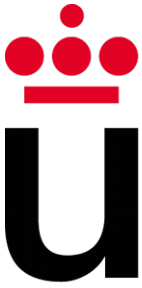

Universidad
Rey Juan Carlos

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 Limitada
Título del Trabajo con Letras Capitales para Sustantivos y Adjetivos

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 3 de de 20XX, siendo calificada por el siguiente tribunal:

Presidente:

Secretario:

Vocal:

y habiendo obtenido la siguiente calificación:

Calificación:

Móstoles/Fuenlabrada, a de de 20XX

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

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

En este capítulo se introduce el proyecto. Debería tener información general sobre el mismo, dando la información sobre el contexto en el que se ha desarrollado.

No te olvides de echarle un ojo a la página con los cinco errores de escritura más frecuentes[1].

Aconsejo a todo el mundo que mire y se inspire en memorias pasadas. Las memorias de los proyectos que he llevado yo están (casi) todas almacenadas en mi web del GSyC[2].

1.1 Sección

Esto es una sección, que es una estructura menor que un capítulo.

Por cierto, a veces me comentáis que no os compila por las tildes. Eso es un problema de codificación. Al guardar el archivo, guardad la codificación de "ISO-Latin-1" a "UTF-8" (o viceversa) y funcionará.

1.1.1 Estilo

Recomiendo leer los consejos prácticos sobre escribir documentos científicos en LaTeX de Diomidis Spinellis[3].

[1] http://www.tallerdeescritores.com/errores-de-escritura-frecuentes
[2] https://gsyc.urjc.es/~grex/pfcs/
[3] https://github.com/dspinellis/latex-advice

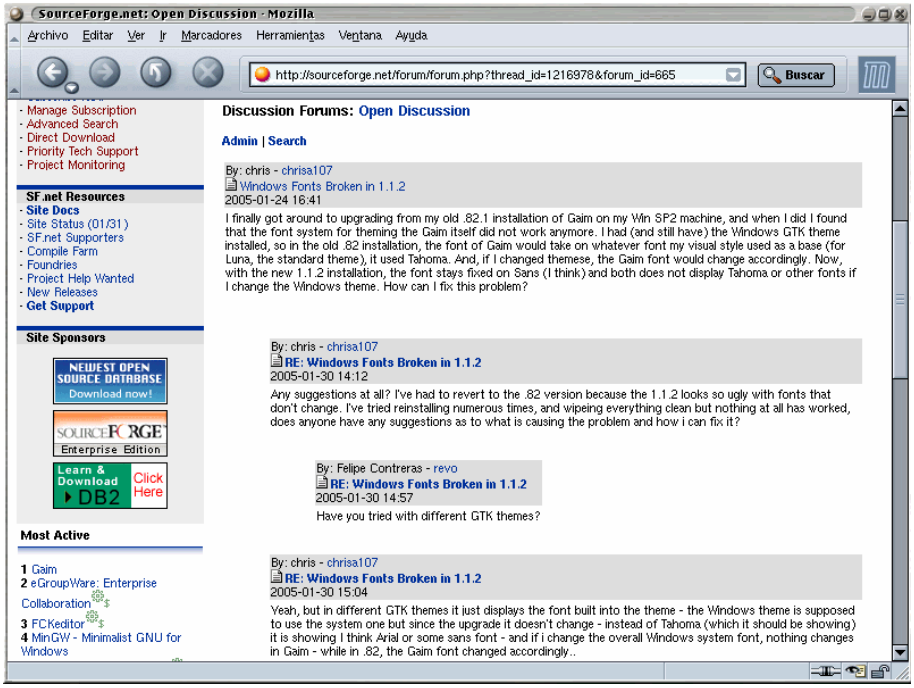

Figura 1.1: Página con enlaces a hilos

Lee sobre el uso de las comas[4]. Las comas en español no se ponen al tuntún. Y nunca, nunca entre el sujeto y el predicado (p.ej. en "Yo, hago el TFG" sobre la coma). La coma no debe separar el sujeto del predicado en una oración, pues se cortaría la secuencia natural del discurso. No se considera apropiado el uso de la llamada coma respiratoria o *coma criminal*. Solamente se suele escribir una coma para marcar el lugar que queda cuando omitimos el verbo de una oración, pero es un caso que se da de manera muy infrecuente al escribir un texto científico (p.ej. "El Real Madrid, campeón de Europa").

A continuación, viene una figura, la Figura 1.1. Observarás que el texto dentro de la referencia es el identificador de la figura (que se corresponden con el "label" dentro de la misma). También habrás tomado nota de cómo se ponen las "comillas dobles" para que se muestren correctamente. Nota que hay unas comillas de inicio (``) y otras de cierre ('') y que son diferentes. Volviendo a las referencias, nota que al compilar, la primera vez se crea un diccionario con las referencias, y en la segunda compilación se "rellenan" estas referencias. Por eso hay que compilar dos veces tu memoria. Si no, no se crearán las referencias.

A continuación un bloque "verbatim", que se utiliza para mostrar texto tal cual. Se puede utilizar para ofrecer el contenido de correos electrónicos, código, entre otras cosas.

```
From gaurav at gold-solutions.co.uk  Fri Jan 14 14:51:11 2005
From: gaurav at gold-solutions.co.uk (gaurav_gold)
Date: Fri Jan 14 19:25:51 2005
```

[4] http://narrativabreve.com/2015/02/opiniones-de-un-corrector-de-estilo-11-recetas-para-escribir-corr
html

```
Subject: [Mailman-Users] mailman issues
Message-ID: <003c01c4fa40$1d99b4c0$94592252@gaurav7klgnyif>
Dear Sir/Madam,
How can people reply to the mailing list?  How do i turn off
this feature? How can i also enable a feature where if someone
replies the newsletter the email gets deleted?
Thanks
From msapiro at value.net  Fri Jan 14 19:48:51 2005
From: msapiro at value.net (Mark Sapiro)
Date: Fri Jan 14 19:49:04 2005
Subject: [Mailman-Users] mailman issues
In-Reply-To: <003c01c4fa40$1d99b4c0$94592252@gaurav7klgnyif>
Message-ID: <PC1730200501141048510578O1b04d55@msapiro>
gaurav_gold wrote:
>How can people reply to the mailing list?  How do i turn off
this feature? How can i also enable a feature where if someone
replies the newsletter the email gets deleted?
See the FAQ
>Mailman FAQ: http://www.python.org/cgi-bin/faqw-mm.py
article 3.11
```

1.2 Objetivos del proyecto

1.2.1 Objetivo general

El objetivo de este Trabajo de Fin de Grado es comprobar la capacidad y eficiencia de una Raspberry Pi 4B para entrenar modelos de aprendizaje automático.

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.

1.2.2 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:

- Un objetivo específico.
- Otro objetivo específico.
- Tercer objetivo específico.
- ...

1.3 Planificación temporal

Es conveniente que incluyas una descripción de lo que te ha llevado realizar el trabajo. Hay gente que añade un diagrama de GANTT. Lo importante es que quede claro cuánto tiempo has consumido en realizar el TFG/TFM (tiempo natural, p.ej., 6 meses) y a qué nivel de esfuerzo (p.ej., principalmente los fines de semana).

1.4 Estructura de la memoria

Por último, en esta sección se introduce a alto nivel la organización del resto del documento y qué contenidos se van a encontrar en cada capítulo.

- En el primer capítulo se hace una breve introducción al proyecto, se describen los objetivos del mismo y se refleja la planificación temporal.
- En el siguiente capítulo se describen las tecnologías utilizadas en el desarrollo de este TFM/TFG (Capítulo 2).
- En el capítulo 3 Se describe el proceso de desarrollo de la herramienta ...
- En el capítulo 4 Se presentan las principales pruebas realizadas para validación de la plataforma/herramienta...(o resultados de los experimentos efectuados).
- Por último, se presentan las conclusiones del proyecto así como los trabajos futuros que podrían derivarse de éste (Capítulo 5).

Capítulo 2

Estado del arte

Descripción de las tecnologías que utilizas en tu trabajo. Con dos o tres párrafos por cada tecnología, vale. Se supone que aquí viene todo lo que no has hecho tú.

Puedes citar libros, como el de Bonabeau et al., sobre procesos estigmérgicos [1]. Me encantan los procesos estigmérgicos. Deberías leer más sobre ellos. Pero quizás no ahora, que tenemos que terminar la memoria para sacarnos por fin el título. Nota que el~añade un espacio en blanco, pero no deja que exista un salto de línea. Imprescindible ponerlo para las citas.

Citar es importantísimo en textos científico-técnicos. Porque no partimos de cero. Es más, partir de cero es de tontos; lo suyo es aprovecharse de lo ya existente para construir encima y hacer cosas más sofisticadas. ¿Dónde puedo encontrar textos científicos que referenciar? Un buen sitio es Google Scholar[1]. Por ejemplo, si buscas por "stigmergy libre software" para encontrar trabajo sobre software libre y el concepto de *estigmergia* (¿te he comentado que me gusta el concepto de estigmergia ya?), encontrarás un artículo que escribí hace tiempo cuyo título es "Self-organized development in libre software: a model based on the stigmergy concept". Si pulsas sobre las comillas dobles (entre la estrella y el "citado por ...", justo debajo del extracto del resumen del artículo, te saldrá una ventana emergente con cómo citar. Abajo a la derecha, aparece un enlace BibTeX. Púlsalo y encontrarás la referencia en formato BibTeX, tal que así:

[1] http://scholar.google.com

```
@inproceedings{robles2005self,
  title={Self-organized development in libre software:
         a model based on the stigmergy concept},
  author={Robles, Gregorio and Merelo, Juan Juli\'an
          and Gonz\'alez-Barahona, Jes\'us M.},
  booktitle={ProSim'05},
  year={2005}
}
```

Copia el texto en BibTeX y pégalo en el fichero `memoria.bib`, que es donde están las referencias bibliográficas. Para incluir la referencia en el texto de la memoria, deberás citarlo, como hemos hecho antes con [1], lo que pasa es que en vez de el identificador de la cita anterior (bonabeau:_swarm), tendrás que poner el nuevo (robles2005self). Compila el fichero `memoria.tex` (`pdflatex memoria.tex`), añade la bibliografía (`bibtex memoria.aux`) y vuelve a compilar `memoria.tex` (`pdflatex memoria.tex`)...y *voilà* ¡tenemos una nueva cita [7]!

También existe la posibilidad de poner notas al pie de página, por ejemplo, una para indicarte que visite la página del GSyC[2].

En este apartado se reaizará una breve descripción del dispositivo principal, que es motivo de esta investigación. También se expondrá todo el software que se ha necesitado instalar para poder implementar los modelos y ponerlos a prueba.

2.1 Machine Learning e IoT

El Machine Learning es una disciplina del campo de la Inteligencia Artificial, dota a los ordenadores de la capacidad de identificar patrones en datos masivos y elaborar predicciones

2.2 Dispositivo Hardware: Raspberry Pi 4 Modelo B

Raspberry Pi[3] 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

[2] http://gsyc.es
[3] https://www.raspberrypi.org/

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 utilizar otros sistemas.

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 cuatro cores físicos y otros cuatro lógicos. 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 por lo que permite que cualquier procesador trabaje en cualquier tarea sin importar su localización en memoria.

Es uno de los dispositivos más populares en la actualidad ya que brinda un sinfín de posibilidades a pesar de su pequeño tamaño. Específicamente dentro de el campo de aprendizaje automático 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.

2.3 Sistema Operativo: Ubuntu 21.10

El sistema operativo Ubuntu[4] 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

[4] https://ubuntu.com/

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.

2.3.1 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.

2.4 Lenguaje de programación: Python

Python[5] 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, 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. Además es multiplataforma y de código abierto 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.

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

2.4.1 Entorno de desarrollo: Jupyter-notebook

Jupyter-notebook es una aplicación web 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 pueden ejecutar para visualizar los datos y ver los resultados de salida. Además se puede utilizar tanto remotamente como en local.

2.4.2 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[6] 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.

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[7] 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.

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

2.5 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 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

Antes de adentrarnos en una explicación en mayor profundadida de lo que se desarrolla en este proyecto

Si tu proyecto es un software, siempre es bueno poner la arquitectura (que es cómo se estructura tu programa a "vista de pájaro").

Por ejemplo, puedes verlo en la Figura **??**. LaTeX pone las figuras donde mejor cuadran. Y eso quiere decir que quizás no lo haga donde lo hemos puesto…Eso no es malo. A veces queda un poco raro, pero es la filosofía de LaTeX: tú al contenido, que yo me encargo de la maquetación.

Recuerda que toda figura que añadas a tu memoria debe ser explicada. Sí, aunque te parezca evidente lo que se ve en la Figura **??**, la figura en sí solamente es un apoyo a tu texto. Así que explica lo que se ve en la Figura, haciendo referencia a la misma tal y como ves aquí. Por ejemplo: En la Figura **??** se puede ver que la estructura del *parser* básico, que consta de seis componentes diferentes: los datos se obtienen de la red, y según el tipo de dato, se pasará a un *parser* específico y bla, bla, bla…

Si utilizas una base de datos, no te olvides de incluir también un diagrama de entidad-relación.

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 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 el cual se crea un entorno virtual donde se instalaron todos los paquetes necesarios para el proyecto como son scikit-learn, pandas, jupyter-notebook, stressberry... Estos paquetes permitirán desarrollar modelos de aprendizaje automático y poder ponerlos a prueba bajo diferentes estados de la Raspberry.

En el Anexo de esta memoria se podrá encontrar en mayor detalle las dificultades y las soluciones a estos problemas encontrados a la hora de realizar todo lo comentado en este apartado.

3.3 Modelos de aprendizaje automático

Con el entorno preparado se generaron mediante scripts de Python cuatro modelos diferentes de aprendizaje supervisado para realizar una clasificación binaria. Utilizando la libreía scikit-learn para generar un modelo de Regresión logística, otro de Máquinas de soporte vectorial, Gradient boosting y un último de Random forest.

3.3.1 Regresión logística

Modelo de aprendizaje supervisado que realiza tareas de clasificación binaria. Esta técnica busca una función acotada (normalmente entre 0 y 1) que divida los datos de ambas clases de forma bien diferenciada.

En scikit-learn por medio la función LogisticRegression[3], que pertenece a la librería

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

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. Por otra párte 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.

3.3.2 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.

Por medio de la función SVC [5], se puede generar un modelo de aprendizaje utilizando esta técnica. 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.

3.3.3 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.

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.

En esta ocasión el método que crea este modelo es GradientBoostingClassifier[2], 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.

3.3.4 Random forest

Random forest se puede usar tanto en problemas de clasificación como de regresión. Utiliza un conjunto de árboles de decisión con bagging, es decir, los árboles de decisión se generan de forma paralela. Al combinar sus resultados unos errores se compensan con otros, obteniendo una predicción que generaliza mejor.

En scikit-learn utilizando la función RandomForestClassifier[4], 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 la misma forma se utiliza este parámetro en el caso de Regresión logística cuando se busca entreanr un problemas de clasificación multiclase.

3.4 DataSet: Room Occupacy

Para realizar la clasificación en un inicio se utilizó el data set de Room Occupancy detection data[6], 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, humedad, 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 sensados 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, 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 tres modelos se pudieron generar sin problemas utilizando las librerías de scikit-learn y pandas. Para valorar la ediciencia del entrenamiento se pueden obtener tres valores diferentes: *accuracy*, *precision* y *recall*. Gracias a la librería *metrics* de scikit-learn se pueden obtener respectivamente estos valores por medio de las funciones *accuracy_score*, *precision_score* y *recall_score*. A estos métodos se les pasa como argumentos la variable que contiene los valores de la salida del set de datos de prueba (generada, como se ha comentado anteriormente, con el 30% de los datos originales), y como segundo argumento el propio modelo generado (después de entrenarlo). A continuación se detalla que significan cada uno de esto valores, siendo TP igual a verdaderos positivos, TN verdaderos negativos, FP falsos positivos y FN falsos negativos.

El valor de *accuracy* representa el porcentaje total de aciertos.

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

Por otra parte *precision* es el porcentaje que se obtiene de dividir el verdadero número de positivos (en este caso el uno o positivo representa que la habitación está ocupada) entre todos los positivos que se han asignado (sean correctos o no), cuanto más alto sea este valor menor error habrá cometido al asignar habitaciones ocupadas.

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

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.

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

Todos conseguían un *accuracy*, *precision* y *recall* superior al 90%, el tiempo de ejecución era de un segundo para regresión logística, dos segundos tardaba el modelo de máquinas de soporte vectorial y gradient tree boosting y random forest eran los que más tardaban con un tiempo de cuatro y tres segundos respectivamente.

3.4.1 Validación cruzada

Se añadió un poco más de dificultad a la Raspberry haciendo que los algoritmos utilizasen validación cruzada, es decir, que dentro del set de datos de entrenamiento (compuesto, una vez más, por el 70% de ejemplos del set de datos original) se hace una subdivisión en otros cinco sets para entrenar y probar el modelo con cada uno de ellos. Una vez realizada la validación cruzada se vuelve a probar la eficiencia del modelo haciendo que clasifique el 30% de datos de la división original, son datos que nunca ha visto ni entrenado con ellos. El objetivo de realizar una validacion cruzada es garantizar que los resultados que obtengamos sean independientes de la partición entre datos de entrenamiento y datos de validación. Los resultados de esta prueba siguieron siendo muy buenos (el acierto seguía estando por encima del 90%). En cuanto a los tiempos de ejecución el modelo de regresión logística incrementó el tiempo de ejecución siendo de cinco segundos. Para el modelo de máquinas de soporte vectorial el tiempo es de seis segundos. Random forest tarda en entrenar unos doce segundos. Y por último, el que más tiempo tarda es gradient tree boosting con un tiempo de diez y seis segundos.

3.5 DataSet: KddCup99

Al realizar algunas pruebas para comprobar 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 Datadata[6], que lo obtuve una vez más desde Kaggle. Se utilizó tanto el fichero *kddcup.data.gz* como *kddcup.data_10_percent.gz*. Una vez descargados se descomprimieron y convirtieron a clase binaria puesto que este dataSet

contiene varias clases. Para ello se utilizó el fichero *Download_dataSet.py*. En dicho programa se leía o bien el dataSet que contenía el diez porciento del dataSet total o se utilizaba el dataSet completo para poder leer algo más que el diez porciento de los datos. En el caso de la Raspberry el máximo de datos que era capaz de leer sin que el proceso muriera era el cuarenta porciento del fichero, en el portátil era un cincuenta porciento.

Para obtener otro porcentaje de dataSet que no fuese el diez porciento se utilizaba una regla de tres para saber la cantidad de datos que se querían leer, pues sabiendo que el diez porciento contenía 494020 líneas, se podía obtener aproximadamente a cuantas líneas equivaldría otro porcentaje.

Para leer estos dataSets se utilizó la función *read_csv* de la librería pandas. Una vez leídos los datos se convertía de multiclase a clase binaria reemplazando todas las clases (excepto la clase *normal.*) por la clase *attack*. El código que realiza estas tareas se puede ver en Código 3.1.

```
#Lectura csv
n= 4940200 # número total de líneas
s= 988040 # Leer solo el 20\%
skip= sorted(random.sample(range(n),n-s))
dataset = pd.read_csv('/home/nuriadj/Documents/TFG/kdd_cup99/kddcup.data', skiprows=skip)

#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 3.1: Lectura del dataset y conversion a clase binaria.

Una vez realizada la transformación se hace un pequeño tratamiento a los datos para que cada modelo pueda entrenar adecuadamente y más fácilmente con ellos. Este tratamiento se ejecuta en el mismo programa que lee los datos (*Download_dataSet.py*). En este fichero lo primero que se hace es eliminar 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 utilizando 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 códifo de cada uno de los modelos los datos estén listos para poder ser utilizados.

Al igual que en el dataSet de Room Occupancy se utiliza el 70% de los datos para entrenar y el 30% para validar.

Capítulo 4

Experimentos y validación

Atención: Este capítulo se introdujo como requisito en 2019.

En este capítulo comprobaremos la eficiencia de la Raspberry a la hora de generar los modelos comentados en el capítulo anterior para que se ajusten al dataSet kdd_cup99 comentado anteriormente. 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 Experimentos en la Raspberry

Para comprobar la capacidad de la Raspberry para generar estos modelos se definieron cuatro niveles de saturación: el nivel "Idle" en el que no se somete a la máquina a ningún tipo de carga extra, el nivel bajo, que consistía 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. Para estresar las cpus se utilizó el comando *stress* que permite estresar, durante el tiempo que se le indique, el número de cpus que se comanden.

Luego cada uno de los modelos se sometía a estos cuatro niveles de carga. La medida que se utiliza para comparar los diferentes niveles 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, el tiempo que ha tardado en ejecutar el proceso como si lo hubiesemos cronometrado con un reloj, este tiempo es igual que el Wall time. Es necesario entender esto para poder comprender los valores que nos devolverán los experimentos.

En caso de querer someter a la Raspberry a un nivel mayor de carga computacional se ejecutaban paralelamente el comando *stress* y el modelo.

4.1.1 Programa para las pruebas: raspberry_test.ipynb

Para realizar las pruebas en la Raspberry de forma más sencilla se creó un programa llamado *raspberry_test.ipynb*. Dicho programa necesita como único argumento el data set con el que se desea hacer el experimento para que pueda realizar su función correctamente.

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. 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 (ver código 4.2). 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 cores 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 número de cores, 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, Accuracy, Presisión y Recall se van guardando cada uno en su respectiva variable gracias a una función del programa llamada *get_data*. Una vez que uno de los modelos ha completado todas las pruebas se genera una tabla mediante la función *add_data* con los resultados obtenidos en cada uno de los casos. Dicha tabla se guardará en un fichero que se encarga de crear *raspberry_test.ipynb*. Este fichero tendrá por nombre el siguiente formato: Raspberry_nombreDataSet_results.txt. Cada una de las tablas generadas por los modelos de aprendizaje serán guardadas en el mismo fichero.

Los nuevos procesos, comentados anteriormente, se crean utilizando la función *Popen*

```
t= 200
list_cpus= [0,1,2,4]
dataSet= sys.argv[1]
name_dataset= "Raspberry_"+dataSet.split("/")[-1].split(".")[0]
f= open(name_dataset+'_results.txt', 'w+')
f.close()

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

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

Código 4.2: Función principal raspberry_test.ipynb

de la librería *subprocess*. Nótese que en el método *model* (a diferencia de *start_stress*) esta función 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 para poder comandar otro *stress* sin tener que esperar a que se cumpla el tiempo de ejecución que le pasamos como argumento como se puede ver en el Código 4.2.

4.1.2 Resultados

Los resultados obtenidos para el veinte por ciento del data set de Kdd_cup99 son los siguientes:

Modelo	Idle		1 cpu		2 cpu		4 cp	
	Cpu time	Wall time	Cpu time	Wall time	Cpu time	Wall time	Cpu time	Wall time
Regresión logísitca	1 min 36 seg	38 seg	1 min 48 seg	50 seg	1 min 33 seg	56 seg	1 min 34 seg	57 seg
SVM	2 min 19 seg	2 min 16 seg	2 min 17 seg	2 min 18 seg	2 min 12 seg	2 min 12 seg	2 min 18 seg	2 min 18 seg
Gradient boosting	3 min 19 seg	3 min 20 seg	3 min 19 seg	3 min 20 seg	3 min 15 seg	3 min 15 seg	3 min 16 seg	3 min 17 seg
Random forest	3 min 4 seg	1 min 3 seg	2 min 30 seg	1 min 5 seg	2 min 3 seg	1 min 10 seg	2 min 10 seg	1 min 18 seg

Tabla 4.1: Tiempos de generación de los modelos en Raspberry para diferentes niveles de saturación

Como se puede observar en la tabla anterior, los tiempos varian entre los diferentes modelos, siendo el de Regresión logística el que menos tiempo tarda en todos los casos.

Un aspecto destacable es que los tiempos de Regresión logísitca, Máquinas de soporte vectorial (SVM) y Gradient boosting apenas varían entre los diferentes niveles de estrés. En el caso de Máquinas de soporte vectorial y Gradient boosting esto se debe a que 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 3.3.1.

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 tiempo 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 cores. El tiempo que ha necesitado cada uno de los cores sería dividir 3min 4seg entre 4 con lo que obtendríamos 46 segundos por core. Sabiendo esto queda explicado por qué el nivel de Idle devuelve un valor tan alto y como esto, que a prioir puede parecer un poco contradictorio, se ajusta a lo esperado.

Para el resto de casos, dado que ya intervienen otros procesos, habrá que saber 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 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 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 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 utilizasen todas las cpus de la Raspberry. Este ejemplo se puede ver en la figura 4.1.

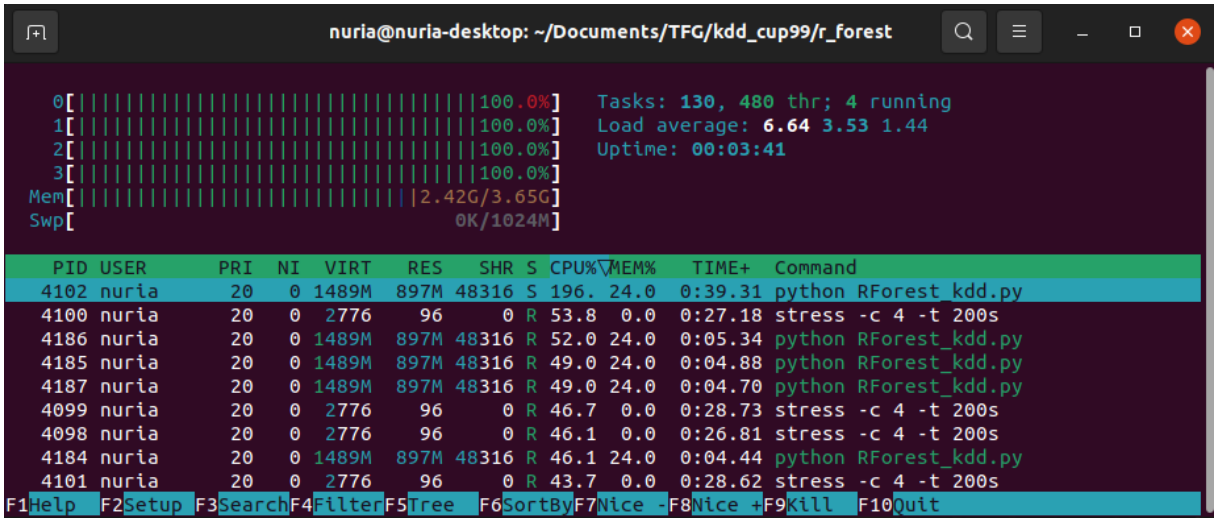

Figura 4.1: Ejecución Random forest con cuatro cpus estresadas

En la figura 4.1 vemos que el proceso de Random forest está consumiendo el 196% de cpu, prácticamente un 200%. Mientras que si sumamos los porcentajes de los cuatro procesos de *stress* obtendremos que se están consumiento en total un 190%. 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 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 cores 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 tiene 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 cores posibles.

Otro aspecto de las ejecuciónes en la Raspberry que será relevante más adelante, es la frecuencia a la que ejecutan los core. 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 Idle podremos ver que todas las cpus están sobre los 1500MHz. Por lo tanto todos los cores se encuentran a la máxima velocidad a la que pueden ejecutar y por ello stress no puede provocar que ejecuten a una velocidad mayor.

4.2 Experimentos en el portátil

Las mismas pruebas que se realizaron en la Raspberry se llevaron a cabo en un portátil. 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 esta máquina.

El portátil que se utilizó para estas pruebas tenía un total de ocho cpus físicas. Es por ello que los parámetros de bajo, medio y alto de carga computacional cambiaron para poner más o menos en la misma situación al portátil y a la Raspberry. Por lo tanto en el portátil 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.

Para realizar las pruebas se ha creado un programa llamado *pc_test.ipynb* que al igual que el fichero *raspberry_test.ipynb* ejecuta cada uno de los modelos con los diferentes niveles de estrés y guarda los datos obtenidos un nuevo fichero que crea el programa con el formato Pc_nombreDataSet_results.txt. Al igual que en *raspberry_test.ipynb* este programa creará dos procesos uno ejecutará stress (cuando se necesite) y el otro los modelos que se encuentran dentro de la carpeta Modelos. Todos los ficheros que generan los modelos 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.

Los resultados de la ejecución del comando stress conforme a estos nivels podremos se pueden observar en la tabla 4.2. Como se puede apreciar los tiempos que se obtienen en el portátil disminuyen bastante con respecto a los que conseguía la Raspberry.

Modelo	Idle		2 cpu		4 cpu		8 cp	
	Cpu time	Wall time	Cpu time	Wall time	Cpu time	Wall time	Cpu time	Wall time
Regresión logísitca	18 seg	7 seg	21 seg	8 seg	29 seg	13 seg	48 seg	27 seg
SVM	31 seg	31 seg	43 seg	243 seg	58 seg	58 seg	1 min 22 seg	1 min 32 seg
Gradient boosting	41 seg	41 seg	54 seg	54 seg	1 min 27 seg	1 min 27 seg	1 min 29 seg	1 min 41 seg
Random forest	1 min 16 seg	13 seg	1 min 5 seg	15 seg	58 seg	19 seg	54 seg	23 seg

Tabla 4.2: Tiempos de generación de los modelos en el portátil para diferentes niveles de saturación

Otro aspecto destacable de los tiempos en el portátil es que estos aumentaban a medida que se estresaba un mayor número de cores a pesar de que los tres primeros procesos siguen siendo monocores. Esto se debe a otro dato que no había tenido en cuenta hasta el momento que es la frecuencia con la que ejecutan las cpus.

En la Raspberry la frecuencia máxima a las que pueden ir los cores es de 1500 MHz mientras que en el portátil es de 3600 MHz. Este aumento de velocidad de frecuencia explicaría por qué en el portátil los tiempos son menores, porque pueden llegar a ir a más velocidad los procesos.

Ejecutando el siguiente comando: podía observar la frecuencia en KHz a la que ejecutaban cada uno de los cores en la Raspberry. Observando que incluso estando en estado idle los cores ejecutaban a la máxima frecuencia, luego esto es otra de las razones por la que los tiempos no incrementan con stress, porque no pueden ir a más velocidad los cores. Mientras que en el pórtatil las frecuencias si variaban al ejecutar stress.

4.3 Incorporación de código en la memoria

Es bastante habitual que se reproduzcan fragmentos de código en la memoria de un TFG/TFM. Esto permite explicar detalladamente partes del desarrollo que se ha realizado que se consideren de especial interés. No obstante, tampoco es conveniente pasarse e incluir demasiado código en la memoria, puesto que se puede alargar mucho el documento. Un recurso muy habitual es subir todo el código a un repositorio de un servicio de control de versiones como GitHub o GitLab, y luego incluir en la memoria la URL que enlace a dicho repositorio.

Para incluir fragmentos de código en un documento LaTeXse pueden combinar varias herramientas:

- El entorno `\begin{listing}[]...\end{listing}` permite crear un marco en el que situar el fragmento de código (parecido al generado cuando insertamos una tabla o una figura). Podemos insertar también una descripción (*caption*) y una etiqueta para referenciarlo luego en el texto.
- Dentro de este entorno, se puede utilizar el paquete `minted` [1], que utiliza el paquete Python Pygments para resaltado de sintaxis (coloreando el código). Como se puede ver en el siguiente ejemplo, hay muchas opciones de configuración que permiten controlar cómo se va a mostrar el código (incluir números de línea, saltos de línea, tamaño y tipo de fuente, espaciado, código de colores para resaltado, etc.).

```
# A dictionary is built to define the data type contained by each column
dtype_scheme ={'budget': np.int64, 'genres': np.object, 'homepage': np.str, 'id':
↪  np.int64, 'keywords': np.object, 'original_language': np.str, 'original_title':
↪  np.str, 'overview': np.str, 'popularity': np.float64, 'production_companies':
↪  np.object, 'production_countries': np.object, 'release_date': np.object, 'revenue':
↪  np.int64, 'runtime': np.float64, 'spoken_languages': np.object,  'status': np.object,
↪  'tagline': np.str, 'title': np.str, 'vote_average': np.float64, 'vote_count':
↪  np.int64}

# When loading the data from the .csv file, we provide the scheme to be followed for data
↪  typing
df1 = dd.read_csv('tmdb_5000_movies.csv', dtype=dtype_scheme)
```

Código 4.3: Lectura de un fichero *.csv y tipado de datos.

Otra ventaja del entorno `listing` es que se puede generar automáticamente un índice (con entradas hiperenlazadas) de fragmentos de código, para incluirlo al comienzo del documento junto con los índices de figuras, tablas, etc.

4.3.1 Fuentes monoespaciadas

A veces se incluyen nombres de archivos, paquetes, etc. como texto monoespaciado, utilizando el comando LaTeX`\texttt{}`. Sin embargo, esto puede generar un problema cuando las palabras en fuente monoespaciada alcanzan el final de una línea. En ese caso, el compilador rehusa muchas veces romper la palabra y deja la línea demasiado larga respecto al resto.

[1] https://es.overleaf.com/learn/latex/Code_Highlighting_with_minted

Para evitar esto, especialmente en párrafos más cortos de lo habitual (como en una lista no numerada), se puede utilizar el comando `\begin{sloppypar}...\end{sloppypar}`, como se muestra a continuación con un ejemplo real.

- Los valores contenidos en las columnas `genres`, `spoken_languages`, `production_companies` y `production_countries`, clasificados originalmente como `np.objects`, se corresponden en realidad con listas de objetos JSON que han sido almacenadas como cadenas de caracteres. A través de la función `get_values(obj, key)` definida específicamente para ello, se transformará dicha cadena de caracteres en una lista de diccionarios a través de la función `json.loads(obj)` y se devolverá una tupla que recopile los valores de los mismos para la clave indicada, un objeto de Python mucho más manejable de cara a realizar consultas sobre el *dataset*.

Capítulo 5

Conclusiones y trabajos futuros

5.1 Consecución de objetivos

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

Aquí viene lo que has aprendido durante el Grado/Máster y que has aplicado en el TFG/TFM. Una buena idea es poner las asignaturas más relacionadas y comentar en un párrafo los conocimientos y habilidades puestos en práctica.

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

Aquí viene lo que has aprendido en el Trabajo Fin de Grado/Máster.

1. Aquí viene uno.
2. Aquí viene otro.

5.4 Trabajos futuros

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, para el desarrollo de este proyecto, surgieron algunas dificultades.

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.

También dentro del entorno creado se instaló todo lo necesario para realizar este proyecto. Entre estos paquetes se instaló la librería de scikit-learn que tiene algoritmos para generar modelos de clasificación, regresión, clustering y reducción de la dimensionalidad. Así como pandas que tiene todas las funcionalidades necesarias para el análisis de datos como pueden ser: cargar, modelar, analizar, manipular y preparar los datos.

Además se instaló stress, un paquete para hacer pruebas bajo diferentes cargas computacionales mostrando los resultados obtenidos por medio de plots. En este caso, dicho paquete se utilizará para poder someter a la Raspberry a diferentes niveles de estrés y de este modo ver su capacidad para entrenar modelos de aprendizaje automático. Para poder ejecutar stressberry hay que tener en cuenta que el usuario que ejecute este comando debe pertenecer al grupo video, de lo contrario la ejecución dará error.

A parte de poder comprobar la capacidad de la Raspberry para ejecutar estos programas bajo diferentes cargas computacionales, también permitirá comparar los tiempos de ejecución con respecto a los tiempos que alcanzan dichos programas en un portátil corriente.

Glosario

JSON JavaScript Object Notation, traducido como notación de objeto de JavaScript, es un formato basado en el uso de texto estándar para representar datos estructurados. Aunque se basa en sintaxis JavaScript puede ser utilizado independientemente y muchos frameworks de programación poseen la capacidad de leer y generar este tipo de objetos.. 28

Referencias

[1] Eric Bonabeau, Marco Dorigo y Guy Theraulaz. *Swarm Intelligence: From Natural to Articial Systems*. Oxford University Press, Inc., 1999.

[2] 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).

[3] 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).

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

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

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

[7] Gregorio Robles, Juan Julián Merelo y Jesús M. González-Barahona. «Self-organized development in libre software: a model based on the stigmergy concept». En: *ProSim'05*. 2005.