloud9am`	Numérica (oktas)	Fracción del cielo cubierto por nubes a las 9:00 (0 = despejado, 8 = cubierto)
`Cloud3pm`	Numérica (oktas)	Fracción del cielo cubierto por nubes a las 15:00
`Temp9am`	Numérica (°C)	Temperatura a las 9:00
`Temp3pm`	Numérica (°C)	Temperatura a las 15:00
`RainToday`	Booleana (0/1)	1 si ha llovido más de 1 mm en 24 h, 0 si no
`RISK_MM`	Numérica (mm)	Estimación de lluvia para el día siguiente *(se elimina para predicción)*
`RainTomorrow`	Categórica (Sí/No)	Variable objetivo: indica si lloverá al día siguiente

6.1.3 PREPROCESAMIENTO DE LOS DATOS

En esta sección se detallan los pasos seguidos durante el preprocesamiento del conjunto de datos antes de entrenar los diferentes modelos de clasificación. Este preprocesamiento tiene el objetivo principal de limpiar, preparar y transformar los datos para su correcto análisis.
El proceso completo de preprocesamiento aplicado al conjunto de datos se muestra en la . A continuación, se muestra el detalle de cada paso.

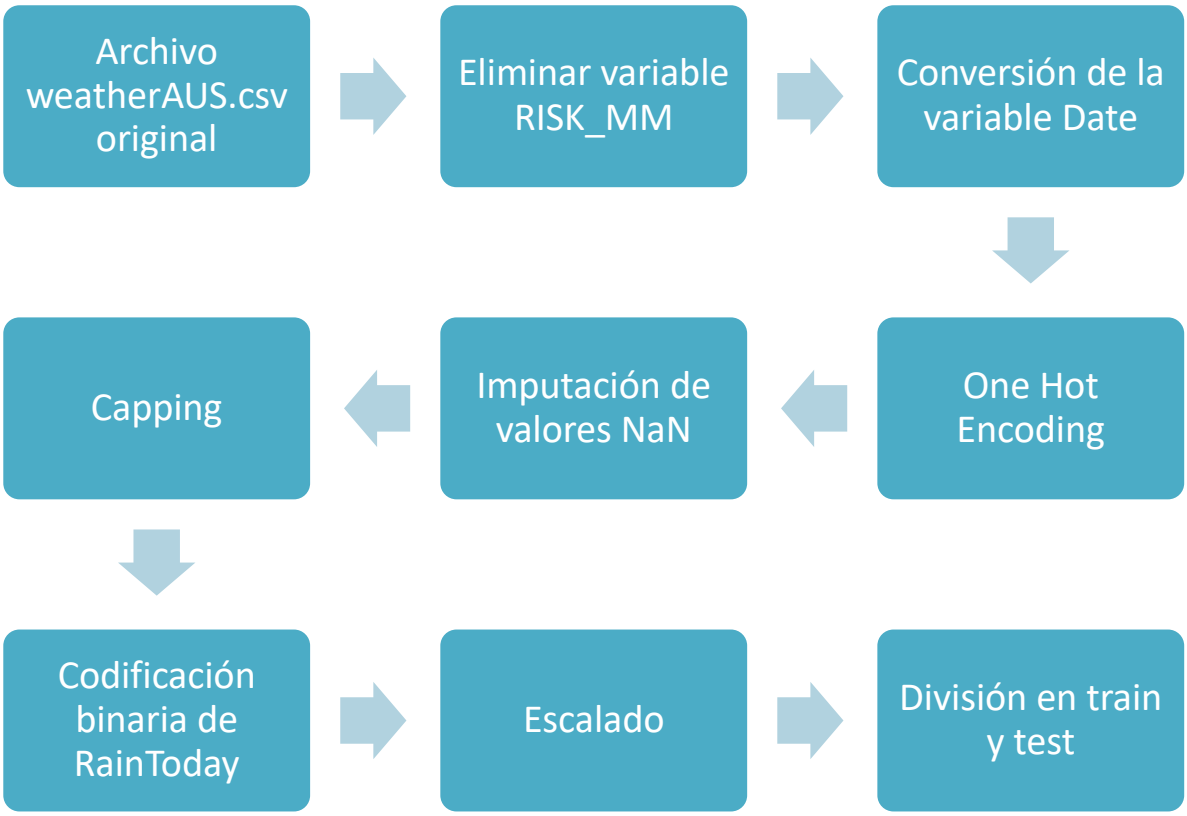

Figura 36: Diagrama de flujo del proceso de preprocesamiento aplicado al dataset `weatherAUS.csv`.

Eliminación de la variable `'RISK_MM'`

Como se ha explicado anteriormente, esta variable representa una estimación directa de la cantidad de lluvia esperada al día siguiente, y por tanto, no puede utilizarse como predictor de la variable objetivo `RainTomorrow`.

```
df.drop(['RISK_MM'], axis=1, inplace=True)
```

Conversión y descomposición de la fecha

La variable `Date`, inicialmente en formato texto, se transformó al formato `datetime` y se descompuso en tres nuevas columnas: año (`Year`), mes (`Month`) y día (`Day`). La variable original se eliminó posteriormente.

```
df['Date'] = pd.to_datetime(df['Date'])
df['Year'] = df['Date'].dt.year
df['Month'] = df['Date'].dt.month
df['Day'] = df['Date'].dt.day
df.drop('Date', axis=1, inplace=True)
```

Identificación y codificación de variables categóricas

Como se describió en la sección 6.1.2, se han identificado cuatro variables categóricas: `Location`, `WindGustDir`, `WindDir9am` y `WindDir3pm`. Estas variables se transformaron mediante *One Hot Encoding*, una técnica que crea nuevas columnas binarias (0 o 1) para cada categoría, evitando así introducir un orden arbitrario. No se entrará en detalle explicando qué categorías tienen todas las variables categóricas para no extenderse demasiado, pero puede verse la tabla obtenida para la columna `WindDir9am` en la Figura 37. Además, se añadió una columna adicional para representar los valores nulos (`NaN`).

```
pd.get_dummies(df.WindDir9am, drop_first=True, dummy_na=True)
```

	ENE	ESE	N	NE	NNE	NNW	NW	S	SE	SSE	SSW	SW	W	WNW	WSW	NaN
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
3	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
4	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figura 37: Codificación obtenida para la variable categórica `'WindDir9am'`.

Imputación de valores nulos

Consiste en reemplazar los valores nulos para evitar eliminar demasiados registros, ya que este conjunto de datos contiene un gran porcentaje de estos valores.

- Para las variables numéricas, se utilizó la mediana del conjunto de entrenamiento.

- Para las variables categóricas, se imputó la moda de cada columna, es decir, el valor más frecuente.

```
# Ejemplo para numéricas
df['WindSpeed3pm'].fillna(X_train['WindSpeed3pm'].median(), inplace=True)
```

Control de *outliers*

Se aplicó un recorte de valores atípicos (*capping*) en algunas variables numéricas, estableciendo umbrales superiores basados en el análisis exploratorio. Para identificar estos valores extremos se ha utilizado el método del rango intercuartílico (IQR), que consiste en calcular el rango entre el primer y el tercer cuartil (Q3-Q1) y definir como *outliers* los valores que están por encima de $Q3 + 3 * IQR$ o por debajo de $Q1 - 3 * IQR$.

En la Tabla 15 se resumen los umbrales superiores aplicados para limitar el impacto de *outliers*. De esta manera, si, por ejemplo, un valor en `Rainfall` supera los 3,2 mm, se sustituye por ese umbral.

Tabla 15: Umbrales superiores aplicados para controlar outliers.

Variable	***Umbral superior***
`Rainfall`	3,2 mm
`Evaporation`	21,8 mm
`WindSpeed9am`	55 km/h
`WindSpeed3pm`	57 km/h

Codificación binaria de la columna `RainToday`

Aunque `RainToday` es una variable booleana, aparece como texto en el dataset (`Yes` / `No`), por lo que se transformó mediante codificación binaria, generando dos columnas (`RainToday_0`, `RainToday_1`) que preservan la información sin introducir orden implícito.

```
from category_encoders import BinaryEncoder
encoder = BinaryEncoder(cols=['RainToday'])
df_encoded = encoder.fit_transform(df)
```

Escalado de variables

Se aplicó un escalado de tipo *MinMax*, que sirve para normalizar todas las variables entre 0 y 1. Esta técnica es especialmente útil para técnicas como redes neuronales o máquinas de vectores de soporte (SVM).

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

División del *dataset* en conjuntos de *train* y *test*

Finalmente, los datos fueron divididos en conjuntos de *train* y *test*, con una proporción del 80% y el 20%, respectivamente.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=0)
```

6.1.4 ANÁLISIS EXPLORATORIO

Antes de entrenar los seis modelos, se ha llevado a cabo un análisis exploratorio para comprender mejor la distribución de las variables, examinar relaciones relevantes entre ellas y detectar posibles desequilibrios en la variable objetivo.

Uno de los primeros aspectos a observar es la distribución de `RainTomorrow`, la variable objetivo que indica si lloverá o no al día siguiente. Si se observa la Figura 38, este conjunto de datos está claramente desbalanceado: la mayoría de los días no llueve al día siguiente, mientras que los días en los que sí llueve son considerablemente inferiores. Este desbalance puede afectar al rendimiento de los clasificadores, especialmente los que son más sensibles al equilibrio entre clases. Por ello, se tendrá en cuenta este aspecto al interpretar las métricas de evaluación.

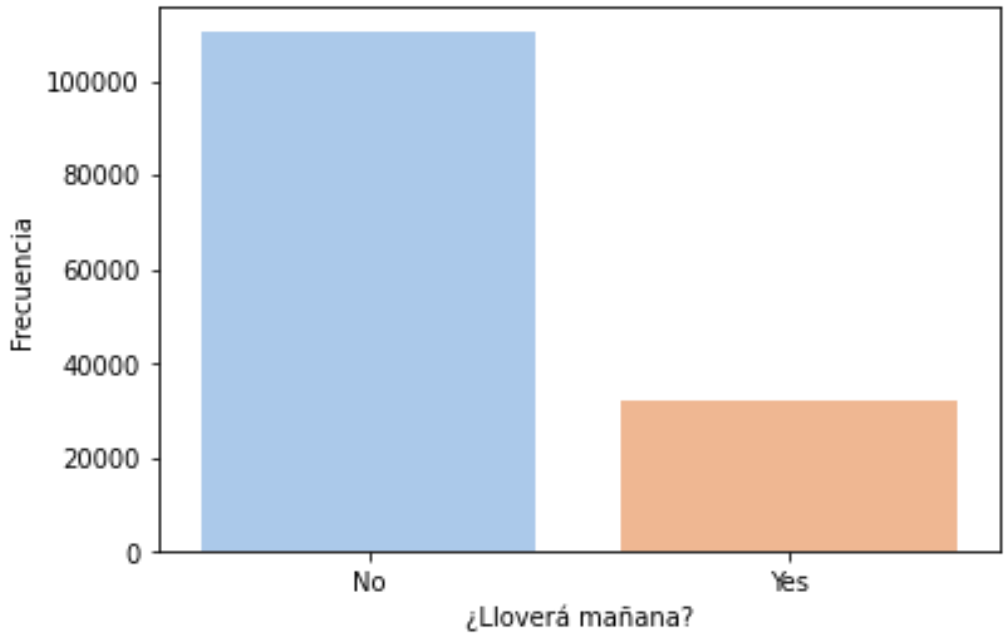

Figura 38: Distribución de la variable objetivo `RainTomorrow`.

Para continuar, la distribución de la variable `Rainfall`, que representa la cantidad de lluvia registrada en un día, es un aspecto relevante al trabajar con este conjunto de datos. Como se aprecia en la Figura 39, la variable presenta una distribución altamente asimétrica, con una gran concentración de valores cercanos a cero y una cola larga hacia valores altos. Esto significa que la mayoría de los días no llueve o llueve muy poco, mientras que los días con precipitaciones intensas son muy poco frecuentes. Además, puede verse que existen *outliers*, lo que justifica la aplicación de técnicas de recorte durante el preprocesamiento.

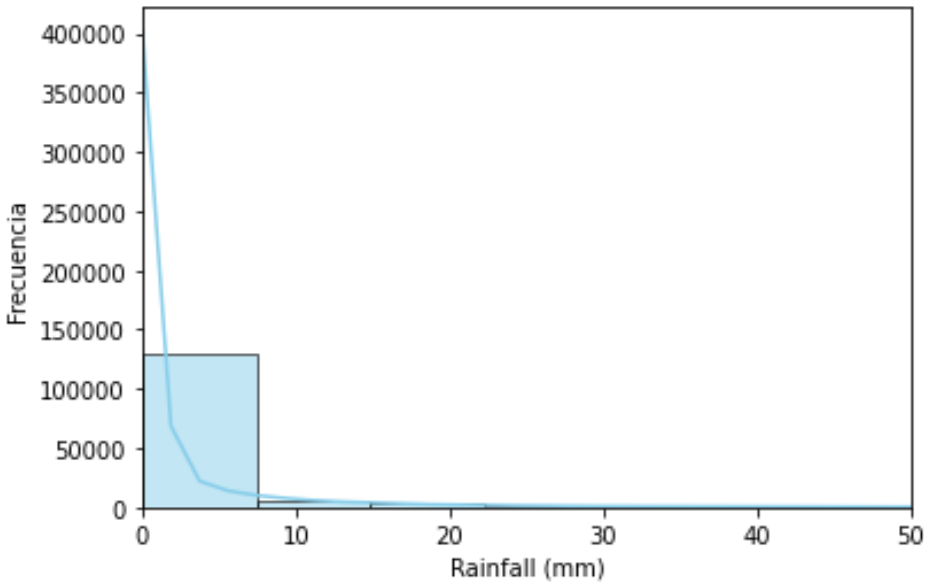

Figura 39: Distribución de la variable `Rainfall`.

También se ha analizado la correlación entre las variables numéricas del dataset, con el fin de identificar relaciones lineales que puedan ser útiles en la predicción de `RainTomorrow`. En la Figura 40 se muestra un mapa de calor con los coeficientes de correlación de Pearson entre todas las variables numéricas.

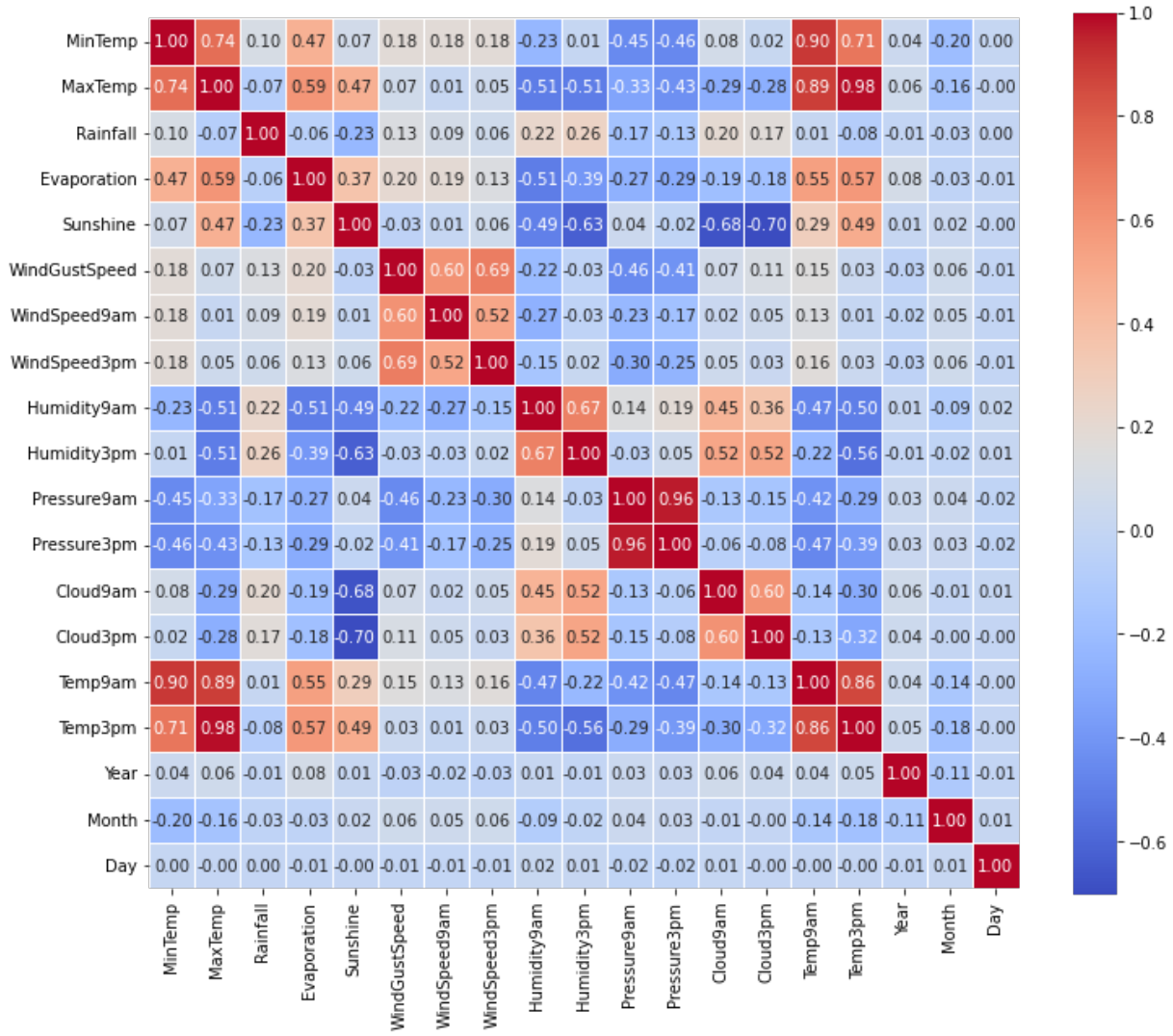

Figura 40: Mapa de calor de correlaciones entre variables numéricas del dataset.

Se pueden observar correlaciones fuertes entre algunas variables relacionadas, como era de esperar. Algunos ejemplos son:

- `MinTemp` y `MaxTemp`: $r = 0.74$.
- `Temp9am` y `Temp3pm`: $r = 0.98$.
- `Humidity9am` y `Humidity3pm`: $r = 0.86$.

Este análisis permite identificar variables que podrían generar problemas de multicolinealidad en ciertos modelos.

Se ha analizado también la proporción de valores nulos en cada variable del dataset. La Figura 41 muestra un gráfico de barras con el porcentaje de valores nulos por columna, ordenado de mayor a menor. Se aprecia que las variables `Sunshine`, `Evaporation`, `Cloud3pm` y `Cloud9am` superan el 30% de valores nulos. Es por ello por lo que en el preprocesamiento se ha optado por la imputación de valores nulos. Este análisis es clave para garantizar que el proceso de entrenamiento se lleve a cabo con normalidad.

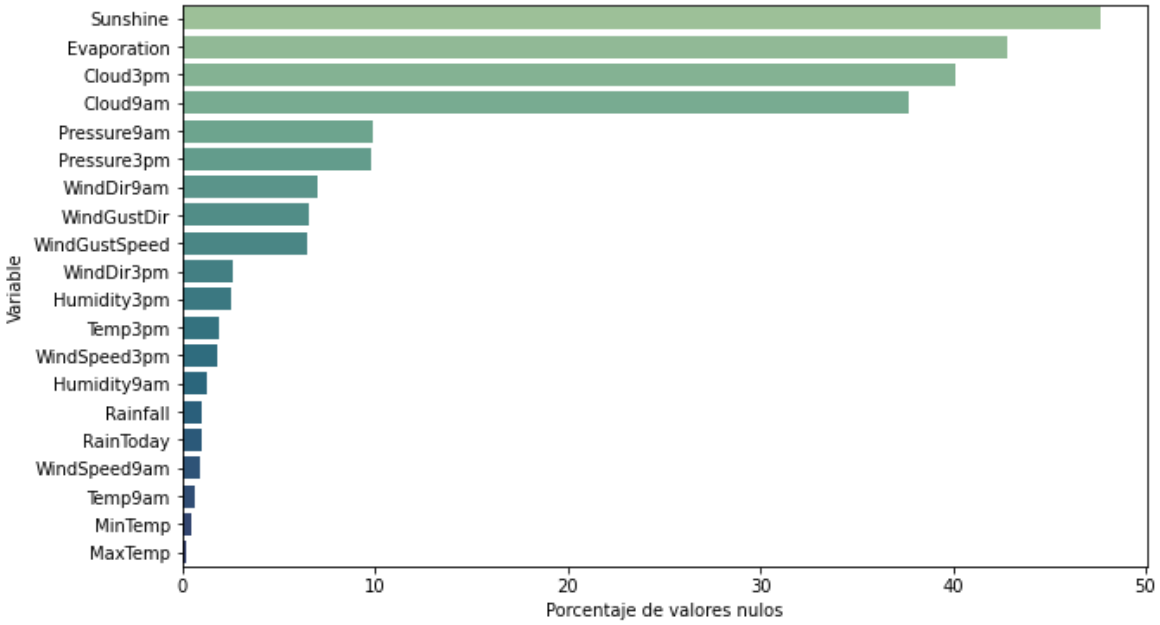

Figura 41: Porcentaje de valores nulos por cada variable del dataset `weatherAUS.csv`.

Para finalizar, se ha generado un resumen estadístico de las variables numéricas del dataset, que incluye medidas como la media, el mínimo, el máximo o la desviación estándar. Esta información permite detectar *outliers* y complementa las visualizaciones anteriores. Puede consultarse en la Tabla 16.

Tabla 16: Resumen estadístico de las variables numéricas.

Variable	*Media*	*Desviación estándar*	*Mín.*	*25%*	*50%*	*75%*	*Máx.*
MinTemp	12	6	-8	8	12	17	34
MaxTemp	23	7	-5	18	23	28	48
Rainfall	2	8	0	0	0	1	371
Evaporation	5	4	0	3	5	7	145
Sunshine	8	4	0	5	8	11	14
WindGustSpeed	40	14	6	31	39	48	135
WindSpeed9am	14	9	0	7	13	19	130
WindSpeed3pm	19	9	0	13	19	24	87
Humidity9am	69	19	0	57	70	83	100
Humidity3pm	51	21	0	37	52	66	100
Pressure9am	1.018	7	980	1.013	1,018	1.022	1.041
Pressure3pm	1.015	7	977	1.010	1,015	1.020	1.040
Cloud9am	4	3	0	1	5	7	9
Cloud3pm	5	3	0	2	5	7	9
Temp9am	17	6	-7	12	17	22	40
Temp3pm	22	7	-5	17	21	26	47
Year	2.013	3	2.007	2.011	2.013	2.015	2.017
Month	6	3	1	3	6	9	12
Day	16	9	1	8	16	23	31

Además del análisis numérico mostrado, se han analizado la presencia de valores atípicos mediante diagramas de caja en aquellas variables sospechosas de tener este tipo de valores, que son `Rainfall`, `Evaporation`, `WindSpeed9am` y `WindSpeed3pm`. Una vez se ha confirmado la

presencia de estos valores, se ha graficado su distribución. Esto se muestra en la Figura 42 y la Figura 43. Todas las variables presentan una distribución sesgada a la derecha, y es por ello por lo que se han aplicado técnicas de recorte para evitar que estos valores afecten negativamente al rendimiento de los modelos.

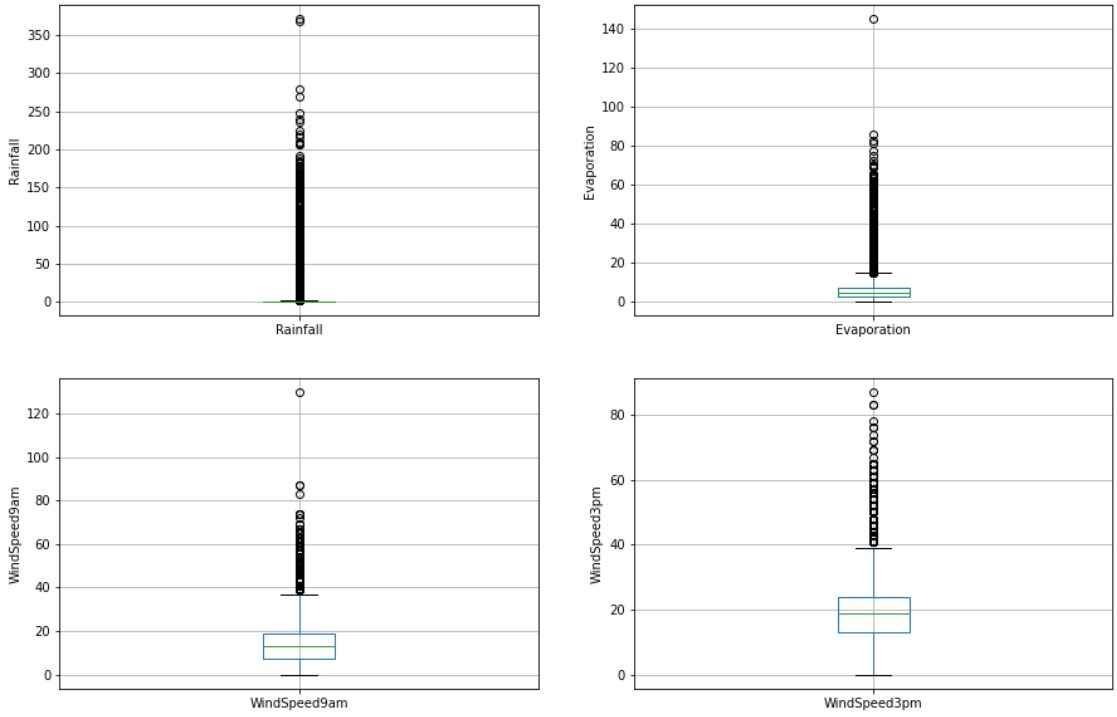

Figura 42: Diagramas de caja de las variables `Rainfall`, `Evaporation`, `WindSpeed9am` *y* `WindSpeed3pm`.

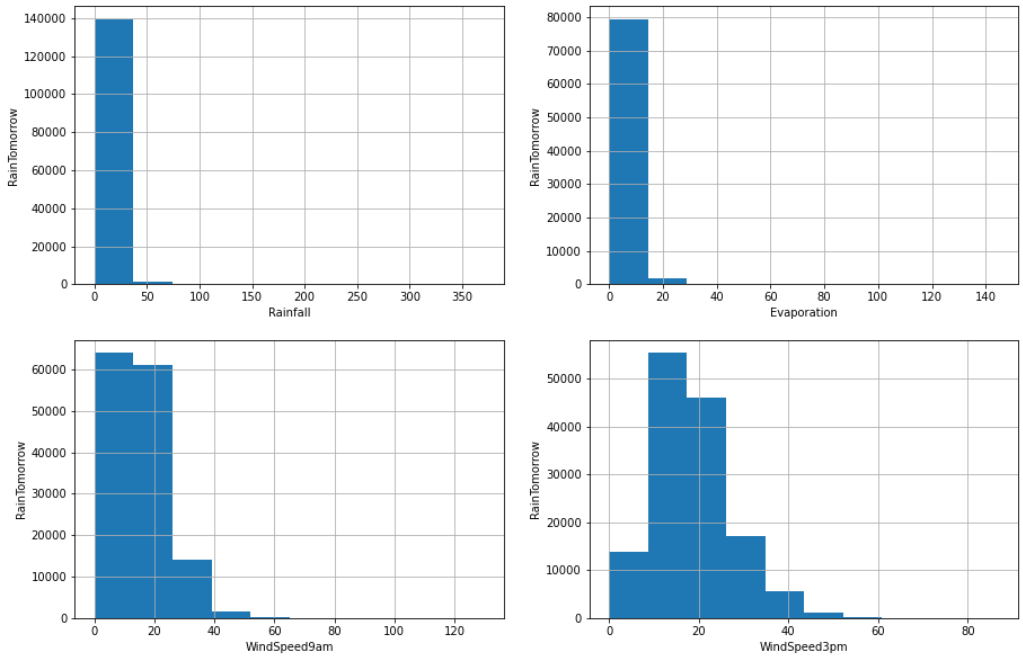

Figura 43: Distribución de las mismas variables antes del tratamiento de outliers.

6.2 ENTRENAMIENTO Y AJUSTE DE LOS MODELOS

Una vez finalizado el preprocesamiento de los datos, se procede al entrenamiento y ajuste de los seis modelos de clasificación seleccionados. La presente sección tiene como objetivo describir detalladamente la configuración de cada modelo, los hiperparámetros utilizados y analizar en qué medida las ventajas y desventajas teóricas se cumplen también en la práctica.

6.2.1 SUPPORT VECTOR MACHINE (SVM)

Uno de los primeros modelos implementados ha sido el de las Support Vector Machines (SVM). Son modelos utilizados por su capacidad de generalización y su robustez frente a valores atípicos (*outliers*), aunque esta última no ha podido comprobarse en este caso ya que los valores atípicos se han eliminado durante el preprocesamiento. Para entrenar este modelo, se han analizado distintas configuraciones, seleccionando finalmente un modelo con *kernel* polinómico y con un valor de $C = 1$ por ser el que obtuvo mejor rendimiento.

El parámetro C regula el equilibrio entre el doble objetivo de maximizar el margen entre clases y minimizar el error cometido. Un valor bajo de C tiende a penalizar poco los errores, permitiendo márgenes mayores a costa de cometer más errores. Al contrario, un valor alto de C penaliza mucho los errores, llevando al modelo a cometer menos errores a costa de un margen menor, lo que puede llevar a sobreajuste [54].

Durante el entrenamiento se probaron cuatro tipos de *kernel*, que son funciones que transforman el espacio de características para permitir la separación no lineal de éstas [54]:

- **Lineal**: asume que los datos son separables por una recta o hiperplano. Es el más sencillo de todos.
- **RBF (*radial basis function*):** Transforma los datos en un espacio de mayor dimensión para separar clases no lineales. Es probablemente el más comúnmente usado.
- **Sigmoidal**: Es similar a la función de activación sigmoide en redes neuronales, aunque suele dar peor rendimiento si no se ajusta bien.
- **Polinómico**: Es costoso computacionalmente, pero útil cuando se necesita encontrar relaciones complejas en los datos. Transforma los datos con combinaciones polinómicas de las variables originales.

La Tabla 17 muestra las puntuaciones de *accuracy* obtenidas en el conjunto de *test* para cada uno de estos cuatros *kernels*:

Tabla 17: Comparación de accuracy según el tipo de kernel en SVM.

Kernel	**Parámetros**	***Accuracy***
RBF	C=1, gamma=auto (por defecto)	0,8591
Lineal	C=1	0,8505
Sigmoidal	C=1	0,7495
Polinómico	C=1	**0,8603**

Como puede verse, el kernel polinómico obtuvo la mejor precisión, superando por poco al RBF y con creces al sigmoidal. Además, se obtuvo una *accuracy* de 0,8743 en entrenamiento, un valor cercano al de *test*, lo que indica una buena capacidad de generalización y que no se ha detectado *overfitting*.

En la Tabla 18 se resumen las métricas finales del modelo seleccionado. Este modelo destaca por su precisión y por su alta capacidad de generalización.

Tabla 18: Métricas del modelo final SVM

Métrica	Valor
Accuracy	0,8603
F1-score (ponderado)	0,8489
Recall (macro)	0,7372
AUC-ROC	0,8854

Es importante mencionar que el conjunto de datos está desbalanceado, lo que condiciona la evaluación del modelo, ya que un clasificador que predijese siempre la clase "No", que es la mayoritaria, tendría una precisión artificialmente alta. Por ello, se consideran métricas más representativas el F1-score ponderado, el recall macro y el AUC-ROC.

El análisis de la matriz de confusión (véase Figura 44) permite observar que el modelo predice sólidamente la clase mayoritaria, pero presenta una sensibilidad menor para la clase minoritaria. En concreto, se han identificado correctamente 3.277 días de lluvia, mientras que 3.095 se han clasificado incorrectamente como días sin lluvia, lo que indica que el modelo todavía presenta margen de mejora en la predicción de la clase "Yes".

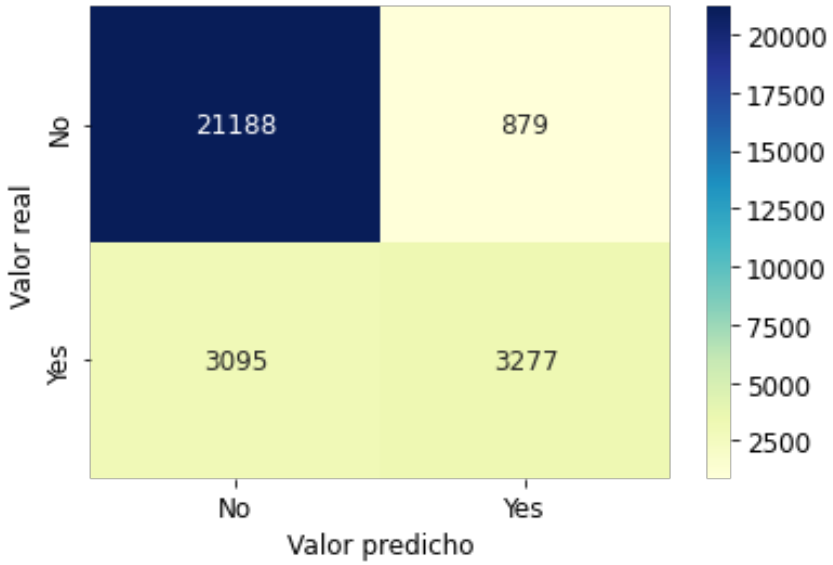

Figura 44: Matriz de confusión del modelo SVM

De forma complementaria, la Tabla 19 resume el informe de la clasificación detallado por clase.

Tabla 19: Informe de clasificación por clase

Clase	***Precision***	***Recall***	***F1-score***	***Soporte***
No	0,87	0,96	0,91	22.067
Yes	0,76	0,49	0,59	6.372
Promedio macro	0,81	0,72	0,75	28.439
Promedio ponderado	0,84	0,85	0,84	28.439

Por otro lado, la curva ROC (véase Figura 45) muestra un área bajo la curva de 0.8854, lo que confirma que el modelo tiene una elevada capacidad de discriminación entre clases, incluso cuando el dataset está desbalanceado.

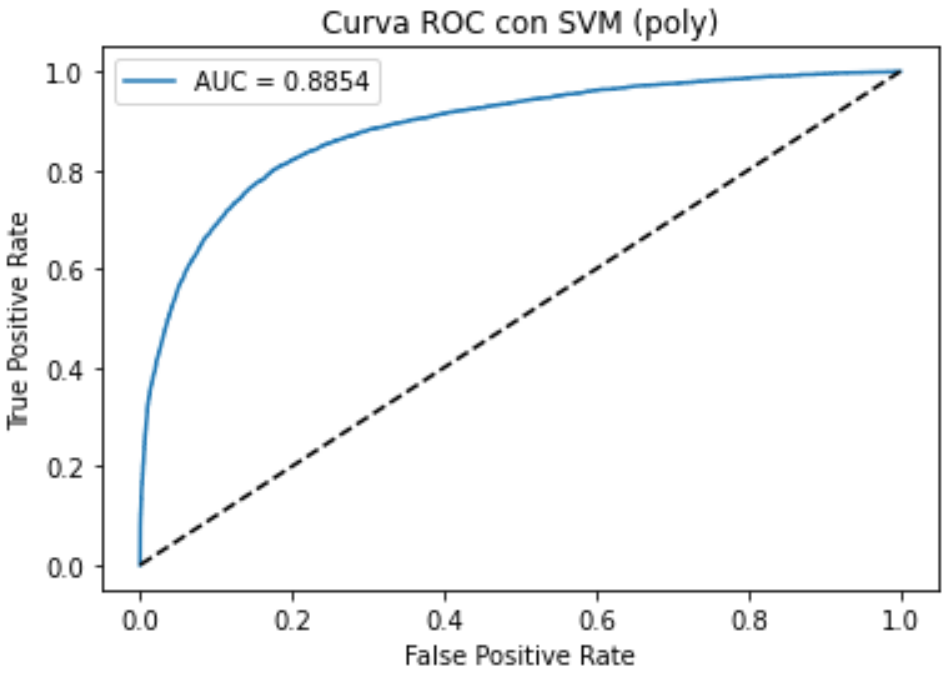

Figura 45: Curva ROC del modelo SVM.

Entrenar los cuatro modelos para conseguir el más adecuado tardó un total de 2 horas y 42 minutos, consumiendo 22,48 Wh de energía. Queda así demostrada la poca eficiencia de estos modelos, ya que son cifras bastante elevadas tratándose de un *dataset* de estas dimensiones. Por otro lado, se ha tardado menos de 10 segundos en ajustar el modelo, un tiempo muy razonable.

6.2.2 ÁRBOLES DE DECISIÓN

Los árboles de decisión son un modelo de clasificación basado en reglas de decisión que permite dividir el espacio de características de forma intuitiva. Durante el entrenamiento de este modelo se han implementado dos árboles distintos, ambos con profundidad máxima 3 para facilitar su interpretabilidad y visualización y evitar *overfitting*, ya que esta técnica de Machine Learning tiende al sobreajuste. Se han comparado dos criterios: el índice de Gini y el de entropía. Para ello, no fue necesario llevar a cabo ningún preprocesado adicional al que ya se indicó en la sección 6.1.3, pero se eliminó el escalado de variables con *OneHotEncoding* y *MinMaxScaler* para mejorar la interpretabilidad de los árboles, lo que

confirma su ventaja de preprocesado sencillo. Por otro lado, no se ha podido comprobar la robustez de este modelo frente a valores atípicos porque han sido eliminados durante el preprocesado de datos.

Criterio de Gini

El primer modelo se entrenó con el parámetro `criterion='gini'`, que utiliza el índice de Gini como criterio para realizar las divisiones del árbol. Este índice mide la impureza de un nodo, es decir, cómo de mezcladas están las clases dentro de él. Un valor cercano a 0 indica una mayor pureza, con presencia de una sola clase. La Figura 46 muestra el árbol resultante.

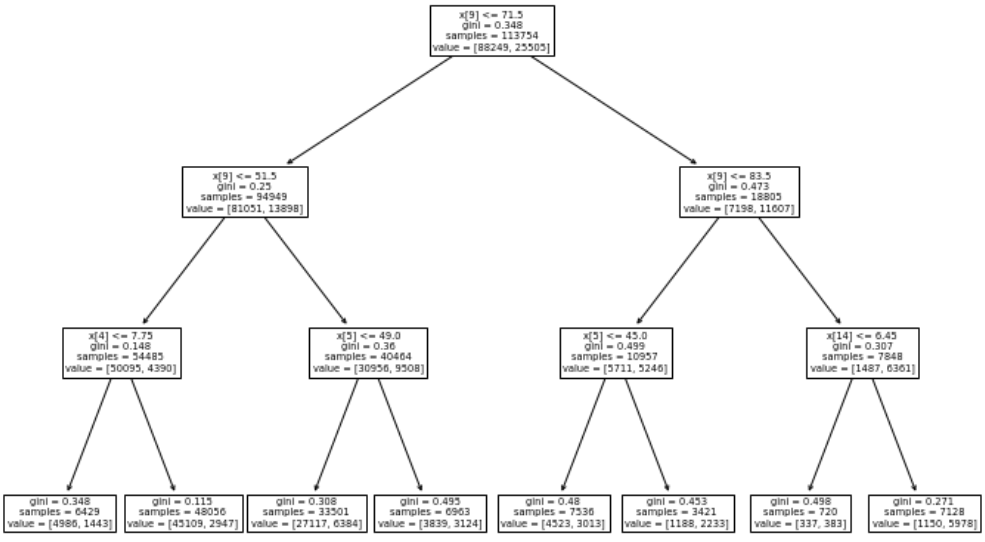

Figura 46: Árbol de decisión entrenado con índice de Gini.

El modelo obtuvo las siguientes estadísticas:

- *Accuracy*: 0,8284
- Recall de la clase “Yes”: 0,3402
- F1-Score de la clase “Yes”: 0,4705
- AUC-ROC: 0,6548

El rendimiento del modelo sobre la clase minoritaria ("Yes") fue bastante limitado. Esto puede asociarse al desequilibrio en las clases del dataset.

Criterio de Entropía

El segundo árbol se entrenó con el criterio de entropía, que mide la reducción de incertidumbre después de cada partición de los datos. El árbol obtenido se muestra en la Figura 47. Para interpretarlo, hay que tener en cuenta cuáles son cada una de las variables:

- x[9]: `Humidity3pm`
- x[4]: `Sunshine`
- x[5]: `WindGustSpeed`
- x[11]: `MinTemp`

Además, `value` muestra primero el número de muestras de la clase "No" y a continuación el número de muestras de la clase "Yes". La interpretabilidad de estos árboles es una de sus mayores ventajas, ya que se podría aplicar el modelo de clasificación sólo viendo el árbol de decisión resultante.

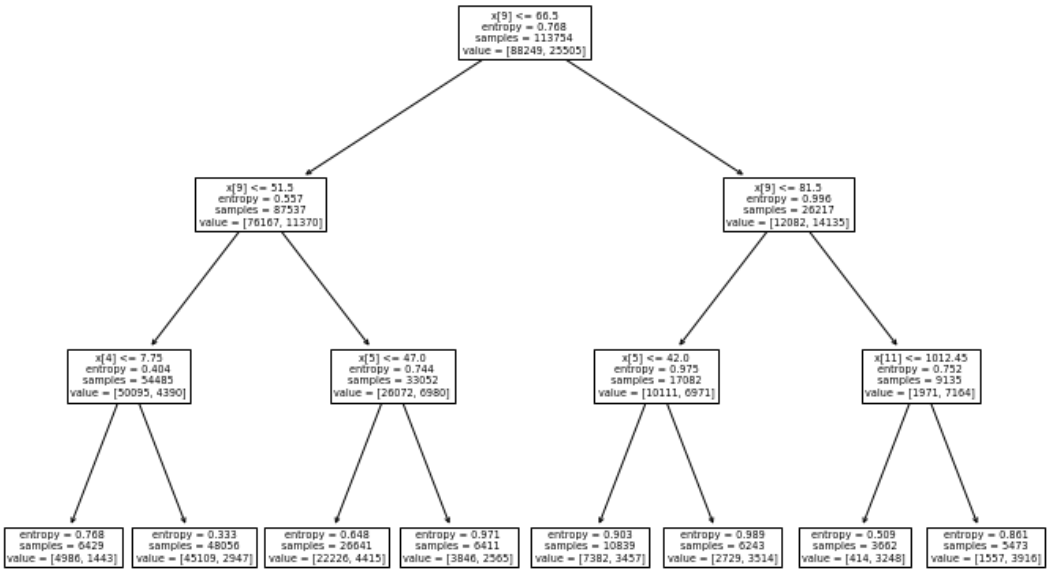

Figura 47: Árbol de decisión entrenado con entropía.

Este modelo obtuvo mejores estadísticas que el árbol anterior en la clase minoritaria, por lo que se seleccionó como modelo final. La diferencia entre ambos modelos puede observarse en la Tabla 20.

Tabla 20: Comparativa de métricas entre los modelos entrenados con índice de Gini y entropía.

Métrica	Árbol Gini	Árbol Entropía
Accuracy	0,8284	**0,8302**
***Recall* (Yes)**	0,3402	**0,4298**
***F1-Score* (Yes)**	0,4705	**0,5314**
AUC-ROC	0,6548	**0,6878**

Cabe señalar que la métrica AUC-ROC de esta tabla fue calculada utilizando las predicciones binarias del modelo, lo que ofrece solo una estimación aproximada del AUC. Sin embargo, para evaluar correctamente la capacidad discriminativa del modelo, se ha representado la curva ROC completa en la Figura 48, utilizando las probabilidades estimadas para la clase "Yes". Con este enfoque, se obtuvo un valor de AUC del 81%.

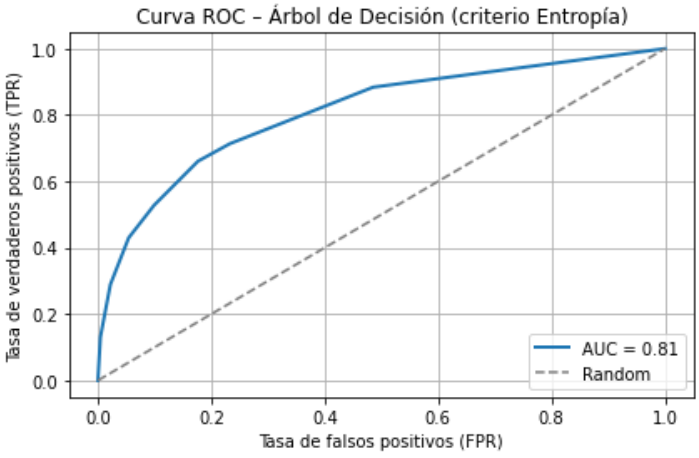

Figura 48: Curva ROC del modelo de Árbol de Decisión.

Además, se ha representado gráficamente cuáles son las 10 variables más importantes del modelo (véase Figura 49). Las variables `Humidity3pm`, `WindGustSpeed`, `Sunshine` y `Pressure3pm` son las más relevantes.

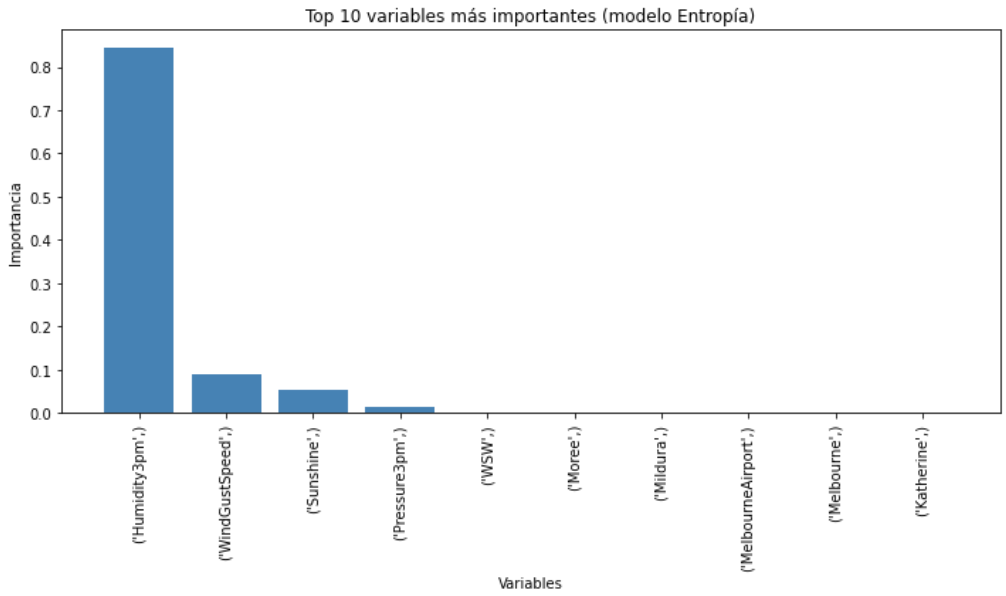

Figura 49: Top 10 variables más importantes del árbol de decisión.

El modelo fue evaluado sobre el conjunto de *test*, obteniendo la matriz de confusión que se muestra en la Figura 50. Existe una tendencia a predecir la clase "No" cuando en realidad la clase debería ser "Yes". Esto refuerza la idea de que el desbalance de clases en los datos está influenciando el modelo.

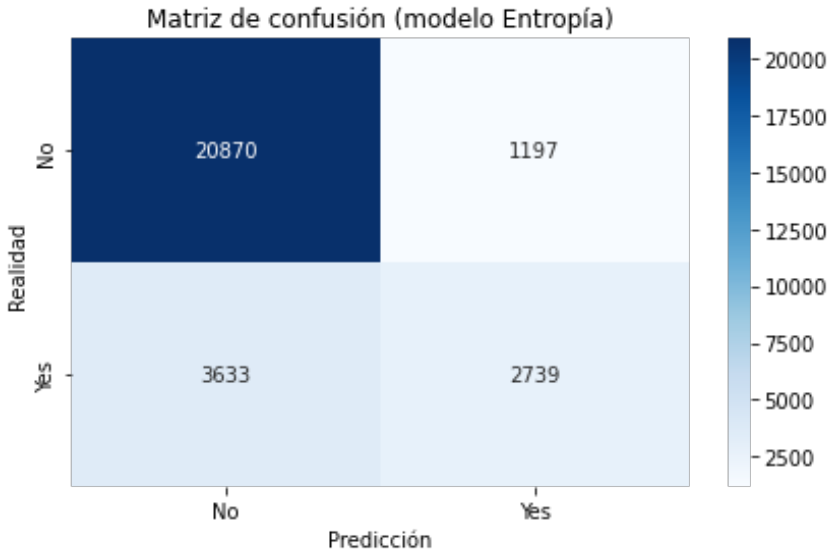

Figura 50: Matriz de confusión del árbol de decisión.

Es importante prestar atención al informe de clasificación obtenido para cada clase, que se muestra en la Tabla 21. La clase "No" tiene muy buenos resultados en todas las métricas, mientras que la clase "Yes" refleja un rendimiento peor. Los valores promedio pueden ayudar a tener una visión más global del modelo.

Tabla 21: Informe de clasificación del árbol de decisión.

Clase	***Precision***	***Recall***	***F1-score***	***Soporte***
No	0,85	0,95	0,90	22.067
Yes	0,70	0,43	0,53	6.372
Promedio macro	0,77	0,69	0,71	28.439
Promedio ponderado	0,82	0,83	0,81	28.439

Por último, el consumo energético de este modelo fue de 0,025 Wh, el tiempo de entrenamiento fue de 4,6 segundos y se tardó aproximadamente 11 segundos en ajustarlo.

Esto indica que el coste computacional no fue tan alto, probablemente por la poca profundidad de los árboles y porque sólo se probaron dos modelos distintos.

6.2.3 RANDOM FOREST

El siguiente modelo entrenado fue el de Random Forest, por su relación con los árboles de decisión. Antes de comenzar a describir detalladamente el entrenamiento, es importante mencionar que en este modelo no se aplicó la codificación *OneHotEncoding* ni el escalado con *MinMax* del que se habló en la sección 6.1.3, sino que se aplicó codificación ordinal a las variables categóricas. Los Random Forest no se ven afectados en su rendimiento por el orden artificial introducido, por lo que codificar las variables categóricas con *OrdinalEncoder* es una mejor solución, ya que se trata de una codificación más eficiente. Por otro lado, su robustez frente a valores atípicos no ha podido comprobarse debido a que estos han sido eliminados durante el preprocesamiento.

Para evaluar su rendimiento, se entrenaron distintos modelos variando el número de árboles de decisión que tendría cada uno. La mejor *accuracy* se obtuvo con 500 árboles: 85.59%. En la Tabla 22 se muestra la *accuracy* obtenida para cada modelo.

Tabla 22: Accuracy obtenida según el número de estimadores en Random Forest.

Número de árboles de decisión	***Accuracy***
10	0,8419
100	0,8536
200	0,8545
500	0,8559

A continuación, se estimó la importancia relativa de cada variable, como se muestra en la Figura 51. A partir de estos resultados, se eliminaron las cinco variables menos relevantes, se obtuvo una *accuracy* final de 0,8547. Aunque es ligeramente menor, eliminar cinco variables mejora notablemente la eficiencia del modelo. Por otro lado, en la imagen puede

verse como la variable `Humidity3pm` aporta más del 16% de la predicción, un valor notablemente más alto que el del resto de las variables, que no llegan al 8%.

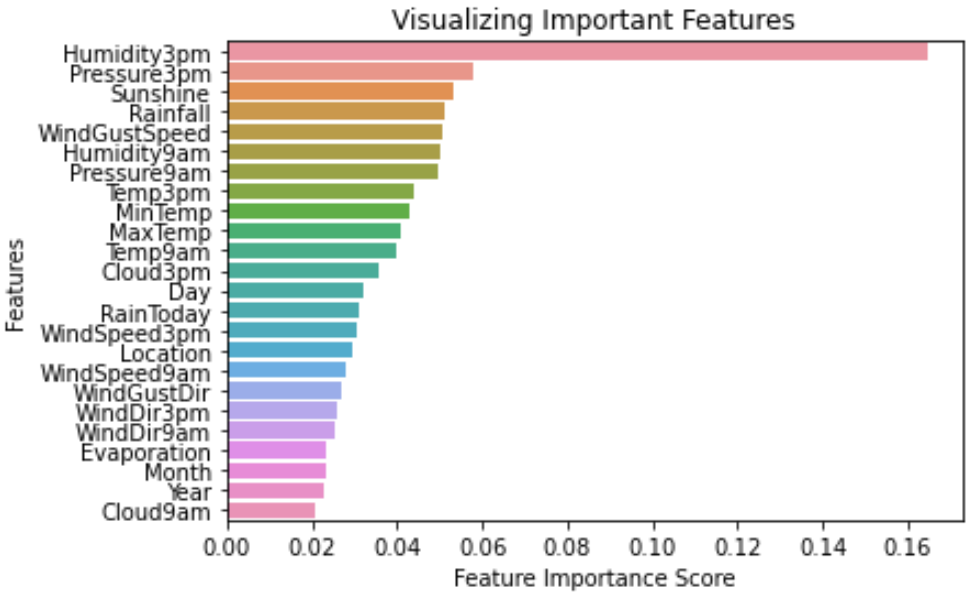

Figura 51: Importancia de las variables del modelo en Random Forest.

Se evaluó el modelo final sobre el conjunto de *test*, obteniéndose las métricas que se muestran en la Tabla 23. El modelo ha demostrado un muy buen rendimiento, y se confirma que el modelo generaliza bien al conjunto de test, por lo que no hay indicios de *overfitting*, a pesar de haber utilizado 500 árboles. Esto concuerda con una de las ventajas principales del algoritmo, su menor propensión al sobreajuste.

Tabla 23: Métricas del modelo Random Forest.

Métrica	***Valor***
Accuracy	85,47%
Recall macro	73%
F1-Score ponderado	84%
AUC-ROC	88,36%

Estas métricas se han conseguido, en su mayoría, a partir del informe de clasificación detallado por clase que se muestra en la Tabla 24. Estos resultados confirman la alta precisión que se atribuye a los modelos de Random Forest, además de un buen equilibrio entre clases, lo que lo convierte en un modelo robusto frente a *datasets* desbalanceados.

Tabla 24: Informe de clasificación para Random Forest.

Clase	***Precision***	***Recall***	***F1-score***	***Soporte***
No	0,87	0,95	0,91	36.466
Yes	0,76	0,51	0,61	10.458
Macro avg	0,81	0,73	0,76	46.924
Weighted avg	0,85	0,85	0,84	46.924

La matriz de confusión de este modelo se muestra en la Figura 52. A partir de ella se puede concluir que el modelo muestra un alto rendimiento para la clase mayoritaria ("No"), pero tiene dificultades para identificar correctamente los días con lluvia. Esto concuerda con el *recall* bajo obtenido para la clase "Yes" del informe de clasificación. Aunque Random Forest es robusto frente a clases desbalanceadas, tiene todavía margen de mejora.

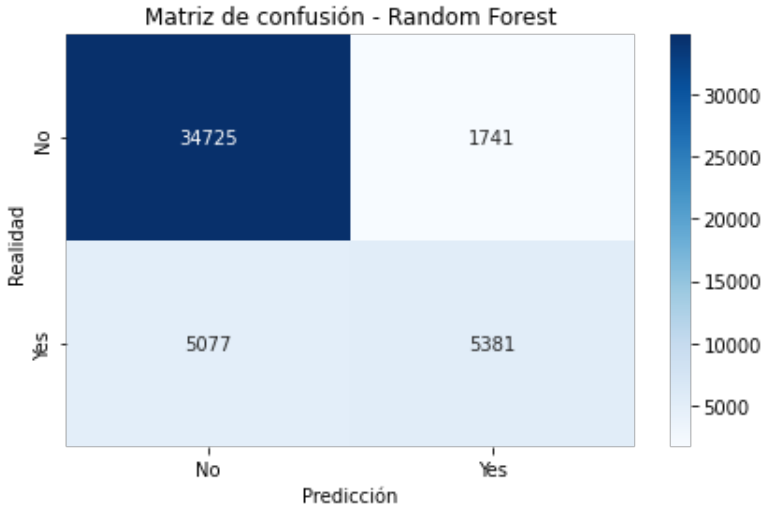

Figura 52: Matriz de confusión para el modelo Random Forest.

La curva ROC se puede visualizar en la Figura 53. Su valor elevado (88,36%) demuestra la buena capacidad del modelo para distinguir las clases.

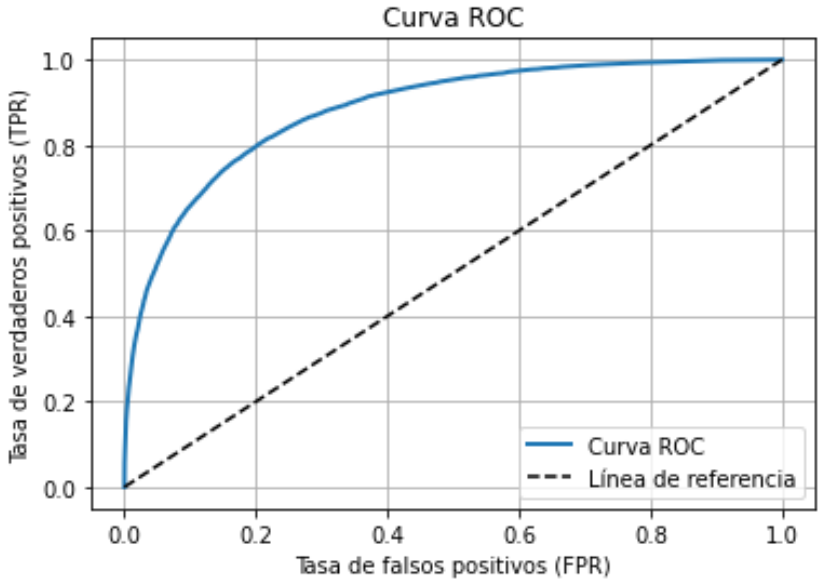

Figura 53: Curva ROC del modelo Random Forest.

Entrenar este modelo ha tardado aproximadamente 9 minutos y ha consumido 1,247 Wh de energía. Aunque no son cifras extremadamente altas, es muy probable que otros modelos

puedan conseguir resultados parecidos con menos recursos de tiempo y energía. Por otro lado, el tiempo de ajuste del modelo ha sido de unos 11 segundos, una cifra muy razonable.

6.2.4 REGRESIÓN LOGÍSTICA

La regresión logística es una de las técnicas más utilizadas en problemas de clasificación binaria. Antes de comenzar con el entrenamiento, se llevó a cabo el preprocesado mencionado en la sección 6.1.3. El modelo fue configurado inicialmente con los siguientes parámetros [55]:

- `solver='liblinear'`: Este parámetro especifica el algoritmo utilizado para optimizar los coeficientes del modelo. `'liblinear'` es una buena opción para bases de datos pequeñas, como en este caso.
- `penalty='l2'`: Indica el tipo de regularización utilizada. Es la regularización asignada por defecto, y añade una penalización proporcional al cuadrado de los coeficientes del modelo.
- `C=1`: Controla la fuerza de la regularización: cuanto más pequeño es este valor, mayor es la regularización.

A continuación, se configuró el mismo modelo, pero con `C=100` para observar los nuevos resultados obtenidos. En la Tabla 25 pueden verse los valores de accuracy en el entrenamiento y en el *test* para estos dos modelos. Se comprueba que aumentar el valor de C no supone una ganancia significativa en el rendimiento del modelo. Este resultado demuestra también la baja tendencia al overfitting que tienen los modelos de regresión logística.

Tabla 25: Accuracy obtenida para distintos valores del parámetro C en regresión logística.

Valor del parámetro C	***Accuracy en el entrenamiento***	***Accuracy en el test***
1	84,76%	85,02%
100	84,78%	85,05%

Para validar sistemáticamente los mejores parámetros, se utilizó `GridSearchCV` con validación cruzada con 5 pliegues `(cv=5)`, es decir, se dividió el conjunto de entrenamiento en 5 partes y se entrenó el modelo 5 veces, utilizando en cada iteración cuatro partes para el entrenamiento y una para *test*. Se llegó así a la conclusión de que los parámetros escogidos inicialmente eran la mejor opción. La validación cruzada dio como resultado una *accuracy* media del 84,74%, lo que indica una buena estabilidad del modelo.

Evaluando el modelo sobre el conjunto de *test*, se obtuvieron los resultados mostrados en la Tabla 26. Además, se calculó la *null accuracy*, que es la precisión que se obtendría prediciendo siempre la clase mayoritaria ("No"), obteniéndose un 77,59%. Comparando este resultado con la *accuracy* que se muestra en la tabla, se confirma que el modelo ha sido capaz de aprender relaciones complejas entre las variables más allá del desbalance de clases, a pesar de las limitaciones típicas de la regresión logística para capturar relaciones complejas.

Tabla 26: Métricas del modelo de regresión logística.

Métrica	***Valor***
Accuracy	0,8502
***Recall* macro**	0,7312
F1 ponderado	0,8400
AUC-ROC	0,8729

Una de las ventajas de estos modelos es que devuelven la probabilidad de que un ejemplo pertenezca a la clase positiva. Esto permite ajustar el umbral de decisión en función de si el objetivo del modelo es priorizar sensibilidad o la precisión en la clase negativa. Para ello, se evaluaron distintos umbrales entre 0,1 y 0,5 y se calculó la *accuracy*, la sensibilidad y la precisión en la clase negativa (*specificity*) para cada uno de ellos (véase Tabla 27). Se observa que reducir el umbral aumenta la sensibilidad, que es la capacidad para detectar días de lluvia, a costa de aumentar los falsos positivos. Esto indica que la regresión logística

ofrece flexibilidad para adaptar el modelo a distintos escenarios según el coste de los errores, lo que constituye una gran ventaja.

Tabla 27: Métricas del modelo en función del umbral de clasificación.

Umbral	***Accuracy***	***Sensibilidad***	***Specificity***
0,1	0,6523	0,9142	0,5767
0,2	0,7808	0,8063	0,7734
0,3	0,8291	0,7061	0,8646
0,4	0,8455	0,6050	0,9150
0,5	0,8502	0,5157	0,9468

En la Figura 54 se muestra la matriz de confusión para el umbral estándar de 0,5. Se observa que la clase mayoritaria se predice con alta precisión, pero la clase minoritaria tiene un rendimiento más limitado.

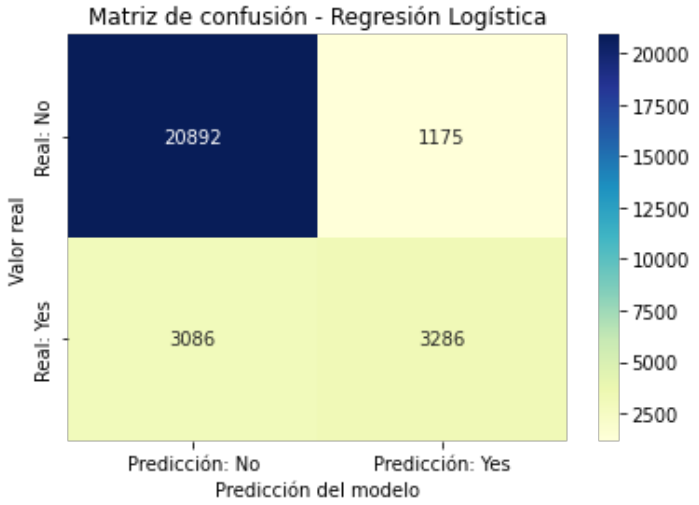

Figura 54: Matriz de confusión del modelo de regresión logística.

La Figura 55 representa la curva ROC, que muestra el rendimiento del modelo en todos los umbrales. Se obtuvo un valor de AUC de 0,8729, que indica una excelente capacidad discriminativa del modelo entre clases.

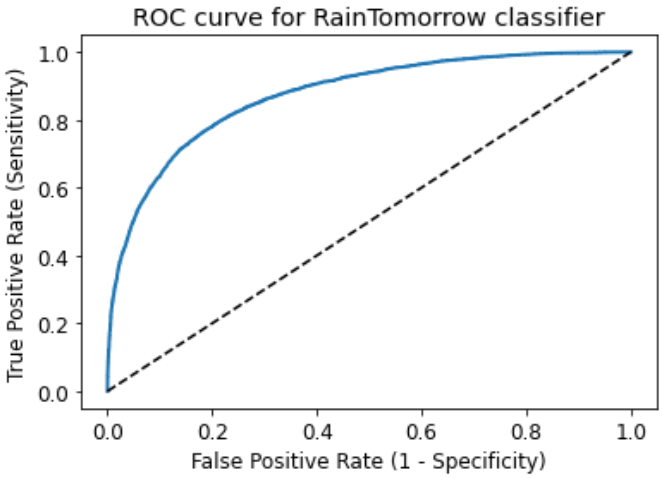

Figura 55: Curva ROC del modelo de regresión logística.

Una de las ventajas principales de la regresión logística es su alta interpretabilidad, ya que cada variable tiene un coeficiente asociado que indica su efecto sobre la probabilidad de que ocurra el evento que se está prediciendo (en este caso, que llueva al día siguiente). En la Tabla 28 se muestran los coeficientes más influyentes del modelo. Se observa que una presión atmosférica elevada a las 3 de la tarde reduce la probabilidad de lluvia al día siguiente, mientras que, por ejemplo, una mayor velocidad del viento aumenta la probabilidad. Esto demuestra que los modelos de regresión logística tienen la ventaja de ser muy interpretables.

Tabla 28: Principales coeficientes del modelo de regresión logística.

Variable	***Coeficiente***
`Pressure3pm`	–10,21
`WindGustSpeed`	6,71
`Pressure9am`	6,44
`Humidity3pm`	5,79
`MaxTemp`	–2,70
`Sunshine`	–1,55
`RainToday_0`	–1,48
`Temp3pm`	1,48
`RainToday_1`	–1,30
`Temp9am`	1,06
Intercepto	-2,77

Por otro lado, la Figura 56 representa un histograma de las probabilidades predichas para la clase positiva. Se puede observar que existe una alta concentración de valores en el intervalo bajo, lo que indica que el modelo asigna una probabilidad de lluvia baja a la mayoría de los ejemplos.

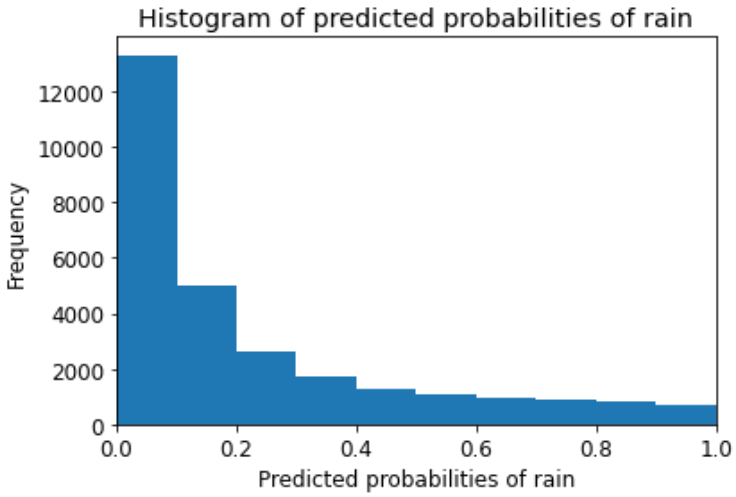

Figura 56: Histograma de probabilidades predichas para la clase positiva en regresión logística.

Para finalizar, se muestra el informe de clasificación para ambas clases con el umbral estándar en la Tabla 29.

Tabla 29: Informe de clasificación del modelo de regresión logística.

Clase	*Precision*	*Recall*	*F1-score*	*Soporte*
No	0,87	0,95	0,91	22067
Yes	0,74	0,52	0,61	6372
Macro avg	0,80	0,73	0,76	28439
Weighted avg	0,84	0,85	0,84	28439

Este modelo ha tardado aproximadamente 2 minutos en entrenarse y ha tenido un consumo energético de 0,309 Wh. Se trata de un modelo rápido y con un bajo coste computacional, fácil de implementar y entrenar. Su ajuste ha tardado 24 segundos, una cifra muy razonable.

6.2.5 REDES NEURONALES MULTICAPA (MLP)

Se ha entrenado un modelo de red neuronal multicapa, de nuevo para predecir si lloverá o no al día siguiente. La arquitectura seleccionada está formada por dos capas ocultas, con 64

y 32 neuronas respectivamente. Se ha escogido la función de activación por defecto (`relu`), una de las más utilizadas por su simplicidad y eficacia. Además, para el entrenamiento se fijó un máximo de 1.000 iteraciones y una semilla fija (`random_state=42`) para asegurar la reproducibilidad del modelo.

Una vez entrenado, el modelo obtuvo unas estadísticas que demuestran un buen rendimiento (véase Tabla 30). El modelo muestra un buen rendimiento global en precisión, pero obtiene un *recall* ligeramente inferior para la clase minoritaria. Este comportamiento puede relacionarse con la sensibilidad al *overfitting* de las redes neuronales multicapa, especialmente cuando no se utilizan conjuntos de datos con clases balanceadas.

Tabla 30: Informe de clasificación del modelo MLP.

Clase	Precisión	*Recall*	*F1-score*	Soporte
No	0,89	0,90	0,90	22 067
Yes	0,65	0,63	0,64	6 372
Macro avg	0,77	0,76	0,77	28 439
Weighted avg	0,84	0,84	0,84	28 439

Las métricas más importantes de este modelo pueden verse en la Tabla 31. Se han obtenido muy buenos resultados.

Tabla 31: Métricas del modelo MLP.

Métrica	*Valor*
Accuracy	0,8400
***Recall* macro**	0,7600
F1 ponderado	0,8400
AUC-ROC	0,8675

A continuación, en la Figura 57, se muestra la matriz de confusión del modelo, observándose que este clasificador comete más errores al identificar casos en los que sí llueve.

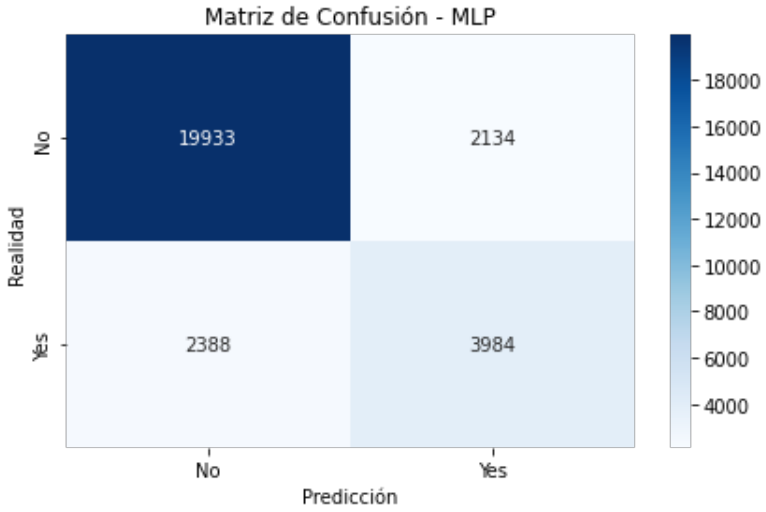

Figura 57: Matriz de confusión del modelo MLP.

Por otro lado, la Figura 58 muestra la curva ROC del modelo, obteniéndose una AUC del 86,75%, lo que indica una elevada capacidad disciminativa.

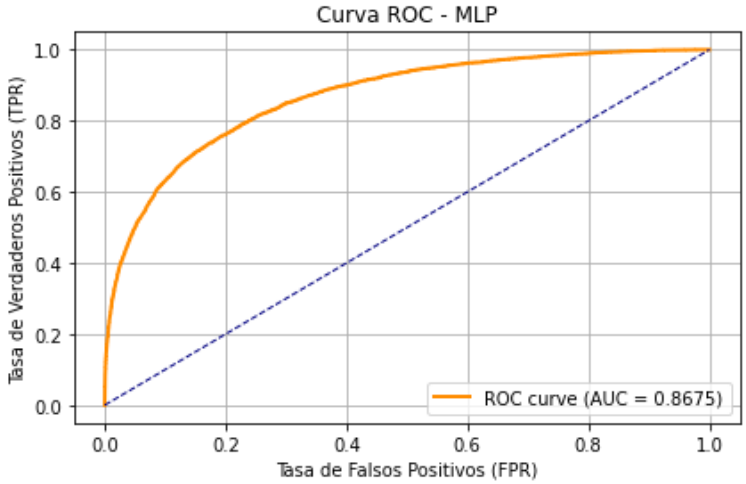

Figura 58: Curva ROC del modelo MLP.

Este modelo ha consumido 0,525 Wh de energía y ha tardado casi 4 minutos en entrenarse, unas cifras relativamente elevadas para tratarse de un modelo sencillo. Al contrario, la predicción ha sido bastante rápida.

6.2.6 KOLMOGOROV-ARNOLD NETWORK (KAN)

En esta sección se describe el proceso de entrenamiento de una red Kolmogorov-Arnold (KAN) sobre el conjunto de datos pequeño. La red fue definida con una arquitectura compuesta por 24 nodos de entrada correspondientes a las variables de la base de datos, tres capas ocultas distribuidas en [10, 5, 2] nodos respectivamente y dos nodos de salida, uno para cada clase. El entrenamiento se ha llevado a cabo utilizando CPU, dado que no se dispone de compatibilidad con CUDA en el entorno de ejecución.

En una fase inicial, se realizó un ajuste de la red con la siguiente arquitectura:

- `grid = 5`
- `k = 3`
- `seed = 0`

Este entrenamiento inicial se realizó en 8 pasos con regularización suave (`lamb = 0.01`, `lamb_entropy = 10`), para comprobar el correcto funcionamiento del modelo e intentar generar una animación de su evolución. Sin embargo, debido al gran número de variables de entrada y a la profundidad de la red, la representación gráfica no pudo llevarse a cabo. En la Figura 59 se muestra uno de los gráficos generados, donde se puede observar la densidad de conexiones y la superposición de los nodos.

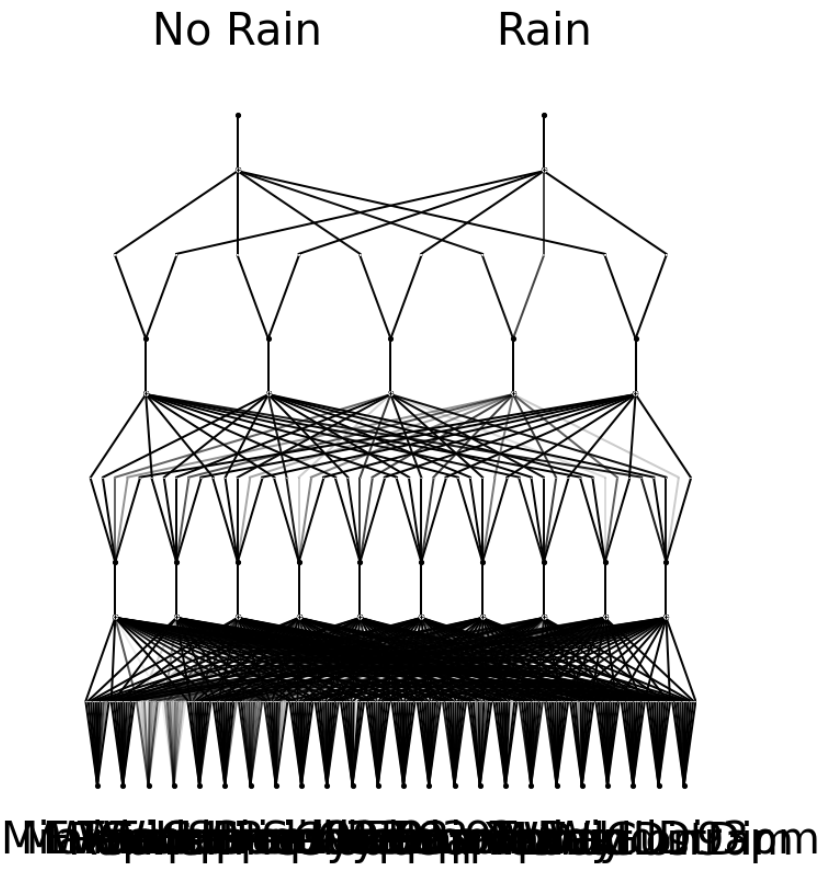

Figura 59: Visualización de la arquitectura del modelo KAN.

Se realizó a continuación un entrenamiento completo de 50 pasos del mismo modelo, pero con función de pérdida `CrossEntropyLoss`, optimizador `Adam`, `lamb = 0,0005` y `lamb_entropy = 0,1` para mejorar la generalización sin bloquear el aprendizaje.

Finalizado el entrenamiento, se aplicó un proceso de poda con umbrales conservadores:

- `node_th = 1e-4`
- `edge_th = 1e-4`

El objetivo fue eliminar conexiones y nodos irrelevantes, simplificando la estructura. Se obtuvieron entonces las métricas que se resumen en la Tabla 32. El modelo alcanzó un buen rendimiento, aunque con un *recall* inferior para la clase minoritaria.

Tabla 32: Métricas de evaluación del modelo KAN.

Métrica	***Valor***
Accuracy	0,8123
***Recall* macro**	0,7300
F1 ponderado	0,8100
AUC-ROC	0,8262

Además, el informe de clasificación que se muestra en la Tabla 33, permite observar más en detalle el rendimiento por clase:

Tabla 33: Informe de clasificación del modelo KAN.

Clase	***Precisión***	***Recall***	***F1-score***	***Soporte***
No Rain	0.88	0.88	0.88	22098
Rain	0.57	0.58	0.58	6341
Macro avg	0.73	0.73	0.73	28439
Weighted avg	0.81	0.81	0.81	28439

Durante el proceso completo, el tiempo de ajuste fue de aproximadamente 4 minutos, el tiempo de entrenamiento fue de aproximadamente 9 minutos y 30 segundos y el consumo energético total fue de 2,496 Wh. Estas cifras reflejan el alto consumo computacional de los modelos KAN.

6.3 COMPARACIÓN DE RESULTADOS

Para finalizar el Capítulo 6, se compararán los resultados obtenidos durante el entrenamiento de los seis modelos con el conjunto de datos de las lluvias en Australia. En la Tabla 34 se

muestran estos resultados. A continuación, se llevará a cabo una comparación detallada de cada métrica.

Tabla 34: Comparativa de resultados de los seis modelos evaluados sobre el caso sencillo.

Modelo	*Accuracy*	*Sensibilidad*	*F1 Score*	*AUC*	*Tiempo de ajuste*	*Tiempo de entrenamiento*	*Uso energético*
SVM	86,03%	73,72%	84,89%	88,54%	9,4 s.	9732,9 s.	22,480 Wh
Árbol de Decisión	83,02%	69,00%	53,14%	80,82%	10,9 s.	4,6 s.	0,025Wh
Random forest	85,47%	73,00%	84,00%	88,36%	11,2 s.	555 s.	1,245Wh
Regresión logística	85,02%	73,12%	84,00%	87,29%	23,9 s.	117,7 s.	0,309Wh
MLP	84,00%	76,00%	84,00%	86,75%	10,8 s.	232,2 s.	0,525Wh
KAN	81,23%	73,00%	81,00%	82,62%	245,2 s.	568,3 s.	2,496Wh

6.3.1 PRECISIÓN O ACCURACY

La precisión global o accuracy mide la proporción de predicciones correctas sobre el total. Es decir, evalúa la capacidad del modelo para distinguir correctamente entre los dos eventos posibles.

En este caso, el mejor resultado lo ha obtenido SVM (86,03%), lo cual es coherente con la ventaja teórica que señala su alta capacidad de generalización. En segundo lugar se sitúa Random Forest (85,47%), con una precisión muy competitiva que respalda su reputación como técnica robusta y precisa. Se encuentra a continuación el modelo de regresión logística (85,02%), que ha logrado una *accuracy* muy próxima a los modelos más complejos a pesar de su simplicidad, lo que demuestra su fiabilidad cuando el preprocesado de datos es adecuado.

El modelo MLP ha obtenido también un valor destacable (84,00%), confirmando que estos modelos son capaces de capturar relaciones complejas entre variables incluso en conjuntos de datos pequeños. Se encuentra a continuación el árbol de decisión (83,02%), cuyo

resultado puede estar relacionado con las limitaciones que presentan estos modelos cuando se trabaja con estructuras poco profundas, como la propensión al *overfitting* o la poca capacidad para representar relaciones complejas.

El modelo KAN se sitúa en el último lugar (81,23%). Si bien el resultado obtenido es razonablemente alto, refleja ciertas limitaciones que pueden estar asociadas a la inestabilidad de la última versión de la librería `pykan`, lo que se describirá en más profundidad en futuros capítulos.

6.3.2 SENSIBILIDAD O RECALL

La sensibilidad o *recall*, que en este caso es el recall macro, calcula la media del recall por clase, otorgando el mismo peso a la clase mayoritaria y minoritaria. Esto es especialmente útil en conjuntos de datos desbalanceados, ya que permite evaluar la capacidad del modelo para identificar ambas clases sin sesgo.

El modelo que ha obtenido la mayor sensibilidad ha sido MLP (76,00%), lo que indica que ha sido el más eficaz detectando los días de lluvia. Esto es coherente con la ventaja teórica de las redes neuronales de capturar relaciones complejas en los datos.

A continuación, se encuentran SVM (73,72%), Regresión Logística (73,12%), Random Forest (73,00%) y KAN (73,00%), todos con valores muy similares, lo cual indica un buen desempeño general en esta métrica. Cabe destacar que KAN iguala a modelos más consolidados en sensibilidad, a pesar de su menor precisión general, lo que sugiere que es competitivo a la hora de detectar eventos positivos.

Por último, el árbol de decisión obtuvo la menor sensibilidad (69,00%). Esta limitación puede explicarse por la baja profundidad del árbol entrenado. Además, la propensión de estos modelos al *overfitting* se ve compensada aquí por una estructura demasiado simple, que pierde capacidad de detección para la clase minoritaria.

6.3.3 F1-SCORE

El F1-score ponderado es una métrica que combina precisión y sensibilidad en una sola medida, ponderando el resultado por el número de muestras de cada clase. En conjuntos de

datos con clases desbalanceadas, esta medida permite tener una visión más realista del rendimiento global del modelo.

En este caso, el modelo con mejor F1-score fue SVM (84,89%), seguido muy de cerca por Random Forest, Regresión Logística y MLP (todos con un 84,00%). Estos cuatro modelos muestran un rendimiento muy similar, lo que sugiere que son capaces de mantener un buen equilibrio entre falsos positivos y falsos negativos.

El modelo KAN obtuvo un F1-score algo inferior (81,00%) al de los modelos anteriores, pero aun así competitivo. El modelo ha logrado un equilibrio razonable entre clases, especialmente considerando su novedad y bajo nivel de madurez tecnológica.

En último lugar se encuentra el árbol de decisión, con un F1-score notablemente más bajo (53,14%). Este resultado refleja su gran desequilibrio en el tratamiento de clases.

6.3.4 ÁREA BAJO LA CURVA ROC (AUC)

El área bajo la curva ROC (AUC) mide la capacidad del modelo para distinguir entre clases, independientemente del umbral de decisión.

Los modelos con mayor AUC han sido SVM (88,54%) y Random Forest (88,36%). Esto confirma la elevada capacidad de discriminación de ambos algoritmos. En el caso de SVM, este resultado es coherente con su ventaja teórica para generar márgenes óptimos entre clases, mientras que en Random Forest se refleja su fortaleza en estabilidad y precisión al combinar múltiples árboles de decisión.

En tercera posición se encuentra Regresión Logística (87,29%), lo cual es destacable teniendo en cuenta la sencillez del modelo. Esto reafirma que es un modelo muy adecuado en tareas de clasificación binaria, especialmente cuando se busca un equilibrio entre rendimiento y eficiencia.

El modelo MLP obtuvo un AUC muy similar al de los modelos anteriores (86,75%). Queda así reflejada la capacidad de este modelo para aprender patrones no lineales y complejos, aunque con cierta sensibilidad al *overfitting*.

Por otro lado, el modelo KAN alcanzó un AUC algo más bajo que el de los modelos anteriores (82,62%). Aunque sigue siendo una buena capacidad discriminativa, esta cifra

indica que aún hay margen de mejora. Tal y como se ha explicado en secciones anteriores, las posibles causas incluyen el uso de la última versión de pykan (0.2.8), la falta de aceleración por GPU y las limitaciones en el ajuste fino de hiperparámetros. Aun así, destaca que el modelo KAN haya superado el umbral del 80% en AUC en un problema con 24 variables.

Finalmente, el Árbol de Decisión simple obtuvo el valor más bajo (80,82%). Este modelo se ve penalizado por su baja capacidad para detectar correctamente la clase minoritaria y por su limitada profundidad.

6.3.5 EFICIENCIA

La eficiencia computacional de un modelo es una métrica clave, especialmente en contextos en los que se requiere escalabilidad o los recursos son limitados. Para ello, se han analizado el tiempo total de ajuste y entrenamiento de los modelos y su consumo energético medido en Wh.

El modelo más eficiente ha sido el árbol de decisión, con un tiempo total de ajuste y entrenamiento inferior a 20 segundos y un consumo de tan solo 0,025 Wh. Su eficiencia lo convierte en una buena opción cuando la rapidez y el bajo coste energético son una prioridad.

Le sigue la regresión logística, que ha requerido únicamente 24 segundos de ajuste y 2 minutos de entrenamiento, con un consumo energético bajo de 0,309 Wh.

En tercera posición se encuentra el modelo MLP, con un tiempo de entrenamiento de aproximadamente 4 minutos y un consumo de 0,525 Wh. Estas cifras pueden considerarse moderadas teniendo en cuenta la complejidad del modelo.

El modelo Random Forest ha mostrado un coste computacional mayor: 555 segundos de entrenamiento y 1,245 Wh de energía consumida. A continuación aparece KAN, con más de 9 minutos de entrenamiento, 245 segundos de ajuste y un consumo de 2,496 Wh. Esto confirma la desventaja teórica de la ineficiencia de las KAN y su lentitud.

Por último, el modelo SVM ha sido el más costoso computacionalmente. Ha necesitado más de 2 horas y 40 minutos de entrenamiento, 9,4 segundos de ajuste y ha consumido 22,480

Wh. Estos resultados refuerzan una de sus desventajas teóricas más señaladas: su ineficiencia en conjuntos de datos grandes, especialmente cuando se prueban varios kernels.

6.3.6 INTERPRETABILIDAD

La interpretabilidad es una característica importante de los modelos de Machine Learning, y puede ser muy relevante en determinados contextos. En la Tabla 35 se resume la interpretabilidad teórica y práctica de cada modelo.

Tabla 35: Comparativa de la interpretabilidad de los modelos del caso sencillo.

Modelo	***Interpretabilidad teórica***	***Aplicabilidad práctica en este caso***
Árbol de decisión	Muy alta	Alta. Reglas claras y visuales gracias a la baja profundidad.
Regresión logística	Muy alta	Elevada. Coeficientes comprensibles directamente.
KAN	Alta	Media. Interpretabilidad limitada por la complejidad.
Random Forest	Media	Media-baja. Interpretabilidad accesible sólo a través de la importancia de variables.
MLP	Baja	Baja. Estructura difícil de analizar.
SVM	Baja	Baja. Especialmente cuando se utilizan kernels no lineales.

Como se puede observar, solo regresión logística y árboles de decisión ofrecen una buena interpretación del modelo. Random Forest proporciona cierta interpretabilidad únicamente a través del análisis de variables más importantes del modelo.

En cuanto al modelo KAN, su diseño teórico permite extraer interpretaciones gráficas. Sin embargo, en este caso, la complejidad del modelo ha dificultado esta tarea.

Por el contrario, tanto SVM como MLP actúan como cajas negras: la transformación de los datos mediante kernels no lineales en el caso de SVM y la acumulación de capas ocultas en MLP impiden la interpretabilidad de ambos modelos.

Capítulo 7. EVALUACIÓN DE MODELOS SOBRE UN CASO COMPLEJO

El presente capítulo tiene como objetivo aplicar y comparar seis técnicas de clasificación sobre un conjunto de datos complejo, con el fin de observar en la práctica sus principales ventajas y limitaciones. Este análisis sirve para entender el rendimiento de cada modelo en problemas que se asemejan a casos más reales.

Para ello, se ha utilizado el dataset basado en simulaciones del Bosón de Higgs, un conjunto de datos que está orientado a predecir si un evento es señal (*signal*, "s") o fondo (background, "b"). A partir de él, se han entrenado seis modelos distintos: Support Vector Machine (SVM), Árboles de decisión, Random Forest, Regresión logística, Redes Neuronales Multicapa (MLP) y Kolmogorov-Arnold Networks (KAN).

En las secciones siguientes se describirá en primer lugar el conjunto de datos. A continuación, se detallará el ajuste y entrenamiento de cada uno de los modelos. Finalmente, se analizarán y compararán los resultados obtenidos.

7.1 DESCRIPCIÓN DEL CONJUNTO DE DATOS

En esta primera sección se describe el origen, finalidad y estructura del dataset, y a continuación se detalla el preprocesamiento de los datos que se ha llevado a cabo para posteriormente construir los modelos. Por último, se ofrece un análisis exploratorio de los datos.

7.1.1 ORIGEN Y FINALIDAD DEL DATASET

Como se ha descrito en la introducción de este capítulo, el conjunto de datos basado en el Bosón de Higgs tiene como objetivo predecir si un evento es `signal` o `background` [56]. La primera etiqueta indica que el evento podría ser compatible con la producción de un bosón

de Higgs, mientras que la segunda indicaría que el evento corresponde al "ruido de fondo", es decir, a interacciones que no indican la presencia del Bosón de Higgs.

Este conjunto de datos es de dominio público y se ha descargado a través de la página web de *Kaggle*. En este caso, el conjunto de datos fue publicado en la plataforma digital para llevar a cabo una de sus competiciones online de Machine Learning. El *dataset* incluye un conjunto de entrenamiento con 250.000 ejemplos y un conjunto de *test* con 550.000 muestras.

Se ha escogido este conjunto de datos por sus dimensiones y por tratarse de un problema de clasificación binaria, ya que esto facilitará la comparación de algunas características de los modelos como la interpretabilidad.

7.1.2 ESTRUCTURA DEL DATASET

En esta sección se describen las columnas que forman parte de este conjunto de datos. Las variables con el prefijo PRI (de *primitive*) son cifras sobre la colisión de protones tal como las mide el detector. Las variables con el prefijo DER (de *derived*) son cifras derivadas a partir de las primitivas, seleccionadas por los físicos del experimento ATLAS [56], [57].

En la Tabla 36 se muestra un resumen de cada variable.

Tabla 36: Descripción de las variables del dataset del bosón de Higgs.

Variable	***Tipo de dato***	***Descripción breve***
`EventId`	Entero	Identificador único del evento (no se usa como entrada del modelo)
`DER_mass_MMC`	*float*	Estimación de la masa mediante integración probabilística
`DER_mass_transverse_met_lep`	*float*	Masa transversal entre MET y el leptón
`DER_mass_vis`	*float*	Masa invariante del leptón y el tau hadrónico (sin neutrinos)
`DER_pt_h`	*float*	Módulo de la suma vectorial de pT de leptón, tau y MET
`DER_deltaeta_jet_jet`	*float*	Diferencia de pseudorapidez entre los dos jets principales (si existen)
`DER_mass_jet_jet`	*float*	Masa invariante de los dos jets principales (si existen)
`DER_prodeta_jet_jet`	*float*	Producto de pseudorapideces de los dos jets principales (si existen)
`DER_deltar_tau_lep`	*float*	Separación R entre el tau y el leptón
`DER_pt_tot`	*float*	Módulo de la suma vectorial de pT de leptón, tau, MET y jets
`DER_sum_pt`	*float*	Suma escalar de pT de leptón, tau y jets
`DER_pt_ratio_lep_tau`	*float*	Ratio entre los momentos transversales del leptón y el tau
`DER_met_phi_centrality`	*float*	Centralidad del ángulo azimutal de MET respecto al leptón y el tau
`DER_lepa_eta_centrality`	*float*	Centralidad de la pseudorapidez del leptón respecto a los dos jets (si existen)
`PRI_tau_pt`	*float*	Momento transversal del tau hadrónico
`PRI_tau_eta`	*float*	Pseudorapidez del tau hadrónico
`PRI_tau_phi`	*float*	Ángulo azimutal del tau hadrónico
`PRI_lep_pt`	*float*	Momento transversal del leptón
`PRI_lep_eta`	*float*	Pseudorapidez del leptón
`PRI_lep_phi`	*float*	Ángulo azimutal del leptón

`PRI_met`	*float*	Magnitud de la energía transversal faltante (MET)
`PRI_met_phi`	*float*	Ángulo azimutal del MET
`PRI_met_sumet`	*float*	Suma total de energía transversal registrada en el evento
`PRI_jet_num`	Entera (0–3)	Número de jets detectados
`PRI_jet_leading_pt`	*float*	Momento transversal del jet principal (si existe)
`PRI_jet_leading_eta`	*float*	Pseudorapidez del jet principal (si existe)
`PRI_jet_leading_phi`	*float*	Ángulo azimutal del jet principal (si existe)
`PRI_jet_subleading_pt`	*float*	Momento transversal del segundo jet (si existen al menos dos)
`PRI_jet_subleading_eta`	*float*	Pseudorapidez del segundo jet (si existen al menos dos)
`PRI_jet_subleading_phi`	*float*	Ángulo azimutal del segundo jet (si existen al menos dos)
`PRI_jet_all_pt`	*float*	Suma escalar de los momentos transversales de todos los jets
`Weight`	*float*	Peso del evento (no se usa como entrada del modelo)
`Label`	Categórica (s/b)	Variable objetivo: `'s'` (*signal*) o `'b'` (*background*)

7.1.3 PREPROCESAMIENTO DE LOS DATOS

En esta sección se detallan los pasos seguidos durante el preprocesamiento del conjunto de datos antes de entrenar los diferentes modelos de clasificación. Este preprocesamiento () tiene el objetivo principal de limpiar, preparar y transformar los datos para su correcto análisis.

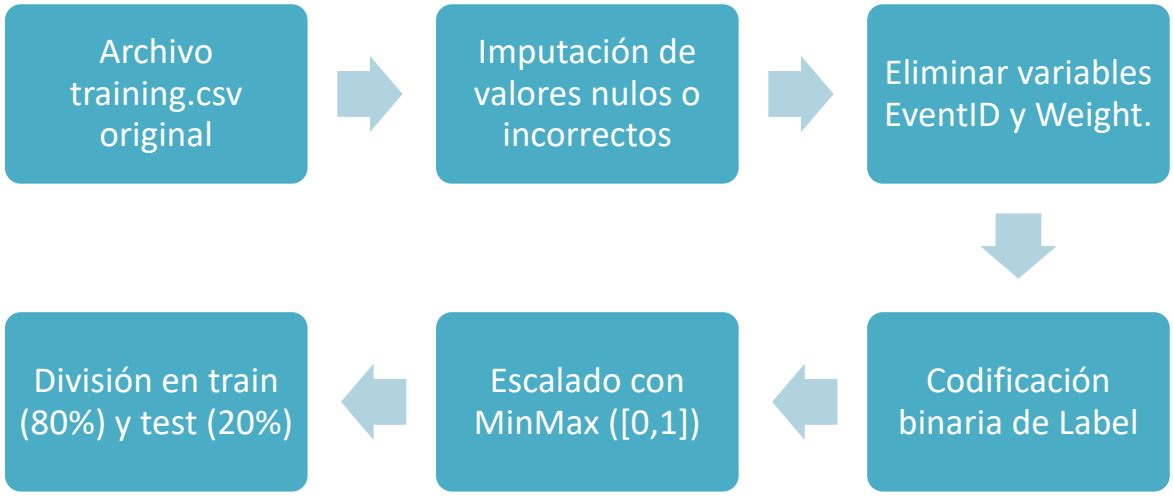

Figura 60: Diagrama de flujo del preprocesamiento aplicado al conjunto de datos complejo.

Gestión de valores nulos o incorrectos

Como en este conjunto de datos algunos valores son calculados a partir de otros, hay ocasiones en las que no se pueden llevar a cabo estas operaciones, y se marcan los valores que no pueden calcularse se codifican como `-999.0`. Estos valores se han reemplazado por valores `NaN`, y, a continuación, se han imputado utilizando la mediana de cada variable. Esta estrategia es robusta frente a `outliers` o valores atípicos.

Selección de variables

Se han eliminado las variables que no deben utilizarse como entradas del modelo. Como ya se mencionó en la sección 7.1.2, dichas variables son `'EventId'` y `'Weight'`.

Codificación binaria de la variable objetivo

La columna `'Label'`, que contiene las clases `'s'` (*signal*) y `'b'` (*background*), ha sido transformada a valores binarios (1 para signal, 0 para background).

Escalado de variables

Se ha aplicado un escalado MinMax a todas las variables, de modo que todas se encuentran en un rango entre 0 y 1. Esto es muy útil especialmente para algunas técnicas como SVM o redes neuronales.

División del conjunto de datos

Finalmente, se ha dividido el conjunto de entrenamiento original en un 80% para entrenamiento y un 20% para validación, utilizando `random_state=42` para garantizar la reproducibilidad.

7.1.4 ANÁLISIS EXPLORATORIO

Antes de entrenar los seis modelos, se ha llevado a cabo un análisis exploratorio para comprender mejor la distribución de las variables, examinar relaciones relevantes entre ellas y detectar posibles desequilibrios en la variable objetivo.

En primer lugar, se ha analizado la distribución de clases del conjunto de datos. Como se observa en la Figura 61, el conjunto de datos está desbalanceado, con la mayoría de los eventos etiquetados como *background*. Esto puede afectar al rendimiento de algunos clasificadores, y se tendrá en cuenta en el posterior análisis.

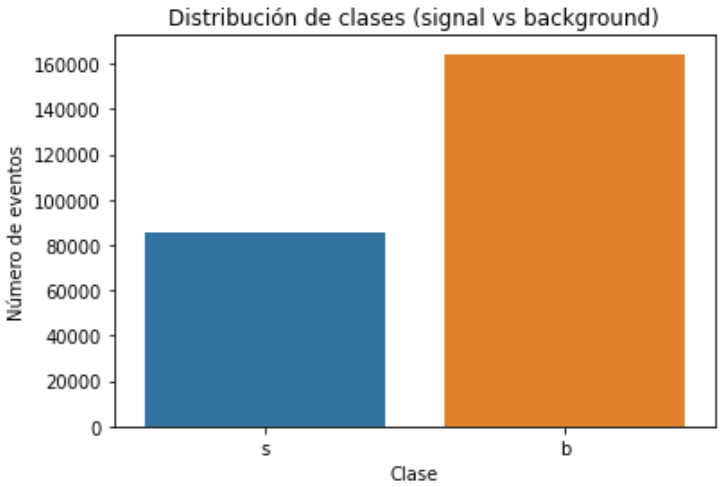

Figura 61: Distribución de clases en el conjunto de datos complejo (`signal` vs. `background`).

A continuación, para detectar relaciones lineales entre variables, se ha graficado la matriz de correlación entre las variables numéricas, como se muestra en la Figura 62. Al existir un número tan grande de variables, no se van a comentar en profundidad, pero se observa que algunas variables están altamente relacionadas, tanto directa como inversamente.

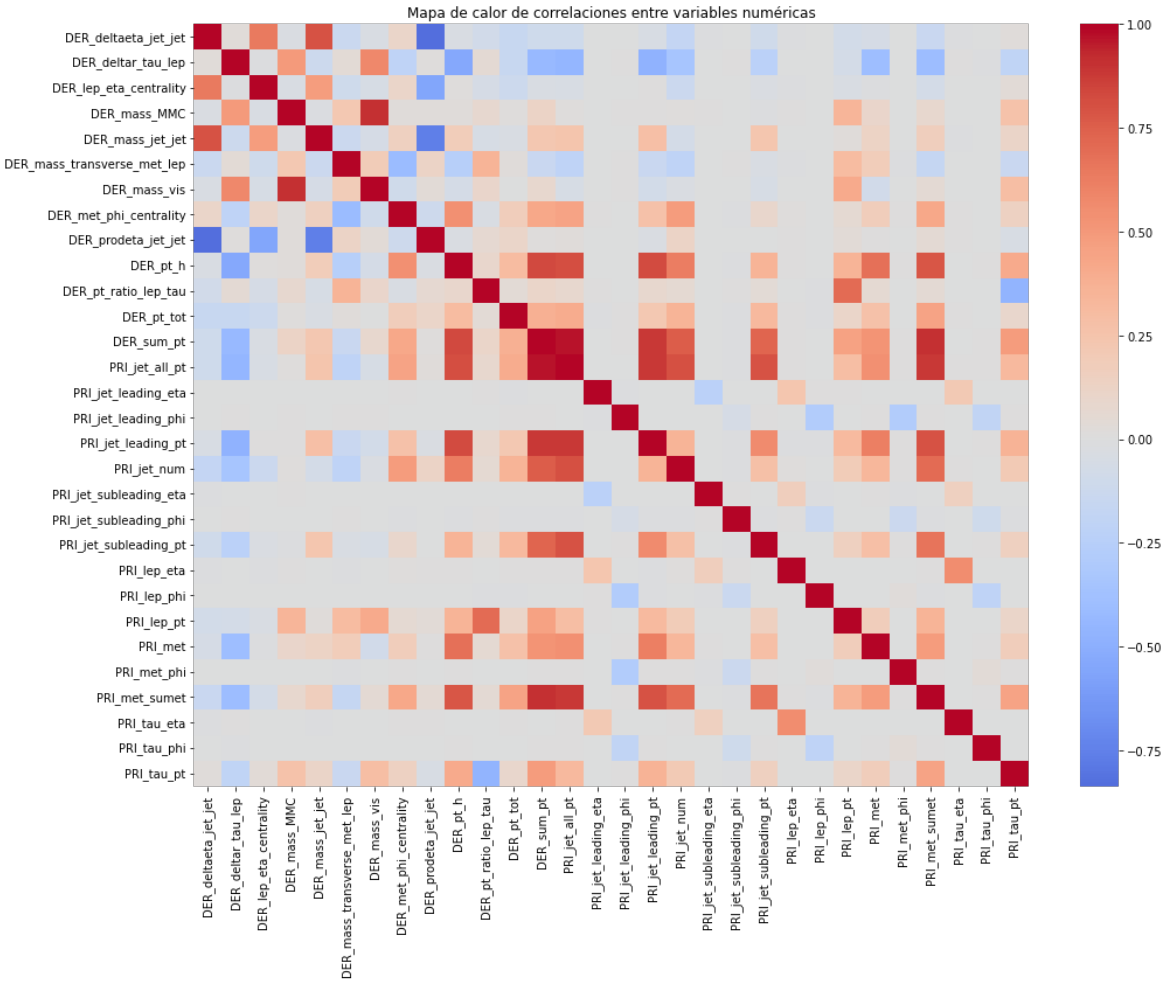

Figura 62: Mapa de calor de las correlaciones entre variables numéricas.

Algunas variables contienen valores -999.0, que se imputaron, como se explicó anteriormente. En la Figura 63 se ha representado el porcentaje de estos valores en las diez variables que más valores inválidos presentan. Se observan hasta siete variables con más de un 50% de valores inválidos, y es por ello por lo que se ha optado por estrategias de imputación y no por la eliminación de estos valores.

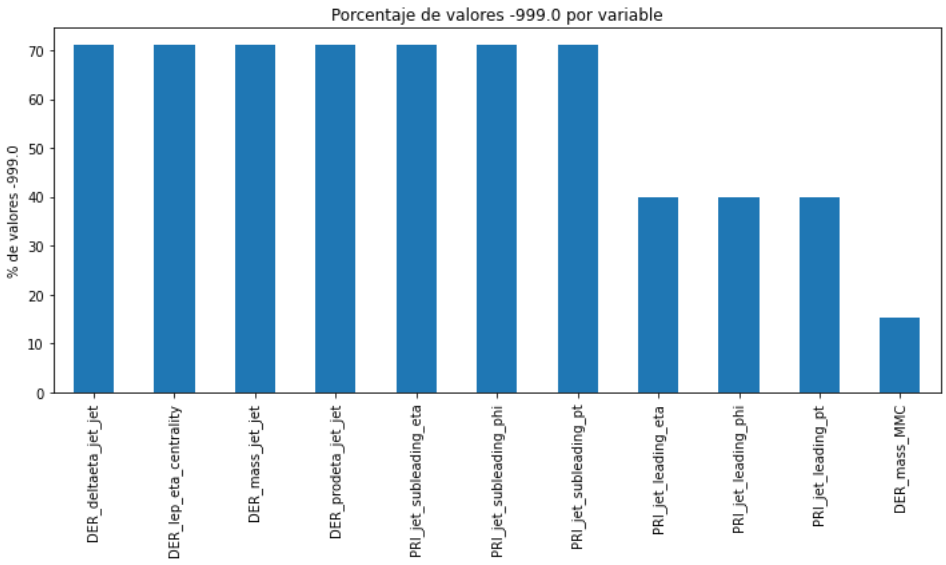

Figura 63: Porcentaje de valores nulos en cada variable.

7.2 ENTRENAMIENTO Y AJUSTE DE LOS MODELOS

Una vez finalizado el preprocesamiento de los datos, se procede al entrenamiento y ajuste de los seis modelos de clasificación seleccionados. La presente sección tiene como objetivo describir detalladamente la configuración de cada modelo, los hiperparámetros utilizados y analizar en qué medida las ventajas y desventajas teóricas se cumplen también en la práctica.

7.2.1 SUPPORT VECTOR MACHINE (SVM)

En la presente sección se describe detalladamente el ajuste y entrenamiento del modelo de Máquina de Vectores de Soporte (SVM) sobre el conjunto de datos del bosón de Higgs. Antes de comenzar, es importante destacar que durante el preprocesado de datos se utilizó *StandardScaler* en lugar de *MinMaxScaler*, ya que este tipo de modelos se benefician más de un escalado centrado en media y varianza.

Para llevar a cabo el ajuste de hiperparámetros, se redujo el conjunto de entrenamiento al 75% de su tamaño, es decir, 150.000 muestras, debido al alto coste computacional de estos modelos. Se definió una rejilla de configuraciones variadas para distintos parámetros y se evaluaron siete modelos diferentes. Las características de estos modelos y los resultados

obtenidos se muestran en la Tabla 37. El modelo con mejor rendimiento fue el número 2, con buenos resultados en *accuracy*, *recall* y *F1-Score* para la clase 1. Este modelo fue posteriormente reentrenado con la totalidad del conjunto de entrenamiento.

Tabla 37: Resultados del tuning de modelos SVM con el 75% del conjunto de entrenamiento para el caso complejo.

Modelo	***Kernel***	***C***	***Gamma***	***Accuracy***	***Recall clase 1***	***F1-Score clase 1***
1	`rbf`	0,5	`scale`	0,8303	0,6957	0,7376
2	`rbf`	1,0	`scale`	0,8320	0,7004	0,7408
3	`rbf`	1,0	0,1	0,8280	0,6828	0,7312
4	`poly`	1,0	`scale`	0,8113	0,6260	0,6945
5	`poly`	0,5	`scale`	0,8077	0,6064	0,6836
6	`linear`	1,0	N/A	0,7492	0,5401	0,5961
7	`sigmoid`	5,0	0,01	0,6796	0,5287	0,5307

A continuación, se muestran los resultados obtenidos por el modelo final en la Tabla 38.

Tabla 38: Métricas del modelo SVM para el caso complejo.

Métrica	***Valor***
Accuracy	0,8338
Recall (clase 1)	0,7036
F1-Score (clase 1)	0,7437
AUC-ROC	0,8954

Finalmente, se ha medido un consumo energético de 26,118 Wh, con un tiempo total de ajuste de 23 minutos y un tiempo total de entrenamiento de aproximadamente 2 horas y 45

minutos. Se confirma así la desventaja de los modelos SVM relacionada con su alto coste computacional y su lentitud, especialmente cuando se trabaja con bases de datos grandes.

7.2.2 ÁRBOLES DE DECISIÓN

En esta sección se explica en detalle el entrenamiento del modelo de árbol de decisión para el conjunto de datos del bosón de Higgs. Antes de comenzar, es importante destacar que en este tipo de modelos no es necesario realizar el escalado de las variables durante el preprocesado para mejorar la interpretabilidad de las reglas de decisión generadas.

Posteriormente, se utilizó *GridSearchCV* para evaluar distintas combinaciones de hiperparámetros: `max_depth`, `min_samples_split`, `min_samples_leaf` y `criterion`. El modelo con los mejores resultados estaba configurado de la siguiente manera:

- criterion = 'entropy'
- max_depth = 10
- min_samples_split = 20
- min_samples_leaf = 5

Se obtuvo así el modelo final, cuyas métricas se muestran en la Tabla 39. Estos resultados reflejan un buen rendimiento general, especialmente para clasificar los eventos pertenecientes a la clase 1, que es la clase minoritaria. Esta sensibilidad hacia la clase menos representada es de especial relevancia, ya que el objetivo principal de este modelo es identificar correctamente los pocos eventos significativos entre muchos de fondo.

Tabla 39: Métricas del árbol de decisión del conjunto de datos complejo.

Métrica	***Resultado***
Accuracy	0,8045
Recall (clase 1)	0,7393
F1-Score (clase 1)	0,7192
AUC-ROC	0,8727

En cuanto a la eficiencia computacional, el entrenamiento tuvo una duración de poco más de 4 minutos, mientras que el ajuste requirió tan solo 10 segundos. Además, se ha registrado un uso total de 0,587 Wh de energía durante el proceso completo.

Además, para facilitar la interpretación del modelo, se ha generado una representación textual de la estructura del árbol, ya que al tener una profundidad de 10 niveles, su representación en una imagen no era una opción viable. Los tres primeros niveles del árbol se muestran a continuación:

```
|--- DER_mass_transverse_met_lep <= 52.48
|   |--- DER_mass_MMC <= 101.14
|   |   |--- DER_deltar_tau_lep <= 2.74
|   |   |   |--- DER_mass_MMC <= 88.71
|   |   |   |   |--- truncated branch of depth 7
|   |   |   |--- DER_mass_MMC >  88.71
|   |   |   |   |--- truncated branch of depth 7
|   |   |--- DER_deltar_tau_lep >  2.74
|   |   |   |--- PRI_tau_pt <= 28.05
|   |   |   |   |--- truncated branch of depth 7
|   |   |   |--- PRI_tau_pt >  28.05
|   |   |   |   |--- truncated branch of depth 7
|   |--- DER_mass_MMC >  101.14
|   |   |--- DER_mass_vis <= 119.66
|   |   |   |--- DER_mass_MMC <= 113.55
|   |   |   |   |--- truncated branch of depth 7
|   |   |   |--- DER_mass_MMC >  113.55
|   |   |   |   |--- truncated branch of depth 7
|   |   |--- DER_mass_vis >  119.66
|   |   |   |--- DER_mass_vis <= 127.66
|   |   |   |   |--- truncated branch of depth 7
|   |   |   |--- DER_mass_vis >  127.66
|   |   |   |   |--- truncated branch of depth 7
|--- DER_mass_transverse_met_lep >  52.48
```

```
|   |--- DER_met_phi_centrality <= 0.52
|   |   |--- PRI_tau_pt <= 34.41
|   |   |   |--- DER_mass_vis <= 55.92
|   |   |   |   |--- truncated branch of depth 7
|   |   |   |--- DER_mass_vis >  55.92
|   |   |   |   |--- truncated branch of depth 7
|   |   |--- PRI_tau_pt >  34.41
|   |   |   |--- DER_mass_vis <= 118.66
|   |   |   |   |--- truncated branch of depth 7
|   |   |   |--- DER_mass_vis >  118.66
|   |   |   |   |--- truncated branch of depth 7
|   |--- DER_met_phi_centrality >  0.52
|   |   |--- DER_mass_MMC <= 170.58
|   |   |   |--- PRI_tau_pt <= 32.78
|   |   |   |   |--- truncated branch of depth 7
|   |   |   |--- PRI_tau_pt >  32.78
|   |   |   |   |--- truncated branch of depth 7
|   |   |--- DER_mass_MMC >  170.58
|   |   |   |--- DER_mass_vis <= 121.92
|   |   |   |   |--- truncated branch of depth 7
|   |   |   |--- DER_mass_vis >  121.92
|   |   |   |   |--- truncated branch of depth 7
```

Si se buscara una mayor interpretabilidad a costa de una menor precisión, podría reducirse la profundidad del árbol.

Por último, se ha comprobado que no existan signos evidentes de sobreajuste, ya que estos modelos tienen cierta propensión al *overfitting*. Se ha obtenido una precisión del 80,45% en *test*, por lo que existe una buena capacidad de generalización.

7.2.3 RANDOM FOREST

Se ha entrenado también el modelo de Random Forest para el conjunto de datos del bosón de Higgs. Antes de comenzar a describir el entrenamiento en profundidad, cabe destacar que, durante el preprocesado de datos, no se aplicó el *MinMaxScaler* como en otros modelos, ya que los algoritmos basados en árboles de decisión no se ven afectados por el rango de valores de entrada.

A continuación, se definió una rejilla de hiperparámetros que incluía distintas configuraciones, y se empleó *GridSearchCV* con validación cruzada de 5 particiones. Se ha utilizado la métrica F1-Score como criterio de evaluación para escoger el mejor modelo, cuyos parámetros se especifican a continuación:

- `criterion='entropy'`: Criterio de impureza.
- `n_estimators=200`: Número de árboles de decisión.
- `max_depth=None`: Profundidad máxima de los árboles.
- `min_samples_split=2`: Mínimo de muestras para dividir un nodo.
- `min_samples_leaf=5`: Número mínimo de muestras por hoja.

Este modelo fue entrenado completamente y evaluado sobre el conjunto de test. En la Tabla 40 se muestra el informe de clasificación para ambas clases.

Tabla 40: Informe de clasificación del modelo Random Forest para el caso complejo.

Clase	***Precisión***	***Recall***	***F1-score***	***Soporte***
0	0,8584	0,9040	0,8806	32 867
1	0,7949	0,7138	0,7522	17 133
Media macro	0,8266	0,8089	0,8164	50 000
Media ponderada	0,8366	0,8388	0,8366	50 000

Por otro lado, las métricas clave del modelo se resumen en la Tabla 41. Los buenos resultados reflejan una buena capacidad del modelo para distinguir entre los dos tipos de eventos.

Tabla 41: Métricas más importantes del modelo Random Forest del caso complejo.

Métrica	***Resultado***
Accuracy	0,8388
Recall (clase 1)	0,7138
F1-Score (clase 1)	0,7522
AUC-ROC	0,9061

Para finalizar, se ha registrado un consumo energético de 9,428 Wh, además de un tiempo para la búsqueda de hiperparámetros de 1 hora y 10 minutos y un tiempo de entrenamiento

posterior también de 1 hora y 10 minutos. Esto refleja un coste computacional relativamente alto, como es habitual en este tipo de modelos.

7.2.4 REGRESIÓN LOGÍSTICA

La regresión logística es un método de clasificación simple y robusto, aplicado en este caso al conjunto de datos del bosón de Higgs. La única diferencia con el preprocesado común aplicado a todos los modelos es que se ha utilizado *StandardScaler* en lugar de *MinMaxScaler*, ya que este tipo de modelos se benefician del escalado centrado en media y varianza.

En una fase inicial, se entrenó el modelo usando únicamente una parte del conjunto de datos, con el objetivo de encontrar una configuración adecuada. Se aplicó validación cruzada con 5 particiones, obteniéndose una media de *accuracy* del 73,68%.

Fue entonces cuando se entrenó el modelo sobre todo el conjunto. La configuración final escogida fue:

- `max_iter=1000`: número máximo de iteraciones.
- `solver='liblinear'`
- `class_weight='balanced'`: para penalizar más los errores sobre la clase minoritaria y compensar el desbalance.

Las métricas obtenidas por este modelo se muestran en la Tabla 42. Se observa un rendimiento equilibrado, con especial énfasis en la detección de la clase minoritaria.

Tabla 42: Métricas obtenidas por el modelo de regresión logística para el conjunto de datos grande.

Métrica	***Resultado***
Accuracy	0,7369
Recall (clase 1)	0,7600
F1-Score (clase 1)	0,6600
AUC-ROC	0,8150

Se muestra también en la Tabla 43 el informe de clasificación por clase del modelo, observándose que la precisión y la F1-Score empeoran cuando se trata de clasificar la clase 1, que es la minoritaria.

Tabla 43: Informe de clasificación para el modelo de regresión logística del caso grande.

Clase	***Precisión***	***Recall***	***F1-score***	***Soporte***
0	0,85	0,72	0,78	32 867
1	0,59	0,76	0,66	17 133

Como se observa en la Figura 64, el modelo comete más errores al clasificar eventos de la clase mayoritaria como si fueran positivos, reflejo del esfuerzo por capturar la clase minoritaria mediante `class_weight='balanced'`.

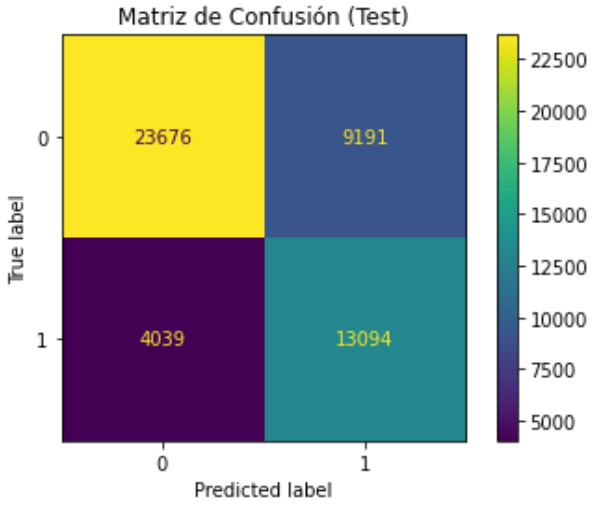

Figura 64: Matriz de confusión del modelo de regresión logística para el caso complejo.

La curva ROC del modelo se muestra en la Figura 65, que, de nuevo, refleja que el modelo tiene una buena capacidad discriminativa entre ambas clases.

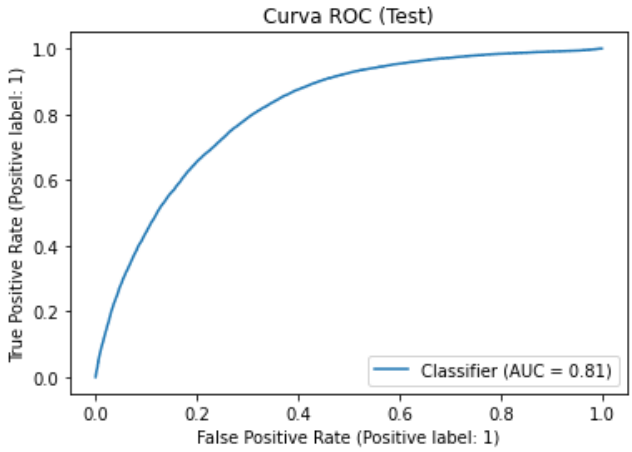

Figura 65: Curva ROC del modelo de regresión logística del caso complejo.

Los modelos de regresión logística presentan la ventaja de que son interpretables. En la Tabla 44 se muestran los diez coeficientes de mayor magnitud, que reflejan las variables con mayor impacto en el modelo.

Tabla 44: Coeficientes más significativos del modelo de regresión logística del caso complejo.

Variable	***Coeficiente***
DER_mass_vis	-1,3287
PRI_lep_pt	1,1810
DER_deltar_tau_lep	1,1071
PRI_jet_all_pt	-0,7972
DER_pt_ratio_lep_tau	-0,7661
PRI_tau_pt	0,7016
DER_mass_jet_jet	0,6713
DER_mass_transverse_met_lep	-0,6570
PRI_jet_leading_pt	0,5683
DER_lep_eta_centrality	0,4496

Por último, cabe destacar que el modelo ha tardado aproximadamente 5 minutos y medio en entrenarse, con un consumo energético total de 1,058 Wh. Estos resultados reflejan el bajo coste computacional y la eficiencia de estos modelos.

7.2.5 REDES NEURONALES MULTICAPA (MLP)

En esta sección se describe el ajuste y entrenamiento del modelo de red neuronal multicapa (MLP) para el conjunto de datos del Bosón de Higgs. Teniendo en cuenta que estos modelos son sensibles al overfitting si no se ajustan correctamente, se diseñó un proceso iterativo de búsqueda de hiperparámetros.

En primer lugar, se definió una rejilla de hiperparámetros moderada, con distintas configuraciones. El clasificador base tenía las siguientes características:

- `max_iter=200`
- `early_stopping=True`
- `random_state=42`

Tras una primera búsqueda, se identificó como mejor modelo la siguiente configuración:

- `'activation': 'tanh'`

- `'alpha': 0.001`
- `'hidden_layer_sizes': (128, 64)`
- `'learning_rate': 'constant'`
- `'solver': 'adam'`

Se obtuvo así una *accuracy* en entrenamiento del 83,72%. Aunque este resultado muestra un buen rendimiento del modelo, se planteó una rejilla más ambiciosa con arquitecturas más profundas, con el objetivo de mejorar esta métrica. Estas configuraciones obtuvieron rendimientos similares, pero no superaron al modelo anterior de forma significativa, por lo que se ha decidido optar por mantener la arquitectura más simple, por su estabilidad y eficiencia computacional. Este modelo alcanzó los valores que se muestran en la Tabla 45.

Tabla 45: Métricas más importantes del modelo MLP del caso complejo.

Métrica	***Resultado***
Accuracy	0,8390
Recall (clase 1)	0,7300
F1-Score (clase 1)	0,7553
AUC-ROC	0,9080

Por otro lado, el informe de clasificación completo se recoge en la Tabla 46. Se observa un comportamiento equilibrado entre clases, aunque con peores resultados para la clase minoritaria.

Tabla 46: Informe de clasificación del modelo MLP del caso complejo.

Clase	***Precisión***	***Recall***	***F1-score***	***Soporte***
0	0,87	0,89	0,88	33 065
1	0,78	0,73	0,76	16 935
Macro avg	0,82	0,81	0,82	50 000
Weighted avg	0,84	0,84	0,84	50 000

Para finalizar, el consumo energético del modelo fue de 2,554 Wh, con un tiempo de ajuste del modelo de 12 segundos y un tiempo de entrenamiento de aproximadamente 4 horas y 30 minutos. Esto confirma el alto coste computacional asociado a estos modelos.

7.2.6 KOLMOGOROV-ARNOLD NETWORK (KAN)

En esta última sección se detalla el proceso de ajuste, entrenamiento y evaluación de una red Kolmogorov-Arnold (KAN) sobre el conjunto de datos del bosón de Higgs, utilizando la versión 0.2.6 de la librería `pykan`. Esta versión no es la más reciente, pero se ha escogido para trabajar con el caso complejo, entre otras cosas, por su mayor estabilidad. Esto se explicará más detalladamente en el Capítulo 9.

El modelo KAN fue definido con una arquitectura compuesta por 24 nodos de entrada, tres capas ocultas de tamaño `[24, 16, 8]`, y una capa de salida de 2 nodos, correspondientes a las dos clases de este conjunto de datos. Se ha utilizado 24 variables de entrada porque se han eliminado las seis menos relevantes y así acelerar el entrenamiento, que se llevó a cabo en CPU, ya que el ordenador utilizado no tiene CUDA disponible. Se ha utilizado `CrossEntropyLoss` como función de pérdida, el optimizador Adam y 200 pasos de ajuste.

Tras este primer entrenamiento, se aplicó un proceso de poda como forma de simplificación, con umbrales bajos. En la Tabla 47 se muestran las métricas obtenidas con ambos modelos. A pesar de la ligera pérdida de *accuracy*, se observa una mejora en el *recall* para la clase minoritaria tras la poda, además de una reducción significativa en la complejidad del modelo. Por ello, se ha seleccionado el modelo podado como modelo final.

Tabla 47: Medidas obtenidas para el modelo KAN inicial y el modelo KAN podado para el caso complejo.

Métrica	***Modelo KAN***	***Modelo KAN Podado***
Accuracy	0,8136	0,8115
Recall clase 1	0,7593	**0,7630**
F1-score clase 1	0,7363	0,7350
AUC-ROC	0,8838	0,8832

El tiempo empleado para el ajuste de hiperparámetros fue de 12 segundos, mientras que el entrenamiento completo del modelo tardó aproximadamente 4 horas y media, con un consumo energético total de 45,321 Wh. Esto pone de manifiesto el alto consumo computacional y la lentitud de las KAN, especialmente cuando se trabaja con bases de datos grandes.

Finalmente, aunque una de las ventajas teóricas de las KAN es su interpretabilidad, esta propiedad se ve limitada en modelos con un número elevado de variables de entrada. Este aspecto se explorará más en profundidad en el Capítulo 8. El elevado número de variables de entrada y la profundidad de la red impiden llevar a cabo representaciones gráficas claras y comprensibles. Lo mismo ocurre con la obtención de la expresión simbólica, que en casos como este tardaría días en llevarse a cabo para acabar obteniendo una fórmula extremadamente larga e inservible.

7.3 COMPARACIÓN DE RESULTADOS

Para finalizar el Capítulo 7, se compararán los resultados obtenidos durante el entrenamiento de los seis modelos con el conjunto de datos del bosón de Higgs. En la Tabla 48 se muestran estos resultados. A continuación, se llevará a cabo una comparación detallada de cada métrica.

Tabla 48: Comparativa de resultados de los seis modelos evaluados sobre el caso complejo.

Modelo	***Accuracy***	***Sensibilidad***	***F1 Score***	***AUC***	***Tiempo de ajuste***	***Tiempo de entrenamiento***	***Uso energético***
SVM	83,38%	70,36%	74,37%	89,54%	1416,3 s.	9746,9 s.	26,118 Wh
Árbol de decisión	80,45%	73,94%	71,93%	87,27%	10,5 s.	253,8 s.	0,587 Wh
Random forest	83,88%	71,38%	75,22%	90,61%	4262 s.	4250,4 s.	9,428 Wh
Regresión logística	73,69%	76,00%	66,00%	81,5%	27 s.	331 s.	1,058 Wh
MLP	83,89%	73,00%	75,53%	90,80%	1159,8 s.	1147,9 s.	2,554 Wh
KAN	81,15%	76,30%	73,50%	88,32%	12,2 s.	16239,3 s.	45,321 Wh

7.3.1 PRECISIÓN O ACCURACY

En este caso, el modelo MLP ha obtenido la mayor precisión (83,89%), seguido muy de cerca por Random Forest (83,88%) y SVM (83,38%). Esta diferencia tan pequeña demuestra que estos modelos, que son más complejos, pueden alcanzar rendimientos similares si se ajustan de la manera adecuada.

A continuación, se encuentra el KAN podado (81,15%). Aunque sea ligeramente inferior, demuestra una buena capacidad de generalización, y supera en este aspecto a los modelos más simples.

Muy de cerca se encuentra el árbol de decisión (80,45%), y, por último, la regresión logística (73,69%). Este último modelo ha obtenido el peor resultado de esta métrica y con una diferencia notable. Este comportamiento concuerda con la desventaja de la regresión logística de capturar únicamente relaciones lineales entre variables, por lo que resulta insuficiente para un problema complejo como este.

7.3.2 SENSIBILIDAD O RECALL

La sensibilidad o *recall*, en este caso medida para la clase 1, que es la minoritaria, indica la capacidad del modelo para identificar correctamente los casos *signal*, que son los más relevantes en este caso, ya que el objetivo es distinguir estos eventos de otros de "fondo". Esta métrica adquiere especial importancia cuando los falsos negativos son críticos.

En este aspecto, el modelo KAN es el ganador (76,30%), seguido muy de cerca por la regresión logística (76,00%). Este resultado es muy interesante, ya que, ambos modelos, a pesar de ser completamente distintos en estructura y complejidad, parecen tener un muy buen rendimiento en la correcta identificación de la clase minoritaria, lo que puede ser de gran utilidad en aplicaciones reales.

Destaca también el modelo del árbol de decisión (73,94%), situado por encima del MLP (73,00%) y Random Forest (71,38%). Esto confirma la ventaja teórica de los modelos basados en árboles de capturar patrones útiles para distinguir clases minoritarias, aunque puedan ser más propensos al *overfitting*.

Por último, el SVM ha obtenido el peor resultado (70,36%). Esto es importante porque este modelo obtuvo la mejor *accuracy*, por lo que podría existir un compromiso entre ambas métricas. Esto encaja con una de las desventajas teóricas de estos modelos, que indica que podrían favorecer la clase mayoritaria cuando no se aplican técnicas de compensación en *datasets* desbalanceados.

7.3.3 F1-SCORE

El modelo con la mejor puntuación es MLP (75,53%), seguido muy de cerca por Random Forest (75,22%). Estos modelos obtuvieron buenos resultados tanto en *accuracy* como en *recall*, por lo que reflejan una gran solidez en problemas complejos como este.

El SVM ha obtenido también una muy buena puntuación (74,37%), aunque ligeramente inferior que las anteriores. Esta métrica refuerza su alta capacidad de generalización. El modelo KAN, apenas por debajo (73,50%), pone en evidencia que es un modelo novedoso pero competitivo, que logra mantener un equilibrio razonable.

Finalmente, los dos peores modelos han sido el del árbol de decisión (71,93%) y el de regresión logística (66%). Aunque el primero sigue encontrándose cerca de los modelos ganadores, el segundo tiene varios puntos de diferencia con ellos, lo que confirma de nuevo su dificultad para equilibrar la precisión y la sensibilidad.

7.3.4 ÁREA BAJO LA CURVA ROC (AUC)

En este aspecto, MLP y Random Forest son los ganadores (90,80% y 90,61% respectivamente), lo que indica una excelente capacidad discriminativa entre clases.

A continuación, se encuentra SVM (89,54%), con un resultado por encima del 85%, que suele ser el umbral de referencia para los buenos clasificadores. Muy cerca se encuentra el modelo KAN (88,32%), lo que indica que, aunque no alcanza el nivel de otros métodos más convencionales, puede ser de gran utilidad.

Se encuentran a continuación los árboles de decisión (87,27%), con un valor todavía por encima del 85%. Por último, el modelo de regresión logística tiene un resultado débil (81,50%), por debajo del 85%, un valor de nuevo coherente con su dificultad a la hora de capturar relaciones complejas en los datos.

7.3.5 EFICIENCIA COMPUTACIONAL

El modelo más eficiente ha sido el árbol de decisión, con un tiempo total de entrenamiento de apenas 4 minutos y 10 segundos y un consumo energético de 0,587 Wh. Su eficiencia lo convierte en una opción interesante si es una prioridad, a pesar de que su precisión no sea la más destacada.

Le sigue la regresión logística, con solo 5 minutos y medio de entrenamiento y un consumo también bajo (1,058 Wh). Este modelo puede ser una buena opción cuando la rapidez y la simplicidad son características deseadas en el modelo.

El modelo MLP tiene un consumo energético moderado (2,554 Wh), pero requiere 4 horas y 30 minutos de entrenamiento. Este resultado pone en evidencia una de sus principales desventajas: su lentitud.

El Random Forest ha mostrado un coste computacional muy alto, tanto en tiempo como en energía: 1 hora y 10 minutos para el ajuste y 1 hora y 10 minutos para el entrenamiento, con un total de 9,428 Wh consumidos. Se trata de un modelo eficaz que requiere recursos.

El SVM, situado a continuación en la lista, es uno de los modelos más costosos: 23 minutos de ajuste y 2 horas y 45 minutos de entrenamiento, con un consumo de 26,118 Wh. Se confirma así la principal desventaja señalada en anteriores secciones de este trabajo: su ineficiencia en conjuntos de datos grandes.

Por último, el modelo con mayor coste ha sido KAN, con 4 horas y media de entrenamiento y un consumo energético de 45,321 Wh, el más alto entre todos los modelos. Aunque su rendimiento es competitivo, estos costes deben considerarse cuidadosamente, especialmente en ausencia de aceleración por GPU, como es este caso.

7.3.6 INTERPRETABILIDAD

En la Tabla 49 se resume la interpretabilidad teórica y práctica de cada modelo.

Tabla 49: Comparativa de la interpretabilidad de los modelos del caso complejo.

Modelo	***Interpretabilidad teórica***	***Aplicabilidad práctica en este caso***
Regresión logística	Muy alta	Elevada. Coeficientes comprensibles y directos.
Árbol de decisión	Muy alta	Alta. Reglas claras, aunque la profundidad limita visualización.
Random Forest	Media	Media-baja. Sólo accesible a través de importancia de variables.
KAN	Alta	Baja. No aplicable en este caso por complejidad del modelo.
MLP	Baja	Baja. Arquitectura no interpretable directamente.
SVM	Baja	Baja. Kernels no lineales.

Como se puede observar, solo la regresión logística y los árboles de decisión ofrecen una gran interpretabilidad. Los modelos MLP y SVM funcionan como cajas negras. Por otro lado, las KAN, a pesar de su potencial interpretativo, aún presentan dificultades para explotar esta cualidad en casos complejos con múltiples variables.

Capítulo 8. ANÁLISIS DE RESULTADOS

Tras el análisis que se ha llevado a cabo en los capítulos 6 y 7 por separado, el presente capítulo ofrece una visión conjunta de los resultados obtenidos en ambos casos de uso. El objetivo es resumir las principales conclusiones observadas, comparar el rendimiento de las técnicas en distintos contextos y valorar sus casos de uso.

8.1 RENDIMIENTO GLOBAL DE LOS MODELOS

En términos generales, los modelos MLP, Random Forest y SVM han obtenido los mejores resultados, con un rendimiento consistente en *accuracy*, sensibilidad y AUC, es decir, precisión, capacidad de discriminación y equilibrio entre clases, respectivamente. Esto confirma la idoneidad de estas técnicas para tareas de clasificación con grados distintos de complejidad, ya que estos modelos se han adaptado correctamente a los datos a pesar de tener estructuras más complejas y requerir procesos de ajuste más cuidadosos.

El rendimiento del modelo KAN merece una mención especial, ya que a pesar de la novedad de la técnica y de su escasa documentación, su rendimiento ha sido muy competitivo, especialmente en el caso complejo, donde se trabajó con una versión de la librería `pykan` más antigua y estable y superó en varias métricas a modelos clásicos. Sin embargo, sus resultados en el caso sencillo fueron más modestos, lo que indica la necesidad de mejorar su estabilidad y optimización en conjuntos de datos más pequeños y sencillos cuando se trabajó con la última versión disponible. La diferencia entre ambas versiones se explicará más detalladamente en el Capítulo 9.

En la Tabla 50 se muestra un resumen con las métricas más representativas para los seis modelos en ambos casos de estudio, lo que permite visualizar de forma directa su comportamiento global.

Tabla 50: Comparación de resultados globales de los seis modelos en ambos casos de estudio.

Modelo	*Accuracy (sencillo)*	*F1 (sencillo)*	*AUC (sencillo)*	*Accuracy (complejo)*	*F1 (complejo)*	*AUC (complejo)*
SVM	86,03%	84,89%	88,54%	83,38%	74,37%	89,54%
Árbol de Decisión	83,02%	53,14%	80,82%	80,45%	71,93%	87,27%
Random Forest	85,47%	84,00%	88,36%	83,88%	75,22%	90,61%
Regresión logística	85,02%	84,00%	87,29%	73,69%	66,00%	81,50%
MLP	84,00%	84,00%	86,75%	83,89%	75,53%	90,80%
KAN	81,23%	81,00%	82,62%	81,15%	73,50%	88,32%

8.2 EFICIENCIA COMPUTACIONAL Y APLICABILIDAD

Las diferencias en consumo energético y tiempos de ajuste y entrenamiento permiten obtener conclusiones relevantes sobre la aplicabilidad de cada técnica.

SVM y KAN han sido los modelos con mayor coste computacional, tanto en energía como en tiempo, especialmente en el caso complejo. Esta es una desventaja que puede limitar la aplicabilidad de estos modelos cuando los recursos disponibles son limitados o el tiempo de respuesta es un factor crítico.

Por el contrario, los modelos de regresión logística y árboles de decisión han destacado por su eficiencia energética y rapidez. Estas técnicas constituyen una buena alternativa cuando se necesita una solución rápida e interpretable.

8.3 CONSIDERACIONES SOBRE INTERPRETABILIDAD

La interpretabilidad sigue siendo una de las principales limitaciones de los modelos más complejos. Durante el desarrollo de este trabajo, solo la regresión logística y los árboles de decisión han ofrecido interpretaciones claras y accesibles. Los modelos KAN, a pesar de su potencial teórico en interpretabilidad mediante representaciones gráficas y simbólicas del modelo, no han permitido explotar esta ventaja en profundidad debido a la complejidad del caso de uso y al alto número de variables de entrada de ambos modelos. Por otro lado, los modelos MLP y SVM han actuado como cajas negras.

8.4 REPOSITORIO DE TRABAJO

Con el fin de fomentar la transparencia, reutilización y reproducibilidad de los resultados obtenidos, se ha creado un repositorio público en GitHub que incluye todos los notebooks de entrenamiento, ajuste, evaluación y generación de visualizaciones, tanto para el caso sencillo como para el complejo.

Dentro del repositorio existen varias carpetas. A continuación, se muestra un resumen del contenido de cada una de ellas:

- `1ST DATASET`: En esta carpeta se encuentran, para el caso sencillo, las visualizaciones del preprocesado de los datos y seis carpetas para cada modelo entrenado que contienen el notebook con el entrenamiento completo y un archivo `emissions.csv` con las emisiones calculadas con `codecarbon`.
- `2ND DATASET`: En esta carpeta se encuentran, para el caso complejo, las visualizaciones del preprocesado de los datos y seis carpetas para cada modelo entrenado que contienen el notebook con el entrenamiento completo y un archivo `emissions.csv` con las emisiones calculadas con `codecarbon`.
- `TUTORIALES`: Aquí se encuentran los notebooks con los tutoriales seguidos para cada técnica.
- `EXAMPLES KAN`: Aquí se encuentra un ejemplo de un modelo KAN que se utilizó como referencia para comparar distintas versiones de `pyKAN`.

Capítulo 9. CONCLUSIONES Y TRABAJOS FUTUROS

Este capítulo final recoge las conclusiones más importantes obtenidas tras el análisis y comparación de los modelos de clasificación evaluados a lo largo de este trabajo. Se dedica una sección específica a las KAN, por su novedad y por ser una de las partes más importantes de este proyecto. Además, se plantea una comparativa entre las versiones utilizadas de la librería `pykan`. Finalmente, se proponen posibles líneas de trabajo futuro relacionadas con la mejora y aplicación de las KAN en otros contextos.

9.1 KAN FRENTE AL RESTO DE MODELOS

Los resultados obtenidos durante el entrenamiento y evaluación de los seis modelos de clasificación han permitido obtener algunas conclusiones importantes.

Por un lado, los resultados han demostrado que modelos como MLP, Random Forest y SVM ofrecen un rendimiento muy competitivo en tareas complejas de clasificación, especialmente si se dispone de los recursos y el tiempo necesarios para su ajuste y entrenamiento. Por otro lado, los modelos de regresión logística y árboles de decisión han demostrado ser rápidos, eficientes e interpretables, pero con un rendimiento inferior en el caso complejo.

En este contexto, los modelos KAN han mostrado un comportamiento especialmente interesante. En el caso complejo se han obtenido resultados muy próximos a los de los modelos convencionales, incluso superándolos en ciertas métricas clave. En el caso sencillo, sus resultados han sido más modestos, pero esto puede explicarse con la diferencia de versiones de `pykan` utilizadas durante el entrenamiento del caso sencillo y el caso complejo. Además, se ha observado que mientras todas las técnicas empeoran su *accuracy* en el caso complejo, el modelo KAN se mantiene constante.

Se refuerza así la idea de que KAN es una técnica con mucho potencial, especialmente cuando se busca explorar nuevas formas de representar relaciones complejas entre variables. Sin embargo, se han puesto de manifiesto algunas limitaciones actuales. Entre ellas, se encuentran:

- Elevado coste computacional.
- Alta sensibilidad a la configuración de hiperparámetros.
- Interpretabilidad práctica limitada en casos con muchas variables.
- La afirmación sobre una mayor precisión respecto a MLP no es generalizada, sino que depende en gran medida del contexto y del tipo de datos utilizados, como se explica en algunos estudios.

9.2 COMPARATIVA ENTRE VERSIONES DE PYKAN

Durante el desarrollo de este trabajo, fue necesario emplear dos versiones distintas de la librería `pyKAN`, debido a las diferencias observadas en su estabilidad, utilización y comportamiento. Concretamente, se utilizó la última versión (0.2.8) en el análisis del caso sencillo y la versión 0.2.6 en el caso complejo.

La última versión incluye mejoras funcionales y mayor compatibilidad con nuevas versiones de *PyTorch*. Sin embargo, durante su uso en el caso sencillo se han detectado algunos problemas:

- Errores en la visualización de animaciones y funciones de activación durante el entrenamiento.
- Fallos en el almacenamiento y acceso a variables internas del modelo, como `train_loss`, que impedían evaluar correctamente el proceso de ajuste.
- Pérdida de reproducibilidad entre ejecuciones consecutivas, incluso usando la misma semilla.
- Problemas de convergencia con determinadas arquitecturas, especialmente en redes más profundas.
- Limitaciones para continuar el ajuste tras la ejecución de los primeros pasos (`stepwise tuning`), que generaban interrupciones del entrenamiento.

A pesar de que se intentaron llevar a cabo soluciones como modificar directamente el archivo MultKAN.py o limitar el número de pasos durante el entrenamiento, la experiencia con esta versión fue inestable y dificultó la obtención de resultados sólidos en el caso sencillo.

Por este motivo, se optó por usar una versión anterior que había sido empleada previamente en tutoriales oficiales y publicaciones previas sobre KAN. Esta versión demostró ser mucho más estable y permitió llevar a cabo entrenamientos más largos con configuraciones más exigentes sin errores inesperados, aunque con funcionalidades limitadas. Para garantizar su correcto funcionamiento, se creó un entorno virtual específico que incluía la versión concreta de `pykan` y la versión compatible de otras librerías como `numpy` y `matplotlib`. Esta solución fue clave para evitar conflictos con otras librerías del sistema y asegurar la consistencia de los resultados.

De esta forma se consiguió que el modelo KAN alcanzase sus mejores resultados en el caso complejo: obtuvo una *accuracy* del 81,15%, un F1-Score del 73,50% y un AUC del 88,32%. Estos valores superaron los logrados en el caso sencillo con la versión 0.2.8, y se situaron por encima de modelos clásicos. Por tanto, se puede afirmar que el mejor rendimiento de KAN en el caso complejo se debe en gran medida a la mayor estabilidad de la versión 0.2.6 y a su ejecución en un entorno virtual aislado, lo que evitó los fallos observados en versiones más recientes.

9.3 TRABAJOS FUTUROS

Teniendo en cuenta los resultados obtenidos y las limitaciones observadas, existen diversas líneas de trabajo que pueden explorarse en el futuro:

- **Aplicación de KAN a otros conjuntos de datos**: Sería interesante evaluar el comportamiento de estas redes en otros problemas de clasificación, especialmente en entornos donde los datos presentan relaciones complejas o donde existe un número elevado de variables.
- **Exploración de contextos donde la interpretabilidad sea clave**: Dado el potencial teórico de las KAN para ofrecer representaciones gráficas mediante B-splines, se propone como línea futura su aplicación en ámbitos como la salud, donde se requiere transparencia e interpretabilidad. Sin embargo, será necesario mejorar las herramientas para poder llevar a cabo dicha práctica.

- **Estabilización de la versión 0.2.8 de** `pykan`: Para aprovechar las funcionalidades más recientes, como la visualización animada del entrenamiento o las mejoras en optimización, será fundamental colaborar o contribuir al desarrollo de versiones más estables. Esto implica depurar errores, mejorar la documentación y definir buenas prácticas de uso, ya que actualmente no existe demasiada información al respecto y se requiere de una gran labor de investigación para entender el funcionamiento de esta librería. El Capítulo 5. este trabajo resume toda la información necesaria para poder utilizar esta librería.
- **Optimización automática de hiperparámetros**: Otra mejora posible sería integrar técnicas de ajuste automático (como *Optuna* o *Hyperopt*) para reducir el tiempo necesario de experimentación y ajustar mejor la arquitectura y parámetros de las redes KAN a cada problema.
- **Paralelización y entrenamiento con GPU**: Sería deseable que futuras versiones de `pyKAN` ofrecieran una mayor eficiencia también en CPU, ya que actualmente los tiempos de entrenamiento son muy elevados si no se dispone de aceleración por GPU mediante CUDA.

Capítulo 10. BIBLIOGRAFÍA

[1] A. Géron, *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow.* (2nd ed.) 2019.

[2] E. Horvitz and D. Mulligan, "Data, privacy, and the greater good," *Science,* vol. 349, *(6245),* pp. 253, 2025. . DOI: 10.1126/science.aac4520.

[3] P. Domingos, "A few useful things to know about machine learning," *Commun. ACM,* pp. 78, 2012. . DOI: 10.1145/2347736.2347755.

[4] F. Tajdini and S. Bayat, "Machine Learning Algorithms - A Review," 2022. .

[5] S. Maldonado, R. Weber and Resumen, "Modelos De Selección De Atributos Para Support Vector Machines." , 2012.

[6] E. J. Carmona Suárez, "Tutorial sobre Máquinas de Vectores Soporte (SVM)," 2012. .

[7] Anonymous (Ago 16). *Máquinas de vectores de soporte: Clasificación y teoría*. Available: https://aprendeia.com/2019/08/16/maquinas-vectores-de-soporte-clasificacion-teoria/.

[8] Fernández-Casal R., Costa J. and Oviedo M, *Métodos Predictivos De Aprendizaje Estadístico.* 2024Available: https://doi.org/10.17979/spudc.9788497498937. DOI: 10.17979/spudc.9788497498937.

[9] Girgin S. (Jul 30). *Day-12: Kernel SVM (Non-Linear SVM)*. Available: https://medium.com/@sametgirgin/day-12-kernel-svm-non-linear-svm-5fdefe77836c.

[10] Dhiraj K, "Top 5 Advantages and Disadvantages of Decision Tree Algorithm," *Medium,* 2019. Available: https://dhirajkumarblog.medium.com/top-5-advantages-and-disadvantages-of-decision-tree-algorithm-428ebd199d9a.

[11] Attard M. *8 Key Advantages and Disadvantages of Decision Trees*. Available: https://insidelearningmachines.com/advantages_and_disadvantages_of_decision_trees/.

[12] A. Chaudhary, S. Kolhe and R. Kamal, "An improved random forest classifier for multi-class classification," *Information Processing in Agriculture,* vol. 3, *(4),* pp. 215, 2016. . DOI: 10.1016/j.inpa.2016.08.002.

[13] ravinderkamatw. (Feb 15). *What are the Advantages and Disadvantages of Random Forest?*. Available: https://www.geeksforgeeks.org/what-are-the-advantages-and-disadvantages-of-random-forest/.

[14] Koehrsen W. (Dic 27). *Random Forest Simple Explanation*. Available: https://williamkoehrsen.medium.com/random-forest-simple-explanation-377895a60d2d.

[15] H. Chitarroni, "La regresión logística," 2002. .

[16] Duggal N. (May 4). *Introduction to Machine Learning - A Step by Step Guide*. Available: https://www.simplilearn.com/tutorials/machine-learning-tutorial/introduction-to-machine-learning#machine_learning_basics__types_of_machine_learning.

[17] S. Banerjee. (Abr 6). *Exploring the Power and Limitations of Multi-Layer Perceptron (MLP) in Machine Learning*. Available: https://shekhar-banerjee96.medium.com/exploring-the-power-and-limitations-of-multi-layer-perceptron-mlp-in-machine-learning-d97a3f84f9f4.

[18] B. Akkaya and N. Çolakoğlu, *Comparison of Multi-Class Classification Algorithms on Early Diagnosis of Heart Diseases.* 2019.

[19] J. C. Cuevas Tello, "Apuntes de redes neuronales artificiales handouts for artificial neural networks," 2017Available: https://arxiv.org/pdf/1806.05298. DOI: 10.48550/arXiv.1806.05298.

[20] Z. Liu *et al*, "KAN: Kolmogorov-Arnold Networks," 2025. .

[21] D. Bethell. (May 13). *Demystifying Kolmogorov-Arnold Networks: A Beginner-Friendly Guide with Code*. Available: https://daniel-bethell.co.uk/posts/kan/.

[22] R. Zimbres. (Jul 11). *Kolmogorov-Arnold Networks: a Critique*. Available: https://medium.com/@rubenszimbres/kolmogorov-arnold-networks-a-critique-2b37fea2112e.

[23] J. Gemini. (May 30). *Redes de Kolmogorov-Arnold (KAN) explicadas*. Available: https://medium.com/@jaspergemini/redes-de-kolmogorov-arnold-kan-explicadas-c865b10e7977.

[24] KindXiaoming. *pykan: Kolmogorov–Arnold Networks in PyTorch*. Available: https://github.com/KindXiaoming/pykan.

[25] N. Agarwal and D. Yadav, "A Comprehensive Analysis of Classical Machine Learning and Modern Deep Learning Methodologies," *International Journal of Engineering Research,* 2024. Available:

https://www.researchgate.net/publication/381193962_A_Comprehensive_Analysis_of_Classical_Machine_Learning_and_Modern_Deep_Learning_Methodologies. DOI: 10.17577/IJERTV13IS050275.

[26] P. V. Yuanhong Xu Aleksander Madry, "Kolmogorov–Arnold Expressivity in Neural Networks," 2024. Available: https://arxiv.org/html/2406.02496v1.

[27] Jitendra. (May 27). *Introduction to Supervised Deep Learning Algorithms!*. Available: https://www.analyticsvidhya.com/blog/2021/05/introduction-to-supervised-deep-learning-algorithms/.

[28] A. González-Muñiz, "Aplicación De Técnicas De Aprendizaje Profundo (Deep Learning) Al Análisis Y Mejora De La Eficiencia En Sistemas De Ingeniería." , 2023.

[29] P. Li, Y. Pei and J. Li, "A comprehensive survey on design and application of autoencoder in deep learning," *Applied Soft Computing,* vol. 138, pp. 110176, 2023. Available: https://www.sciencedirect.com/science/article/pii/S1568494623001941. DOI: 10.1016/j.asoc.2023.110176.

[30] Anonymous (Jun 26). *Is deep learning supervised or unsupervised?*. Available: https://www.aiplusinfo.com/blog/is-deep-learning-supervised-or-unsupervised/.

[31] Anonymous . *¿Qué es Python?*. Available: https://aws.amazon.com/es/what-is/python/.

[32] Anonymous . *MATLAB*. Available: https://la.mathworks.com/products/matlab.html.

[33] S. Worsley. (Jul 1). *¿Qué es R? Introducción a la potencia del cálculo estadístico*. Available: https://www.datacamp.com/es/blog/all-about-r.

[34] Anonymous . *User Guide - scikit-learn 1.7.0 documentation*. Available: https://scikit-learn.org/stable/user_guide.html.

[35] C. Chang and C. Lin, "LIBSVM : a library for support vector machines," *ACM Transactions on Intelligent Systems and Technology,* vol. 2, *(3),* pp. 27:1–27:27, 2011, 2011. .

[36] Anonymous . *XGBoost Documentation - xgboost 3.0.2 documentation*. Available: https://xgboost.readthedocs.io/en/stable/.

[37] Anonymous . *CatBoost: Una herramienta esencial para el Machine Learning*. Available: https://datascientest.com/es/catboost-que-es.

[38] Anonymous "Welcome to LightGBM's documentation! - LightGBM 4.6.0," Available: https://lightgbm.readthedocs.io/en/stable/.

[39] Anonymous . *statsmodels 0.14.4*. Available: https://www.statsmodels.org/stable/index.html.

[40] Ángel. (Jun 16). *¿Qué es TensorFlow y para qué sirve?*. Available: https://www.incentro.com/es-ES/blog/que-es-tensorflow.

[41] D. Bergmann and C. Stryker. (Nov 23). *¿Qué es PyTorch?*. Available: https://www.ibm.com/es-es/topics/pytorch.

[42] R. Ali, S. Lee and T. C. Chung, "Accurate multi-criteria decision making methodology for recommending machine learning algorithm," *Expert Syst. Appl.,* vol. 71, pp. 257–278, 2017. Available: https://www.sciencedirect.com/science/article/pii/S0957417416306698. DOI: 10.1016/j.eswa.2016.11.034.

[43] B. Bahr, "14.2 Classification Metrics: Accuracy, Precision, Recall, F1-score, and ROC-AUC – Statistical Prediction," *Fiveable,* 2024. Available: https://library.fiveable.me/modern-statistical-prediction-and-machine-learning/unit-14/classification-metrics-accuracy-precision-recall-f1-score-roc-auc/study-guide/wwl93Vnj2LTRDsTL.

[44] D. Didmanidze. (Nov 8). *Redes de Kolmogorov-Arnold (KAN): Guía de aplicación*. Available: https://www.datacamp.com/es/tutorial/kolmogorov-arnold-networks.

[45] H. Sheikh. (May 7). *Understanding Kolmogorov–Arnold Networks (KAN)*. Available: https://medium.com/data-science/kolmogorov-arnold-networks-kan-e317b1b4d075.

[46] Dr. Ashish Bamania. (May 8). *Kolmogorov-Arnold Networks (KANs) Might Change AI As We Know It, Forever*. Available: https://intoai.pub/p/kolmogorov-arnold-networks-kans-might.

[47] Z. Liu. *Hello, KAN!*. Available: https://kindxiaoming.github.io/pykan/intro.html.

[48] Z. Liu. *API Demos*. Available: https://kindxiaoming.github.io/pykan/demos.html.

[49] U. Kumar. (Mar 11). *Numerical optimization based on the L-BFGS method*. Available: https://medium.com/data-science/numerical-optimization-based-on-the-l-bfgs-method-f6582135b0ca.

[50] Diederik P. Kingma and Jimmy Ba, "Adam: A Method for Stochastic Optimization," 2017. .

[51] PyTorch. *torch.nn — Loss functions*. Available: https://pytorch.org/docs/stable/nn.html#loss-functions.

[52] Anonymous . *Rain in Australia*. Available: https://www.kaggle.com/datasets/jsphyg/weather-dataset-rattle-package/data.

[53] Anonymous . *Kaggle*. Available: https://www.kaggle.com/.

[54] Interactive Chaos. *Support Vector Machines*. Available: https://interactivechaos.com/es/manual/tutorial-de-machine-learning/support-vector-machines-0.

[55] Scikit-Learn. *LogisticRegression*. Available: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html.

[56] C. Adam-Bourdarios *et al*, "Learning to discover: the Higgs boson machine learning challenge," 2014. .

[57] Joycenv. *Higgs Boson Machine Learning Challenge*. Available: https://www.kaggle.com/competitions/higgs-boson/data.

ANEXO I: ALINEACIÓN DEL PROYECTO CON LOS ODS

Este trabajo se alinea con los Objetivos de Desarrollo Sostenible (ODS) establecidos por la Agenda 2030 de las Naciones Unidas, en particular con el ODS 13: Acción por el clima.

A lo largo del proyecto, se ha prestado especial atención al consumo energético de los modelos de clasificación evaluados mediante la librería de Python codecarbon. Se han medido y comparado los tiempos de ajuste y entrenamiento y el gasto energético (en Wh) de cada modelo, tanto en un caso de uso sencillo como en uno complejo.

Esta dimensión energética no suele estar presente en estudios puramente técnicos, pero resulta fundamental en un contexto global donde la demanda de recursos computacionales crece de forma exponencial. Con esta perspectiva, el trabajo fomenta una conciencia crítica sobre la sostenibilidad del uso de la inteligencia artificial, promoviendo el desarrollo y elección de modelos más eficientes, especialmente en entornos con recursos limitados.

En particular, la comparación entre modelos ha permitido identificar soluciones con un buen equilibrio entre rendimiento y consumo energético, así como poner de manifiesto los costes asociados a técnicas más novedosas o exigentes computacionalmente, como las KAN.

En resumen, el proyecto contribuye al ODS 13 al integrar el análisis de eficiencia energética en el desarrollo de soluciones basadas en Machine Learning, promoviendo una inteligencia artificial más sostenible, responsable y respetuosa con el medio ambiente.