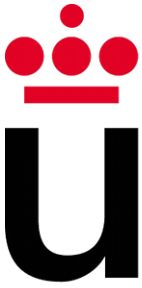

Universidad
Rey Juan Carlos

INGENIERÍA DE ROBÓTICA SOFTWARE

Curso Académico 2021/2022

Trabajo Fin de Grado

UN TÍTULO DE PROYECTO LARGO EN DOS LÍNEAS

Autor/a : Nuria Díaz Jérica
Tutor/a : Dr. Nombre del Profesor/a

Trabajo Fin de Grado/Máster

Entrenamiento y Aplicación de Modelos de Aprendizaje Automático en Dispositivos con Capacidad de Cómputo

Autor/a : Nuria Díaz Jérica
Tutor/a : Dr. Nombre del profesor/a

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

Presidente:

Secretario:

Vocal:

y habiendo obtenido la siguiente calificación:

Calificación:

Fuenlabrada, a de de 20XX

Aquí normalmente
se inserta una dedicatoria corta

Agradecimientos

Aquí vienen los agradecimientos...

Hay más espacio para explayarse y explicar a quién agradeces su apoyo o ayuda para haber acabado el proyecto: familia, pareja, amigos, compañeros de clase...

También hay quien, en algunos casos, hasta agradecer a su tutor o tutores del proyecto la ayuda prestada...

Resumen

Este Trabajo de Fin de Grado tiene como objetivo valorar el dispositivo de capacidades ruducidas, Raspberry Pi para poder determinar si tiene la potencia suficiente como para ser capaz de desarrollar tareas de Machine Learning. Este conocimiento será muy útil para determinar si se puede utilizar este dispositivo tan popular en diferentes tipos de aplicaciones tanto para Machine Learning como de IoT pudiendo facilitar, por ejemplo, la implementación de casas y ciudades inteligentes (Smart cities) creando ciudades más sostenibles, conectadas y optimizadas, entre otras muchas aplicaciones que se le pueden dar a esto.

Para ello, se utilizará `scikit-learn` para la creación de los modelos. En total se crearán cuatro modelos diferentes de aprendizaje automático supervisado, cada uno será entrenado con tres *datasets* dsitintos. Para poder validar la Raspberry como dispositivo capaz de realizar este tipo de tareas, se le someterá a varias pruebas durante el entrenamiento de los modelos. De estas pruebas se obtendrán resultados que nos ayudarán a valorar las capacidades de la Raspberry y dar una respuesta a la duda que se plantea en este proyecto.

En paralelo se realizará lo mismo en un dispositivo de mayores capacidades, para poder comparar ambos comportamientos y detrminar como de bien o mal actua la Raspberry.

Además no solo se pondrá a prueba la capacidad de la Raspberry para entrenar modelos, sino que también se comprobará si es lo suficientemente potente como para poder generar su propio *dataset*, por medio de información recogida por los sensores. Y entrenar los modelos mencionados con este nuevo *dataset*.

...

El proyecto está desarrollado de forma que cualquiera, con los componentes hardware necesarios, se pueda realizar las pruebas sin ningún tipo de dificultad.

Aquí viene un resumen del proyecto. Ha de constar de tres o cuatro párrafos, donde se presente de manera clara y concisa de qué va el proyecto. Han de quedar respondidas las siguientes preguntas:

- ¿De qué va este proyecto? ¿Cuál es su objetivo principal?

- ¿Cómo se ha realizado? ¿Qué tecnologías están involucradas?
- ¿En qué contexto se ha realizado el proyecto? ¿Es un proyecto dentro de un marco general?

Lo mejor es escribir el resumen al final.

Summary

Here comes a translation of the “Resumen” into English. Please, double check it for correct grammar and spelling. As it is the translation of the “Resumen”, which is supposed to be written at the end, this as well should be filled out just before submitting.

Índice general

Índice de figuras

Índice de fragmentos de código

Capítulo 1

Introducción

Una de las herraminetas más importantes que nos proporcionan las ciencias computacionales a día de hoy es el Machine Learning (o Aprendizaje Automático en español), cuyo origen se encuentra en la Inteligencia Artificial.

Por medio de una gran cantidad de datos los ordenadores, haciendo uso del aprendizaje automático, pueden llegar a identificar patrones y elaborar nuevas predicciones, sin ser específicamente programados para ello. Ofreciendo una gran variedad de oportunidades y posibilidades a la hora de realizar aplicaciones por medio dme estos métodos. Muchas de estas aplicaciones las podemos observar en nuesto día a día, como por ejemplo en las redes sociales, sugiriendonos contenido que puede ser de nuestro interés, en el correo electrónico para proteger la cuenta de correos sospechosos o fraudulentos, en los videojuegos para dar vida a los bots... Y por supuesto todo esto ayuda a desarrollar la tecnología de un futuro no tan lejano como pueden ser los coches autónomos o robots más inteligentes.

Sin embargo, la gran cantidad de recursos y consumo de energía que necesitan estos modelos para poder realizar las predicciones provoca que no se pueda llegar a implementar estos algoritmos en cualquier dispositivo. Haciendo que muchas aplicaciones tengan que ser descartadas directamente por no cumplir ciertos requisitos mínimos como para que un modelo de aprendizaje automático pueda ser entrenado. De este problema nace lo que se conoce como Tiny Machine Learning.

Dentro del mundo del Machine Learning (ML) se puede definir un subcampo de investigación denominado como Tiny Machine Learning, cuyo objetivo es utilizar los algoritmos de aprendizaje automático en dispositivos más baratos, y por lo tanto con recursos, capacidad de almacenamiento y energía mucho más limitados que una máquina estándar. Esto supone un reto puesto que el proceso de entrenamiento de un modelo de ML es un proceso costoso a nivel de recursos, poniendo en un aprieto a los disitivos que lo quieran ejecutar, y más si además se pretende que dicho dispositvo también recoja y procese los datos antes de

entrenar con ellos para poder obtener una respuesta inmediata en vez de tener que esperar a que los datos se envien y procesen en la nube.

Podemos encontrar aplicaciones de TinyML tan comunes como "OK Google" o "Alexa", además de otras aplicadas a diferentes campos como pueden ser la agricultura, por ejemplo detectando si hay alguna planta enferma, en el mantenimiento industrial, prediciendo anticipadamente fallos en la maquinaria, o también en la atención médica, un ejemplo de esto es el proyecto "Solar Scare Mosquito"[1] en el cual se detectan las condiciones de reproducción de los mosquitos, que pueden propagar enfermedades, haciendo que el propio dispositivo mueva el agua para evitar que se reproduzcan.

Luego el objetivo del TinyML es llevar los modelos de aprendizaje automátivo al extremo, utilizando dispositivos basados en microcontroladores alimentados por bateria para realizar tareas de ML con capacidad de respuesta en tiempo real. Pudiendo beneficiarnos de su gran portabilidad, bajo consumo y mayor seguridad puesto que no es necesario enviar los datos a la nube, lo que puede ser muy relevante cuando se trabaja en aplicaciones de IoT (Internet of Things).

Los dispositivos de IoT permiten monitorear continuamente un entorno utilizando pequeños sensores. Estos dispositivos están conectados a una red por medio de la cual comparten continuamente la información obtenida. Añadiendo ML a un dispositivo de estas características permite detectar patrones en todos esos datos recogidos, pudiendo desarrollar una amplia gama de aplicaciones inteligentes y experiencias de usuario mejoradas. Dado que estos dispositivos son normalemente también de características reducidas, es clave el uso de técnicas de TinyML para su desarrollo.

Actualmente hay unos 250 billones de dispositivos embebidos activos en el mundo. Estos dispositivos están continuamente recogiendo grandes cantidades de datos, por lo que procesar toda esa información en la nube representa todo un deafío. Tiny Machine Learning podria poner solución a toda esa cantidad de datos que se están "desperdiciando" al no poder se tratados ni procesados, para poder crear nuevas aplicaciones en diferentes tipos de insutrias.

Por lo que debido a la gran importancia que están adquiriendo todos estos elementos, es importante conocer qué dispositivos cumplen las caracterísitcas necesarias para poder desarrollar este tipo de tareas. Es por ello que en este TFG se comprobará la capacidad de uno de los dispositivos de capacidades reducidas más populares del mercado, la Raspberry Pi 4B.

[1] https://hackaday.io/project/174575-solar-scare-mosquito-20

1.1 Objetivos del proyecto

1.1.1 Objetivo general

El objetivo de este Trabajo de Fin de Grado es evaluar la capacidad y eficiencia de la Raspberry Pi 4B para entrenar modelos de aprendizaje automático. Para ello se hará uso de diversos *datasets* con diferentes cantidades de datos para observar el comportamiento ante diferentes entradas de datos. Y de esta forma poder concluir si este dispositivo puede ser empleado para desarrollar tareas de Machine Learning e IoT o no.

Aquí vendría el objetivo general en una frase: Mi Trabajo Fin de Grado/Master consiste en crear de una herramienta de análisis de los comentarios jocosos en repositorios de software libre alojados en la plataforma GitHub.

Recuerda que los objetivos siempre vienen en infinitivo.

1.1.2 Objetivos específicos

Los objetivos específicos se pueden entender como las tareas en las que se ha desglosado el objetivo general. Y, sí, también vienen en infinitivo.

Lo mejor suele ser utilizar una lista no numerada, como sigue:

- Generar diferentes modelos de aprendizaje automático utilizando la Raspberry.
- Poner a prueba los algoritmos que entrenan los modelos, de forma que se pueda comprobar el comportamiento de la Raspberry bajo diferentes condiciones.
- Crear y poner a prueba los mismos modelos de aprendizaje automático, pero esta vez en un dispositivo con mayor capacidad.
- Crear un *dataset* propio para entrenar los modelos, utilizando varios sesnores.

1.2 Planificación temporal

A inicios de este curso, cuando tuve que meditar sobre de qué tema queria realizar mi TFG, me dí cuenta de que uno de los que más me habían gustado durante la carrera había sido Machine Learning, que justo había comenzado a verlo durante ese primer cuatrimestre. Es un tema que me interesa mucho debido a la infinidad de aplicaciones que le veo y

de la capacidad que hay detrás de todos esos algoritmos para poder facilitar y mejorar la vida de las personas.

Sin embargo, no tenía del todo claro de qué forma enfocarlo. Fue entonces cuando descubrí que Katia junto con Felipe ofrecian un proyecto sobre este tema y no dudé en informarme sobre exactamente en qué estaba enfocado el trabajo. Después de contactar con Katia y proponerme varias perspectivas, me decanté por el enfoque de Tiny Machine Learning debido a que me llamaba la atención la posibilidad de poder observar las capacidades de la Raspberry para realizar este tipo de tareas ya que, como se verá en los siguientes capitulos, es un dispositivo económico con el que se podrian amplificar las utilidad de estas técnicas.

Una vez acordado el tema lo primero que hice, a recomendación de Katia, fue informarme a cerca de cómo se podian aplicar técnicas de Machine Learning en la Raspberry Pi. En Noviembre me entregaron una caja con todo el material necesario para montar y utilizar la Raspberry, con lo que el primer objetivo fue prepararla y comprobar que todo funcionase correctamente creando algoritmos simples, como por ejemplo encender y apagar un led. Además, esta caja también traía tres sensores diferentes para utilizar con la Raspberry, en concreto venia una fotoresistencia, un sensor de humedad y un sensor de temperatura, con los que realicé varias pruebas para verificar que funcionaban correctamente.

A partir de este momento y tras la primera tutoría con ambos profesores comenzó el proceso de instalación y preparación del entorno para poder implementar algoritmos de aprendizaje automático, en el que se tuvieron que superar algunas dificultades. Con el entorno instalado se procedió a implementar y realizar las pruebas oportunas.

Primero se comenzó creando los modelos de aprendizaje automático, en un principio eran tres pero más adelante se añadió un nuevo modelo de aprendizaje. Una vez que el código de los modelos estaba implementado fue momento de comenzar a realizar las pruebas para observar sus comportamientos. Estas pruebas duraron varios meses puesto que no terminaba de quedar del todo claro cómo estaba reaccionando la Raspberry ante las diferentes pruebas, es por ello que durante este periodo se utilizaron diferentes *datasets* para seguir haciendo más pruebas y entender lo que estaba sucediendo.

Una vez esclarecidas las dudas se trató de ir más allá haciendo que la Raspberry generase su propio *dataset* y entrenase con él. Este fue el último paso antes de centrarme completamente en la redacción de la memoria. A lo largo de los meses, además de seguir avanzando con la investigación, también trataba de ir redactando poco a poco la memoria.

A pesar de que el desarrollo de este trabajo comenzó en Noviembre no fue hasta finalizar el primer cuatrimestre cuando le empecé a dedicar como mínimo hora y media o dos horas todos los días, salvo algunas escepciones. Siendo los fines de semana cuando le dedicaba aún más tiempo por no tener ninguna otra ocupación. En la Figura 1.1 se puede observar un diagrama de Gantt donde se puede visualizar mejor el desarrollo y duración

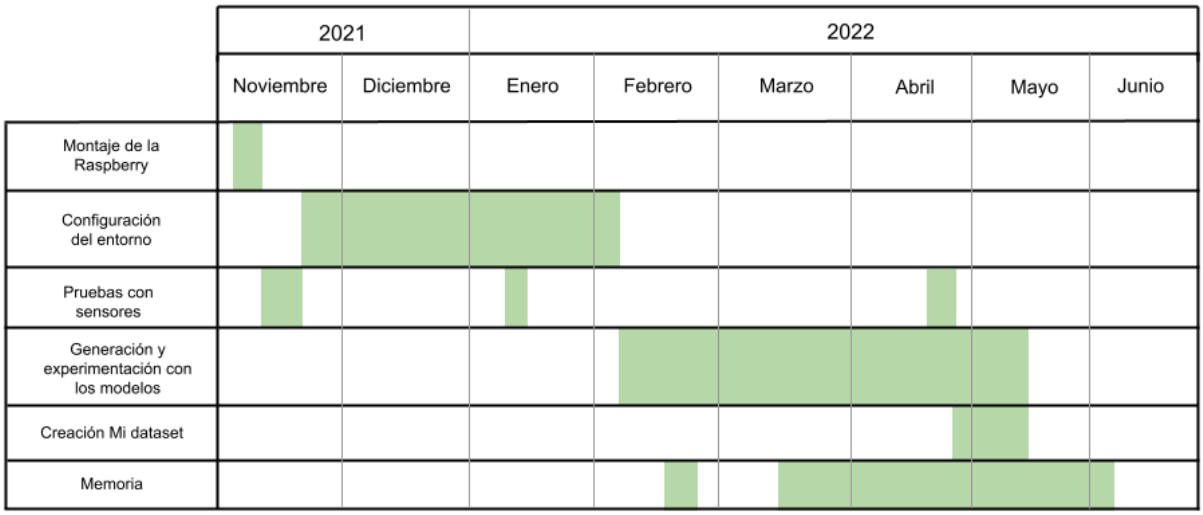

Figura 1.1: Diagrama de Gantt donde se representa la planificación temporal llevada a cabo en este proyecto.

de cada una de las etapas del proyecto.

1.3 Estructura de la memoria

Al inicio de esta memoria se puede leer un resumen de la misma que indica en qué consiste el trabajo desarrollado. A continuación se sitúan los agradecimientos además de tres índices, el general, el de las figuras y el índice de fragmentos de código. Por último, se encuentran los 6 capítulos en los que se divide la memoria.

- **Capítulo 1: Introducción.** En este capítulo se realiza una descripción general de en qué consiste el proyecto. Especificando los objetivos del proyecto, a planificación del mismo y cual es la estructura de la memoria.
- **Capítulo 2: Estado del arte.** Aquí se explican todas las tecnologías utilizadas para el desarrollo del trabajo.
- **Capítulo 3: Diseño e implementación.** Se describe de forma detallada la arquitectura general del proyecto, la configuración del entorno en la Raspberry para llevar a cabo la investigación, así como el código empleado para generar los modelos de aprendizaje y los *datasets* utilizados para entrenarlos.

- **Capítulo 4: Experimentos y validación**. Por una parte se explica en que consisten las pruebas realizadas tanto en la Raspberry como en el portátil, además de explicar cómo y qué código se utiliza para ejecutarlas. Por otra parte se exponen y explican los resultados obtenidos para cada uno de los modelos generados por los diferentes *datasets* en cada una de las pruebas.

- **Capítulo 5: Conclusiones**. Se exponen las conclusiones obtenidas gracias a esta investigación. Ademas de detallar posibles proyectos derivados de este proyecto.

- **Capítulo 6: Anexo.** Se detallan algunas características y especificaciones para la instalación correcta del entorno.

1.4 Enlaces de interés

En el siguiente URL: se puede encontrar un resumen general en inglés de todo aplicado y desarrollado en este trabajo. Además desde ese enlace también se puede acceder al repositorio de *GitHub* que contiene este proyecto asi como al propio pdf de la memoria.

Capítulo 2

Estado del arte

En este capítulo se describen todas las tecnologías utilizadas en el desarrollo de este proyecto.

2.1 Machine Learning

Para saber si la Raspberry es un dispoditivo lo suficientemente potente para realizar tareas de Machine Learning primero debemos tener claros estos conceptos para poder entender el proyecto.

Machine Learning o aprendizaje automático, es una disciplina del campo de la Inteligencia Artificial que permite a las máquinas aprender sin ser explícitamente programadas para ello. Para aprender utilizan algoritmos que permiten a los ordenadores identificar los patrones que existen en grandes cantidades de datos y utilizar estos patrones para poder elaborar predicciones del futuro.

Los datos que se proporcionan al algoritmo normalmente son ficheros con varias filas y columnas. Cada fila representa un ejemplo de datos que se pasa al modelo para entrenar. Mientras que si posee varías columnas significa que hay varios valores que definen un ejemplo, se dice que estas columnas son las caracterísitcas del *dataset*.

Un ejemplo de lo que contiene un *dataset* es la tabla 2.1, donde se proporciona información sobre cuando un niño va a jugar a la pelota o no. En este caso tendríamos tres características (previsión, temperatura y amigos), la columna jugar es la decisión que tome el niño ante los valores de esas caracterísitcas. Por lo tanto, cada fila define una situación diferente, es decir es un ejemplo de lo que puede suceder.

Previsión	Temperatura	Amigos	Jugar
Soleado	25 ºC	2	Si
Lluvioso	12 ºC	0	No
Nublado	18 ºC	4	Si

Tabla 2.1: Ejemplo de un *dataset*

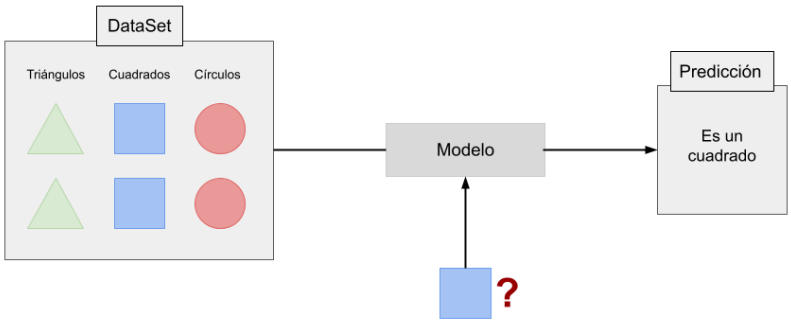

Figura 2.1: Estructura aprendizaje supervisado para clasificación.

Los algoritmos de Machine Learning se dividen en tres categorías:

- **Aprendizaje supervisado**. En este tipo de aprendizaje se le proporciona al algoritmo un conjunto de datos de entrada con su salida correspondiente. De esta forma el programa puede "aprender" la relación que existe entre la salida y el conjunto de entradas. Dependiendo del tipo de salida podremos encontrar dos clases de modelos: modelos de regresión o de clasificación.

  Si la salida es un valor numérico continuo, estaríamos ante un problema de regresión en el que se busca obtener la recta que mejor se ajusta a los datos de entrada para poder predecir otros valores. Si, en cambio, la salida es una etiqueta de clase, es decir un valor discreto, se correspondería a un problema de clasificación, en los que se busca estimar a qué clase pertenecen los datos de entrada.

  Un ejemplo de este último tipo se puede ver en la Figura 2.1, donde cada una de las formas serían las entradas y las etiquetas de: triángulo, cuadrado o círculo serían la salidas, cada dato de entrada tiene asignada una de estas etiquetas. Todo esto formaría lo que se denomina *dataset* que es el conjunto de datos que se pasan al modelo para entrenar y poder encontrar esas relaciones entre entradas y salidas. En este caso cuando al modelo se le pregunte a qué clase pertenece una figura (del mismo tipo que ha visto durante el entrenamiento), si es lo suficientemente robusto, debería de ser capaz de identificarlo como un cuadrado.

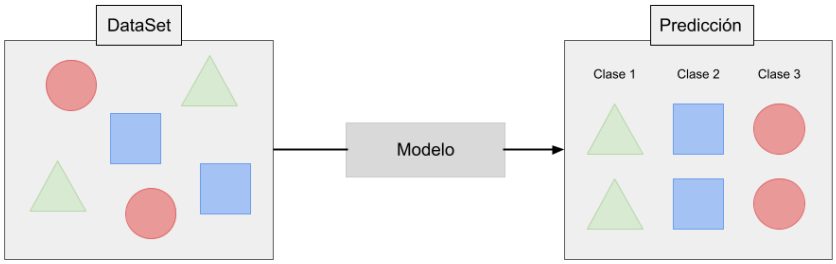

Figura 2.2: Estructura aprendizaje no supervisado para segmentación.

Existen multitud de modelos de Machine Learning que sirven para resolver problemas mediante aprendizaje supervisado, como por ejemplo algoritmos de Regresión logística, Árboles de decisión, Máquinas de soporte vectorial...

- **Aprendizaje no supervisado**. Al algoritmo de aprendizaje se le proporciona únicamente el conjunto de datos de entrada, sin ningún tipo de dato de salida. El objetivo es que el método de aprendizaje utilizado detecte posibles patrones de interés en el conjunto de datos de entrada.

  Hay principalmente dos tipos de tareas que estos algoritmos tratan de realizar. Una de ellas es el clustering (o segmentación), que sirve para encontrar diferentes grupos dentro de los datos de entrada. En la Figura 2.2 se muestra un ejemplo esquemático de la ejecución de un problema de este estilo, donde una vez más las entradas serían cada una de las formas pero en esta ocasión no están etiquetadas como veíamos en la Figura 2.1. El modelo. en esta ocasión, viendo las características de cada uno de los datos, los agrupará en diferentes clases.

  La otra misión que puede tener este tipo de aprendizaje es reducir la dimensionalidad de los datos, es decir, reducir el número de características del conjunto de datos de entrada asumiendo que muchas de ellas son redundantes y por lo tanto no aportan información nueva.

  Al igual que en aprendizaje supervisado existen varios algoritmos que buscan resolver estas tareas, como por ejemplo el método KNN, K-Means...

- **Aprendizaje por refuerzo**. El algoritmo aprende a desarrollar la tarea que se le asigna en base a un esquema de recompensas y penalizaciones ante las decisiones que toma el programa en cada una de las iteraciones. Luego, por cada acción que realice el modelo recibirá una recompensa o una penalización, de esta forma podrá tener conocimiento de si ha tomado una buena o mala decisión. El objetivo es que llegue a realizar la tarea encomendada consiguiendo la mayor recompensa posible.

Gracias a que todos estos algoritmos pueden estar aprendiendo constantemente de nuevos datos consiguen ir perfeccionando su comportamiento, lo que hace que las técnicas de Machine Learning sean cada vez más importantes debido a la multitud de aplicaciones que se le pueden dentro de la robótica, los vehículos autónomos, diagnósticos médicos, markenting....

2.2 Modelos de aprendizaje automático

Aunque existen muchos algoritmos para generar modelos de aprendizaje automático, en este proyecto vamos a utilizar concretamente cuatro, uno de Regresión logística, otro de Máquinas de soporte vectorial, Gradient boosting y un último de Random forest. Para ello se utilizará la librería de `scikit-learn`, que veremos más adelante.

2.2.1 Regresión logística

Modelo de aprendizaje supervisado que realiza tareas de clasificación binaria, es decir solo existen dos posibles clases. Esta técnica busca una función acotada (normalmente entre 0 y 1) que divida los datos de ambas clases de forma bien diferenciada. Normalmente dicha función se denomina como logística o sigmoide y se define de la siguiente forma:

$$h_w(x^{(i)}) = \frac{1}{1 + e^{\sum_{j=0}^{d} w_j x_j^{(i)}}} \tag{2.1}$$

Dicha expresión devolverá un valor entre cero y uno. Según lo próximo que este el valor a cero o uno pertenecerá a una clase u otra.

Los parámetros representados por w son los pesos, unos valores númericos que utiliza la función para realizar la predicción. Cada peso está multiplicando a una caracerística, simbolizada mediante x_j. Luego, para conocer a qué clase pertenece el ejemplo $x^{(i)}$ habrá que aplicar la fórmula anterior, por lo que se tendrá que realizar el sumatorio de todos los pesos multiplicados por sus respectivas caracterísitcas (desde la cero hasta la d-ésima).

Durante la fase de entrenamiento el modelo ajusta los pesos mediante un método de optimización, a fin de que las predicciones que se realicen mediante ellos tengan el mínimo error posible. Para obtener los valores óptimos de los pesos se utiliza el método de descenso por gradiente, mediante el cual se irán actualizando los valores de los pesos conforme vaya entrenando con más ejemplos. La función de descenso por gradiente se define de la siguiente manera:

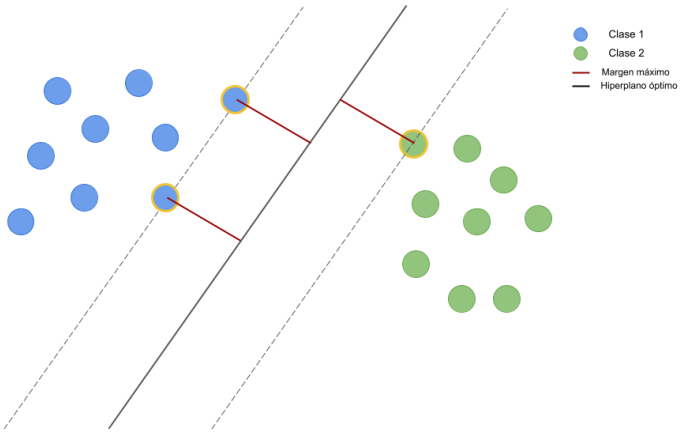

Figura 2.3: Ejemplo clasificación con Máquinas de soporte vectorial

$$w_j = w_j - \alpha \frac{1}{n} \sum_{i=1}^{n} (h_w(x^{(i)}) - y^{(i)}) x_j^{(i)} \tag{2.2}$$

Para calcular los pesos se utiliza el error que existe entre la salida predecida con los pesos estimados hasta ahora $(h_w(x^{(i)}))$ y el valor real, es decir, la salida que verdaderamente corresponde a ese ejemplo $(y^{(i)})$. También es necesario definir el valor del ratio de aprendizaje (α), si se le da un valor muy elevado puede llegar a no converger el modelo en ninguna solución. Esto se realiza para todos los valores de j, desde $j = 0$ hasta $j = d$, que es el número total de caracterísitcas.

Se dará por finalizada la fase de aprendizaje cuando se consiga que los valores de los pesos minimicen una función de coste:

$$J(w) = \frac{1}{2} \sum_{i=1}^{n} (h_w(x^{(i)}) - y^{(i)})^2 \tag{2.3}$$

2.2.2 Máquinas de soporte vectorial

Modelo de aprendizaje supervisado utilizado para resolver tareas de clasificación binaria, aun que existen máquinas de vector soporte para resolver problemas de regresión o de clasificación multiclase. Se basa en la generación de un hiperplano que separe de forma óptima los puntos de una clase respecto de otra. Es decir, que exista la máxima distancia entre el hiperplano y los puntos más cercanos a este, los puntos más cercanos al hiperplano de separación se denominan como vectores soporte. Luego el hiperplano óptimo pasará justo por el medio de los vectores soporte de ambas clases.

En la Figura 2.3 se puede ver un ejemplo de separación óptimo entre ambas clases, sien-

do los círculos rodeados en amarillo los vectores soporte. Todos tienen la máxima distancia posible con el hiperplano generado, que es capaz de separar perfectamente ambas clases. El hiperplano de separación se puede definir como una función lineal:

$$h(x) = \sum_{i=0}^{n} w_i x_i \tag{2.4}$$

En esta ocasión también se utilizan unos pesos (w) para encontrar dicho hiperplano.

Sin embargo, hay veces puede llegar a ser imposible hallar un hiperplano que divida correctamente las clases. Ante estos casos la resolución del problema se encuentra en aumentar la dimensionalidad de los datos para comprobar si en esa nueva dimensión existe un hiperplano capaz de separarlos adecuadamente.

2.2.3 Gradient boosting

Técnica de aprendizaje supervisado que se utiliza tanto para problemas de regresión como de clasificación. Se basa en la combinación de modelos predictivos débiles, normalmente árboles de decisión, para crear un modelo predictivo fuerte. Los árboles de decisión son uno de los algoritmos más utilizados para la toma de decisiones debido a su sencilla implementación y fácil interpretación. Dado un conjunto de datos, se crean diagramas lógicos que sirven para representar una serie de condiciones sucesivas para resolver un problema. Por ejemplo en la Figura 2.4 hay representado un árbol de decisión que sirve para poder saber si podemos salir a la calle a correr.

En Gradient boosting se generan árboles de decisión de forma secuencial, haciendo que cada árbol corrija los errores del árbol anterior. De forma general suelen ser árboles de un máximo de tres niveles de profundidad.

2.2.4 Random forest

Random forest se puede usar tanto en problemas de clasificación como de regresión. Utiliza un conjunto de árboles de decisión que se generan de forma paralela. Cada uno de los árboles de decisión se entrena con un subconjunto aleatorio de los datos de entrenamiento.

Cuando se quiera, por ejemplo, estimar la clase a la que pertenece un dato cada uno de los árboles "votará" por la clase a la que cree que el dato pertenece. La clase con mayoría de votos es la predicción final del modelo. Por lo tanto, aumentando el número de árboles de

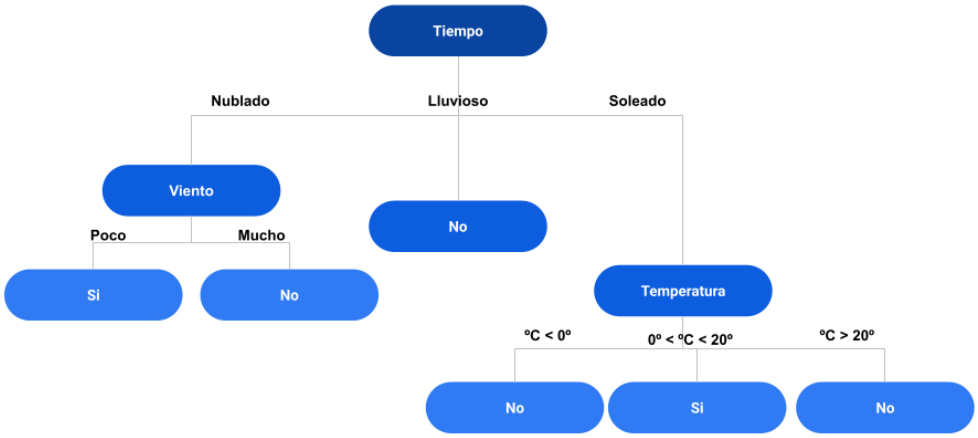

Figura 2.4: Ejemplo árbol de decisión.

decisión se aumenta la precisión del modelo. Con lo que se consigue construir un modelo robusto a partir de varios modelos que no tienen por qué ser tan robustos.

En la Figura 2.5 podemos ver un ejemplo de Random forest. Con el *dataset* que se le pasa generará, en este caso, tres árboles de decisión a partir de subconjuntos aleatorios del *dataset* original. Cuando se pretenda clasificar un nuevo dato cada árbol le asignará una clase a ese dato según lo que haya decidido. La clase que más se repite en este ejemplo es la 2, luego esa será la clase a la que pertenezca el dato.

2.3 IoT

Uno de los campos en los que son de gran importancia las técnicas de aperndizaje automático es en los proyectos de IoT (Internet of Things). El Internet de las cosas (o Internet of Things) es la interconexión de sensores y dispositivos, como por ejemplo pueden ser los electrodomésticos, termostátos, coches, ropa... Todos ellos se conectan a través de una red por medio de la cual pueden interaccionar entre ellos y compartir datos sin necesidad de que un humano intervenga, esta red puede ser por cable, WiFi, Bluetooth etc.

Los dispositivos conectados generan una gran cantidad de datos que llegan a una plataforma IoT que recolecta, procesa y analiza dichos datos. Utilizando Machine Learning se puede generar, por medio de todos estos datos recopilados, nuevos modelos de aprendizaje automático cuyas predicciones se pueden usar para poder actuar sobre el entorno. Las posibles aplicaciones que ofrece esto son prácticamente infinitas, algunos ejemplos pueden ser:

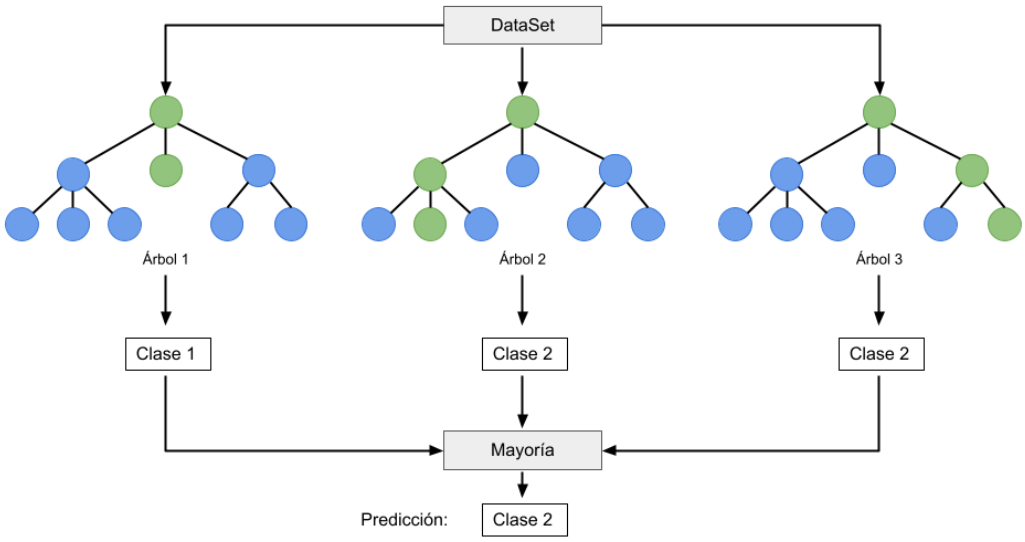

Figura 2.5: Ejemplo Random forest.

- Smart Homes. Permite que se puedan controlar varios de los dispositivos que hay comúnmente en los hogares haciendo que, por ejemplo, esté todo lo que necesite el usuario para cuando este llegue a casa.

- Telemedicina. Por medio de dispositivos que lleve puesto el paciente se puede tener un control sobre su estado de salud e incluso hacer diagnósticos anticipados.

- Vehículos autónomos. Los vehículos puedan estar conectados a la red de tráfico en tiempo real para poder evitar accidentes.

2.4 Dispositivo con capacidad de cómputo limitada: Raspberry Pi 4 Modelo B

Raspberry Pi[1] es un ordenador de bajo coste y tamaño reducido desarrollado por Raspberry Pi Foundation. Este dispositivo se puede emplear en multitud de aplicaciones pero su principal objetivo es hacer accesible la informática a todos los usuarios. A parte de poder realizar todas las tareas que se esperan de un ordenador, también puede interactuar con el entorno a través de sensores conectados a sus pines GPIO.

El sistema operativo que ofrece es Raspbian Pi OS, una versión adaptada de Debian. Sin embargo, permite poder utilizar otros sistemas.

[1] https://www.raspberrypi.org/

Desde su primer lanzamiento se han ido desarrollando y comercializado nuevos modelos. En este proyecto se utilizará la última versión de estos dispositivos denominado como Raspberry Pi 4 Modelo B con 4GB de RAM. Dicho modelo posee un total de 40 pines GPIO, 2 puertos micro HDMI y 4 puertos USB. Puede realizar conexiones inalámbricas ya que tiene Bluetooth 5.0 y Wi-Fi, aun que también puede utilizar Ethernet. Por último, la alimentación viene dada por un cable USB de tipo C con el que puede alcanzar un total de 1.2 A.

Dicho modelo posee un procesador ARM Cortex-A72 que soporta SIMD avanzado, un tipo de unidad de ejecución que se utiliza para ejecutar la misma instrucción a varios datos al mismo tiempo, consiguiendo paralelismo a nivel de datos. En total tiene cuatro cores físicos.

Otro detalle destacable de la Raspberry para el desarrollo de este trabajo es que posee una arquitectura SMP (Symmetric Multi-Processing), un tipo de arquitectura de computador en el que las unidades de procesamiento comparten una única memoria central, lo que permite que cualquier procesador trabaje en cualquier tarea sin importar su localización en memoria.

Actualmente es uno de los dispositivos más populares ya que brinda un sinfín de posibilidades a pesar de su pequeño tamaño. Específicamente dentro de el campo de aprendizaje automático e IoT, se podrían desarrollar multitud de aplicaciones gracias a la interacción que tiene con el entorno por medio de los sensores y actuadores que se pueden conectar. Por ello, en este trabajo de investigación se busca examinar el rendimiento de este pequeño ordenador y comprobar hasta donde se podría llegar con él, dentro del campo del aprendizaje automático.

2.5 Sensores

En este proyecto también se utilizaron sensores para tomar medidas del entorno y generar nuestro propio *dataset* con el que entrenar los modelos de aprendizaje automático. En concreto se utilizaron dos: una fotoresistencia (LDR) y un sensor BME280, a continuación se darán más detalles de ambos sensores.

2.5.1 Fotoresistencia

Esta resistencia también denominada como LDR por sus siglas en inglés Light Dependent Resistor, varia su resistencia dependiendo de la cantidad de luz que incida sobre su superficie. Funciona gracias al principio de la fotoconductividad que es un fenómeno óp-

tico en el que la conductividad del material aumenta cuando la luz es absorbida. Luego cuando la luz cae sobre la resistencia los electrones son excitados lo que provoca que empiece a fluir cada vez más corriente a través del dispositivo, por lo que la resistencia del dispositivo dismimnuye. Por lo tanto, cuando la LDR recibe luz, su resistencia será más baja y cuando esté a oscuras la resistencia será más alta. La resistencia del LDR está en torno a los 5kΩ

Para que la Raspberry pueda obtener el valor que devuelve este sensor es necesario utilizar un conversor. En este caso utilizaremos el HW-103, que al conectarlo a la Raspberry y a el sensor podremos conseguir una señal digital. Cuando a la Raspberry le llegue un cero significará que hay luz mientras que si lee un uno es que no hay luz.

2.5.2 BME280

El sensor BME280[2], desarrollado por Bosh, es un sensor creado especialmente para aplicaciones móviles y prendas de vestir donde el tamaño y el bajo consumo de energía son elementos claves. Integra en un mismo dispositivo tres sensores diferentes que proveen medidas de alta precisión. Los tres sensores que posee son: uno de temperatura, otro de presión atmosférica y por último, uno de humedad relativa.

El rango de la temperatura va desde los -40 ºC hasta los 85 ºC, con una precisión de ± 0.5 ºC. El sensor de presión tiene un rango de entre 300hPa y 1100hPa. Por útlimo el rango de la humedad relativa va desde el 0 hasta el 100%.

La comunicación con este sensor es muy sencilla ya que puede usar tanto I2C o SPI. Tiene una tensión de funcionamiento de 3.3V. Y a pesar de que integra tres sensores diferentes su tamaño es pequeño, concretamente de 19x18mm.

Este tipo de sensores se utilizan en aplicaciones como: monitorización del clima, automatización del hogar, sistemas de autopiloto para drones...Y se ajusta muy bien a las necesidades de este proyecto ya que tiene un consumo y tamaño muy pequeño, por lo que sería factible implementar una aplicación de IoT con él.

[2] https://www.bosch-sensortec.com/products/environmental-sensors/humidity-sensors-bme280/

2.6 Sistema Operativo: Ubuntu 21.10

El sistema operativo Ubuntu[3] es una distribución de código abierto basada en Debian, está compuesto de software normalmente distribuido bajo una licencia libre o de código abierto. La empresa responsable de su creación y mantenimiento es Canonical, una empresa de programación de ordenadores con base en Reino Unido fundada por el empresario Mark Shuttleworth.

La primera versión de Ubuntu fue la 4.10 lanzada el 20 de Octubre de 2004. A partir de la versión 13.4 las versiones estables sin soporte a largo plazo se liberan cada seis meses, y Canonical proporciona soporte técnico y actualizaciones de seguridad durante 9 meses. Las versiones LTS (Long Term Support) ofrecen un soporte técnico durante 5 años a partir de la fecha de lanzamiento.

La última versión fue lanzada el 14 de Octubre de 2021, que es la versión Ubuntu 21.10. Dicha versión utiliza el kernel 5.13, con cambios y mejoras para componentes que daban problemas. También tiene un nuevo instalador, escrito desde cero en Flutter, que facilita la instalación del sistema operativo. Una de las novedades más importantes es que actualiza su escritorio a GNOME 40.

2.6.1 Gestor de paquetes: Miniforge

Miniforge es un instalador mínimo de conda especifico de conda-forge, que permite instalar el manejador de paquetes conda con una serie de configuraciones predifinidas. Es muy similar a un instalador de Miniconda.

Miniconda es un sistema de gestión de paquetes y entornos virtuales. Mediante él se instala una pequeña versión de arranque de Anaconda que incluye solo conda, Python, los paquetes de los que dependen y otros pocos paquetes útiles como pueden ser pip, zlib...

Ofrece casi lo mismo que Anaconda pero es mucho más ligero, lo que lo hace idóneo para poder desarrollar el proyecto en el dispositivo utilizado. Además, facilita la replicación de un entorno concreto ya que permite tener un mayor control y orden sobre los paquetes que se instalan.

[3] https://ubuntu.com/

2.7 Lenguaje de programación: Python

Python[4] es un lenguaje de programación que nace a principios de los años 90 gracias al informático holandés Guido Van Rossum. Su objetivo era crear un lenguaje de programación que fuera fácil de aprender, escribir y entender.

Es un lenguaje de alto nivel (sencillo de leer y escribir debido a su similitud con el lenguaje humano), interactivo, interpretado y orientado a objetos. Al ser interpretado permite poder ejecutarlo sin necesidad de compilarlo previamente, reduciendo el tiempo entre la escritura y la ejecución del código. Utiliza módulos y paquetes, lo que fomenta la modularidad y la reutilización de código.

Además es multiplataforma y de código abierto, por lo tanto gratuito, lo que ha ayudado a que Python sea el lenguaje con mayor crecimiento y uno de los más utilizados en la actualidad.

Entre los campos en los que más se emplea este lenguaje se encuentra la inteligencia artificial, big data, machine learning y data science entre otros, ya que facilita la creación de códigos entendibles de rápido aprendizaje como los que son necesarios en este tipo de proyectos.

2.7.1 Entorno de desarrollo: Jupyter-notebook

Jupyter-notebook es una aplicación web lanzada en 2015, de código abierto desarrollada por la organización Proyecto Jupyter. Permite crear y compartir documentos computacionales que siguen un esquema versionado y una lista ordenada de celdas de entrada y salida.

Estas celdas pueden contener código, texto en formato Markdown, fórmulas matemáticas y ecuaciones, o también contenido multimedia. Cada celda se puede ejecutar para visualizar los datos y ver los resultados de salida. Los documentos creados en Jupyter pueden exportarse en otros formatos como PDF, Python o HTML. Además se puede utilizar tanto remotamente como en local.

Los dos componentes principales de Jupyter Notebook son un conjunto de núcleos (intérpretes) y el Dashboard, donde se visualizan las salidas de cada celda. Es compatible con 49 núcleos que permiten trabajar en diferentes lenguajes como R, Julia, C++ o Java. Sin embargo, el kernel que utiliza por defecto es IPython para programar con Python.

[4] https://www.python.org/

2.7.2 Librerías

Para el desarrollo del proyecto se utilizaron varias librerías que permitiesen la creación de códigos de aprendizaje automático. A continuación se hace mención de las principales y más importantes.

Scikit-learn

Scikit-Learn[5] es una librería, escrita principalmente en Python, que cuenta con algoritmos de clasificación, regresión, clustering y reducción de dimensionalidad. Fue inicialmente desarrollada por David Cournapeau como proyecto de Google Summer of code en 2007 y publicada tres años más tarde.

La gran variedad de algoritmos y utilidades de Scikit-learn la convierten en una herramienta muy eficaz para generar aplicaciones de aprendizaje automático.

Pandas

Pandas[6] es una librería de Python de código abierto especializada en el manejo y análisis de estructuras de datos. Es muy útil en el ámbito de Data Science y Machine Learning, ya que ofrece unas estructuras muy poderosas y flexibles que facilitan la manipulación y tratamiento de datos.

Tiene todas las funcionalidades necesarias para el análisis de datos como pueden ser: cargar, modelar, analizar, manipular y preparar los datos.

2.8 Dispositivo con capacidad de cómputo: HP Notebook 15s-fq1008ns

En este trabajo también se utilizará un dispositivo de mayor capacidades con el objetivo de poder observar la diferencia entre el comportamiento de este y el de la Raspberry. Teniendo siempre como referencia la actuación del dispositivo de mayor capacidad podremos obtener un conocimiento más profundo sobre como de bien o mal está actuando la Raspberry.

[5] https://scikit-learn.org/stable/
[6] https://pandas.pydata.org/

La máquina que se va a utilizar para esto es un portátil HP Notebook 15s-fq1008ns. Este portátil posee un procesador Intel® Core™ i5-1035G1 cuya frecuencia base es de 1GHz, y que al igual que la Raspberry soporta SIMD. Además tiene un total de cuatro cpus físicas y ocho lógicas, siendo de 3.6GHz la frecuencia máxima a la que puede ir un núcleo.

Por otra parte, tiene una memoria SDRAM DDR4-2666 de 8 GB, varios puertos externos como un puerto HDMI, 1 USB 3.1 tipo C, 2 USB 3.1 tipo A, un Smart Pin CA y un combo de auriculares/micrófono. Para la alimentación tiene un adaptador de CA de 45W, que carga una bateria de 41Wh.

Por defecto viene instalado el sistema operativo Windows 10, pero en este se instaló Ubuntu 20.04 para poder utilizarlo como portátil de trabajo durante la carrera.

2.9 GitHub

GitHub es una plataforma para crear proyectos de aplicaciones o de herramientas utilizando el sistema de control de versiones de Git. Además de alojar tu repositorio de código tambien te brinda herramientas que te permiten colaborar, planififcar y seguir proyectos de desarrolladores de todo el mundo.

Para el desarrollo de este trabajo se utilizó esta plataforma para tener un seguimiento de las diferentes versiones tanto del código como de la memoria. Además de contener archivos importantes como los csv de los *datasets* o información relevante de las instalaciones.

Además tambien se aprovechó una caracterísitca muy útil que tiene GitHub llamada *GitHub Pages*, que te permite mostrar el proyecto en vivo en una página web estática sin necesidad de pagar por hosting. Utiliza los archivos HTML, CSS y JavaScript que hay en el repositorio para crear el sitio web. De esta forma hice una página web para realizar un resumen en inglés de todo lo que se va a explicar a lo largo de esta memoria.

2.10 Redacción de la memoria: LaTeX/Overleaf

LaTeX es un sistema de composición tipográfica de alta calidad que incluye características especialmente diseñadas para la producción de documentación técnica y científica. Estas características, entre las que se encuentran la posibilidad de incluir expresiones matemáticas, fragmentos de código, tablas y referencias, junto con el hecho de que se distribuya como software libre, han hecho que LaTeX se convierta en el estándar de facto para la redacción y publicación de artículos académicos, tesis y todo tipo de documentos científico-técnicos.

Por su parte, Overleaf es un editor LaTeX colaborativo basado en la nube. Lanzado originalmente en 2012, fue creado por dos matemáticos que se inspiraron en su propia experiencia en el ámbito académico para crear una solución satisfactoria para la escritura científica colaborativa.

Además de por su perfil colaborativo, Overleaf destaca porque, pese a que en LaTeX el escritor utiliza texto plano en lugar de texto formateado (como ocurre en otros procesadores de texto como Microsoft Word, LibreOffice Writer y Apple Pages), éste puede visualizar en todo momento y paralelamente el texto formateado que resulta de la escritura del código fuente.

Capítulo 3

Diseño e implementación

En este capítulo se realiza una descipción detallada sobre la arquitecura del proyecto, las instalaciones necesarias para poder realizar el trabajo, una breve explicación sobre los modelos de aprendizaje que se implementaron así como los data sets utilizados.

3.1 Arquitectura general

Como ya se ha comentado anteriormente, las aplicaciones que utilizan Machine Learning e IoT son cada vez de mayor relevancia. Por ello se desea comprobar el rendimiento de la Raspberry Pi para realizar este tipo de tareas, puesto que su pequeño tamaño, bajo coste y capacidad de interacción con el entorno hacen que sea una buena opción para algoritmos con estos fines. Sin embargo, posee menos recursos que un ordenador común, lo que puede suponer un gran inconveniente para realizar estas tareas.

Para ello, y como se explicará en mayor detalle en los siguientes capítulos, pondremos a prueba a la máquina con diferentes test. En la Figura 3.1 se puede visualizar un esquema general de en qué consisten dichas pruebas. Como se puede ver en esta figura los experimentos se llevarán a cabo tanto en la Raspberry como en un portátil para poder comparar y comprender mejor los resultados obtenidos.

Como se explicó en la sección 2.1 hay varios tipos de aprendizaje automático y dentro de cada clase existen aún más métodos para resolver un mismo problema de formas diferentes. En este proyecto utilizaremos algoritmos de aprendizaje supervisado para realizar problemas de clasificación binaria, es decir, solo existen dos clases a las que pueden pertenecer los datos. Concretamente (y como se puede ver en la Figura 3.1) se utilizarán cuatro algoritmos de Machine Learning que son: Regresión logística, Máquinas de soporte vectorial, Gradient boosting y Random forest.

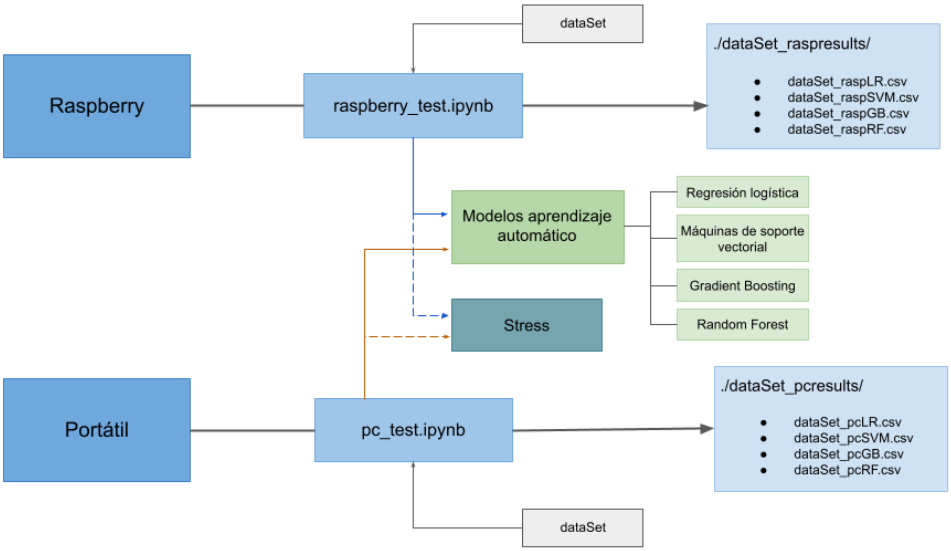

Figura 3.1: Estructura de los experimentos

Aprovechando los puertos GPIO de los que está provista la Raspberry, en este proyecto también se utilizan un par de sensores para poder generar un nuevo *dataset* con el que poder entrenar los modelos de aprendizaje automático. Estos dos sensores que se utlizan son los que se mencionaron en el apartado 2.5.

En la Figura 3.2 se puede observar la estructura de como se lleva a cabo la recolección de información de los sensores por parte de la Raspeberry. Con todo conectado correctamente, se generará un nuevo fichero csv con los datos medidos durante el tiempo que el usuario haya estado ejecutando el programa que se puede ver en esta figura. Este *dataset* podrá utilizarse para entrenar los modelos de aprendizaje. Además, mediante la información recogida se creará una figura que contiene estos datos en cada instante de tiempo.

3.2 Configuración del entorno

Para poder desarrollar correctamente este trabajo es necesario preparar adecuadamente el entorno, una vez acondicionado todo se dará pie al motivo principal de esta investigación.

El primer paso para esto fue montar adecuadamente la Raspberry Pi conforme las instrucciones de Okdo[1], empresa de la que procede el kit con el hardware utilizado en el proyecto.

Una vez está listo el hardware, hay que instalar el software necesario para la generación

[1] https://www.okdo.com/getstarted/

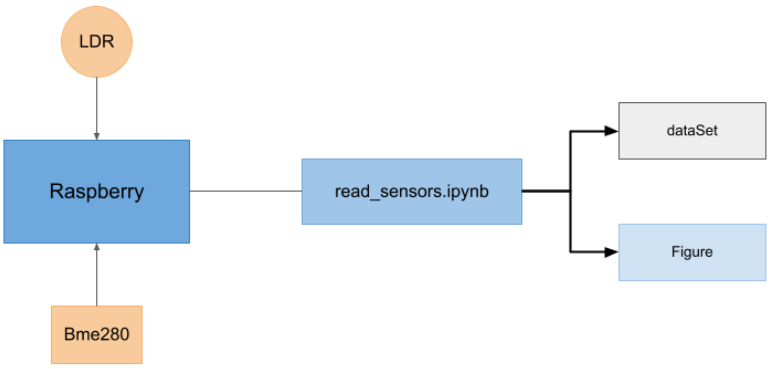

Figura 3.2: Estructura de creación de un nuevo de *dataset* con la información de los sensores

de modelos de Machine Learning. Comenzando por cambiar el sistema operativo, en vez de utilizar el que viene por defecto, Raspbian Pi OS, se instaló Ubuntu 21.10[2].

A continuación se instaló Miniforge, por medio del cual se crea un entorno virtual donde se instalaron todos los paquetes necesarios para el proyecto como son scikit-learn, pandas, jupyter-notebook... Estos paquetes permitirán desarrollar modelos de aprendizaje automático. Para mayor detalle de los paquetes instalados, en los ficheros *rasp_enviroment.yaml* y *pc_enviroment.yaml* se encuentran listados los paquetes que se instalaron en cada uno de los entornos creados en cada máquina.

Además se instaló stress, un comando de Linux que sirve para estresar durante un tiempo que se le indique el número de cpus que se le especifican. En este caso, dicho comando se utilizará para poder someter a la Raspberry a diferentes niveles de carga computacional y de este modo ver su capacidad para entrenar modelos de aprendizaje automático.

En el Anexo (Capítulo 6) de esta memoria se podrá encontrar en mayor detalle las dificultades y las soluciones a los problemas hallados a la hora de realizar todo lo comentado en este apartado.

3.2.1 Conexión de los sensores

Como se comentaba en el capítulo 2.5, en este proyecto se utilizaron en total dos sensores, un LDR y un sensor BME280. Para obtener las medidas sensadas hubo que conectar

[2] https://ubuntu.com/tutorials/how-to-install-ubuntu-desktop-on-raspberry-pi-4

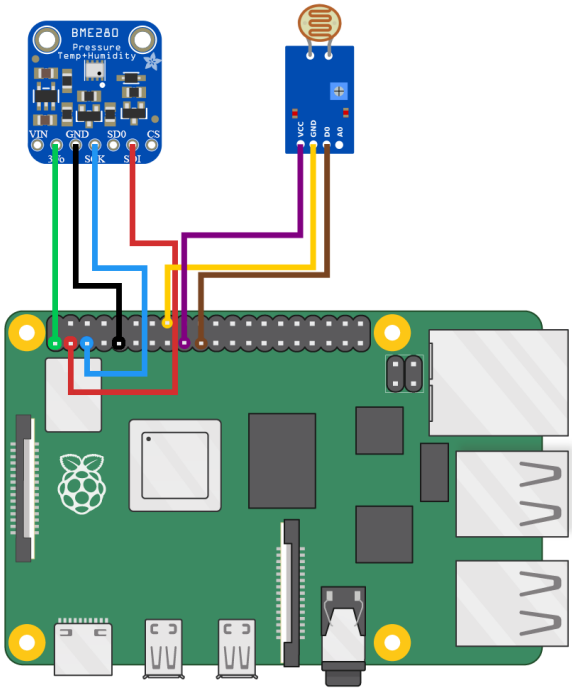

Figura 3.3: Conexiones de los sensores a la Raspberry.

ambos tal y como se puede ver en la Figura 3.3. En esta figura el sensor de la izquierda es el BME280, mientras que en la derecha se representa el conversor HW-103 conectado al LDR.

En el caso del sensor BME280 se utiliza comunicación por I2C para poder leer las medidas de temperatura, humedad y presión, por eso está conectado a los pines dos y tres de la Raspberry. Mientras que el LDR se conecta al pin 10 para saber si hay luz o no.

Para poder obtener los valores del LDR fue necesario instalar el paquete LGPIO, los detalles de su instalación se encuentran en el Anexo.

Para la comunicación I2C hubo que instalar el paquete `i2c-tools` en la Raspberry. Para poder utilizar este paquete es necesario tener permisos de root, sin embargo se dieron permisos a un usuario de forma que no tuviese que ser root para poder ejecutarlo[3]. Dentro del entorno de conda donde se desarrolla el proyecto se tuvieron que instalar además otros dos paquetes: `smbus2` y `RPi.bme280`. Finalizadas estas instalaciones podremos leer la información proporcionada por los sensores.

[3] https://lexruee.ch/setting-i2c-permissions-for-non-root-users.html

3.3 Generación de los modelos de Aprendizaje automático

Como se ha mencionado anteriormente, en este proyecto se utilizarán un total de cuatro algoritmos de aprendizaje automático. Los programas que se encargan de generar estos modelos se encuentran dentro de la carpeta `Modelos` de este proyecto. Hay un fichero por cada modelo que se desea entrenar y poner a prueba.

Todos estos algortimos se han implementado utilizando la libreria de scikit-learn que proporciona todas las funcionalidades necesarias para ello. A la hora de ejecutarlos será necesario pasarles un único argumento que es el *dataset* con el que se desea entrenar el modelo.

A continuación se nombrará y se explicará brevemente el contenido de estos ficheros.

- **lRegression.ipynb**

  Este fichero entrenará un modelo de Regresión logísitca para el *dataset* que se le pase como argumento.

  En scikit-learn, por medio de la función LogisticRegression[2] que pertenece a la librería *sklearn.linear_model*, se puede generar este modelo de aprendizaje automático para entrenarlo y posteriormente predecir con él.

  Entre los parámetros que se pueden asignar a esta función hay dos que son destacables. El primero de ellos es *max_iter*, máximo número de iteraciones que se permiten para encontrar la solución que converja, por defecto tiene un valor igual a 100. En el caso de `lRegression.ipynb` se igualó el valor *max_iter* a 400, pues es el valor mínimo necesario para que consiga encontrar una solución para algunos de los *datasets* que veremos más adelante.

  Por otra parte está el parámetro *n_jobs*, que permite declarar el número de cpus que se desean utilizar para paralelizar el proceso. Sin embargo, la asignación de este valor solo tiene efecto si se realiza una clasificación multiclase, de lo contrario utilizará un único core independientemente del valor asignado. Por lo que en este caso este parámetro no se utilizará.

- **svm.ipynb**

  En esta ocasión este fichero creará un modelo de Máqiunas de vector soporte. Para implementar este tipo de modelo de aprendizaje en scikit-learn se puede utilizar la función SVC [4]. En esta ocasión este método no soporta multiprcesamiento, por lo tanto la máquina que lo ejecute solo podrá usar un core tanto en el entrenamiento como en la predicción.

- **gBoosting.ipynb**

  Con este fichero se implementa un modelo de Gradient Boosting. En esta ocasión el método que crea este modelo es GradientBoostingClassifier[1], que se encuentra dentro de la librería *sklearn.esemble*. Al igual que para la función de máquinas de soporte vectorial, este modelo no soporta multiprocesamiento y por lo tanto solo se utilizará una cpu para entrenar y predecir con este modelo.

- **rForest_pc.ipynb // rForest_pc.ipynb**

  Por último tendremos dos ficheros que crean un modelo basado en Random Forest.

  En scikit-learn utilizando la función RandomForestClassifier[3], que también está dentro de la librería *sklearn.esemble*, se puede generar un modelo de aprendizaje utilizando esta técnica.

  Para este método existe un parámetro destacable denominado *n_jobs*, que permite declarar el número de cpus que se desean utilizar para paralelizar el proceso. Para que se note el efecto de este parámetro es necesario que el conjunto de datos con el que se desea entrenar sea grande. De lo contrario el coste de distribuir los recursos entre el número de cores indicados es más elevado que ejecutarlo todo en una única cpu, y por lo tanto los tiempos de ejecución serían más elevados a mayor número definido en este parámetro.

  Por defecto si no se declara este parámetro utilizará un único core. Además se pueden asignar valores negativos, en caso de igualar *n_jobs* a -1 se utilizarán todos los cores disponibles en ese momento. De esta misma forma se utiliza este parámetro en Regresión logística cuando se busca entrenar un problemas de clasificación multiclase.

  En esta ocasión se han creado dos ficheros para generar este modelo, puesto que dependiendo de la máquina en la que se esté ejecutando este código el valor del parámetro *n_jobs* variará.

Estos ficheros imprimirán por el terminal cuatro valores diferentes que se utilizan para valorar la eficiencia del entrenamiento. Dichos valores son: *accuracy*, *accuracy trainning*, *precision* y *recall*.

A continuación se detalla qué significan cada uno de esto valores. Para ello se ha definido la siguiente terminología: TP es el número de verdaderos positivos, TN verdaderos negativos, FP falsos positivos y FN falsos negativos. Siendo los valores positivos los que pertenecen a una clase y los negativos la clase contraria. En este caso diremos que los valores positivos serán los que pertenecen a la clase uno y los negativos los que pertenece a la clase cero.

Para esta explicación nos ayudaremos de la Figura 3.4. En la cual tenemos en total dieciocho datos, diez pertenecen a la clase de los positivos (zona de la izquierda de la figura)

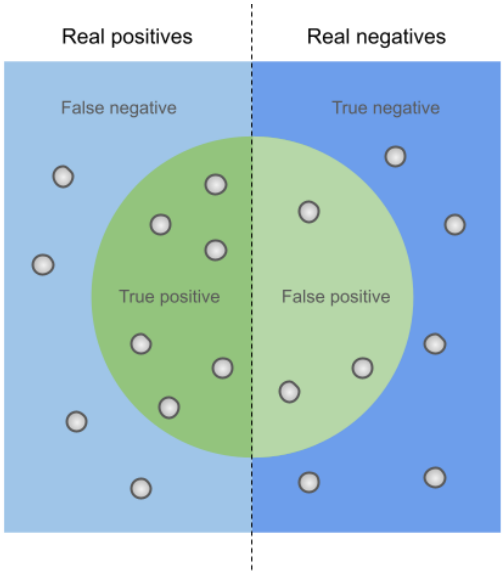

Figura 3.4: Ejemplo de estimación de un modelo Machine Learning.

y ocho que son de la clase negativa. Los círculos grises representan donde ha colocado el modelo de Machine learning cada dato. Como vemos ha asignado cuatro como negativos pero en realidad son positivos (zona azul claro) y tres como positivos pero pertenecen realmente a los negativos (zona verde claro).

El valor de *accuracy* representa el porcentaje total de aciertos. Este dato se calcula por medio de la siguiente ecuación:

$$\boxed{\text{Accuracy} = \frac{(\text{TP} + \text{TN})}{(\text{TP} + \text{TN} + \text{FP} + \text{FN})}} \tag{3.1}$$

El *trainning accuracy* es igual que el *Accuracy*, lo único que cambia son los datos con los que se hacen los cálculos.

Precision es el porcentaje que se obtiene al dividir el número de positivos que ha acertado (en este caso el valor positivo, o lo que es igual, la clase uno) entre todos los positivos que se han asignado (sean correctos o no).

Cuanto más alto sea este valor de *precision* menor error habrá cometido al asignar erróneamente la clase uno. Dicho valor viene dado por:

$$\boxed{\text{Precision} = \frac{\text{TP}}{(\text{TP} + \text{FP})}} \tag{3.2}$$

Utilizando el ejemplo de la Figura 3.4, será dividir seis (que es el número de datos

dentro de el color verde oscuro) entre la suma de esos más los que están en el color verde claro.

Por último, para obtener el valor de *recall* se dividen el número de verdaderos positivos entre la suma de los verdaderos positivos más los que son positivo y ha asignado como negativos. Es decir, se divide el número de veces que se ha asignado la clase uno entre la suma del número verdadero de datos que pertenecen a la clase uno más el número de veces que se ha estimado la clase uno pero en realidad los datos pertenecían a la clase cero. Siendo la fórmula por medio de la cual obtenemos este valor la siguiente:

$$\boxed{\text{Recall} = \frac{\text{TP}}{(\text{TP} + \text{FN})}} \tag{3.3}$$

Según la Figura 3.4 se dividirán el número de datos dentro de la zona verde oscuro entre la suma de esos más los que están en la zona azul claro.

Gracias a la librería *metrics* de scikit-learn se pueden obtener estos valores por medio de las funciones *accuracy_score*, *precision_score* y *recall_score*.

Pero para obtener estos valores es necesario ejecutar primero la función *predict* a la que se le pasa como argumento el set de datos del que se desea realizar una predicción y devolverá las salidas que a estimado el modelo para cada una de las entradas, esto se puede ver en el Código 3.1. Esta salida se les pasa como primer argumento a las funciones mecionadas antes y como segundo argumento el propio modelo generado (después de entrenarlo). Salvo en el caso de *trainning accuracy*, para el resto de valores habrá que utilizar las predicciones hechas sobre las entradas del datset que se utilizan para probar el modelo. Para el *accuracy trainning* necesitaremos las predicciones de los datos con los que ha entrenado el propio modelo.

Como se puede ver en el Código 3.1 para los valores de Precision y Recall se pasa un tercer parámetro, que es la clase respecto al que se desea calcular dichos valores. Ya que en vez de calcularlos para la clase positiva (como hemos visto anteriormente) se podrían calcular también respecto de la clase negativa.

3.4 DataSet: Room Occupacy

Para realizar la clasificación en un inicio se utilizó el data set de Room Occupancy detection data[5], obtenido de Kaggle, que contiene unas 20560 muestras. Tal y como su nombre indica, este *dataset* proporcionará información sobre si una habitación se encuentra en un determinado instante ocupada o no. Cada ejemplo tiene las medidas de temperatura, hu-

```
occup_pred_train= l_regr.predict(x_scaled)
occup_pred = l_regr.predict(x_test_scaled)
label= y[0]

accuracy = metrics.accuracy_score(y_train, occup_pred_train)*100
print("Trainning Accuracy: ","{:.1f}".format(accuracy),"%")

accuracy = metrics.accuracy_score(y_test, occup_pred)*100
print("Accuracy: ","{:.1f}".format(accuracy),"%")
precision = metrics.precision_score(y_test, occup_pred, pos_label=label)*100
print("Precision: ","{:.1f}".format(precision),"%")
recall = metrics.recall_score(y_test, occup_pred, pos_label=label)*100
print("Recall: ","{:.1f}".format(recall),"%")
```

Código 3.1: Obtención de los valores de Accuracy, Trainning accuracy, Precision y Recall.

medad, CO2 y luz de una habitación de oficina de unos quince metros cuadrados. La última columna de cada fila indica la clase a la que pertenece la muestra. En este caso, al ser una clasificación binaria esta última columna solo puede tener dos valores, cero o uno. Si para un ejemplo contiene un uno significa que para esos valores la habitación está ocupada. Si por el contrario hay un valor de cero la sala está vacia.

Para generar los modelos se dividió el set de datos en dos partes, una primera parte para entrenar (que contenía el 70% de ejemplos del set de datos original) y otra para comprobar la eficiencia del modelo a la hora de clasificar si la estancia está ocupada o no (es decir, un set de datos de prueba). En esta segunda parte se utilizó el 30% restante de muestras del set de datos original, que no se utilizaban en el entrenamiento y por lo tanto el modelo nunca los había visto, son totalmente nuevos para él.

Un aspecto importante a destacar de este set de datos es que hay una mayor cantidad de ejemplos de habitación no ocupada que de ocupada. En otros sets de datos esto podría representar un problema ya que puede dar lugar a que al realizar esta división de forma aleatoria, el conjunto de datos de entrenamiento apenas tenga ejemplos de una de las clases. Sin embargo, al tener una gran número de ejemplos en el que ambas clases tienen una gran cantidad de muestras, como es este caso, una división aleatoria no representa ningún inconveniente dado que hay una alta probabilidad de que el set de entrenamiento siempre tenga como mínimo la cantidad de muestras necesarias de ambas clases para entrenar correctamente. Aun así, la división se realizó de forma estratificada, para que hubiese la misma proporcion de datos de una clase u otra, que en el set de datos original. Y de esta forma nos aseguramos que a la hora de tanto entrenar, como de comprobar el modelo generado, se tengan ejemplos de ambas clases en la misma proporción que aparecen en el set de datos original.

Con esta división y preparación de los datos, los cuatro modelos se pudieron generar sin problemas utilizando los programas que se encuentran dentro de la carpeta de "`Modelos`".

3.4.1 Validación cruzada

Para comprobar mejor la calidad de los modelos se utilizó validación cruzada. Esto consiste en hacer que dentro del set de datos de entrenamiento (en este caso compuesto, una vez más, por el 70% de ejemplos del set de datos original) se hace una subdivisión en de otros cinco sets (en el caso de este proyecto) para observar la eficiencia que ha tenido el modelo al entrenar con cada uno de estos *subdatasets*.

Para ello, se utilizan dos métodos el primero de ellos es *KFold* que devuelve los índices por donde se van a dividir los datos para obtener varios sets (el número de sets dependerá del valor que se le asigne al parámetro *n_splits*). La segunda función que se utiliza es *cross_val_score*, a la que se le pasa el modelo que se quiere probar junto con los datos de entrenamiento y los índices obtenidos por *KFold*. Esta función nos devolverá el *Accuracy* que ha conseguido cada uno de estos *subdatasets*.

Una vez realizada la división se entrena con todo el *dataset* de entrenamiento y se vuelve a comprobar la eficiencia (de ese modelo entrenado) con el 30% de datos de la división original, los que nunca ha visto ni entrenado con ellos.

El objetivo de realizar una validación cruzada es garantizar que los resultados que obtengamos sean independientes de la partición entre datos de entrenamiento y datos de validación. Es decir, poder saber si la precisión que obtenemos al hacer la predicción es real o se ha sobreajustado a un *dataset* concreto.

Para utilizar validación cruzada se añadió a los ficheros de la carpeta `"Modelos"` los métodos mencionados anteriormente. Por lo que, además de los datos que ya mostraban estos ficheros por su salida, se añadirá un nuevo *print* para obtener la media total de las precisiones de cada una de las subdivisiones.

3.5 DataSet: KddCup99

Al realizar algunas pruebas para observar el comportamiento de la Raspberry ante diferentes situaciones de estrés (tal y como se explicará más adelante) los resultados obtenidos no encajaban del todo con lo esperado. Luego para tratar de comprender mejor lo que estaba pasando se decidió usar un *dataset* más grande que el anterior.

El *dataset* elegido fue KDD Cup 1999 Data[7], obtenido una vez más desde la página web de Kaggle. Este data set se utilizó en el tercer concurso internacional de herramientas de descubrimiento de conocimientos y minería de datos, el objetivo del concurso era generar un modelo predictivo que fuese capaz de distinguir entre conexiones *"malas"*, es decir, intrusiones o ataques, y conexiones *"buenas"*. Este set de datos contiene una gran variedad

de intrusiones en un entorno de red militar.

De este data set se utilizó tanto el fichero `kddcup.data.gz` como `kddcup.data_10_percent.gz`. Una vez descargados se descomprimen y se ejecuta un programa para tratar los datos que contienen y poder utilizarlos en la generación de los modelos de Machine Learning.

Al igual que en el *dataset* de Room Occupancy para la generación de los modelos se utilizará el 70% de los datos para entrenar y el 30% para validar.

3.5.1 Preprocesamiento de kddCup99

Para que este *dataset* se pueda utilizar para entrenar a modelos de aprendizaje automático de clasififcación binaria, es necesario realizar un preprocesameinto a los datos que contiene. De esto se encarga el programa *prepare_kddcup.ipynb*. Este fichero permite leer las líneas de este *dataset*, procesar y guardar los datos en un nuevo fichero para que puedan usarlos los modelos.

Para leer los datos se utiliza la función *read_csv* de pandas. Este programa no solo se utilizará para leer un *dataset* al completo, sino que también podremos leer otra porción diferente de líneas. Es por esto que para que el programa ejecute correctamente hay que pasarle un número como único argumento, cuyo valor puede estar entre cero o cien. Este valor representa el porcentaje de datos que se desea leer. Un ejemplo de ejecución de este fichero para que lea el 50% de los datos de kddCup99 es:

```
$ ipython prepare_kddcup.ipynb 50
```

En el caso de `kddcup.data` (que contiene más de un millón de líneas) interesa poder leer diferentes cantidades de datos del fichero puesto que tanto la Raspberry como el portátil no son capaces de leer este *dataset* al completo, los procesos morían antes de terminar. En el caso de la Raspberry el máximo de datos que es capaz de leer, sin que muera el proceso, es el cuarenta porciento del *dataset* total mientras que el portátil llegaba al cincuenta porciento del total.

Para leer otro porcentaje del *dataset* total, que no fuese únicamente el diez porciento, se utiliza una regla de tres. Sabiendo que el diez porciento del data set (el fichero `kddcup.data_10_percent`) contenía 494020 líneas, se podía obtener aproximadamente tanto a cuantas líneas equivaldría otro porcentaje como la cantidad total de líneas que debía tener el *dataset* completo.

La función *random.sample* devuelve una lista que contiene valores numéricos aleatorios, para realizar esta función necesita que se le pasen dos argumentos. El primero de ellos es

```
#Obtener número de lineas a leer
total= 4940200 # numero de lineas aproximado del fichero kddcup.data
per_lines= int((float(sys.argv[1])*total)/100)
#Leer dataSet
rm_lines= sorted(random.sample(range(total),total-per_lines))
dataset = pd.read_csv('/home/nuriadj/Documents/TFG/kdd_cup99/kddcup.data',
↪  skiprows=rm_lines)
print("Reading: %2.f" % round((len(dataset)*100)/total,2),"% of the csv")

#Conversión multiclase a binaria
dataset['normal.'] = dataset['normal.'].replace(['back.', 'buffer_overflow.',
↪  'ftp_write.', 'guess_passwd.', 'imap.', 'ipsweep.', 'land.', 'loadmodule.',
↪  'multihop.', 'neptune.', 'nmap.', 'perl.', 'phf.', 'pod.', 'portsweep.', 'rootkit.',
↪  'satan.', 'smurf.', 'spy.', 'teardrop.', 'warezclient.', 'warezmaster.'], 'attack')
```

Código 3.2: Lectura del *dataset* y conversion a clase binaria.

el valor máximo que puede aparecer en la lista, siendo el mínimo igual a cero. El segundo argumento es la longitud total de la lista, es decir, cuántos números aleatorios va a generar. Con este método se obtendrá una lista con valores aleatorios que se le asignará al parámetro *skiprows* de *read_csv*. Haciendo que todos esos datos, que representan números de líneas del fichero original, no se lean, consiguiendo hacer que se lea únicamente el porcentaje que ha pedido el usuario.

La implementación de esto se puede ver en el Código 3.2, donde *total* es el valor máximo de líneas que hay en el fichero *kddcup.data* (obtenido mediante la regla de tres comentada anteriormente), *per_lines* es el número de líneas que sí se desean leer (obtenido al aplicar una regla de tres al porcentaje que se ha pasado como argumento a *prepare_kddcup.ipynb*). Ambos valores los usa *random.sample* para obtener líneas aleatorias, que se ordenarán utilizando *sorted* para poder pasárselas a skiprows y por lo tanto no se leerán.

Otro aspecto relevante de este *dataset* es que contiene datos que pertenecen a varias clases. Por lo tanto, dado que estamos resolviendo problemas de clasificación binaria, habrá que hacer un tratamiento previo a la información que contiene el *dataset* para que se puedan agrupar los datos en dos clases en vez de en múltiples.
Este programa también hace esta conversión de multiclase a dos clases. Para ello, una vez leídos todos los datos se reemplazaban todas las clases (excepto la clase *normal.*) por la clase *attack*[8], de esta forma se consigue tener solo dos clases. El código que realiza esta tarea se puede ver en el Código 3.2.

Una vez realizada la transformación de multiclase a clase binaria, se hace un pequeño tratamiento a los datos para que cada modelo pueda entrenar adecuadamente y más fácilmente con ellos[6]. Para ello se elimina la columna 19 y 20, ya que estas contienen todo el rato el mismo valor, que es cero. A continuación se transforman los datos categóricos y por último se eliminan las filas duplicadas. Con este tratamiento los datos se guardan utilizan-

do la función *to_csv* que proporciona pandas y se guarda el nuevo set de datos tratados sin la cabecera ni el índice, de forma que desde el código de cada uno de los modelos los datos estén listos para poder ser utilizados.

En el repositorio de este proyecto dentro de la carpeta "kdd_cup99" se pueden encontrar varios csv con diferentes porcentajes de datos, listos para ser utilizados solo hay que descomprimirlos.

3.6 Dataset: Mi dataSet

A parte de los *datasets* anteriores se creó uno utilizando la información recopilada por lo sensores, mediante el programa `read_sensors.ipynb`. Este *dataset*, con medidas de temperatura, humedad, presión y luz, busca generar un modelo que nos permita saber si la habitación está ocupada o no en un determinado instante, al igual que sucedia con el *dataset* de Room Occupancy.

Para ello durante varias horas se estuvieron tomando las medidas en una habitación de forma controlada, es decir, sabiendo en cada instante si la habitación estaba ocupada o no. Durante diferentes periodos de tiempo la habitación estuvo ocupada o vacía. Cuando la habitación estaba vacía algunas veces se cerraba la persiana de la ventana de la habitación para que también tomase medidas bajo esta nueva condición.

Una vez finalizado el tiempo que se deseaba hacer la prueba se paraba el programa `read_sensors.ipynb`, se abría el nuevo csv generado y se añadia al csv una nueva columna que representa el estado de la habitación en cada instante, asignando un uno cuando estaba ocupada y un cero en el caso contrario.

3.6.1 read_sensors.ipynb

Como se ha comentado anteriormente este programa es el encargado de tomar las medidas de los sensores conectados a la Raspberry, guardarlas y generar con ellas un nuevo csv y una imagen que muestra de forma gráfica los valores obtenidos en cada instante de tiempo, para que visualmente sea más fácil interpretar lo que ha pasado en ese periodo de tiempo.

Para que pueda ejecutar correctamente es necesario que se le pase como argumento el nombre con el que se desea guardar el *dataset* que se va a generar. Este argumento también se utilizará para nombrar la gráfica que genera el programa con el siguiente formato: `Dara_graph_miDataSet`, siendo miDataSet el string que se ha pasado como argumento.

```
measures= [] #array que contiene los valores del sensor

try:
    while True:

        bme280_data = bme280.sample(bus, address)
        humidity= bme280_data.humidity
        pressure= bme280_data.pressure
        temp= bme280_data.temperature

        light= lgpio.gpio_read(h, LDR)
        light= 1 - light

        measures.append([humidity, pressure, temp, light])

        print(humidity, pressure, temp, light)
        time.sleep(1)
except KeyboardInterrupt:
   pass
```

Código 3.3: Bucle que guarda los datos detectados por los sensores.

Este programa necesita usar la comunicación I2C para obtener los valores del sensor BME280 pero también necesitará acceder directamente a los pines por medio de la librería LGPIO. Para poder utilizar LGPIO es necesario ejecutar el programa con permisos de root pero manteniendo el entorno conda actual (pues es donde se encuentran todas las librerías instalaldas), es por ello que para ejecutarlo habrá que utilizar la siguiente línea de comando:

```
$ sudo env "PATH=$PATH" ipython read_sensors.ipynb miDataSet
```

En la segunda celda de este programa se puede ver como se prepararán los puertos para poder leer los datos.

A continuación se quedará ejecutando indefinidamente un bucle while, que es el que se encarga de guardar los datos detectados por los sensores cada segundo. Dicho bucle se puede ver en el Código 3.3. Algo destacable es que para obtener el valor de la luz (proporcionada por el LDR) siempre se le resta a uno el valor obtenido por el LDR. Esto se hace así para que los datos se puedan interpretar mejor, puesto que cuando hay luz a la Raspberry le llega un cero. De esta forma cuando haya luz se guarda un uno en el fichero en vez de un cero. Este bucle estará continuamente mostrando por el terminal las medidas tomadas hasta que el usuario haga Ctr-C , de esta forma se indica que ya no se desean guardar más datos.

Una vez que este bucle ha finalizado, se ejecuta otro que iterará sobre los datos guardados en la lista *measures* para poder pintar esos valores en la gráfica que devuelve este programa. Dicho bucle se puede ver en el segmento del Código 3.4. Hay dos cosas destacables de este trozo de código. La primera es que el valor de las presiones se divide entre

```
hum= []
temp= []
pres= []
light= []
x= []

for i in range(len(measures)):
    hum.append(measures[i][0])
    pres.append(measures[i][1]/10)
    temp.append(measures[i][2])
    light.append((measures[i][3]*50))#multiplico por 50 para verlo mejor en el plot
    x.append(i)
```

Código 3.4: Bucle para generar la gráfica.

diez. Esto se debe a que las presiones que devuelve el sensor BME280 son valores muy superiores a los que puede conseguir cualquiera de los otros sensores, por lo que en vez de devolver la medida en hPa se divide entre diez para obtenerla en KPa y de esta forma poder apreciar mejor los cambios en los datos de todos los sensores.

El segundo aspecto destacable es sobre el valor de la luz, que se multiplica por cincuenta. Al igual que en el caso de las presiones esto se hace para que el cambio de estado entre oscuridad y luz sea apreciable en la gráfica, de lo contrario el salto entre ceros y unos apenas se podrían ver.

Mediante todo esto se obtuvo un nuevo *dataset* denominado como occup_detec_2h.csv. A este *dataset* se le añadió una nueva columna para indicar en qué instantes de tiempo la habitación estaba ocupada o vacía, y por ello se guardó con el siguiente nombre occup_detec_2h_Clean.csv, para de esta forma indicar que el *dataset* estaba listo para poder entrenar modelos con él.

La gráfica generada por este *dataset* se puede observar en la Figura 3.5. Donde el eje x representa los minutos que han pasado desde que comenzó a ejecutarse el código.

Para la generación de este *dataset* primero estuvo la habitación ocupada durante media hora. Pasada esa media hora la habitación estuvo vacía durante 40 minutos. Después volvió a estar ocupada durante una hora entera, se vació una última vez durante 25 minutos para estar ocupada los 10 últimos minutos antes de finalizar la recogida de datos.

Sabiendo en qué momentos ha estado ocupada o vacía y observando la Figura 3.5 se puede ver que los valores de los datos varían entre el estado ocupado y vacío (a excepción de la presión que no cambia), siendo la humedad la variable donde más se diferencian los valores entre ambos estados. La temperatura varia algo menos, pero igualmente se puede ver un pequeño descenso cuando la habitación está vacía.

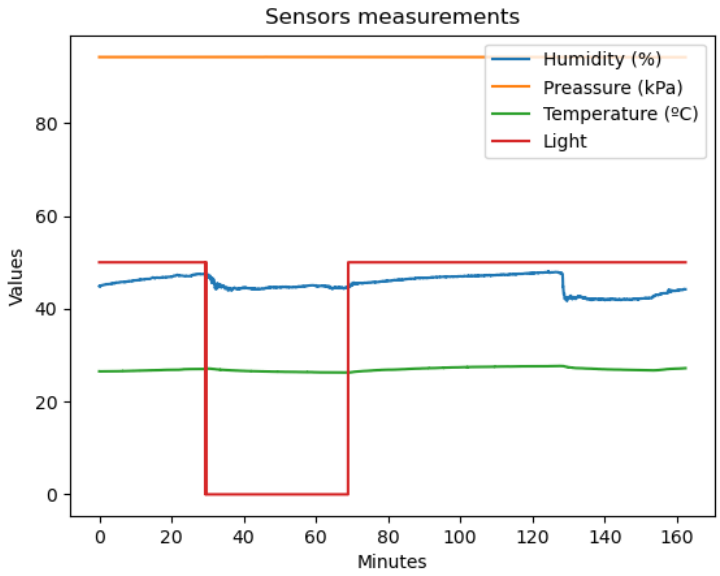

Figura 3.5: Ejemplo gráfica de las medidas de los sensores.

Capítulo 4

Experimentos y validación

En este capítulo comprobaremos la eficiencia de la Raspberry a la hora de generar los modelos de aprendizaje automático para que se ajusten los *datasets* mencionados en el capítulo anterior. Las pruebas consistirán en someter a la máquina a diferentes niveles de cargas computacionales para observar como lidia la Raspberry con la generación del modelo a la vez que con estas cargas. Además nos ayudaremos de un portátil para poder comparar los resultados de ambos dispositivos.

4.1 Estructura de los experimientos

Para realizar los experimentos se definieron cuatro niveles de saturación, dependiendo del dispositivo en el que se ejecuten, las pruebas variarán un poco.

El primer nivel es el "Idle" en el que no se somete a la máquina a ningún tipo de carga extra, este es igual en ambos dispositivos. Salvo este primer nivel, el resto sí variarán ligeramente entre ambas máquinas.

En el caso de la Raspberry el nivel bajo consiste en estresar una única cpu, el nivel medio dos cpus y por último en el nivel alto se estrasaban todas las cpus de la Raspberry, es decir, cuatro cpus.

Mientras que en el portátil, dado que este tiene un total de ocho cpus físicas, la cantidad de carga computacional a la que se somete en los niveles bajo, medio y alto, cambia para poner más o menos en la misma situación a ambas máquinas. Por lo tanto en este caso el nivel bajo de carga será estresar dos de sus cpus, el nivel medio cuatro y el alto los ocho cores de los que dispone la máquina. De esta forma se podrán comparar los tiempos de ejecución en un dispositivo de mayor capacidad y sabremos cuanto dista el comportamiento de la

Raspberry del de una máquina con mayor potencia.

Cada uno de los códigos encargados de generar uno de los modelos se ejecutan con estos cuatro niveles de carga. Para generar esta carga se usa el comando *stress* que permite estresar, durante el tiempo que se le indique, el número de cpus que se comanden.

Una de las medidas que se utilizan para comparar los diferentes niveles de estrés es el tiempo que tarda cada uno de los modelos en terminar de ejecutar. Para obtener dicho tiempo se utiliza el comando *time* de Linux, con el que se ejecutaba cada modelo de la siguiente forma:

```
$ time ipython modelo.ipynb
```

El comando *time* devuelve tres valores. Uno de ellos es el tiempo de usuario, que es la cantidad de tiempo que se ha gastado el proceso en modo usuario. El Sys time es el tiempo de cpu invertido en el kernel dentro del proceso. Con estos dos tiempos se puede obtener el Cpu time, el tiempo total que ha utilizado la cpu para completar la ejecución del proceso. Por otra parte time también proporciona el real time, que es el tiempo que ha tardado en ejecutar el proceso como si lo hubiesemos cronometrado con un reloj, este tiempo es el mismo que el Wall time. Es necesario entender esto para poder comprender los valores que nos devolverán los experimentos.

4.1.1 Ejecución de los experimentos

Para realizar todas las pruebas, tal y como se ha explicado, se creó un programa para cada una de las máquinas de forma que fuese más sencillo obtener los resultados de los experimentos. En el caso de la Raspberry este programa se denominó `raspberry_test.ipynb`. En el caso del portátil se nombró `pc_test.ipynb`. Dado que las puebas en ambos dispositivos son prácticamente las mismas, ambos programas ejecutan prácticamente lo mismo con pequeñas diferencias.

Dichos programas necesitan como único argumento el dataset con el que se desea hacer el experimento para que puedan realizar su función correctamente. Hay que tener en cuenta que este dataset debe de estar limpio, es decir, que se le haya hecho un preprocesamiento para borrar características no deseadas o redundantes. Tampoco debe de tener cabecera. Si no se hace este preprocesamiento podrá dar error la ejecución.

A continuación se desarrollará en mayor profundidad qué es lo que hace cada programa para realizar las pruebas.

raspberry_test.ipynb

La misión de este código es ejecutar cada uno de los modelos de aprendizaje automático (que se utilizan en este proyecto), con los diferentes niveles de saturación comentados anteriormente, en la Raspberry. Para realizar esto se usan dos bucles anidados, el primero de ellos itera sobre los modelos que se desean poner a prueba. El segundo recorre una lista que contiene el número de cores que se desean estresar con cada uno de los modelos, ambos bucles los podemos ver en el Código 4.5. Si el valor sobre el que se está iterando de la lista es mayor que cero significa que se desea estresar cierto número de cpus, por lo tanto se llama a la función *start_stress* a la que se le pasa como argumento el número de procesadores a estresar y durante cuanto tiempo, con estos datos se encargará de crear un nuevo proceso para ejecutar el comando *stress*.

Por otra parte, independientemente del nivel de saturación de la prueba, siempre se llamará a la función *model* que creará un nuevo proceso para ejecutar, utilizando el comando *time* de Linux, uno de los ficheros que se encuentran dentro de la carpeta "`Modelos`". Cada uno de los ficheros que se encuentran dentro de dicha carpeta generará uno de los modelos de aprendizaje automático que se han explicado en este proyecto.

Cuando se completa la generación del modelo para uno de los niveles, los resultados de *Cpu time*, *Wall time*, *Mean Folds Accuracy*, *Accuracy trainning*, *Accuracy*, *Presisión* y *Recall* se guardan cada uno en su respectiva lista gracias a una función del programa llamada *get_data*. Una vez que uno de los modelos ha completado todas las pruebas se genera un nuevo fichero csv, mediante la función *add_data*, con los resultados obtenidos en cada uno de los casos. Este fichero tendrá por nombre el siguiente formato: nombreDataSet_raspMD.csv, donde MD son las siglas del modelo que se ha ejecutado. Por ejemplo, en el caso de Regresión logísitca el nombre será nombreDataSet_raspRL.csv, o en el caso de Máquinas de soporte vectorial será nombreDataSet_raspSVM.csv. En total el programa creará cuatro csv diferentes, un por cada modelo. Estos ficheros se guardarán en una nueva carpeta creada con el siguiente formato: nombreDataSet_raspresults, esto se puede observar como se hace en el Código 4.5. Todos los ficheros se guardarán dentro de esta carpeta, para que los resultados de un mismo dataSet estén todos juntos.

Los nuevos procesos, comentados anteriormente, se crean utilizando la función *Popen* de la librería *subprocess*. Nótese que en la función *model* (a diferencia de *start_stress*) el método *Popen* tiene añadido al final un *.communicate()* de esta forma se parará la ejecución principal hasta que el proceso de *Popen* termine. Si una vez terminado dicho proceso *stress* sigue ejecutando se matará el proceso para poder comandar otro *stress* sin tener que esperar a que se cumpla el tiempo de ejecución que le pasamos como argumento, esto se puede ver en el Código 4.5.

```
t= 2000
list_cpus= [0,1,2,4]
dataSet= sys.argv[1]
name_dataSet= dataSet.split("/")[-1].split(".csv")[0]
folder_name= name_dataSet+"_raspresults"

#Crear la carpeta
if(os.path.exists("./"+folder_name)== False):
    os.mkdir(folder_name)
name_dataSet= "./"+folder_name+"/"+name_dataSet

lst_header= []
lst_Wt= []
lst_Ct= []
lst_Af= []
lst_TAc= []
lst_Ac= []
lst_Pr= []
lst_Re= []

for i in range(4):
    for j in list_cpus:
        num_cores= j

        if(j > 0):
            pid_stress= start_stress(j, t)
        else:
            print("\nCpu Idle")

        model_num= i
        model(model_num)

        if j > 0:
            os.killpg(os.getpgid(pid_stress.pid), signal.SIGKILL)

        print("Sleeping for 5 sec...\n")
        time.sleep(5)
```

Código 4.5: Función principal raspberry_test.ipynb

pc_test.ipynb

Este código es prácticamente idéntico al de `rasp_test.pynb` salvo pequeñas diferencias. Una de ellas es que en este caso el fichero que generará tras la ejecución de cada modelo tendrá el siguiente formato: nombreDataSet_pcMD.csv, siendo MD una vez más las siglas del modelo de aprendizaje automático. Al igual que en el caso anterior, se creará una nueva carpeta para guardar todos estos ficheros con el formato: nombreDataSet_pcresults.

La función principal también utilizará dos bucles anidados para iterar sobre los modelos de aprendizaje con los que se desean experimentar y el número de cores que se van a estresar en cada prueba. En este caso la lista denominada como *num_cores* variará respecto a la que se declaró en `raspberry_test.ipynb` ya que, como se ha comentado antes, en esta ocasión tendremos que saturar otras cantidades de cores. Este programa creará (si es necesario) dos procesos, uno ejecutará stress (cuando se desee estresar alguna cpu) y el otro uno de los modelos que se encuentran dentro de la carpeta `"Modelos"`. Todos los ficheros que generan los modelos de aprendizaje son los mismos que en la Raspberry salvo el de Random Forest que tiene un fichero específico para el portátil puesto que en este caso, el parámetro *n_jobs* tendrá que ser igual a ocho, para que el modelo pueda disponer de todos los cores que tiene el portátil.

4.2 Experimetos con datos sintéticos

Como se comentó en el Capitulo 3 en este proyecto se han utilizado varios *datasets* para realizar las pruebas. Dos de estos son ficheros con datos sintéticos, es decir, que no se han generado a través de las medidas de los sensores utilizados en este trabajo. A continuación se explicará en mayor detalle los resultados obtenidos por ambos en cada una de las máquinas.

4.2.1 Raspberry

Resultados Room Occupancy

Los tiempos de cpu para la generación de estos modelos se pueden ver en la Tabla 4.1

Los tiempos obtenidos para este *dataset* apenas varian entre los diferentes niveles de saturación (salvo el caso de Random Forest), siendo el modelo de Regresión logítica el que menos tarda en entrenar en todos las pruebas.

Modelo	Idle		1 CPU		2 CPUs estresadas		4 CPUs estresadas	
	Cpu time	Wall time	Cpu time	Wall time	Cpu time	Wall time	Cpu time	Wall time
Regresión logísitca	17.79 seg	18.83 seg	26.8 seg	15.2 seg	24.19 seg	16.62 seg	17.69 seg	13.85 seg
SVM	16.52 seg	16.37 seg	16.59 seg	16.55 seg	16.63 seg	16.62 seg	16.95 seg	17.01 seg
Gradient boosting	28.61 seg	28.74 seg	28.4 seg	28.38 seg	28.3 seg	28.31 seg	28.51 seg	28.66 seg
Random forest	59.15 seg	24.59 seg	52.44 seg	26.15 seg	48.92 seg	30.67 seg	49.29 seg	34.34 seg

Tabla 4.1: Resultados de los tiempos de ejecución para el Occupancy dataSet

Modelo	Accuracy cv (%)	Accuracy trainning (%)	Acuracy (%)			
			Idle	1 cpu	2 cpu	4 cpu
Regresión Logísitca	98.9	98.9	98.8	99.0	99.0	98.9
SVM	98.9	99.0	98.9	99.0	98.8	99.2
Gradient Boosting	99.0	99.4	98.9	99.1	98.7	98.9
Random Forest	99.2	100.0	99.3	99.3	99.32	99.3

Tabla 4.2: Resultados de la precisión de los modelos para el Occupancy dataSet en Raspberry

Para los modelos de Regresión logistica, Máquinas de soporte vectorial y Gradient boosting, es normal que no varien apenas sus valores dado que tal y como se explicó en el Capítulo 3 estos tres modelos son *monocores*, es decir, solo pueden ejecutar en una cpu.

Sin embargo el modelo de *Random forest* no terminaba de encajar del todo con lo espreado, es por ello que se decidió probar a ejecutar los modelos con un *dataset* con mayor cantidad de datos. Con un *dataset* de mayor tamaño se podrán observar mejor las razones por las que se devuelven estos tiempos.

Por otra parte, podemos ver como los tiempos de *Wall time* se mantienen iguales a pesar de someter a más carga a la Raspberry, a excepción de el modelo de *Random forest* que es el único que va incrementando a medida que aumenta el nivel de la prueba.

En cuanto a la precisión (*Accuracy*) en cada uno de los modelos se obtienen los resultados de la Tabla 4.2

Como se puede ver en dicha tabla, en este caso la diferencia entre la precisión de los datos de entrenamiento y otros datos que no ha visto hasta entonces es muy similar. Luego los modelos que se están generando son bastente fiables puesto que todos están por encima del 90% de precisión. Y sabemos que no se están sobreajustando los modelos puesto que

Modelo	Idle		1 CPU estresada		2 CPUs estresadas		4 CPUs estresadas	
	Cpu time	Wall time	Cpu time	Wall time	Cpu time	Wall time	Cpu time	Wall time
Regresión logísitca	447.38 seg	142.72 seg	506.49 seg	191.09 seg	422.57 seg	223.33 seg	387.74 seg	211.3 seg
SVM	528.55 seg	529.74 seg	517.1 seg	518.07 seg	494.24 seg	494.31 seg	495.47 seg	497.24 seg
Gradient boosting	853.97 seg	855.97 seg	876.88 seg	879.19 seg	853.42 seg	853.49 seg	858.95 seg	860.68 seg
Random forest	1048.87 seg	296.41 seg	829.86 seg	306.78 seg	669.36 seg	348.86 seg	706.96 seg	381.14 seg

Tabla 4.3: Resultados de los tiempos de ejecución para el Kdd_cup99 dataSet en la Raspberry

en la columna `Accuracy cv`, que es la precisión que se obtiene de la validación cruzada, en todos los casos también es superior al 90%. Este porcentaje se obtiene de la media de la fila `Mean Folds Accuracy` de los csv con los datos generados en cada modelo.

Resultados KddCup99

Los resultados obtenidos para el veinte por ciento del *dataset* de KddCup99 se encuentran en la Tabla 4.3. En esta ocasión los tiempos de ejecución aumentarán respecto al *dataset* de Room Occupancy puesto que este *dataset* tiene muchas más líneas con las que entrenar (hay más de cuatro millones de líneas de diferencia entre ambos *datasets*).

Como se puede observar en la tabla 4.3, los tiempos de ejecución varian entre los diferentes modelos, siendo el de Regresión logística el modelo más rápido de todos, tal y como ocurría con el *dataset* de Room Occupancy.

Un vez más, los tiempos de cpu de Regresión logísitca, Máquinas de soporte vectorial (SVM) y Gradient boosting apenas varían entre los diferentes niveles de estrés, esto se debe a las mismas razónes comentadas antes. En el caso de Máquinas de soporte vectorial y Gradient boosting, *scikit-learn* no admite *multi-threading*, por lo que a pesar de estresar una, dos o cuatro cpus siempre devolverán el mismo tiempo puesto que estos modelos solo pueden utilizar un único core para ejecutar.
Regresión logística podría utilizar varios cores si se le indica mediante el parámetro *n_jobs*. Pero dado que estamos en un problema de clasificación binaria este valor no tendrá ningún efecto al aumentar o disminuir el número de cpus estresadas como vimos en el apartado 3.3.

En cuanto al modelo de *Random forest*, para poder entender los tiempos que devuelve,

es necesario explicar primero como se comporta la Raspberry cuando varios procesos le piden más recursos de los que tiene y como afecta el parámetro *n_jobs* a los tiempos de este modelo.

Comencemos explicando como afecta el parámetro *n_jobs* a los tiempos de ejecución. Como se ha visto anteriormente, *Random forest* admite el parámetro *n_jobs*, que en este caso (a diferencia de Regresión logística) sí que afecta a como ejecuta el modelo independientemente de si es un problema de clasificación binaria o no. Por lo tanto para poder optimizar el proceso habrá que hacer que el parámetro *n_jobs* sea igual al número de cores del dispositivo, que en este caso serán cuatro.

Con este parámetro ajustado hay que saber que el tiempo de cpu que obtenemos del comando *time* es la suma de los tiempos que ha necesitado cada uno de los cores utilizados por el proceso para la ejecución. Esta es la razón por la que podemos ver como el tiempo en *Idle* es bastante mayor que el resto de casos, puesto que es en esta ocasión donde el programa está pudiendo hacer un uso completo de los cuatro cores, ya que no hay ningún otro proceso que necesite usarlos. Por lo tanto el *Cpu time* que tenemos en este nivel es en realidad la suma de los tiempos de ejecución de los cuatro procesadores. El tiempo que ha necesitado cada uno de los cores sería dividir los 1048.87 segundos entre 4 con lo que obtendríamos 262 segundos por core. Sabiendo esto, queda explicado por qué el nivel de *Idle* devuelve un valor tan alto y como esto, que a *priori* puede parecer un poco contradictorio, se ajusta a lo esperado.

Para el resto de casos, dado que ya intervienen otros procesos, habrá que conocer como se comporta la Raspberry ante una mayor demanda de recursos. Lo primero que hay que saber es que el comando *stress* estresará el número de cpus que se le pasan como argumento siempre y cuando pueda. En el momento en el que tenga que compartir cpu con otros procesos, la Raspberry repartirá los recursos haciendo que *stress* no utilice todas las cpus que se le ha comandado estresar para que el otro proceso también pueda ejecutar. Por ejemplo, cuando se realiza la prueba de nivel alto de carga al modelo de Máquinas de soporte vectorial, si se comprueban los valores de cpu que utiliza cada uno de los procesos (por medio del comando *htop* de Linux), a pesar de que se puede ver que *stress* tiene cuatro procesos, la suma de lo que utiliza la cpu cada uno de ellos será igual a un total de tres cores más o menos, es decir utilizan un 300% de los recursos. Mientras que el proceso del modelo utiliza una cpu, un 100%. Siendo el valor total de los recursos de los que dispone la Raspberry igual a 400%, un 100% para cada uno de los cores de la Raspberry.

Otro ejemplo de esto puede ser el caso en el que se ejecuta el modelo de *Random forest*, teniendo el parámetro de *n_jobs* igual a cuatro para utilizar todos los cores disponibles, a la vez que *stress* para estresar las cuatro cpus. Si observamos mediante el comando *htop* los procesos, se podrá ver como se distribuyen los cores para que puedan ejecutar a la vez tanto los procesos de *stress* como los del modelo de aprendizaje. Por lo tanto *stress* utilizará en total dos cpus y *Random forest* hará lo mismo, a pesar de que a ambos se les indicó que

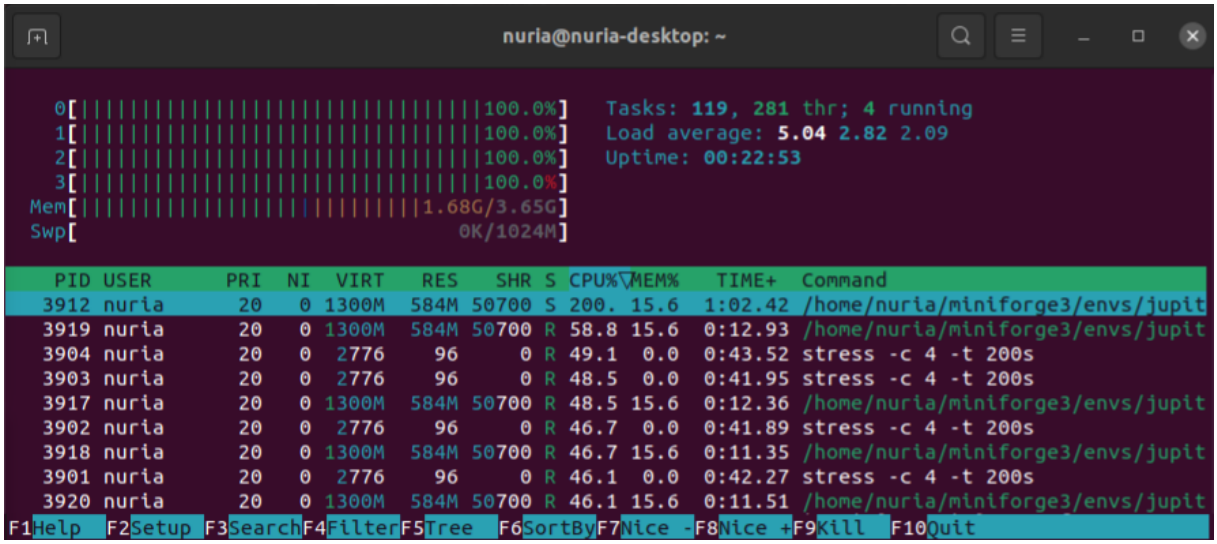

Figura 4.1: Ejecución Random forest con cuatro cpus estresadas.

utilizasen todas las cpus de la Raspberry. Este ejemplo se puede ver en la Figura 4.1, siendo la cpu total que utiliza *Random forest* la que se indica en el primer proceso que muestra *htop* en la imagen. Al sumar el resto de procesos de RF su valor de cpu es igual al del primer proceso.

En la Figura 4.1 vemos que el primer proceso de RF está consumiendo el 200% de cpu. Mientras que si sumamos los porcentajes de los cuatro procesos de *stress* obtendremos que se están consumiento en total un 190%, prácticamente un 200%. Por lo que, a pesar de lo que se ha comandado, la Raspberry dividirá los recursos para que ambos procesos puedan usar lo máximo posible.

Sabiendo esto, los niveles de prueba medio y alto para el modelo de *Random forest* obtienen prácticamente los mismos tiempos puesto que al tratar de ejecutar a al vez tanto el proceso de *stress* como el del modelo, se están pidiendo usar más cores de los que hay. En consecuencia la Raspberry tendrá que dividir los recursos de los que dispone equitativamente entre ambos procesos. Por lo que en el fondo *stress* estará estresando el mismo número de cpus tanto en el nivel medio como en el alto y al proceso del modelo le sucederá lo mismo, utilizará el mismo número de procesadores tanto en un caso como en el otro. En el nivel intemedio, donde se estresan dos cores, el modelo no podrá utilizar las cuatro cpus que se le indican con *n_jobs* puesto que estaría superando el número de cores que posee la Raspberry. De modo que la Raspberry dividirá los recursos haciendo que *stress* pueda ejecutar en dos cores, y por lo tanto el modelo de aprendizaje dispondrá únicamente de las otras dos cpus para su ejecución. Mientras que en el nivel más alto de carga la Raspberry dividirá las cuatro cpus para darle la mitad de ellas a la ejecución de *stress* y la otra mitad al modelo, haciendo que ambas utilicen el máximo de procesadores posibles. Esta es la explicación para los tiempos del nivel medio y alto.

En el nivel bajo encontraremos que el modelo solo puede estar usando tres cores para dejarle el cuarto a *stress*. Como hemos visto el tiempo de cpu es el tiempo total que han usado los tres procesadores, luego por cada core el proceso ha tardado unos 276 seg (el resultado de dividir 829.86 seg entre los tres cores). Por lo tanto el tiempo aumenta un poco del estado respecto al nivel *Idle*.

Algo destacable de los resultados obtenidos en este *dataset* (a diferencia del de Room Occuapncy) es que salvo en el nivel de *Idle*, el modelo de *Gradient boosting* llega a ser más lento que el modelo de *Random forest*, modelo que en el anteior dataser era con diferencia el que tardaba más tiempo de todos.

En cuanto al *Wall time* obtenido vemos que por un lado *Random forest* aumenta su tiempo a medida que aumenta la carga computacional, mientras que tanto el modelo de SVM como de GB se mantienen más o menos constantes. Y Regresión logísitca es el que más varía entre las diferentes pruebas.

Otro aspecto de las ejecuciones en la Raspberry que será relevante más adelante, es la frecuencia a la que ejecutan los cores. Utilizando el comando *lscpu* de Linux, se puede obtener información relevante sobre los cores como por ejemplo la frecuencia máxima y mínima a la que pueden ejecutar, que en el caso de la Raspberry la mínima será igual a 600MHz y la máxima 1500MHz.

Para saber la frecuencia de cada uno de los cores en tiempo real se puede ejecutar el comando:

```
$ watch -n.1 "
sudo cat /sys/devices/system/cpu/cpu0/cpufreq/cpuinfo_cur_freq &&
sudo cat /sys/devices/system/cpu/cpu1/cpufreq/cpuinfo_cur_freq &&
sudo cat /sys/devices/system/cpu/cpu2/cpufreq/cpuinfo_cur_freq &&
sudo cat /sys/devices/system/cpu/cpu3/cpufreq/cpuinfo_cur_freq"
```

Mostrará las frecuencias de ejecución de los cores cada 0.1 segundos. Si utilizamos dicho comando para ver las frecuencias cuando la Raspberry está tanto *Idle* como estresada podremos ver que todas las cpus están sobre los 1500MHz. Por lo tanto todos los cores siempre están ejecutando a la máxima velocidad que pueden.

Por otra parte los resultados de precisión para este *dataset* se pueden observar en la Tabla 4.4. Al igual que en el caso del Room Occupancy DataSet se obtienen muy buenos resultados, estando prácicamente todos cerca del 100% de *Accurancy*, y sabiendo que no se están sobreajustando puesto que la precisión de la validación cruzada también es prácticamente el 100% en todos los casos.

Por último, añadir que con este *dataset* se hicieron varias pruebas con distintas canti-

Modelo	Accuracy cv (%)	Accuracy trainning (%)	Acuracy (%)			
			Idle	1 cpu	2 cpu	4 cpu
Regresión Logísitca	99.4	99.4	99.4	99.4	99.4	99.4
SVM	99.9	99.9	99.8	99.9	98.8	99.8
Gradient Boosting	99.9	100.0	96.4	98.8	99.4	73.0
Random Forest	100.0	100.0	97.5	97.9	96.9	98.6

Tabla 4.4: Resultados de la precisión de los modelos para el Kdd_cup99 dataSet en Raspberry

Modelo	**Idle**		**2 CPU estresada**		**4 CPUs estresadas**		**8 CPUs estresadas**	
	Cpu time	Wall time	Cpu time	Wall time	Cpu time	Wall time	Cpu time	Wall time
Regresión logísitca	2.31 seg	1.9 seg	2.56 seg	1.615 seg	2.63 seg	1.71 seg	5.53 seg	3.5 seg
SVM	3.29 seg	3.3 seg	3.65 seg	3.65 seg	6.16 seg	6.17 seg	6.24 seg	7.52 seg
Gradient boosting	7.5 seg	7.54 seg	8.13 seg	8.14 seg	10.44 seg	10.45 seg	17.16 seg	21.47 seg
Random forest	18.22 seg	3.97 seg	16.43 seg	4.55 seg	16.61 seg	6.16 seg	21.64 seg	10.83 seg

Tabla 4.5: Resultados de los tiempos de ejecución para el Occupancy dataSet en el portátil

dades (porcentajes) de datos procedentes del *dataset* total, obteniendo en todos ellos los mismos comportamientos. Lo único que variaba eran los tiempos totales de ejecución que para porcentajes más pequeños decrementaban y para mayores aumentaban. La única diferencia notable que se encontró al realizar las pruebas para *datasets* de porcentajes pequeños (que contenían menos del 10% de los datos totales) era que el modelo de *Gradient boosting* y Regresión logísitca tenían prácticamente los mismos tiempos de ejecución en los diferentes niveles de saturación.

4.2.2 Portátil

Resultados Room Occupancy

Los tiempos de ejecución que empleó el portátil para entrenar los diferentes modelos con este *dataset* se encuentran en la Tabla 4.5

En esta ocasión, a diferencia del comportamiento de la Raspberry los tiempos de cpu van aumentando con las diferentes pruebas, a pesar de que los tres primeros modelos solo

Modelo	Accuracy cv (%)	Accuracy trainning (%)	Acuracy (%)			
			Idle	2 cpu	4 cpu	8 cpu
Regresión Logísitca	98.9	99.0	98.9	98.9	98.7	99.0
SVM	99.0	99.0	98.9	98.8	99.0	98.8
Gradient Boosting	99.0	99.3	99.0	98.9	98.8	99.0
Random Forest	99.2	100.0	99.2	99.3	99.1	99.2

Tabla 4.6: Resultados de la precisión de los modelos para el Occupancy dataSet en el portátil

pueden ejecutar en una cpu y por lo tanto no deberían de afectarles que hubiese más de un core estresado. Sin embargo, este incremento en los tiempos se puede deber a la frecuencia con la que ejecutan las cpus. En el portátil la frecuencia máxima a la que pueden ejecutar los cores es de 3600MHz mientras que la mínima son 400MHz.

Ejecutando el siguiente comando:

```
$ watch -n.1 "grep \"^[c]pu MHz\" /proc/cpuinfo"
```

Se puede ver la frecuencia de ejecución de todos los cores cada 0.1 segundos. En esta ocasión al observar estas velocidades cuando se ejecutaban los algoritmos de aprendizaje automático se podía ver una variación en los valores dependiendo del nivel de estrés que se estuviese ejecutando, a diferencia de lo que ocurría en la Raspberry que se mantenía siempre a la máxima frecuencia. Luego esto explicaría porque en el caso del portátil los tiempos incrementan según aumentan el número de cores estresados.

Por otra parte, el *Wall time* tiene un comportamiento similar al de *Cpu time* aumentando a medida que se incrementa el nivel de carga computacional en todos los modelos.

Veamos a continuación cual es el *Accuracy* que consiguen los modelos con este *dataset* en esta máquina.

En la Tabla 4.6 se puede ver que en este caso también se consiguen unos modelos de gran precisión tanto en la validación cruzada, como con los datos de entrenamiento además de en cada una de las diferentes pruebas.

Resultados kdd_cup99

Los resultados de la ejecución del comando *stress* conforme a estos niveles se pueden observar en la tabla 4.7. Como se puede apreciar los tiempos que se obtienen en el portátil disminuyen bastante con respecto a los que conseguía la Raspberry.

Según esta tabla, *Random forest* es el modelo que más tiempo de cpu tarda cuando la máquina se encuentra en estado *Idle* y se mantienen el resto de pruebas sobre los 300 segundos de ejecución. Esto se debe a las mismas razones dadas en la sección 4.2.1. El hecho

Modelo	Idle		2 CPUs estresadas		4 CPUs estresadas		8 CPUs estresadas	
	Cpu time	Wall time	Cpu time	Wall time	Cpu time	Wall time	Cpu time	Wall time
Regresión logísitca	67.95 seg	20.28 seg	99.58 seg	29.96 seg	126.63 seg	44.05 seg	168.25 seg	77.76 seg
SVM	114.81 seg	116.45 seg	166.42 seg	168.05 seg	317.07 seg	318.72 seg	327.69 seg	365.5 seg
Gradient boosting	176.09 seg	177.74 seg	232.99 seg	234.63 seg	268.3 seg	270.03 seg	390.09 seg	429.31 seg
Random forest	392.73 seg	55.88 seg	327.2 seg	59.44 seg	291.76 seg	69.48 seg	256.92 seg	81.99 seg

Tabla 4.7: Resultados de los tiempos de ejecución para el kdd_cup99 dataSet en el portátil.

Modelo	Accuracy cv (%)	Accuracy trainning (%)	Acuracy (%)			
			Idle	2 cpu	4 cpu	8 cpu
Regresión Logísitca	99.4	99.4	99.4	99.4	99.4	99.4
SVM	99.9	99.9	99.9	99.8	99.8	99.9
Gradient Boosting	99.9	100.0	96.4	96.5	99.3	99.3
Random Forest	100.0	100.0	99.7	97.4	99.7	96.8

Tabla 4.8: Resultados de la precisión de los modelos para el Kdd_cup99 dataSet en Pc

de que los tiempos aumenten en los modelos *monocore* se deben a las mismas razones que las comentadas en la ejecución del *dataset* de Room Occupancy en el portátil 4.2.2.

Y al igual que en el anterior *dataset*, el *Wall time* aumenta conforme más cores se estresan.

Respecto a la precisión de los modelos generados podemos ver los resultados en la Tabla 4.8. Que al igual que el resto de resultados obtenidos en el otro *dataset* los datos no se sobreajustan y se están consiguiendo resultados de gran precisión en todas las pruebas.

4.2.3 Conclusiones datos sintéticos

Tras haber observado los resultados obtenidos en la ejecución de cada uno de los modelos en diferentes situaciones de estrés en ambos dispositivos con los dos *datasets*, se ha podido comprobar qué a pesar de la menor capacidad de la Raspberry, esta es capaz de tener un comportamiento bastante bueno.

Donde más se nota la falta de recursos de la Raspberry es en los tiempos de ejecución. Para los mismos modelos y los mismos datos, tarda más en ejecutar la Raspberry que el portátil. En la Figura 4.2 se muestra cuánto de lenta es la Raspberry respecto al portátil

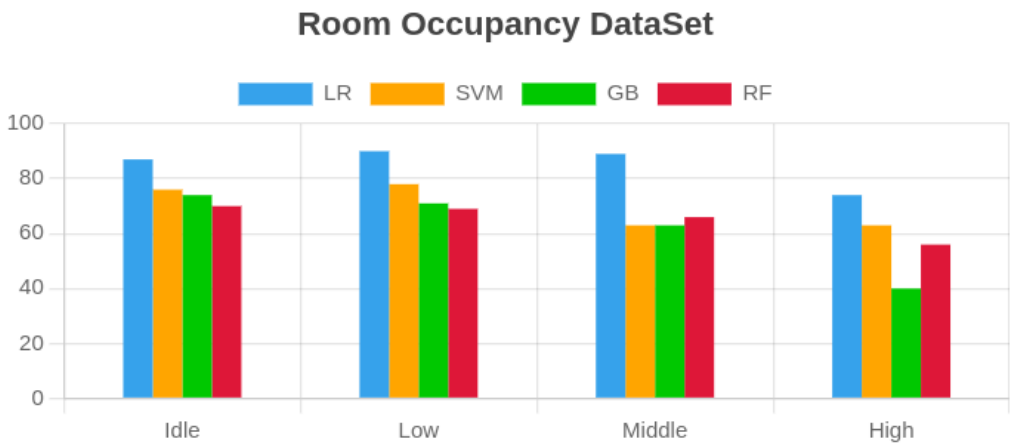

Figura 4.2: Porcentaje de lentitud de la Raspberry respecto al portátil para Room Occupancy.

por medio de porcentajes en cada uno de los niveles de estrés para el Room Occupancy DataSet.

En dicha figura podemos ver como a medida que el nivel de saturación aumenta (eje x) es menor el porcentaje de lentitud, con lo que cada vez es menor la diferencia entre los tiempos de ejecución de la Raspberry y del portátil. Siendo el modelo de RL en el que mayor diferencia hay entre ambas máquinas, a pesar de que este modelo es el que menos tarda en ejecutar en el portátil no sucede lo mismo en la Raspberry lo que hace que se termine convirtiendo en el modelo que peor se comporta en la Raspberry al compararlo con el Pc.

Por otra lado, es el modelo de *Gradient boosting* el que consige estar por debajo del 50% de lentitud. Cuando ambas máquinas están con todos los cores estresados, GB es solo un 40% más lenta en la Raspberry que en el portátil.

De la misma forma, en la Figura 4.3 se muestra estos mismos porcentajes pero esta vez para el caso del *dataset* KddCup99.

Al igual que antes los porcentajes van decrementando a medida que mayor cantidad de cpus hay estresadas. En este caso sigue siendo el modelo de RL el que peor comportamiento realiza en comparación con el portátil. Mientras que esta vez, es el modelo de SVM el que consigue que su porcentaje de lentitud sea menor del 40% para los dos últimos niveles y por lo tanto los tiempos en ambos casos son los que menos se diferencian con los del portátil.

Una de las razones por las que la Raspberry es más lenta que el portátil es debido a las frecuencias con las que ejecutan ambas máquinas. Observandolas vemos que la máxima velocidad a la que pueden ir los cores en la Raspberry es a 1500MHz mientras que el portátil tiene su máximo en 3600MHz, más del doble que la Raspberry. Luego esto permite que los procesos en el portátil puedan terminar antes que en la Raspberry siendo uno de

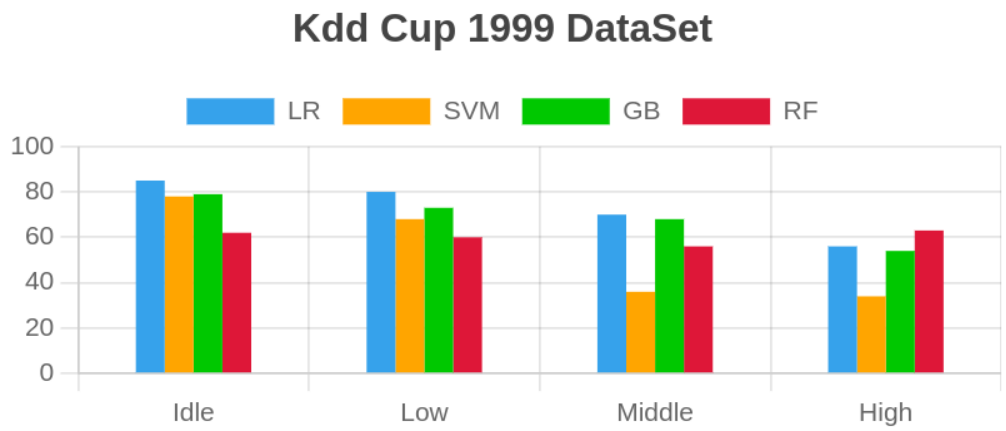

Figura 4.3: Porcentaje de lentitud de la Raspberry respecto al portátil para KddCup99.

los factores por los que la Raspberry puede ser más lenta que el portátil, ya que los cores pueden ir a mayor velocidad.

Otro factor que puede estar ayudando a que la diferencia de tiempos entre ambas máqinas sea mayor es, como cabe esperar, debido a que la Raspberry dispone de la mitad de cores que posee el portátil.

Ambos aspectos son los que pueden hacer notoria la gran diferencia de tiempos de ejeución en ambas máquinas.

Sin embargo, a parte de la diferencia de los tiempos, todos los modelos tanto entrenando en la Raspberry como en el portátil alcanzaban un *Accuracy*, *Precision* y *Recall* mayor al 90% prácticamente siempre en todos los casos.

4.3 Experimentos con los datos capturados por sensores

Tal y como se comentó en el apartado 3.6 se generó un *dataset* mediante los datos proporcionados por los sensores. El *dataset* que que se generó de esta forma se denomina como occup_detec_2h_Clean.csv, en total contiene 9745 líneas de datos que equivalen a unas dos horas y cuarenta minutos de recolección de datos.

4.3.1 Raspberry

Los resultados de los tiempos de ejecución para el *dataset* generado por medio de los datos, obtenidos de los sensores, se pueden ver en la Tabla 4.9.

Modelo	Idle		1 CPUs estresadas		2 CPUs estresadas		4 CPUs estresadas	
	Cpu time	Wall time	Cpu time	Wall time	Cpu time	Wall time	Cpu time	Wall time
Regresión logísitca	10.73 seg	9.51 seg	19.39 seg	12.52 seg	18.73 seg	13.77 seg	17.4 seg	13.38 seg
SVM	10.48 seg	10.35 seg	10.32 seg	10.24 seg	10.26 seg	10.21 seg	10.27 seg	10.44 seg
Gradient boosting	17.28 seg	17.17 seg	17.15 seg	17.1 seg	17.11 seg	17.08 seg	17.07 seg	17.14 seg
Random forest	42.58 seg	20.21 seg	38.37 seg	21.2 seg	35.41 seg	23.93 seg	36.78 seg	27.68 seg

Tabla 4.9: Resultados de los tiempos de ejecución para Mi dataSet en la Raspberry.

Modelo	Accuracy cv (%)	Accuracy trainning (%)	Acuracy (%)			
			Idle	1 cpu	2 cpu	4 cpu
Regresión Logísitca	98.9	99.0	99.2	99.1	99.2	99.0
SVM	99.5	99.6	99.5	99.5	98.6	98.5
Gradient Boosting	99.9	100.0	99.9	99.8	99.5	98.9
Random Forest	99.9	100.0	99.8	99.9	99.7	99.7

Tabla 4.10: Resultados de la precisión de los modelos para Mi dataSet en la Raspberry.

Como se puede observar, se mantiene un comportamiento tanto en los tiempos de cpu como en *Wall time* bastane similar a lo visto en los anteriores *datasets* ejecutados en la Raspberry.

Mientras que la precisión obtenida por medio de este *dataset* se encuentra en la Tabla 4.10.

A pesar de ser el *dataset* con el menor número de muestras de todos los que se han visto en este proyecto, la precisión que alcanzan los modelos sigue siendo bastante alta.

4.3.2 Portátil

Los resultados de los tiempos de ejecución para el *dataset* generado por medio de los datos obtenidos de los sensores se pueden ver en la Tabla 4.11.

Que una vez más se ajusta a lo obtenido en los anteriores *datasets*.

Mientras que la precisión obtenida por medio de este *dataset* se encuentra en la Tabla 4.12.

Modelo	Idle		2 CPUs estresadas		4 CPUs estresadas		8 CPUs estresadas	
	Cpu time	Wall time	Cpu time	Wall time	Cpu time	Wall time	Cpu time	Wall time
Regresión logísitca	2.0 seg	1.36 seg	2.27 seg	1.55 seg	2.96 seg	2.24 seg	3.27 seg	2.81 seg
SVM	1.65 seg	1.66 seg	1.77 seg	1.78 seg	2.71 seg	2.72 seg	2.8 seg	3.23 seg
Gradient boosting	3.67 seg	3.67 seg	4.12 seg	4.13 seg	4.28 seg	4.29 seg	6.18 seg	7.18 seg
Random forest	13.29 seg	3.35 seg	12.28 seg	3.87 seg	11.82 seg	4.57 seg	13.29 seg	8.39 seg

Tabla 4.11: Resultados de los tiempos de ejecución para Mi dataSet en la Raspberry.

Modelo	Accuracy cv (%)	Accuracy trainning (%)	Acuracy (%)			
			Idle	2 cpu	4 cpu	8 cpu
Regresión Logísitca	99.0	99.0	99.0	99.0	99.1	99.8
SVM	99.5	99.5	99.7	99.5	98.6	98.6
Gradient Boosting	99.9	100.0	99.7	99.9	99.5	99.8
Random Forest	99.9	100.0	99.5	99.8	99.7	99.5

Tabla 4.12: Resultados de la precisión de los modelos para Mi dataSet en la Raspberry.

Obteniendo como cabía esperar resultados de precisión muy buenos, y sabiendo que en ningún momento se ha sobreajustado ningún modelo gracias a el porcentaje de validación cruzada tan elevado que se consigue en todos los casos.

4.3.3 Conclusiones de los datos capturados por los sensores

Con los resutltados obtenidos podemos llegar a las mismas conclusiones que en el caso de los datos sintéticos, a pesar de las limitaciones que tiene la Raspberry esta es capaz de llevar a cabo perfectamente las tareas encomendadas.

En este caso los porcentajes de lentitud de la Raspberry respecto del portátil se pueden ver en la Figura 4.4.

En esta ocasión, salvo en el caso *Idle*, sigue siendo RL el modelo en el que hay más diferencia entre lo que tarda en ejecutar en la Raspberry y en el portátil, siendo en todos los casos como mínimo un 80% más lenta. Mientras que el modelo de RF es el que tiene menor diferencia de tiempos.

Además, gracias a los resultados obtenidos podemos afirmar que la Raspberry tiene

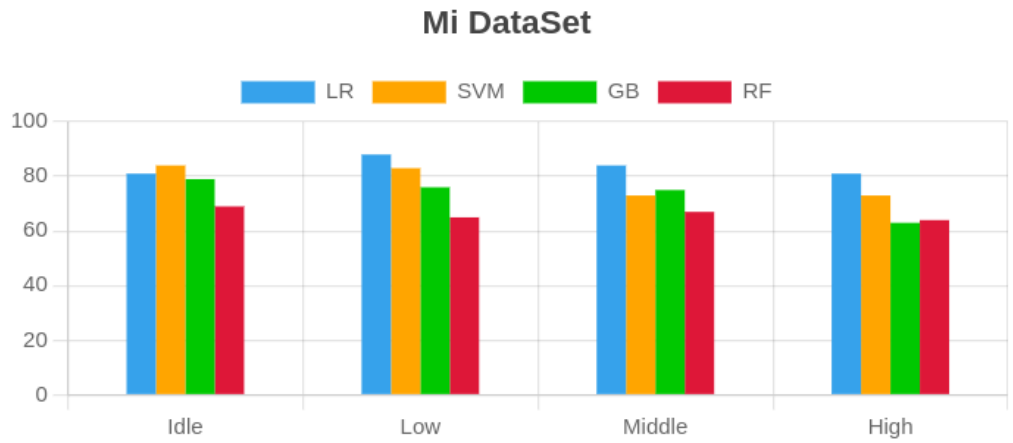

Figura 4.4: Porcentaje de lentitud de la Raspberry respecto al portátil para Mi dataSet.

suficiente capacidad para poder ir reentrenando periodicamente los modelos. De forma que cada cierto intervalo de tiempo, se capturase durante una cantidad de tiempo los datos proporcionados por los sensores, una vez terminado ese intervalo de tiempo se guardan los datos recogidos en memoria local y se vuelve a entrenar al modelo con ellos. Con cada reentrenamiento se conseguiría cada vez un comportamiento más preciso y por lo tanto mucho más fiable.

Por lo tanto, sabiendo esto podemos conseguir hacer que la Raspberry adquiera datos, los almacene, procese y analice de forma completamente autónoma. Con esto se podrian implementar muchas aplicaciones totalmente autónomas en las que la Raspberry tomando medidas el entorno pueda reaccionar correctamente a él. Concretamente dentro del campo del internet de las cosas, puesto que habiendo obtenido estas conclusiones podemos afirmar que la Raspberry puede ser capaz de actuar como un dispositivo IoT.

4.4 Conclusión final

Como se ha expuesto a lo largo de esta memoria, el objteivo del trabajo era poder determinar si la Raspberry podia encargarse de realizar tareas de Machine Learning, de la misma forma que es capaz de hacerlo un dispositivo de mayor capacidad. Gracias a los resultados obtenidos en esta investigación se puede afirmar que la Raspberry Pi 4B tiene potencial suficiente como para poder trabajar con estos modelos.

Y no solo entrenarlos o hacer pruebas con ellos, sino que también puede recoger datos y generar o reentrear modelos de forma totalmente autónoma, como se ha comentado anteriormente.

Luego, a pesar de las limitaciones notables de la Raspberry ésta no tiene nada que envidiar a un portátil, puesto que es capaz de cumplir perfectamente con los objetivos marcados aún siendo un dispositivo bastante más pequeño, de menor coste y, por supuesto, de menor capacidad.

Esto abre un abanico de infinitas posibilidades en las que se puede utilizar la Raspberry como dispositivo para poder obtener información e interactuar con el entorno de manera inteligente y completamente autónoma. Concretamente dentro del mundo de IoT y Tiny Machine Learning, puesto que cumple todos los requisitos necesarios como para poder utilizarla para implementar una tarea de ese estílo.

4.4.1 Conclusiones finales de cada uno de los modelos

Como conclusiones finales de cada uno de los modelos de Machine Learning ejecutado en ambas máquinas y utilizando las Figuras 4.2, 4.3 y 4.4, podemos mencionar los siguientes aspectos.

El modelo de Regresión Logísitica es el modelo que más tarda en terminar la ejecución en comparación con el portátil. Como ya mencionaba en secciones anteriores, a pesar de que es el modelo que más rápido ejecuta en el portátil, no sucede lo mismo en la Raspberry lo que hace que se termine convirtiendo en el modelo con mayor diferencia de tiempos entre una máquina y otra.

Maquinas de Soporte Vectorial es el modelo más rápido en la Raspberry, ya que siempre tarda menos en ejecutar que el de RL pero no en el portátil. Y sin embargo es el segundo modelo con peor comportamiento respecto al portátil en prácticamente todos los casos, puesto que suele tener altos porcentajes de lentitud en comparación con el resto de modelos.

En cuanto al mododelo de *Gradient Boosting* es el tercer modelo más lento tanto en la Raspberry como en el portátil. Por otro lado, es el segundo modelo que tiene mejor relación entre los tiempos del portátil y la Raspberry, puesto que en todas las gráficas de porcentajes siempre es el segundo con menor porcentaje.

Por último, *Random Forest* es para ambas máquinas el modelo más lento con diferencia. Y por ello, también es el modelo que por lo general hay menor diferencia de tiempos hay entre la ejecución en una máquina u otra.

Pero sin duda algo que tienen en común todos los modelos es la gran precisión con los que son capaces de predecir, mantiendo siempre un *Accuracy*, *Cross-validation*, *Recall* y *Precision* sobre el 90%. Por lo que podemos confiar en que ante nuevas entradas de datos los modelos serían capaces de poder clasificar correctamente los datos.

Capítulo 5

Conclusiones y trabajos futuros

En este capítulo se exponen las conclusiones finales de este tranajo de fin de grado, además de posibles aicaciones futuras.

5.1 Consecución de objetivos

En este proyecto se establecieron varios objetivos 1.1.2 de forma que al ir cumpliendo poco a poco con cada uno de ellos pudiesemos obtener respuesta a la gran pregunta que plantea este proyecto, ¿Es la Raspberry capaz de realizar tareas que impliquen la implementación de modelos de Machine Learning?, e incluso ir un paso más allá planteando la posibilidad de que este dispositivo fuese capaz de recoger sus datos creando su propio *dataset* y utilizarlo para entrenar los modelos, tal y como un dispositivo IoT haría.

Por en este apartado se verán si se han cumplido dichos objetivos y de qué manera.

El primero de ellos era poder implementar diferentes modelos de Machine Learning en la Raspberry, esto queda resuelto en la sección 3.3 donde se explica de qué forma se han implementado los diferentes códigos para poder entrenar los cuatro modelos especificados en la sección 2.2.

En el Capítulo 4 se explica primeramente en que van a consisitir las distintas pruebas para poder evaluar a la Raspberry. En la sección 4.1.1 se detalla como se ha implementado el código encargado de ejecutar las diferentes pruebas. Tanto en la Raspberry como en el dispositivo de mayor capacidad.

Además, todas estas pruebas se han realizado utilizando varios *datasets* para el entrenamiento de los modelos, de forma que nos ha permitido comprobar como actúa la Raspberry

ante diferentes tipos de problemas. Puesto que tal y como se comenta en el Capítulo 3 cada uno ha sido obtenido de sitios diferentes poseyendo cada uno características y datos totalmente diferentes. Incluyendo un *dataset* generado completamente a partir de los datos obtenidos de los sensores, cumpliendo con el último objetivo propuesto y demostrando de esta forma que la Raspberry cumple todos los requisitos para poder ser utilizada como dispositivo IoT.

Todo esto desarrollado dentro de un repositroio de GitHub que contiene todo lo necesario para poder replicar todos los experimentos.

Esta sección es la sección espejo de las dos primeras del capítulo de objetivos, donde se planteaba el objetivo general y se elaboraban los específicos.

Es aquí donde hay que debatir qué se ha conseguido y qué no. Cuando algo no se ha conseguido, se ha de justificar, en términos de qué problemas se han encontrado y qué medidas se han tomado para mitigar esos problemas.

Y si has llegado hasta aquí, siempre es bueno pasarle el corrector ortográfico, que las erratas quedan fatal en la memoria final. Para eso, en Linux tenemos aspell, que se ejecuta de la siguiente manera desde la línea de *shell*:

```
aspell --lang=es_ES -c memoria.tex
```

5.2 Aplicación de lo aprendido

A continuación hago mención de algunas asignaturas que he cursado durante mi grado que me han permitido desarrollar y elaborar este Trabajo de Fin de Grado.

1. **Fundamentos de la programación**. Fue el primer contacto que tuve con el mundo de la programación, en la que pude aprender lo básico de la programación mediante Python.

2. **Sensores y actuadores**. En esta asignatura tuve la oportunidad de utilizar una Raspberry por primera vez en mi vida. Además aprendí lo necesario para poder saber como obtener información de los sensores por medio de los puertos gpio.

3. **Aprendizaje automático**. Esta asignatura desperó en mi el interés sobre el aprendizaje automático y la gran cantidad de posibilidades que este nos ofrece. Pude aquirir conociemintos sobre los principales tipos de aprendizaje, así como varios modelos de cada uno de estos tipos como por ejemplo Regresión Logísitca, Árboles de decisión, Redes neuronales...

5.3 Lecciones aprendidas

Gracias a este Trabajo de Fin de Grado he adquirido los nuevos conocimientos que detallo a continuación:

1. Desarrollar diferentes tipos de modelos de aprendizaje automático utilizando la librería `scikit-learn`.
2. Saber como tratar diferentes tipos de *dataset* para facilitar y mejorar el entrenamiento de los modelos de Machine Learning.
3. Poder comprender mejor el funcionamiento interno de la Raspebrry.
4. Preparando la página web del proyecto he aprendido los conocimientos básicos para poder crear una página web por medio de HTML y cómo utilizar GitHub Pages como servidor.

5.4 Trabajos futuros

Por medio de este trabajo hemos podido vislumbrar las capacidades que tiene la Raspberry para poder realizar tareas de Machine Learning. Sabiendo esto se podrían desarrollar infinitos proyectos aplicando estos algoritmos. A continuación se hace mención a posibles trabajos futuros que se podrían realizar con estos conocimientos:

- Desarrollar una aplicaicón práctica de IoT utilizando la Raspberry como punto de recolección, procesado de datos y entrenamiento/reentrenamiento de los datos. Con las predicciones obtenidas de los modelos o bien se podría hacer que la Raspberry interactuase directamente con el entorno por medio de algunos actuadores o bien comunicar el resultado de la predicción a otros dispositivos IoT para ampliar el alcance y la repercusión de los datos obtenidos.
- Seguir investigando a cerca de otros dispositivos de capacidad de cómputo limitada para ampliar la lista de dispositivos de estas características que puedan desarrollar este tipo de tareas.

Ningún proyecto ni software se termina, así que aquí vienen ideas y funcionalidades que estaría bien tener implementadas en el futuro.

Es un apartado que sirve para dar ideas de cara a futuros TFGs/TFMs.

Capítulo 6

Anexo

En la preparación del entorno 3.2, para el desarrollo de este proyecto, surgieron algunas dificultades. En este capítulo se comentarán todos estos problemas y la solución que se halló a ellos.

Como se comenta en el capítulo 3 uno de los primeros pasos fue la instalación de Miniforge, pero antes de intentar usar este gestor de paquetes se intentó instalar Miniconda en el sistema operativo que viene por defecto en la Raspberry (Raspbian Pi OS), de forma que permitiese crear un entorno virtual con una versión de Python superior a la 3.7. Sin embargo, debido a la arquitectura de 32-bit empleada por dicho sistema operativo, no era posible instalar una versión de Python superior a la 3.6 por medio de Miniconda, pues para esas versiones se requería una arquitectura de 64-bit.

Por lo que fue necesario instalar Ubuntu 21.10 cuya arquitectura es de 64-bit. Aún así, tampoco se pudo instalar Miniconda con una versión de Python 3.8 o 3.9. La solución recayó en instalar Miniforge que proporciona un administrador de paquetes conda, muy similar a la función que desempeña Miniconda.

Una vez creado el entorno virtual con una verisón de Python igual a la 3.9, se procedió a intentar acceder a los pines GPIO desde este mismo entorno. Para poder acceder a ellos comunmente siempre se ha utilizado un paquete denominado RPi.GPIO, pero los métodos utilizados por dicho paquete, para la comunicación con los pines de la Raspberry, dejaron de funcionar en versiones de kernels de Linux iguales o superiores a la 5.11. La versión de kernel utilizada en este trabajo es la 5.13, por tanto la librería GPIO no puede resolver la comunicación con los pines.

Para versiones de Ubuntu iguales o superiores a la 21.04, existe un nuevo paquete llamado LGPIO que implementa las funciones necesarias para poder acceder a los pines. Para poder utilizar este paquete dentro del entorno virtual creado, fue necesario instalarlo pri-

meramente fuera de este, utilizando sudo apt-get install para después mover manualmente los ficheros instalados dentro del directorio del entorno virtual. Con esto, y ejecutando el fichero con permisos de root, finalmente se puede acceder a los pines y por lo tanto leer o escribir en ellos.
Cuando se quiere instalar un paquete dentro del entorno se utilizan los comandos conda install o bien pip3 install, sin embargo, por ninguno de estos dos medios se pudo obtener LGPIO de forma funcional.

Siglas

CPU Central processing unit. 64

CSV Comma Separated Values (Valores Separados por Comas). 64

GPIO General Purpose Input Ouput. 64

IoT Internet of Things. 64

LDR Light Dependent Resistor. 64

ML Machine Learning (Aprendizaje automático). 64

SIMD Single Instruction/Multiple Data. 64

SMP Symmetric Multi-Processing. 64

TFG Trabajo de Fin de Grado. 64

Glosario

Kaggle Portal web que reune la comunidad más numerosa de científicos de datos y estudiantes de machine learning del mundo. Propiedad de Google. 64

Machine Learning Machine Learning o Aprendizaje supervisado, es una disciplina del campod de la Inteligencia Artificial que permite a las máquinas aprender sín ser explicitamente programadas para ello.. 64

Tiny Machine Learning Tiny Machine Learning es un subcampo del Machine Learning dedicado a implementar loas algoritmos de aprendizaje automaico en dispositivos con capacidades limitadas.. 64

Referencias

[1] Scikit-learn developers. *Gradient Tree Boosting Documentation*. Scikit-learn. URL: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html (visitado 09-03-2021).

[2] Scikit-learn developers. *Logistic Regression Documentation*. Scikit-learn. URL: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html (visitado 09-03-2021).

[3] Scikit-learn developers. *Random Forest Documentation*. Scikit-learn. URL: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html (visitado 09-03-2021).

[4] Scikit-learn developers. *Support Vector Machine Documentation*. Scikit-learn. URL: https://scikit-learn.org/stable/modules/svm.html (visitado 09-03-2021).

[5] kukuroo3. *Room Occupancy detection data (IoT sensor)*. Kaggle. URL: https://www.kaggle.com/kukuroo3/room-occupancy-detection-data-iot-sensor (visitado 14-02-2022).

[6] timeamagyar. *kdd-cup-99-python*. timeamagyar. URL: https://github.com/timeamagyar/kdd-cup-99-python/blob/master/kdd%20preprocessing.ipynb (visitado 02-03-2022).

[7] Irvine University of California. *KDD Cup 1999 Data*. University of California, Irvine. URL: https://www.kaggle.com/datasets/galaxyh/kdd-cup-1999-data.

[8] uptodiff. *kdd-cup-99-Analysis-machine-learning-python*. uptodiff. URL: https://github.com/uptodiff/kdd-cup-99-Analysis-machine-learning-python/blob/master/kdd_binary_classification_naive_bayes.py.