ESCUELA DE INGENIERÍA DE FUENLABRADA

GRADO EN INGENIERÍA EN TECNOLOGÍAS DE LA TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

HERRAMIENTAS PARA EVALUAR EL IMPACTO MEDIOAMBIENTAL GENERADO POR MODELOS DE APRENDIZAJE AUTOMÁTICO

Autora : María Isabel Sánchez Sánchez
Tutor : José Felipe Ortega Soto

Curso Académico 2022/2023

Trabajo Fin de Grado/Máster

Herramientas para evaluar el impacto medioambiental generado por modelos de aprendizaje automático

Autor/a : María Isabel Sánchez Sánchez
Tutor/a : Dr. José Felipe Ortega Soto

La defensa del presente Proyecto Fin de Grado/Máster se realizó el día XX de
de 20XX, siendo calificada por el siguiente tribunal:

Presidente:

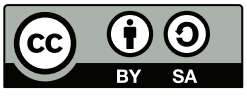
Secretario:

Vocal:

y habiendo obtenido la siguiente calificación:

Calificación:

Móstoles/Fuenlabrada, a de de 20XX



A mi madre y Alex,
mis mejores animadores

Agradecimientos

Quisiera expresar mis más sinceros agradecimientos a aquellas personas que me han acompañado y ayudado en esta recta final hacia mi titulación. Sin ellos no habría sido posible culminar con éxito este proyecto.

En primer lugar, deseo dar las gracias a mi familia, amigos y pareja por su constante apoyo y comprensión durante esta etapa de mi vida académica. Todos ellos han sido un gran impulso para afrontar este desafío y perseverar hasta el final.

Igualmente, me gustaría manifestar mi gratitud hacia mi tutor, José Felipe Ortega, cuya guía y asesoramiento a lo largo de todo el proceso han sido esenciales para dar forma a este Trabajo de Fin de Grado.

Resumen

En la época actual, estamos experimentando una revolución en el ámbito del aprendizaje automático, donde los avances realizados en este campo no cesan de generar nuevas aplicaciones y oportunidades dentro de la esfera empresarial. No obstante, hasta el momento, los expertos dedicados a esta área de la Inteligencia Artificial aún deben reflexionar sobre el impacto que puede llegar a causar el gran consumo energético de ciertos algoritmos sobre el medioambiente, contribuyendo, en consecuencia, al cambio climático.

Como respuesta a la concienciación que ha comenzado a surgir a raíz de este problema, el presente TFG tiene como principales objetivos el estudio y la comparativa de las herramientas *software* disponibles, con la intención de obtener la estimación del consumo y la huella de carbono durante el entrenamiento de modelos. Seguidamente, se desea demostrar que el uso de algoritmos que requieren un menor gasto energético es factible sin perder calidad de predicción.

Para ello, primero se han puesto a prueba en nuestro entorno de producción todas las herramientas que han sido escogidas y encontradas para dicho fin tras una búsqueda inicial. En base a lo observado durante este proceso, se han logrado determinar las dos mejores opciones. A continuación, como metodología de comparación, se ha desarrollado un banco de pruebas a través de Python, con la meta de contrastar el consumo eléctrico calculado por las herramientas seleccionadas y la eficiencia de predicción alcanzada frente a una misma configuración de *dataset* y algoritmo (implementado en Scikit-learn). Finalizado este experimento y gracias a los resultados obtenidos, se ha conseguido comprobar la veracidad del segundo objetivo expuesto previamente.

Summary

Currently, we are experiencing a revolution in the field of machine learning, where advances in this field continue to generate new applications and opportunities within the business sphere. However, until now, experts dedicated to this area of Artificial Intelligence have yet to reflect on the impact of the high energy consumption of certain algorithms on the environment, thus contributing to climate change.

In response to the awareness that has begun to emerge as a result of this problem, the main objectives of this TFG are to study and compare the software tools available in common data science languages for the estimation of consumption and carbon footprint during model training and, subsequently, to demonstrate that the use of algorithms that require less energy expenditure is feasible, without losing prediction quality.

To achieve this, the selected tools found for this purpose after an initial search have been tested in our production environment. The two best-performing tools have been determined based on the observations made during this process. As a comparison methodology, a benchmark has been developed using the Python programming language to contrast the electrical consumption calculated by the selected tools in the previous stage with the prediction efficiency achieved using the same dataset and algorithm configuration (Scikit-learn). By the end of this experiment and thanks to the obtained results, the validity of the second objective stated earlier has been confirmed.

Índice general

Índice de figuras

Índice de tablas

Índice de listados de código

Capítulo 1

Introducción

En la actualidad, nos encontramos ante un mundo cada vez más complejo donde la cantidad casi ilimitada de datos, las tecnologías de almacenamiento progresivamente más asequibles y el procesamiento computacional menos costoso y más potente han impulsado el crecimiento del aprendizaje automático. Estas técnicas se han convertido en una herramienta esencial dentro del ámbito empresarial. Lo anterior se debe a que el desarrollo de modelos capaces de analizar grandes cantidades de datos permite identificar más rápidamente oportunidades rentables y riesgos potenciales, lo que le otorga a las empresas una clara ventaja competitiva en la toma de decisiones.

Para el desarrollo de este proyecto, es necesario otorgar una definición de lo que se entiende como aprendizaje automático, algo que no resulta tan sencillo teniendo en cuenta que existen diversas definiciones sobre el mismo término. Podemos considerar el aprendizaje automático (*machine learning* o ML, por sus siglas en inglés) como la rama del campo de la Inteligencia Artificial que busca dotar a las máquinas de capacidad de aprendizaje para realizar de forma autónoma cierta tarea. Sin embargo, es difícil delimitar una definición específica de aprendizaje automático ya que, por ejemplo, Nvidia lo define como [21]:

> "*La práctica de usar algoritmos para analizar datos, aprender de ellos y luego hacer una determinación o predicción sobre algo en el mundo*".

En cambio, Stanford sugiere que es:

> "*La ciencia de hacer que las computadoras actúen sin estar programadas explícitamente*".

Esta rama de las tecnologías proporciona una serie de ventajas que se aplican a nivel global en múltiples disciplinas: sistemas de recomendación como los de Netflix, YouTube y Spotify; motores de búsqueda como Google y Baidu; feeds de redes sociales como Facebook y Twitter; asistentes de voz como Siri y Alexa; etc. Ello ha suscitado mucho interés por parte de los expertos, además de varios estudios e investigaciones a lo largo de los últimos años, destacando sobre todo lo referente al *deep learning* [19].

No obstante, el uso del aprendizaje automático conlleva, a su vez, una serie de inconvenientes. Los investigadores de Arquitectura de Computadores han estado investigando el consumo energético durante décadas para poder ofrecer procesadores energéticamente eficientes y de última generación. Sin embargo, los investigadores dedicados al campo del *machine learning* se han centrado principalmente en producir modelos de alta precisión, sin considerar el consumo de energía como un factor importante. Este es el caso del *deep learning* (aprendizaje profundo), donde el objetivo ha sido producir un modelo más profundo y preciso sin restricciones en términos de cálculo. A cambio, estos algoritmos requieren altos niveles de poder computacional durante su entrenamiento, ya que necesitan ser entrenados con grandes cantidades de datos, lo que implica un gran gasto energético y la generación de grandes volúmenes de emisiones de CO_2 que aumentan el nivel de la huella de carbono. Esta consiste en una métrica ambiental que calcula la totalidad de las emisiones de gases de efecto invernadero generadas directa e indirectamente [17, 25].

Recientemente, ha comenzado a surgir cierta concienciación sobre el consumo energético en aprendizaje automático, a partir de algunas investigaciones como en [34] y competiciones como en [2]. En dicha investigación, realizada por la Universidad de Massachussets, se cuantifican las emisiones de CO_2, la potencia y el coste de entrenar algunos de los modelos más avanzados y comunes de la Inteligencia Artificial. Entre los distintos hallazgos que se exponen en el artículo, se destaca que el proceso de entrenamiento de estos modelos puede emitir más de 626.000 libras de CO_2, lo que equivale a casi cinco veces las emisiones generadas por un automóvil estadounidense durante su vida útil, incluyendo su fabricación [17, 28].

Teniendo en cuenta lo comentado anteriormente, a pesar de los inmensos avances que posibilita el aprendizaje automático y de las ventajas que éste puede ofrecer, se demuestra que el gran coste energético que conlleva la ejecución y el entrenamiento de ciertos modelos supone un enorme problema medioambiental y, por tanto, puede llegar a contribuir de forma alarmante en el cambio climático, siendo capaz de intensificar sus efectos más nocivos. Entre dichos efectos, destacan el calentamiento global, las condiciones meteorológicas extremas, los riesgos para la salud humana y los costes para la sociedad y la economía. Asimismo, el gran coste financiero que implica dificulta de manera económica a los académicos, estudiantes e investigadores de esta rama de la Inteligencia Artificial, además de a las empresas que aplican estos modelos en sus planes de negocios [1].

Se sospecha que las razones por las que la comunidad de *machine learning* no había mostrado más interés en el consumo de energía son debidas a la falta de familiaridad con los enfoques actuales para estimar la energía, sumado a la falta de modelos de potencia en los *frameworks* de aprendizaje automático existentes. Adicionalmente, se cree que el motivo por el que muchas de las investigaciones de este campo no informan sobre las métricas de energía y carbono se debe a la complejidad de recopilar estos datos, ya que para esta tarea sería necesario registrar salidas de energía de GPU y CPU mediante la aplicación de diferentes herramientas [17].

[1] https://ec.europa.eu/clima/climate-change/climate-change-consequences_es

Por todo ello, resulta interesante y vital el desarrollo y el uso de herramientas como la creada por los investigadores de *Stanford*, *Facebook AI Research* y la Universidad de McGill [20], que permitan medir de una forma rápida y sencilla la cantidad de electricidad y el nivel de huella de carbono generados. De esta forma, sería posible conocer el impacto medioambiental y económico que se produciría en un proyecto de aprendizaje automático. Con el conocimiento de esta información, se logra cuantificar la eficiencia de los algoritmos y se ayuda a decidir si el gasto energético y económico creado es rentable en comparación con la calidad de los resultados obtenidos tras la ejecución del modelo. Así, seríamos capaces de tomar medidas para optimizarlo mediante, a modo de ejemplo, una reducción del número de parámetros y del tiempo total de ejecución, o incluso utilizando otro tipo de modelo más sencillo con el que se obtengan resultados de calidad similar, con el objetivo de reducir el consumo [5].

La importancia del proyecto que se describe a continuación reside en proporcionar datos, analizarlos y extraer conclusiones sobre el impacto medioambiental y la efectividad de los algoritmos más comunes en el campo del aprendizaje automático. La meta principal es ayudar a la optimización y la reducción del consumo para, de este modo, lograr en un futuro un *machine learning* más limpio y ecológico.

Este trabajo se enfocará inicialmente en el análisis y la comparación de las diferentes herramientas software actuales para la estimación de la huella de carbono en los algoritmos de aprendizaje automático, centrándonos especialmente en aquellas que emplean lenguajes comunes en *data science*. Una vez llevada a cabo esta tarea, se aplicarán dichas herramientas para medir la huella de carbono de los distintos modelos, yendo desde los más sencillos o clásicos (regresión lineal, *random forests*, etc.), hasta los más complejos (redes neuronales, etc.), con el fin de efectuar una comparativa en función de su consumo y de la efectividad a la hora de ofrecer resultados.

1.1 Objetivos del proyecto

1.1.1 Objetivo general

Con el propósito de hacer frente a la problemática expuesta en la introducción (1), este Trabajo de Fin de Grado aspira a cumplir estos dos objetivos durante su desarrollo:

- Realizar un estudio sobre el panorama actual de las herramientas existentes para la estimación de la huella de carbono en el campo de la Inteligencia Artificial, las cuales hayan sido creadas y diseñadas con el objetivo de ser empleadas en lenguajes de programación habituales en ciencia de datos.
- Poner a prueba dichas herramientas frente a distintos algoritmos emblemáticos de ML, con el fin de llevar a cabo un *benchmarking* y determinar si es posible escoger y aplicar un modelo de mayor eficiencia energética sin sacrificar significativamente el nivel de predicción.

1.1.2 Objetivos específicos

Los objetivos generales de este proyecto se han dividido en las siguientes tareas:

- Estudiar y experimentar con todas las herramientas que hayan sido encontradas tras la investigación inicial.
- Crear una tabla que compare estas herramientas en función de sus características.
- A partir de la información recopilada a lo largo de la ejecución de los dos objetivos anteriores, seleccionar las dos mejores herramientas, teniendo en cuenta que estas sean similares y comparables entre sí.
- Someter a prueba las herramientas escogidas con objeto de efectuar un banco de pruebas que compare el consumo y la eficiencia de predicción calculados frente a una misma combinación de algoritmo y *dataset*.
- Analizar los resultados obtenidos en el *benchmarking* y, en base a ello, tratar de responder estas preguntas:
    1. Para un mismo *dataset* y algoritmo, ¿aportarán un valor similar de gasto energético ambas herramientas?
    2. En caso contrario, ¿cuáles serían las posibles razones de las discrepancias entre las estimaciones de estas herramientas para un mismo caso de uso?
    3. ¿Sería posible conocer algunos detalles adicionales sobre el método que sigue cada herramienta para calcular el consumo provocado?
    4. ¿Podríamos obtener una calidad de predicción similar con modelos de aprendizaje automático que requieran un menor consumo energético y, de esta forma, disminuir la huella de carbono?

1.2 Planificación temporal

El completo desarrollo de este TFG ha exigido, en total, una cantidad aproximada de 200 horas, las cuales han transcurrido entre el inicio del curso 2021/2022 (septiembre) y el final del curso 2022/2023 (julio), cuyo mayor nivel de esfuerzo fue llevado a cabo a lo largo del último curso durante los fines de semana y los periodos vacacionales de dicho intervalo.

1.3 Estructura de la memoria

Por último, se presenta la estructura utilizada para organizar los capítulos de esta memoria:

- En el primer capítulo se hace una breve introducción al desafío que trata de abordar este proyecto, se describen los objetivos que aspira a cumplir el mismo y se refleja la planificación temporal.
- En el siguiente capítulo (Capítulo 2) se describen las tecnologías empleadas en el desarrollo del presente TFG y se realiza el estudio y las conclusiones iniciales sobre las herramientas analizadas para la estimación de la huella de carbono.
- En el capítulo 3 se expone la arquitectura del banco de pruebas y se detallan todos los elementos que participarán en dicha configuración.
- Posteriormente, en el capítulo 4, se explica el procedimiento seguido para emprender y ejecutar el *benchmarking* a efectos de mostrar los resultados obtenidos y las conclusiones alcanzadas.
- Como remate final (Capítulo 5), se dan a conocer las objetivos completados, así como las futuras aplicaciones y mejoras que podrían surgir del mismo.

Capítulo 2

Estado del arte

El objetivo del capítulo 2 es describir y presentar las razones por las que se han escogido las distintas tecnologías para elaborar este TFG. A continuación, se realizará una primera presentación de los estimadores de la huella de carbono en ML del escenario actual, poniendo en evidencia el estudio de cada uno de ellos con su respectiva comparación y conclusiones.

2.1 Herramientas para el desarrollo del proyecto

2.1.1 Python

Python se trata de un lenguaje de programación creado por Guido van Rossum y lanzado por primera vez el 20 de febrero de 1991 que despliega estas características:

Figura 2.1: Características de Python.

Todos estos atributos hacen que se conciban múltiples razones para su utilización y lo convierten en un lenguaje ideal para un sinnúmero de aplicaciones. En cuanto al desarrollo de este proyecto, se ha escogido Python como lenguaje de programación principalmente por tres motivos:

1. Si estudiamos cuáles son los lenguajes más populares actualmente, vemos que *TIOBE Programming Community* [1] y *PYPL PopularitY of Programming Language* [2], dos indicadores de la popularidad de los lenguajes de programación, posicionan a Python como el lenguaje más popular.

2. El segundo y principal motivo por el que se ha optado por Python en lugar de otros lenguajes como Java o C/C++ es debido a que Python se trata del lenguaje de programación más empleado en el ámbito del aprendizaje automático. Esto se justifica en vista de las características presentadas en 2.1 y por el hecho de que tiene un gran ecosistema de librerías que ayudan a los científicos de datos a realizar su trabajo de forma mucho más sencilla. Algunas de las librerías más utilizadas en ML son las presentadas en la tabla inferior:

Tabla 2.1: Librerías más utilizadas en ML. [11]

Nombre	**Funcionalidad principal**
Pandas	Análisis y manipulación de estructuras de datos de alto nivel.
Matplotlib	Creación de visualizaciones estáticas, animadas e interactivas.
NumPy	Computación de datos en forma de matrices multidimensionales.
Scikit-learn	Procesamiento de datos y algoritmos de aprendizaje automático.
TensorFlow	*Deep learning*.

3. Finalmente, todas las herramientas *software* que se van a presentar en la sección 2.2 para el cálculo de la huella de carbono durante el entrenamiento de modelos están desarrolladas en Python. Por lo tanto, estaremos obligados a usar este lenguaje de programación cuando se realice el estudio de las herramientas que se expondrán seguidamente.

2.1.2 Miniconda

Una vez que se ha determinado que Python va a ser el lenguaje de programación que se va a aplicar durante todo el proyecto, el siguiente paso es decidir cómo se va a llevar a cabo la creación del entorno virtual donde se realizará todo el proceso de elaboración.

[1] https://www.tiobe.com/tiobe-index/
[2] https://pypl.github.io/PYPL.html

Para ello existen diversas alternativas. En este caso, la decisión se encuentra entre Anaconda y Miniconda, ya que son las dos distribuciones más enfocadas al aprendizaje automático que existen en la actualidad.

Anaconda es una distribución de código abierto y de fácil instalación de los lenguajes de programación Python y R. Proporciona un entorno de trabajo que se utiliza para informática científica, ciencia de datos, análisis estadístico y aprendizaje automático, donde destacan las siguientes funciones [3]:

- Un sistema de administración de paquetes y entornos virtuales de código abierto llamado **conda**, que permite:
  - La instalación y actualización de paquetes.
  - La creación y manejo de entornos virtuales.
- Incluye un gran número de librerías enfocadas en los campos del aprendizaje automático (TensorFlow, Scikit-learn y Theano), la ciencia de datos (Pandas, NumPy y Dask) y de visualización (Bokeh, Datashader, Matplotlib y Holoviews).

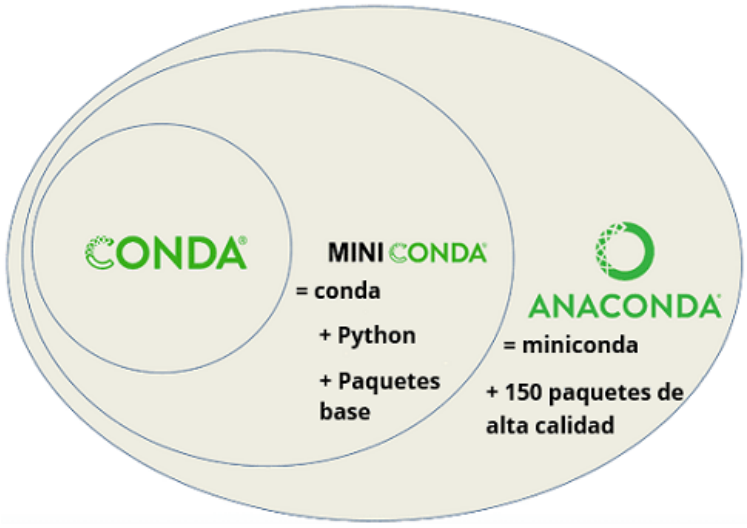


Figura 2.2: Esquema Conda, Anaconda y Miniconda [29].

A pesar de todas las funciones y ventajas que nos aportaría la utilización de Anaconda, el proyecto que tenemos por objetivo efectuar es ligero en cuanto al número de paquetes que necesitamos aplicar para su ejecución. Por esta razón, el uso de Anaconda daría lugar a la creación de un entorno virtual pesado y lento, consumiendo varios giga-bytes de espacio en el disco duro. Todos estos motivos nos llevan a concluir que la mejor opción como distribución para la creación de nuestro entorno de trabajo es Miniconda.

Dicho lo anterior, Miniconda [4] se trata de una versión de Anaconda que incluye únicamente conda, Python, los paquetes de los que dependen y una pequeña cantidad de otros

[3] https://www.venturelessons.com/what-is-anaconda/
[4] https://docs.conda.io/en/latest/miniconda.html

paquetes útiles (pip, zlib, entre otros). Gracias a la gran reducción en el número de paquetes que se ha efectuado en la versión de Miniconda, se consigue un entorno de trabajo ligero y ágil sin consumir mucho espacio en el disco duro. No obstante, como desventaja, si necesitamos algún paquete que no viene incluido en el repertorio por defecto de Miniconda, este deberá se añadido de forma manual.

2.1.3 PyCharm

PyCharm es un entorno de desarrollo integrado (IDE) de Python desarrollado por la compañía checa *JetBrains* que proporciona una amplia gama de herramientas esenciales para sus desarrolladores. Estas herramientas de las que dispone PyCharm están específicamente integradas con el objetivo de crear un entorno adecuado para el desarrollo web y la ciencia de datos [5].

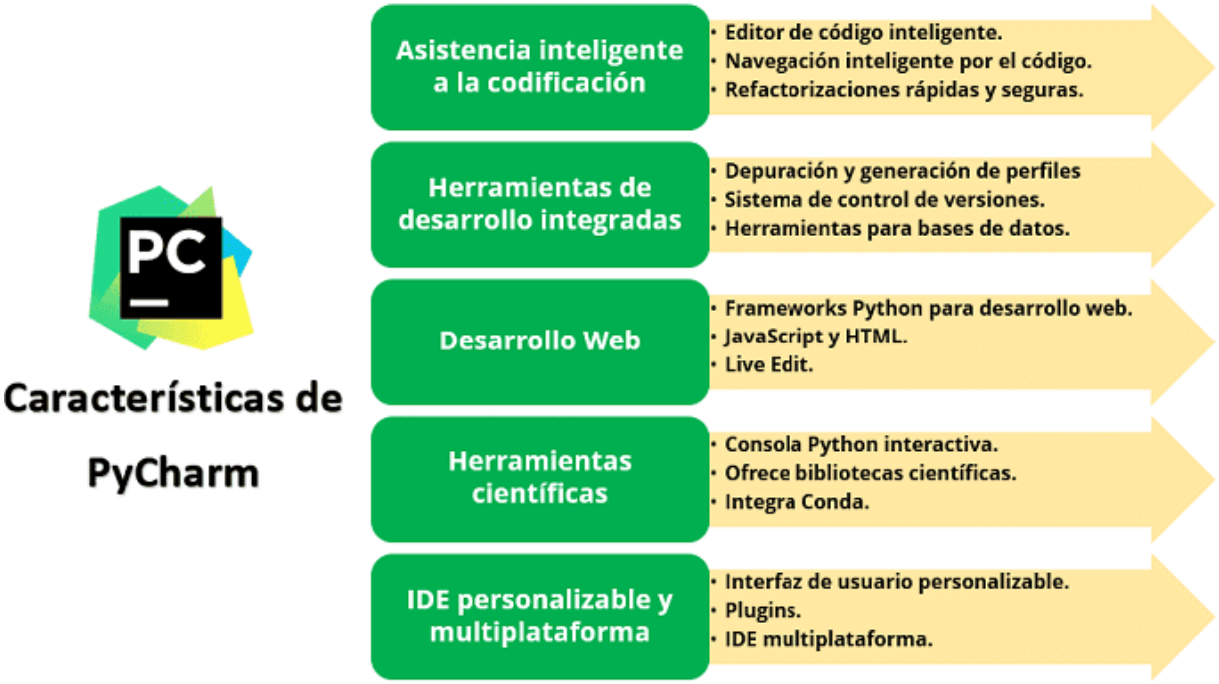


Figura 2.3: Características de PyCharm.

Pese a la existencia de una gran cantidad de IDEs enfocadas en ofrecer un entorno adecuado para la creación de proyectos en Python (KDevelop, Thonny o Visual Studio), las características de PyCharm expuestas lo convierten en la mejor IDE de Python para desarrolladores en la actualidad, siendo este el primer motivo de su elección para la elaboración de este trabajo.

PyCharm está disponible en tres versiones [6]:

[5] https://www.jetbrains.com/es-es/pycharm/features/
[6] https://www.jetbrains.com/help/pycharm/2021.2/quick-start-guide.html

1. *Community* (gratuita y de código abierto): Orientada al desarrollo inteligente de Python. Incluye la asistencia de código, refactorizaciones, depuración visual y control de versiones.

2. *Professional* (pago): Orientada al desarrollo profesional de Python, web y ciencia de datos. Incluye la asistencia de código, refactorizaciones, depuración visual, control de versiones, configuraciones remotas, soporte para *frameworks* populares como Django y Flask, soporte de bases de datos y herramientas científicas.

3. *Edu* (gratuita y de código abierto): Orientada al aprendizaje de lenguajes de programación y tecnologías relacionadas con herramientas educativas integradas.

Entre todas las versiones especificadas, se toma la decisión de emplear como IDE **la versión Profesional de PyCharm**, la cual se puede adquirir de forma gratuita gracias a nuestra cuenta universitaria. Esta versión se enfoca en el desarrollo profesional de proyectos en Python en las áreas de ciencia de datos y aprendizaje automático, incluyendo librerías científicas esenciales como las indicadas en la tabla 2.1 e integrando conda, lo que convierte a esta versión en la opción perfecta.

2.1.4 GitHub

Git es un sistema de control de versiones que se utiliza para realizar un seguimiento de los cambios en los archivos de un determinado proyecto. Entre las diversas funciones de las que dispone *Git*, las cuales lo convierten en una herramienta de primera categoría para los programadores de todo el mundo, se pueden destacar las siguientes [15]:

- Coordinación del flujo de trabajo entre los miembros del equipo del proyecto.

- Supervisión del progreso a lo largo del tiempo.

- Trabajo simultáneo de varios usuarios sin interrupciones.

Una vez explicado el concepto de *Git* y algunas de sus aplicaciones más relevantes, será más sencillo entender con mayor claridad la definición de GitHub que se expondrá a continuación.

GitHub es un servicio de alojamiento de repositorios *Git* que proporciona una interfaz gráfica basada en una página web. Gracias a GitHub, los programadores pueden encontrar de manera sencilla códigos fuente en muchos lenguajes de programación diferentes y utilizar la interfaz de línea de comandos *Git* para realizar un seguimiento de los cambios. Además, GitHub ayuda a todos los miembros del equipo que conforman un proyecto a trabajar juntos desde cualquier ubicación mientras facilita la colaboración entre ellos. También, puede revisar versiones anteriores de los ficheros que forman parte del repositorio.

Figura 2.4: Características de GitHub.

Se ha escogido GitHub como sistema de control de versiones porque se trata de la comunidad de codificación más grande a día de hoy. En consecuencia, tiene la gran ventaja de que si decidimos crear nuestro código en un proyecto de GitHub, ello aumentará la exposición generalizada del programa en cuestión, haciéndolo mucho más accesible a los usuarios.

Aparte de GitHub, existe otro mecanismo de control de versiones muy similar a este llamado GitLab, con el cual los estudiantes de la facultad de telecomunicaciones están más familiarizados dada su aplicación en diversas materias. Sin embargo, esta opción no ha sido seleccionada y, en su lugar, se ha escogido GitHub para aprender y practicar su manejo, ya que su uso está más extendido a nivel global.

2.1.5 Scikit-learn

Scikit-learn [7] se trata de una librería Python construida sobre las librerías NumPy, SciPy y Matplotlib, siendo hoy en día la librería más útil, robusta y utilizada para el aprendizaje automático en dicho lenguaje. Fue desarrollada inicialmente por David Cournapeau como un proyecto de *Google Summer of Code* en 2007. Scikit-learn, en lugar de encargarse de la manipulación de datos como otro tipo de librerías, se enfoca en modelar los datos. Para

[7] https://scikit-learn.org/stable/

ello, facilita una selección de herramientas eficientes para ML y modelado estadístico, entre las que se incluyen muchos de los algoritmos de aprendizaje supervisado y no supervisado más populares [8].

Figura 2.5: Funciones de Scikit-learn.

Su aplicación tendrá por objetivo la creación de los modelos necesarios para probar la efectividad de cada una de las herramientas que se presentarán más adelante en la sección 2.2. Entre todas las ventajas que aporta Scikit-learn, las que han logrado que nos decantemos por su uso son las siguientes [38]:

1. Se trata de una librería de código abierto. Es distribuida bajo la licencia BSD, lo que la hace gratuita con mínimas restricciones legales y de licencia.
2. Gracias a lo mencionado en el punto anterior, Scikit-learn está respaldado y actualizado por numerosos autores y colaboradores, disponiendo de una vasta comunidad internacional. Por consiguiente, existe una documentación excepcional sobre el mismo, lo que nos facilitará la resolución de las posibles dudas que puedan ir surgiendo durante la ejecución de este trabajo.
3. Por último, esta librería es muy sencilla de emplear, puesto que cuenta con una interfaz que se caracteriza por ser simple, muy consistente y de aprendizaje rápido.

[8] https://www.tutorialspoint.com/scikit_learn/scikit_learn_quick_guide.htm

Además, cubre la mayor parte de las tareas que se pueden realizar en *machine learning*, comentadas previamente en la imagen 2.5.

Si bien todas estas ventajas han convertido a *Scikit-learn* en la librería de ML elegida para el proceso de implementación del presente TFG, se debe resaltar como desventaja que no es la mejor opción si se desean desarrollar modelos de aprendizaje profundo. Para esa finalidad, existen otras librerías como *TensorFlow* que se centran exclusivamente en ese ámbito.

2.2 Herramientas para la estimación de la huella de carbono

Notas

La infraestructura que ha sido empleada para llevar a cabo el análisis de las herramientas consiste en una **Máquina Virtual** con los siguientes **metadatos**:

- **Sistemas operativos**: Ubuntu 22.04.1 LTS y Ubuntu 16.04.07 LTS
- **Versión de Python**: 3.10
- **RAM**: 4,9 GiB
- **Número de CPUs**: 4
- **Modelo de la CPU**: Intel(R) Core(TM) i5-9400F CPU @ 2.90GHz
- **Capacidad de disco**: 107,9 GB

2.2.1 CodeCarbon

Como se ha explicado detalladamente en la introducción (1), aunque la IA puede beneficiar a la sociedad de múltiples y diversas maneras, la cantidad de energía necesaria para soportar la computación masiva que la respalda puede llegar a tener un alto coste para el medioambiente. Además, la informática representa en la actualidad, aproximadamente, el 0,5% del consumo energético mundial. En los próximos años, se prevé que ese porcentaje crezca más del 2%, lo que se traducirá en un aumento significativo de las emisiones globales de CO_2. A la vista de estos hechos, resulta importante cuantificar y rastrear el alcance de este uso de energía con el fin de minimizar las emisiones tanto como sea posible [31, 14].

Por este motivo, demostrando su compromiso con el desarrollo de tecnología responsable con el ecosistema, el equipo de expertos formado por *Mila*, líder mundial en investigación de IA; *BCG GAMMA*, equipo global de ciencia de datos e IA de BCG; *Haverford College* y *Comet*, proveedor líder de soluciones MLOps, lanzaron CodeCarbon como propuesta.

CodeCarbon es una librería Python que permite rastrear y calcular las emisiones de carbono producidas por la nube, o por los recursos informáticos personales que hayan sido

utilizados durante la ejecución de un determinado código en experimentos de aprendizaje automático. Estos experimentos pueden abarcar numerosas y diversas áreas, partiendo desde el desarrollo de algoritmos sencillos, hasta la elaboración de redes neuronales profundas (*deep neural networks*). Para proporcionar dicha estimación, CodeCarbon tiene en cuenta estos parámetros [9]:

- Infraestructura informática (GPU, CPU, ...).
- Ubicación.
- Uso.
- Tiempo de ejecución.

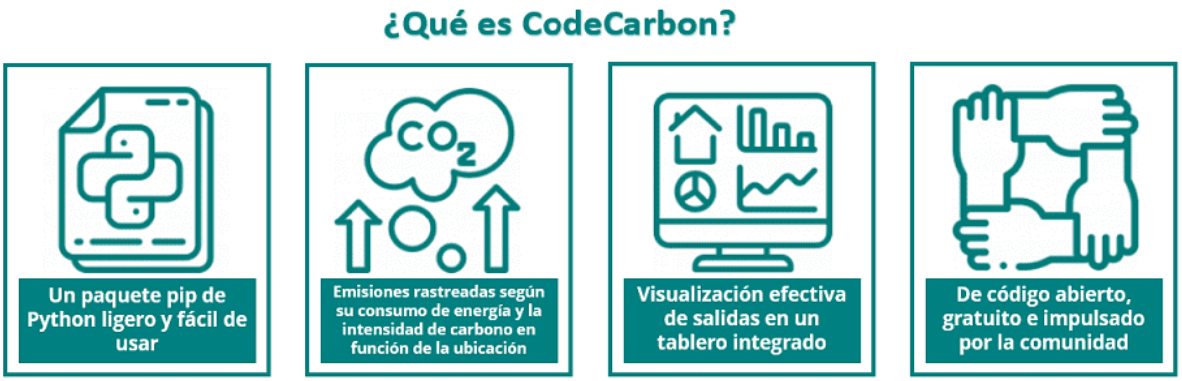


Figura 2.6: ¿Qué es CodeCarbon?

Asimismo, CodeCarbon dispone de una herramienta de visualización la cual, a partir de los resultados estimados, es capaz de generar gráficos como los siguientes [10]:

- **Resumen y equivalencias:** Esta aplicación, además de mostrar al beneficiario las emisiones generadas y la potencia neta consumida por un determinado proyecto, proporciona equivalencias que se corresponden con ciertas actividades de la vida diaria:
  1. Energía semanalmente consumida por un hogar estadounidense promedio.
  2. Número de millas conducidas.
  3. Tiempo de visualización de la televisión (LCD TV 32 pulgadas).
- **Comparaciones regionales:** Paralelamente, se enseña un gráfico comparativo de las emisiones (*emissions equivalent*) y la combinación de energía (*energy mix*) procedentes de la red eléctrica de diferentes países.
- **Regiones de la nube:** De igual forma, se muestra una ilustración que compara las emisiones equivalentes en diferentes regiones del proveedor de servicios en la nube y recomienda la región más ecológica para albergar la infraestructura en cuestión.

[9] https://codecarbon.io/index.html
[10] https://mlco2.github.io/codecarbon/visualize.html

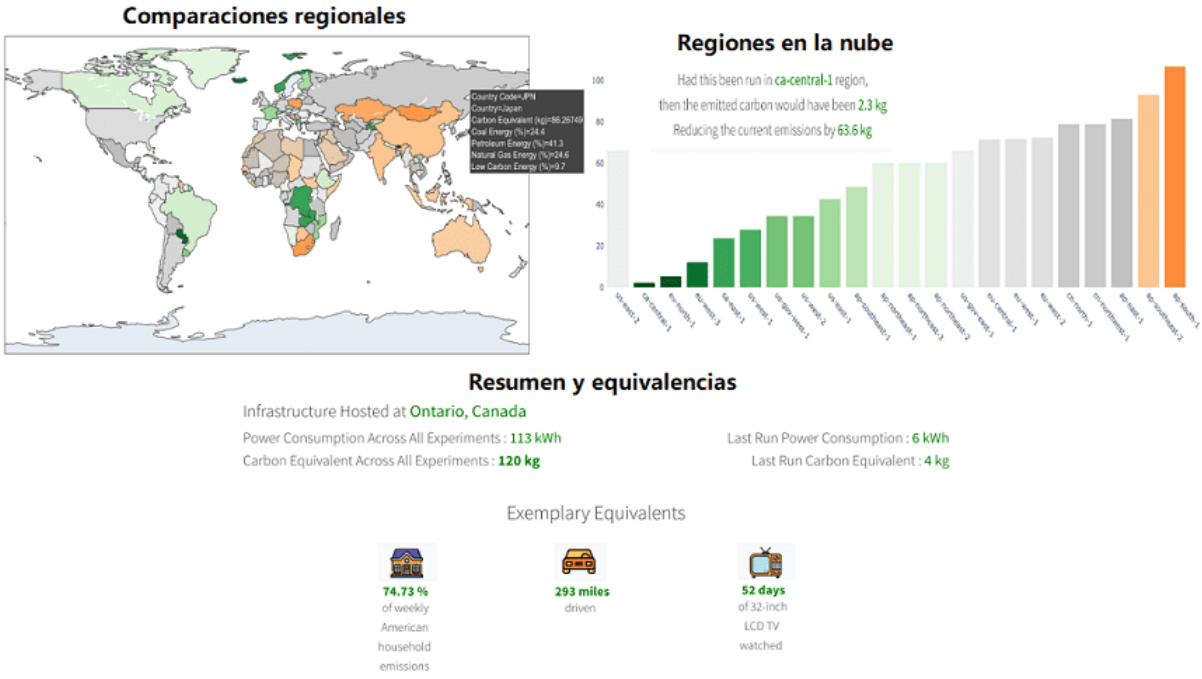


Figura 2.7: Ilustraciones de CodeCarbon [11].

Toda esta información en su conjunto, permite a los equipos de desarrollo obtener una mayor visión de la cantidad de emisiones causadas por los proyectos y, también, facilita la toma de decisiones como la región donde se debería situar la infraestructura y/o el proveedor de la nube, con la perspectiva de reducir al máximo el impacto medioambiental.

La solución propuesta por CodeCarbon es, sin duda, un paso adelante en el desarrollo de un planeta más sostenible. Se espera por parte de sus creadores que esta herramienta cumpla la sucesiva serie de objetivos:

- Se prevé que CodeCarbon sea una forma de ayudar a reducir la huella de carbono causada por la IA. De este modo, se aspira a aumentar la capacidad de los desarrolladores para utilizar los recursos energéticos de manera inteligente y, por lo tanto, reducir el impacto de su trabajo en un entorno cada vez más frágil.
- Es deseable que los desarrolladores e investigadores que aprovechen dicha herramienta den *feedback* y contribuyan a mejorarla con nuevas capacidades.
- Para aumentar la conciencia sobre el impacto ambiental de la informática, los creadores recomiendan encarecidamente que los usuarios informen de la huella de carbono de sus experimentos en trabajos de investigación, artículos y blogs de tecnología.
- Por último, los participantes de este proyecto tienen la expectativa de que esta herramienta ayude también a introducir una mayor transparencia en la comunidad de desarrolladores, permitiéndoles medir e informar de las emisiones generadas por sus actividades informáticas.

[11] https://mlco2.github.io/codecarbon/visualize.html

Instalación de CodeCarbon

La librería de CodeCarbon se encuentra disponible en el repositorio de PyPI, por lo que es posible instalarla de manera sencilla a partir del instalador de paquetes pip: [12]

- **Sin herramienta de visualización:**

```
(codecarbon) root@ubuntu:~$ pip install codecarbon
```

- **Con herramienta de visualización:**

```
(codecarbon) root@ubuntu:~$ pip install codecarbon[viz]
```

Además del método de instalación descrito, también existe la opción de instalar dicha herramienta a través del repositorio de conda:

```
(codecarbon) root@ubuntu:~$ conda install -c conda-forge codecarbon
```

Los siguientes paquetes pip son utilizados por CodeCarbon y serán instalados junto con la propia librería:

arrow	fuzzywuzzy	sutil	pynvml
click	pandas	py-cpuinfo	requests

Nota

Para el uso de esta herramienta se recomienda utilizar Python 3.7 o superior.

Uso de CodeCarbon

Tras su última actualización, CodeCarbon ofrece a los usuarios dos opciones para comenzar con el rastreo de sus emisiones [13]:

1. **Monitorización de su máquina**: Para llevar a cabo esta acción, simplemente se debe ejecutar el siguiente comando:

```
(codecarbon) root@ubuntu:~$ codecarbon monitor
```

[12] https://mlco2.github.io/codecarbon/installation.html
[13] https://github.com/mlco2/codecarbon

En este caso el paquete rastreará las emisiones **independientemente de su código**.

2. **Seguimiento de su código Python**: El paquete será capaz de rastrear las emisiones generadas por la ejecución de su código Python.

Con objeto de efectuar el rastreo de nuestro código, CodeCarbon dispone de la capacidad de trabajar en dos modos con tres posibilidades diferentes [14]:

1. **Modo *Online*:** Constituye el uso más sencillo del paquete. Su aplicación es posible si se tiene acceso a Internet, el cual es necesario para recopilar información sobre la ubicación geográfica:

   - **Objeto explícito:** Se recomienda cuando se utiliza Jupyter Notebook:

```python
from codecarbon import EmissionsTracker

tracker = EmissionsTracker()
tracker.start()
# Compute intensive code goes here
tracker.stop()
```

Código 1: Rastreador CodeCarbon: modo *online* y objeto explícito.

   - ***Context Manager:*** Se recomienda cuando se desea monitorear un bloque de código específico:

```python
from codecarbon import EmissionsTracker

with EmissionsTracker() as tracker:
    # Compute intensive training code goes here
```

Código 2: Rastreador CodeCarbon: modo *online* y *context manager*.

   - ***Decorator:*** Se recomienda si se tiene una función de entrenamiento:

```python
from codecarbon import track_emissions

@track_emissions
def training_loop():
    # Compute intensive training code goes here
```

Código 3: Rastreador CodeCarbon: modo *online* y *decorator*.

2. **Modo *Offline*:** Está enfocado a máquinas sin acceso a Internet, pero requiere la definición manual del código ISO del país correspondiente.

[14] https://mlco2.github.io/codecarbon/usage.html

- **Objeto explícito:**

```python
from codecarbon import OfflineEmissionsTracker

tracker = OfflineEmissionsTracker(country_iso_code="CAN")
tracker.start()
# GPU intensive training code
tracker.stop()
```

Código 4: Rastreador CodeCarbon: modo *offline* y objeto explícito.

- ***Context Manager:***

```python
from codecarbon import OfflineEmissionsTracker

with OfflineEmissionsTracker() as tracker:
    # GPU intensive training code goes here
```

Código 5: Rastreador CodeCarbon: modo *offline* y *context manager*.

- ***Decorator:*** El decorador **`track_emissions`** requiere de dos parámetros:
  - **`offline`**: Debe establecerse como **`True`**.
  - **`country_iso_code`**: Código ISO del país.

```python
from codecarbon import track_emissions

@track_emissions(offline=True, country_iso_code="CAN")
def training_loop():
    # training code goes here
    pass
```

Código 6: Rastreador CodeCarbon: modo *offline* y *decorator*.

Ejemplos

Antes de comenzar, es necesario asignar a nuestro ejemplo un **`experiment_id`**. Para ello, solo se debe escribir este comando:

```
(codecarbon) root@ubuntu:~$ codecarbon init
```

La identificación generada se almacenará en el fichero **`.codecarbon.config`** en la raíz del proyecto:

```
[codecarbon]
#the experiment_id you get with init
experiment_id = 86d8d916-4af9-41f0-ba2f-e6cb822a7f67
```

- **Ejemplo 1: Entrenamiento de un modelo ML:** Para demostrar cómo usar dicha herramienta, se empleará el *dataset iris*. Este es un famoso conjunto de datos que contiene la longitud y el ancho de sépalos y pétalos de 150 flores iris de tres diferentes especies: *Iris Setosa*, *Iris Versicolor* e *Iris Virginica* [18].

  A partir de este *dataset*, se entrenará un clasificador para detectar el tipo de flor *Iris Virginica* basándonos únicamente en la anchura del pétalo y, seguidamente, se estimará con CodeCarbon el impacto causado durante su entrenamiento:

  Nota

  Este ejemplo se empleará como base para las herramientas restantes.

  – **Lectura del *dataset***: Este código coincidirá con todas las demostraciones que se llevarán a la práctica para el resto de herramientas de este capítulo:

```python
from sklearn import datasets

iris = datasets.load_iris()
X = iris['data'][:,3:] # petal width
y = (iris['target'] == 2).astype('int64')  # 1 if Virginica, else 0
```

Código 7: Ejemplo iris: Lectura del *dataset* y obtención de datos.

  – **Rastreo del entrenamiento**: Esta parte del código se verá modificada en función de la herramienta que se esté ensayando. En el caso de CodeCarbon, dispondrá del siguiente aspecto:

```python
from codecarbon import EmissionsTracker
from sklearn.linear_model import LogisticRegression

tracker = EmissionsTracker()
log_reg = LogisticRegression()

tracker.start()
log_reg.fit(X, y) # Train Logistic Regression model
emissions: float = tracker.stop()
print(f'Emissions: {emissions} kg')
```

Código 8: Ejemplo iris: Rastreo del entrenamiento con CodeCarbon.

Si se lleva a cabo la ejecución del código expuesto, este mostrará a la salida el resultado que se visualiza en la imagen:

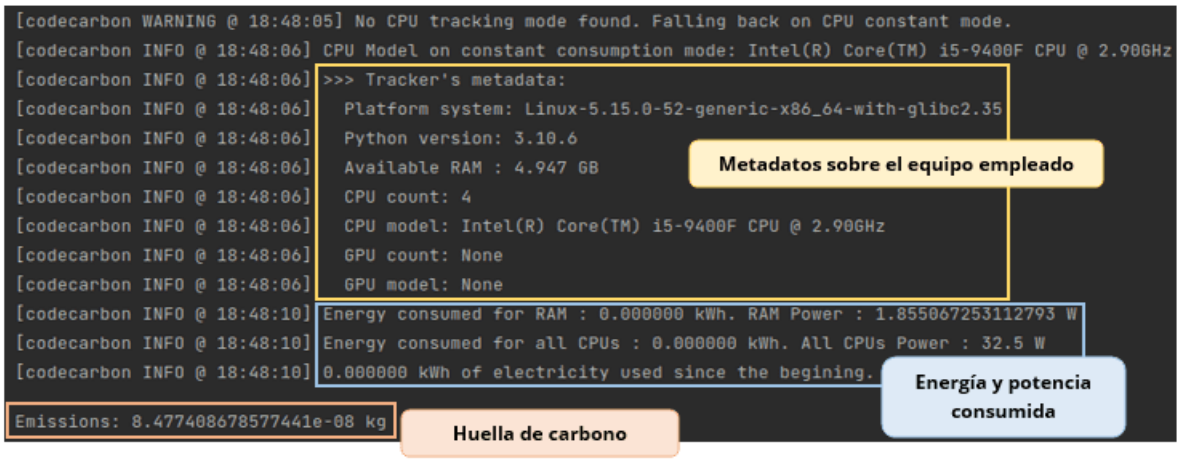


Figura 2.8: Resultado del rastreo de un modelo ML con CodeCarbon.

Esto escribirá un archivo llamado **`emissions.csv`** en el directorio actual con los resultados e información adicional que no ha sido aportada en la salida anterior.

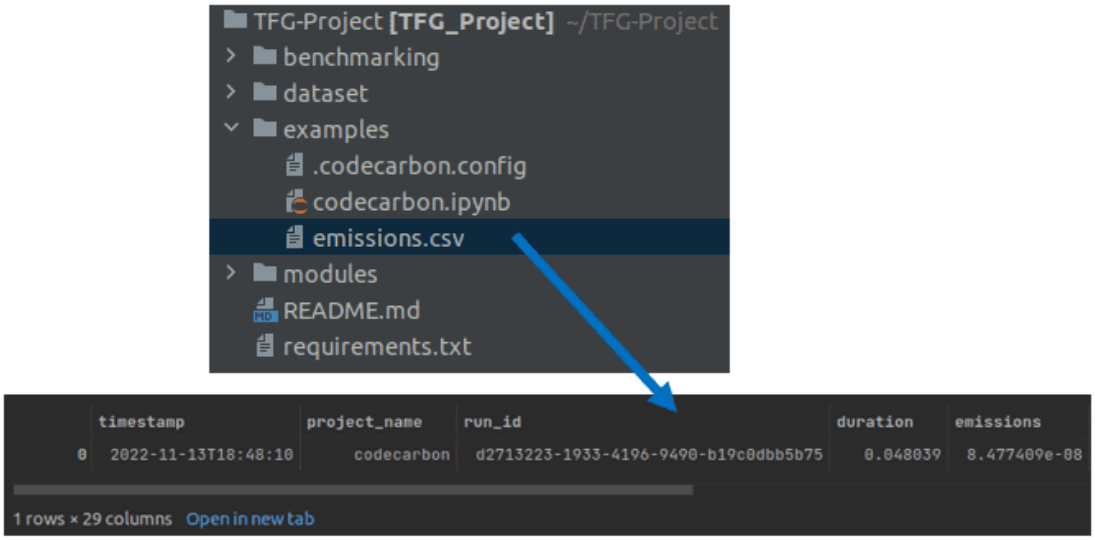


Figura 2.9: Fichero resultado, `emissions.csv`.

El fichero almacena en su interior **una tabla con 29 columnas y tantas filas como ejecuciones hayan sido realizadas**. En este ejemplo, como únicamente se ha realizado una ejecución, solo almacenará una única fila.

Las 29 columnas que componen el fichero se definen seguidamente [15]:

[15] https://mlco2.github.io/codecarbon/output.html

Tabla 2.2: Datos registrados por CodeCarbon.

Dato	Descripción
`timestamp`	Fecha y hora del experimento.
`project_name`	Nombre del proyecto, por defecto es codecarbon.
`run_id`	id de la ejecución.
`duration`	Duración del cálculo en segundos.
`emissions`	Emisiones como equivalentes de CO_2 $[CO_2eq]$ en kg.
`emissions_rate`	`emissions` divididas entre `duration`.
`cpu_power`	Potencia de CPU consumida (W).
`gpu_power`	Potencia de GPU consumida (W).
`ram_power`	Potencia de RAM consumida (W).
`cpu_energy`	Energía utilizada por CPU (kWh).
`gpu_energy`	Energía utilizada por GPU (kWh).
`ram_energy`	Energía utilizada por RAM (kWh).
`energy_consumed`	Suma de `cpu_energy`, `gpu_energy` y `ram_energy`.
`country_name`	Nombre del país donde está alojada la infraestructura.
`country_iso_code`	Código ISO del país respectivo.
`region`	Región donde está alojada la infraestructura.
`on_cloud`	`Y` (equipo en la nube), `N` (equipo privado).
`cloud_provider`	Proveedor de nube (`aws/azure/gcp`).
`cloud_region`	Región geográfica del respectivo proveedor de nube.
`os`	Sistema operativo del dispositivo.
`python_version`	Versión de Python.
`cpu_count`	Número de CPUs.
`cpu_model`	Modelo del CPU(s).
`gpu_count`	Número de GPUs.
`gpu_model`	Modelo de GPU(s).
`longitude`	Longitud. Precisión reducida: alcance de $11km/123km^2$.
`latitude`	Latitud. Precisión reducida: alcance de $11km/123km^2$.
`ram_total_size`	RAM total disponible.
`Tracking_mode`	`machine` o `process`, por defecto `machine`.

- **Ejemplo 2: Entrenamiento de un modelo DL:** Para probar la librería en esta situación, se utilizará el ejemplo **mnist.py** disponible en el repositorio GitHub de CodeCarbon. Este ejemplo consiste en el entrenamiento de un modelo de *deep learning* a partir del clásico *dataset mnist* mediante Tensorflow [16]. El conjunto de datos *mnist* constituye un grupo de 70.000 pequeñas imágenes de dígitos escritos a mano por estudiantes de secundaria y empleados del *US Census Bureau*. Cada imagen está etiquetada con el dígito que representa [18].

  Si accedemos al archivo **mnist.py**, nos encontramos con el próximo código:

  Nota

  Se ha reducido el número de **epochs** a 3 para disminuir el tiempo de ejecución.

```python
import tensorflow as tf
from codecarbon import EmissionsTracker

# Split mnist data into train and test
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

# Create the model with Tensorflow
model = tf.keras.models.Sequential(
    [
        tf.keras.layers.Flatten(input_shape=(28, 28)),
        tf.keras.layers.Dense(128, activation="relu"),
        tf.keras.layers.Dropout(0.2),
        tf.keras.layers.Dense(10),
    ]
)

# Set up the model to get the losses and metrics.
loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
model.compile(optimizer="adam", loss=loss_fn, metrics=["accuracy"])

# Track with codecarbon the model's training
tracker = EmissionsTracker()
tracker.start()
model.fit(x_train, y_train, epochs=3)
emissions: float = tracker.stop()
print(emissions)
```

Código 9: Ejemplo de CodeCarbon, `mnist.py`.

Si ejecutamos el ejemplo expuesto, se obtiene en la terminal la salida que se muestra a continuación:

[16] https://mlco2.github.io/codecarbon/examples.html

```
[codecarbon INFO @ 18:38:01] >>> Tracker's metadata:
[codecarbon INFO @ 18:38:01]   Platform system: Linux-5.15.0-56-generic-x86_64-with-glibc2.35
[codecarbon INFO @ 18:38:01]   Python version: 3.10.8
[codecarbon INFO @ 18:38:01]   Available RAM : 4.947 GB
[codecarbon INFO @ 18:38:01]   CPU count: 4
[codecarbon INFO @ 18:38:01]   CPU model: Intel(R) Core(TM) i5-9400F CPU @ 2.90GHz
[codecarbon INFO @ 18:38:01]   GPU count: None
[codecarbon INFO @ 18:38:01]   GPU model: None
Epoch 1/3
1147/1875 [=================>............] - ETA: 7s - loss: 0.3533 - accuracy: 0.8958[codecarbon INFO @ 18:38:19]
Energy consumed for RAM : 0.000008 kWh. RAM Power : 1.8550643920898438 W
[codecarbon INFO @ 18:38:19] Energy consumed for all CPUs : 0.000136 kWh. All CPUs Power : 32.5 W
1875/1875 [==============================] - 19s 10ms/step - loss: 0.2934 - accuracy: 0.9136
Epoch 2/3
1875/1875 [==============================] - 7s 4ms/step - loss: 0.1422 - accuracy: 0.9575
Epoch 3/3
 358/1875 [====>.........................] - ETA: 6s - loss: 0.1077 - accuracy: 0.9659[codecarbon INFO @ 18:38:34]
Energy consumed for RAM : 0.000016 kWh. RAM Power : 1.8550643920898438 W
[codecarbon INFO @ 18:38:34] Energy consumed for all CPUs : 0.000274 kWh. All CPUs Power : 32.5 W
1875/1875 [==============================] - 10s 5ms/step - loss: 0.1065 - accuracy: 0.9675
[codecarbon INFO @ 18:38:42] Energy consumed for RAM : 0.000020 kWh. RAM Power : 1.8550643920898438 W
[codecarbon INFO @ 18:38:42] Energy consumed for all CPUs : 0.000348 kWh. All CPUs Power : 32.5 W
[codecarbon INFO @ 18:38:42] 0.000368 kWh of electricity used since the begining.
6.994120774628666e-05
```

Figura 2.10: Resultado del rastreo de un modelo DL con Codecarbon.

En la imagen se observa cómo se obtiene la misma información que en el ejemplo anterior. Sin embargo, en esta ocasión, se aportan **medidas intermedias sobre la energía y potencia consumidas**. Igualmente, se incluye una nueva fila en nuestro fichero `emissions.csv` con las métricas recopiladas durante esta nueva ejecución.

2.2.2 Experiment-impact-tracker

En el momento actual, la mayoría de los trabajos de investigación de ML no informan regularmente de las métricas de energía consumida y emisiones de carbono producidas. El equipo formado por investigadores de *Stanford*, *Facebook AI Research* y *McGill University*, sugiere que esto es debido a la complejidad que existe a la hora de obtener este tipo de datos y expone las siguientes razones por las cuales es importante incluir esta clase de métricas en los informes [20]:

- **Concienciación**: Concienciar a los investigadores del impacto ecológico de sus modelos con el propósito de que se adopten estrategias de mitigación e impulsar un comportamiento respetuoso con el medioambiente.
- **Incentivos alineados**: Aunque los informes actuales se centran en las métricas de rendimiento, si se estandarizara el reporte de la energía consumida y la huella de carbono producida se podrían conseguir más incentivos para lograr reducir el impacto medioambiental causado.
- **Análisis del coste-beneficio**: Gracias a la inclusión de esta clase de métricas en los informes de ML es posible llevar a cabo el análisis de coste-beneficio, el cual sería imposible de realizar sin estos datos.

Con el fin de brindar una solución a dicho problema, este equipo presenta la herramienta experiment-impact-tracker, un *framework* liviano diseñado para efectuar de forma más fácil, consistente y precisa informes sobre el impacto energético y la huella de carbono causada por los sistemas de aprendizaje automático. Para ofrecer todo lo mencionado al usuario, esta herramienta permite medir de manera sencilla y rápida la cantidad de electricidad que utilizará un proyecto de ML, indicando cuánto significa ese consumo en emisiones de carbono [17].

Los participantes en este asunto afirman que el principal objetivo de experiment-impact-tracker es proporcionar un mecanismo comprensible, que se pueda replicar y que resulte cómodo de implementar, de modo que los informes de ML puedan contar con la capacidad de reportar resúmenes sobre el impacto de carbono originado e incluir apéndices adicionales donde se muestren métricas detalladas sobre energía, carbono y computación.

Con objeto de cumplir esta meta, los creadores de experiment-impact-tracker consideraron cinco principios fundamentales para llevar a cabo el diseño de este *framework*:

1. **Usabilidad**: Mantener la cantidad de acción requerida por parte del usuario al mínimo para facilitar el uso del *framework*.

2. **Interpretabilidad**: Las herramientas ofrecidas en el *framework* para la construcción de informes serán capaces de generar gráficas y páginas web con miras a ayudar en la interpretación. Además, se proporciona un mecanismo que traduce la huella de carbono en coste social.

3. **Extensibilidad**: Diseño del *framework* de forma modular, de tal manera que la comunidad de aprendizaje automático pueda agregar nuevas métricas, información y otras capacidades fácilmente.

4. **Reproducibilidad**: El *framework* contribuye a automatizar la reproducibilidad de los resultados obtenidos por los algoritmos. Esta acción se ejecuta mediante el registro de métricas adicionales como información del *hardware*, versiones de paquetes Python, etc.

5. **Tolerancia a fallos**: Se intenta registrar toda la información en bruto para que en caso de fallo, se pueda recrear la contabilidad y actualizarla en función de la nueva información.

En definitiva, experiment-impact-tracker consiste en una herramienta que está orientada a constituir un método simple dedicado al seguimiento y al cálculo de las emisiones derivadas del uso de energía y de la utilización informática del sistema. De esta forma, se espera aliviar a los equipos de desarrollo la carga que conlleva la creación de un informe ML que incluya métricas sobre la energía, la huella de carbono y la computación, dada su compleja medición. Con este objetivo en mente, los creadores de este *framework* animan a

[17] https://github.com/Breakend/experiment-impact-tracker

la comunidad a contribuir en la expansión de esta herramienta a través de la implementación de mejoras de diverso tipo, a fin de convertirla en una herramienta útil y globalmente funcional.

Instalación de experiment-impact-tracker

La librería de experiment-impact-tracker también se encuentra accesible a partir del repositorio de PyPI. Por consiguiente, solo es necesario ejecutar el siguiente comando para comenzar su instalación [18]:

```
(eit) root@ubuntu:~$ pip install experiment-impact-tracker
```

Aviso

A pesar de que es posible instalar experiment-impact-tracker mediante el instalador de paquetes pip, se recomienda **no utilizar este método de instalación**. Esto se debe a que la versión que reside en pip **no se encuentra actualizada** con respecto al repositorio GitHub de la herramienta, lo que significa que la versión instalada con pip **no dispone de todos los módulos**, teniendo por consecuencia que no funcione como se indica en la documentación y de lugar a errores.

Considerando el aviso, se aconseja seguir estos pasos para ser capaces de emplear la herramienta con éxito:

- **Paso 1:** Clonar el repositorio de experiment-impact-tracker:

  ```
  (eit) root@ubuntu:~$ git clone
     https://github.com/Breakend/experiment-impact-tracker.git
  ```

- **Paso 2:** Copiar la carpeta `experiment_impact_tracker` dentro de nuestro entorno de prueba:

  ```
  (eit) root@ubuntu:~$ cp -R Origen Destino
  ```

  1. `-R`: Para copiar directorios.
  2. `Origen`: `experiment-impact-tracker/experiment_impact_tracker`
  3. `Destino`: `miniconda3/envs/eit/lib/python3.10/site-packages`

[18] https://github.com/Breakend/experiment-impact-tracker

Uso de experiment-impact-tracker

Con experiment-impact-tracker solo son necesarias unas pocas líneas de código para consolidar el seguimiento de nuestro proyecto:

```
from experiment_impact_tracker.compute_tracker import ImpactTracker

tracker = ImpactTracker(<your log directory here>)
tracker.launch_impact_monitor()
#Training code
tracker.stop()
```

Código 10: Rastreador experiment-impact-tracker: objeto explícito.

Esto iniciará un proceso de Python separado que recopilará información de cómputo/energía/carbono en segundo plano.

Nota

A causa de la forma en que funciona el multiprocesamiento de Python, el proceso iniciado por la herramienta no interrumpirá al proceso principal, incluso si este falla.

Debido a ello, es posible agregar la siguiente sentencia para leer periódicamente la información más reciente del registro y verificar si han ocurrido errores durante el rastreo.

```
info = tracker.get_latest_info_and_check_for_errors()
```

Otra alternativa a la expuesta anteriormente es el uso de un gestor de contexto:

```
from experiment_impact_tracker.compute_tracker import ImpactTracker

experiment_logdir = tempfile.mkdtemp()
with ImpactTracker(experiment_logdir):
    do_something()
```

Código 11: Rastreador experiment-impact-tracker: *context manager*.

Una vez haya finalizado el rastreo, es posible acceder a la información recopilada a través de la clase **`DataInterface`** del modulo **`data_interface.py`**:

```
from experiment_impact_tracker.data_interface import DataInterface

data_interface = DataInterface([experiment_logdir])
```

```
total_power = data_interface.total_power
kg_carbon = data_interface.kg_carbon
exp_len_hours = data_interface.exp_len_hours
```

Código 12: Acceso a los datos estimados por experiment-impact-tracker.

Asimismo, el repositorio de experiment-impact-tracker dispone de un *script* llamado **generate-carbon-impact-statement**, cuya ejecución crea automáticamente una declaración de impacto de carbono para un determinado proyecto:

```
(eit) root@ubuntu:~$
    generate-carbon-impact-statement experiment_directories ”ESP”
```

Ejemplos

Finalizamos poniendo a prueba la herramienta adaptando el ejemplo anterior, el cual ha sido descrito en la sección 2.2.1:

```
from sklearn.linear_model import LogisticRegression
from experiment_impact_tracker.compute_tracker import ImpactTracker
import tempfile

log_reg = LogisticRegression()

# Create a temporary dir
tmp_dir = tempfile.mkdtemp()

with ImpactTracker(tmp_dir):
    log_reg.fit(X,y)

print(f”Please find your experiment logs in: {tmp_dir}”)
```

Código 13: Ejemplo iris: Rastreo del entrenamiento con EIT.

Si se procede a la ejecución de este código, obtenemos a la salida la información que se muestra a continuación:

```
loading region bounding boxes for computing carbon emissions region, this may take a moment...
 454/454... rate=534.25 Hz, eta=0:00:00, total=0:00:00

INFO:Gathering system info for reproducibility...
```

Figura 2.11: Resultado a la salida con EIT (Parte 1).

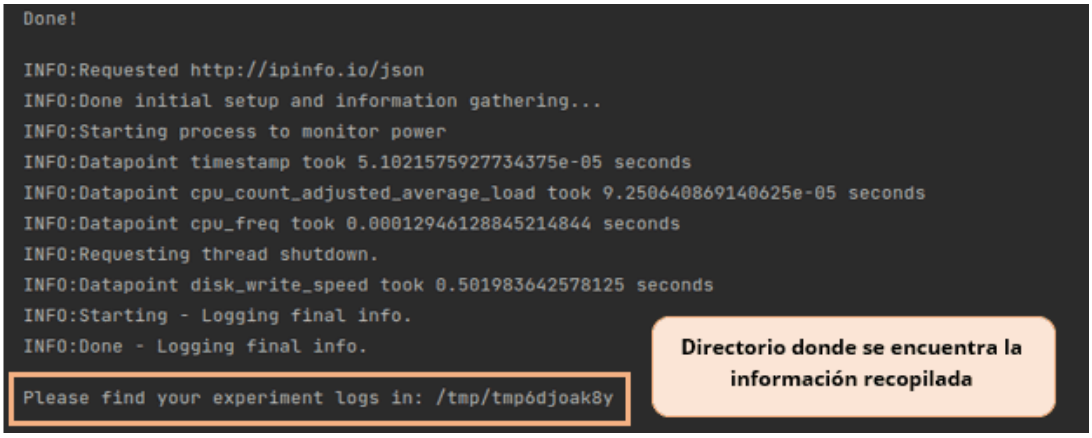


Figura 2.12: Resultado a la salida con EIT (Parte 2).

Después, iremos al directorio temporal para revisar el registro del experimento:

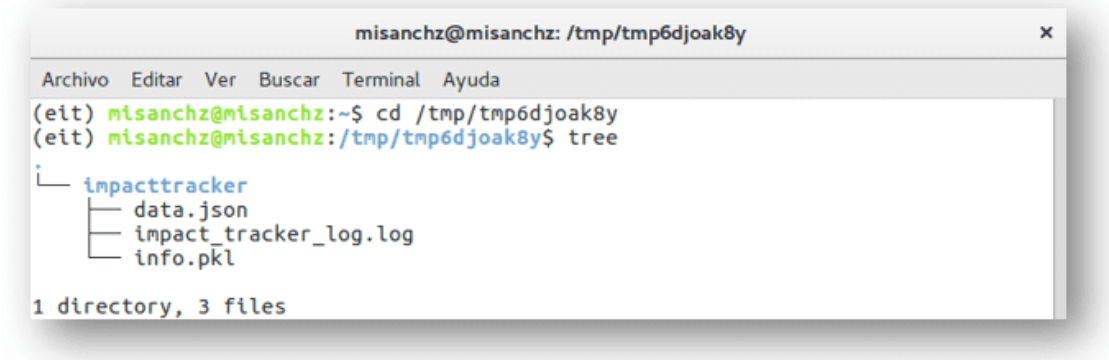


Figura 2.13: Árbol de directorios creado por experiment-impact-tracker.

Si nos adentramos en el fichero **`data.json`**, nos encontramos con la información que ha sido recopilada por EIT durante el entrenamiento del modelo:

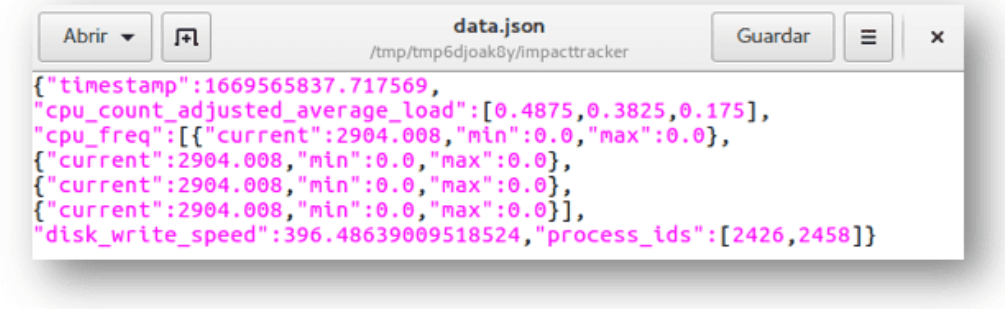


Figura 2.14: Contenido del fichero `data.json`.

A continuación, procedemos a aplicar la clase **`DataInterface`** para conocer cuál ha sido la huella de carbono y el consumo energético total de nuestro experimento:

```
from experiment_impact_tracker.data_interface import DataInterface

data_interface = DataInterface([tmp_dir])

print('kg: ', data_interface.kg_carbon)
print('total_power: ', data_interface.total_power)
print('exp_len_hours', data_interface.exp_len_hours)
```

Código 14: Ejemplo iris: Acceso a los datos estimados por EIT.

No obstante, al ejecutar dicha porción de código obtenemos el error **`ValueError: Unable to get either GPU or CPU metric`**. De igual modo, si intentamos utilizar el *script* **`generate-carbon-impact-statement`** con objeto de conocer el impacto producido por el entrenamiento de nuestro modelo, obtenemos el mismo error, debido a que también aplica **`DataInterface`** en su propio código.

Para finalizar, se deben destacar las siguientes cuestiones que han ido surgiendo durante la prueba de experiment-impact-tracker:

- Con el fin de testar esta herramienta, se han utilizado los sistemas operativos Ubuntu 22.04.1 LTS, debido a que es una de las versiones más recientes y Ubuntu 16.04.07 LTS, ya que, aunque la documentación nos indica que la herramienta ha sido probada con éxito en el sistema operativo Ubuntu 16.04.5 LTS, esta versión no fue localizada para ser instalada dentro de nuestra máquina virtual. Sin embargo, **en ambos sistemas operativos se nos han presentado los mismos problemas y errores**.
- Asimismo, se ha puesto en práctica en nuestro entorno de trabajo el ejemplo presente en el repositorio GitHub de experiment-impact-tracker, **`my_experiment.py`**. En dicho ejemplo, se entrena un modelo DL creado con Pytorch y se estima su impacto con experiment-impact-tracker. En esta ocasión, también **nos hemos topado con el mismo error** al utilizar el *script*.
- Teniendo en cuenta el error manifestado al usar la clase **`DataInterface`**, deducimos que este problema se debe principalmente a dos motivos:
  1. **Al modelo de CPU empleado en nuestra infraestructura**, ya que nuestra CPU no se encuentra entre aquellos modelos que fueron probados con éxito según la documentación.
  2. Debido a que nuestra infraestructura se trata de una **máquina virtual**, esta no dispone del directorio **`/sys/class/powercap/intel-rapl/`** donde se conservan las lecturas de energía y potencia realizadas por el procesador. Experiment-impact-tracker accede a esas métricas para calcular el gasto, por lo que es por ello que nos aparece el error mencionado.
- En último lugar, es importante resaltar que esta librería tampoco es compatible con el entorno GPU disponible en Google Colab.

2.2.3 Carbontracker

La popularidad de resolver problemas mediante el aprendizaje profundo ha aumentado rápidamente en los últimos tiempos y, con ello, la necesidad de modelos cada vez más potentes que requieren un inmenso poder computacional y precisan una alta demanda energética. En consecuencia, la tendencia exponencial de la utilización del DL en el campo de la Inteligencia Artificial supondrá un contribuyente significativo en la problemática del cambio climático. Este contexto puede convertirse en un gran inconveniente si no se empiezan a explorar medidas que consigan mejorar la eficiencia energética en el ámbito del *deep learning* [6].

A pesar de los numerosos estudios que demuestran claramente la existencia de este problema, como la investigación [34], el consumo energético y la huella de carbono asociada a la actividad efectuada en el entrenamiento y ejecución de modelos, en general, rara vez se miden debido a su complejidad. En respuesta a dicha situación, dos estudiantes del Departamento de Ciencias de la Computación de la Universidad de Copenhague, Lasse F. Wolff Anthony y Benjamin Kanding, junto con el profesor asistente Raghavendra Selvan, han desarrollado un programa *software* al que han llamado Carbontracker [12].

Carbontracker es una herramienta de código abierto escrita en Python con disponibilidad a través de PyPi, cuya función consiste en rastrear y predecir el consumo de energía y las emisiones de carbono generadas durante el proceso de entrenamiento de los modelos de DL. La herramienta se implementa como un programa multihilo, es decir, utiliza hilos separados para recopilar medidas de potencia y obtener la intensidad de carbono en tiempo real. Por tanto, no se interrumpe el entrenamiento del modelo en el hilo principal y se consigue aumentar la eficiencia. La metodología seguida es similar a la de experiment-impact-tracker (2.2.2), pero se diferencia en dos aspectos principales:

1. Se permite una mayor y más proactiva intervención enfocada a impulsar la reducción de emisiones de carbono. El entrenamiento de un modelo puede ser detenido, a discreción del usuario, si la ejecución supera un determinado coste ambiental.

2. Se admiten una variedad de entornos y plataformas diferentes tales como *clusters*, ordenadores de escritorio y cuadernos de Google Colab [19], haciendo posible la experiencia *plug-and-play* (capacitar a un dispositivo informático para ser conectado a un ordenador sin necesidad de configuración).

La filosofía de diseño que ha guiado el desarrollo de Carbontracker puede resumirse en estos principios:

1. ***Pythonic***: La mayoría del *software* en aprendizaje automático está escrito en Python. Por este motivo, se espera que esta herramienta pueda integrarse de manera sencilla en los *frameworks* Python ya disponibles.

2. **Usabilidad**: El esfuerzo requerido y el código a añadir deben ser mínimos y no perturbar la estructura del código existente.

[19] https://colab.research.google.com

3. **Extensible**: Agregar y mantener el soporte para cambiar las APIs e incorporar nuevo *hardware* debe ser sencillo.

4. **Flexible**: El usuario debe tener control total sobre lo que se supervisa y cómo se realiza esa supervisión.

5. **Rendimiento**: El impacto en el rendimiento al utilizar Carbontracker debe ser insignificante y la computación del mismo debe ser mínima. No debe afectar al entrenamiento del modelo.

6. **Interpretable**: Se tienen que aportar sistemas de conversión para facilitar la interpretación del impacto obtenido como resultado.

Los creadores de Carbontracker aspiran a que esta herramienta favorezca el reporte de la energía empleada y la huella de carbono producida durante el entrenamiento de modelos de DL. La principal meta de Carbontracker se basa en concienciar a los desarrolladores del impacto medioambiental ocasionado durante los proyectos de aprendizaje profundo y puedan, de esta forma, tomar medidas activas para reducir dicha repercusión siempre que sea posible. En último término, confían en que Carbontracker ayude a promover la responsabilidad informática en la IA y a fomentar la investigación de redes neuronales profundas que sean enérgicamente eficientes.

Instalación de Carbontracker

Al igual que se ha comentado en las herramientas anteriores, Carbontracker se puede instalar mediante el instalador de paquetes pip: [20]

```
(carbontracker) root@ubuntu:~$ pip install carbontracker
```

Uso de Carbontracker

El uso de Carbontracker solo precisa la imitación de la presente estructura sintáctica en nuestro código:

```
from carbontracker.tracker import CarbonTracker

tracker = CarbonTracker(epochs=max_epochs)
```

Código 15: Rastreador Carbontracker (Parte 1).

[20] https://github.com/lfwa/carbontracker

```
for epoch in range(max_epochs):
    tracker.epoch_start()
    # Your model training.
    tracker.epoch_end()

tracker.stop()
```

Código 16: Rastreador Carbontracker (Parte 2).

Nota

`tracker.stop()` es opcional. Su función consiste en parar el proceso de seguimiento en caso de fallo antes de que se hayan monitoreado todos los *epochs*. El propósito de este método es garantizar el informe del consumo real en cualquier circunstancia.

Seguidamente, se exponen los argumentos que pueden ser incluidos en la clase `CarbonTracker` para crear nuestro rastreador:

Tabla 2.3: Argumentos para la clase `CarbonTracker`.

Argumentos	Descripción
`epochs`	*Epochs* totales del ciclo de entrenamiento (**Obligatorio**).
`epochs_before_pred`	*Epochs* para rastrear tras retornar el consumo. (*default*=1)
`monitor_epochs`	Número total de *epochs* a monitorear. (*default*=1)
`update_interval`	Intervalo en segundos entre las mediciones. (*default*=10)
`interpretable`	Si es `True`, el CO_2eq se convierte en números interpretables (ej. distancia recorrida en automóvil). (*default*=`True`)
`stop_and_confirm`	Si es `True`, el proceso principal se pausará después de `epochs_before_pred` épocas para generar la predicción. El usuario deberá confirmar para continuar con el entrenamiento. (*default*=`False`)
`ignore_errors`	Si es `True`, los errores harán que se detenga el rastreo pero el entrenamiento continuará. (*default*=`False`)
`components`	Componentes a monitorear. (*default*=`all`)
`devices_by_pid`	Si es `True`, solo se miden los dispositivos asociados con el proceso principal. (*default*=`False`)
`log_dir`	Ruta al directorio de *logs*. (*default*=`None`)
`verbose`	Establece el nivel de verbosidad. (*default*=1)
`decimal_precision`	Precisión decimal de los valores informados. (*default*=6)

Ejemplos

- **Ejemplo 1: Entrenamiento de un modelo ML:** Continuamos con la prueba de este paquete para el caso del entrenamiento de un algoritmo de aprendizaje automático. A tal efecto, se recurrirá al mismo ejemplo aplicado en CodeCarbon (2.2.1), pero modificándolo para que sea compatible con Carbontracker:

```python
from sklearn.linear_model import LogisticRegression
from carbontracker.tracker import CarbonTracker

max_epochs = 10
tracker = CarbonTracker(epochs=max_epochs)
log_reg = LogisticRegression()

for epoch in range(max_epochs):
    tracker.epoch_start()
    log_reg.fit(X,y)
    tracker.epoch_end()

tracker.stop()
```

Código 17: Ejemplo iris ML: Rastreo del entrenamiento con Carbontracker.

La ejecución de este ejemplo provoca la salida que se percibe en la imagen:

```
CarbonTracker: The following components were found: CPU with device(s) .
CarbonTracker: WARNING - Epoch duration is too short for a measurement to be collected.
CarbonTracker:
Actual consumption for 1 epoch(s):
    Time:    0:00:00
    Energy: 0.000000 kWh
    CO2eq:   0.000000 g
    This is equivalent to:
    0.000000 km travelled by car
CarbonTracker:
Predicted consumption for 10 epoch(s):
    Time:    0:00:01
    Energy: 0.000000 kWh
    CO2eq:   0.000000 g
    This is equivalent to:
    0.000000 km travelled by car
CarbonTracker: Finished monitoring.
```

Figura 2.15: Resultado a la salida con Carbontracker (Modelo ML).

Si estudiamos la salida, contemplamos cómo Carbontracker ha generado un reporte sobre cuál ha sido el tiempo, la energía y la huella de carbono producidos a lo largo del entrenamiento para un *epoch* y para el máximo número de *epochs* que hayan sido establecidos (en este caso 10).

En ambos casos, nos damos cuenta que para estas métricas **el resultado estimado por la librería ha sido igual a 0**. Esto se debe a que **Carbontracker no fue diseñado para rastrear modelos de ML** ya que, si probamos este mismo ejemplo en Google Colab en un entorno con GPU, obtendremos el mismo resultado:

```
CarbonTracker: WARNING - Epoch duration is too short for a measurement to be collected.
CarbonTracker:
Actual consumption for 1 epoch(s):
        Time:   0:00:00
        Energy: 0.000000 kWh
        CO2eq:  0.000000 g
        This is equivalent to:
        0.000000 km travelled by car
CarbonTracker:
Predicted consumption for 10 epoch(s):
        Time:   0:00:00
        Energy: 0.000000 kWh
        CO2eq:  0.000000 g
        This is equivalent to:
        0.000000 km travelled by car
CarbonTracker: Finished monitoring.
```

Figura 2.16: Resultado con Carbontracker, Google Colab y GPU (Modelo ML).

Cabe resaltar que, si ejecutamos el ejemplo en un entorno con CPU (`None`), Carbontracker dará como salida la excepción: **`NoComponentsAvailableError: No components were available. CarbonTracker supports Intel CPUs with the RAPL interface and NVIDIA GPUs`**.

Por lo tanto, podemos concluir que nos encontramos ante la misma situación vista durante el análisis de la herramienta experiment-impact-tracker. Esto significa que **Carbontracker, para estimar el consumo realizado por la CPU, también accede a la información almacenada en el directorio `/sys/class/powercap/intel-rapl/`**, el cual no se encuentra disponible ni en nuestra máquina virtual ni en Google Colab.

- **Ejemplo 2: Entrenamiento de un modelo DL:** Para este supuesto, Carbontracker no cuenta con un ejemplo en su repositorio GitHub. Por esta razón, se ha desarrollado el mismo ejemplo utilizado en el caso de aprendizaje automático, pero con un modelo de *deep learning* creado con Tensorflow [27]:

```python
from sklearn import datasets
import tensorflow as tf
from carbontracker.tracker import CarbonTracker

# Step 1: Split the data into X and y
iris = datasets.load_iris()
X = iris["data"][:, 3:]  # petal width
y = (iris["target"] == 2).astype('int64')  # 1 if Virginica, else 0
```

Código 18: Ejemplo iris DL: Conjunto de entrenamiento.

```
# Step 2: Define a model
model = tf.keras.Sequential([
    tf.keras.layers.Dense(10, activation='relu'),
    tf.keras.layers.Dense(10, activation='relu'),
    tf.keras.layers.Dense(3, activation='softmax')
  ])

# Step 3: Compile the model
model.compile(optimizer='rmsprop',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Step 4: Train the model and track with carbontracker
max_epochs = 5
tracker = CarbonTracker(epochs=max_epochs)

# Training loop.
for epoch in range(max_epochs):
    tracker.epoch_start()
    model.fit(X, y, batch_size=50)
    tracker.epoch_end()

tracker.stop()
```

Código 19: Ejemplo iris DL: Rastreo del entrenamiento con Carbontracker.

La ejecución del nuevo ejemplo ilustrará este reporte como resultado del rastreo:

```
CarbonTracker: The following components were found: CPU with device(s) .
3/3 [==============================] - 1s 29ms/step - loss: 0.9919 - accuracy: 0.6667
CarbonTracker: WARNING - Epoch duration is too short for a measurement to be collected.
CarbonTracker:
Actual consumption for 1 epoch(s):
        Time:   0:00:01
        Energy: 0.000000 kWh
        CO2eq:  0.000000 g
        This is equivalent to:
        0.000000 km travelled by car
CarbonTracker:
Predicted consumption for 5 epoch(s):
        Time:   0:00:04
        Energy: 0.000000 kWh
        CO2eq:  0.000000 g
        This is equivalent to:
        0.000000 km travelled by car
CarbonTracker: Finished monitoring.
3/3 [==============================] - 0s 26ms/step - loss: 0.9785 - accuracy: 0.6667
3/3 [==============================] - 0s 34ms/step - loss: 0.9693 - accuracy: 0.6667
3/3 [==============================] - 0s 8ms/step - loss: 0.9612 - accuracy: 0.6667
3/3 [==============================] - 0s 14ms/step - loss: 0.9539 - accuracy: 0.6667
```

Figura 2.17: Resultado a la salida con Carbontracker (Modelo DL).

A pesar de que se está llevando a cabo el seguimiento de un modelo de aprendizaje profundo, **continuamos obteniendo 0 como resultado** en las medidas de energía y huella de carbono. Teniendo en cuenta todo lo descrito hasta ahora, deducimos que estamos ante la misma situación del ejemplo anterior. **Carbontracker no es capaz de acceder a las métricas de consumo de nuestro procesador CPU**, ya que el directorio donde deberían encontrarse los ficheros RAPL con dichas medidas no existe en el entorno de trabajo utilizado.

Si trasladamos el ejemplo a Google Colab en un entorno con GPU, comprobamos como, en este caso, sí se obtiene un resultado numérico en todas las métricas del reporte:

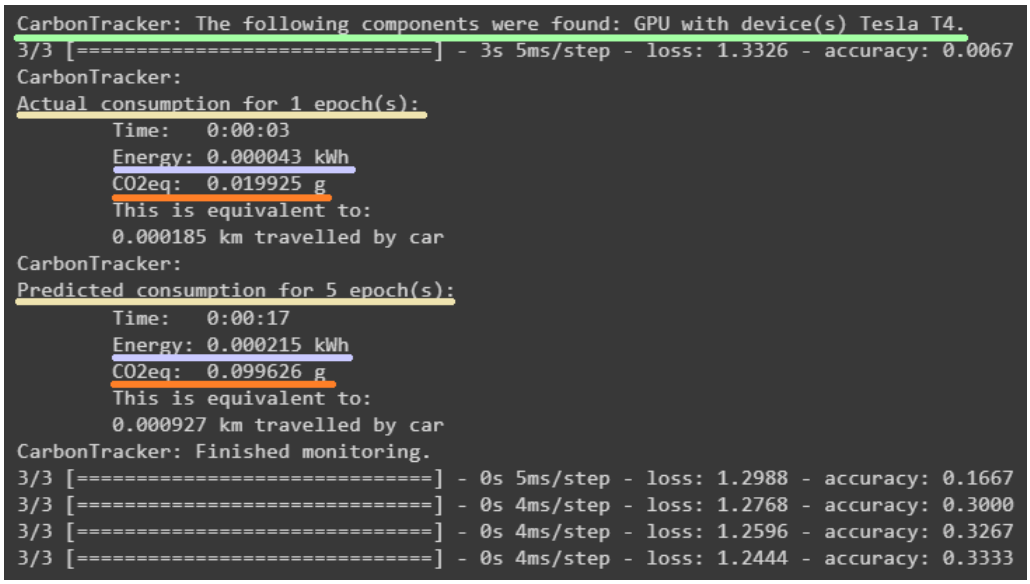


Figura 2.18: Resultado con Carbontracker, Google Colab y GPU (Modelo DL).

2.2.4 CUMULATOR

"*El coste de entrenar máquinas se está convirtiendo en un problema*". La presente afirmación constituye el título de un artículo de la revista *The Economist*, publicado en junio de 2020 [13], donde se destaca el asombroso y despreciado impacto financiero de la IA. Este coste no se limita únicamente al coste monetario, sino también al coste ecológico, cuyo origen está asociado al consumo energético necesario para el entrenamiento de estos algoritmos. Por consiguiente, con la creciente popularidad del ML y la progresiva digitalización de todos los sectores, urge la necesidad de desarrollar una mayor concienciación sobre el impacto medioambiental de estas tecnologías y de su potencial contribución al cambio climático [37].

Con ese efecto en mente, Tristan Trébaol inició un proyecto con el análisis panorámico de las herramientas, centradas en el cálculo de la huella de carbono y el consumo energético, que se encontraban disponibles hasta ese momento, como experiment-impact tracker (2.2.2). El análisis tenía por objetivo efectuar una comparativa entre todas las herramientas

y, a partir de ellas, crear una herramienta propia a la cual Trébaul llamó posteriormente CUMULATOR. Paralelamente y bajo el mismo contexto de esta herramienta, implementado dentro de la misma en forma de *script* se encuentra el Protocolo de Declaración de Carbono (*Carbon Statement Protocol*), cuya intención es ayudar a los investigadores y estudiantes a cuantificar la huella de carbono de sus trabajos.

Para complementar la exposición anterior, CUMULATOR es una API de código abierto que cuantifica e informa de la huella de carbono producida durante la computación y comunicación de los métodos de aprendizaje automático. Asimismo, CUMULATOR manifiesta las siguientes características:

- **Alcance operativo y accesibilidad**: CUMULATOR puede codificarse en muchos lenguajes de programación. Sin embargo, Python fue el lenguaje escogido debido a que es el que mejor se adapta a los requisitos de accesibilidad y de rango operativo.
- **Disponible para uso abierto**: CUMULATOR es un paquete almacenado en PyPi y está disponible para todo tipo de público.
- **Abierto a contribuciones**: El código fuente de CUMULATOR se almacena en GitHub. Por esta razón, tan pronto como se tienen derechos de acceso, se puede crear una rama propia en el repositorio para llevar adelante las aportaciones. De ahí que contribuir a la actualización de la herramienta sea muy sencillo.

CUMULATOR ofrece una aplicación web para estimar automáticamente la precisión y el consumo de energía de cuatro algoritmos diferentes (*linear regression*, *random forest*, *decision tree*, *neural network*), en función del conjunto de datos introducido como entrada.

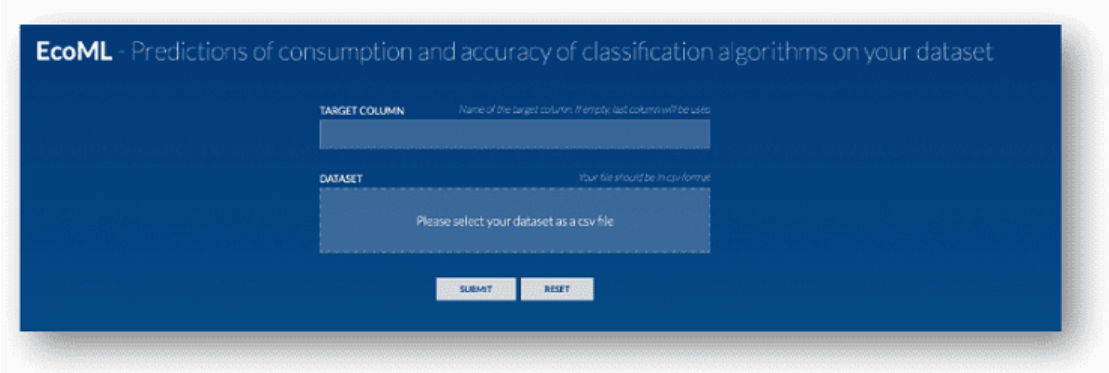


Figura 2.19: Aplicación web de CUMULATOR [21].

Las opciones de diseño escogidas para el desarrollo de CUMULATOR se resumen en la tabla 2.4. El estudio sobre dichas herramientas puso en evidencia la incapacidad de estas para estimar el impacto ecológico procedente de las comunicaciones realizadas por las aplicaciones ML alojadas en la nube.

Es por ello, como se aprecia en la tabla mencionada, que lo anterior se ha tenido en cuenta a la hora de diseñar CUMULATOR con la intención de subsanar dicho problema.

[21] https://github.com/epfl-iglobalhealth/cumulator

En cuanto a la evaluación del consumo energético, para mejorar esta nueva herramienta con respecto a otras como experiment-impact-tracker, se pretende que CUMULATOR sea compatible con las interfaces RAPL (*Running Average Power Limit*) y AMP (*Application Power Management*), debido a que los dispositivos con interfaz RAPL pueden llegar a ser un recurso limitado para los usuarios.

Tabla 2.4: Decisiones de diseño de CUMULATOR.

Variables	Tiempo (opcional: valores de referencia para el tipo de *hardware*, energía por byte del modelo e intensidad de carbono).
Costes computacionales	Rastreador basado en el tiempo.
Costes de comunicación	Se utiliza *Shift Project* para evaluar el impacto medioambiental causado por el tráfico de datos de aplicaciones ML alojadas en la nube (*1byte energy model*).
Precisión de conversión	Estimación aproximada (*Rough estimation*).
Formato	Paquete PyPi.
Alta Compatibilidad	Se necesita únicamente la librería estándar de Python.
Integración simple dentro de proyectos ML	Puede ser instalado usando el sistema de gestión de paquetes pip.
Abierto a contribuciones	Sí (GitHub: iGH (*intelligent Global Health*)).
Estreno	2020.

El desarrollador de CUMULATOR y las personas que han colaborado en el mencionado estudio esperan que esta herramienta y el protocolo *Carbon Statement Protocol* logren cumplir el sucesivo conjunto de objetivos:

1. Crear conciencia sobre la huella de carbono de las tecnologías digitales.
2. Fomentar una mayor optimización de los métodos de ML, junto al uso racional de las herramientas impulsadas por la IA. Todo ello con el propósito de reducir las emisiones procedentes de estas fuentes.

Instalación de CUMULATOR

CUMULATOR, del mismo modo que las librerías descritas previamente, se puede instalar a través de la ejecución del comando pip [22]:

[22] https://github.com/epfl-iglobalhealth/cumulator

```
(cumulator) root@ubuntu:~$ pip install cumulator
```

Adicionalmente, es necesario instalar las siguientes dependencias para el correcto funcionamiento de CUMULATOR:

```
geocoder
geopy
GPUtil
py-cpuinfo
```

Aviso

Una vez terminado el proceso de instalación, si se procede a utilizar CUMULATOR nos aparecerá el error **`ModuleNotFoundError: No module named 'src'`**.

El motivo detrás de este fallo de ejecución se encuentra en que la herramienta dentro de su repositorio GitHub se sitúa en una carpeta llamada `src`, a diferencia de la versión instalada con pip.

Para solucionar el problema de manera sencilla y rápida, solo es necesario insertar la librería en el interior de un directorio de nombre `src`, imitando la estructura de su repositorio ejecutando estos comandos:

- **Paso 1:** Crear directorio `src`:

```
(cumulator) root@ubuntu:~$ mkdir path/src
```

- **Paso 2:** Desplazar el directorio `cumulator` dentro de `src`:

```
(cumulator) root@ubuntu:~$ mv  path/cumulator path/src
```

Nota

El parámetro **`path`** hace referencia a la ruta donde se localiza el directorio que contiene las librerías instaladas del entorno conda que se esté empleando. En nuestro caso, `miniconda3/envs/cumulator/lib/python3.10/site-packages`.

Uso de CUMULATOR

CUMULATOR presenta dos opciones al usuario para calcular la huella de carbono causada por la actividad computacional de nuestra infraestructura:

- **Opción 1:** Este método consiste en activar y desactivar el cronómetro usando **`cumulator.on()`** y **`cumulator.off()`** en cada iteración del entrenamiento de un algoritmo:

```python
from src.cumulator import base

cumulator = base.Cumulator()

# Training loop.
for epoch in range(max_epochs):
    cumulator.on()
    # Your model training
    cumulator.off()
```

Código 20: Rastreador CUMULATOR, opción 1.

- **Opción 2:** La segunda opción se basa en rastrear automáticamente los costes computacionales de una función genérica (entrenamiento o predicción) con **`cumulator.run(function, *args, **kwargs)`**:

```python
from src.cumulator import base

cumulator = base.Cumulator()

# without output and with keywords arguments
cumulator.run(model.fit, X=train_X, y=train_y)

# with output and without keywords arguments
y = cumulator.run(model.predict, test_X)
```

Código 21: Rastreador CUMULATOR, opción 2.

Por otro lado, CUMULATOR facilita a los desarrolladores estos métodos para visualizar los datos calculados:

- **`cumulator.computation_costs()`**: Huella de carbono en gCO_2eq generada por la actividad computacional de nuestra máquina.
- **`cumulator.communication_costs()`**: Huella de carbono en gCO_2eq debida a los costes de comunicación provocados durante el tráfico de datos ejercido entre modelos del experimento.

- `cumulator.total_carbon_footprint()`: Suma de la huella de carbono producida por los costes computacionales y de comunicación.
- `cumulator.display_carbon_footprint()`: Reporte completo sobre el impacto medioambiental del experimento.

Ejemplos

- **Ejemplo 1: Entrenamiento de un modelo ML:** Someteremos la herramienta a prueba ajustando el ejemplo aplicado en los casos anteriores para que realice la misma función mediante CUMULATOR:

```
from sklearn.linear_model import LogisticRegression
from src.cumulator import base

cumulator = base.Cumulator()
log_reg = LogisticRegression()
cumulator.run(log_reg.fit, X=X, y=y)

# Show results
cumulator.display_carbon_footprint()
```

Código 22: Ejemplo iris ML: Rastreo del entrenamiento con CUMULATOR.

Posteriormente, si emprendemos la ejecución del ejemplo con dichas modificaciones, se adquiere esta salida como resultado:

```
CPU recognized: TDP set to 65
########
Overall carbon footprint: 3.70e-04 gCO2eq
########
Carbon footprint due to computations: 3.70e-04 gCO2eq
Carbon footprint due to communications: 0.00e+00 gCO2eq
This carbon footprint is equivalent to 1.86e-07 kilograms of coal burned.
```

Figura 2.20: Resultado del rastreo de un modelo ML con CUMULATOR.

Nota

Cabe destacar que, a diferencia de las herramientas que han sido testadas con anterioridad, **CUMULATOR no brinda métricas relacionadas con el consumo y la potencia** destinados al entrenamiento.

- **Ejemplo 2: Entrenamiento de un modelo DL:** Con el fin de poner a prueba la librería en este contexto, se aprovechará el ejemplo de *deep learning* que fue explicado y empleado con Carbontracker (2.2.3). Para ello, se realizará la modificación correspondiente:

```python
from src.cumulator import base
...

# Step 4: Train the model and track with cumulator
max_epochs = 10
cumulator = base.Cumulator()

for epoch in range(max_epochs):
    cumulator.on()
    model.fit(X, y, batch_size=50)
    cumulator.off()

cumulator.display_carbon_footprint()
```

Código 23: Ejemplo iris DL: Rastreo del entrenamiento con CUMULATOR.

La ejecución de este ejemplo produce el resultado que se observa en la imagen:

```
CPU recognized: TDP set to 65
3/3 [==============================] - 1s 10ms/step - loss: 1.1165 - accuracy: 0.1867
3/3 [==============================] - 0s 15ms/step - loss: 1.1069 - accuracy: 0.3333
3/3 [==============================] - 0s 16ms/step - loss: 1.1001 - accuracy: 0.3667
3/3 [==============================] - 0s 50ms/step - loss: 1.0944 - accuracy: 0.5067
3/3 [==============================] - 0s 30ms/step - loss: 1.0893 - accuracy: 0.6533
3/3 [==============================] - 0s 5ms/step - loss: 1.0844 - accuracy: 0.6667
3/3 [==============================] - 0s 19ms/step - loss: 1.0797 - accuracy: 0.8800
3/3 [==============================] - 0s 7ms/step - loss: 1.0757 - accuracy: 0.8400
3/3 [==============================] - 0s 5ms/step - loss: 1.0710 - accuracy: 0.9267
3/3 [==============================] - 0s 20ms/step - loss: 1.0665 - accuracy: 0.9667
########
Overall carbon footprint: 5.85e-03 gCO2eq
########
Carbon footprint due to computations: 5.85e-03 gCO2eq
Carbon footprint due to communications: 0.00e+00 gCO2eq
This carbon footprint is equivalent to 1.17e-12 wind turbines running for a year.
```

Figura 2.21: Resultado del rastreo de un modelo DL con CUMULATOR.

Examinando el reporte, advertimos que **CUMULATOR nos proporciona los mismos datos y sigue el mismo método para ambos tipos de algoritmos** (ML y DL).

2.2.5 Eco2AI

En la actualidad, muchas empresas han comenzado a desarrollar sus propias estrategias ESG (*Environment, Social, and Corporate Governance*), asignando funciones y departamentos completos a dicha agenda. La agenda ESG busca que las empresas y los inversores consideren no solo los aspectos económicos, sino también el impacto ambiental y social de sus actividades. Entre los objetivos que estas empresas desean alcanzar a partir de estas

estrategias, destacan la publicación de informes anuales sobre desarrollo sostenible y la suministración de fondos adicionales para la investigación, donde se incluyen las tecnologías digitales y la IA [10].

A pesar de la creciente influencia de la agenda ESG en el mundo empresarial, en este momento no se dispone de un método de evaluación cuantitativo, transparente y objetivo sobre el cumplimiento de esta agenda por parte de las instituciones, sobre todo en lo referente a la protección del medioambiente. Este contexto representa un problema principalmente para la industria IT, ya que alrededor de un 1% de la electricidad mundial es consumida por la computación en la nube, cuya participación se encuentra continuamente en crecimiento.

No obstante, la Inteligencia Artificial y el aprendizaje automático, cuyos ámbitos comprenden una gran parte de la industria IT actual, podrían ser la solución a este desafío. Existen varias formas en las que la IA y el ML podrían mitigar los problemas ambientales y el impacto inducido por el hombre y, en última instancia, mejorar nuestra comprensión y control del entorno.

Si bien la IA puede ayudar a reducir los efectos de la crisis climática, también es crucial tener en cuenta las emisiones CO_2 que son producidas durante el entrenamiento de sus algoritmos. El impacto ecológico causado por su uso es una cuestión importante que se debe tener en consideración para estar al tanto de los posibles riesgos. Es necesario asegurar que los modelos ML sean ambientalmente sostenibles y que se optimicen no solo en términos de precisión de la predicción, sino también en cuanto al consumo de energía e impacto ambiental. Con esto en mente, se propone el siguiente gráfico para resumir las tesis anteriores y describir la relación que existe entre la IA y los objetivos de sostenibilidad:

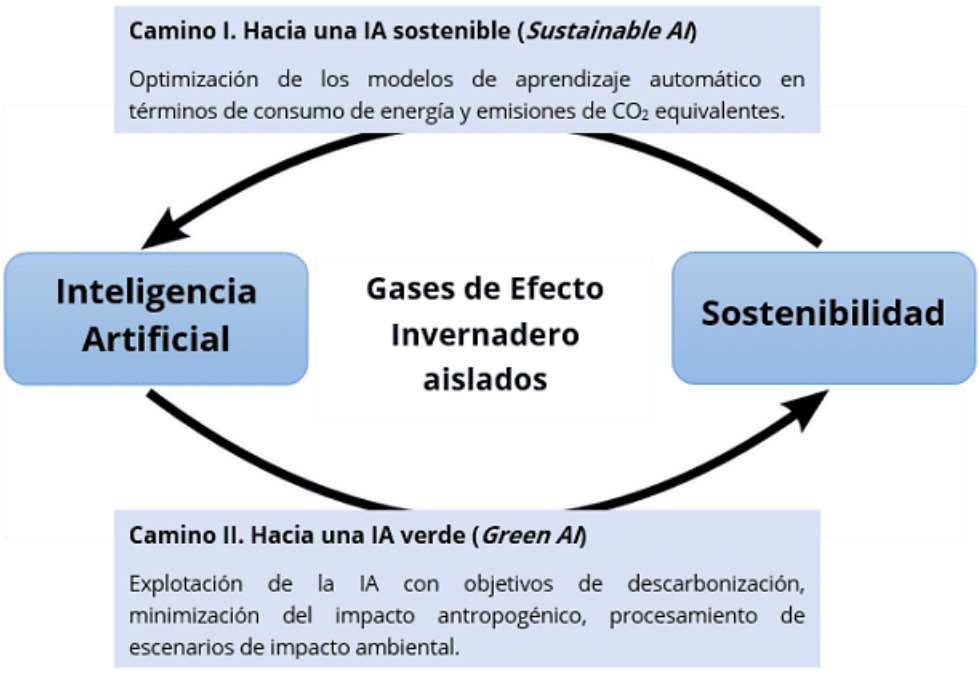


Figura 2.22: Relación entre la IA y los objetivos de sostenibilidad.

En resumen, aunque la IA posee un gran potencial para resolver problemas ambientales, esta en sí misma puede ser una fuente de huella de carbono indirecta. Con motivo de ello, el equipo de desarrolladores formado por Semen Budennyy, Vladimir Lazarev, Nikita Zakharenko y otros personajes importantes expertos en la materia, presentan la herramienta eco2AI.

Eco2AI se trata de una librería de código abierto, disponible como paquete Python, capaz de rastrear las emisiones de carbono equivalentes mientras se realiza el proceso de entrenamiento o inferencia de modelos basados en Python. Para llevar a cabo esta función y estimar correctamente el resultado, eco2AI tiene en cuenta el consumo de energía efectuado por la CPU, GPU y dispositivos RAM. Asimismo, cabe resaltar que, durante el diseño de esta herramienta, los creadores pusieron especial énfasis en que fuera capaz de ofrecer una gran precisión del seguimiento del consumo de energía y de la contabilidad correcta de las emisiones regionales de CO_2. Lo previo es gracias a la medición precisa de la carga del proceso y a la extensa base de datos con los coeficientes de emisión regionales y los dispositivos de CPU, a la cual tiene acceso.

Para comprender el rol de la librería eco2AI en el ciclo expuesto en 2.22, se muestra la próxima ilustración:

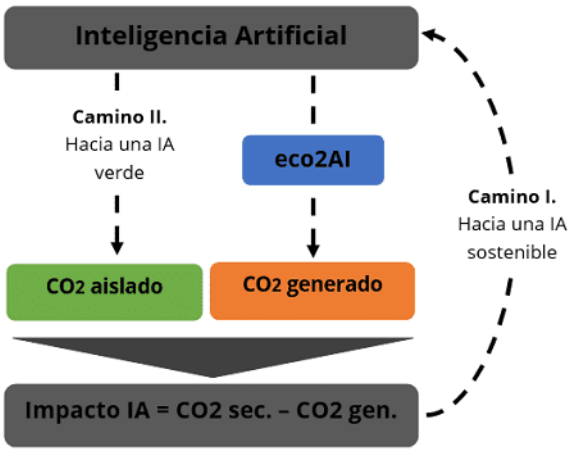


Figura 2.23: Rol de eco2AI en 2.22.

1. Primero, eco2AI motiva a optimizar la propia tecnología de IA.
2. En segundo lugar, si la IA está dirigida a aislar los gases de efecto invernadero (GEI), entonces el efecto total debe evaluarse teniendo en cuenta el equivalente generado. Es decir, debe calcularse como la resta entre las emisiones que se han logrado aislar con la IA (CO_2sec) y las emisiones generadas por esta durante el proceso (CO_2gen).

Los integrantes de la unidad de desarrollo de eco2AI desean que esta herramienta satisfaga una serie de objetivos:

- Ayudar a los científicos de datos e investigadores a rastrear el consumo de energía y emisiones de carbono causadas por sus modelos de una manera sencilla.
- Brindar a los usuarios una comprensión clara del impacto ecológico de la IA, con el fin de motivar y alentar a la comunidad científica a buscar nuevas arquitecturas óptimas que consuman menos recursos informáticos.
- Promover la investigación de algoritmos más eficientes desde el punto de vista computacional.

Instalación de eco2AI

La instalación de la librería eco2AI solo requiere de la ejecución del próximo comando a través del terminal [23]:

```
(eco2ai) root@ubuntu:~$ pip install eco2ai
```

Nota

Eco2AI es incompatible con la versión más actual del paquete pandas (1.5.2). Para ser capaces de emplear esta librería, es necesario que la versión de pandas sea menor o igual a 1.4.3, o mayor o igual a 1.4.0 ($1.4.0 \leq pandas \leq 1.4.3$).

Uso de eco2AI

La interfaz de eco2AI es bastante simple, siendo esta su forma de uso más fácil:

```
import eco2ai

tracker = eco2ai.Tracker(project_name="YourProjectName",
                         experiment_description="YourExperimentDescription")
tracker.start()
#<your gpu &(or) cpu calculations>
tracker.stop()
```

Código 24: Rastreador eco2AI: objeto explícito.

También, permite usar un **decorador** sobre nuestras funciones de entrenamiento:

```
from eco2ai import track

@track
def train_func(model, dataset, optimizer, epochs):
    ...
```

Código 25: Rastreador eco2AI: *decorator*.

Para aumentar la comodidad de sus usuarios, eco2AI garantiza que **cada vez que se cree una instancia del objeto `Tracker` con sus parámetros personalizados, esta configuración se guardará hasta que se elimine la librería**. De esta manera, cada nuevo rastreador se creará con su configuración personalizada hasta la creación de un rastreador con nuevos parámetros (estos reemplazarán los antiguos):

[23] https://github.com/sb-ai-lab/Eco2AI

```
import eco2ai

tracker = eco2ai.Tracker(project_name="YourProjectName",
                         experiment_description="training model",
                         file_name="emission.csv")
tracker.start()
#<your gpu &(or) cpu calculations>
tracker.stop()

tracker = eco2ai.Tracker()
# tracker = eco2ai.Tracker(project_name="YourProjectName",
#     experiment_description="training the <your model> model",
#     file_name="emission.csv")

tracker.start()
#<your gpu &(or) cpu calculations>
tracker.stop()
```

Código 26: Reutilización del rastreador eco2AI.

Nota

Para lograr un cálculo correcto del consumo de energía:

- Se debe crear un objeto **`Tracker`** antes de cualquier uso de la CPU o la GPU.
- Es necesario crear un nuevo objeto **`Tracker`** para cada nuevo cálculo.

Asimismo, pueden establecerse previamente los parámetros del rastreador con la función **`set_params()`**:

```
from eco2ai import set_params, Tracker

set_params(project_name="My_default_project_name",
           experiment_description="We trained...",
           file_name="my_emission_file.csv")

tracker = Tracker()
# tracker = Tracker(project_name="My_default_project_name",
#     experiment_description="We trained...",
#     file_name="my_emission_file.csv")

tracker.start()
#<your gpu &(or) cpu calculations>
tracker.stop()
```

Código 27: Uso de la función `set_params()`.

Dentro de la clase **`Tracker()`** se pueden integrar todos estos parámetros con el propósito de personalizar, según nuestras necesidades, el objeto rastreador:

Tabla 2.5: Parámetros para la clase `Tracker`.

Parámetros	Descripción
`project_name`	Nombre del proyecto (*default*: `None`).
`experiment_description`	Descripción del experimento (*default*: `None`).
`file_name`	Nombre del archivo donde se guardan los resultados (*default*: `emission.csv`).
`measure_period`	Período de mediciones del consumo de energía en segundos (*default*: 10).
`emission_level`	La masa de CO_2 en kilos que se produce por cada MWh de energía consumida (*default*: `None`).
`alpha_2_code`	Código ISO del país (*default*: `None`).
`region`	Región/estado/distrito del país (*default*: `None`).
`cpu_processes`	**`current`** para calcular el uso de la CPU solo para el proceso de ejecución actual y **`all`** para calcular el uso total de la CPU.
`pue`	Eficiencia de utilización de energía (*default*: 1).
`encode_file`	Si es `True`, la información se registra de forma encriptada (*default*: `None`).
`electricity_pricing`	Diccionario con intervalos de tiempo como claves y precio de la electricidad durante esos intervalos como valores.
`ignore_warnings`	Si es `True`, el usuario no será notificado de todos los avisos (*default*: `False`).

Finalmente, eco2AI implementa la función **`summary()`**. Esta función realiza un resumen del archivo csv especificado con la duración, el consumo y las emisiones de CO_2 para cada proyecto y para el total de todos ellos.

```
# Obtain summary report from emission.csv file
eco2ai.summary("emission.csv", kwh_price=0.117)
```

Si el usuario define el argumento **`kWh_price`**, la información sobre los costes financieros para cada uno de los proyectos se calcularán adicionalmente en función de la duración y de los precios proporcionados por el usuario.

Ejemplos

Aviso

Eco2AI **funciona únicamente si esta se utiliza en Jupyter Notebook o en Google Colab**. Si se intenta emplear en un fichero Python (con extensión `.py`) nos aparecerá el siguiente error: **`ValueError: '%CPU' is not in list`**. Este no se mostrará y la herramienta funcionará correctamente si se hace uso de los entornos mencionados.

- **Ejemplo 1: Entrenamiento de un modelo ML:** Con la intención de poner en funcionamiento la librería, en este supuesto, se efectuarán los cambios que se especifican a continuación en el código:

```
import eco2ai
from sklearn.linear_model import LogisticRegression

log_reg = LogisticRegression()
tracker = eco2ai.Tracker(
    project_name="Eco2AI_Test_ML",
    experiment_description="training logistic regression model")

tracker.start()
log_reg.fit(X,y)
tracker.stop()
```

Código 28: Ejemplo iris ML: Rastreo del entrenamiento con eco2AI.

La ejecución del código expuesto nos devuelve esta salida:

```
/home/misanchz/miniconda3/envs/TFG/lib/python3.10/site-packages/eco2ai/emission_track.py:135:
UserWarning:
If you use a VPN, you may have problems with identifying your country by IP.
It is recommended to disable VPN or
manually install the ISO-Alpha-2 code of your country during initialization of the Tracker() class.
You can find the ISO-Alpha-2 code of your country here: https://www.iban.com/country-codes

  warnings.warn(
/home/misanchz/miniconda3/envs/TFG/lib/python3.10/site-packages/eco2ai/tools/tools_gpu.py:39:
NoGPUWarning:

There is no any available GPU devices or your GPU is not supported by Nvidia library!
The tracker will consider CPU usage only

  warnings.warn(message="\n\nThere is no any available GPU devices or your GPU is not supported by
  Nvidia library!\nThe tracker will consider CPU usage only\n",
```

Figura 2.24: Resultado del rastreo de un modelo ML con eco2AI.

Si analizamos la imagen, vemos que nos retorna dos *warnings*. Con eco2AI podemos evitar la aparición de estos avisos estableciendo **`ignore_warning`** a **`True`** durante la creación de nuestro rastreador con la clase `Tracker()`.

Adicionalmente, siguiendo el mismo comportamiento observado en la herramienta CodeCarbon (2.2.1), **eco2AI genera un archivo csv** en nuestro directorio llamado `emission.csv` con todos los cálculos efectuados:

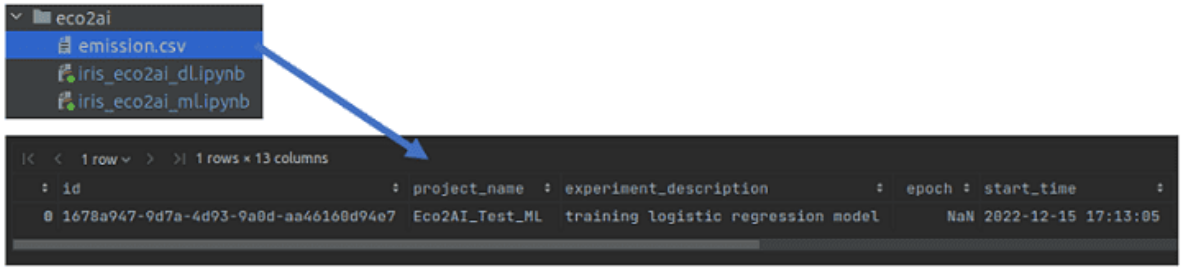


Figura 2.25: Fichero resultado, `emission.csv`.

Este fichero, igual que ocurría con CodeCarbon, **contiene en su interior una tabla de 13 columnas** y **tantas filas como ejecuciones se hayan llevado a cabo**. Actualmente este fichero solo almacena una fila, ya que simplemente se ha realizado una única ejecución. Las 13 columnas que conforman el archivo representan la siguiente información:

Tabla 2.6: Datos registrados para cada experimento en eco2AI.

Dato	Descripción
`id`	id de la ejecución.
`project_name`	Nombre del proyecto.
`experiment_description`	Descripción del experimento.
`epoch`	Número de *epochs*.
`start_time`	Comienzo del experimento yyyy-mm-dd hh:mm:ss.
`duration(s)`	Duración del experimento en segundos.
`power_consumption(kWh)`	Potencia total consumida (kWh).
`CO2_emissions(kg)`	Huella de carbono generada (kg).
`CPU_name`	Modelo de la CPU.
`GPU_name`	Modelo de la GPU.
`OS`	Nombre del sistema operativo.
`region/country`	Región y país donde se encuentra la infraestructura.
`cost`	Coste total de la electricidad consumida.

- **Ejemplo 2: Entrenamiento de un modelo DL:** Aunque el repositorio GitHub de eco2AI ofrece un tutorial con diferentes ejemplos de *deep learning*, se utilizará el ejemplo que ha sido aplicado anteriormente en 2.2.3 y 2.2.4 para esta ocasión, dado que dicho tutorial es demasiado extenso y complejo.

  Si configuramos el ejemplo para ser compatible con eco2AI, tendrá el aspecto que se expone a continuación:

```
import eco2ai
...

# Step 4: Train the model and track with eco2ai
max_epochs = 5
tracker = eco2ai.Tracker(project_name="Eco2AI_Test_DL",
    experiment_description="training tensorflow model",
    ignore_warnings=True)

tracker.start()
model.fit(X, y, epochs=max_epochs, batch_size=50)
tracker.stop()
```

Código 29: Ejemplo iris DL: Rastreo del entrenamiento con eco2AI.

Su ejecución, al igual que en el ejemplo 1, **agrega una nueva fila en `emission.csv`** con todos los resultados estimados por eco2AI en el transcurso de este experimento.

En última instancia, si hacemos uso de la función **`summary()`** para nuestro fichero `emission.csv`, obtenemos esta tabla como resultado:

3 rows | 3 rows × 5 columns

project_name	total duration(s)	total power_consumption(kWh)	total CO2_emissions(kg)	total electricity cost
Eco2AI_Test_DL	2.848665	8.078722e-07	1.611826e-07	NaN
Eco2AI_Test_ML	0.717891	9.116551e-09	1.818889e-09	NaN
All the projects	3.566555	8.169887e-07	1.630015e-07	NaN

Figura 2.26: Resultado tras la ejecución de `summary()`.

Cada columna brinda estos datos acerca de nuestros proyectos:

- **`total_duration(s)`**: Suma de la duración de todas las ejecuciones de un proyecto.
- **`total_power_consumption(kWh)`**: Suma de la potencia consumida durante todas las ejecuciones de un proyecto.
- **`total_CO2_emissions(kg)`**: Suma de la huella de carbono generada durante todas las ejecuciones de un proyecto.
- **`total_electricity_cost`**: Coste total de la electricidad en un proyecto.

2.2.6 Green Algorithms

El cambio climático está afectando profundamente a casi todos los aspectos de la vida en la Tierra, incluida nuestra sociedad. En el presente, varias acciones humanas son responsables de las emisiones de gases de efecto invernadero, entre las cuales, cada vez está cobrando más fuerza el impacto causado por las actividades informáticas. Los avances en este campo desempeñados en *hardware*, *software* y algoritmos han permitido a la comunidad científica progresar a un ritmo sin precedentes. No obstante, aunque se han alcanzado grandes avances científicos gracias al desarrollo de equipos de alto rendimiento computacional, el impacto medioambiental resultante ha sido subestimado [23].

Estudios anteriores como [34], [20] y [22] han logrado hacer progresos con respecto a la estimación de las emisiones GEI generadas debido a la computación. Pese a ello, los métodos de estimación expuestos en estos estudios tienen limitaciones que impiden su amplia aplicación. Por consiguiente, para favorecer el desarrollo de una informática menos contaminante y extender su uso a los usuarios, se considera urgente la necesidad de establecer una metodología general que permita calcular la huella de carbono producida por cualquier tarea computacional.

Quizás, el desafío más crucial para conseguir una informática más ecológica es convertir la estimación y el informe de las emisiones GEI en una práctica estandarizada. Para ello, como se ha indicado previamente, es imprescindible disponer de una metodología transparente y sencilla de usar que lleve a cabo estas tareas. Como propuesta, el equipo formado por Loïe Lannelongue, Jason Grealey y Michael Inouye han desarrollado una herramienta *online* y gratuita que ha sido nombrada como Green Algorithms [24].

Esta proporciona un *framework* metodológico y simple que permite al usuario estimar e informar de forma confiable y estandarizada la huella de carbono de casi cualquier tipo de computación. Por otra parte, Green Algorithms se trata de una herramienta que requiere información mínima y no interfiere con el código existente, integrándose fácilmente con los procesos informáticos que haya en ese instante ejecutándose en el equipo empleado. Además, ofrece una amplia gama de CPUs, GPUs, computación en la nube (*cloud computing*), servidores locales y máquinas de escritorio, con el fin de realizar con mayor precisión el cálculo del impacto medioambiental inducido.

Este trabajo mejora y amplía sustancialmente los *frameworks* existentes hasta ese momento para estimar la huella de carbono originada en las diversas actividades computacionales. En particular, se han integrado y formalizado parámetros previamente poco claros, como el factor de uso y el consumo de energía unitario (por núcleo o por GB de memoria). Como resultado de este hecho, la huella de carbono de un algoritmo puede dividirse en una pequeña cantidad de elementos clave, fácilmente cuantificables, como el número de núcleos, el tamaño de la memoria y el factor de uso. En consecuencia, se logra reducir la carga para el usuario debido a que, de esta manera, este no está obligado a medir manualmente el consumo energético del *hardware* o utilizar una gama limitada de proveedores de servicios en la nube. Todo lo comentado hace que el método de estimación aplicado por

[24] http://www.green-algorithms.org/

Green Algorithms sea muy flexible en comparación con los enunciados en trabajos anteriores.

Al igual que la herramienta CodeCarbon (2.2.1), Green Algorithms aporta equivalencias que se corresponden con ciertas actividades de la vida cotidiana, que pretenden ayudar a la interpretación y contextualización de la huella de carbono obtenida como resultado.

Green Algorithms se puede generalizar a casi cualquier trabajo computacional y, adicionalmente, se puede aprovechar como base para ayudar en otros aspectos de la informática ecológica. Hoy en día, la huella de carbono debida a la computación es significante, llegando incluso a intensificar los efectos del cambio climático y agravar dicho fenómeno. Ante esta situación, los fundadores de Green Algorithms tienen la expectativa de que esta nueva herramienta y las métricas ofrecidas por la misma logren generar conciencia sobre los problemas expuestos, así como aportar soluciones pragmáticas que sean capaces de disminuir el impacto medioambiental, producto de la computación moderna.

Instalación de Green Algorithms

Green Algorithms, a excepción de las herramientas que han sido estudiadas a lo largo de este capítulo, **no requiere de instalación**, ya que no se trata de una librería Python sino de una página web. Por lo tanto, solo es necesario acceder a la dirección **http://calculator.green-algorithms.org/** para comenzar con su utilización.

Uso de Green Algorithms

El uso de Green Algorithms es realmente sencillo, solo necesitamos introducir los datos que nos solicita la página en la sección *Details about your algorithm* situada a la izquierda. Por ejemplo:

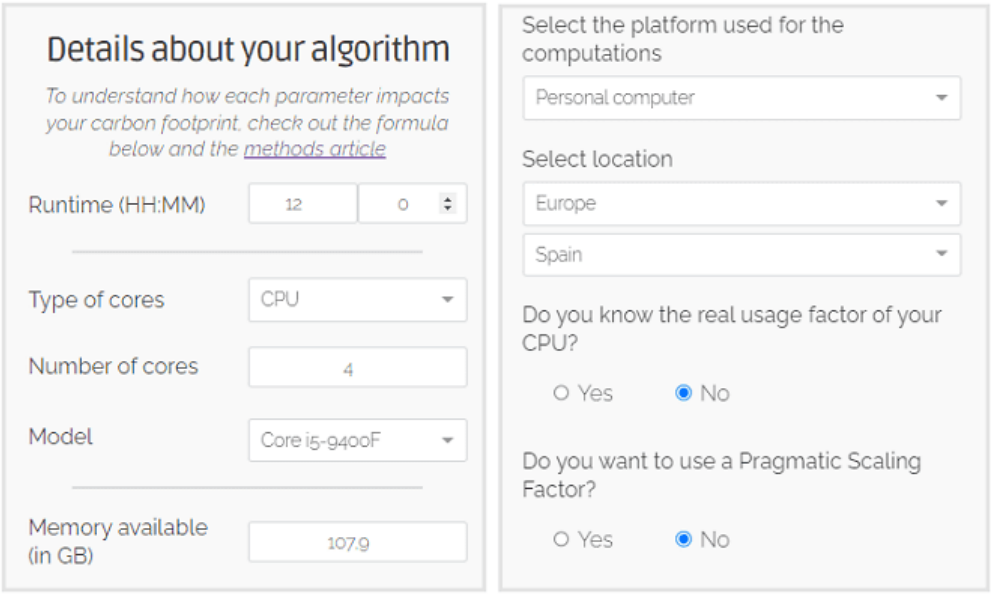


Figura 2.27: Datos introducidos en Green Algorithms.

Nota

El término *Pragmatic Scaling Factor* se emplea para **perfeccionar el método de estimación aplicado por Green Algorithms**. Este concepto permite estimaciones empíricas en relación con las **ejecuciones repetidas para una tarea en particular**, como ajuste de parámetros, pruebas y solución de errores [23].

Una vez ingresada toda la información expuesta en 2.27, la sección localizada a la derecha se actualiza automáticamente mostrando los resultados correspondientes:

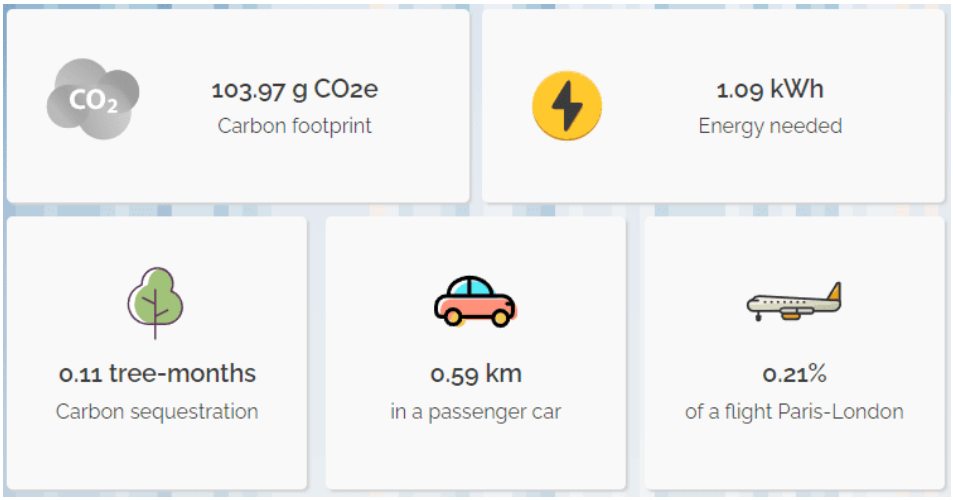


Figura 2.28: Resultado ofrecido por Green Algorithms.

El informe creado por Green Algorithms en base a los datos introducidos muestra la presente información:

1. La huella de carbono expresada en gCO_2eq que sería generada durante las condiciones indicadas.

2. La energía en kWh requerida para ejecutar dicha tarea.

3. Equivalencias similares a las aportadas por la librería CodeCarbon para facilitar la interpretación de los datos anteriores.

2.3 Conclusiones

Con objeto de finalizar este capítulo, se expondrá la siguiente tabla (2.7), cuya misión consiste en comparar, según las características observadas a lo largo del análisis practicado en las secciones anteriores, las herramientas orientadas a la estimación de la huella de carbono causada por los algoritmos de Inteligencia Artificial:

Tabla 2.7: Tabla comparativa de las herramientas según sus características [10].

	Code Carbon	EIT	Carbon tracker	CUMU LATOR	Eco2AI	Green Algorithms
Sistemas Operativos:						
• Linux	✓	✓	✓	✓	✓	✓
• Windows	✓	✗	✓	✓	✓	✓
Algoritmos:						
• ML	✓	✓	✗	✓	✓	✓
• DL	✓	✓	✓	✓	✓	✓
Resultados:						
• Potencia	✓	✓	✗	✗	✓	✗
• Energía	✓	✗	✓	✗	✗	✓
• CO_2eq	✓	✓	✓	✓	✓	✓
• Tiempo	✓	✓	✓	✗	✓	✗
• Equivalencias	✓	✗	✓	✓	✗	✓
• Fichero	✓	✗	✗	✗	✓	✗
Extensiones:						
• `.py`	✓	✓	✓	✓	✗	✗
• `.ipynb`	✓	✓	✓	✓	✓	✗
Sintaxis:						
• Objeto	✓	✓	✓	✓	✓	✗
• *Decorator*	✓	✗	✗	✗	✓	✗
• *Context Manager*	✓	✓	✗	✗	✗	✗
Interfaz web	✓	✗	✗	✓	✗	✓
Máquina Virtual	✓	✗	✗	✓	✓	✓
Google Colab:	✓	✗	✓	✗	✓	✗
Instalación	Fácil	Difícil	Fácil	Difícil	Fácil	Fácil
Utilización	Fácil	Difícil	Media	Media	Fácil	Fácil

Una vez completado el estudio de las herramientas y creada la tabla donde se realiza su comparación en base a sus características, se extrae, a partir de la información recopilada, la próxima sucesión de conclusiones de cada una de ellas:

Tabla 2.8: Conclusiones iniciales de cada herramienta (Parte 1).

Herramientas	Conclusiones
CodeCarbon	• Se trata de la herramienta más completa para la estimación de la huella de carbono causada por nuestros modelos. • Es la herramienta más fácil y cómoda de utilizar, debido a que es la librería que más opciones de uso ofrece a los usuarios. • Asimismo, se trata de la librería que más datos calcula y aporta como resultado en un formato fácil y sencillo de interpretar. • Es flexible en cuanto a infraestructura, ya que no se limita al uso de la interfaz RAPL para obtener las métricas de consumo. • Aunque solo nos hayamos enfocado en la tarea de estimar la huella de carbono, CodeCarbon brinda varias funciones adicionales, como la monitorización de nuestra máquina.
Experiment-impact-tracker	• La versión del gestor de paquetes pip se encuentra desactualizada con la del repositorio GitHub. Este hecho provoca que no pueda ser aplicada en Google Colab y que su instalación y uso sea muy confuso. • Esta librería requiere que nuestro procesador CPU disponga de una interfaz RAPL para su correcto funcionamiento, lo que hace que no sea ajustable a todo tipo de infraestructuras. • Tras la ejecución de un experimento, la librería crea un directorio con una serie de ficheros cuyo contenido es difícil de interpretar. Por ello, en comparación con el resto, sus resultados son los más complejos de visualizar y descifrar.
Carbontracker	• Es la librería más restrictiva en lo que respecta al tipo de algoritmo ejecutado, dado que solo admite modelos de DL. • Al igual que sucedía con experiment-impact-tracker, su correcto desempeño depende de que nuestra CPU tenga acceso a la interfaz RAPL, limitando el equipo de trabajo que es posible emplear. • Visto el punto anterior, solo es posible su uso en Google Colab y en Windows, concretamente en entornos de ejecución con GPU(s). • Después de CodeCarbon, Carbontracker es la librería que más información presenta a los usuarios como resultado del rastreo.

Tabla 2.9: Conclusiones iniciales de cada herramienta (Parte 2).

Herramientas	Conclusiones
CUMULATOR	• Su instalación resulta complicada a causa de que la versión disponible en su repositorio GitHub no coincide con la instalada con pip, lo que hace que no sea compatible con Google Colab. • A diferencia de sus semejantes, CUMULATOR requiere de la instalación de varias librerías por separado para actuar adecuadamente. • CUMULATOR es la que menos métricas proporciona y estima durante la ejecución de un experimento. Esto dificulta que pueda ser empleada para una posterior equiparación con el resto de herramientas.
Eco2AI	• Por detrás de CodeCarbon, asumimos que eco2AI es la herramienta más práctica y sencilla de utilizar para la evaluación del impacto medioambiental originado por los algoritmos ejecutados. • No obstante, cuenta con la desventaja de haber sido configurada para funcionar únicamente en Jupyter Notebook, siendo la herramienta menos flexible en lo que a extensiones Python se refiere. • Por otra parte, no ha sido diseñada para operar con la última versión de `pandas`, la cual constituye una de las librerías más importantes y utilizadas por los científicos de datos.
Green Algorithms	• De todas las candidatas, al ser una página web que no requiere de conocimientos previos de programación y a la que se puede acceder desde cualquier equipo sin necesidad de instalación, se trata de la herramienta más sencilla de todas, pudiendo ser utilizada por todo tipo de usuarios. • A pesar de ello, al no tratarse de una librería Python como el resto, su método de estimación no se realiza durante la ejecución del experimento, sino a partir de los datos introducidos por el usuario. Esto conlleva que su cálculo no se base ni en el código, ni en el tipo de algoritmo aplicado. Por este motivo, creemos que esta herramienta no puede ser partícipe de una comparación posterior.

Por lo tanto, en función de lo expuesto en este capítulo, se concluye que las mejores herramientas para la evaluación del impacto medioambiental en los algoritmos, cuyas características son las más óptimas para ser comparadas en el *benchmarking* que se desarrollará en los capítulos posteriores, son **CodeCarbon** y **eco2AI**.

Capítulo 3

Diseño e implementación

En el capítulo 3 se expondrá la arquitectura del *benchmarking* que será aplicada en el siguiente capítulo para proceder a la comparación y obtención de resultados. Adicionalmente, se describirán en detalle todos los parámetros que configuran dicha estructura y cuales han sido todas las decisiones de diseño, metodologías y elementos escogidos para efectuar el experimento.

3.1 Arquitectura general

La arquitectura establecida para implementar el banco de pruebas sobre las herramientas seleccionadas tiene el aspecto que se muestra en el esquema presentado:

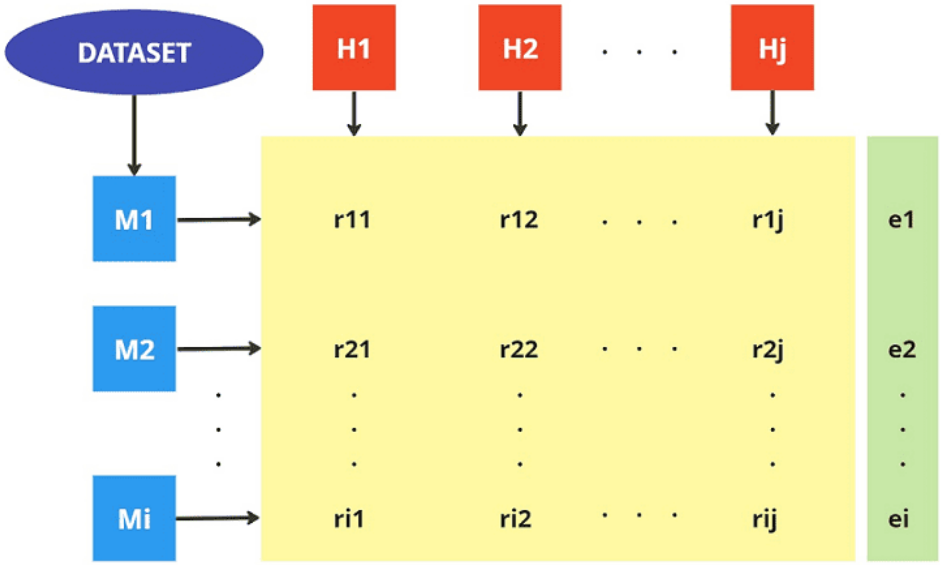


Figura 3.1: Arquitectura general del *benchmarking*.

Si analizamos la estructura definida en el gráfico, se observan que estos son los distintos parámetros que toman parte en su configuración:

- M_i: Modelos escogidos para representar a las distintas familias en *machine learning*.
- *Dataset*: Conjunto de datos que será empleado en el entrenamiento de los algoritmos.
- H_j: Herramientas de estimación del consumo en ML que participarán en el *benchmarking* con el fin de ser comparadas.
- r_{ij}: Resultado calculado por la herramienta *j* para un determinado modelo (i) y conjunto de datos.
- e_i: Métricas de evaluación con el propósito de calcular la eficiencia de los algoritmos seleccionados.

3.2 Parámetros establecidos

En esta sección se indicarán cuáles han sido las opciones escogidas y las metodologías adoptadas para cada uno de los parámetros que intervienen en la arquitectura general del *benchmarking* (3.1) que deseamos construir.

3.2.1 Modelos (Mj)

A continuación, se procede a resumir en qué consisten y cuáles son las características básicas de cada uno de los algoritmos que han sido escogidos para comparar las librerías especificadas:

Logistic Regression

La regresión logística constituye un modelo de aprendizaje automático que se usa comúnmente para **estimar la probabilidad de que una entidad pertenezca a una clase en particular** [18].

Si la probabilidad estimada es superior al 50%, entonces el modelo predice que dicha entidad pertenece a esa clase (llamada clase positiva y etiquetada como "1") y, en caso contrario, predice que no (llamada clase negativa y etiquetada como "0"). Estas características convierten a este modelo en un **clasificador binario**.

La regresión logística, del mismo modo que el algoritmo de regresión lineal, hace la predicción calculando la suma de las características de entrada con una constante llamada término de sesgo, como se muestra a continuación:

$$\widehat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + ... + \theta_n x_n$$

Modelo de predicción de la regresión lineal (3.1)

- $\widehat{y}$: Valor predicho por el modelo.
- n: Número de características.
- x_i: Valor de la característica *i*.
- θ_j: Parámetro *j* del modelo (incluyendo el término de sesgo θ_0 y los pesos de cada característica $\theta_1, \theta_2, ..., \theta_n$).

$$\widehat{p} = h_\theta(x) = \vec{\theta} \,.\, \vec{x}$$

Modelo de predicción de la regresión lineal (forma vectorizada) (3.2)

- $\vec{\theta}$: Vector de parámetros del modelo. Este vector contiene el término de sesgo (θ_0) y los pesos de las distintas características ($\theta_1, \theta_2, ..., \theta_n$).
- $\vec{x}$: Vector de características que va desde x_0 hasta x_n, con $x_0 = 1$.
- $\vec{\theta} \,.\, \vec{x}$: Producto escalar entre $\vec{\theta}$ y $\vec{x}$.

Sin embargo, en lugar de calcular un resultado directamente ($\widehat{y}$) como lo hace la regresión lineal, primero **genera la logística de dicho resultado**:

$$\widehat{p} = h_\theta(x) = \sigma(\vec{\theta} \,.\, \vec{x})$$

Probabilidad estimada por el modelo de regresión logística (forma vectorizada) (3.3)

La logística, denominada $\sigma(t)$, es una función sigmoide (es decir, con forma de S) que produce un número entre 0 y 1 y tiene esta forma:

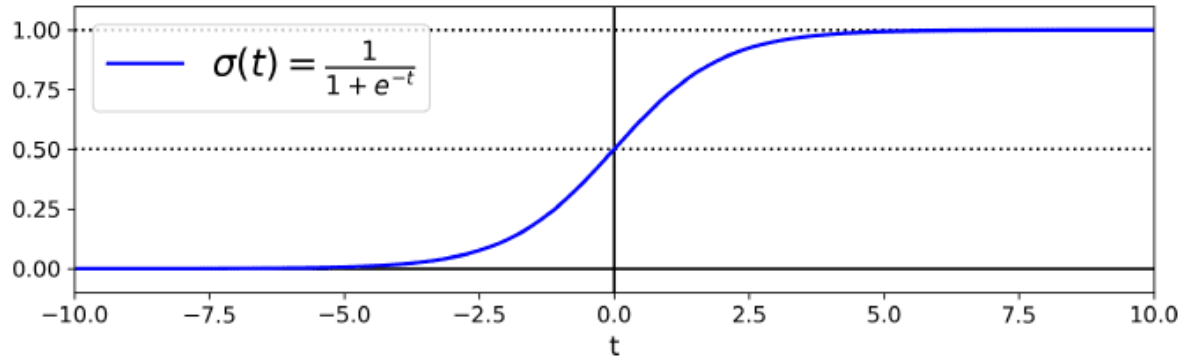


Figura 3.2: Función logística [18].

Si se estudia la función, observamos que $\sigma(t) < 0,5$ cuando $t < 0$ y $\sigma(t) \geq 0,5$ cuando $t \geq 0$. Por lo tanto, este algoritmo predice "1" cuando $\vec{\theta} \,.\, \vec{x}$ es positivo y "0" cuando es negativo.

Una vez que el modelo de regresión logística ha estimado la probabilidad ($\widehat{p}$) de que una entidad x pertenezca a la clase positiva, puede hacer su predicción $\widehat{y}$:

$$\widehat{y} = \begin{cases} 0 & \text{si } \widehat{p} < 0,5 \\ 1 & \text{si } \widehat{p} \geq 0,5 \end{cases}$$

Modelo de predicción de la regresión logística (3.4)

Random Forests

Los bosques aleatorios (*random forests*) son considerados como uno de los algoritmos de aprendizaje automático más potentes, siendo de los más utilizados debido a su precisión, simplicidad y flexibilidad. Este modelo cae dentro del grupo de técnicas de aprendizaje supervisado y es, sin duda, uno de los más populares para ser aplicado tanto para fines de regresión, como de clasificación.

El algoritmo *random forest* se basa en la técnica de ***ensemble learning***. Esta técnica consiste en apilar muchos modelos y combinar sus resultados para mejorar el rendimiento.

Asimismo, los bosques aleatorios también pueden ser definidos como una mejora del algoritmo conocido como *decision tree*, ya que estos no son más que **un conjunto de árboles de decisión** [3].

El árbol de decisión también es un modelo de *machine learning* supervisado y no paramétrico, que puede llevar a cabo trabajos de regresión, clasificación y tareas multisalida. Todas estas características lo convierten, al igual que los bosques aleatorios, en uno de los algoritmos más eficaces y versátiles, siendo capaz de ajustarse a *datasets* complejos. Dicho algoritmo tiene una estructura de árbol jerárquica que consta de un nodo raíz, ramas, nodos internos y nodos hoja [18], [1]:

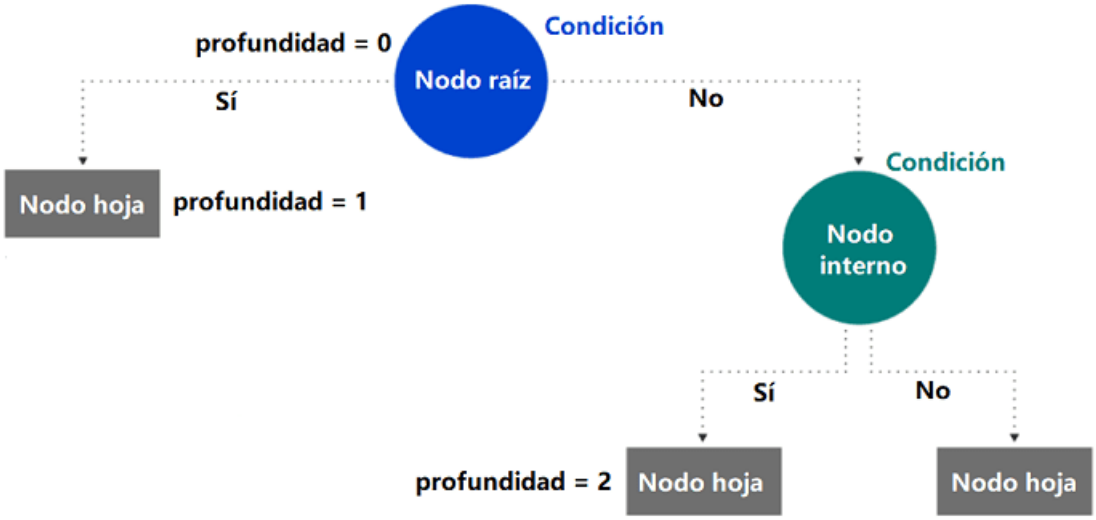


Figura 3.3: Arquitectura árbol de decisión.

Como es posible visualizar en el esquema, **el método de predicción que sigue un árbol de decisión se basa en reglas binarias** (sí o no). Hacer una predicción requiere atravesar el árbol desde el nodo raíz hasta un nodo hoja, por lo que **la clase o el valor final estimado coincidirá con el del nodo hoja alcanzado** tras el recorrido.

[1] https://www.ibm.com/topics/decision-trees

Sin embargo, aunque se trate de un modelo que ofrece un gran número de ventajas, cuenta con dos grandes inconvenientes, los cuales han sido solventados en los bosques aleatorios gracias a su arquitectura:

1. **Alta varianza:** Los árboles de decisión son estimadores de alta varianza. Esto implica que pequeñas variaciones dentro de los datos pueden producir un árbol de decisión muy diferente, alterando de esta forma el resultado final.

   Por otro lado, un ***random forest* introduce aleatoriedad al cultivar un gran número de árboles de decisión**. Este hecho tiene por consecuencia una mayor diversidad que intercambia un mayor sesgo por una menor varianza, lo que generalmente produce un mejor modelo. También, un bosque aleatorio normalmente es entrenado mediante la técnica de *bagging*, cuyo objetivo consiste en disminuir la varianza.

2. **Sobreajuste** (*overfitting*): Una de las grandes limitaciones de los árboles de decisión es el sobreajuste. El sobreajuste ocurre cuando un modelo estadístico se ajusta exactamente a sus datos de entrenamiento y, cuando esto sucede, el algoritmo no puede actuar con precisión frente a datos no vistos anteriormente, lo que anula su propósito.

   Los bosques aleatorios logran superar dicha limitación al tomar la predicción de cada árbol y, en función de la media (regresión) o de la clase más votada (clasificación) de todos ellos , se predice el resultado. Cuanto mayor sea el número de árboles que componen el bosque aleatorio, mejor será la precisión del modelo y se reducirá el problema del sobreajuste [3].

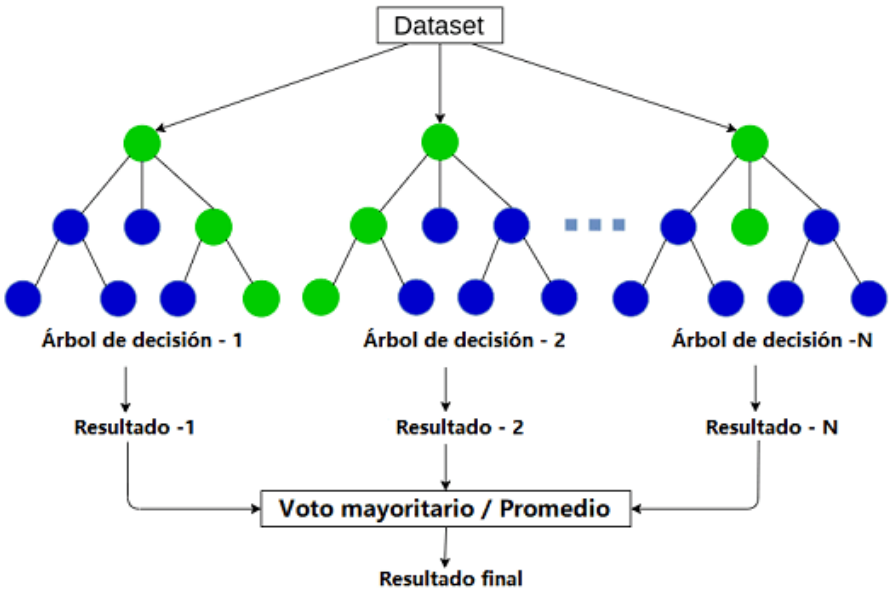


Figura 3.4: Arquitectura bosque aleatorio [33].

Support Vector Machines

SVM se define como un algoritmo de aprendizaje supervisado muy eficaz y versátil capaz de realizar actividades de clasificación, regresión y detección de *outliers*, siendo, al

mismo tiempo, adecuado para la clasificación de *datasets* complejos de tamaño medio a pequeño. Esto lo convierte en uno de los modelos más empleados en *machine learning*, destacando su uso en aplicaciones médicas de procesamiento de señales, procesamiento del lenguaje natural y reconocimiento de imágenes y voz [18].

El principal objetivo de este algoritmo es encontrar un **hiperplano** dentro de un espacio N-dimensional (N: número de características) que clasifique los puntos de datos [16].

Los **hiperplanos** son límites de decisión que ayudan a separar los datos en sus respectivas clases. Para ejecutar la clasificación existen múltiples hiperplanos posibles, debiéndose escoger siempre como hiperplano aquel que presente el **margen máximo**, es decir, la distancia máxima entre datos de ambas clases. Por otra parte, los **vectores de soporte** son los puntos más cercanos al hiperplano y son los que determinan e influyen su posición y orientación. Mediante estos vectores es como se consigue maximizar el margen del clasificador.

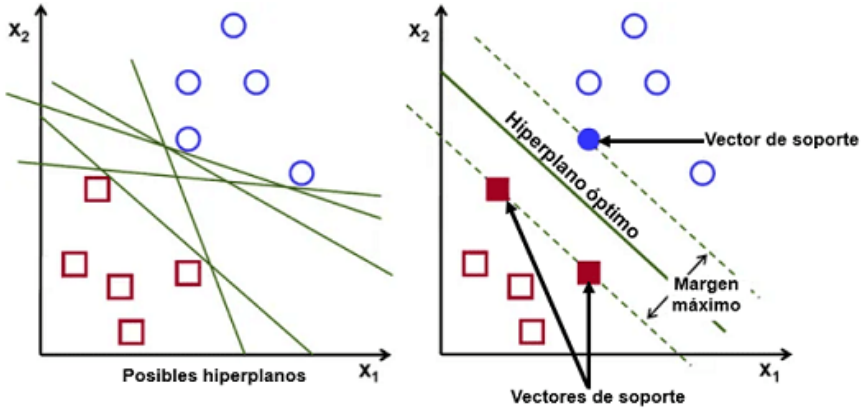


Figura 3.5: Búsqueda de hiperplano en SVM [16].

Para realizar con éxito la tarea de clasificación existen dos tipos de *support vector classifiers* (SVC) [39]:

1. **SVC Lineal:** Este tipo hace referencia al modelo SVM utilizado para clasificar datos que pueden ser separados linealmente. Esto implica que un conjunto de datos puede ser dividido en clases o categorías con la ayuda de una sola línea recta.

2. **SVC No lineal:** A diferencia del algoritmo anterior, este se aplica a *datasets* cuyos datos no pueden ser separados linealmente y, por lo tanto, no es posible usar una línea recta para llevar a cabo la distinción entre clases.

   En esta situación, se emplea la técnica conocida como ***kernel trick***. Este truco se basa en el uso de una **función *kernel*** para transformar las muestras de un conjunto de datos. Las funciones *kernel* asignan los datos a una dimensión diferente, normalmente superior, con el objetivo de facilitar la clasificación al ser capaces, de estar forma, de dividir los datos linealmente a través de un plano [2].

[2] https://es.mathworks.com/discovery/support-vector-machine.html

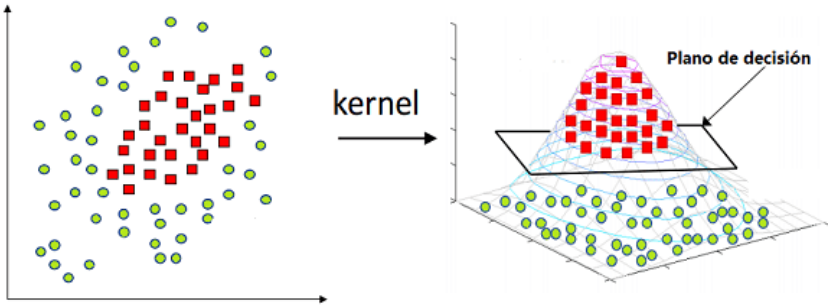


Figura 3.6: *Kernel trick* en SVC no lineal [41].

De igual manera, tenemos disponibles diferentes tipos de SVC no lineal dependiendo de la función *kernel* adoptada:

Tabla 3.1: Tipos de SVC en función del *kernel*.

Tipo de SVC	**Función *Kernel***
Gaussiana (RBF)	$K(x_1, x_2) = exp(-\frac{\|x_1 - x_2\|^2}{2\sigma^2})$
Lineal	$K(x_1, x_2) = x_1^T x_2$
Polinómica	$K(x_1, x_2) = (x_1^T x_2 + 1)^\rho$
Sigmoide	$K(x_1, x_2) = tanh(\beta_0 x_1^T x_2 + \beta_1)$

Por regla general, para decidir el *kernel* que mejor se adapte a nuestro caso de uso, siempre debemos probar, como primera opción, el *kernel* lineal (SVC lineal), especialmente si se dispone de un conjunto de datos muy grande o con muchas características. No obstante, como el *dataset* que será utilizado en nuestro *benchmarking* es pequeño, **optaremos por aplicar el *kernel* gaussiano RBF** (**SVC gaussiano RBF**), debido a que esta versión del algoritmo funciona bien en la mayoría de situaciones [18].

Multilayer Perceptron

El perceptrón multicapa es un algoritmo supervisado que se establece dentro de la rama del *deep learning* como una **red neuronal artificial (ANN)** que es adecuada para resolver situaciones en las que se dan esta serie de condiciones [30]:

- Los datos son proporcionados en formato **tabular** (archivo CSV, hoja de cálculo).
- Problemas de **clasificación** donde a los datos de entrada se les asigna una clase.
- Problemas de **regresión** donde se predice un valor real dado un conjunto de entrada.

Las **ANN** se encuentran en el mismo núcleo del aprendizaje profundo, tratando de replicar, a partir de su configuración, el funcionamiento del cerebro humano. Son redes

versátiles, potentes y escalables, ideales para abordar grandes problemas de aprendizaje automático que implican una alta complejidad como la clasificación de miles de imágenes (Google), reconocimiento de voz (Siri de Apple), recomendación de vídeos (Youtube) y detección de enfermedades (COVID-19, cáncer) [18].

El **perceptrón** representa una de las arquitecturas ANN más simples, cuya lógica se encuentra inspirada en una neurona artificial ligeramente diferente denominada ***threshold logic unit*** (**TLU**). Ambas neuronas artificiales son similares, pero la diferencia radica en que en el perceptrón tanto la entrada como la salida son números en lugar de valores binarios, como ocurre en TLU, donde cada conexión de entrada tendrá asociado un peso específico.

Igualmente, el perceptrón (neurona en una red neuronal) posee una estructura simple pero ingeniosa que consta de las cuatro partes que se ilustran en esta imagen:

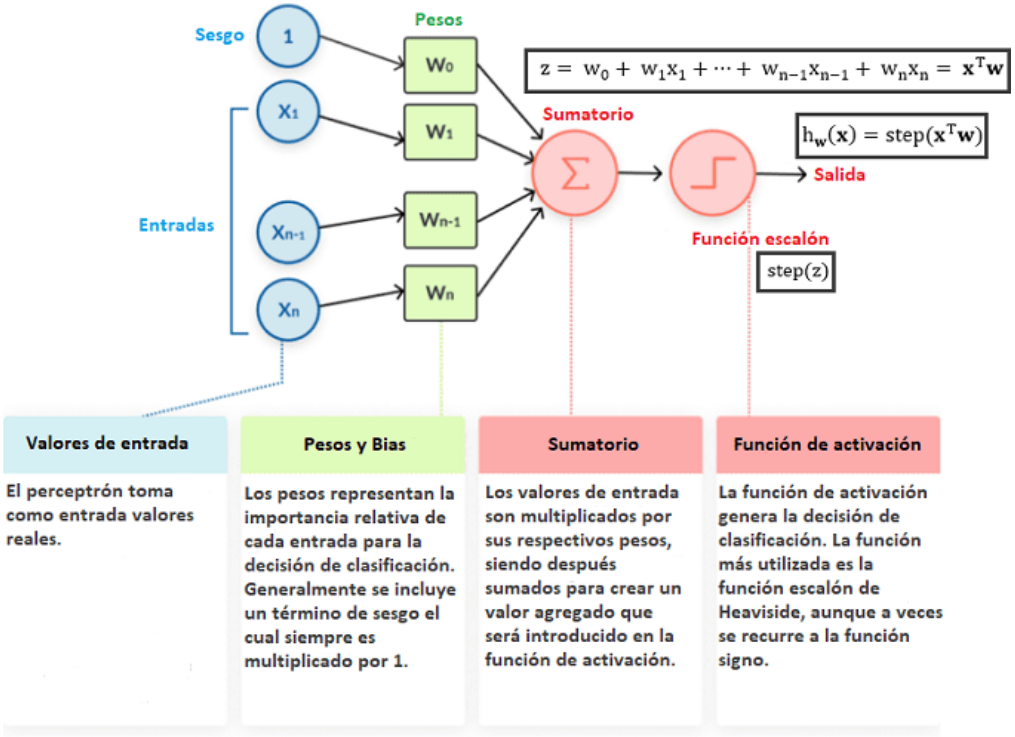


Figura 3.7: Estructura del perceptrón [3].

En 1969, la monografía "*Perceptrons*", redactada por los científicos Marvin Minsky y Seymour Papert [24], dejó en evidencia las debilidades de este algoritmo. En particular, se destaca el hecho de ser **incapaz de resolver problemas triviales como la clasificación XOR y la clasificación de datos no lineales**.

Las limitaciones expuestas fueron la motivación que dio lugar al desarrollo de la ANN conocida como ***multilayer perceptron*** (**MLP**). Este nuevo algoritmo goza de las características que se enumeran a continuación [8, 4]:

[3] https://indiantechwarrior.com/perceptrons-and-multi-layer-perceptrons/

1. Como es posible visualizar en 3.8, la arquitectura de MLP está configurada por **tres tipos de capa**: una capa de entrada (*input layer*), una o más capas ocultas (*hidden layers*) y una capa final de salida (*output layer*).
2. Cada capa, exceptuando la de salida, incluye una **neurona de sesgo** (*bias neuron*) y está **totalmente conectada** con las neuronas de la siguiente capa (*fully connected layer*).
3. La **capa oculta** es la innovación que fue agregada al perceptrón para aumentar su rendimiento. De esta forma, **el perceptrón múltiple está cualificado para solucionar casos de alta complejidad** (XOR, no linealidad).
4. MLP entra dentro de la categoría de **algoritmos *feedforward***, los cuales siguen el funcionamiento expuesto en el gráfico 3.7. No obstante, en este caso, a diferencia del perceptrón, cada combinación lineal se propaga a la siguiente capa.

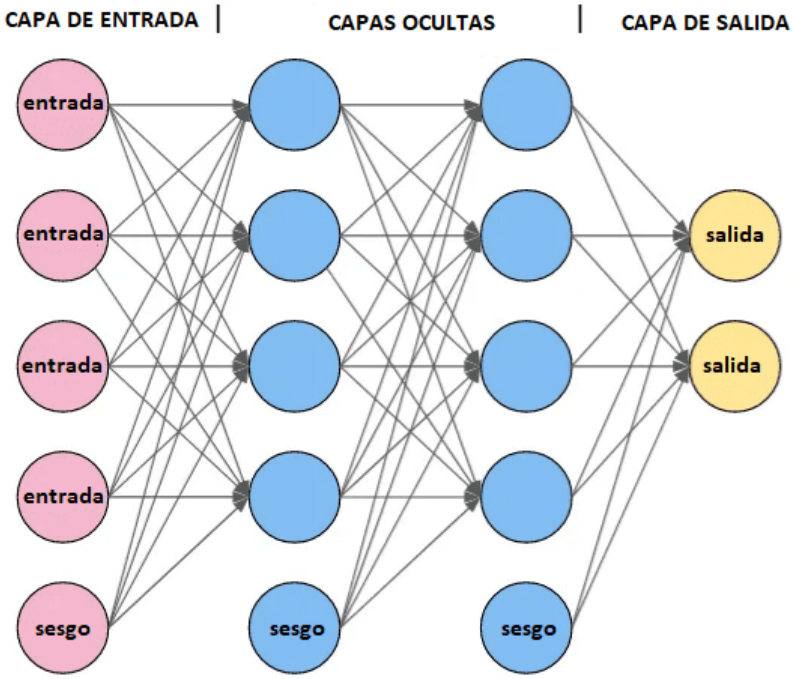


Figura 3.8: Estructura básica de MLP [4].

Metodología de ajuste y evaluación de modelos

Cuando creamos un estimador, se nos presentan multitud de opciones de diseño sobre cómo definir la arquitectura del mismo. Debido a ello, es vital poner en práctica un procedimiento que nos permita determinar el mejor ajuste para nuestros modelos antes de iniciar con el entrenamiento. Así, el **ajuste de un algoritmo** se define como el proceso experimental, fundamentado en la búsqueda de los hiperparámetros más óptimos, llevado a cabo con el propósito de maximizar el rendimiento de nuestro modelo.

Se entiende por **hiperparámetros** a las variables que controlan el proceso de aprendizaje durante el entrenamiento y que precisan cómo se estructura realmente nuestro modelo.

Este conjunto de variables, al ser externas al estimador, **no pueden ser calculadas mediante los datos de entrenamiento, sino que deben ser establecidas por el usuario**.

De esta forma, se evidencia la importancia de aplicar una metodología que nos garantice la correcta selección de este tipo de parámetros, con el fin de alcanzar la mayor tasa de rendimiento posible. Para ello, existen dos estrategias para desempeñar adecuadamente esta tarea [4]:

- **Ajuste manual:** En este planteamiento, los valores de los hiperparámetros se fijan en función de la intuición o de la experiencia pasada, repitiendo el ajuste hasta que se encuentra el valor óptimo para cada uno de ellos.
- **Ajuste automático:** Aquí, en cambio, se utiliza un algoritmo con la intención de encontrar rápidamente su mejor grupo de valores, donde destacan los métodos *grid search*, *randomized search* y *bayesian optimization*.

Teniendo en cuenta las características del presente proyecto, **se ejecutará un ajuste automático**, dado que el ajuste manual conforma una perspectiva lenta y tediosa que no resulta ser práctica cuando hay muchos hiperparámetros con un amplio rango, como es nuestro caso. En consecuencia, se emplearán las siguientes clases de la librería Scikit-learn para completar dicho proceso:

1. `GridSearchCV()`: *Grid search* es una de las técnicas más básicas utilizadas para emprender este cometido, a causa de su enfoque basado en la **fuerza bruta**:
   - Se construyen todas las combinaciones de modelos posibles en base a una cuadrícula de parámetros que contiene todos los valores que el usuario desea probar.
   - Se entrena y se evalúa el rendimiento de cada algoritmo mediante **validación cruzada** (*cross validation*) para, finalmente, extraer el de mayor puntuación.

   Al considerar todas y cada una de las opciones durante la realización del ajuste, este método puede resultar muy costoso desde el punto de vista computacional, implicando un tiempo significativo. Por este motivo, su uso se recomienda y es apropiado cuando se están explorando relativamente pocas alternativas.
2. `RandomizedSearchCV()`: Sin embargo, cuando el espacio de búsqueda de hiperparámetros es grande, a menudo es preferible recurrir a **`RandomizedSearchCV`** en su lugar. Esta clase, en contraste con la anterior, evalúa a través de validación cruzada únicamente un número determinado de combinaciones, seleccionando un candidato aleatorio para cada hiperparámetro en cada una de las iteraciones del entrenamiento.

Valorando las ventajas y desventajas que supone el uso de los recursos `GridSearchCV` y `RandomizedSearchCV`, se determina que el mejor procedimiento para culminar el ajuste de modelos, según el perfil del banco de pruebas que tenemos por objetivo desarrollar, es el recogido en esta tabla:

[4] https://www.dominodatalab.com/data-science-dictionary/model-tuning

Tabla 3.2: Metodología de ajuste y evaluación para *benchmarking*.

Algoritmo	Ajuste con Scikit Learn
Logistic Regression	`GridSearchCV`
Random Forest	`RandomizedSearchCV`
Support Vector Machine	`GridSearchCV`
Multilayer Perceptron	`RandomizedSearchCV`

Dado que en los algoritmos MLP y *random forest* intervienen en su configuración un gran número de hiperparámetros y candidatos dentro de los mismos, se considera que el uso de `RadomizedSearchCV` es más eficiente para dichos casos. De este modo, logramos reducir notablemente el tiempo de ejecución, siendo mínima la diferencia de puntuación obtenida con respecto a `GridSearchCV`.

La **validación cruzada** es un método estadístico cuya labor consiste en evaluar el rendimiento general de un estimador dado. Esta metodología es conocida por ser más estable y exhaustiva que el uso de un conjunto de validación (método *holdout*), solucionando los problemas que dicha práctica implica. La versión más utilizada de validación cruzada, siendo también la integrada en las clases descritas con anterioridad, es la ***k-fold cross validation***:

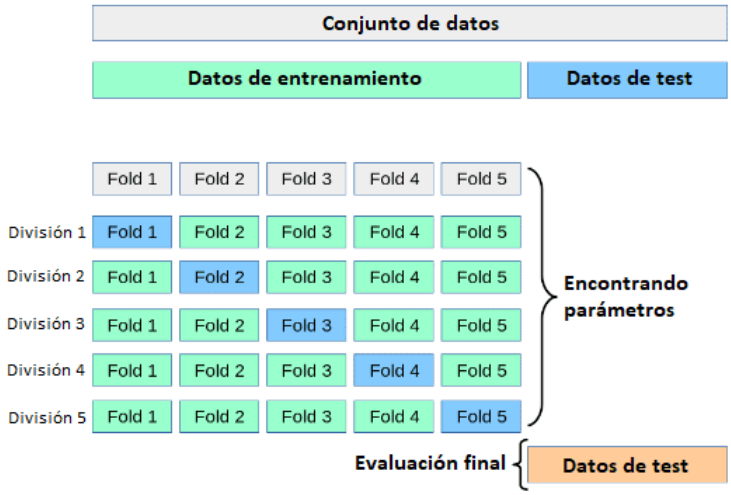


Figura 3.9: *K-fold Cross validation* [5].

Los datos de entrenamiento son divididos en ***k*** conjuntos más pequeños llamados ***folds***, donde la variable *k* es un número especificado por el usuario (normalmente 5 o 10). Para cada uno de los *folds*, se sigue este procedimiento:

[5] https://scikit-learn.org/stable/modules/cross_validation.html

- Un modelo se entrena usando ***k - 1*** de los *folds* como datos de entrenamiento.
- El modelo resultante se valida con la parte restante de los datos.
- La medida de rendimiento calculada por *k-fold cross validation* es el **promedio de los valores calculados en cada iteración del bucle**.

En última instancia, `GridSearchCV` y `RadomizedSearchCV` disponen del parámetro de entrada **`cv`**, cuyo cometido es fijar la estrategia de división que empleará el método de validación cruzada. En nuestro caso, dejaremos el valor establecido por defecto, **`cv=None`**, el cuál aplica ***5-fold cross validation*** (cada combinación será entrenada 5 veces).

3.2.2 Dataset

El conjunto de datos escogido con objeto de entrenar los distintos modelos implicados en el banco de pruebas se denomina ***Room Occupancy Estimation Dataset***, el cual se encuentra accesible en formato csv (**`Occupancy_Estimation.csv`**), a través de la comunidad de Kaggle. El *dataset* mencionado fue diseñado con el propósito de enseñar a los algoritmos la tarea de **estimación del número exacto de ocupantes en una habitación a partir de múltiples sensores ambientales no intrusivos**.

La recolección de datos fue llevada a cabo durante cuatro días en una sala con dimensiones 6m x 4,6m, donde se establecieron 7 nodos sensores y un nodo de borde (*edge node*) en configuración estrella [6]:

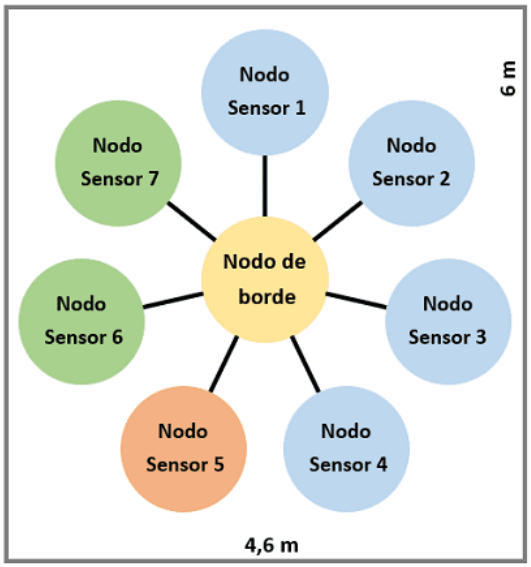


Figura 3.10: Recolección de datos.

- Los nodos sensores **transmiten datos cada 30 segundos al nodo de borde** usando transceptores inalámbricos.
- En este experimento, se utilizaron **5 tipos diferentes de sensores no intrusivos** con la siguiente distribución:
  - **Nodos sensores 1, 2, 3 y 4:** Temperatura, luz y sonido.
  - **Nodo sensor 5:** CO_2.
  - **Nodos sensores 6 y 7:** PIR (*Passive Infrared*, detección de movimiento).

Seguidamente, si se procede al análisis de la distribución de los datos en el archivo `Occupancy_Estimation.csv`, comprobamos que existen **10.129 muestras** con un total de **19 columnas (10.129 x 19)**, cuyo significado se redacta brevemente en la tabla inferior:

[6] https://archive.ics.uci.edu/ml/datasets/Room+Occupancy+Estimation

Tabla 3.3: Columnas del `Occupancy_Estimation.csv`.[7]

Columna	Tipo de dato	Descripción
`Date`	`object`	Fecha del experimento (YYYY/MM/DD).
`Time`	`object`	Tiempo del experimento (HH:MM:SS).
`Si_Temp`	`float64`	Temperatura medida por el sensor i en grados Celsius (i = 1, 2, 3 ,4).
`Si_Light`	`int64`	Luz medida por el sensor i en luxes (i = 1, 2, 3, 4).
`Si_Sound`	`float64`	Sonido medido por el sensor i en voltios (i = 1, 2, 3, 4).
`S5_CO2`	`int64`	CO_2 medido por el sensor cinco en PPM.
`S5_CO2_Slope`	`float64`	Pendiente de los valores de CO_2 calibrados en una ventana deslizante.
`Si_PIR`	`int64`	Detección de movimiento por el sensor i en valor binario (i = 6, 7).
`Room_Occupancy_Count`	`int64`	Número de personas que ocupan la habitación.

Transformación de características

A pesar de que el *dataset* seleccionado no implica una gran dificultad, se ha tomado la decisión de efectuar los siguientes cambios en sus columnas con la finalidad de simplificar el desafío que este trata de abordar:

1. **Recodificación de la columna de salida `Room_Occupancy_Count`**:

   La columna `Room_Occupancy_Count`, la cual contiene la variable de salida, indica el número preciso de personas en la sala donde han sido instalados los sensores no intrusivos, como se muestra en 3.10. Este hecho evidencia que el conjunto de datos fue concebido con la intención de afrontar un problema de **regresión**. Sin embargo, para llevar a cabo la resolución de una cuestión más sencilla, se transformará el problema original en uno de **clasificación binaria**.

   Con esto en mente, **se modificará la columna de salida, de modo que cualquier dato superior a 0 será recodificado como 1**. De esta forma, `Room_Occupancy_Count` quedará traducida en formato binario, donde el valor 0 representará la ausencia de personas en la sala y el 1, la presencia de estas.

2. **Eliminación de las columnas temporales `Date` y `Time`**:

   Si se analizan los tipos de dato de cada una de las características en la tabla 3.3, advertimos que todas integran muestras numéricas, exceptuando las designadas como

[7] https://www.kaggle.com/code/chaunguynnghunh/eda-room-occupancy-estimation

`Date` y `Time`. Las columnas mencionadas, a diferencia del resto, son objetos que incluyen caracteres especiales, los cuales son **fuente de errores a la hora de poner en práctica el entrenamiento de los modelos**. Debido a ello, se optará por su **filtración y por dejarlas fuera del proceso**.

Aunque se incluyeran estas dos columnas en el procedimiento, por ejemplo, uniendo ambas partes en una única columna y transformando sus unidades en formato `epoch` como `longint`, **los valores de muestras sucesivas de dicha columna estarían totalmente correladas entre sí**. En consecuencia, se echarían a perder la mayoría de algoritmos seleccionados, siendo necesarios modelos de alta complejidad para tratar con la relación temporal existente entre las distintas filas.

Aviso

Es fundamental recalcar que este acto no significa que nos estemos olvidando de la información temporal para asumir la independencia de cada muestra con el resto. En realidad, sabemos que esto no es así. En cambio, **se plantea que dicho conocimiento no nos aporta un valor añadido a la hora de predecir si la habitación está ocupada o vacía**.

Nota

Una vez hayan sido aplicadas las transformaciones precisadas en este apartado, las dimensiones del *dataset* se verán alteradas de modo que, **10.129 x 19** → **10.129 x 17**

División del conjunto de datos

En ciencia de datos, la división del *dataset* en dos o más subconjuntos para su uso en las distintas fases del desarrollo de un algoritmo es una actividad común. En la práctica, supone un aspecto muy importante e imprescindible cuando se procede a la creación de un modelo basado en datos. Generalmente, suelen llevarse a cabo dos tipos de divisiones con las siguientes particiones [40]:

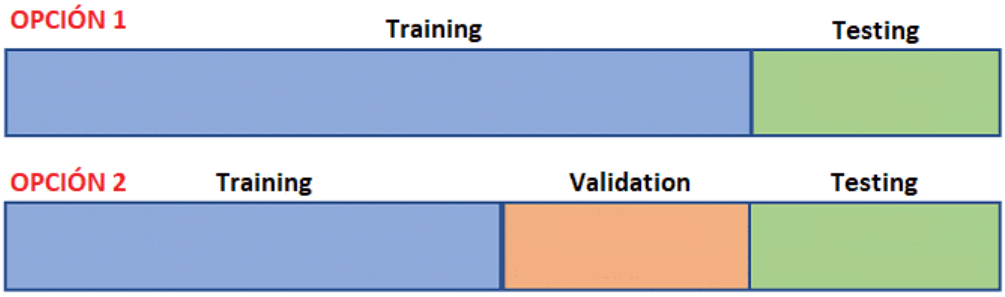


Figura 3.11: Opciones de división del *dataset*.

- ***Training set***: Es el subconjunto responsable de entrenar el modelo. A partir de esta división, el algoritmo aprenderá a predecir a partir de los patrones y relaciones que se dan dentro de los datos.

- ***Validation set***: Es el grupo que será aplicado para entender y validar el rendimiento del modelo en comparación con diferentes alternativas y opciones de hipeparámetros. Estos datos nos ayudan a seleccionar el mejor algoritmo.
- ***Testing set***: Por último, el conjunto de *testing* contiene los datos con los que probaremos el modelo entrenado y validado. Por lo tanto, este nos indicará qué tan eficiente es nuestro estimador, verificando su precisión final.

Con motivo de la metodología que será empleada posteriormente para efectuar el ajuste y evaluación de los modelos del *benchmarking* (3.2.1), **no precisaremos del subconjunto de validación**. En consecuencia, en esta propuesta, **el conjunto de datos será dividido en dos porciones, *training* y *testing***, como es posible visualizar en 3.11.

Nota

Con objeto de realizar la separación de los datos como se ha explicado en el párrafo anterior, se recurrirá a la función `train_test_split` de Scikit-learn, cuya tarea consiste en repartir aleatoriamente las muestras en los dos grupos deseados.

No obstante, uno de los principales problemas a los que se enfrentan los profesionales en esta materia es el **cómo se deben dividir los datos para los procesos de entrenamiento y prueba**. Aunque al principio pueda parecer un problema simple, si se ejecuta la división de manera deficiente, esto puede provocar efectos impredecibles en los resultados del modelo como los fenómenos [32]:

- ***Overfitting*:** El modelo no puede generalizarse y se ajusta demasiado a los datos de entrenamiento (↑ *training set*).
- ***Underfitting*:** El modelo no puede capturar la relación entre las variables de entrada y salida con precisión (↓ *training set*).

Como todos los desafíos que se plantean en *machine learning*, no se dispone de una respuesta única y sencilla, pero, a pesar de ello, sabemos que **el tamaño del *dataset* supone un factor que influye directamente en la decisión de en qué ratio deberíamos llevar a cabo dicha división**:

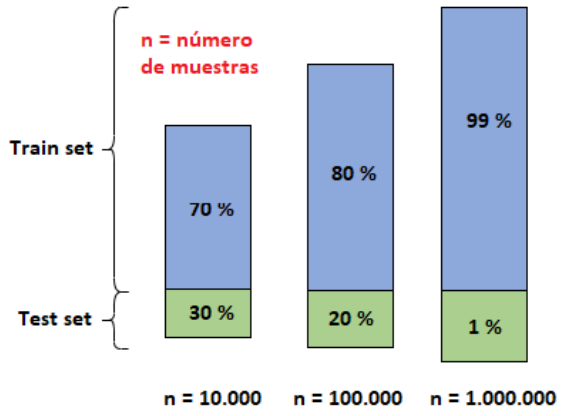


Figura 3.12: Ratios para el *dataset* [36].

Teniendo en cuenta las recomendaciones expuestas en el gráfico y el volumen del conjunto de datos escogido (**10.129 muestras**), concluimos que las proporciones que mejor se adaptan a nuestro caso de uso son:

- **Conjunto de entrenamiento**: 70 % de las muestras del *dataset*.
- **Conjunto de prueba**: 30 % de las muestras del *dataset*.

Escalado de características

El **escalado de características** se precisa como una técnica de preprocesamiento que involucra la transformación de los valores de cada una de las características de un conjunto de datos a una escala similar. Su utilización es **esencial** cuando se trabaja con *datasets* donde las muestras tienen **diferentes rangos, unidades de medida u órdenes de magnitud**, debido a que **existen modelos que son sensibles ante esta situación** [9].

Los **árboles de decisión** y los **bosques aleatorios** son dos de los pocos algoritmos de los cuales no debemos preocuparnos por la escala de las características, dado que ambos son **invariantes** en este aspecto y ello no influye en su capacidad de predicción [26]. Sin embargo, la mayoría de modelos en *machine learning* no funcionan correctamente cuando los atributos numéricos de entrada tienen magnitudes muy diferentes [18], por ejemplo:

- Algoritmos que utilizan el **descenso de gradiente** (en inglés, *descent gradient*) como técnica de optimización: Regresión lineal, regresión logística, redes neuronales, etc.
- Algoritmos que se basan en la **distancia entre los puntos de datos** para determinar su similitud y agruparlos: *K-Nearest Neighbors*, *K-means clustering*, SVM, etc.

Así pues, si evaluamos el estado actual de nuestro conjunto de datos, se concluye que **el desarrollo de esta práctica será necesaria**, ya que las características recogidas dentro del mismo presentan los factores descritos:

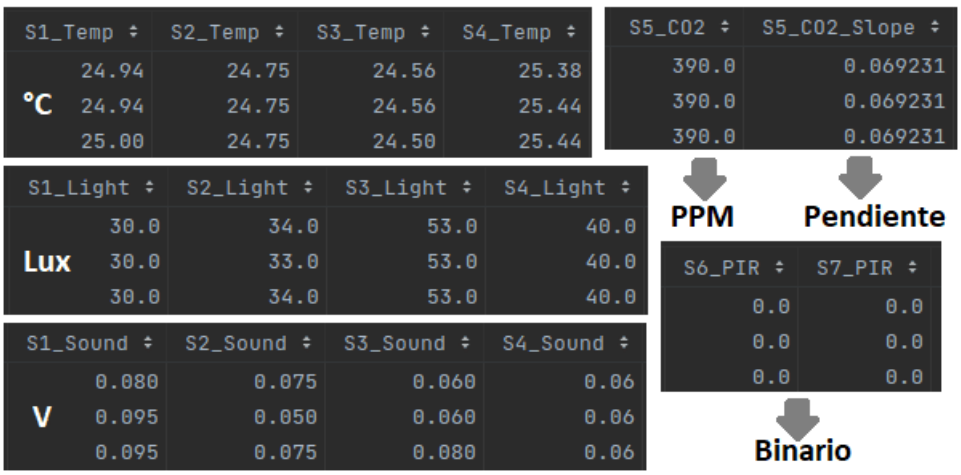


Figura 3.13: Escalas y unidades de `Occupancy_Estimation.csv`.

El módulo **`sklearn.preprocessing`**, propiedad de Scikit-learn, facilita una gran variedad de funciones y clases para desempeñar eficazmente esta tarea. Entre las distintas opciones que se ofrecen, estas son las tres clases de escalado más destacadas [34, 26]:

1. `MinMaxScaler()` (*normalización*): Es la forma de proceder más sencilla, donde los datos son cambiados de tal manera que todas las características se encuentren exactamente dentro de un intervalo específico (por defecto, entre 0 y 1).

$$x' = \frac{x - min(x)}{max(x) - min(x)}$$

Escalado de x con MinMaxScaler (3.5)

2. **`StandardScaler()`** (*estandarización*): Asegura que, para cada característica, su media sea 0 ($\mu = 0$) y su desviación estándar 1 ($\sigma = 1$), coincidiendo con la distribución normal estándar. De esta forma, todas las características tienen la misma magnitud.

$$x' = \frac{x - \mu_x}{\sigma_x}$$

Escalado de x con StandardScaler (3.6)

3. **`RobustScaler()`**: Funciona de manera similar a `StandardScaler`. En cambio, en este caso, se elimina el valor de la mediana y se escala la información según el primer y tercer cuartil (Q_1 y Q_3) del conjunto de datos. De este modo, los valores más extremos y atípicos son menos pronunciados.

$$x' = \frac{x - Q_1(x)}{Q_3(x) - Q_1(x)}$$

Escalado de x con RobustScaler (3.7)

Adicionalmente, se pueden visualizar los resultados de aplicar estas tres técnicas de escalado sobre las características de un *dataset* en el gráfico posterior:

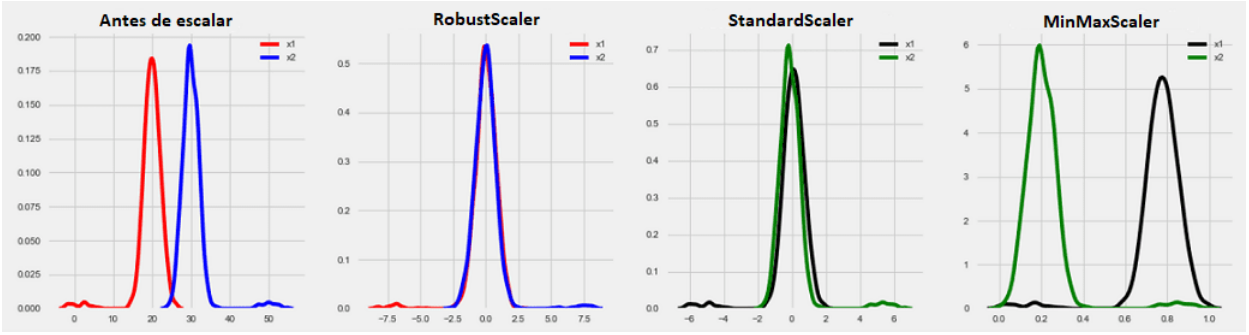


Figura 3.14: `RobustScaler` vs `StandardScaler` vs `MinMaxScaler` [7].

Como se puede apreciar, cada una de ellas tiene sus propios efectos sobre los datos. Si reflexionamos acerca de su impacto y tenemos en mente que nuestro objetivo es abordar un problema de clasificación binaria, se deduce que **el uso de `MinMaxScaler` no es una opción viable**. El hecho de que la normalización obligue a todos los datos a establecerse dentro de un rango determinado puede provocar que **dichos puntos se compacten, dificultando, en consecuencia, la distinción de clases**.

Por ende, solamente se hará uso de las clases **`StandardScaler` y `RobustScaler` para llevar a término el escalado de características** en nuestro *dataset*, dado que son las opciones más versátiles:

Tabla 3.4: Escalado de características para *benchmarking*.

Algoritmo	Escalado con Scikit-learn
Logistic Regression	`RobustScaler()`
Random Forest	✗
Support Vector Machine	✗
Multilayer Perceptron	`StandardScaler()`

Nota

La estrategia seguida para determinar la mejor técnica de escalado en cada modelo es la conocida como **prueba y error**. Es decir, se ha iterado entre distintas soluciones hasta dar con la configuración de mayor rendimiento.

3.2.3 Herramientas (Hi)

En base a lo observado durante el estudio del capítulo 2, relativo a las herramientas existentes para el cálculo del consumo dentro del panorama de la Inteligencia Artificial, se llegó finalmente a la conclusión de que **las librerías CodeCarbon** (2.2.1) **y eco2AI** (2.2.5) **son las candidatas más adecuadas** para ser comparadas.

Metodología de las herramientas

A continuación, se muestra de manera gráfica y resumida la metodología que sigue cada una de las herramientas escogidas para estimar el consumo y la huella de carbono durante la ejecución de un experimento [10], [8]:

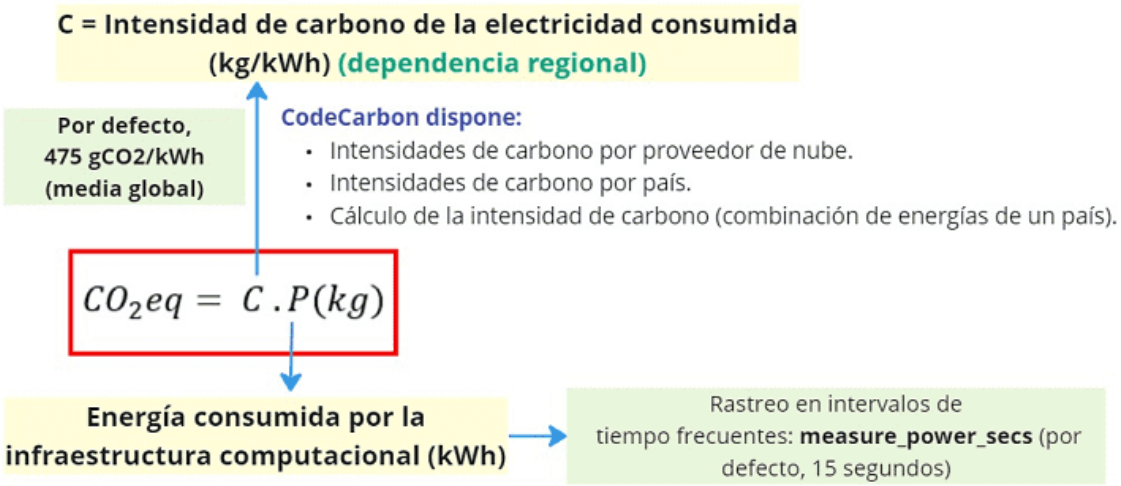


Figura 3.15: Metodología CodeCarbon (Parte 1).

[8] https://mlco2.github.io/codecarbon/methodology.html

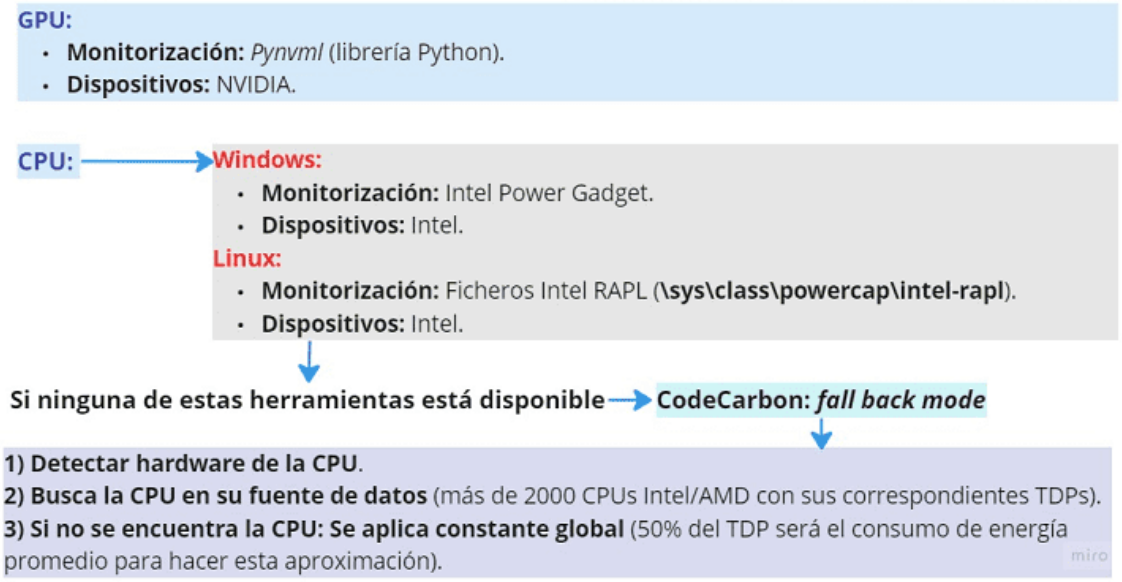


Figura 3.16: Metodología CodeCarbon (Parte 2).

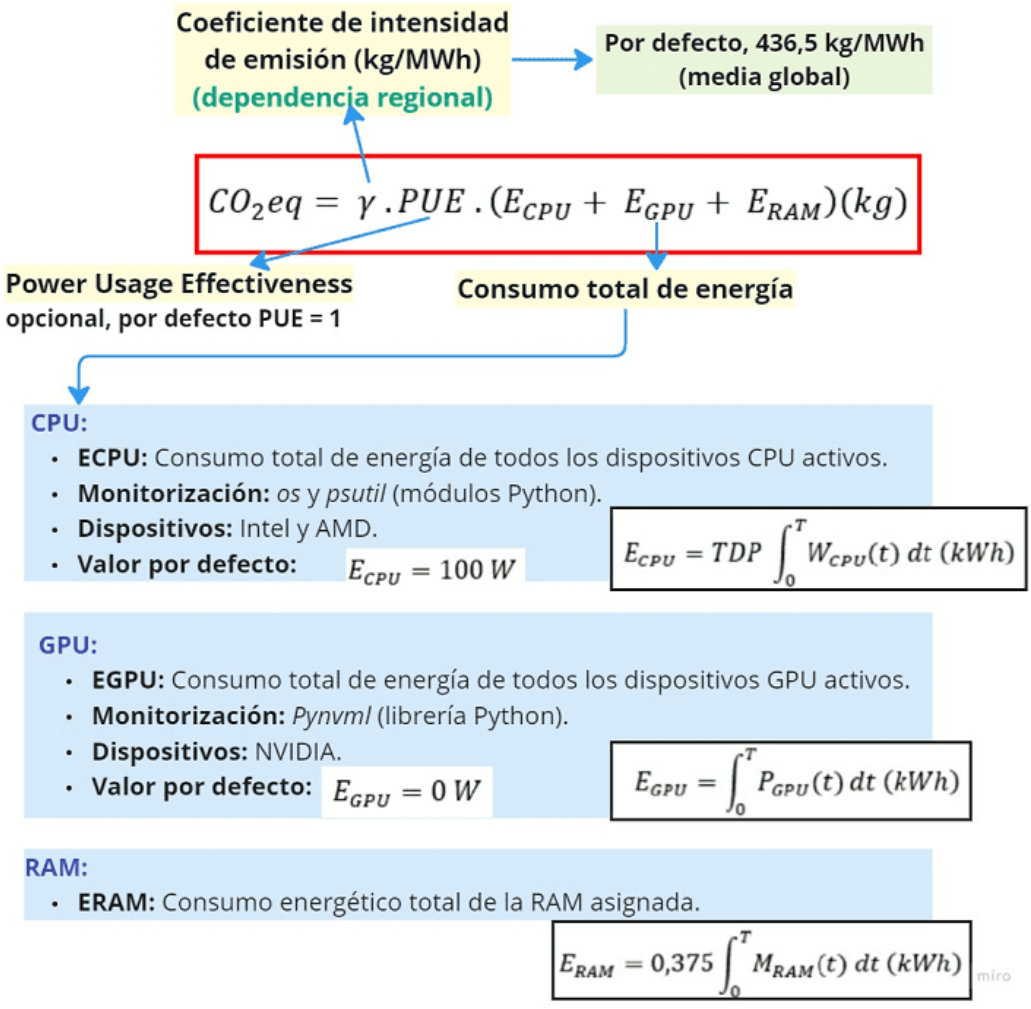


Figura 3.17: Metodología Eco2AI.

Aviso

Se considera de vital importancia conocer estas metodologías ya que, de esta manera, seremos capaces de **entender con mayor facilidad las posibles discrepancias que puedan existir entre los resultados calculados por ambas herramientas**.

Infraestructura de trabajo

En definitiva, el equipo que será utilizado para esta fase del proyecto será idéntico al empleado en el capítulo 2 para la investigación de las herramientas:

Equipo

- **Infraestructura**: Máquina virtual
- **Sistema operativo**: Ubuntu 22.04.1 LTS
- **Versión de Python**: 3.10
- **RAM**: 4,9 GiB
- **Número de CPUs**: 4
- **Modelo de la CPU**: Intel(R) Core(TM) i5-9400F CPU @ 2.90GHz
- **Capacidad de disco**: 107,9 GB

3.2.4 Resultados (rij)

Con vistas a completar con éxito los objetivos específicos propuestos en 1.1.2, el tipo de resultado que será recopilado durante el transcurso del banco de pruebas es la **energía eléctrica que ha sido consumida por la tarea de entrenamiento del algoritmo en *kilowatios* hora (kWh)**. Dicha energía será estimada por las herramientas escogidas con tal función:

- `energy_consumed`: Columna del fichero resultado (`emissions.csv`) generado por **CodeCarbon**, la cual almacena la energía total utilizada en *kilowatios* (**kWh**).
- `power_consumption`: Columna del fichero resultado (`emission.csv`) generado por **eco2AI** que guarda la potencia total consumida en *kilowatios* hora (**kWh**).

Nota

Observamos que el resultado que eco2AI nos ofrece y define como potencia (**`power_consumption`**) en la práctica es **energía**, debido a que sus unidades pertenecen a esta magnitud (**kWh**). Por ello, se ha escogido el valor de esta columna como resultado.

3.2.5 Métricas de evaluación (ei)

La **evaluación de un modelo** representa una **etapa crucial** en todo flujo de trabajo que implique el desarrollo de sistemas de aprendizaje automático. Esta etapa conlleva **el uso de**

distintas métricas de evaluación para entender el comportamiento de un modelo frente a datos nunca vistos y conocer así sus puntos fuertes y débiles.

Las **métricas de evaluación** son utilizadas para medir la calidad de un algoritmo de aprendizaje automático [9]. Asimismo, existen diferentes tipos de métricas en función de si el modelo que deseamos evaluar fue entrenado para realizar tareas de clasificación o regresión [1]. No obstante, teniendo en consideración el contexto de **este proyecto**, se estudiarán y se escogerán únicamente las **métricas orientadas a la evaluación de clasificadores**.

Uno de los mejores métodos para desempeñar este último cometido es a través de la **matriz de confusión**:

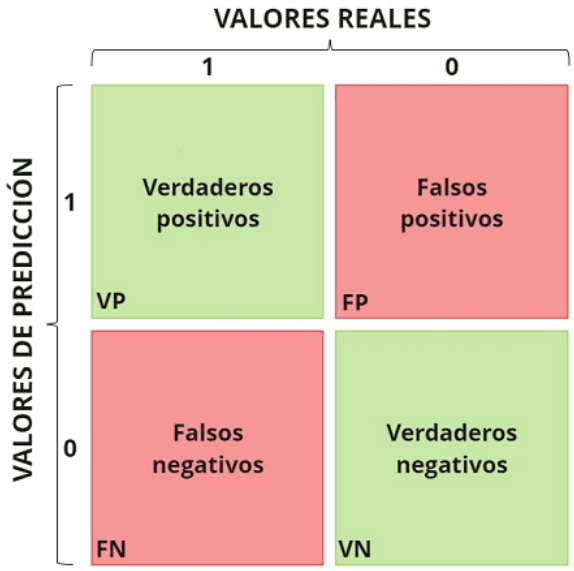


Figura 3.18: Matriz de confusión.

Es una matriz NxN empleada para analizar el rendimiento de un modelo de clasificación, donde N es el número de clases objetivo. Por otro lado, aporta un resumen del número de predicciones correctas e incorrectas realizadas por un clasificador en cuatro categorías [35]:

- **VP**: Predice positivo y es cierto.
- **FP**: Predice positivo y no es cierto (**Error tipo 1**).
- **VN**: Predice negativo y es cierto.
- **FN**: Predice negativo y no es cierto (**Error tipo 2**).

De acuerdo con lo constatado hasta ahora, la matriz de confusión supone una herramienta muy útil a la hora de ofrecer una mejor idea sobre qué tan bueno es nuestro modelo clasificando. Aun así, existen ocasiones, como es este caso, en las que se prefieren métricas más concisas. Por ello, estas serán la métricas que calcularemos, partiendo de la matriz, con el fin de evaluar los clasificadores de nuestro *benchmarking* [18, 34]:

Nota

La métrica de evaluación ***accuracy*** **no será recogida durante el desarrollo del banco de pruebas**. La razón detrás de esta decisión se debe a que, generalmente, esta **no suele ser la medida de rendimiento más óptima para los clasificadores**, especialmente cuando se trata con *datasets* sesgados [18].

- ***Precision*** (*positive predictive value, PPV*): Mide cuántas de las muestras que han sido asignadas por el modelo como positivas (1), son realmente positivas. Se suele recurrir a esta métrica cuando el objetivo es limitar el número de falsos positivos (FP).

[9] https://deepai.org/machine-learning-glossary-and-terms/evaluation-metrics

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

Precisión (3.8)

- ***Recall*** (*true positive rate, TPR*): Esta métrica, en cambio, define el ratio de casos positivos que han sido detectados correctamente por el clasificador. Acostumbra a ser utilizado cuando es importante evitar los falsos negativos (FN).

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

Recall (3.9)

- ***F-score***: Si se requiere una visión completa del funcionamiento de un clasificador, es posible resumir las métricas descritas con anterioridad a través de la media armónica de ambas, conocida como *f-score*:

$$F = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

F-score (3.10)

3.3 Arquitectura final

En último lugar, se pone de manifiesto **la arquitectura final del *benchmarking*** que tenemos por objetivo crear, donde se han implementado, sobre la arquitectura general (3.1) expuesta al inicio, todas las decisiones de diseño descritas durante la redacción de este capítulo:

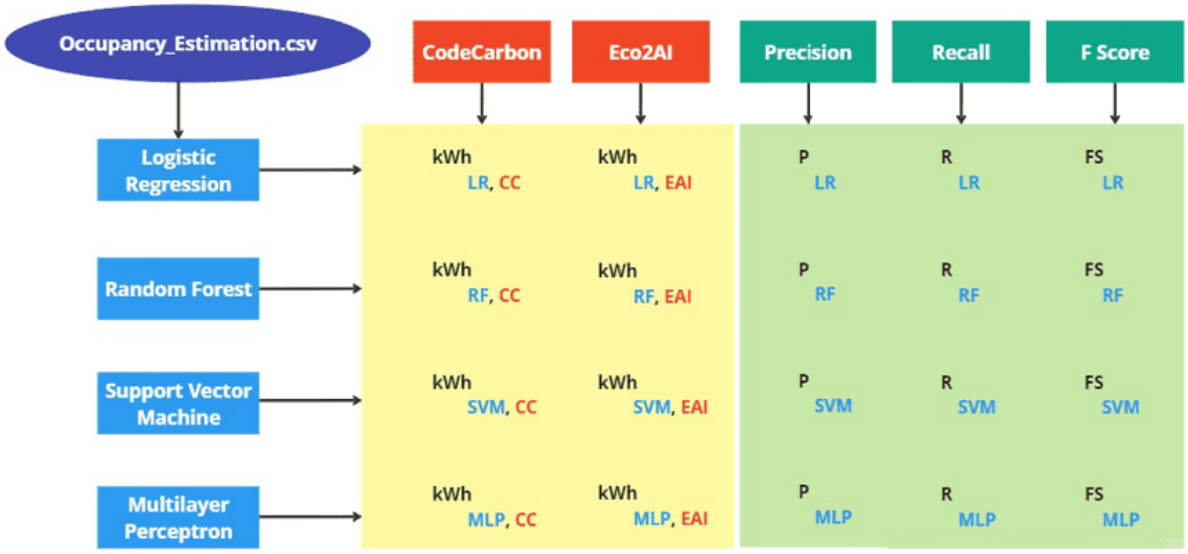


Figura 3.19: Arquitectura final del *benchmarking*.

Capítulo 4

Experimentos y validación

En el capítulo 4 se explicará con detalle cuáles son los pasos a seguir para ejecutar el *benchmarking*, construido con la meta de culminar los objetivos propuestos en este proyecto. Al mismo tiempo, se ilustrará su funcionamiento interno y, finalmente, se presentarán los resultados obtenidos.

4.1 Guía de configuración y ejecución

En primer lugar, se aportan instrucciones detalladas y precisas sobre cómo debe ser integrado el programa para su puesta en producción en la máquina local del usuario. Por ello, en esta guía, se dividirán las indicaciones citadas en **3 etapas** consecutivas:

Paso 1: Clonar repositorio

Inicialmente, el principal requisito es disponer del repositorio GitHub de este TFG en el equipo donde se aspira a desempeñar su aplicación. Con ello en mente, es posible clonar dicho repositorio a través del siguiente comando:

```
(base) root@ubuntu:~$
   git clone https://github.com/misanchz98/TFG-Project.git
```

Paso 2: Creación del entorno de trabajo

Posteriormente, se recomienda llevar a cabo la creación de un entorno virtual para su configuración. Por ejemplo, si se efectúa a través del gestor de paquetes **conda**:

```
(base) root@ubuntu:~$ conda create --name TFG
```

Paso 3: Instalación de requisitos

Dentro del sistema de directorios que estructura el desarrollo *software* realizado, se cuenta con el archivo **`requirements.txt`**, cuyo contenido acoge todas las dependencias Python precisadas por el proyecto. Su instalación requiere de la ejecución de un único comando en la ruta donde se sitúa el fichero y en el entorno generado previamente:

```
(TFG) root@ubuntu:~$ pip install -r requirements.txt
```

Finalizado este paso, ya estamos preparados para la puesta en práctica del material proporcionado y para obtener los resultados de los distintos experimentos abordados.

Nota

Si durante la ejecución de uno de los programas surge algún problema relacionado con las librerías de estimación de consumo, se aconseja revisar su sección en 2.2.

4.2 Procedimiento del banco de pruebas

Tras la información comunicada en la sección anterior, se expone, de manera concisa y simplificada, cuál ha sido el procedimiento diseñado y establecido para efectuar la construcción del *benchmarking*:

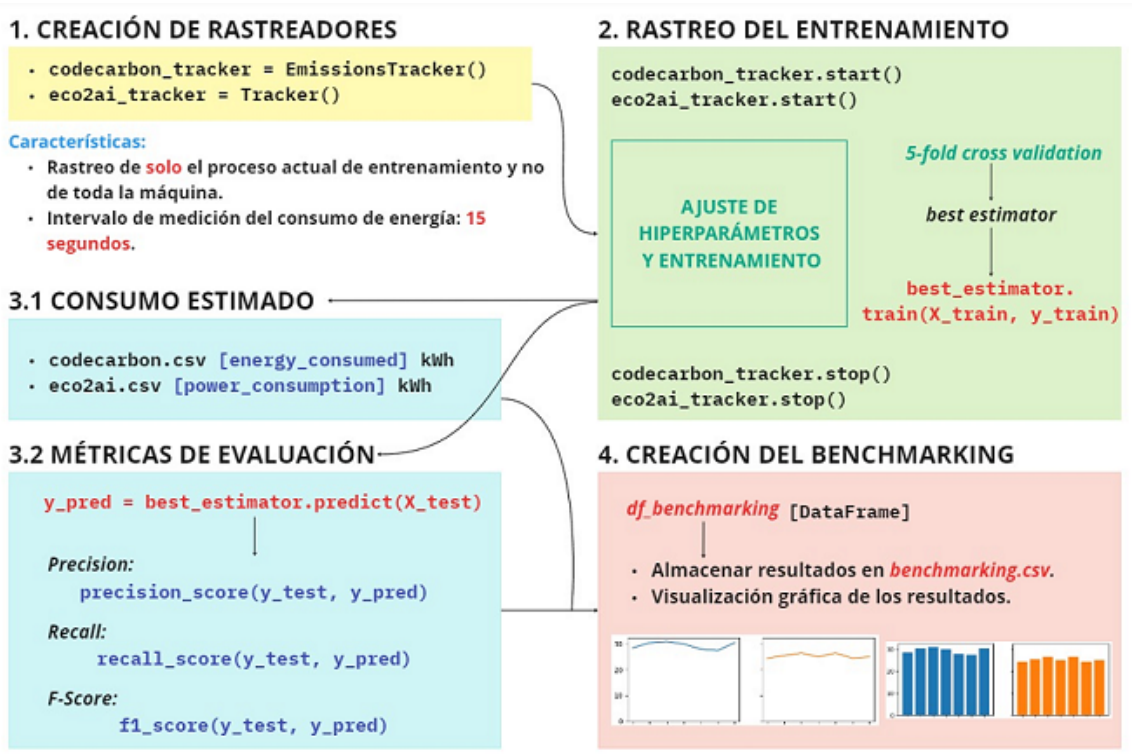


Figura 4.1: Proceso de desarrollo del banco de pruebas.

4.3 Resultados

A efectos de dar conclusión a este capítulo, se revela qué tipo de resultados numéricos han sido obtenidos durante la ejecución del banco de pruebas desplegado en la infraestructura escogida (3.2.3):

Tabla 4.1: Banco de pruebas final .

Algoritmos	CodeCarbon (kWh)	Eco2AI (kWh)	Precision	Recall	F Score
Logistic Regression	0,000274	0,000174	1	0,998299	0,999149
Random Forest	0,000441	0,000193	1	0,998299	0,999149
Support Vector Machines	0,000675	0,000273	1	0,998299	0,999149
Multilayer Perceptron	0,003086	0,004724	1	0,998299	0,999149

De igual modo, se agregan las gráficas inferiores de forma que se facilite la interpretación de los datos expuestos en la tabla superior:

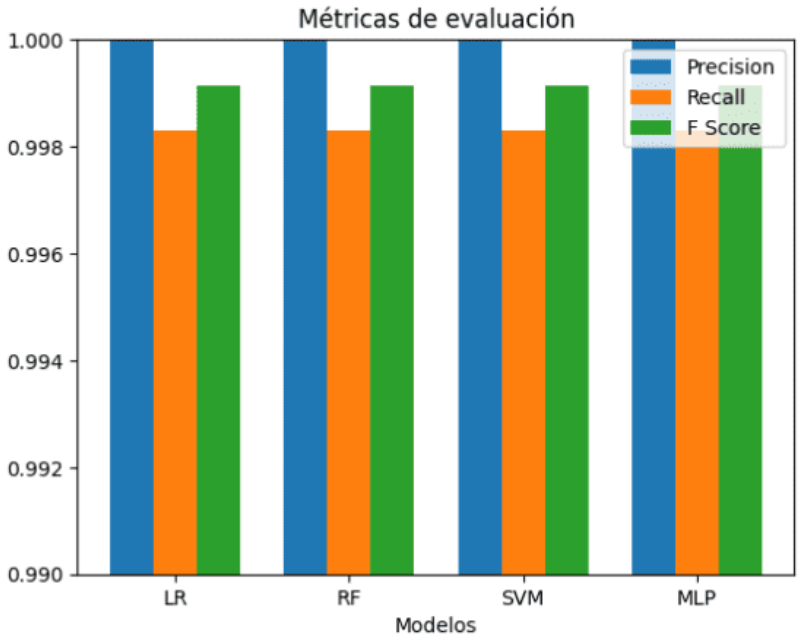


Figura 4.2: Comparación de modelos según métricas de evaluación.

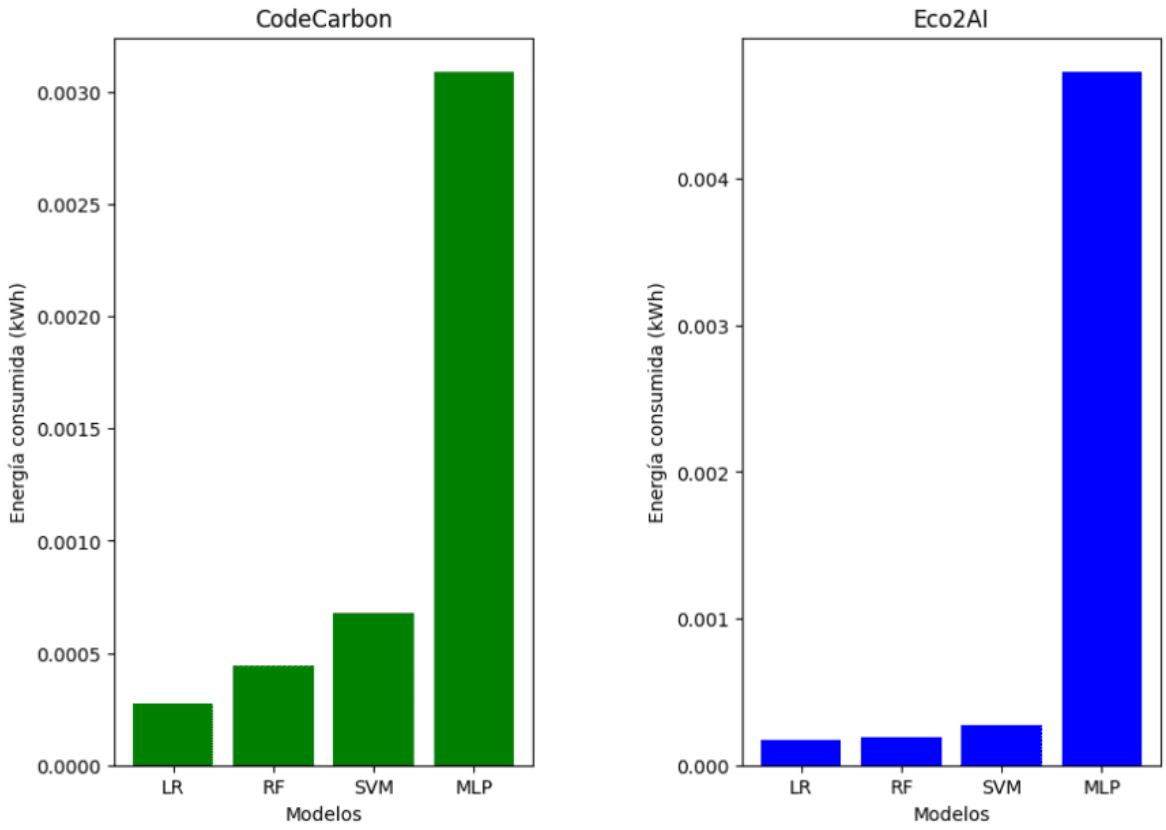


Figura 4.3: Comparación de modelos según el consumo estimado.

Aviso

Es vital subrayar que los datos presentados como resultado en 4.1 con respecto al consumo, **son solamente una de las múltiples estimaciones que han sido retornadas** por el programa durante las diversas ejecuciones efectuadas.

Esto significa que, **en cada ejecución, se han obtenido consumos de energía distintos, pero relativamente similares**. Por ello, se advierte que la información depositada en 4.1 es **aproximada**.

Al final, se deduce que el motivo principal detrás de este fenómeno es el **tiempo de ejecución**, ya que el cálculo del consumo depende de este factor y su valor puede variar en cada caso dependiendo de la capacidad de la máquina empleada.

4.4 Conclusiones

Conforme a los resultados abordados en el apartado anterior, se intentará dar respuesta a las preguntas planteadas en el último punto expuesto en 1.1.2:

Como punto de partida, si se lleva adelante un breve análisis sobre los datos divulgados en 4.1, es posible apreciar a simple vista que **los valores de consumo estimados para una**

misma configuración (*dataset* y algoritmo) por las herramientas **CodeCarbon** y **eco2AI**, aunque presentan la misma escala, **no son similares en términos numéricos**.

Por consiguiente, tras la realización de varias ejecuciones y considerando todos los aspectos descritos hasta el momento, se deduce la existencia de **dos causas probables** que desencadenan las discrepancias vistas entre las dos librerías:

1. **Metodologías de estimación**: Repasando los esquemas 3.15, 3.16 y 3.17, nos percatamos que ambos métodos presentan una estructura similar. Esto parece indicar que siguen la misma idea a la hora de calcular la huella de carbono producida durante el experimento.

   Sin embargo, si nos centramos en cómo llevan a cabo la computación del gasto energético originado por la infraestructura utilizada, vemos que aplican procedimientos distintos. En nuestro caso de uso:

   - **CodeCarbon** → CPUs → Linux → ✗ Ficheros Intel RAPL → *fall back mode*.
   - **Eco2AI** → CPUs → `os` y `psutil` (módulos Python).

2. **Duración del experimento**: Después de la exploración realizada sobre los ficheros resultado de CodeCarbon (`codecarbon.csv`) y eco2AI (`eco2ai.csv`), notamos que **la duración medida por ambas siempre difiere aproximadamente en un segundo**.

   La razón detrás de esta ligera variación puede deberse al hecho de **iniciar el rastreo con cada herramienta de forma secuencial**, en lugar de emprender dicho proceso en paralelo.

Por otro lado, es relevante destacar que, observando los distintos resultados alcanzados a lo largo del experimento, **no se logra deducir ningún detalle adicional con respecto al procedimiento de estimación seguido** por CodeCarbon y eco2AI.

La metodología que ha sido diseñada e implementada en este proyecto con el fin de comparar las herramientas escogidas, **no nos proporciona los medios ni la información necesaria para intuir nuevos detalles sobre el funcionamiento interno de las mismas**, aparte de las ya conocidas gracias a sus respectivas documentaciones.

Para terminar, si dirigimos nuestra atención a las figuras 4.2 y 4.3, se advierte rápidamente cómo se ha conseguido **la misma eficiencia de predicción con todos los modelos, a pesar de que haya algoritmos** como MLP que **han requerido un consumo de energía mucho mayor** para completar el proceso de entrenamiento.

En consecuencia, estos acontecimientos evidencian y nos llevan a concluir que **sí es posible utilizar modelos de aprendizaje automático energéticamente más eficientes sin perder calidad de predicción**, siempre que se realice un ajuste adecuado y se tenga en cuenta la complejidad del problema que se busca abordar.

Capítulo 5

Conclusiones y trabajos futuros

En este último capítulo se presentarán los objetivos que han sido finalmente completados y las enseñanzas que han sido adquiridas en el transcurso de este trabajo. Simultáneamente, se propondrán distintas mejoras e ideas con base en lo realizado, las cuales pueden dar lugar por sí mismas a futuros proyectos.

5.1 Consecución de objetivos

Después de completar el estudio teórico y práctico que ha involucrado este TFG, somos capaces de extraer una serie de conclusiones que nos permiten evaluar si hemos alcanzado los dos principales objetivos establecidos al comienzo de esta memoria.

En primera instancia, a pesar de las numerosas dificultades enfrentadas durante el proceso de experimentación, se ha logrado llevar a cabo con éxito el estudio de las herramientas disponibles para el cálculo del consumo provocado por los algoritmos de ML. Como prueba de ello, se expusieron en el capítulo 2 las tablas 2.7, 2.8 y 2.9, de las cuales deducimos y escogimos las dos mejores candidatas para participar en la siguiente etapa de este trabajo.

Al mismo tiempo, se ha cumplido con el desarrollo de un banco de pruebas que sigue la estructura y configuración expuestas en las figuras 3.19 y 4.1, que nos ha permitido comparar las dos herramientas seleccionadas a partir del consumo calculado y de las métricas de evaluación alcanzadas para un mismo grupo de *dataset* y algoritmo. De la misma manera, se ofrece en 4.4, de acuerdo con los resultados obtenidos e ilustrados en 4.3, las respuestas a las preguntas formuladas como objetivo en 1.1.

5.2 Aplicación de lo aprendido

Gracias al Grado en Ingeniería en Tecnologías de la Telecomunicación cursado, se ha tenido la oportunidad de aprender en asignaturas con distintos docentes cuyas enseñanzas

han hecho posible la elaboración de este TFG. De las numerosas materias a las que se ha asistido, destacan las siguientes:

1. **Servicios y Aplicaciones Telemáticas**: Asignatura impartida por Jesus M. González Barahona y Gregorio Robles Martínez, donde se introdujo por primera vez Python y se adquirieron las nociones básicas de desarrollo *software* a través del mismo. Los conocimientos alcanzados en estos contenidos han sido esenciales para materializar la parte práctica del proyecto.
2. **Introducción a la Bioingeniería**: Materia en la que se mencionó por primera vez, de la mano de Cristina Soguero Ruíz, el término *machine learning*, donde se explicaron los conceptos básicos en relación con este ámbito. La formación recibida en este curso fue la que inició e impulsó mi interés en este tema y sobre la que se fundamenta el plano teórico de este trabajo.
3. **Ingeniería de Sistemas de Información**: En esta ocasión, fue el profesor Pedro de las Heras quien transmitió todos los principios imprescindibles para emplear adecuadamente el sistema de control de versiones *Git*, los cuales han sido aplicados constantemente a lo largo del desarrollo emprendido.

5.3 Lecciones aprendidas

Acto seguido, se enumeran las competencias alcanzadas en virtud de este Trabajo de Fin de Grado:

1. Mejora de habilidades en programación científica y desarrollo de proyectos a través de PyCharm.
2. Instrucción en el uso de las librerías Python más populares en ciencia de datos, como pandas, Scikit-learn, numpy, etc.
3. Aprendizaje de los aspectos y conceptos básicos que engloba el *machine learning*.
4. Toma de conciencia sobre la situación actual respecto al gasto energético que implica la IA y sus posibles efectos en el medioambiente.
5. Conocimiento y utilización de las principales herramientas orientadas a la estimación del consumo durante el entrenamiento de modelos.
6. Evolución en la capacidad de redacción y desenvoltura con LaTeX.

5.4 Trabajos futuros

En definitiva, el proyecto presentado tiene la facultad de fomentar nuevas expansiones y mejoras, como las que se especifican seguidamente:

- Emplear una infraestructura que permita agregar en el *benchmarking* un mayor número de herramientas.

- Ampliar el banco de pruebas incluyendo una mayor cantidad de modelos representativos del aprendizaje profundo.
- Diseñar un procedimiento para determinar qué herramienta es la más exacta y precisa calculando el consumo producido tras el entrenamiento de un algoritmo.
- Estudiar con mayor profundidad la metodología aplicada por cada herramienta, con el propósito de conocer y entender los motivos reales detrás de las posibles discrepancias entre resultados.
- Llevar a cabo la ejecución de los rastreadores en paralelo, de modo que todas las herramientas que forman parte del experimento se encuentren en igualdad de condiciones y se eviten posibles conflictos entre los resultados.

Referencias

[1] Sumeet Kumar Agrawal. *Metrics to Evaluate your Classification Model to take the right decisions*. https://www.analyticsvidhya.com/blog/2021/07/metrics-to-evaluate-your-classification-model-to-take-the-right-decisions/. Dic. de 2022.

[2] Sergei Alyamkin et al. «2018 Low-Power Image Recognition Challenge». En: *CoRR* abs/1810.01732 (2018). arXiv: 1810.01732.

[3] Hyacinth Ampadu. *Random Forests Understanding*. https://ai-pool.com/a/s/random-forests-understanding. Mayo de 2021.

[4] SangGyu An. *Introduction to how an Multilayer Perceptron works but without complicated math*. https://medium.com/codex/introduction-to-how-an-multilayer-perceptron-works-but-without-complicated-math-a423979897ac. Oct. de 2021.

[5] Edmund L. Andrews. *AI's Carbon Footprint Problem*. https://hai.stanford.edu/news/ais-carbon-footprint-problem. Jul. de 2020.

[6] Lasse F. Wolff Anthony, Benjamin Kanding y Raghavendra Selvan. *Carbontracker: Tracking and Predicting the Carbon Footprint of Training Deep Learning Models*. ICML Workshop on Challenges in Deploying and monitoring Machine Learning Systems. arXiv:2007.03051. Jul. de 2020.

[7] ashwinsharmap. *StandardScaler, MinMaxScaler and RobustScaler techniques – ML*. https://www.geeksforgeeks.org/standardscaler-minmaxscaler-and-robustscaler-techniques-ml/. Abr. de 2023.

[8] Carolina Bento. *Multilayer Perceptron Explained with a Real-Life Example and Python Code: Sentiment Analysis*. https://towardsdatascience.com/multilayer-perceptron-explained-with-a-real-life-example-and-python-code-sentiment-analysis-cb408ee93141. Sep. de 2021.

[9] Aniruddha Bhandari. *Feature Engineering: Scaling, Normalization, and Standardization (Updated 2023)*. https://www.analyticsvidhya.com/blog/2020/04/feature-scaling-machine-learning-normalization-standardization/#h-what-is-feature-scaling. Abr. de 2023.

[10] Semen Budennyy et al. *Eco2AI: carbon emissions tracking of machine learning models as the first step towards sustainable AI*. 2022. DOI: 10.48550/ARXIV.2208.00406. URL: https://arxiv.org/abs/2208.00406.

[11] Adrián Ceja. *¿Por qué los Data Scientist utilizan Python?* https://www.neoland.es/blog/por-que-los-data-scientist-utilizan-python. Ago. de 2020.

[12] University of Copenhagen. *Students develop tool to predict the carbon footprint of algorithms*. https://news.ku.dk/all_news/2020/11/students-develop-tool-to-predict-the-carbon-footprint-of-algorithms/. Nov. de 2020.

[13] The Economist. *The cost of training machines is becoming a problem*. https://www.economist.com/technology-quarterly/2020/06/11/the-cost-of-training-machines-is-becoming-a-problem. Jun. de 2020.

[14] FySelf. *CodeCarbon: for a cleaner technology*. https://medium.com/fyselfcom/codecarbon-for-a-cleaner-technology-35b47c8b3738. Dic. de 2020.

[15] Ishan Gaba. *What is GitHub And How To Use It?* https://www.simplilearn.com/tutorials/git-tutorial/what-is-github. Sep. de 2021.

[16] Rohith Gandhi. *Support Vector Machine — Introduction to Machine Learning Algorithms*. https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47. Jun. de 2018.

[17] Eva García-Martín et al. «Estimation of energy consumption in machine learning». En: *J. Parallel Distributed Comput.* 134 (2019), págs. 75-88. DOI: 10.1016/j.jpdc.2019.07.007.

[18] Aurélien Géron. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow*. O'Reilly Media, Inc., 2022.

[19] Karen Hao. *What is machine learning?* https://www.technologyreview.com/2018/11/17/103781/what-is-machine-learning-we-drew-you-another-flowchart/. Nov. de 2018.

[20] Peter Henderson et al. «Towards the Systematic Reporting of the Energy and Carbon Footprints of Machine Learning». En: *CoRR* abs/2002.05651 (2020). arXiv: 2002.05651.

[21] Quantilus Innovation. *Why is Machine Learning Important and How Will It Impact Business*. https://quantilus.com/why-is-machine-learning-important-and-how-will-it-impact-business/. Ago. de 2020.

[22] Alexandre Lacoste et al. «Quantifying the Carbon Emissions of Machine Learning». En: *CoRR* abs/1910.09700 (2019). arXiv: 1910.09700.

[23] Loïc Lannelongue, Jason Grealey y Michael Inouye. «Green Algorithms: Quantifying the carbon emissions of computation». En: *CoRR* abs/2007.07610 (2020). arXiv: 2007.07610.

[24] Minsky Marvin y A Papert Seymour. «Perceptrons». En: *Cambridge, MA: MIT Press* 6 (1969), págs. 318-362.

[25] GreenPeace México. *AI's Carbon Footprint Problem*. https://www.greenpeace.org/mexico/blog/9386/huella-de-carbono/. Dic. de 2020.

[26] Sebastian Raschka & Vahid Mirjalili. *Python Machine Learning*. Packt Publishing Ltd., 2019.

[27] Nutan. *TensorFlow Deep Learning Model With IRIS Dataset*. https://medium.com/@nutanbhogendrasharma/tensorflow-deep-learning-model-with-iris-dataset-8ec344c49f91. Mayo de 2021.

[28] Rob Matheson | MIT News Office. *Reducing the carbon footprint of artificial intelligence*. https://news.mit.edu/2020/artificial-intelligence-ai-carbon-footprint-0423. Consultado: 25-9-2021. Abr. de 2020.

[29] David R. Pugh. *Getting Started with Conda*. https://towardsdatascience.com/managing-project-specific-environments-with-conda-b8b50aa8be0e. Abr. de 2020.

[30] Vatsal Raval. *When to use MLP, CNN or RNN?* https://medium.com/@iamvatsalraval/when-to-use-mlp-cnn-or-rnn-4aadba56073e. Ene. de 2019.

[31] Press Releases. *Top AI Experts Create CodeCarbon, a Tool to Track and Reduce Computing's CO2 Emissions*. https://www.bcg.com/press/1december2020-top-ai-experts-create-codecarbon. Dic. de 2020.

[32] sanyamdhawan. *Splitting Data for Machine Learning Models*. https://www.geeksforgeeks.org/splitting-data-for-machine-learning-models/. Ago. de 2020.

[33] Abhishek Sharma. *Random Forest vs Decision Tree | Which Is Right for You?* https://www.analyticsvidhya.com/blog/2020/05/decision-tree-vs-random-forest-algorithm/. Abr. de 2023.

[34] Emma Strubell, Ananya Ganesh y Andrew McCallum. «Energy and Policy Considerations for Deep Learning in NLP». En: *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*. Ed. por Anna Korhonen, David R. Traum y Lluís Màrquez. Association for Computational Linguistics, 2019, págs. 3645-3650. DOI: 10.18653/v1/p19-1355.

[35] Anuganti Suresh. *What is a confusion matrix?* https://medium.com/analytics-vidhya/what-is-a-confusion-matrix-d1c0f8feda5. Nov. de 2020.

[36] A. Aylin Tokuç. *Splitting a Dataset into Train and Test Sets*. https://www.baeldung.com/cs/train-test-datasets-ratio. Nov. de 2022.

[37] Martin Jaggi Tristan Trebaol Mary-Anne Hartley y Hossein Shokri Ghadikolaei. «A tool to quantify and report the carbon footprint of machine learning computations and communication in academia and healthcare». En: *Infoscience EPFL: record 278189* (2020).

[38] Pavan Vadapalli. *Scikit-learn in Python: Features, Prerequisites, Pros & Cons*. https://www.upgrad.com/blog/scikit-learn-in-python/. Jun. de 2020.

[39] Pavan Vadapalli. *Support Vector Machines: Types of SVM* [*Algorithm Explained*]. https://www.upgrad.com/blog/support-vector-machines/. Oct. de 2022.

[40] Data Science Wizards. *A Guide to Data Splitting in Machine Learning*. https://medium.com/@datasciencewizards/a-guide-to-data-splitting-in-machine-learning-49a959c95fa1. Nov. de 2022.

[41] Grace Zhang. *What is the kernel trick? Why is it important?* https://medium.com/@zxr.nju/what-is-the-kernel-trick-why-is-it-important-98a98db0961d. Nov. de 2018.