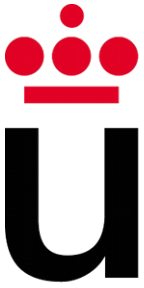


Universidad
Rey Juan Carlos

ESCUELA DE INGENIERÍA DE FUENLABRADA

GRADO EN INGENIERÍA EN TECNOLOGÍAS DE LA TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

Comparativa del consumo energético de algoritmos de aprendizaje automático

Autora : Laura González Fernández
Tutor : José Felipe Ortega Soto

Curso Académico 2023/2024

Trabajo Fin de Grado

Comparativa del Consumo Energético de Algoritmos de Aprendizaje Automático

Autora : Laura González Fernández
Tutor : José Felipe Ortega Soto

La defensa del presente Proyecto Fin de Grado se realizó el día de de 2024, siendo calificada por el siguiente tribunal:

Presidente:

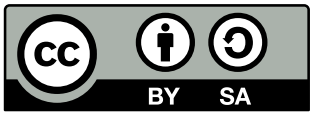
Secretario:

Vocal:

y habiendo obtenido la siguiente calificación:

Calificación:

Fuenlabrada, a de de 2024



Agradecimientos

Quisiera aprovechar esta oportunidad para agradecer a las muchas personas que han contribuido de manera significativa a la realización de esta memoria de grado.

En primer lugar, agradezco a mi familia por su apoyo constante y paciencia a lo largo de mi trayectoria académica. Siempre confiaron en mí, incluso en los momentos de duda. Gracias también por su ayuda durante este proyecto, ya sea con la recopilación de datos o revisando mis resultados. A mi hermana, especialmente, gracias por soportarme estos últimos años mientras avanzaba en este proyecto.

Extiendo mi agradecimiento a mis supervisores por su orientación, experiencia y valiosos comentarios durante el desarrollo de la memoria. Su dedicación y disposición para compartir su conocimiento han sido esenciales.

A mis amigos y compañeros, gracias por estar a mi lado durante este proceso. Su apoyo y las frecuentes preguntas sobre el estado de mi interminable proyecto me motivaron a seguir adelante.

Esta memoria es el resultado de años de esfuerzo y dedicación, y agradezco a todos los que me apoyaron en este camino.

Resumen

El aprendizaje automático se ha desarrollado a pasos de gigante durante los últimos años, permitiendo a los sistemas informáticos abstraer relaciones complejas entre datos y hacer predicciones precisas. Sin embargo, estos avances también han incrementado el consumo energético debido al procesamiento de enormes cantidades de datos, lo que genera una huella ambiental significativa. En este contexto, se vuelve imperativo equilibrar la precisión y eficacia de los modelos de aprendizaje automático con su impacto energético.

Este proyecto tiene como objetivo proporcionar una herramienta que permita comparar modelos de aprendizaje automático en términos de precisión y consumo energético. Para ello, se ha desarrollado una aplicación utilizando scikit-learn, Python y CodeCarbon, junto con conjuntos de datos públicos. La arquitectura de la aplicación está diseñada para medir las emisiones de carbono producidas durante el entrenamiento de varios modelos, abarcando todo el proceso de aprendizaje desde la preparación de datos y la validación cruzada de los modelos hasta la recopilación de consumo energético usando CodeCarbon.

Para ofrecer una muestra del funcionamiento de la aplicación se llevaron a cabo dos experimentos principales. El primero comparó las emisiones y la precisión de varios modelos en diferentes conjuntos de datos de creciente complejidad. El modelo de vecinos más cercanos mostró un consumo energético bajo y precisión moderada, mientras que el modelo de bosque aleatorio ofreció alta precisión con consumo moderado. Las redes neuronales, aunque precisas, presentaron un consumo significativamente mayor. El segundo experimento evaluó las emisiones en máquinas con distintas configuraciones de recursos, usando máquinas virtuales en Microsoft Azure. Los resultados indicaron que el uso de paralelismo y mayor cantidad de procesadores aumenta el consumo energético por unidad de tiempo, pero reduce el tiempo total de entrenamiento, compensando el incremento en consumo.

En conclusión, este proyecto no solo proporciona una herramienta útil para la evaluación de modelos de aprendizaje automático desde una perspectiva de sostenibilidad, sino que también destaca la necesidad de tener en cuenta el impacto ambiental en el desarrollo de soluciones tecnológicas avanzadas. Futuros trabajos podrán expandir sobre esta base, explorando nuevos modelos y técnicas de optimización para reducir aún más el consumo energético sin comprometer la precisión.

Summary

Machine learning has experienced great progress in recent years, allowing computer systems to abstract complex relationships between data and make accurate predictions. However, these advances have also increased energy consumption due to the cost of processing vast amounts of data, resulting in a significant environmental footprint. In this context, it is imperative to balance the precision and efficiency of machine learning models with their energy impact.

This project aims to provide a tool to compare machine learning models in terms of their precision and energy consumption. To achieve this, an application was developed using scikit-learn, Python, and CodeCarbon, along with publicly available datasets. The application's architecture is designed to measure carbon emissions produced during the training of various models, covering the entire learning process from data preparation and model cross-validation to energy consumption collection using CodeCarbon.

To demonstrate the application's functionality, two main experiments were conducted. The first compared the emissions and precision of various models on different datasets of increasing complexity. The k-nearest neighbors model showed low energy consumption and moderate precision, while the random forest model offered high precision with moderate consumption. Neural networks, although precise, had significantly higher consumption. The second experiment evaluated emissions on machines with different resource configurations using virtual machines on Microsoft Azure. The results indicated that using parallelism and more processors increases energy consumption per unit of time but reduces total training time, compensating for the increased consumption.

In conclusion, this project provides a useful tool for evaluating machine learning models from a sustainability perspective, highlighting the importance of considering environmental impact in developing advanced technological solutions. Future work can build on this foundation by exploring new models and optimization techniques to further reduce energy consumption without compromising precision.

Índice general

Índice de figuras

Capítulo 1

Introducción

El aprendizaje automático (*machine learning* o ML en inglés) es una rama de la Inteligencia Artificial y las Ciencias de la Computación que se centra en el uso de datos y algoritmos para imitar la forma en la que los humanos aprenden con el objetivo de aumentar gradualmente su precisión. Es un componente fundamental del campo de la Ciencia de Datos, cuya importancia ha experimentado un gran crecimiento, especialmente en las últimas dos décadas [1]. El aprendizaje automático hace uso de métodos estadísticos y computacionales para entrenar algoritmos que realizan, entre otras tareas, clasificaciones o predicciones y que permiten descubrir piezas clave de información dentro de proyectos de procesamiento de datos. Posteriormente, esta información puede utilizarse en la toma de decisiones dentro de distintas aplicaciones y negocios, con una gran capacidad para impactar en el crecimiento de los mismos.

1.1 Aprendizaje automático y consumo energético

Gracias al desarrollo de nuevas tecnologías computacionales, el aprendizaje automático que se utiliza hoy en día es muy diferente de como era en el pasado. El modelo actual surgió del reconocimiento de patrones y de la teoría de que los ordenadores pueden aprender a resolver tareas específicas, cuando los investigadores interesados en la Inteligencia Artificial se empezaron a plantear si los ordenadores serían capaces de aprender a partir de datos. De aquí surge la importancia del aspecto iterativo de este aprendizaje, que permite que el sistema se adapte de forma independiente cada vez que nuevos datos son incorporados y sea capaz de aprender de cada computación previa para producir decisiones y resultados que sean confiables y repetibles.

De esta forma, aunque una gran parte de los algoritmos utilizados en el aprendizaje automático son conocidos desde hace décadas, la habilidad de aplicar complejos cálculos

matemáticos a grandes cantidades de datos una y otra vez, cada vez más rápidamente, es un desarrollo muy reciente conseguido gracias a los avances en componentes informáticos y la disminución de costes de grandes sistemas computacionales con enormes capacidades de memoria y procesamiento. En concreto, esto sucede en el campo del aprendizaje profundo (*deep learning*), donde los modelos han crecido en cálculos para llegar a alcanzar típicamente el orden de los GigaFlops y en requisitos de memoria que se encuentran comúnmente en el orden de los millones de parámetros.

Sin embargo, este gran poder de procesamiento también trae consigo un gran gasto energético. El consumo de energía en la arquitectura de computadores ha sido foco de atención de investigadores interesados en obtener procesadores energéticamente eficientes de última generación durante décadas. Por otro lado, los investigadores interesados en el aprendizaje automático se han centrado principalmente en la producción de modelos cada vez más profundos y precisos, sin poner ningún límite en términos computacionales más allá de la disponibilidad de procesadores capaces de llevar a cabo estos cálculos[2].

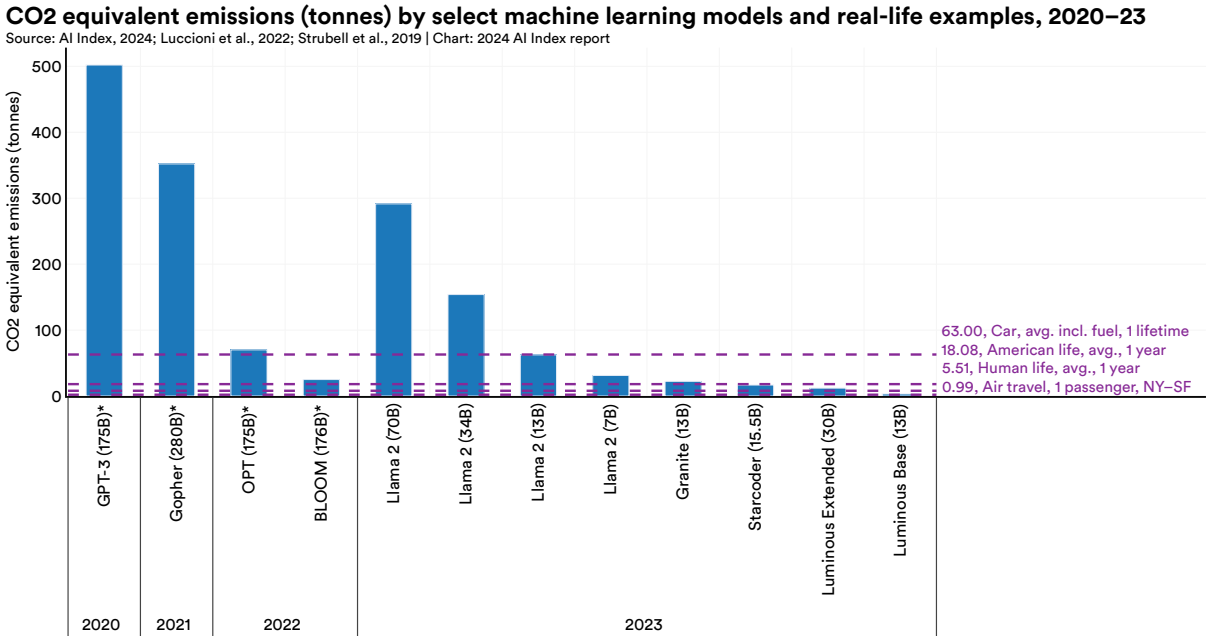


Figura 1.1: Emisiones de CO_2 producidas en el entrenamiento de varios modelos de gran escala en los últimos años.
Fuente: *The AI Index 2024 Annual Report* [3].

Adicionalmente, con la reciente popularización de los grandes modelos de aprendizaje, especialmente de los modelos de lenguaje de gran escala (*Large Language Models*, LLMs), pero también de la visión por computador y el aprendizaje en la nube, la investigación sobre el impacto energético se hace aún más necesaria. Los grandes modelos de lenguaje, por ejemplo, requieren enormes cantidades de datos y recursos computacionales para su entrenamiento, lo que se traduce en un consumo energético considerable [4]. GPT-3

[5], uno de los modelos más conocidos, cuenta con 175 mil millones de parámetros y requiere una cantidad masiva de energía para su entrenamiento. Del mismo modo, las aplicaciones de visión por computador, especialmente aquellas que involucran redes convolucionales profundas (CNN), también son intensivas en energía [6]. La figura 1.1 muestra las emisiones de CO_2 estimadas producidas durante el entrenamiento de varios modelos recientes, comparadas con referencias comunes como el consumo medio por una persona durante toda su vida.

El aprendizaje en la nube agrava estos problemas al trasladar el consumo energético a los centros de datos, que pueden tener diferentes niveles de eficiencia y fuentes de energía. Este tipo de aprendizaje, aunque ofrece flexibilidad y escalabilidad, puede tener implicaciones negativas para el consumo energético. Los centros de datos que alimentan estos servicios requieren grandes cantidades de electricidad, y su impacto depende en gran medida de la fuente de energía utilizada. Algunos centros de datos están migrando hacia fuentes de energía renovable para mitigar este impacto, pero la transición no es uniforme en todas las regiones.

1.2 Desarrollo de una aplicación comparativa: *MLCost*

En este contexto temporal de modelos de aprendizaje que cada vez consumen más energía, se hace más necesario contar con una medida de la energía consumida y las emisiones producidas durante el proceso de entrenamiento. Esta información es vital para poder realizar una decisión informada en la elección de los modelos a utilizar en una aplicación concreta de aprendizaje automático. Este proyecto propone el desarrollo de una aplicación, con el nombre MLCost, que permita a los investigadores obtener esta información de forma sencilla mediante el uso de herramientas existentes de medición. Para ello, la aplicación hace uso de librerías como CodeCarbon, que permite abstraer el método de calculo de emisiones, y scikit-learn, que permite entrenar modelos sin considerar gran parte del desarrollo de bajo nivel asociado con la implementación de los algoritmos utilizados y la preparación de conjuntos de datos para el entrenamiento. De este modo, la aplicación posibilita la medición del consumo y las emisiones producidas por distintos modelos a estudiar en cualquier conjunto de datos que se quiera utilizar, así como la comparación con gráficos sencillos de los resultados obtenidos.

1.3 Objetivos del proyecto

1.3.1 Objetivo general

Este Trabajo de Fin de Grado tiene como objetivo crear una herramienta que permita la comparación sistemática del consumo energético y el impacto de la huella de carbono en los modelos más representativos de técnicas de clasificación de aprendizaje automático supervisado.

1.3.2 Objetivos específicos

Para esta finalidad se han tenido en cuenta los siguientes objetivos específicos:

- Estudiar las herramientas disponibles para aprendizaje automático.
- Estudiar el uso de herramientas de medida del consumo energético.
- Comparar el consumo energético en los algoritmos de clasificación más importantes.
- Analizar la relación entre la precisión de los modelos y su consumo energético en conjuntos de datos de distintos tamaños y características.

1.4 Estructura de la memoria

La memoria de este proyecto se estructura en cinco capítulos, con una introducción, seguida de tres capítulos que reflejan las tres fases principales del proyecto: investigación, desarrollo y validación, para finalizar con las conclusiones y trabajos futuros.

El primer capítulo introduce el contexto en el cual se ha desarrollado la aplicación, proporcionando información de fondo y destacando los objetivos principales del proyecto. Este capítulo establece el escenario para los capítulos siguientes, ofreciendo una visión general del alcance y propósito del proyecto.

El segundo capítulo revisa la literatura relevante y el estado del arte en cuanto a métodos de medición de emisiones y estudios sobre el consumo energético del aprendizaje automático. Además, proporciona una introducción general al aprendizaje automático, describe las librerías existentes utilizadas en el desarrollo del proyecto y detalla los modelos y conjuntos de datos utilizados durante las pruebas de la aplicación.

El tercer capítulo describe la arquitectura de la aplicación desarrollada, incluyendo el proceso de preparación de los datos y los métodos de evaluación de las predicciones de los modelos. Este capítulo se centra en los aspectos técnicos del diseño y la implementación, explicando las decisiones de diseño y cómo se implementaron las diferentes funcionalidades.

El cuarto capítulo recoge todas las pruebas realizadas con la aplicación, incluyendo dos experimentos concretos. El primero compara varios modelos en distintos conjuntos de datos de creciente complejidad, mientras que el segundo compara varios modelos en diferentes máquinas con diversos recursos de procesamiento. Este capítulo detalla la metodología de los experimentos y presenta los resultados obtenidos, analizando el impacto del consumo energético en el rendimiento de los modelos.

El último capítulo concluye la memoria resumiendo los principales hallazgos del proyecto y evaluando el grado de consecución de los objetivos planteados. Además, este capítulo incluye una reflexión sobre el conocimiento adquirido y sugiere posibles direcciones para trabajos futuros, proponiendo mejoras y ampliaciones a los estudios realizados.

Capítulo 2

Estado del arte

2.1 Revisión de la literatura

El impacto energético del aprendizaje automático ha empezado ha desarrollarse recientemente como una preocupación significativa debido a la creciente complejidad y escala de los modelos utilizados en este campo. A pesar de la mayor concienciación existente entorno al consumo de los grandes modelos computacionales, muchas de las investigaciones actuales se ocupan únicamente de obtener las mejores predicciones posibles, independientemente del coste computacional que esto suponga. Sin embargo, algunas investigaciones recientes sí han intentado cuantificar y mitigar el consumo energético asociado con el entrenamiento y la inferencia de modelos de aprendizaje automático, desarrollando herramientas que permitan a los científicos de datos medir las emisiones de sus modelos. Lottick y col., 2019 [7] fue uno de los primeros artículos en proponer una declaración sistematizada de las emisiones de carbono como parte del proceso de elección de modelos de aprendizaje, proponiendo un modelo de informe energético fácilmente accesible para desarrolladores sin necesidad de análisis a nivel industrial.

Varios estudios han reportado el consumo energético de modelos de aprendizaje automático, destacando el alto costo energético de los grandes modelos de lenguaje natural y visión por computador. Por ejemplo, un estudio de 2019 de Strubell, Ganesh y McCallum [8] encontró que el entrenamiento de un modelo Transformer para búsqueda de arquitectura neuronal (NAS, una técnica para automatizar el diseño de redes neuronales artificiales) puede consumir tanta energía como el gasto de varios automóviles durante toda su vida útil. Otros estudios como Dodge y col., 2022 [9] se centran en el gasto de la inteligencia artificial cuando los modelos son entrenados en instancias en la «nube» (centros de datos distribuidos), estudiando como pueden reducirse las emisiones de CO_2 considerando la elección de región geográfica y hora del día en la que se lleva a

cabo el entrenamiento. Estos hallazgos han impulsado la investigación en técnicas de optimización y eficiencia, tales como la poda de redes neuronales, la cuantización de pesos, y el uso de hardware especializado como TPU (Tensor Processing Units) y FPGA (Field-Programmable Gate Arrays), que pueden reducir significativamente el consumo energético [10].

A día de hoy existen un gran número de herramientas para medir el consumo de energía en los modelos de aprendizaje. Sin embargo, el uso de esta métrica no se ha generalizado todavía como parte del proceso. En García-Martín y col., 2019 [2] se hace una revisión de varias de estas herramientas y los métodos más destacados para estimar el consumo. En ella se observa que se debe encontrar un equilibrio entre las herramientas que proporcionan un resultado detallado de la energía con una gran carga adicional en el desarrollo y aquellas que proporcionan un resultado muy aproximado pero de fácil disponibilidad. Son estas últimas las que serán más interesantes para integrar de forma sencilla en un proceso de elección de modelos de aprendizaje, al proporcionar información en tiempo real sobre los algoritmos utilizados.

Entre los proyectos destacados se encuentra CodeCarbon [11], una iniciativa que se ha centrado en medir la huella de carbono de las actividades computacionales en aprendizaje automático mediante un paquete para Python fácilmente integrable con librerías existentes. Otras herramientas ([12]) proponen un método de estimación sin necesidad de llegar a entrenar los modelos, a través de la recopilación previa de medidas de calidad de consumo energético en modelos de base. El *Machine Learning Emissions Calculator* (MLCO2)[13] presenta otro modelo de medida basado en la localización, la red eléctrica y el tiempo utilizado por el servidor responsable del entrenamiento. Otras herramientas disponibles para Python incluyen *experiment-impact-tracker* [14], Carbontracker [15] y Eco2AI [16]. Estos paquetes utilizan métodos similares de estimación de la energía utilizada por CPUs y GPUs y su correlación con las emisiones de carbono de acuerdo a la red eléctrica utilizada.

Los resultados obtenidos de estas investigaciones resaltan la necesidad de considerar la eficiencia energética como un criterio fundamental en el diseño y entrenamiento de modelos de aprendizaje automático. Numerosas iniciativas han mostrado que, mediante la optimización de código y la selección de infraestructuras más sostenibles, se pueden lograr reducciones significativas en el consumo de energía y emisiones de carbono. Otros estudios han demostrado que la implementación de modelos más eficientes no necesariamente compromete la precisión, sugiriendo que es posible alcanzar un balance entre rendimiento y sostenibilidad.

En conclusión, la literatura destaca la importancia de desarrollar métodos y tecnologías que reduzcan el consumo energético en el aprendizaje automático. La medición del consumo eléctrico o las emisiones de carbono son cruciales para aumentar la concienciación sobre este problema y proporcionar herramientas prácticas para medir y reducir la huella de carbono. A medida que la demanda de modelos grandes y complejos continúa

creciendo, es imperativo que la comunidad de investigación se enfoque en soluciones sostenibles que equilibren la eficiencia energética con el rendimiento del modelo.

2.2 Introducción al aprendizaje automático

El aprendizaje automático se ha consolidado como una disciplina fundamental dentro del campo de la ingeniería y la informática, debido a su capacidad para desarrollar sistemas que pueden aprender y mejorar a partir de la experiencia sin ser explícitamente programados. En el contexto de este Trabajo de Fin de Grado, se desarrollará una aplicación para medir el consumo energético de proyectos de aprendizaje automático, un aspecto crítico dado el creciente uso de estos modelos en diversas aplicaciones y su consecuente impacto ambiental. Para explicar su funcionamiento, es necesario describir primero los conceptos básicos de esta materia.

El aprendizaje automático puede dividirse principalmente en dos categorías: aprendizaje supervisado y no supervisado [17]. En el aprendizaje supervisado, el modelo se entrena utilizando un conjunto de datos etiquetados, es decir, cada muestra del conjunto de datos está asociada con una etiqueta que representa el resultado esperado. Las tareas más comunes en este tipo de aprendizaje son la clasificación, donde el objetivo es asignar muestras a una de varias categorías predefinidas, y la regresión, donde se predice un valor continuo. En contraste, el aprendizaje no supervisado no utiliza etiquetas. En su lugar, el modelo intenta identificar patrones y estructuras inherentes en los datos. Las tareas típicas en este enfoque incluyen la agrupación (*clustering*), donde el objetivo es agrupar muestras similares, y la reducción de dimensionalidad, que busca simplificar los datos manteniendo su estructura esencial. Este proyecto se centrará en tareas de clasificación mediante aprendizaje supervisado.

Dentro de las tareas de clasificación se pueden distinguir dos casos: la clasificación binaria y la multiclase. La clasificación multiclase es una técnica de aprendizaje automático supervisado en la que el objetivo es categorizar cada muestra de datos en una de tres o más clases posibles. A diferencia de la clasificación binaria, que se limita a dos clases, la clasificación multiclase es más compleja debido a varios factores. En primer lugar, el aumento en el número de clases incrementa la dificultad para separar las categorías correctamente, ya que el modelo debe aprender a distinguir entre múltiples fronteras de decisión. Además, el desequilibrio entre clases puede ser más pronunciado, lo que complica el entrenamiento del modelo. Por último, las métricas de evaluación se vuelven más complejas, ya que es necesario considerar el rendimiento del modelo en todas las clases para obtener una visión completa de su precisión y robustez.

Entre los modelos más habituales en el aprendizaje automático [18] se encuentran los árboles de decisión, los modelos de Naive Bayes, las máquinas de soporte vectorial

(SVM por sus siglas en inglés) y las redes neuronales. Estas últimas, inspiradas en la estructura del cerebro humano, son particularmente relevantes en problemas complejos de reconocimiento de patrones y se destacan por su capacidad de aproximar funciones no lineales mediante la composición de varias capas de neuronas artificiales. Otros modelos, como los bosques aleatorios (Random Forests) y los modelos de potenciación de gradiente (Gradient Boosting Machines), también son ampliamente utilizados debido a su capacidad para mejorar la precisión a través de la combinación de múltiples predictores.

La calidad y estructura de los conjuntos de datos juegan un papel crucial en el éxito de los modelos de aprendizaje automático. Un conjunto de datos típico está compuesto por muestras (o instancias), cada una descrita por una serie de atributos (o características) y, en el caso de aprendizaje supervisado, asociada con una etiqueta, la variable objetivo. Los atributos pueden ser de diversos tipos: continuos, categóricos, binarios u ordinales.

Para evaluar el rendimiento de un modelo, los datos se dividen generalmente en un conjunto de entrenamiento y un conjunto de prueba. El conjunto de entrenamiento se utiliza para ajustar los parámetros del modelo, mientras que el conjunto de prueba se emplea para evaluar su capacidad de generalización a datos no vistos. Este proceso ocurre en varias etapas clave. Primero, los datos se recopilan y preprocesan para asegurar su calidad y pertinencia. Luego, el modelo se entrena utilizando el conjunto de entrenamiento, ajustando sus parámetros internos para minimizar el error en la predicción. Este ajuste se realiza mediante algoritmos de optimización, como el descenso del gradiente, que iterativamente modifica los parámetros para encontrar la configuración que produce el menor error. Finalmente, el modelo se utiliza para predecir la variable objetivo en los datos del conjunto de prueba y se evalúa comparando las predicciones con las etiquetas reales de los datos [19].

La precisión de un modelo de aprendizaje automático está influenciada por diversos factores. La calidad y cantidad de los datos de entrenamiento son cruciales; datos ruidosos o insuficientes pueden llevar a modelos inexactos. Además, la elección del modelo y sus hiperparámetros, como la profundidad de los árboles en un bosque aleatorio o la tasa de aprendizaje en un modelo de potenciación de gradiente, también juegan un papel determinante. La complejidad del modelo debe equilibrarse cuidadosamente para evitar el sobreajuste, donde el modelo se ajusta demasiado bien a los datos de entrenamiento y falla en generalizar a nuevos datos. Otro problema común en el aprendizaje automático es el desequilibrio de clases, donde una clase puede estar significativamente subrepresentada en el conjunto de datos, afectando la precisión del modelo, que tiende a ignorar las clases minoritarias. Algunas estrategias para subsanar este problema son la penalización de las clases mayoritarias mediante la aplicación de pesos distintos a cada clase o la utilización de métodos de ensamblaje que no son sensibles al desequilibrio de clases [20].

A medida que aumentan la escala y complejidad de los modelos, el consumo energético de los modelos de aprendizaje automático se volverá un aspecto más crítico. Modelos más

complejos, como las redes neuronales profundas, requieren un mayor poder computacional y, por ende, un mayor consumo de energía durante el entrenamiento y la inferencia. Otros factores que afectan el consumo energético incluyen la arquitectura del modelo, la cantidad de datos, el hardware utilizado (como CPUs, GPUs o TPUs) y la eficiencia de los algoritmos de optimización. Medir y optimizar el consumo energético es esencial para desarrollar soluciones sostenibles y escalables, especialmente en aplicaciones a gran escala como el procesamiento de grandes volúmenes de datos o la inteligencia artificial en tiempo real [2].

Este proyecto se centrará en la implementación de una aplicación para medir el consumo energético de proyectos de aprendizaje automático. Dicha aplicación considerará varios aspectos técnicos y metodológicos del aprendizaje automático, incluyendo la selección y evaluación de modelos, la gestión de conjuntos de datos, y la mitigación de problemas comunes, con el objetivo de contribuir a la eficiencia energética y sostenibilidad en el desarrollo de estas tecnologías.

2.3 Librerías empleadas

2.3.1 Entorno de desarrollo: Visual Studio Code y WSL

VSCode es un editor de código fuente abierto altamente extensible que se ha convertido en una herramienta predilecta entre desarrolladores debido a su interfaz amigable, integración con Git y soporte para una amplia gama de lenguajes de programación y extensiones. WSL (Windows Subsystem for Linux) es una característica de Windows que permite ejecutar un entorno de Linux completo directamente sobre Windows sin la necesidad de una máquina virtual o sistemas de arranque dual [21].

Es posible combinar VSCode con WSL mediante una extensión específica ofrecida por VSCode para WSL que permite a los usuarios abrir carpetas y archivos directamente en el sistema de archivos de Linux, beneficiándose de las capacidades avanzadas de depuración y análisis de código que ofrece el editor. Además, la extensión proporciona a los desarrolladores acceso a un terminal de Linux real, donde pueden ejecutar y probar su código en el mismo entorno en el que se desplegará.

2.3.2 CodeCarbon

CodeCarbon [11][22] es un paquete creado con la intención de permitir a desarrolladores monitorizar las emisiones de dióxido de carbono (CO_2) producidas por aplicaciones

en Inteligencia Artificial y modelos de Aprendizaje Automático, que surge de la motivación de contar con una forma de registrar las enormes cantidades de energía que el auge de la IA ha provocado en la industria. El incremento del rendimiento y la precisión de los modelos de Aprendizaje Automático que se ha producido en años recientes se ha logrado a cambio de la utilización de enormes cantidades de información para conseguir el aprendizaje de los patrones y características subyacentes. Así, los modelos más avanzados emplean cantidades significativas de poder computacional, entrenando en procesadores avanzados durante semanas o meses y consumiendo en el proceso una gran cantidad de energía. Dependiendo de la red eléctrica utilizada, este desarrollo puede comportar la emisión de grandes cantidades de gases de efecto invernadero como el CO_2.

CodeCarbon estima la huella de carbono de una aplicación medida como kilogramos de CO_2 equivalentes, o CO_2eq, una medida estandarizada utilizada para expresar la capacidad de calentamiento global de varios gases de efecto invernadero como la cantidad de CO_2 que causaría un impacto ambiental equivalente. Para tareas de computación, que emiten CO_2 por medio de la electricidad que están consumiendo y que es generada como parte de la red eléctrica (por ejemplo, mediante la quema de combustibles fósiles como el carbón) las emisiones de carbono se miden en kilogramos de CO_2 equivalentes por kilovatio-hora. De esta forma, las emisiones de dióxido de carbono totales se calculan como el producto de la intensidad de carbono de la electricidad utilizada para la computación y la energía consumida por la infraestructura.

La intensidad de carbono de la electricidad se calcula como la media ponderada de las emisiones de las distintas fuentes de energía usadas para generar electricidad, incluyendo combustibles fósiles y renovables. En la herramienta se asigna un valor conocido de dióxido de carbono emitido por kilovatio-hora generado para cada uno de los combustibles (carbón, petróleo y gas natural). Otras fuentes renovables o consideradas como de bajo carbono incluyen la energía solar, hidroeléctrica, biomasa o geotérmica. La intensidad de carbono de cada combustible individual se calcula en base a medidas de generación de carbono y electricidad en los Estados Unidos, y aplicadas de forma generalizada en el resto del mundo. Cada red eléctrica local incluye una mezcla distinta de fuentes de energía y tiene por lo tanto asignada una intensidad de carbono total particular.

2.3.3 scikit-learn

Scikit-learn [23] es un módulo desarrollado para Python que integra un amplio rango de algoritmos de aprendizaje automático de última generación para problemas tanto supervisados como no supervisados. Este paquete pretende llevar el aprendizaje automático a desarrolladores no especialistas mediante el uso de un lenguaje generalista de alto nivel. Se hace hincapié en la facilidad de uso, el rendimiento, la documentación y la consistencia de la API [24]. Tiene las mínimas dependencias necesarias y está distribuido

bajo la licencia BSD, con el objetivo de incentivar su uso tanto en ambientes educativos como comerciales.

Scikit-learn expone una gran variedad de algoritmos de aprendizaje utilizando una interfaz consistente y orientada a la resolución de tareas, lo que permite una comparación sencilla entre distintos métodos de aprendizaje para una misma aplicación. Al depender del ecosistema científico de Python, puede ser integrado con facilidad en aplicaciones que se salgan del rango tradicional del análisis estadístico de datos. Además, los algoritmos, que han sido implementados en un lenguaje de alto nivel, pueden ser utilizados como bloques de construcción para desarrollar estrategias más complejas que se adecuen a cada caso particular [25].

2.3.4 Matplotlib

Matplotlib es una de las librerías más destacadas y utilizadas en el ecosistema de Python para la creación de gráficos y visualizaciones de datos. Esta herramienta permite a los desarrolladores y científicos de datos generar una amplia variedad de gráficos estáticos, animados e interactivos con relativa facilidad. Matplotlib se caracteriza por su flexibilidad y capacidad para crear visualizaciones de alta calidad y personalizables, que van desde simples gráficos de líneas y barras hasta complejas visualizaciones tridimensionales y de mapas de calor. Además, su integración con otras librerías populares de Python, como NumPy y pandas, facilita el proceso de análisis y visualización de datos, convirtiéndola en una opción preferida en ámbitos académicos y profesionales.

El diseño de Matplotlib sigue una filosofía similar a la de MATLAB, lo que hace que sea especialmente accesible para usuarios familiarizados con ese entorno. Sin embargo, a diferencia de MATLAB, Matplotlib es de código abierto y gratuito. Las capacidades avanzadas de esta librería incluyen la personalización detallada de todos los elementos del gráfico, la posibilidad de exportar gráficos en múltiples formatos (como PNG, PDF y SVG), y la creación de gráficos interactivos utilizando bibliotecas adicionales como mpld3 y Plotly.

2.3.5 Microsoft Azure

Microsoft Azure es una plataforma de servicios en la nube que ofrece una amplia gama de herramientas y recursos para el despliegue y gestión de aplicaciones y servicios. Entre sus múltiples servicios, Azure permite a los usuarios crear y gestionar máquinas virtuales (VMs) con diversas configuraciones de recursos, adaptándose a las necesidades específicas de cada proyecto. Esta capacidad es especialmente valiosa en el campo del aprendizaje automático, donde las tareas de entrenamiento y prueba de modelos requieren

recursos computacionales significativos y variados. Con Azure, los desarrolladores e investigadores pueden seleccionar configuraciones específicas de CPU, GPU, memoria y almacenamiento para optimizar el rendimiento y costo de sus experimentos de aprendizaje automático. Microsoft Azure ofrece un programa especial para estudiantes llamado "Azure for Students", que proporciona acceso gratuito a una variedad de servicios en la nube. Este programa incluye un crédito inicial de $100 USD para usar en cualquier servicio de Azure durante 12 meses, sin necesidad de una tarjeta de crédito para registrarse.

2.4 Modelos utilizados

2.4.1 Modelos lineales: regresión logística

Los modelos lineales son una clase fundamental de técnicas estadísticas y de aprendizaje automático que se utilizan para predecir una variable objetivo a partir de una o más variables independientes. La característica principal de estos modelos es la relación lineal entre las variables independientes y la variable objetivo. En su forma más simple, el modelo lineal se representa mediante la ecuación $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n$, donde y es la variable dependiente, $x_1, x_2, \ldots, x_n$ son las variables independientes, y $\beta_1, \beta_2, \ldots, \beta_n$ son los coeficientes que representan el impacto de cada variable independiente sobre la variable dependiente.

Dentro de los modelos lineales, la regresión logística es especialmente relevante para problemas de clasificación [26]. A diferencia de la regresión lineal, que se utiliza para problemas de predicción continua, la regresión logística se emplea cuando la variable objetivo es categórica. En su forma binaria más simple, la regresión logística predice la probabilidad de que una observación pertenezca a una de dos posibles categorías. La función logística o sigmoide,

$$P(y = 1 \mid x) = \frac{1}{1 + e^{-\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n}},$$

transforma cualquier valor real de la combinación lineal de las variables independientes en un valor entre 0 y 1, lo que se interpreta como una probabilidad. Los parámetros desconocidos β_i son estimados habitualmente a través del método de máxima verosimilitud.

El uso de la regresión logística en problemas de clasificación es ventajoso por varias razones. Primero, es un modelo interpretativo, ya que los coeficientes obtenidos pueden proporcionar una idea clara de cómo cada variable independiente afecta la probabilidad de pertenecer a una categoría específica. Además, la regresión logística es relativamente fácil de implementar y eficiente computacionalmente, lo que la hace adecuada para grandes conjuntos de datos. Por último, aunque su capacidad para capturar relaciones complejas es limitada en comparación con modelos no lineales más avanzados, su simplicidad y

efectividad la convierten en una opción sólida para establecer una línea base en proyectos de clasificación, permitiendo comparaciones posteriores con modelos más complejos.

La librería scikit-learn proporciona una interfaz para construir y evaluar modelos de regresión logística a través de su clase `LogisticRegression` [27]. Esta clase permite a los usuarios configurar parámetros como la regularización, la penalización y el algoritmo de optimización. Esta implementación admite dos tipos de penalización: `L1` (Lasso) y `L2` (Ridge) que ayudan a prevenir el sobreajuste. Además, permite ajustar la fuerza de la regularización a través de un parámetro `C`. Esta flexibilidad facilita la adaptación del modelo a diferentes problemas y conjuntos de datos. La personalización aumenta al elegir un algoritmo de optimización mediante el parámetro `solver`, que admite varias opciones que pueden adaptarse al tamaño o el tipo de atributos del conjunto de datos a utilizar. Adicionalmente, `LogisticRegression` es capaz de manejar problemas de clasificación binaria y multiclase, con diferentes estrategias de acuerdo al algoritmo de resolución escogido.

2.4.2 Árboles de decisión: bosque aleatorio

Otra técnica popular de aprendizaje automático utilizada para problemas tanto de clasificación como de regresión son los árboles de decisión [18, 28]. Su estructura jerárquica consiste en nodos de decisión que dividen iterativamente el conjunto de datos en subconjuntos más homogéneos, facilitando así la predicción de la variable objetivo. Cada nodo del árbol representa una prueba sobre un atributo específico, y cada rama denota el resultado de la prueba. Los árboles de decisión son fáciles de interpretar y visualizar, lo que los hace muy útiles para entender las decisiones del modelo y comunicar los resultados a personas no técnicas.

Sin embargo, los árboles de decisión tienen algunas limitaciones, como su tendencia a sobreajustarse a los datos de entrenamiento, lo que puede llevar a un desempeño deficiente en datos nuevos. Para abordar estas limitaciones, se emplean técnicas de ensamblado como los bosques aleatorios (*Random Forests*). Un bosque aleatorio es un conjunto de árboles de decisión entrenados sobre diferentes subconjuntos del conjunto de datos y con diferentes características seleccionadas aleatoriamente. La predicción final se obtiene mediante el promedio de las predicciones individuales de los árboles (en el caso de regresión) o mediante un voto mayoritario (en el caso de clasificación). Esta metodología mejora significativamente la precisión del modelo y reduce el riesgo de sobreajuste.

El uso de bosques aleatorios en problemas de clasificación ofrece varias ventajas. Primero, al combinar múltiples árboles, el modelo se vuelve más robusto y generalizable, mitigando la influencia de outliers y ruido en los datos. Además, los bosques aleatorios proporcionan una medida de importancia de las características, lo que permite identificar las variables más relevantes para la predicción. Esta capacidad es especialmente útil en

contextos académicos y profesionales donde la interpretación del modelo y la identificación de factores clave son cruciales para la toma de decisiones informadas.

La implementación de un modelo de bosque aleatorio en la librería scikit-learn se ofrece a través de la clase `RandomForestClassifier` [18, 29], que permite ajustar diversos parámetros como el número de árboles (`n_estimators`), la profundidad máxima de los árboles (`max_depth`) y el número mínimo de muestras por hoja (`min_samples_leaf`).

2.4.3 Máquinas de vector soporte (SVM)

En las máquinas de vector soporte (*Support Vector Machines*, SVM) [30] el principal objetivo es encontrar el hiperplano óptimo que maximiza el margen entre las diferentes clases en un espacio de características. Este margen es la distancia más amplia posible entre el hiperplano y los puntos de datos más cercanos de cualquier clase, conocidos como vectores soporte. La SVM es particularmente eficaz en espacios de alta dimensionalidad y es robusta frente al sobreajuste, lo que la hace adecuada para conjuntos de datos complejos.

Una de las ventajas clave de las SVM es su capacidad para manejar problemas no lineales mediante el uso de funciones kernel. Estos núcleos transforman los datos en un espacio de mayor dimensión donde un hiperplano lineal puede separarlos. Los tipos de kernel más comunes incluyen el lineal, polinómico, radial (RBF), y sigmoide. Esta flexibilidad permite a las SVM abordar una amplia gama de problemas de clasificación, incluso cuando las relaciones entre las características y las etiquetas son altamente no lineales. Sin embargo, para que funcione de forma adecuada es necesario seleccionar el kernel apropiado y ajustar sus parámetros, por ejemplo, mediante validación cruzada.

En el contexto de problemas de clasificación, las SVM son especialmente útiles debido a su alta precisión y su capacidad para manejar tanto clases linealmente separables como no separables. En escenarios de clasificación binaria, las SVM son capaces de encontrar la frontera de decisión que maximiza la separación entre las dos clases. Para problemas de clasificación multiclase, se pueden aplicar estrategias para descomponer el problema en múltiples problemas de clasificación binaria. Aunque las SVM pueden ser computacionalmente intensivas, especialmente con grandes conjuntos de datos, tienen una gran eficacia en términos de rendimiento y de capacidad para generalizar bien a datos no vistos.

La clase `SVC` [31] de scikit-learn permite ajustar parámetros clave como el tipo de kernel, el parámetro de regularización (`C`) y coeficientes específicos del kernel elegido.

2.4.4 Vecinos más cercanos (k-NN)

El método de vecinos más cercanos (k-Nearest Neighbors, k-NN) es una técnica de aprendizaje supervisado utilizada para resolver problemas tanto de clasificación como de regresión [32]. Este método se basa en la premisa de que objetos similares generalmente se encuentran cerca en el espacio de características. En problemas de clasificación, el algoritmo k-NN asigna una etiqueta a una nueva instancia basándose en las etiquetas de sus k vecinos más cercanos en el conjunto de datos de entrenamiento. La cercanía o similitud entre las instancias se mide generalmente utilizando distancias métricas, como la distancia euclidiana, Manhattan, o Minkowski.

Una de las principales ventajas del k-NN es su simplicidad y facilidad de implementación. A diferencia de otros algoritmos que requieren un proceso de entrenamiento explícito, k-NN es un algoritmo perezoso que almacena todas las instancias del conjunto de entrenamiento y realiza el cálculo de las distancias en el momento de la predicción. Esta característica lo convierte en un método intuitivo y fácil de entender, aunque también implica que puede ser computacionalmente costoso, especialmente con grandes conjuntos de datos. Además, la elección del número de vecinos (k) y la métrica de distancia son cruciales para el rendimiento del modelo. Valores de k demasiado bajos pueden llevar a un sobreajuste, mientras que valores demasiado altos pueden conducir a un subajuste.

En el contexto de problemas de clasificación, el k-NN es especialmente útil en situaciones donde las fronteras de decisión entre clases no son lineales y pueden ser complejas. Debido a su naturaleza basada en instancias, el k-NN puede capturar patrones locales en los datos y adaptarse a cambios en la distribución de las clases. Sin embargo, el rendimiento de k-NN puede verse afectado por la presencia de ruido y características irrelevantes, lo que subraya la importancia de la preprocesamiento de datos, como la normalización y la selección de características. A pesar de estas limitaciones, el k-NN sigue siendo una herramienta valiosa para la clasificación debido a su simplicidad y flexibilidad.

La clase `KNeighborsClassifier` [33] de la librería scikit-learn proporciona la interfaz para configurar y utilizar el algoritmo k-NN. Esta clase ofrece la posibilidad de especificar el número de vecinos a considerar mediante el parámetro `n_neighbors` y de seleccionar la métrica de distancia adecuada utilizando el parámetro `metric`, que incluye opciones como la distancia euclidiana y Manhattan. Además, scikit-learn permite ajustar parámetros adicionales como el peso de los vecinos, que puede ser uniforme o basado en la distancia.

2.4.5 Naive Bayes (Naive Bayes gaussiano)

Los métodos Naive Bayes [18, 34] son un conjunto de algoritmos de aprendizaje supervisado basados en la aplicación del teorema de Bayes con la suposición "ingenua"

(en inglés, *naive*) de independencia condicional entre cada par de características dado el valor de la variable de clase. El teorema de Bayes postula la siguiente relación, dada la variable de clase y y los vectores característica dependientes x_1 a x_n:

$$P(y \mid x_1, \ldots, x_n) = \frac{P(y)P(x_1, \ldots, x_n \mid y)}{P(x_1, \ldots, x_n)}$$

Usando la suposición ingenua de independencia condicional de forma que

$$P(x_i \mid y, x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) = P(x_i \mid y),$$

para todo i, esta relación se simplifica a

$$P(y \mid x_1, \ldots, x_n) = \frac{P(y) \prod_{i=1}^{n} P(x_i \mid y)}{P(x_1, \ldots, x_n)}$$

Como $P(x_1, \ldots, x_n)$ es constante dada la entrada, se puede utilizar la siguiente regla de clasificación:

$$P(y \mid x_1, \ldots, x_n) \propto P(y) \prod_{i=1}^{n} P(x_i \mid y) \Rightarrow \hat{y} = \arg\max_{y} P(y) \prod_{i=1}^{n} P(x_i \mid y),$$

y se puede usar la estimación de máximo a posteriori (MAP) para estimar $P(y)$ y $P(x_i \mid y)$; y la primera expresión es entonces la frecuencia relativa de la clase y en el conjunto de entrenamiento.

Los diferentes clasificadores Naive-Bayes difieren sobre todo en la suposición que realicen sobre la distribución de $P(x_i \mid y)$. El algoritmo Naive-Bayes gaussiano es una variante específica de Naive-Bayes que se utiliza cuando las características son continuas y se asumen distribuidas según una distribución normal (gaussiana), lo que es común en muchas aplicaciones reales. En este caso, el clasificador estima los parámetros de la distribución (media y varianza) para cada característica en cada clase utilizando los datos de entrenamiento. Durante la clasificación de una nueva instancia, el algoritmo calcula la probabilidad de que esta instancia pertenezca a cada clase basándose en las distribuciones gaussianas estimadas y asigna la clase con la probabilidad a posteriori más alta.

A pesar de sus suposiciones aparentemente simplificadas, los clasificadores Naive-Bayes obtienen buenos resultados en muchos problemas del mundo real, especialmente en aplicaciones como la clasificación de documentos y filtros de spam; y requieren pequeñas cantidades de datos de entrenamiento para estimar los parámetros necesarios. Estos clasificadores pueden llegar a ser extremadamente rápidos comparados a otros métodos más sofisticados. La separación de las características condicionales de clase significa que cada distribución puede ser estimada independientemente como una

distribución unidimensional. Esto a su vez ayuda a aliviar problemas provocados por la dimensionalidad de las distribuciones.

La implementación de este algoritmo en la librería scikit-learn se encuentra en la clase `GaussianNB` [35].

2.4.6 Métodos de ensamblaje (ensemble): máquinas de potenciación de gradiente

El método de ensamblaje por máquinas de potenciación de gradiente (Gradient Boosting Machines, GBM) es una técnica de aprendizaje automático que combina múltiples modelos débiles para crear un modelo más fuerte y preciso. La idea central de GBM es construir el modelo de forma secuencial, donde cada modelo sucesivo intenta corregir los errores del modelo anterior. En el contexto de problemas de clasificación, cada nuevo árbol de decisión se ajusta a los residuos de los árboles anteriores, utilizando el gradiente de la función de pérdida como guía. Esta estrategia permite que GBM capture relaciones complejas y no lineales en los datos, mejorando considerablemente la precisión del modelo.

Uno de los mayores beneficios de usar máquinas de potenciación de gradiente para problemas de clasificación es su capacidad para manejar datos heterogéneos y características con diferentes escalas y distribuciones. GBM puede ajustarse a los datos de manera muy detallada, lo que lo hace adecuado para problemas donde las relaciones entre las variables no son lineales ni simples. Sin embargo, este alto nivel de ajuste también puede llevar a un sobreajuste si no se controla adecuadamente. Por ello, es crucial utilizar técnicas como la validación cruzada y ajustar parámetros como el número de árboles, la tasa de aprendizaje, y la profundidad máxima de los árboles para encontrar el equilibrio adecuado entre sesgo y varianza.

En problemas de clasificación, GBM es particularmente eficaz debido a su capacidad para manejar clases no balanceadas y proporcionar probabilidades de clasificación en lugar de simplemente etiquetas de clase. Esto es especialmente útil en aplicaciones como la detección de fraudes, diagnóstico médico y otros escenarios donde la probabilidad de pertenecer a una clase particular puede ser más informativa que la clasificación binaria. Además, las implementaciones modernas de GBM, como XGBoost, LightGBM y CatBoost, han optimizado significativamente la velocidad y eficiencia del algoritmo, permitiendo su aplicación en grandes conjuntos de datos y en tiempo real.

La implementación de máquinas de potenciación de gradiente en la librería scikit-learn es accesible a través de la clase GradientBoostingClassifier [36] permite a los usuarios ajustar una variedad de hiperparámetros para optimizar el rendimiento del modelo. Los usuarios pueden especificar el número de árboles, la tasa de aprendizaje, y otros

parámetros clave para controlar la complejidad y la capacidad de generalización del modelo.

2.4.7 Redes neuronales (Deep Neural Networks, DNN): Multi-layer Perceptron

Las redes neuronales están compuestas por capas de neuronas artificiales que imitan el comportamiento de las neuronas en el cerebro humano. Cada neurona realiza una operación matemática simple sobre la entrada y pasa la salida a la siguiente capa. A través de múltiples capas, la red es capaz de capturar patrones y características complejas en los datos. Estás propiedades pueden aplicarse a situaciones de aprendizaje supervisado mediante el uso de algoritmos como el perceptrón multicapa (Multilayer Perceptron, MLP).

El algoritmo del perceptrón multicapa [37] es un tipo de red neuronal supervisada que consiste en una capa de entrada, una o más capas ocultas y una capa de salida. Cada capa está totalmente conectada a la siguiente, y cada conexión tiene un peso ajustable que se optimiza durante el proceso de entrenamiento. En problemas de clasificación, el MLP es especialmente útil debido a su capacidad para aprender representaciones no lineales complejas de los datos. El algoritmo utiliza funciones de activación no lineales, como la función sigmoide o la ReLU (Rectified Linear Unit), que permiten que la red capture y modele relaciones no lineales en los datos. El entrenamiento de un MLP se realiza mediante un proceso llamado retropropagación del error (backpropagation), que ajusta iterativamente los pesos para minimizar una función de pérdida, típicamente la entropía cruzada en problemas de clasificación.

El uso del MLP ofrece varias ventajas. Primero, debido a su estructura de múltiples capas, el MLP puede aprender y generalizar bien a partir de datos complejos y de alta dimensionalidad. Esto es particularmente útil en dominios como el reconocimiento de imágenes, la detección de voz y la clasificación de texto, donde las relaciones entre las características no son triviales. Además, las redes neuronales pueden ser adaptadas a problemas multiclase con facilidad, utilizando técnicas como la salida softmax en la capa final para obtener probabilidades de clase. Sin embargo, el MLP también requiere una cantidad considerable de datos y poder computacional para entrenar eficazmente, y la selección de la arquitectura y los hiperparámetros adecuados puede ser un proceso desafiante que a menudo implica ensayo y error y validación cruzada.

La librería scikit-learn implementa la clase MLPClassifier [38] para el algoritmo del perceptrón multicapa. Los usuarios pueden definir parámetros como el número de capas ocultas y el número de neuronas por capa (hidden_layer_sizes), la función de activación (activation), y el algoritmo de optimización (solver). Scikit-learn también permite ajustar

la tasa de aprendizaje (learning_rate) y utilizar técnicas de regularización para prevenir el sobreajuste.

2.5 Datos utilizados

Todos los conjuntos de datos utilizados en los análisis realizados están disponibles de forma libre en la web y proceden de dos fuentes: el repositorio de aprendizaje automático de la Universidad de California Irvine y la plataforma OpenML.

El repositorio de aprendizaje automático de la UCI [39] es una colección de bases de datos, teorías de dominios y generadores de datos que son utilizados por la comunidad de aprendizaje automático para el análisis empírico de algoritmos de aprendizaje automático. El archivo fue creado en 1987 como un servidor FTP por David Aha y otros compañeros estudiantes en la UCI. Desde entonces ha sido ampliamente utilizado por estudiantes, educadores e investigadores de todo el mundo como una fuente primaria de conjuntos de datos para aprendizaje automático.

OpenML [40] es una plataforma abierta para compartir conjuntos de datos, algoritmos y experimentos, que tiene como objetivo lograr que las investigaciones sobre aprendizaje automático sean más fácilmente accesibles y reutilizables. Esta plataforma contiene un repositorio con más de cinco mil conjuntos de datos y resultados de experimentos que otros usuarios han realizado con ellos. Además, ofrece librerías propias para integrar la recuperación de sus datos directamente con el código de desarrollo de modelos de aprendizaje. Scikit-Learn ofrece también una función para descargar los datos fácilmente de esta plataforma, `fetch_openml` [41], que toma el nombre de un conjunto de datos y devuelve un dataframe de Pandas con toda la información disponible.

La tabla 2.1 recoge las características más importantes de los conjuntos de datos empleados, que serán descritos a continuación en mayor detalle.

Tabla 2.1: Características de los conjuntos de datos utilizados

Nombre	Características numéricas	Características categóricas	Nº clases	Nº muestras
Iris	4	0	3	150
Ionosfera	34	0	2	351
Autenticación de billetes	4	0	2	1372
Fonemas	5	0	2	5404
Electroencefalograma	14	0	2	14980
Electricidad	7	1	2	45312

2.5.1 Iris

Iris esta entre las primeras bases de datos recogidas, y es una de las más conocidas y utilizadas en la literatura sobre reconocimiento de patrones. Los datos originales fueron publicados por R.A. Fisher en 1936, y la versión utilizada en este proyecto procede de la UCI (1988) [42]. El conjunto contiene tres clases distintas con 50 ejemplos cada una, donde cada clase se refiere a una especie del género de plantas Iris. Cada ejemplo contiene cuatro atributos que describen la longitud y la anchura del sépalo y el pétalo de la planta en centímetros. Las tres opciones de clasificación son Iris Setosa, Iris Versicolour e Iris Virginica, donde una de las clases es linealmente separable de las otras dos, y estas últimas no son separable entre sí de forma lineal.

La versión utilizada corresponde a la implementación `load_iris` de scikit-learn.

2.5.2 Ionosfera

Se trata de un conjunto de datos con 351 muestras procedente del repositorio de la UCI [43]. Contiene datos de radar obtenidos por el grupo de física espacial de la Universidad John Hopkins y donados por Vince Sigillito en 1989. El sistema radar está ubicado en Goose Bay, Labrador y consiste en un array de 16 antenas de alta frecuencia. El objetivo es la medición de electrones libres en la ionosfera y su clasificación binaria entre ”buenas” respuestas del radar que indican evidencia de algún tipo de estructura en la ionosfera y ”malas” respuestas en las que las señales simplemente pasan a través de la ionosfera. Las

señales recibidas se procesaron utilizando una función de autocorrelación con el tiempo de pulso y el número de pulso como argumentos y cada una de las muestras del conjunto de datos está descrita por dos atributos continuos para cada uno de los 17 números de pulso, correspondientes al valor complejo obtenido de la señal electromagnética compleja. Hay por lo tanto un total de 34 características continuas por muestra.

2.5.3 Autenticación de billetes

Este conjunto de datos contiene información extraída de 1372 imágenes tomadas para evaluar un procedimiento de autenticación de billetes. Fue donado en Agosto de 2012 por Volker Lohweg de la Universidad de Ciencias Aplicadas de Ostwestfalen-Lippe, Alemania, al repositorio de la UCI [44]. Para la digitalización de las imágenes tomadas se empleó una cámara industrial normalmente utilizada para la inspección de impresiones. Las imágenes finales tienen un tamaño de 400x400 píxeles con una resolución de alrededor de 660 dpi en escala de grises. Posteriormente, se empleó una herramienta de transformada ondícula para extraer cuatro características continuas de las imágenes. A partir de estas cuatro características, el objetivo es clasificar los billetes en dos clases, representadas en la tabla de datos con un cero para billetes falsos y un uno para billetes auténticos.

2.5.4 Fonemas

Este conjunto de datos tiene como objetivo diferenciar entre sonidos nasales (clase 0) y sonidos orales (clase 1) mediante el análisis de cinco atributos que representan las amplitudes de los cinco primeros armónicos normalizados con la energía total. El formato actual de los datos corresponde al repositorio KEEL [45], aunque originalmente los datos fueron alojados por el proyecto ELENA y obtenidos para el proyecto europeo ESPRIT 5516, ROARS, cuyo objetivo era desarrollar sistemas de reconocimiento del habla para español y francés.

La base de datos consiste en vocales de 1809 sílabas aisladas, con tres puntos de observación por vocal, lo que resulta en 5427 instancias iniciales, que posteriormente se reducen a 5404 tras eliminar aquellas muestras con una amplitud armónica nula. Las observaciones incluyen el momento de máxima energía total y otro dos momentos 8 ms antes y después de este pico. Esta estrategia permite distinguir vocales nasales y orales examinando sus características armónicas en el tiempo. Cada observación contiene cinco atributos correspondientes a los armónicos de cinco fonemas: ”sh” (como en inglés en *she*), ”dcl” (como en *dark*), ”iy” (como la vocal en *she*), ”aa” (como la vocal en *dark*), y ”ao” (como la primera vocal en *water*).

2.5.5 Electroencefalograma

Los datos proceden de una medición continua de un electroencefalograma tomada con el dispositivo neuronal *Emotiv EEG* [46]. La medición tuvo una duración total de 117 segundos,en los que se obtuvieron 14980 muestras individuales. Adicionalmente, el estado del ojo fue detectado mediante una cámara durante la medición del EEG y añadido posteriormente a los datos mediante el análisis de los fotogramas del vídeo. Para cada muestra, se identifico de forma manual el estado de ojo cerrado, al que se asignó un valor de ”1”, y de ojo abierto, con un valor ”0”. La base de datos está ordenada de forma cronológica con el primer valor medido al principio de los datos. Los atributos de cada instante de tiempo corresponden a las 14 medidas EEG del dispositivo, originalmente etiquetadas AF3, F7, F3, FC5, T7, P, O1, O2, P8, T8, FC6, F4, F8, AF4, en ese orden.

2.5.6 Electricidad

Este conjunto de datos fue descrito por M. Harries [47] en 1999 y ha sido ampliamente utilizado para aprendizaje automático desde entonces. Los datos fueron obtenidos del mercado eléctrico australiano de Nuevo Gales del Sur. En este mercado, los precios no son fijos, sino que cambian según la demanda y el abastecimiento de electricidad cada cinco minutos. Además, se tienen en cuenta las transferencias de electricidad para aliviar fluctuaciones procedentes del estado contiguo de Virginia.

El conjunto de datos (denominado originalmente ELEC2) contiene 45.312 muestras fechadas entre el 7 de mayo de 1996 y el 5 de diciembre de 1998. Cada ejemplo del conjunto hace referencia a un periodo de 30 minutos, lo que resulta en 48 muestras individuales para un periodo de un día. Hay ocho atributos para cada muestra: la fecha, el día de la semana, la hora, la demanda eléctrica y el precio en Nuevo Gales del Sur, la demanda eléctrica y el precio en Victoria y la cantidad planeada de electricidad a transferir entre los dos estados. Estos atributos están normalizados entre 0 y 1. La clasificación es binaria entre dos categorías: el cambio a la alza (UP) o a la baja (DOWN) del precio de la electricidad en Nuevo Gales del Sur relativo a una media móvil de las últimas 24 horas (para eliminar el impacto de las tendencias de precio a más largo plazo).

Capítulo 3

Diseño e implementación

Este capítulo proporciona una visión detallada sobre la metodología y las herramientas utilizadas para medir el consumo energético y evaluar el rendimiento de modelos de aprendizaje automático en el contexto de la aplicación desarrollada: `MLCost`. Se comenzará describiendo la arquitectura de la aplicación y los módulos que la componen, para posteriormente profundizar en la metodología del proceso de aprendizaje. Este proceso empezará con la preparación de los datos, donde destacan las técnicas de preprocesamiento y la selección de modelos representativos de diversas familias algorítmicas.

Durante la fase de entrenamiento, se empleará la biblioteca CodeCarbon para medir las emisiones de carbono y el consumo de energía, utilizando técnicas como la validación cruzada para evaluar la precisión y el rendimiento de los modelos. La evaluación de las predicciones se realiza a través de métricas estándar como precisión, exhaustividad y F-score, para obtener una estimación robusta del desempeño del modelo. Para finalizar, se presentarán los resultados con distintas herramientas de visualización y se discutirá la gestión eficaz de recursos, enfocándose en la configuración del procesador y el uso de múltiples núcleos para optimizar el rendimiento y minimizar el sesgo en las mediciones de consumo energético.

3.1 Arquitectura

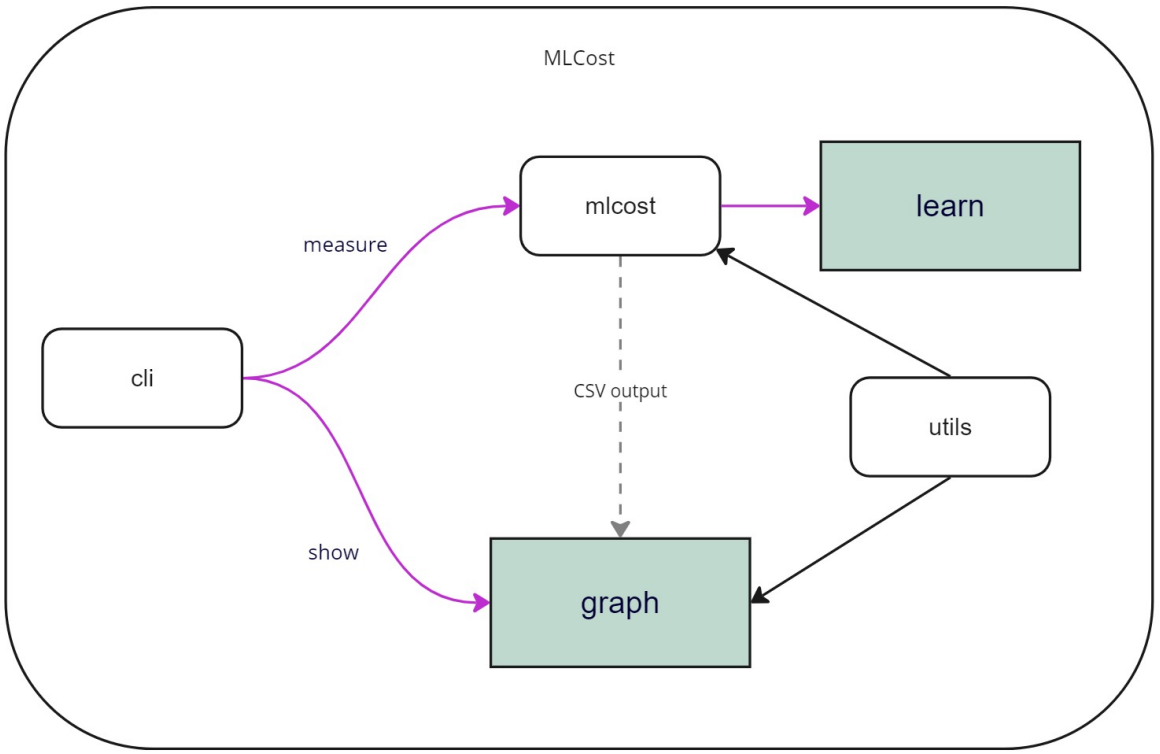


Figura 3.1: Arquitectura de la aplicación desarrollada

La arquitectura de la aplicación se organiza en torno a varios componentes clave que se integran en el paquete MLCost para ofrecer un entorno robusto para la evaluación de modelos de aprendizaje automático, tal y como muestra la figura 3.1.

El punto de entrada de la aplicación es el módulo `cli`, que se encarga de gestionar la interfaz de línea de comandos (*command-line interface*, CLI) que facilita la interacción del usuario con las funcionalidades principales de la aplicación. Este módulo permite a los usuarios ejecutar la aplicación desde la terminal, proporcionando una manera flexible y accesible de realizar diversas operaciones, como la selección de modelos de aprendizaje automático, la configuración de parámetros de ejecución y la especificación de archivos de datos de entrada. El módulo `cli` utiliza la librería `click` para facilitar el procesamiento de los argumentos de la línea de comandos. Este enfoque permite definir una variedad de opciones y argumentos que los usuarios pueden especificar al ejecutar la aplicación. Por ejemplo, los usuarios pueden seleccionar que opciones de limpieza de datos serán utilizadas, establecer el número iteraciones para la validación cruzada, y decidir si serán ejecutadas en paralelo en el procesador. El conjunto completo de opciones de línea de comandos que se pueden utilizar está recogido en el Anexo A.1. Una vez que se capturan los argumentos, `cli` invoca las funciones correspondientes definidas en el modulo `mlcost`.

Este modulo se encarga de la integración del seguimiento de emisiones de carbono y consumo energético durante el entrenamiento y la evaluación de los modelos de aprendizaje automático. Este módulo utiliza la biblioteca CodeCarbon para realizar el seguimiento del impacto ambiental de estos procesos computacionales. El módulo es responsable de gestionar el proceso de entrenamiento, creando un objeto de la clase `Trainer` por cada modelo a evaluar y asegurándose de que los datos son preprocesados para mejorar la precisión del modelo. Una vez que los datos están preparados, el módulo comienza la medición de emisiones, encarga el entrenamiento del modelo y, al finalizar, detiene el rastreador de emisiones y recupera los datos finales de consumo. Estos datos incluyen la duración del seguimiento, el consumo energético y las emisiones equivalentes de carbono. Los resultados, junto con los datos de rendimiento obtenidos durante la validación del modelo, se registran en un archivo para posterior referencia.

El núcleo de la aplicación reside en el modulo `learn`, que expone la clase `Trainer` ya mencionada, donde se definen las tareas principales de carga de datos, entrenamiento y evaluación de modelos, así como la recopilación de métricas de desempeño. Estos procesos serán descritos en mayor detalle en las secciones posteriores.

Para finalizar, la aplicación contiene dos módulos auxiliares que facilitan el desarrollo y proporcionan utilidades para examinar los resultados obtenidos.

El modulo `graphs` maneja la visualización de los resultados mediante diversas funciones de creación de gráficas que utilizan la librería matplotlib para crear gráficos de dispersión, de líneas y de barras. Estas visualizaciones ayudan a interpretar las relaciones entre las emisiones, el consumo de energía y las métricas de rendimiento de los modelos evaluados. Las gráficas también permiten comparar el desempeño de diferentes modelos bajo diversas cargas de CPU, ofreciendo una visión clara de cómo los recursos del sistema afectan la eficiencia y la precisión del modelo.

El componente `utils` proporciona funciones auxiliares para la aplicación, tales como la impresión de resultados y la recopilación de información del sistema operativo, así como la gestión de archivos de salida en formato CSV. Este archivo incluye utilidades para manejar los datos de emisiones y consumo energético, formateando la información de manera que sea fácilmente interpretable.

3.2 Lectura y limpieza de los datos

El componente más importante de todo proyecto de aprendizaje automático son sin duda los datos. Por este motivo se han desarrollado con el tiempo una gran cantidad de librerías para facilitar la tarea de los desarrolladores que quieren trabajar con información de forma estructurada. En Python una de las más importantes es la librería `pandas` [48],

que ofrece la clase `DataFrame` con la que se pueden procesar grandes cantidades de datos de forma similar a como se trabajaría con una tabla o hoja de cálculo, con filas y columnas con distintos tipos de información.

Los conjuntos de datos que están disponibles de forma libre en repositorios de universidades y otras organizaciones de ciencia de datos no siempre siguen un mismo formato. Por este motivo el primer paso para aplicar métodos de aprendizaje automático en datos específicos será siempre deshacerse del formato de presentación y guardarlos en estructuras de datos operables por un ordenador, como `DataFrames` de `pandas`. En este proyecto, la intención ha sido ser capaz de procesar datos presentados con varios formatos distintos. Para ello, se han creado distintas argumentos para la línea de comandos que pueden ser utilizados al lanzar la aplicación especificando las características concretas del conjunto de datos a tratar.

En general, el proceso de lectura y limpieza de los datos va seguir siempre las siguientes fases:

1. Lectura
   (a) Leer el archivo que contiene los datos.
   (b) Separar los datos de entrenamiento de los datos de testeo.
   (c) Identificar el tipo de datos de cada columna.
2. Limpieza
   (a) Eliminar columnas que no pueden ser utilizadas.
   (b) Reemplazar valores numéricos que falten.
   (c) Reemplazar columnas categóricas por columnas booleanas.
   (d) Escalar las características numéricas.

El primer paso es leer los datos de uno o varios archivos, que serán generalmente archivos de texto en formato `.txt` o `.csv`. Es en este paso en el que se dan mayores diferencias entre distintos conjuntos de datos y la razón de que se hayan añadido las distintas opciones de línea de comandos al programa para resolverlo. Los archivos de texto que contienen los datos pueden utilizar diferentes caracteres separadores entre columnas (como coma o espacio), representar valores que no han sido tomados con diferentes símbolos (como '?' o '-'), presentar o no una fila inicial con los nombres de las columnas, identificar la columna de las etiquetas de distintas maneras e incluso separar en distintos archivos los conjuntos de entrenamiento y de testeo de forma previa. Todas estas opciones son tenidas en cuenta durante la lectura para convertir los archivos de texto en `dataframes` sobre los que las librerías de aprendizaje pueden operar. En el Anexo A.1 se incluye un compendio de todas las opciones disponibles en la aplicación.

Una vez que los datos están recogidos, el siguiente paso es separar los datos de entrenamiento de los de testeo. Para ello, primero se descartarán todas las filas de datos que no estén etiquetadas, si las hubiera. Por defecto, la separación se hace al 80-20 y de forma aleatoria, excepto si los conjuntos están previamente separados en dos archivos. Para terminar el proceso de lectura, las columnas que contienen características numéricas se separan de las que contienen características categóricas, para poder tratarlas de forma específica durante la limpieza.

En la sección de limpieza, el objetivo es eliminar las características que puedan crear obstáculos en el entrenamiento de los modelos. Tres problemas básicos son tratados: falta de datos en columnas numéricas, datos presentados de forma categórica y diferentes escalas de los datos numéricos. Para lidiar con ello se utilizará un tipo de clases provistas por la librería `scikit-learn` denominadas transformadores [49]. Estos transformadores encapsulan distintas herramientas de preprocesado y limpieza que son usadas a menudo. Para la falta de datos numéricos, se utilizara un introductor simple de medidas (`SimpleImputer`), que rellenará los datos que falten con la media de los datos disponibles.

Respecto a las columnas categóricas, muchos algoritmos de aprendizaje no están diseñados para trabajar directamente con variables no numéricas (generalmente, porque limitaría la eficiencia de los algoritmos). Para resolverlo, se utilizará un transformador denominado `OneHotEncoder`. Este transformador reemplaza una característica categórica con varias características booleanas, de forma que para cada posible valor de la categoría se crea una nueva columna con un valor de sí o no dependiendo de a cual pertenece cada dato. Para simplificar, características categóricas con más de diez valores distintos posibles serán descartadas completamente. Lo mismo ocurrirá con cualquier otra columna que no pueda ser identificada como numérica o categórica y con las filas en las que falten datos de tipo categórico.

Un último transformador, `StandardScaler`, será aplicado a las características numéricas para alinear la escala de todas ellas mediante una técnica denominada normalización de características. Este proceso consiste en transformar las características numéricas para que se sitúen dentro de un rango común, generalmente entre 0 y 1 o para que tengan media cero y varianza uno, como se hace con el `StandardScaler`. Esto ayuda a obtener mejores resultados en los modelos de aprendizaje automático por dos razones. Primero, evita que características con valores grandes dominen a aquellas con valores más pequeños, asegurando que todas las características contribuyan de manera equitativa al modelo. Segundo, algunos algoritmos, como los basados en distancias (por ejemplo, vecinos más cercanos o máquinas de vector soporte), funcionan mejor y convergen más rápido cuando las características están en la misma escala.

3.3 Entrenamiento

Una vez que los datos están preparados para su uso comienza la fase de entrenamiento. En este proyecto, el objetivo es medir el gasto energético de distintos modelos y compararlo con la calidad de sus predicciones. Para ello, se han elegido una serie de modelos representativos de las familias de algoritmos más utilizadas. Estos modelos elegidos serán entrenados uno detrás de otro con el conjunto de datos preparado mientras se mide el consumo energético mediante las herramientas proporcionadas por CodeCarbon.

El procedimiento es sencillo. En primer lugar se comienzan las mediciones mediante la creación se un objeto de tipo `EmissionsTracker` que cuenta con simples métodos `start()` y `stop()`. A continuación, se entrena el modelo en la parte del conjunto de datos reservada para entrenamiento, y seguidamente se aplica el modelo entrenado a la parte del conjunto de datos reservada para testeo para intentar predecir correctamente la etiqueta de cada entrada que contiene. Para finalizar, se detiene la medición de emisiones y se almacenan los resultados obtenidos.

3.3.1 Modelos escogidos

Dentro de los modelos de aprendizaje automático existen numerosas clasificaciones de acuerdo al tipo de tareas a realizar y la naturaleza de los datos. Este proyecto se centrará en tareas de clasificación por aprendizaje supervisado. En este subgrupo, destacan una serie de familias de algoritmos que suelen obtener buenos resultados para una gran variedad de tipos de datos.

1. Modelos lineales. Se trata de modelos sencillos, fáciles de interpretar y eficientes computacionalmente, que funcionan bien cuando las relaciones entre las características de entrada y la salida son aproximadamente lineales. Uno de sus modelos más representativos es el de regresión logística (ver 2.4.1).

2. Árboles decisores. Estos algoritmos pueden modelar relaciones complejas en los datos y lidiar con no linealidad. Dentro de esta familia destaca el modelo de Bosque Aleatorio, que agrega predicciones de varios árboles decisores para mejorar la robustez del modelo (ver 2.4.2).

3. Máquinas de vector soporte (2.4.3). Estos modelos son efectivos tanto en tareas de clasificación lineales como no lineales y destacan por su gran versatilidad. Funcionan de forma óptima en conjuntos de datos relativamente pequeños pero de gran complejidad.

4. Vecinos más cercanos. Su máximo representante, k vecinos más cercanos (k-NN, ver 2.4.4), es un algoritmo simple e intuitivo que se basa en buscar relaciones locales entre los datos y puede ser efectivo en tareas tanto de regresión como de clasificación.

5. Naive Bayes (bayesiano ingenuo). Se trata de una familia de modelos que destaca por su gran eficiencia, especialmente frente a conjunto de datos de gran complejidad, y especialmente útil en tareas de clasificación de texto. El clasificador más representativo es el Naive Bayes gaussiano (2.4.5).

6. Métodos de conjuntos (ensemble). Estos métodos combinan las características de múltiples modelos para intentar mejorar el desempeño total. Entre ellos destacan las máquinas de potenciación de gradiente, que construyen sólidos modelos predictivos de forma iterativa (ver 2.4.6).

7. Redes neuronales. Las redes neuronales, especialmente los modelos de aprendizaje profundo como las redes neuronales profundas (Deep Neural Networks, DNN, ver 2.4.7), pueden formar representaciones jerárquicas complejas a partir de los datos, y son especialmente efectivas en tareas que involucran grandes cantidades de datos con patrones complejos.

En general, se espera que modelos de aprendizaje más complejos como los métodos de conjuntos, las redes neuronales y las máquinas de vector soporte produzcan mejores predicciones a cambio de un mayor gasto energético que otros modelos comparativamente más sencillos computacionalmente, como los modelos lineales, Naive Bayes y vecinos más cercanos. En el capitulo 4, se analizará si los resultados obtenidos durante los experimentos se corresponden con esta aproximación teórica.

3.4 Registro de resultados

3.4.1 Evaluación de las predicciones

Para poder obtener una medida de utilidad de los distintos modelos, es necesario evaluar la calidad de las predicciones que realizan. Existen dos métodos principales realizar esta evaluación. El primero y más sencillo consiste en aplicar la función de predicción del modelo a un subconjunto de muestras que hayan sido aisladas previamente para no formar parte del proceso de entrenamiento. Estas predicciones se comparan con las etiquetas correctas de las muestras para determinar si cada predicción ha sido acertada. Cuatro resultados distintos son posibles por muestra y clase concreta: verdadero positivo (TP, identificada correctamente como perteneciente a la clase), falso positivo (FP, identificada incorrectamente como perteneciente a la clase), falso negativo (FN, identificada incorrectamente como no perteneciente a la clase), y verdadero negativo

(TN, identificada correctamente como no perteneciente a la clase). Una forma común de visualizar estos resultados es mediante una matriz de confusión, como la mostrada en la figura 3.2.

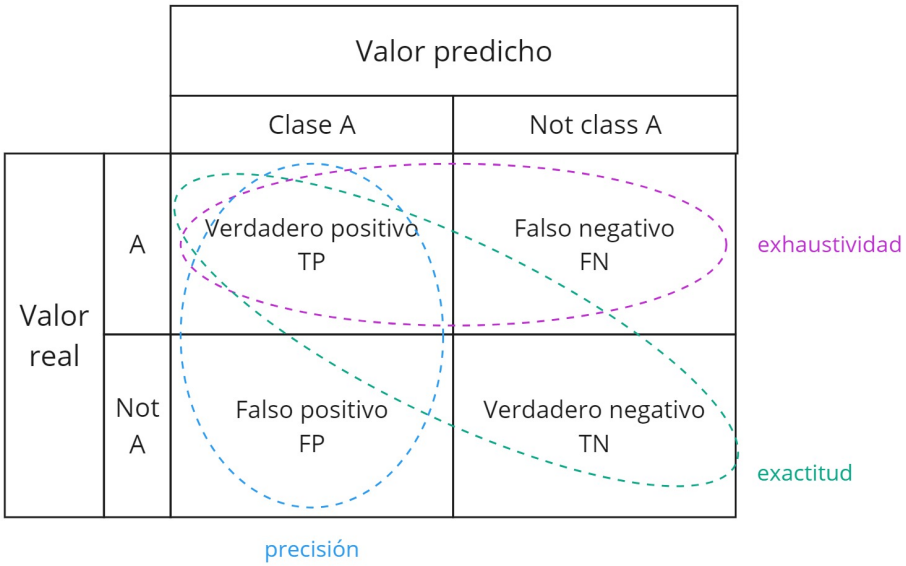


Figura 3.2: Matriz de confusión que muestra las relaciones entre posible resultados de la predicción.

A partir de las relaciones entre el número de muestras en cada uno de estos grupos se pueden extraer varias métricas del modelo, siendo las más comunes la exactitud, la precisión, la exhaustividad y el *F-score*[50]. Estas métricas dan lugar a un valor entre 0 y 1, donde 0 es el peor resultado y 1 el mejor. También son comúnmente expresadas en porcentaje.

La **exactitud** (*accuracy*) se define como la cercanía de la predicciones a su valor real. En tareas de clasificación se calcula como el número de predicciones correctas entre el número de muestras totales, como se puede ver en la ecuación 3.1. Una variante interesante de la exactitud que `scikit-learn` permite calcular es la exactitud balanceada, que evita medidas infladas de exactitud en conjuntos de datos no balanceados (con una o más clases sobrerrepresentadas en el conjunto) mediante la ponderación de cada muestra de acuerdo a la prevalencia inversa de su verdadera clase.

$$a = \frac{TP + TN}{\text{muestras totales}} \tag{3.1}$$

La **precisión** (*precision*) y la **exhaustividad** (*recall*) son métricas que analizan la relevancia de las muestras asignadas a cada clase y se calculan individualmente por clase. La precisión analiza el número de muestras correctamente clasificadas dentro de todas

las muestras asignadas a una clase concreta, como muestra la ecuación 3.2, mientras que la exhaustividad analiza el número de muestras correctamente clasificadas en relación al número total de muestras reales existentes, como muestra la ecuación 3.3. Estas dos métricas pueden ser promediadas en función del peso relativo de cada clase un el conjunto para obtener una medida global de la precisión y exhaustividad del modelo.

$$p = \frac{TP}{TP + FP} \tag{3.2}$$

$$r = \frac{TP}{TP + FN} \tag{3.3}$$

En último lugar, es interesante mencionar el valor-F (*F-score*), que se puede interpretar como una media harmónica de la precisión y la exhaustividad y tiene distintas variantes dependiendo de la importancia relativa de estas dos medidas. La denominada medida-F_1 da la misma importancia a la precisión y a la exhaustividad. Para cada clase, se calcula como muestra la ecuación 3.4.

$$F_1 = \frac{2 \cdot TP}{2TP + FP + FN} = \frac{2pr}{p + r} \tag{3.4}$$

Estas cuatro medidas son calculadas por defecto en la aplicación desarrollada para todos los modelos entrenados. Para todos los casos, se utiliza la opción de scikit-learn `average='weighted'` para obtener las medidas, de forma que se puedan obtener valores más realistas en conjuntos con clases no balanceadas.

Validación cruzada

La aplicación desarrollada ofrece un segundo método para evaluar el rendimiento de un modelo mediante validación cruzada [51, 52]. Este método se basa en las mismas métricas ya mencionadas, pero con la peculiaridad de que éste se entrena varias veces de forma sucesiva con distintas distribuciones de los datos en un conjunto de entrenamiento y un conjunto de prueba. Posteriormente, se puede analizar la media y la desviación estándar de las métricas de la calidad de las predicciones obtenidas para cada distribución de las muestras y así obtener una idea más exacta del desempeño del modelo. En `scikit-learn`, la validación cruzada se puede implementar mediante el uso de la clase `KFold`, que divide el conjunto de datos en un número de pliegues especificados de forma aleatoria, y la función `cross_validate`, que entrena el modelo y calcula las métricas deseadas para cada uno de los pliegues de forma sucesiva. Para determinar los plieges, la aplicación MLCost utiliza una clase derivada llamada `StratifiedKFold`, que ayuda a mantener el mismo ratio de muestras por clase en cada pliegue para conjuntos de datos no balanceados.

La validación cruzada ofrece varias ventajas significativas al evaluar el rendimiento de un modelo de aprendizaje automático. Una de las principales ventajas es que proporciona una estimación más robusta y fiable del desempeño del modelo al utilizar diferentes subconjuntos del conjunto de datos para entrenamiento y prueba en cada iteración. Esto ayuda a mitigar el riesgo de sobreajuste y ofrece una visión más generalizada de cómo se comportará el modelo con datos no vistos. Además, calcular la media y la desviación estándar de las métricas a través de los distintos pliegues permite una evaluación más precisa y detallada, identificando variaciones y asegurando que el modelo no solo es preciso sino también consistente.

Sin embargo, la validación cruzada también tiene desventajas. Uno de los principales inconvenientes es el aumento significativo en el tiempo de cómputo, ya que el modelo debe entrenarse y evaluarse múltiples veces, lo cual puede ser particularmente costoso en términos de tiempo y consumo energético, especialmente para conjuntos de datos grandes o modelos complejos. La utilización de un entorno de paralelización puede mitigar esta desventaja al permitir que los pliegues se procesen en paralelo, reduciendo así el tiempo total necesario para completar la validación cruzada. No obstante, la paralelización puede no ser igualmente efectiva en todos los tipos de hardware. Además, en entornos compartidos o con recursos limitados, la paralelización podría causar conflictos de recursos, afectando negativamente el rendimiento global del sistema. Por lo tanto, aunque la paralelización puede acelerar significativamente la validación cruzada, es crucial evaluar su uso en conjunción con las capacidades y limitaciones del hardware disponible. Este efecto será examinado durante las pruebas llevadas a cabo en la sección 4.3.

3.4.2 Medición de emisiones

Durante la ejecución de su rastreador de emisiones, la librería CodeCarbon calcula varias medidas distintas para identificar el consumo energético. En primer lugar, las emisiones están geolocalizadas de una de las siguientes formas: mediante una conexión a internet automática que permita identificar la localización mediante rastreo de IP, o mediante la especificación de un país determinado en el código al crear el objeto rastreador de emisiones. Esta localización es necesaria para convertir los kilovatios consumidos durante el proceso de entrenamiento en emisiones de carbono equivalentes, que dependerán de la mezcla especifica de producción de energía que haya establecido cada país. De esta forma, si la energía estuviera producida en gran medida por energías renovables, las emisiones de carbono serían mucho menores que si la energía fuera producida en su totalidad en una planta de quema de carbón. En la sección de resultados se compararán las diferencias de emisiones producidas entrenando modelos en un mismo conjunto de datos en diferentes localizaciones.

Con estos datos, la librería CodeCarbon calcula las emisiones a partir de medidas de la capacidad del procesador y gráfica de la máquina, del porcentaje de su uso que

corresponde al proceso de entrenamiento observado y de la duración total del proceso. En este proyecto, por cada modelo entrenado se guardan tres de estas medidas para su posterior análisis y comparativa: la energía consumida (en kilovatios hora, kWh), las emisiones calculadas (en kilogramos equivalentes de carbono, $\mathrm{kg}\ [\mathrm{CO_2eq}]$) y la duración (en segundos). Durante el proceso de entrenamiento, estos valores son escritos en un archivo de texto de tipo CSV (*comma-separated values*) junto con las medidas de calificación de las predicciones mencionadas en el apartado anterior. Este archivo de texto será utilizado posteriormente para dibujar gráficas de las que extraer conclusiones con una herramienta de gráficos desarrollada para este proyecto.

3.4.3 Herramientas de visualización

La visualización de los resultados obtenidos es crucial para interpretar y analizar las métricas de consumo energético y rendimiento de los modelos de aprendizaje automático evaluados. A través de gráficos, es posible identificar patrones, tendencias y relaciones entre distintas variables que de otra manera serían difíciles de detectar en tablas de datos. Esto facilita la toma de decisiones informadas y permite comunicar los hallazgos de manera más efectiva.

El motor de gráficos utilizado en este proyecto es Matplotlib (ver 2.3.4, una biblioteca de Python ampliamente utilizada para la creación de gráficos estáticos, animados e interactivos. Matplotlib proporciona una gran flexibilidad y control sobre la generación de gráficos, permitiendo a los desarrolladores personalizar todos los aspectos visuales de sus representaciones gráficas.

Para generar los gráficos, es necesario haber ejecutado previamente una medición de emisiones con la opción `-log` activada. Esto crea un archivo CSV que contiene los datos recopilados durante las pruebas, incluyendo métricas de rendimiento y consumo energético. Este archivo CSV es esencial para la visualización, ya que contiene toda la información requerida para producir los gráficos. Los gráficos se generan utilizando el comando `mlcost plot -f <output-file.csv>`. Este comando lee el archivo CSV especificado y produce una variedad de gráficos que ayudan a visualizar los resultados de las pruebas.

El módulo `graph` incluye varias funciones auxiliares para la creación de diferentes tipos de gráficos. Entre ellas se encuentran funciones para generar gráficos de dispersión con tres o cuatro variables, así como gráficos de líneas y barras. Las funciones de dispersión permiten comparar variables como emisiones y precisión a través de diferentes modelos y conjuntos de datos, utilizando diferentes marcadores y colores para distinguir entre categorías. Por otro lado, los gráficos de líneas y barras se utilizan para visualizar tendencias y comparaciones categóricas, aprovechando las capacidades de la librería

Pandas para agrupar los datos recogidos en `DataFrames` y producir gráficas con ellos de manera eficiente.

3.5 Gestión de recursos

Un aspecto importante para obtener mediciones precisas del consumo energético y del rendimiento de los modelos de aprendizaje automático es la gestión de los recursos disponibles en la máquina que está tomando las medidas. Los resultados obtenidos por la librería CodeCarbon para estimar el consumo eléctrico se basan en el modelo y el tiempo de utilización del procesador. De esta forma, CodeCarbon rastrea el uso de recursos durante el entrenamiento de los modelos y calcula las emisiones de CO_2 correspondientes.

La carga del procesador en el momento de tomar las medidas de consumo energético es un factor crucial. Una alta carga del procesador puede indicar que el sistema está ejecutando múltiples tareas simultáneamente, lo cual puede sesgar los resultados. Para mitigar este efecto, CodeCarbon intenta aislar el proceso de aprendizaje del resto de tareas del ordenador al utilizar la opción `tracking_mode="process"` en su clase medidora de emisiones `EmissionTracker`. Esta opción permite que la herramienta enfoque la toma de medidas en el proceso específico que se está evaluando, minimizando la interferencia de otras operaciones del sistema.

Sin embargo, este enfoque añade una capa adicional de aproximación al cálculo de emisiones, por lo que para obtener medidas más consistentes será interesante mantener una carga baja del procesador al comenzar el proceso. Al inicio de la aplicación, ésta imprime al terminal tanto la información del sistema y del modelo del procesador como su carga de trabajo inicial en porcentaje de utilización. Además, el archivo de datos recopilados generado por el programa contiene una columna con la carga al finalizar el entrenamiento de cada modelo. Una carga baja al comienzo del experimento significará que la máquina no está realizando otras tareas pesadas que podrían influir en las mediciones de consumo energético y rendimiento. Esto ayudará a que los datos reflejen de manera más precisa el impacto del entrenamiento del modelo en el consumo energético.

3.5.1 Procesamiento multinúcleo

El uso de ordenadores con varios núcleos en el procesador o de varios procesadores asignados a la misma tarea puede afectar tanto al rendimiento como a las emisiones. En términos de rendimiento, la capacidad de realizar múltiples tareas simultáneamente (*multitasking*) permite que los modelos se entrenen más rápido, ya que pueden aprovechar el paralelismo inherente a muchos de los algoritmos de aprendizaje automático. Sin

embargo, esta mejora en el rendimiento puede venir acompañada de un aumento en el consumo energético, ya que más núcleos en uso implican un mayor consumo de energía.

La biblioteca scikit-learn gestiona el paralelismo a través del parámetro `n_jobs`, el cual se puede especificar en diversas funciones y modelos para indicar el número de procesos paralelos que se deben utilizar. La aplicación desarrollada acepta este parámetro al ejecutarla por línea de comandos y lo comunica a scikit-learn, permitiendo que se configure la cantidad de CPUs disponibles para ejecutar tareas en paralelo. Esta configuración puede reducir significativamente el tiempo de cómputo en experimentos de gran escala.

Internamente, scikit-learn utiliza el contexto joblib.parallel_backend para gestionar el paralelismo. joblib.parallel_backend es una función que permite seleccionar el backend de paralelización que se utilizará durante la ejecución de las operaciones paralelas. El backend de joblib puede manejar diferentes tipos de paralelismo, como threading y multiprocessing, adaptándose a las características del hardware y a las necesidades del usuario. Cuando se especifica `n_jobs`, joblib.parallel_backend se encarga de dividir las tareas entre los procesos disponibles y de gestionar la sincronización y la recolección de resultados.

Dentro del proceso de aprendizaje, hay varias funciones en las que scikit-learn ofrece la posibilidad de ejecutar tareas en paralelo. Una de ellas es la validación cruzada, que es particularmente adecuada para la paralelización, ya que cada pliegue del conjunto de datos puede ser procesado de manera independiente. Por otra parte, varios modelos pueden ser entrenados directamente en paralelo dentro de una única iteración especificando el número de procesos a utilizar. Entre los modelos que admiten `n_jobs` se encuentran el bosque aleatorio, las máquinas de potenciación de gradiente, y vecinos más cercanos. Sin embargo, otros modelos como los lineales, las máquinas de vector soporte, Naive Bayes, y las redes neuronales no siempre permiten especificar un número de procesos en paralelo de forma nativa.

Para lidiar con estas diferencias, la aplicación desarrollada se limita a aplicar el número de procesos únicamente a la validación cruzada y no al entrenamiento directo de los modelos. Esta decisión se toma para poder comparar todos los modelos en igualdad de condiciones. De esta forma, se asegura que cualquier mejora en el tiempo de ejecución sea atribuible únicamente a la paralelización del proceso de validación cruzada y no a diferencias intrínsecas en la implementación de cada modelo.

El efecto que la paralelización y la gestión de los recursos de la máquina donde se realiza el entrenamiento se podrá observar en el experimento de la sección 4.3. En esta prueba, se entrenarán los modelos en diferentes máquinas virtuales desplegadas en Microsoft Azure, cada una con configuraciones de procesador y memoria RAM distintas y alternando entre utilizar paralelización o no durante el entrenamiento. Este experimento permitirá estudiar los resultados de emplear varios procesadores en el entrenamiento por validación cruzada, analizando cómo el paralelismo y la configuración de hardware afectan tanto al rendimiento de los modelos como al consumo energético.

Capítulo 4

Experimentos y validación

El objetivo de este capítulo es mostrar el funcionamiento de la aplicación en un caso de uso real en el que se tratará de extraer conclusiones generales acerca del consumo eléctrico de cada modelo y de si este consumo irá necesariamente acompañado de una mejora de los resultados de predicción. Para ello se emplearán las herramientas descritas anteriormente para evaluar el consumo y el rendimiento de una serie de modelos formada por representantes de las principales familias de modelos de aprendizaje automático y recogidos en la sección 2.4. Estos modelos serán aplicados a los conjuntos de datos de distintas características definidos en la sección 2.5.

4.1 Estructura de los experimentos

Durante la validación de la aplicación se llevarán acabo dos experimentos distintos. El primero examinará el consumo energético en base al modelo seleccionado. En esta sección se tomarán varias medidas de consumo y rendimiento por modelo y conjunto de datos en una máquina con unos recursos de procesamiento concretos para analizar que modelos consumen más que otros y que características de los conjuntos de datos hacen incrementar este consumo. El segundo experimento seleccionará un único conjuntos de datos de tamaño mediano y tomará medidas de consumo en tres entornos con distintos recursos de procesamiento dedicados a la tarea de aprendizaje automático para observar el efecto de los recursos disponibles en el consumo energético de cada modelo.

A través de este análisis, se pretende obtener una comprensión profunda de cómo diferentes modelos de aprendizaje automático consumen energía bajo diversas condiciones de trabajo. Además se intentarán identificar patrones de menor consumo que puedan utilizarse como base para el diseño y la implementación de modelos más sostenibles y eficientes en el futuro.

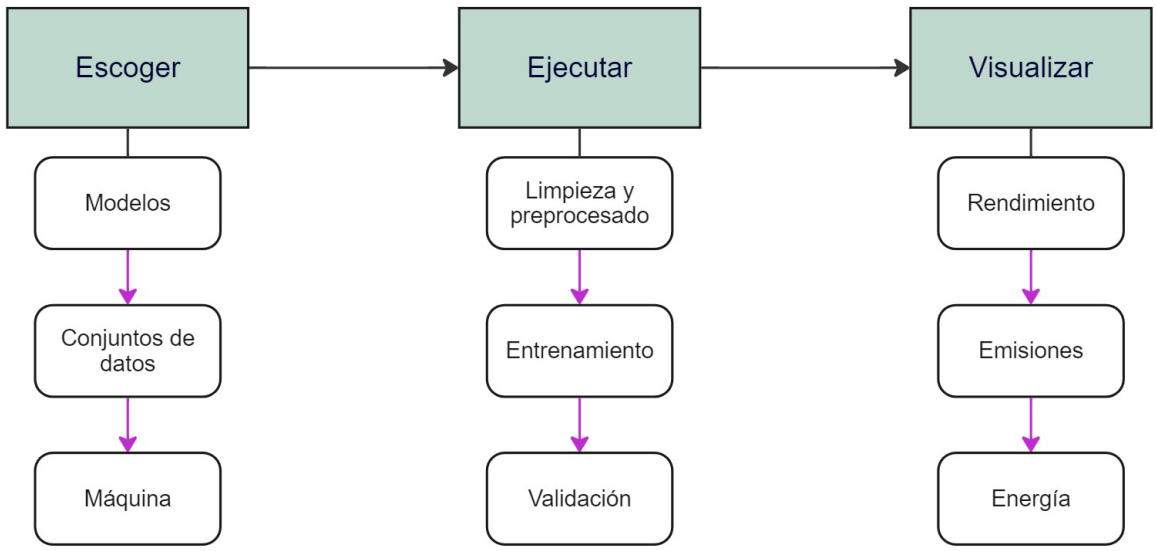


Figura 4.1: Esquema general a seguir durante los experimentos

La figura 4.1 muestra un esquema general que seguirán ambos experimentos. En primer lugar, se plantearán los modelos a analizar, los conjuntos de datos en los que serán entrenados y los recursos la máquina en la que se realizará el entrenamiento. Posteriormente, se utilizará la función "`mlcost measure`" de la aplicación, especificando los modelos y datos escogidos. De esta manera se llevarán a cabo las tres fases del proceso de medición por cada combinación modelo-conjunto posible: limpieza de los datos, entrenamiento del modelo y validación de la calidad de las predicciones. Para finalizar, se recogerán los datos resultantes del paso anterior y se alimentarán a la función "`mlcost plot`", que producirá las gráficas necesarias para analizar las relaciones entre el rendimiento de los modelos y su consumo energético y emisiones producidas.

4.2 Consumo energético en función del modelo seleccionado

Objetivos

En esta sección se examinará el consumo energético una serie de modelos representativos aplicados a varios conjuntos de datos. El objetivo de este análisis será abordar las siguientes cuestiones clave:

- Identificar los modelos con mayor consumo energético.
- Determinar los modelos cuyo consumo energético incrementa significativamente al aumentar el número de muestras.
- Evaluar que modelos ofrecen mejores predicciones con menor consumo energético.

Dónde sea posible, se tratará de analizar estas cuestiones de forma general y obtener conclusiones que sean extrapolables más allá de los conjuntos de datos concretos que se hayan medido. Sin embargo, debido a la gran cantidad de variables involucradas y a las variaciones de consumo presentes entre unos casos y otros, es posible en otros conjuntos de datos muy diferentes de los considerados se observen distintos patrones de consumo.

Metodología

Para analizar estas cuestiones todas las medidas de consumo serán tomadas con la aplicación desarrollada ejecutando en una misma máquina. Las características de la máquina utilizada están recogidas en la tabla 4.1.

Tabla 4.1: Características técnicas de la máquina utilizada para tomar las medidas

Característica	**Descripción**
Modelo	Dell XPS 15 9500
Sistema Operativo	Ubuntu 20.04.6 LTS x86_64
Procesador	Intel(R) Core(TM) i9-10885H CPU @ 2.40GHz
Versión Python	3.12.2
Memoria	7,63 GB

Para cada modelo y conjunto de datos, se tomarán medidas de consumo y rendimiento utilizando validación cruzada con cinco iteraciones con un tamaño definido para los datos de testeo del 20% del conjunto de datos. Esta técnica proporcionará una evaluación robusta y precisa tanto del comportamiento energético de los modelos como de su precisión y exactitud, ya que evitará en gran medida la presencia de valores atípicos y el riesgo de sobreajuste de los modelos. La aplicación será ejecutada con el siguiente comando para cada conjunto de datos distinto, en el cual `[dataset]` será sustituido por el archivo que contenga cada conjunto de datos. Adicionalmente, cualquiera de las opciones de lectura de datos descritas en la sección 3.2 podrá ser utilizada si el formato en el que se encuentren los datos lo requiere.

```
mlcost measure --log -cv 5 -d [dataset] [dataset-options]
```

Al ejecutar este comando, el programa escribirá al terminal en primer lugar una descripción de las características técnicas de la máquina donde se realiza el entrenamiento, con la información mostrada en la tabla 4.1. A continuación, se imprimirá un resumen del preprocesamiento realizado en el conjunto de datos elegido, incluyendo las características que se usarán en el entrenamiento, el número de muestras en los conjuntos de entrenamiento y de prueba y la distribución de las clases permitidas.

Para obtener los resultados buscados, se deberá ejecutar el comando mencionado para todos los conjuntos a analizar. A continuación, se enumeran las opciones exactas utilizadas para tratar de forma adecuada cada uno de ellos. La salida del terminal con todos los resultados de la limpieza y preprocesado se encuentran recogida en el apéndice A.2.

```
# Iris (conjunto por defecto)
mlcost measure --log -cv 5
# Ionosfera
mlcost measure --log -cv 5 -d data/ionosphere/ionosphere.data --no-header
# Autenticación de billetes
mlcost measure -d data/banknote/banknote.txt --no-header
# Fonemas
mlcost measure --log -cv 5 -openml -d phoneme
# Electroencefalograma
mlcost measure --log -cv 5 -openml -d eeg-eye-state
# Electricidad
mlcost measure --log -cv 5 -openml -d electricity
```

Resultados

La ejecución de los comando anterior producirá un archivo tipo tabla de datos en formato CSV. La tabla 4.2 recoge un extracto de los resultados obtenidos en el ordenador de referencia para seis conjuntos de datos distintos. El archivo completo está disponible en el repositorio de la aplicación[1].

Tabla 4.2: Extracto de los resultados de entrenamiento.

Dataset	Modelo	CPU load (%)	Accuracy	Precision	F-score	Recall	Fit time (s)	Total (s)	Emisiones (kg)	Energía (kWh)	N
Banknote	Linear	2.7	0.98	0.98	0.98	0.98	0.007	0.071	2.13E-07	1.10E-06	1372
Banknote	Linear	2.7	0.97	0.97	0.97	0.97	0.006	0.071	2.13E-07	1.10E-06	1372
Banknote	Linear	2.7	0.97	0.97	0.97	0.97	0.006	0.071	2.13E-07	1.10E-06	1372
Banknote	Linear	2.7	0.99	0.99	0.99	0.99	0.005	0.071	2.13E-07	1.10E-06	1372
Banknote	Linear	2.7	0.99	0.99	0.99	0.99	0.005	0.071	2.13E-07	1.10E-06	1372
Banknote	Forest	2.7	0.99	0.99	0.99	0.99	0.184	1.429	3.27E-06	1.69E-05	1372
Banknote	Forest	2.7	1.00	1.00	1.00	1.00	0.171	1.429	3.27E-06	1.69E-05	1372
Banknote	Forest	2.7	0.99	0.99	0.99	0.99	0.154	1.429	3.27E-06	1.69E-05	1372
Banknote	Forest	2.7	1.00	1.00	1.00	1.00	0.172	1.429	3.27E-06	1.69E-05	1372
Banknote	Forest	2.7	1.00	1.00	1.00	1.00	0.158	1.429	3.27E-06	1.69E-05	1372
					...						
Electricity	Neural	102.4	0.82	0.83	0.83	0.83	132.768	518.905	1.19E-03	6.15E-03	45312

Cada fila en la tabla corresponde a las medidas tomadas durante una iteración de entrenamiento de cada modelo por validación cruzada. Al haber escogido utilizar validación cruzada de cinco iteraciones, la tabla de resultados cuenta con cinco filas por modelo y conjunto de datos. Sin embargo, algunas de las medidas, como la carga del procesador, el número de muestras del conjunto de datos, el tiempo total de entrenamiento, las emisiones del proceso y la energía consumida, son tomadas de forma global al finalizar todas las iteraciones de entrenamiento de cada modelo y conjunto. Para cada iteración individual se recogen las medidas estadísticas de exactitud, precisión, exhaustividad y valor-F calculadas. Además, la implementación de validación cruzada de `scikit-learn` proporciona una medida del tiempo de entrenamiento empleado en cada iteración (fit time). Este valor puede ser utilizado junto con las emisiones y el tiempo totales de todas las iteraciones para calcular las emisiones de cada iteración de entrenamiento, como muestra la ecuación 4.1.

$$E_1 = \frac{E_T}{t_T} \cdot t_1 \ , \tag{4.1}$$

[1] https://github.com/l-gonz/tfg-gitt-mlcost/blob/main/out/all-models-laptop.csv

donde: E_1 = emisiones de una iteración (kg $[CO_2eq]$),
E_T = emisiones totales (kg $[CO_2eq]$),
t_T = tiempo total (s),
t_1 = tiempo de entrenamiento de la iteración (s).

A partir de los resultados obtenidos se puede dibujar un diagrama de dispersión para visualizar cómo varían las emisiones. En la figura 4.2 se ha utilizado el valor-F como medida de la calidad de las predicciones (eje Y), ya que es habitualmente más informativa en casos de distribuciones de clases no balanceadas. En el eje X, se han dibujado las emisiones en kilogramos de carbono equivalentes con una escala logarítmica. Los distintos conjuntos de datos analizados se indican con formas geométricas distintas de los marcadores y cada modelo se indica en un color distinto. Los conjuntos están ordenados en la leyenda de la parte inferior de la gráfica por número de muestras, de menor a mayor tamaño.

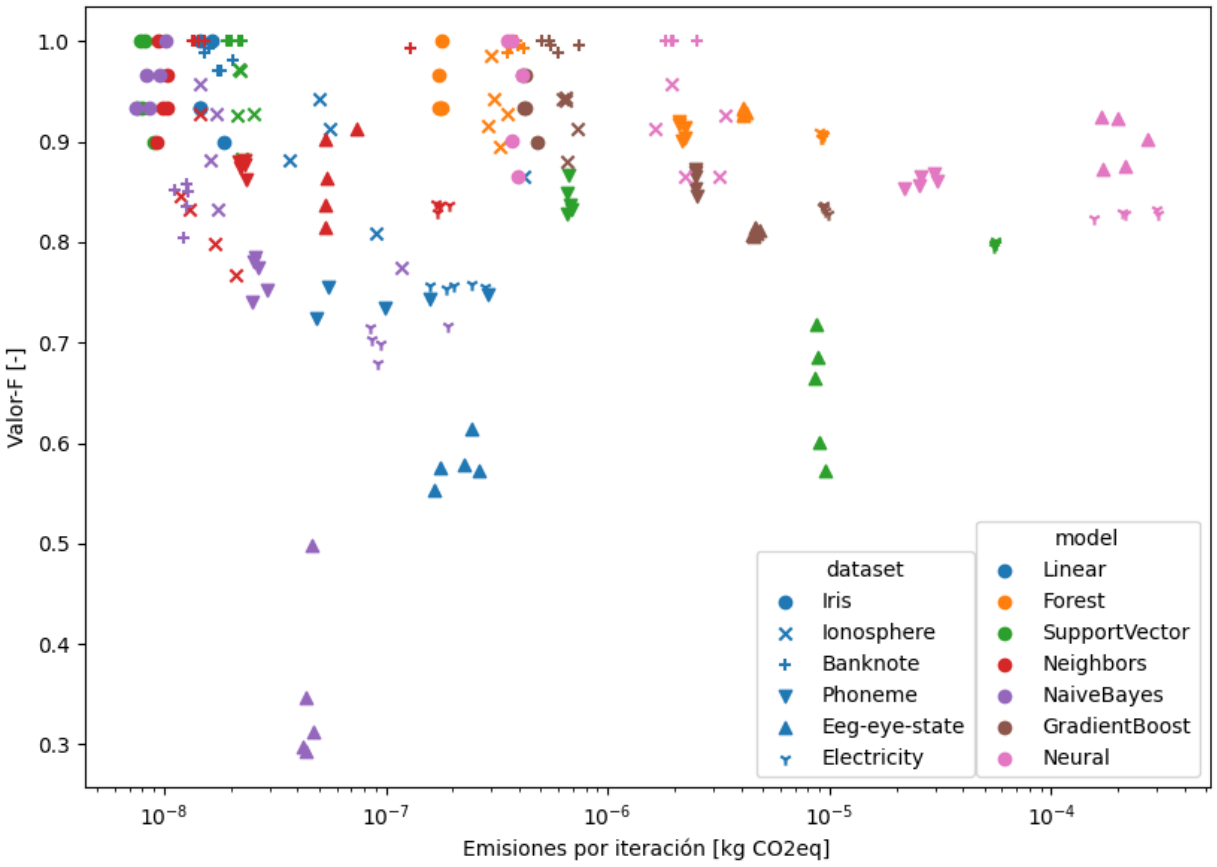


Figura 4.2: Valor-F alcanzado por el modelo frente a las emisiones de carbono necesarias para entrenarlo, por modelo empleado y conjunto de datos utilizado.

Este tipo de representación de los resultados permite observar las relaciones entre cuatro variables al mismo tiempo. De esta forma, se puede observar como cambia la relación entre la puntuación del modelo y sus emisiones de carbono para todas las distintas combinaciones de conjunto de datos y modelo de aprendizaje. Analizando esta gráfica, se puede obtener una idea de que modelos consiguen mantener una buena puntuación con un consumo bajo en la mayoría de las ocasiones.

En primer lugar, cabe destacar que la mayoría de las predicciones tienen una puntuación que se encuentra por encima del 70%. Sin embargo, en la mitad inferior de la gráfica 4.2 destacan los resultados obtenidos para el conjunto del electroencefalograma (*eeg-eye-state*). Este conjunto presenta un valor-F que llega a estar incluso por debajo del 50% al utilizar el modelo Naive Bayes, lo que representa un resultado peor incluso que un predictor que simplemente eligiera la clase de cada muestra de forma aleatoria entre los dos valores posibles. Para este conjunto, también se obtienen resultados bastante deficientes con el modelo lineal y el de vector soporte, que se encuentran entre el 50 y el 70%. Estos resultados podrían indicar un problema en la fase de limpieza y preprocesamiento de las características, que no actúa de manera adecuada para un caso específico que presente este conjunto de datos. En la gráfica se puede apreciar que modelos que suelen ser en general menos sensibles al utilizarse con conjuntos poco preparados, como el bosque aleatorio o las redes neuronales, consiguen obtener resultados más aceptables para el conjunto del electroencefalograma.

Para el resto de resultados que se encuentran en el rango superior del valor-F, se observa en general que las mayores variaciones vienen dadas por el modelo empleado, de forma que los resultados para un modelo particular para todos los conjuntos se encuentran ligeramente agrupados en una sección de la gráfica, con las emisiones aumentando a medida que se incrementa el tamaño del conjunto de datos. Sin embargo, para el caso específico del modelo de vector soporte, se observa una variación mucho más extrema de las emisiones con el aumento de tamaño del conjunto utilizado, a la vez que la puntuación obtenida en las predicciones disminuye considerablemente en comparación con los resultados obtenidos para los conjuntos más pequeños, que dominan la parte superior izquierda de la gráfica.

Otro caso de emisiones especialmente elevadas se produce al usar el modelo de redes neuronales que destaca por conseguir puntuaciones de valor-F bastante elevadas para todos los conjuntos de datos, incluso para los datos del electroencefalograma, pero a costa de una utilización de recursos que aumenta exponencialmente y produce las emisiones más elevadas de todos los modelos para los conjuntos más grandes y complejos.

Los últimos modelos fácilmente descartables en la mayoría de los casos son el modelo lineal y Naive Bayes, que presentan unos resultados muy similares. Estos modelos consigue mantener unas emisiones particularmente bajas para todos los conjuntos analizados. Sin embargo, presentan una tendencia claramente a la baja en el valor-F al aumentar

la complejidad del conjunto de datos utilizado, incluso sin tener en cuenta los valores anómalos obtenidos en el conjunto del electroencefalograma.

De entre los modelos restantes, se distinguen claramente dos patrones distintos. Por un lado el modelo de vecinos más cercanos (*neighbors*) sigue un patrón de emisiones similar al caso lineal, que comienza en la parte izquierda de la gráfica y consigue incrementar muy poco en emisiones al aumentar la complejidad del conjunto de datos. Sin embargo, a diferencia del caso lineal, el valor-F no se muestra tan afectado por el aumento del tamaño del conjunto y se mantiene entre el 80 y el 90%.

El último caso se da con los modelos de bosque aleatorio y de potenciación de gradiente. Estos modelos se encuentran en la parte media de la gráfica en cuanto a emisiones, con resultados muy parecidos entre los dos modelos. Sin embargo, su puntuación de valor-F se mantiene particularmente elevada para todos los conjuntos de datos, sin muchas variaciones. Dentro de estos dos modelos, el de bosque aleatorio obtiene los mejores resultados, consiguiendo emisiones más reducidas para conjuntos de pequeño tamaño y manteniendo una mayor puntuación para los conjuntos de mayor tamaño.

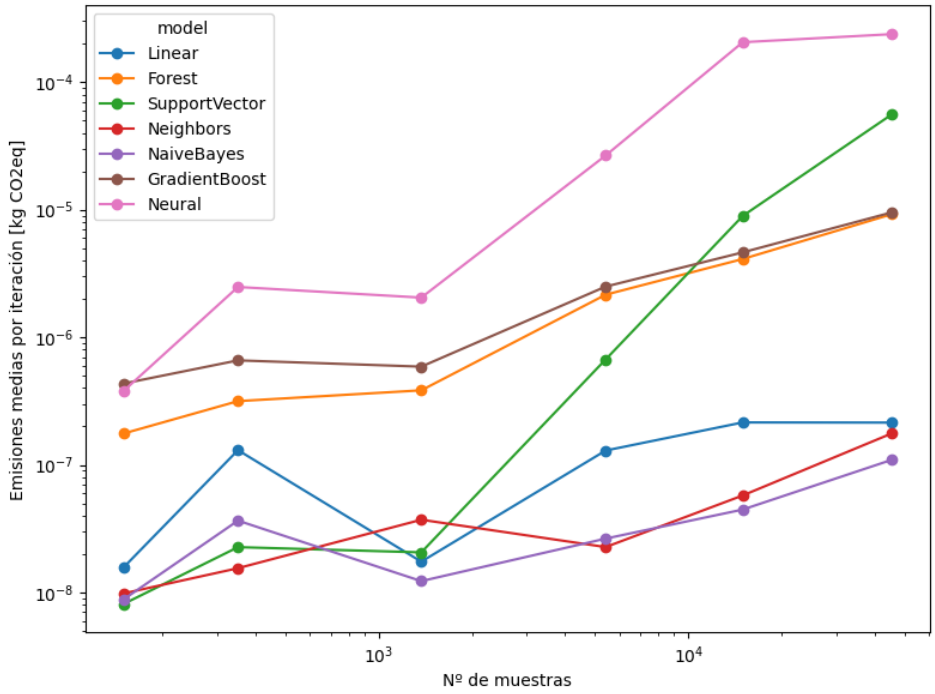


Figura 4.3: Evolución de las emisiones de carbono con el aumento de número de muestras del conjunto de datos

Para observar estos resultados de una forma más clara, se puede presentar la información en forma de gráfico de líneas. Las figuras 4.3 y 4.4 muestran de forma independiente las variaciones de la cantidad de emisiones y del valor-F al aumentar el número de muestras del conjunto de datos. Para estas gráficas, se han utilizado la media de emisiones y puntuación medidas en todas las iteraciones para cada modelo y conjunto.

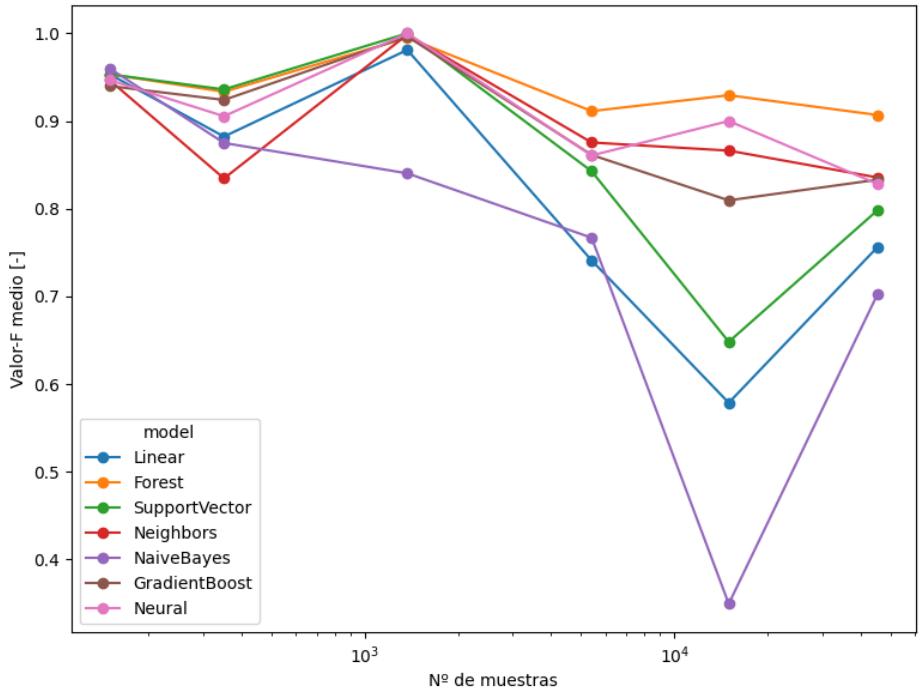


Figura 4.4: Evolución de las emisiones de carbono con el aumento de número de muestras del conjunto de datos

En la figura 4.3, es fácilmente apreciable la gran velocidad de crecimiento de las emisiones con el aumento de complejidad que presentan parte de los modelos (redes neuronales y vector soporte). Este tipo de gráfica permite seleccionar modelos interesantes en cuanto a las emisiones de forma sencilla, observando la inclinación de cada línea y su punto de partida. En la figura 4.4, el mayor beneficio que ofrece es poder descartar modelos que no alcancen un mínimo de puntuación en todos los escenarios, dependiendo de las necesidades específicas de cada caso. De este modo, se observa que los modelos de vector soporte, lineal y Naive Bayes serían fácilmente descartables como los más adecuados en este experimento concreto.

Conclusiones

En el experimento, se analizaron las emisiones de carbono y el rendimiento de varios modelos de aprendizaje automático utilizando validación cruzada con cinco iteraciones. Los resultados se exportaron a un archivo CSV, y se observaron variaciones en las emisiones y puntuaciones de valor-F. La figura de dispersión mostró cómo las emisiones y el valor-F varían entre los modelos y conjuntos de datos, destacando las diferencias significativas entre ellos.

El modelo de vecinos más cercanos se destacó por sus bajas emisiones y un valor-F constante entre 80% y 90%. El modelo de bosque aleatorio y el de potenciación de gradiente mostraron emisiones moderadas y un alto valor-F. Sin embargo, el modelo de red neuronal, aunque obtuvo puntuaciones elevadas, tuvo las emisiones más altas, especialmente con conjuntos de datos grandes. Los modelos lineal y Naive Bayes presentaron bajas emisiones, pero su valor-F disminuyó con la complejidad del conjunto de datos. Los resultados indican que la selección del modelo debe considerar tanto el rendimiento como las emisiones, siendo el modelo de vecinos más cercanos y el de bosque aleatorio opciones destacadas por su equilibrio entre ambos aspectos.

4.3 Consumo energético en función de los recursos disponibles

Objetivos

En el experimento anterior se observó que el modelo utilizado en el entrenamiento determina en gran medida el consumo energético. Sin embargo, hay una dimensión más que se puede estudiar para intentar reducir este consumo: los recursos computacionales dedicados a la tarea. Al utilizar un ordenador con menores recursos, el proceso de entrenamiento sin duda llevará más tiempo que en una máquina con mayores recursos. Sin embargo, su consumo energético por unidad de tiempo será comparativamente menor.

En esta sección pretende analizar que configuración de recursos actuará de forma más eficiente durante el entrenamiento de un conjunto de datos de gran tamaño, atendiendo a las siguientes cuestiones:

- Si es posible compensar más tiempo con menos consumo.
- Si se puede encontrar alguna combinación de recursos especialmente eficiente.
- Como afectan los recursos empleados a la calidad de las predicciones.

Metodología

La característica principal a analizar de las máquinas utilizadas será el número de procesadores dedicados a las tareas de entrenamiento. En un ordenador de un solo núcleo, cada iteración de la validación cruzada debe realizarse secuencialmente. Sin embargo, al aumentar el número de núcleos del procesador, es posible realizar las tareas de forma paralela. Para utilizar este mecanismo, es necesario activarla indicando a `scikit-learn` la opción `n_jobs=-1` para indicar el número de procesos a ejecutar en paralelo, donde -1 indica utilizar todos los disponibles.

Para analizar como afecta el número de procesadores disponibles al consumo energético, se tomarán medidas en máquinas con tres configuraciones diferentes. Estas máquinas serán desplegadas como máquinas virtuales en la nube, lo que facilitará controlar los recursos disponibles para cada configuración y ofrecerá un entorno similar de trabajo para todos los casos. La plataforma elegida para el despliegue es Microsoft Azure (sección 2.3.5), que ofrece una gran capacidad de personalización en los recursos desplegados y cuenta con una oferta de créditos gratuitos para el sector educativo que pueden ser empleados en minutos de ejecución de maquinas virtuales. La configuración elegida para cada una de las máquinas donde se tomarán las medidas está recogida en la tabla 4.3. Todas las máquinas utilizarán la misma versión de sistema operativo y el mismo tipo de procesador, con variaciones en el número de CPUs virtuales y la cantidad de memoria RAM disponible.

Tabla 4.3: Características técnicas de las máquina virtuales comparadas

Característica	**Test 1**	**Test 2**	**Test 3**
Sistema operativo	Ubuntu 20.04.6 LTS x86_64		
Procesador	Intel(R) Xeon(R) Platinum 8272CL CPU @ 2.60GHz		
Versión Python	3.12.3		
Configuración Azure	B1s	B2ms	B4ms
Memoria (GB)	0,87	7,75	15,57
vCPUs	1	2	4

El proceso completo seguido para cada configuración de recursos puede resumirse en los siguientes pasos:

1. Desplegar una máquina virtual en la nube con la configuración deseada.
2. Conectarse a la máquina virtual mediante SSH.
3. Descargar el repositorio del proyecto.
4. Ejecutar el script `azure/install.sh`.
5. Ejecutar el script `azure/parallel.sh`.

Este proceso será similar para cada combinación de recursos elegida. En primer lugar es necesario desplegar la máquina virtual con una red virtual configurada que permita conexiones mediante `ssh`, para poder ser controlada por línea de comandos. Los nombres que asigna Azure a las combinaciones de recursos específicas utilizadas están recogidos en la tabla 4.3.

Una vez que la máquina virtual se ha inicializado, será posible conectarse a ella mediante el protocolo `ssh`. Para no tener que configurar claves de acceso, es posible conectarse directamente con un terminal en línea que ofrece Azure directamente en su portal en la página de configuración de la máquina virtual. Cuando la conexión se establezca, el siguiente paso es descargar el repositorio del proyecto para tener acceso al código de la aplicación y a los recursos de instalación incluidos.

A continuación, se ejecutará el script `azure/install.sh` ubicado en el repositorio del proyecto[2]. Este script instalará la aplicación MLCost y todas sus dependencias, tomará una serie de medidas de entrenamiento en un conjunto de datos concreto, y exportará los resultados para su posterior análisis.

Por último, se ejecutará el script `azure/parallel.sh`[3]. Este archivo tomará dos medidas distintas. La primera simplemente medirá la puntuación y las emisiones de todos los modelos de aprendizaje en un conjunto de datos con validación cruzada de forma similar al experimento anterior. La segunda medida, sin embargo, utilizará la opción `-parallel` de la aplicación, que provoca que las distintas iteraciones de entrenamiento se ejecuten de forma paralela. Al finalizar el proceso de entrenamiento de todos los modelos, los archivos CSV con las medidas tomadas se exportarán automáticamente al repositorio de la aplicación para poder ser analizados fuera de la máquina virtual. Este proceso se repetirá para cada una de las tres máquinas virtuales desplegadas, obteniendo medidas de entrenamiento de seis casos en total. Los comandos utilizados en este script para realizar las medidas en cada una de las máquinas serán los siguientes:

```
mlcost measure --log -cv 5 --openml -d electricity
mlcost measure --log -cv 5 --openml -d electricity --parallel
```

[2] https://github.com/l-gonz/tfg-gitt-mlcost/blob/main/azure/install.sh
[3] https://github.com/l-gonz/tfg-gitt-mlcost/blob/main/azure/parallel.sh

El conjunto de datos elegido para analizar este proceso será *electricity* (ver 2.5.6), que contiene alrededor de 45K muestras. Este tamaño moderado permite ejecutar los experimentos en un tiempo razonable del orden de pocos minutos por modelo. Al mismo tiempo, tiene un tamaño suficiente para que las diferencias de consumo generadas por las distintas configuraciones de recursos sean visibles.

Resultados

Tras la ejecución de las seis mediciones tomadas los resultados se exportan a un archivo CSV de la misma manera que en el experimento anterior. El archivo completo con los resultados obtenidos está disponible en el repositorio de la aplicación[4]. En este caso, en vez de múltiples conjuntos de datos, se comparan las distintas configuraciones utilizadas. La figura 4.5 representa todas las medidas obtenidas con sus puntuaciones de valor-F y emisiones producidas por iteración de entrenamiento. A diferencia de en el experimento anterior, ahora las distintas medidas obtienen resultados muy similares para cada iteración, por lo que los puntos de la gráfica se encuentran muy agrupados. Esto se debe a que la configuración de la máquina de entrenamiento produce un efecto mucho menor en las emisiones que la complejidad del conjunto de datos a analizar. Además, al ejecutar el experimento en una máquina virtual dedicada únicamente a esta tarea, con muy pocos programas adicionales instalados, las iteraciones obtienen resultados mucho más consistentes, que apenas se desvían en emisiones o valor-F obtenido. Por esta razón, se observarán directamente los valores medios de emisiones y puntuación por modelo y configuración.

[4] https://github.com/l-gonz/tfg-gitt-mlcost/blob/main/out/output-parallel-azure-electricity.csv

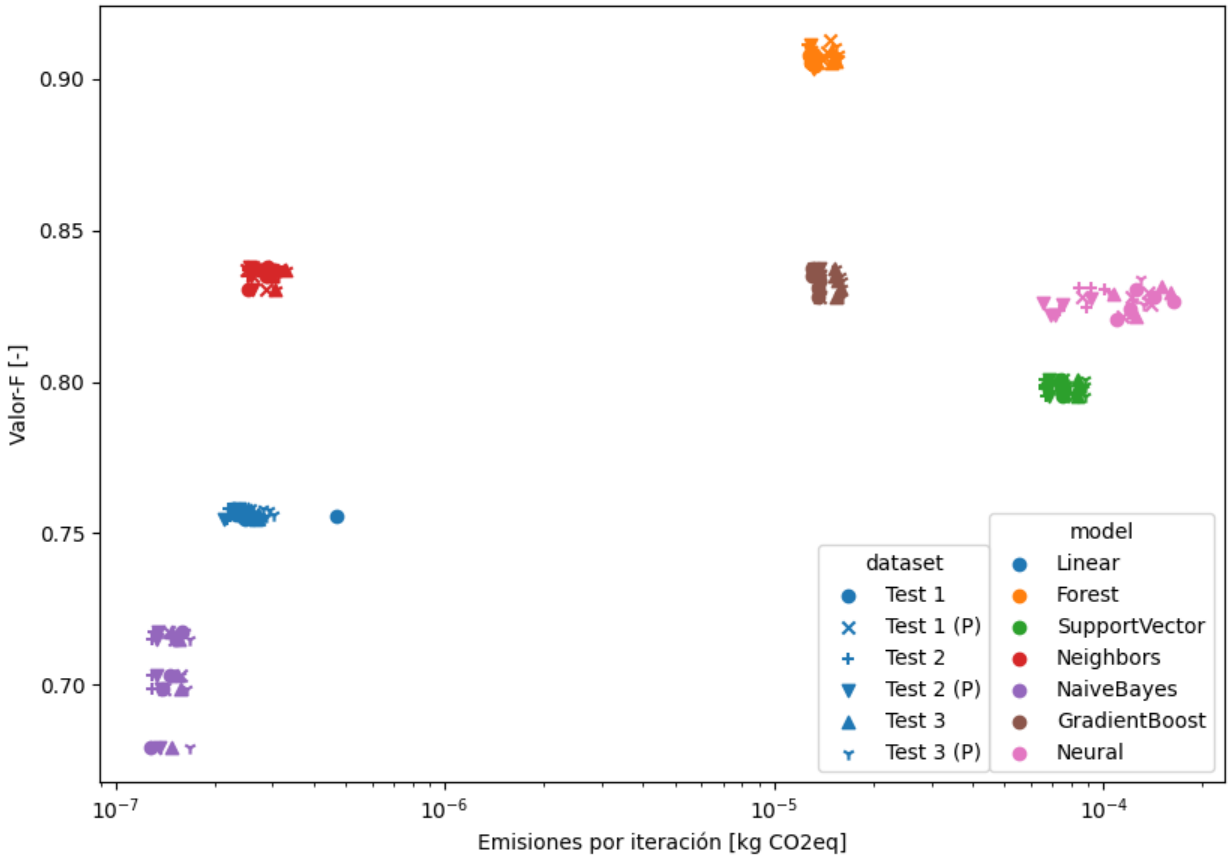


Figura 4.5: Valor-F alcanzado por el modelo frente a las emisiones de carbono necesarias para entrenarlo, por modelo empleado y configuración de la máquina utilizada.

La figura 4.6 muestra el mismo gráfico de dispersión con estos valores medios calculados. En primer lugar, se puede observar que el valor-F medio obtenido se mantiene completamente constante para todas las configuraciones analizadas, y solo depende del modelo utilizado. Por lo tanto, para este experimento se puede obviar este valor, ya que no hay diferencia con el análisis de la sección anterior.

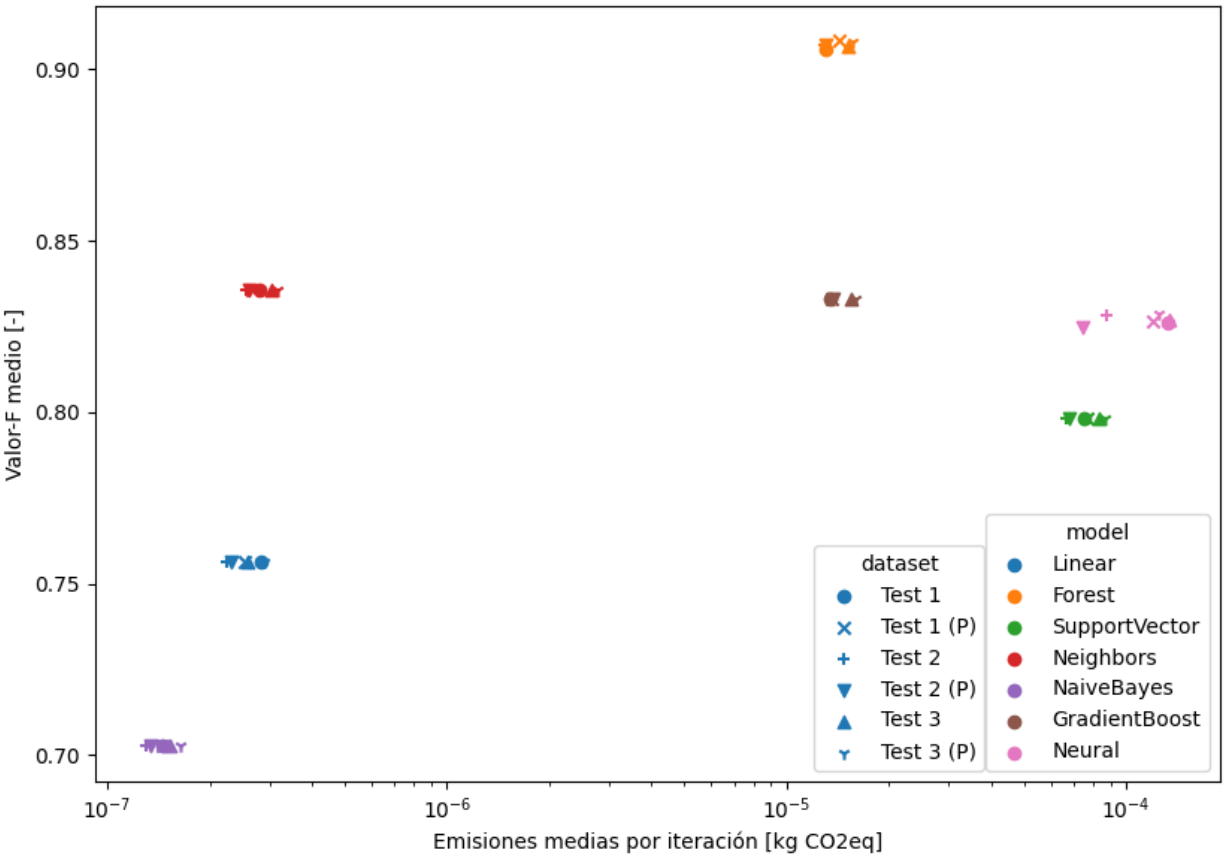


Figura 4.6: Valor-F medio de las iteraciones frente a las emisiones de carbono medias por iteración

En cuanto a las emisiones obtenidas por iteración , se puede apreciar que existe algún tipo de separación entre las diferentes máquinas. Estos cambios son difíciles de analizar cuando la diferencia entre emisiones es mucho más grande entre distintos modelos que entre distintas configuraciones de recursos. Para poder obtener una visión más concreta del efecto de los recursos empleados, se pueden calcular las emisiones por segundo de computo de forma global, teniendo en cuenta las emisiones totales del proceso de entrenamiento y el tiempo total empleado, en vez de las emisiones por cada iteración. Esta nueva medida será calculada siguiendo la formula 4.2.

$$E(t) = \frac{E_T}{t_T}, \tag{4.2}$$

donde: $E(t)$ = emisiones por segundo (kg $[CO_2eq]$ / s),
E_T = emisiones totales (kg $[CO_2eq]$),
t_T = tiempo total (s).

A partir de los resultados obtenidos, se puede utilizar un gráfico de líneas para observar las variaciones por configuración de recursos, como muestra la figura 4.7. Con esta representación, se puede observar claramente que las emisiones aumentan al utilizar paralelismo, y en mayor medida al utilizar más procesadores en paralelo. Este resultado corresponde con lo esperado, ya que una máquina con mayores recursos utilizados consumirá más energía. Es interesante destacar que no se encuentran grandes diferencias en emisiones entre las distintas configuraciones si no se usa paralelismo, lo que indica que no se están aprovechando todos los recursos disponibles.

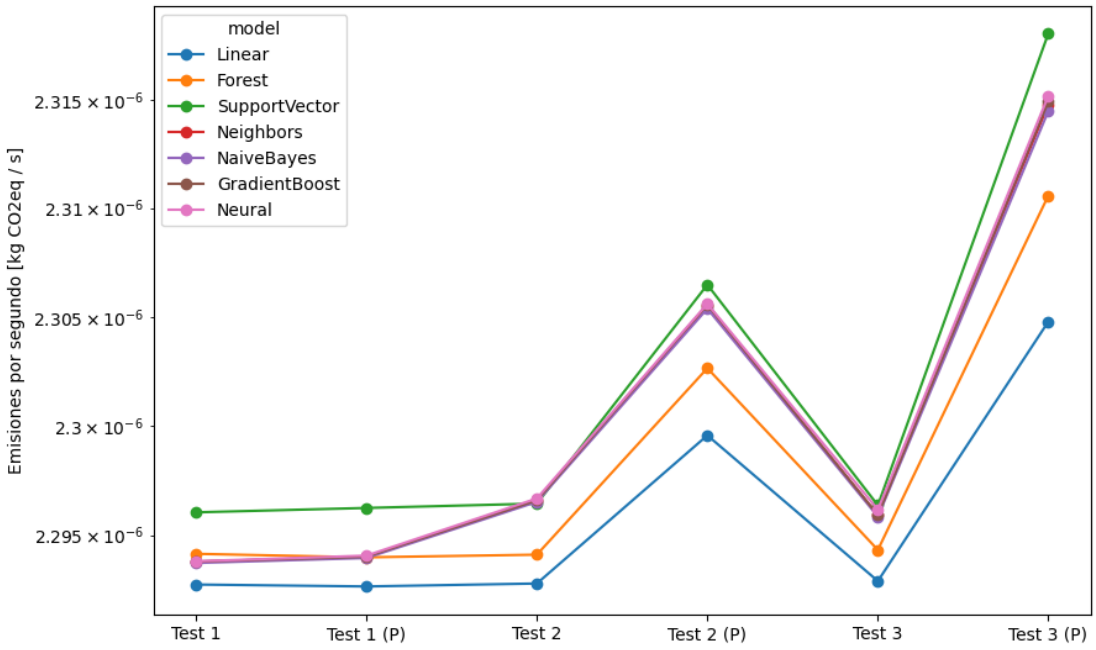


Figura 4.7: Distribución de la energía por segundo para cada modelo y test efectuado

Sin embargo, una medida de emisiones por segundo no tiene en cuenta el tiempo total empleado en el proceso de entrenamiento. Como ya se ha comentado anteriormente, el mayor consumo de energía de una configuración más potente podría ser compensado con creces al emplear un tiempo mucho menor en el entrenamiento. La figura 4.8 muestra las emisiones totales por configuración. Aquí se pude observar que para todos los modelos, las opciones que utilizan procesamiento en paralelo tienen unas emisiones totales menores, siendo la diferencia más significativa con un mayor numero de procesadores utilizados (T3-P).

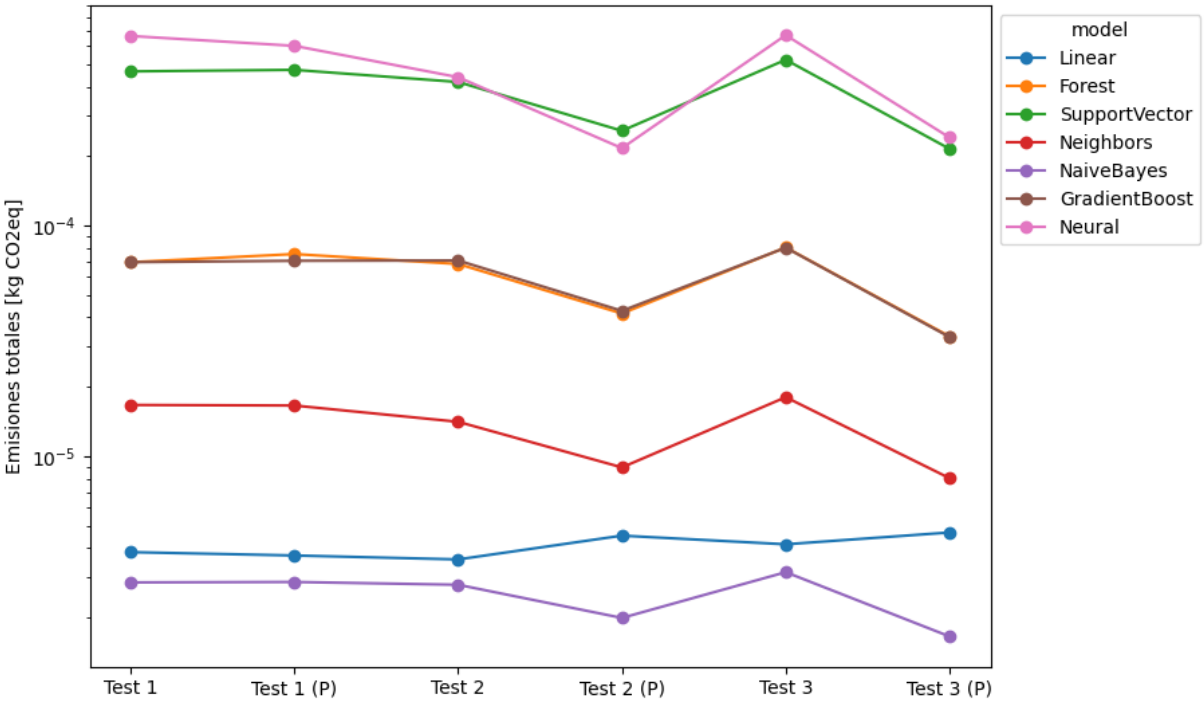


Figura 4.8: Distribución de las emisiones totales para cada modelo y test efectuado

Conclusiones

En este experimento, se realizaron seis mediciones cuyos resultados fueron exportados a un archivo CSV. A diferencia del experimento anterior, las configuraciones de hardware utilizadas en este experimento mostraron resultados muy consistentes en términos de emisiones y valor-F, debido a la uniformidad de la máquina de entrenamiento y la dedicación exclusiva de la máquina virtual.

El análisis de los valores medios reveló que el valor-F medio se mantuvo constante independientemente de la configuración, dependiendo únicamente del modelo utilizado. Las emisiones por iteración variaron más entre diferentes modelos que entre configuraciones de recursos. Se observó que las emisiones aumentan con el uso de paralelismo y más procesadores en paralelo, como era de esperarse, dado que el uso de más recursos incrementa el consumo de energía. Sin embargo, no se encontraron grandes diferencias en emisiones entre configuraciones sin paralelismo. Finalmente, se constató que aunque una configuración más potente consume más energía por segundo, el menor tiempo de entrenamiento requerido compensa este consumo, resultando en menores emisiones totales para opciones con procesamiento en paralelo.

Capítulo 5

Conclusiones y trabajos futuros

Este capítulo resume las conclusiones principales obtenidas durante el proyecto y los obstáculos más importantes que surgieron durante el desarrollo. A continuación, se examinarán los objetivos iniciales para comprobar si han sido conseguidos, se analizarán los conceptos adquiridos durante el proyecto y se discutirán posibles trabajos posteriores para profundizar en este tema.

El logro más importante de este proyecto ha sido el desarrollo de una herramienta que permite medir tanto el rendimiento de un modelo de aprendizaje como sus emisiones de carbono y consumo eléctrico, de una forma altamente personalizable para distintos conjuntos de datos y modelos representativos.

Los resultados obtenidos al utilizar esta herramienta para medir el impacto ambiental de varios modelos representativos muestran la importancia de considerar las emisiones producidas como una métrica más en el proceso de elección de modelo de aprendizaje para un problema concreto. Esto se debe a que muchos modelos ofrecen resultados de precisión muy similares, pero al analizar las emisiones se observan claras diferencias en las tendencias de consumo energético. Por lo tanto, es factible reducir las emisiones sin comprometer el rendimiento de la tarea de aprendizaje.

Durante el desarrollo de la aplicación, los principales obstáculos encontrados han estado relacionados con el preprocesamiento de los datos. Para conseguir que la herramienta pudiera comparar las emisiones producidas en una gran variedad de conjuntos de datos, se debieron encontrar los métodos más generales para limpiar y homogeneizar los datos, que consiguieran que todos los modelos pudieran obtener resultados de forma comparable. Al aplicar el mismo proceso de preparación para todos los conjuntos de datos, es imposible obtener el mismo rendimiento que se obtendría con un preprocesamiento específico. Sin embargo, el objetivo buscado ha sido sólo que fuera suficientemente cercano para todos los modelos por igual.

5.1 Consecución de objetivos

El objetivo principal de este proyecto era obtener una comparación del consumo y el impacto medioambiental de diferentes técnicas de clasificación. Este objetivo ha sido conseguido mediante la creación de una herramienta que usa técnicas de medición modernas para calcular las emisiones producidas durante el aprendizaje. La aplicación de esta herramienta a diferentes modelos representativos permite obtener una comparación de su rendimiento en múltiples situaciones.

Respecto a los objetivos específicos, la sección 2.1 estudió distintas herramientas ya existentes usadas en el campo de la medición de emisiones del aprendizaje automático. De estas, se eligió scikit-learn como motor de aprendizaje y CodeCarbon como base para las mediciones del impacto ambiental, debido a su facilidad de desarrollo e integración con otras librerías. Durante el desarrollo de la aplicación, ambas herramientas han sido ampliamente estudiadas para asegurar su buen funcionamiento en las pruebas llevadas a cabo.

Estas pruebas tenían como objetivo comparar el consumo energético de los algoritmos más importantes y analizar la relación entre la precisión y el consumo. El experimento de la sección 4.2 recoge varias de estas pruebas y extrae conclusiones sobre que modelos pueden ser más interesantes en el contexto de la limitación de las emisiones de carbono producidas.

5.2 Aplicación de lo aprendido

En el desarrollo de este proyecto, fue necesario emplear gran parte del conocimiento adquirido durante el Grado en Ingeniería en Tecnologías de la Telecomunicación completado. Las siguientes asignaturas han sido especialmente relevantes, al proporcionar la experiencia necesaria para completar este proyecto:

- **Fundamentos de la programación y de la informática.** Esta asignatura ofrece una primera introducción a la informática y la programación en el Grado. Además, provee el conocimiento fundacional de como escribir buen código, necesario para desarrollar un proyecto de estas características.
- **Procesamiento digital de la información.** Esta asignatura supone una introducción a los procesos de clasificación y estimación en el aprendizaje automático y estudia, incluyendo fundamentos teóricos de teoría de la decisión y diseño de clasificadores como vecinos más cercanos y regresión logística. Estos fundamentos teóricos construyen la base de los modelos de aprendizaje comparados en este proyecto.

- **Introduction to Artificial Intelligence** (Erasmus - NTNU). Esta asignatura proporciona una introducción a los conceptos y técnicas de la inteligencia artificial, utilizando Python como lenguaje de programación. Abarca desde algoritmos básicos hasta aplicaciones prácticas en aprendizaje automático y redes neuronales, sentando las bases necesarias para el desarrollo y la implementación de sistemas de inteligencia artificial en proyectos complejos.
- **Ingeniería de sistemas de la información.** Esta asignatura ofrece herramientas importantes para gestionar el control de versiones de cualquier proyecto de software y, especificamente, el uso de la plataforma GitHub para albergar el código del proyecto y mantener un registro de las modificaciones efectuadas.
- **Servicios y aplicaciones telemáticas.** La principal aportación de esta asignatura para este proyecto es la experiencia adicional en programación en Python, especialmente su uso en proyectos complejos de mayor tamaño. Además, introduce la utilización de la herramienta `pip` para la administración de paquetes, utilizada en el desarrollo de este proyecto.

5.3 Lecciones aprendidas

Durante el desarrollo de este proyecto, varios desafíos han requerido expandir el conocimiento mencionado en la sección anterior y adquirir nuevas competencias para encontrar soluciones a los problemas que surgían. Estas son algunas de las principales habilidades desarrolladas:

- Programación con Python: como organizar proyectos de gran tamaño con múltiples submódulos y crear paquetes personalizados. Adicionalmente, experiencia en varias librarías muy extendidas:
  - Gestión de vectores y matrices con la librería Pandas
  - Gestión de argumentos de línea de comandos con la librería Click.
  - Creación de gráficas personalizables para diferentes motores gráficos con la librería Matplotlib.
- Conocimiento de modelos de aprendizaje, ampliando el conocimiento teórico a la aplicación de los modelos a diferentes conjuntos de datos.
- Uso de scikit-learn como motor de aprendizaje automático, abstrayendo la implementación teórica de los algoritmos a analizar.
- Métodos de cálculo de emisiones, incluyendo el método utilizado por CodeCarbon en función de la energía consumida y la mezcla energética de la red eléctrica de cada país.

5.4 Trabajos futuros

El trabajo de este proyecto está enfocado en presentar pruebas básicas del funcionamiento de la aplicación comparativa desarrollada. Por lo tanto, existen muchas avenidas adicionales de estudio para ampliar este análisis.

Ampliación a modelos personalizados

Una posible línea de investigación futura es la ampliación del número y la diversidad de los modelos utilizados, incluyendo la posibilidad de trabajar con algoritmos adicionales además de los incluidos en la librería scikit-learn. De este modo, se podría comparar el consumo energético en contextos más específicos. Además, probar una gama más amplia de modelos de aprendizaje automático, incluidos aquellos basados en arquitecturas modernas como transformadores y redes neuronales profundas, podría ofrecer una visión más completa del consumo energético y el rendimiento.

Optimización Energética

Otro enfoque relevante sería investigar técnicas de optimización energética específicamente diseñadas para el entrenamiento de modelos de aprendizaje automático. Esto podría incluir el desarrollo de algoritmos de ajuste de hiperparámetros que consideren no solo el rendimiento del modelo, sino también su consumo energético. Asimismo, explorar técnicas de compresión de modelos y aprendizaje federado (aprendizaje colaborativo a través de una arquitectura descentralizada en múltiples dispositivos) podría reducir significativamente las emisiones asociadas al entrenamiento.

Paralelismo y Distribución de Tareas

El estudio de métodos más avanzados de paralelismo y distribución de tareas puede ser beneficioso. Investigaciones adicionales podrían centrarse en la eficiencia energética de diferentes enfoques de paralelización, como el uso de GPU y TPU, y en cómo la asignación dinámica de recursos en entornos distribuidos afecta el consumo energético. Además, la evaluación del impacto de técnicas de balanceo de carga y gestión de colas de tareas en sistemas de computación en la nube puede ofrecer perspectivas valiosas para optimizar el uso de recursos.

Medición en Entornos Reales

Realizar mediciones en entornos de producción reales podría proporcionar datos más precisos y aplicables sobre el consumo energético de los modelos. Esto incluiría la integración de herramientas de monitoreo de energía en sistemas de despliegue y la evaluación del impacto de diversas cargas de trabajo y patrones de uso en el consumo energético.

Apéndice A

Output del programa

A.1 Interfaz de comandos de la aplicación

```
Usage: mlcost measure [OPTIONS]

Options:
  -d, --dataset FILE              filepath to dataset, uses Iris dataset if
                                  none given. If using the --openml option, it
                                  is the dataset id in openml
  -l, --labels TEXT               labels column, defaults to last column
  -t, --test FILE                 filepath to test set, if none will get split
                                  test set from dataset
  -s, --separator TEXT            separator, defaults to a comma (no spaces)
  -f, --codecarbon-file TEXT      filename for default output from codecarbon,
                                  can be used with codecarbon's own
                                  visualization tool
  -m, --model TEXT                specify one model to run, default is to run
                                  everything
  -cv, --cross-validate INTEGER   cross validate the model, specify the number
                                  of folds (default is 1, no cross validation)
  --online                        use Codecarbon Emission Tracker in online
                                  mode
  --log                           output to additional csv file with whole
                                  experiment data
  --no-header                     do not consider the first row in the data to
                                  be the header
  --openml                        fetch dataset from openml
  --parallel                      parellelize cross validation between all
                                  cores, needs -cv option
  -h, --help                      Show this message and exit.


Usage: mlcost plot [OPTIONS]

Options:
```

```
  -f, --file FILE  filepath to csv file that contains the data  [required]
  -h, --help       Show this message and exit.
```

A.2 Resultados de la limpieza y preprocesado de los conjuntos de datos

A.2.1 Iris

```
$ mlcost measure --log -cv 5

---------------------------
DATA PREPROCESSING SUMMARY
Original data: 4.932e+03 bytes

Discarded features: 0
Discarded rows for missing labels: 0
Trained numerical features: ['sepal length (cm)', 'sepal width (cm)',
↪  'petal length (cm)', 'petal width (cm)']
Trained categorical features: []

Removed rows from missing categorical values - Train: 0 , Test: 0
Final train set rows: 120, test set rows: 30

Target distribution:
        train      test
target
0       0.350  0.266667
1       0.325  0.366667
2       0.325  0.366667
---------------------------
```

A.2.2 Ionosfera

```
$ mlcost measure --log -cv 5 -d data/ionosphere/ionosphere.data
↪  --no-header

---------------------------
DATA PREPROCESSING SUMMARY
```

```
Original data: 9.560e+04 bytes

Discarded features: 0
Discarded rows for missing labels: 0
Trained numerical features: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,
↪  14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30,
↪  31, 32, 33]
Trained categorical features: []

Removed rows from missing categorical values - Train: 0 , Test: 0
Final train set rows: 280, test set rows: 71

Target distribution:
       train      test
34
g   0.628571  0.690141
b   0.371429  0.309859
---------------------------
```

A.2.3 Autenticación de billetes

```
$ mlcost measure --log -cv 5 -d data/banknote/banknote.txt --no-header

---------------------------
DATA PREPROCESSING SUMMARY
Original data: 4.404e+04 bytes

Discarded features: 0
Discarded rows for missing labels: 0
Trained numerical features: [0, 1, 2, 3]
Trained categorical features: []

Removed rows from missing categorical values - Train: 0 , Test: 0
Final train set rows: 1097, test set rows: 275

Target distribution:
      train      test
4
0  0.559708  0.538182
1  0.440292  0.461818
---------------------------
```

A.2.4 Fonemas

```
$ mlcost measure --log -cv 5 -openml -d phoneme

---------------------------
DATA PREPROCESSING SUMMARY
Original data: 2.163e+05 bytes

Discarded features: 0
Discarded rows for missing labels: 0
Trained numerical features: ['V1', 'V2', 'V3', 'V4', 'V5']
Trained categorical features: []

Removed rows from missing categorical values - Train: 0 , Test: 0
Final train set rows: 4323, test set rows: 1081

Target distribution:
          train      test
Class
1      0.702984  0.720629
2      0.297016  0.279371
---------------------------
```

A.2.5 Electroencefalograma

```
$ mlcost measure --log -cv 5 -openml -d eeg-eye-state

---------------------------
DATA PREPROCESSING SUMMARY
Original data: 1.678e+06 bytes

Discarded features: 0
Discarded rows for missing labels: 0
Trained numerical features: ['V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7',
↪  'V8', 'V9', 'V10', 'V11', 'V12', 'V13', 'V14']
Trained categorical features: []

Removed rows from missing categorical values - Train: 0 , Test: 0
Final train set rows: 11984, test set rows: 2996

Target distribution:
```

```
          train      test
Class
1      0.550067  0.555741
2      0.449933  0.444259
----------------------------
```

A.2.6 Electricidad

```
$ mlcost measure --log -cv 5 -openml -d electricity

----------------------------
DATA PREPROCESSING SUMMARY
Original data: 2.584e+06 bytes

Discarded features: 0
Discarded rows for missing labels: 0
Trained numerical features: ['date', 'period', 'nswprice', 'nswdemand',
↪  'vicprice', 'vicdemand', 'transfer']
Trained categorical features: ['day']

Removed rows from missing categorical values - Train: 0 , Test: 0
Final train set rows: 36249, test set rows: 9063

Target distribution:
         train      test
class
DOWN   0.57403  0.581154
UP     0.42597  0.418846
----------------------------
```

Referencias

[1] William S. Cleveland. «Data Science: An Action Plan for Expanding the Technical Areas of the Field of Statistics». En: *International Statistical Review / Revue Internationale de Statistique* 69.1 (2001), págs. 21-26. ISSN: 03067734, 17515823.

[2] Eva García-Martín y col. «Estimation of energy consumption in machine learning». En: *Journal of Parallel and Distributed Computing* 134 (2019), págs. 75-88. ISSN: 0743-7315. DOI: 10.1016/j.jpdc.2019.07.007. URL: https://www.sciencedirect.com/science/article/pii/S0743731518308773.

[3] Nestor Maslej y col. *The AI Index 2024 Annual Report*. Inf. téc. AI Index Steering Committee, Institute for Human-Centered AI, Stanford University, Stanford, CA, 2024. eprint: 2405.19522. URL: https://aiindex.stanford.edu/report/.

[4] Siddharth Samsi y col. «From Words to Watts: Benchmarking the Energy Costs of Large Language Model Inference». En: *2023 IEEE High Performance Extreme Computing Conference (HPEC)*. 2023, págs. 1-9. DOI: 10.1109/HPEC58863.2023.10363447.

[5] Tom Brown y col. «Language models are few-shot learners». En: *Advances in neural information processing systems* 33 (2020), págs. 1877-1901.

[6] Sergei Alyamkin y col. «Low-power computer vision: Status, challenges, and opportunities». En: *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 9.2 (2019), págs. 411-421.

[7] Kadan Lottick y col. «Energy Usage Reports: Environmental awareness as part of algorithmic accountability». En: *arXiv preprint arXiv:1911.08354* (2019).

[8] Emma Strubell, Ananya Ganesh y Andrew McCallum. «Energy and policy considerations for deep learning in NLP». En: *arXiv preprint arXiv:1906.02243* (2019).

[9] Jesse Dodge y col. «Measuring the carbon intensity of ai in cloud instances». En: *Proceedings of the 2022 ACM conference on fairness, accountability, and transparency*. 2022, págs. 1877-1894.

[10] Abhinav Goel y col. «A survey of methods for low-power deep learning and computer vision». En: *2020 IEEE 6th World Forum on Internet of Things (WF-IoT)*. IEEE. 2020, págs. 1-6.

[11] Victor Schmidt y col. «CodeCarbon: Estimate and Track Carbon Emissions from Machine Learning Computing». En: (2021). DOI: 10.5281/zenodo.4658424.

[12] Johannes Getzner, Bertrand Charpentier y Stephan Günnemann. «Accuracy is not the only Metric that matters: Estimating the Energy Consumption of Deep Learning Models». En: *arXiv preprint arXiv:2304.00897* (2023).

[13] Alexandre Lacoste y col. «Quantifying the carbon emissions of machine learning». En: *arXiv preprint arXiv:1910.09700* (2019).

[14] Peter Henderson y col. «Towards the systematic reporting of the energy and carbon footprints of machine learning». En: *Journal of Machine Learning Research* 21.248 (2020), págs. 1-43.

[15] Lasse F Wolff Anthony, Benjamin Kanding y Raghavendra Selvan. «Carbontracker: Tracking and predicting the carbon footprint of training deep learning models». En: *arXiv preprint arXiv:2007.03051* (2020).

[16] Semen Andreevich Budennyy y col. «Eco2ai: carbon emissions tracking of machine learning models as the first step towards sustainable ai». En: *Doklady Mathematics*. Vol. 106. Suppl 1. Springer. 2022, S118-S128.

[17] John D Kelleher, Brian Mac Namee y Aoife D'arcy. *Fundamentals of Machine Learning for Predictive Data Analytics: Algorithms, worked examples, and case studies*. MIT press, 2020.

[18] T. Hastie, R. Tibshirani y J.H. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. 2ª ed. Springer Series in Statistics. Springer, 2009. ISBN: 9780387848846.

[19] J. Watt, R. Borhani y A.K. Katsaggelos. *Machine Learning Refined: Foundations, Algorithms, and Applications*. 2ª ed. Cambridge University Press, 2020. ISBN: 9781108480727.

[20] Alberto Fernández y col. *Learning from imbalanced data sets*. Vol. 10. 2018. Springer, 2018.

[21] Hayden Barnes. *Pro Windows Subsystem for Linux (WSL)*. APress, 2021.

[22] Benoit Courty y col. *mlco2/codecarbon: v2.3.4*. Ver. v2.3.4. Ene. de 2024. DOI: 10.5281/zenodo.10594225.

[23] F. Pedregosa y col. «Scikit-learn: Machine Learning in Python». En: *Journal of Machine Learning Research* 12 (2011), págs. 2825-2830.

[24] Lars Buitinck y col. «API design for machine learning software: experiences from the scikit-learn project». En: *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*. 2013, págs. 108-122.

[25] J. VanderPlas. *Python Data Science Handbook*. 2ª ed. O'Reilly Media, 2022. ISBN: 9781098121198.

[26] A. Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*. 3ª ed. O'Reilly Media, 2022. ISBN: 9781098122478.

[27] Scikit-learn developers. *LogisticRegression — scikit-learn 1.5.0 documentation*. Scikit-learn. URL: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression (visitado 11-06-2024).

[28] Scikit-learn developers. *1.10. Decision Trees — scikit-learn 1.5.0 documentation*. Scikit-learn. URL: https://scikit-learn.org/stable/modules/tree.html (visitado 11-06-2024).

[29] Scikit-learn developers. *RandomForestClassifier — scikit-learn 1.5.0 documentation*. Scikit-learn. URL: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html (visitado 11-06-2024).

[30] Scikit-learn developers. *1.4. Support Vector Machines — scikit-learn 1.5.0 documentation*. Scikit-learn. URL: https://scikit-learn.org/stable/modules/svm.html (visitado 12-06-2024).

[31] Scikit-learn developers. *SVC — scikit-learn 1.5.0 documentation*. Scikit-learn. URL: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html (visitado 11-06-2024).

[32] G. James y col. *An Introduction to Statistical Learning: With Applications in Python*. Springer International Publishing, 2023. ISBN: 9783031387487.

[33] Scikit-learn developers. *KNeighborsClassifier — scikit-learn 1.5.0 documentation*. Scikit-learn. URL: https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html (visitado 11-06-2024).

[34] Scikit-learn developers. *1.9. Naive Bayes — scikit-learn 1.5.0 documentation*. Scikit-learn. URL: https://scikit-learn.org/stable/modules/naive_bayes.html (visitado 12-06-2024).

[35] Scikit-learn developers. *GaussianNB — scikit-learn 1.5.0 documentation*. Scikit-learn. URL: https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html (visitado 12-06-2024).

[36] Scikit-learn developers. *GradientBoostingClassifier — scikit-learn 1.5.0 documentation*. Scikit-learn. URL: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html (visitado 12-06-2024).

[37] I. Goodfellow, Y. Bengio y A. Courville. *Deep Learning*. Adaptive Computation and Machine Learning series. MIT Press, 2016. ISBN: 9780262035613.

[38] Scikit-learn developers. *MLPClassifier — scikit-learn 1.5.0 documentation*. Scikit-learn. URL: https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html (visitado 12-06-2024).

[39] Dheeru Dua y Casey Graff. *UCI Machine Learning Repository*. 2017. URL: http://archive.ics.uci.edu/ml.

[40] Joaquin Vanschoren y col. «OpenML: networked science in machine learning». En: *SIGKDD Explorations* 15.2 (2013), págs. 49-60. DOI: 10.1145/2641190.2641198.

[41] Scikit-learn developers. *fetch_openml — scikit-learn 1.5.0 documentation*. Scikit-learn. URL: https://scikit-learn.org/1.5/modules/generated/sklearn.datasets.fetch_openml.html (visitado 10-06-2024).

[42] R. A. Fisher. *Iris*. UCI Machine Learning Repository. 1988. DOI: 10.24432/C56C76.

[43] V. Sigillito y col. *Ionosphere*. UCI Machine Learning Repository. 1989. DOI: 10.24432/C5W01B.

[44] Volker Lohweg. *Banknote Authentication*. UCI Machine Learning Repository. 2013. DOI: 10.24432/C55P57.

[45] Jesús Alcalá-Fdez y col. «KEEL Data-Mining Software Tool: Data Set Repository, Integration of Algorithms and Experimental Analysis Framework.» En: *Journal of Multiple-Valued Logic and Soft Computing* 17.2-3 (2011), págs. 255-287. URL: https://sci2s.ugr.es/keel/pdf/keel/articulo/2011-KEEL-dataset-MVLSC.pdf.

[46] Oliver Roesler. *EEG Eye State*. UCI Machine Learning Repository. 2013. DOI: 10.24432/C57G7J.

[47] Michael Harries. «SPLICE-2 Comparative Evaluation: Electricity Pricing». En: (1999).

[48] pandas developers. *User Guide — pandas 2.2.2 documentation*. URL: https://pandas.pydata.org/docs/user_guide/index.html (visitado 11-07-2024).

[49] A. Zheng y A. Casari. *Feature Engineering for Machine Learning: Principles and Techniques for Data Scientists*. O'Reilly Media, 2018. ISBN: 9781491953198.

[50] *Metrics and scoring: quantifying the quality of predictions*. Scikit-learn. URL: https://scikit-learn.org/stable/modules/model_evaluation.html (visitado 13-03-2024).

[51] M. Kuhn y K. Johnson. *Applied Predictive Modeling*. Springer New York, 2013. ISBN: 9781461468493.

[52] Sebastian Raschka. «Model Evaluation, Model Selection, and Algorithm Selection in Machine Learning». En: *CoRR* abs/1811.12808 (2018). arXiv: 1811.12808. URL: http://arxiv.org/abs/1811.12808.