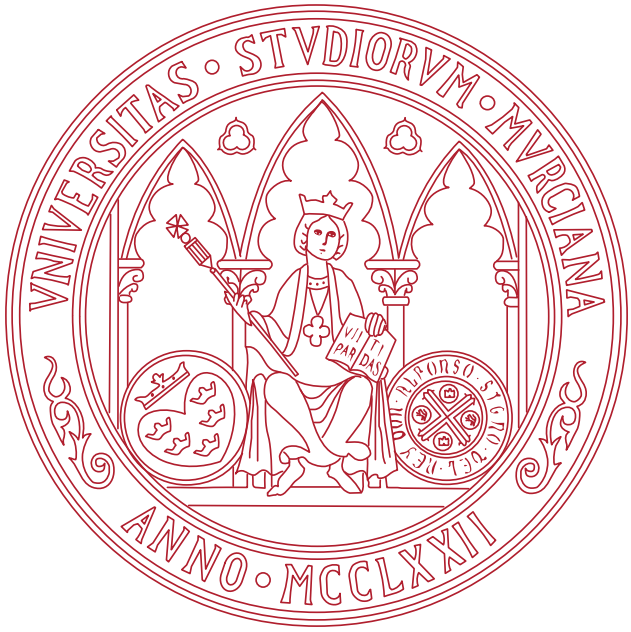


Universidad de Murcia

Trabajo Final de Grado

Identificación de dispositivos IoT a través de huellas hardware

Autor:
Sergio Marín Sánchez
sergio.marins@um.es

Tutores:
Gregorio Martínez Pérez
gregorio@um.es
Pedro Miguel Sánchez Sánchez
pedromiguel.sanchez@um.es

4 de julio de 2022

Me gustaría expresar mi agradecimiento a todos los amigos, profesores y familia que han hecho posible que llegue a este punto.

Índice general

Índice de figuras

Índice de tablas

Declaración firmada sobre la originalidad del trabajo

D. Sergio Marín Sánchez, con DNI 49197868V, estudiante de la titulación de Grado en Ingeniería Informática de la Universidad de Murcia y autor del TF titulado "Identificación de dispositivos IoT a través de huellas hardware".

De acuerdo con el Reglamento por el que se regulan los Trabajos Fin de Grado y de Fin de Máster en la Universidad de Murcia (aprobado C. de Gob. 30-04-2015, modificado 22-04-2016 y 28-09-2018), así como la normativa interna para la oferta, asignación, elaboración y defensa delos Trabajos Fin de Grado y Fin de Máster de las titulaciones impartidas en la Facultad de Informática de la Universidad de Murcia (aprobada en Junta de Facultad 27-11-2015).

DECLARO:

Que el Trabajo Fin de Grado presentado para su evaluación es original y de elaboración personal. Todas las fuentes utilizadas han sido debidamente citadas. Así mismo, declara que no incumple ningún contrato de confidencialidad, ni viola ningún derecho de propiedad intelectual e industrial.

Murcia, a 4 de julio de 2022
Fdo.: Sergio Marín Sánchez
Autor del TF

Resumen

En este proyecto se ha diseñado un modelo capaz de identificar dispositivos conectados en red local en base a las diferencias entre sus relojes internos comparados con un reloj que se fija como exacto. Con esto ha sudo creado un mecanismo que permitirá realizar conexiones más seguras.

Este trabajo se puede dividir en tres partes. Por un lado, la obtención de los datos, para lo que han sido utilizados 5 dispositivos iguales en hardware y software (Raspberry Pi 4 Model B). De estos dispositivos se obtienen marcas de tiempo cada segundo de dos formas distintas. Por otro lado, el análisis estadístico de los datos obtenidos, en el que se generan nuevos datos a partir de los previos. Por último, el desarrollo de un modelo de Machine Learning que sea capaz de automatizar todo el proceso de distinguir los dispositivos partiendo de los datos estadísticos que se han obtenido previamente.

Para obtener los datos se ha hecho uso de una arquitectura cliente-servidor entre los dispositivos a analizar y el dispositivo observador, con el fin de controlar el ritmo de envío de los paquetes. Las capturas se han realizado de dos formas, en secuencial y en paralelo. Secuencialmente hace referencia a que primero se obtienen todas las marcas de tiempo de un dispositivo y posteriormente se analiza el siguiente. Por otro lado, paralelamente hace referencia a que se obtienen marcas de tiempo de todos los dispositivos simultáneamente.

Una vez se tienen todas las marcas de tiempo, se obtienen sus desviaciones respecto al dispositivo que se designó como referencia. De estas desviaciones se obtiene su incremento entre cada dos puntos, con estos valores usamos una ventana deslizante de 1 minuto, que equivale a 60 paquetes (se toman muestras cada segundo), y con de ellos se obtienen diversas variables estadísticas.

Una vez se tienen las variables estadísticas se analizarán los modelos de Machine Learning que han sido escogidos para este trabajo. Los modelos que se van a analizar son Random Forest, MLP, KNN, Naive Bayes, Árboles de decisión y SVM con kernel lineal.

Para comparar los modelos entre sí se usará el modelo entrenamiento/validación/test como forma de dividir los datos. Se entrenarán todos los modelos con el conjunto de entrenamiento y se comprobará su capacidad de generalización con el conjunto de validación. A la hora de entrenar los modelos se comprobarán diferentes parrillas de hiperparámetros para ajustar los modelos lo más posible a los datos de entrenamiento.

Al realizar estos entrenamientos se puede apreciar que los modelos basados en árboles son los mejores, tanto árboles de decisión (97.57 %) como Random Forest (99.51 %). Finalmente

se elige el algoritmo de Random Forest y se procede a entrenarlo con la totalidad de los datos de entrenamiento y los hiperparámetros usados anteriormente.

Como resultado final se tiene que el modelo de Random Forest con los hiperparámetros ajustados obtiene un valor de accuracy de 99.44 %, fallando como mucho un 1 % en la identificación de un dispositivo en específico. Gracias a este resultado, se llega a la conclusión de que es posible automatizar el proceso de distinguir dispositivos idénticos de forma remota.

Con vistas a futuro, se puede concluir que este trabajo aporta una solución que potencialmente puede ayudar a la lucha contra la ciberdelicuencia, pero se deben resolver algunos inconvenientes. Por un lado, hay servicios que modifican constantemente el reloj del sistema y esto dificulta la tarea de crear una huella para los dispositivos. Por otro lado, y tal vez el más importante, es que esta huella ha de ser creada para cada dispositivo concreto, lo cual es una labor incomensurable.

Extended abstract

Nowadays, the number of devices connected to the internet has increased significantly due to smartphones, IoT devices, autonomous cars, etc. These devices must be always connected to the internet to be able to perform the actions for which they are intended.

These kinds of devices contain and share a lot of information about the people who use them. For this reason, many people try to get this data for purposes that may not be legal. One way to do it would be pretending that you are the device of the person that you want to access. In order to succeed, it is necessary to change all the identifiers to those that the criminal wants to impersonate. By doing this, people will connect to the criminal's device without being aware of it.

Devices that have the same hardware and software components are not exactly the same, there are some details during the manufacturing process that cannot be copied. Taking advantage of this, it will be created a system with the ability to identify those differences and as a consequence, between many devices.

As mentioned, in this article it will be created a system that can identify those details mentioned above. Many studies have already researched device identification, but most of the time they focus on distinguishing devices, but these investigations do not bring the focus to these devices that are identical in both hardware and software.

The work developed by Pascal Oser et al. [19] seeks to create a system that identifies devices among a total of 562 at the CERN research center. To execute it, they obtain the timestamps contained in the TCP header of the packets sent and received by these devices. Using these timestamps, they measure how often there is an overflow of the counter of this header and with it, they train a machine learning model capable of recognizing when each device does it.

Another example is the one developed by Salma Abdalla Hamad et al. [12] in which a device authentication system is created for the purpose of giving access to a private network. These fingerprints are created considering several packet characteristics that were sent by the devices, such as packet length, destination IP, etc. By those features, devices can be classified as potential threats and not let them access the network.

Another remarkable work is the one developed by Ahmet Aksoy et al. [2] which uses a genetic algorithm to obtain representative headers of the packet. Then, it is used different classification algorithms to first group them by brand and then identify them individually.

Finally, Hossein Jafari et al. [14] generates fingerprints of each device using radio frequencies. The process is done by obtaining samples of the SNR value in 5 different levels on each device. Then, they train three deep learning models in order to identify each one.

In this case, to identify different devices, it will be used their absolute timestamps i.e. the time elapsed since 1st January, 1970. The small differences that distinguish these devices will make that as time progresses, their internal clocks will fluctuate. This error will be accumulated and it will become more noticeable each time.

From this point, the first step will be to calculate different statistical values of how this error changes over time in the different devices. After that, a machine learning algorithm will be trained, and it will be capable of performing this process automatically.

The test scenario made for this project consists of six IoT devices connected to a local network. One of them will act as a client and the other ve as servers. Every second the client will send a timestamp request to the servers, and they will reply with it. When the client receives this timestamp, it saves it as a record: " (time from start), (absolute client time), (absolute server time), (time difference), (server ip)". The client's time will be taken as a reference, and the differences will be analyzed between the other devices and the client.

The first problem appeared at this time. Taking into account that it is going to been analyzed very small differences, it must have great accuracy of the times. Initially, it was thought that the best option was using the timestamps contained in the headers of the TCP and ICMP protocols. This idea was rejected since these headers have few bytes and with them it can only represent times in milliseconds, which does not provide enough information. For this reason, it was decided to use the body section of the packets; to be able to send data of any length. In this case, it will send absolute timestamps in nanoseconds, which will be encoded with 64 bits. Finally, it was chosen the TCP protocol because in this way, it does not have to start a new connection with a device.

These timestamps will be taken in two different ways. On the one hand, it will be taken a sequential sample and it will listen for 2 hours (7200 samples) on each device, one after another. This period of time represents a 10 hours sample (36000 samples). On the contrary, it will carry out a parallel sample of all the devices for 12 hours (43200 samples per device) Another issue that has arised in this point is that the device's internal clock is altered by other processes, such as the NTP protocol. As a result, it is necessary to use an internal clock that does not decrease its value (`steady_clock`).

The next step is set on to the data analysis part. Firstly, it be will obtained the increase in the deviation between each sample of a device. Due to these increases, it will be generated a boxplot of each sample (sequential and parallel), in which each box will represent one of the devices under analysis. These graphs correspond to Fig. 4.4 and Fig. 4.6. In these graphs, since it is being worked with such small values, the outliers hide completely the results it is necessary to see, so they will not be shown.

The median is expected to be approximately 0 and the interquartile range is expected to be very similar on each device. These results are expected since it is being analyzed clonic devices and they do not suffer from clock skew, at least theoretically.

Looking both graphs, it will be observed that the median is actually close to 0, but

both the interquartile range and the non-outliers values range oscillate much more in the sequential 8 sample. Therefore, from this point on and because of the training of the models that perform the identification process, This sample will not be taken into account; the focus will be laid on the parallel sample.

At this point the statistical values are obtained, it is going to be used a 1 minute sliding window (60 samples) to obtain them. The statistical values that will be obtained will be: sum, mean, median, mode, standard deviation, interquartile range, kurtosis, skewness, maximum and minimum.

Before training the models, it is going to be checked if there is a correlation between the different statistical values, since having correlated data does not provide information. To do it, a correlation matrix between all the statistical variables is generated and those variables that have a high correlation value with another variable are eliminated. In order to train the different models, it is going to be used the `scikit-learn` library for Python, along with utilities such as `numpy` and `pandas`. In this work it will be used supervised learning algorithms, and those that are going to be used are:

- **Decision trees**: starting from a set of elements in the parent node, binary questions are asked (yes or no) such that the set can be divided into two purer ones.
- **Random forest**: set of decision trees working in parallel. Each tree will perform the divisions randomly, with which variability can be achieved in the results of each one. Finally, the output is taken as the class that has been the majority in the results.
- **Multilayer Perceptron (MLP)**: feed-forward neural network algorithm i.e. without cycles. This algorithm is trained using backpropagation, which together with an optimization such as gradient descent updates the weights of every node, with the aim of minimizing the Loss Function.
- **Naive Bayes**: Bayesian algorithms try to calculate the probability that a data belongs to a class. Once it has the probabilities of belonging to every class, the algorithm assigns to that data the class whose probability is higher. To do this the algorithm calculates the conditional probability of the attributes of the data, but this is very expensive and therefore the hypothesis must be relaxed. That's why this algorithm is called naive.
- **K Nearest Neighbours (KNN)**: this algorithm groups data by distances. When the algorithm wants to classify a new data, it looks at the nearest k's. The majority class among those k will define the class of the new data.
- **Support Vector Machines (SVM)**: this algorithm places the data as n-dimensional points. The goal is to divide the space by hyperplanes in such a way that the points in the same region belong to the same class and they are as far as possible from the others.

In order to train different models and obtain the best possible results, it must be set their hyperparameters. To make this adjustment, it will be used a smaller set of data, but it must be representative of the totality of the data since each training costs a considerable amount of time. It is going to be used a training/validation/test model.

the set of all the data will be divided into two, one with 70 % of the data and another

with 30 %. The training set will be used to train the final model and the test model will be used to see the generalization capacity of the model.

To obtain the final model it is pivotal to choose the algorithm and hyperparameters that give the best results according to the data. To get it, each algorithm must be adjusted to obtain its best results and with them decide which algorithm to use.

The process of adjusting an algorithm takes time since each algorithm have to be trained with each combination of hyperparameters that is required for being tested. For this reason, the volume of data will be reduced (70 % of the total) and it will be keet only 35 % of it. This subset will also be divided into a 70/30 ratio in order to validate the results of training with data that the model has not seen before.

To make these partitions, random samples are taken, but as the data had temporal correlation, it is needed to reorder them so that this correlation can be kept and the models are able to recognize it.

To adjust the hyperparameters of an algorithm, an object called grid, which allows to specify all the hyperparameter that will be tested, is going to be used. This tool is very useful since it automates the whole process of testing different hyperparameters and allows us to obtain, in a single run, the results of an algorithm with each combination of hyperparameters.

Once all the models have been adjusted, it is easy to see that the ones with the best accuracies are those that are based on trees; both decision trees and random forest. In particular random forest is the one that gives us the best results, so, this random forest algorithm will be chosen to be trained as the final model.

Finally, the random forest algorithm with its custom hyperparameters and the training set in its completeness is the only one that have to be trained. Once this has been done, its ability to generalize is checked with the test set and it is obtained a final accuracy of 99.44 %.

It can be concluded that it is possible to identify theoretically identical devices automatically, but it should be noted that all of this process has been made on a private network. If the same study had been done over the Internet, the result would have been different. This is because even if all the devices were on the same local network and only the observer was out, each timestamp of each device could be routed differently, which would cause measurement errors. A possible solution to this problem would be to take much longer samples in time, since statistically the packets between two devices will be routed most of the time along the same path, leaving those who are not as outliers.

On the other side of the argument, the graph of the deviation was expected to be linear, but this is not the case even if a clock that doesn't decrease its value is being used and the NTP service has been disabled.

CAPÍTULO 1

Introducción

En los últimos años el número de dispositivos conectados a internet se ha incrementado en gran medida [17]. Esto se debe al uso de smartphones, tablets y demás dispositivos que requieren de conexión a internet para llevar a cabo la mayoría (o la totalidad) de tareas para las que han sido diseñados.

Cada dispositivo conectado a internet tiene asociados varios identificadores, como la dirección IP y la dirección MAC. Estos identificadores deberían servir para identificar unívocamente a un dispositivo, pero en la práctica no se da esta situación. Las direcciones IP pueden cambiar automáticamente debido al direccionamiento IP dinámico (mediante servidores DHCP), pero también pueden ser modificadas por las propias personas.

Estas modificaciones pueden ser por temas únicamente de privacidad, pero en muchas ocasiones están relacionadas con la ciberdelincuencia. Los delincuentes pueden intentar falsificar sus identificadores con el objetivo de que las personas, buscando conectarse a un servicio legítimo, acaben conectándose a sus equipos.

Debido a esto es necesario saber si el equipo con el que se quiere establecer es el correcto o no. En este punto se debe de hablar de la identificación de dispositivos de forma remota, existen varias alternativas para realizar esto. Por una parte, se puede identificar la clase del dispositivo, es decir, si es un ordenador, una impresora, un dispositivo IoT, etc. Por otra parte, se puede identificar el modelo del dispositivo, por ejemplo, distinguir un dispositivo de una marca A y modelo B de uno que sea de marca X y modelo Y. Por último, y la parte más difícil, se pueden distinguir dispositivos a nivel individual, aunque presenten el mismo hardware.

Los fines de estas suplantaciones son, por ejemplo, cifrar los datos de un equipo (ataque de ransomware), o bien, introducir virus en muchos equipos con el objetivo de realizar ataques DDoS mediante miles de equipos infectados (ataques de denegación de servicio distribuidos), entre otros.

Por todas las razones expresadas anteriormente, existe la pregunta sobre cómo podemos saber a con qué equipos se está estableciendo una comunicación, o qué diferencia a un

dispositivo de otro en internet si ambos presentan los mismos identificadores.

Una respuesta a estas preguntas es que los dispositivos aunque presenten el mismo hardware, tengan los mismos identificadores y ejecuten el mismo software, nunca serán exactamente iguales. Esto es debido a que en el proceso de fabricación de los dispositivos siempre habrá diferencias (por pequeñas que sean) que harán que los dispositivos sean distinguibles entre sí, por ejemplo, un dispositivo ejecuta una función en 1.2 ns y otro en 1.4 ns. Las diferencias son mínimas, pero existen.

Por contra, varias de las soluciones encontradas y comentadas en el capítulo de *Estado del arte* (Capítulo 2) trabajan de forma distinta. Algunas de ellas se basan en identificar un modelo de dispositivo, pero no un dispositivo individual.

En el panorama actual del Big Data y el Machine Learning podemos explotar las diferencias comentadas anteriormente de tal forma que se generen huellas de cada dispositivo y con ello saber si realmente nos estamos conectando con el dispositivo adecuado o no.

En este marco de trabajo es en el que se centra este proyecto. Se busca crear un sistema que partiendo de un reloj exacto, compare las desviaciones de los relojes de los distintos dispositivos y con ello cree una huella estadística del comportamiento de cada uno. Posteriormente se automatizará el proceso de analizar esos valores estadísticos mediante un modelo de Machine Learning.

Para lograr el objetivo del trabajo, en la identificación de dispositivos idénticos de forma automática, se han establecido diversas metas intermedias.

- **Objetivo 1**. Presentar la arquitectura IoT, así como sus diversas aplicaciones.
- **Objetivo 2**. Presentar distintas soluciones dentro del campo del Machine Learning que pueden ser aplicadas a nuestro problema.
- **Objetivo 3**. Analizar las distintas formas de obtener una marca de tiempo, con suficiente precisión, de un dispositivo.
- **Objetivo 4**. Generar un dataset con las distintas desviaciones de reloj de los dispositivos bajo análisis.
- **Objetivo 5**. Analizar estadísticamente las diferencias entre los distintos relojes de los dispositivos, con el fin de ver si son estadísticamente diferenciables.
- **Objetivo 6**. Generar un nuevo dataset con distintas variables estadísticas de las desviaciones previas.
- **Objetivo 7**. Dividir el nuevo dataset en conjuntos de entrenamiento y test para los modelos de Machine Learning, de forma que no se pierdan las características del mismo.
- **Objetivo 8**. Evaluar distintos algoritmos de Machine Learning para la tarea de distinguir entre los dispositivos.
- **Objetivo 9**. Describir las futuras vías de investigación de trabajos similares a este.

La estructura de este documento está formada en primer lugar por un resumen, tanto en español como en inglés (de forma extendida), seguidos de 6 capítulos.

En el capítulo de *Introducción* (Capítulo 1), que es el capítulo actual, se presentan el contexto, la motivación y los objetivos del trabajo. En el capítulo de *Estado del arte* (Capítulo 2) se realiza una presentación de la arquitectura IoT y sus aplicaciones, así como, una presentación del Machine Learning y algunos de sus algoritmos. En el capítulo de *Análisis de objetivos y metodología* (Capítulo 3) se analizan los objetivos propuestos y se establece una metodología de trabajo. En el capítulo de *Diseño y resolución* (Capítulo 4) se hablará de nuestra propuesta para abordar este problema. Se obtendrán varios dataset y con ellos se entrenarán diversos modelos de Machine Learning. En el capítulo de *Resultados* (Capítulo 5) se analizarán los resultados obtenidos, en concreto, se evaluarán los distintos clasificadores usados. Finalmente en el capítulo de *Conclusiones y vías futuras* se exponen las conclusiones finales del trabajo y se comentan posibles vías futuras para esta línea de investigación.

CAPÍTULO 2

Estado del arte

2.1 Dispositivos IoT

2.1.1 Introducción

El Internet de las Cosas, más conocido por Internet of Things (IoT), es el conjunto de dispositivos capaces de conectarse a internet, recolectar e intercambiar datos de forma autónoma. Una cosa que distingue a los dispositivos IoT de los que no lo son, es que estos dispositivos pueden interactuar entre ellos sin necesidad de intervención humana.

El concepto de IoT fue primeramente mencionado por Kevin Ashton en 1999 mientras realizaba una presentación sobre dispositivos RFID [10]. A pesar de esto el primer dispositivo de este tipo fue una máquina de refrescos en 1982 [28], que se conectaba a internet para informar los refrescos que contenía, y si estos se encontraban fríos.

2.1.2 Arquitectura IoT

Cada vez hay más dispositivos IoT conectados a internet, lo que dada su naturaleza, implica que son varias miles de millones de ellos. Por este motivo, los protocolos de la pila TCP/IP no están preparados para este número tan grande de dispositivos que manejar.

Existen diversas propuestas sobre arquitecturas multi capa que puedan asegurar los requisitos de calidad de servicio (QoS) y seguridad requeridos. En la propuesta de Muhammad Umar Farooq et al. [8] se hace mención a una arquitectura de 6 capas.

Por otro lado, en la propuesta de Ibrar Yaqoob et al. [30] se explica una propuesta más general, combinando varias de las capas que comparten gran parte de su cometido. A continuación se explican estas 3 capas:

1. **Capa sensitiva**: esta es la capa más básica. Es la que se encarga de recolectar toda la información que sea capaz el dispositivo a través de sus múltiples sensores.

2. **Capa de transporte**: esta capa es la que realiza todo el intercambio de información entre las otras dos capas. Se encarga de las operaciones de red.

3. **Capa de aplicación**: esta capa se encarga de extraer la información valiosa de los datos producidos por la capa sensitiva, y que le llegan a través de la capa de transporte. En esta capa se agrupan distintos servicios como data mining o computación en la nube.

2.1.3 Aplicaciones del IoT

En la actualidad, se vive rodeados de dispositivos inteligentes. Estos dispositivos permiten realizar todo tipo de acciones, por ejemplo, un teléfono móvil permite enviar y recibir emails, enviar mensajes instantáneos, utilizar redes sociales, etc. A pesar de todas esas funciones, todos estos aparatos comparten un "problema", necesitan de la intervención humana para realizar sus tareas.

De estas necesidades nacen los dispositivos IoT, por ejemplo, un frigorífico inteligente. Este frigorífico es capaz de llevar el inventario de productos que contiene y detectar cuando alguno de ellos está próximo a terminarse. De forma autónoma este dispositivo es capaz de conectarse a internet y realizar una pedido online al supermercado de nuestra preferencia para reabastecerse de estos productos.

Este es sólo un ejemplo, dentro del hogar se pueden encontrar también los muy conocidos robots aspiradora o los asistentes inteligentes, pero estos dispositivos pueden ir mucho más lejos. Estos dispositivos se pueden usar para un control inteligente del tráfico, donde un coche autónomo puede recibir en tiempo real las rutas menos congestionadas que seguir hacia un destino.

Otro de sus posibles usos está en la medicina. Donde se pueden instalar dispositivos de este tipo para medir distintas métricas de salud de los pacientes, como la frecuencia cardíaca o el nivel de glucosa en sangre. De esta forma, en caso de que una persona presente algún desorden, una ambulancia puede ser avisada de forma inmediata aunque dicha persona esté sola e inconsciente.

Como último ejemplo se mencionará al sector agrícola. En este campo se pueden desplegar drones que detecten el nivel de humedad del terreno y, por ejemplo, activen el riego de cierta zona en caso de ser necesario. También pueden haber sensores a nivel del suelo que detecten la calidad del suelo, midiendo los distintos nutrientes y actuando en consecuencia.

2.2 Machine Learning

2.2.1 Introducción

El Machine Learning se trata de una búsqueda, una búsqueda de formas alternativas de hacer algo. Para ello se busca en estructuras muy similares a las previas. Para modificar una estructura y seguir con la búsqueda nos valemos de sucesos que hayan acontecido, entonces, evaluamos si han mejorado y nos quedaremos como estructura para la siguiente iteración

con la que más mejore a la previa. Para realizar estas operaciones necesitamos una forma de cambiar de estructura y una forma de evaluar si hemos mejorado.

Con esta forma de trabajar se puede ver que el programa no tiene que resolver la tarea sino autoajustarse para obtener mejores evaluaciones cada vez. El programador por tanto no pone el programa sino proporcionar al programa la mejor estructura modificable y a continuación alimentar al sistema con datos. Aportamos por tanto ejemplos o premios que ayudan al programa a autoajustarse. Existen tres tipos de aprendizaje:

- **Supervisado**: al sistema se le proporcionan ejemplos de cuales la solución es conocida. Una variante sería el semi-supervisado en el que sólo una parte de los datos tienen solución conocida.
- **No Supervisado**: al sistema se le proporcionan datos que no tienen una solución conocida, esperando que el sistema nos proporcione conocimiento que intuimos que existe en dichos datos. Un ejemplo sería el *clustering* que agrupa los datos por similitudes.
- **Por refuerzo**: al sistema no se le pasan ejemplos de datos, sino que se le premia o castiga por distintas conductas que desarrolla automáticamente. Al final el sistema, que busca obtener más premios, se comportará de la forma que queremos.

2.2.2 Tratamiento de los datos

Para obtener un resultado de cuan bueno es un modelo (algoritmo) que se quiere usar se debe dividir el conjunto de datos inicial en dos conjuntos: **datos de entrenamiento** y **datos de test**. Con los datos de entrenamiento el algoritmo se ajusta y aprende de los ellos, y con los datos de test se analizan los resultados obtenidos del algoritmo ante datos que no ha visto antes.

Para realizar una comparación entre distintos modelos donde se quiere ver cual es el que mejor se ajusta a los datos se debe dividir el conjunto de datos de entrenamiento en otros dos conjuntos: **conjunto de entrenamiento** y **conjunto de validación**. Con este último se puede evaluar la capacidad de generalizar de nuestro algoritmo.

Con esta segunda división se obtiene el algoritmo que mejor se adapta a los datos, pero no un modelo final. Para tener un modelo final se entrenará este con el conjunto de datos de entrenamiento en su totalidad y se evaluará con los datos de test.

2.2.3 Algoritmos de aprendizaje supervisado

Dentro de los algoritmos de aprendizaje supervisado se encuentran algoritmos que clasifican y algoritmos que se dedican a la regresión.

Los algoritmos clasificadores tratan de agrupar a los distintos vectores de entrada en distintas clases. Un ejemplo de esto sería un algoritmo de clasificación binaria, en la que existen ejemplos **positivos** (pertenecen a la clase) o **negativos** (no pertenecen). Esto genera los conceptos de **falso positivo** (el sistema dice que pertenece y es falso) y **falso negativo** (el sistema dice que no pertenece y es falso).

Los algoritmos regresores buscan dar una salida numérica como resultado en lugar de la pertenencia a una clase, por tanto no sirven para este trabajo. A continuación se explicarán los algoritmos que se han usado para el desarrollo de este trabajo.

2.2.3.1 Árboles de decisión

La idea principal del algoritmo es que partiendo de un conjunto de elementos en el nodo padre, haciendo una pregunta binaria tal que se pude dividir el conjunto en otros 2 que sean más puros, es decir, que los elementos entre sí compartan (al menos) la característica por la que hemos preguntado.

Si los atributos son categóricos se puede crear una partición diciendo si pertenecen o no a la clase. Por contra, si los atributos son ordinales se pueden tener preguntas del tipo $x \leq x_c$.

2.2.3.2 Random Forest

Este algoritmo es un conjunto de árboles de decisión trabajando en paralelo. Cada árbol realizará de forma aleatoria una partición distinta a los otros para cada división de una rama, con esto se obtiene variabilidad en los resultados de los árboles. Por último, el algoritmo toma como salida aquella clase que haya sido el resultado de más árboles. [13] [4]

2.2.3.3 Multilayer Perceptron (MLP)

Se trata de un modelo de red neuronal artificial de tipo *feed-forward* (es decir, no existen ciclos en el grafo que forma la red). Consiste en una serie de múltiples capas (multilayer) de nodos que conforman grafos dirigidos, estando cada capa conectada con la siguiente.

MLP entrena la red utilizando la propagación hacia atrás (backpropagation) que, empleada junto con una técnica de optimización como gradient descent, calcula el gradiente de una función de pérdida respecto a todos los pesos en la red, de manera que se pasa el valor del gradiente al método de optimización y este lo usa para actualizar los pesos, con el objetivo de minimizar la función de pérdida.

2.2.3.4 Naive Bayes

Los algoritmos de aprendizaje bayesianos tratan de encontrar la probabilidad de que un dato pertenezca a una clase. Una vez se tienen todas las probabilidades de pertenencia a una clase se toma como salida del algoritmo aquella clase con la mayor probabilidad.

Para realizar este proceso el algoritmo usa distintos atributos de cada dato $a_1, \ldots, a_n$. Ahora este proceso se trata de conoces la probabilidad condicionada de que este conjunto de atributos pertenezca a una determinada clase. Esto es muy costoso y por tanto hay que relajar esta hipóstesis. Para ello se asume independencia entre los atributos de ahí el nombre de *naive*.

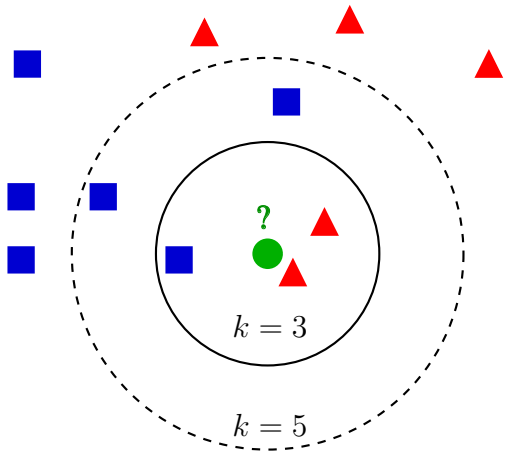


Figura 2.1: Funcionamiento del algoritmo KNN [5]

2.2.3.5 K Nearest Neighbours (KNN)

El algoritmo KNN es un algoritmo que se usa mayoritariamente para clasificación. En este algoritmo los datos se agrupan por distancias.

A la hora de clasificar un nuevo dato, este algoritmo busca los k datos más cercanos. Una vez tiene esos datos comprueba sus clases. La clase con mayor representación entre los k datos es la que se asignará al nuevo dato. Podemos verlo con un ejemplo.

En la Fig. 2.1 se quiere clasificar el dato ●, una opción es usar múltiples valores de k y con ello obtener diferentes resultados:

- Asignando $k = 3$ sólo se tendrán en cuenta los 3 datos más próximos, que serán 2 de la clase ▲ y uno de la clase ■. Con esto la clase que se asignará al dato ● será ▲.
- Asignando $k = 5$ sólo se tendrán en cuenta los 5 datos más próximos, que serán 2 de la clase ▲ y tres de la clase ■. Con esto la clase que se asignará al dato ● será ■.

2.2.3.6 Máquinas de vector soporte (SVM)

Las máquinas de vector soporte son un conjunto de algoritmos de aprendizaje supervisado que se utilizan tanto para clasificación como para regresión, en este caso se analizará únicamente en clasificación.

Estos algoritmos toman los datos como puntos en un espacio $n-$dimensional. El objetivo es dividir estos datos mediante hiperplanos de forma que los puntos que contenidos en una región del espacio delimitada por los mismos hiperplanos sean una misma clase y que estén lo más separados posible. Esto se puede ver más fácil con un ejemplo.

En la Fig. 2.2 se puede ver como hay distintos hiperplanos $(\mathcal{H}_1, \mathcal{H}_2, \mathcal{H}_3)$ que pueden dividir el espacio.

- $\mathcal{H}_1$ no divide de forma correcta todos los puntos de la entrada, puesto que hay puntos negros juntos con blancos.

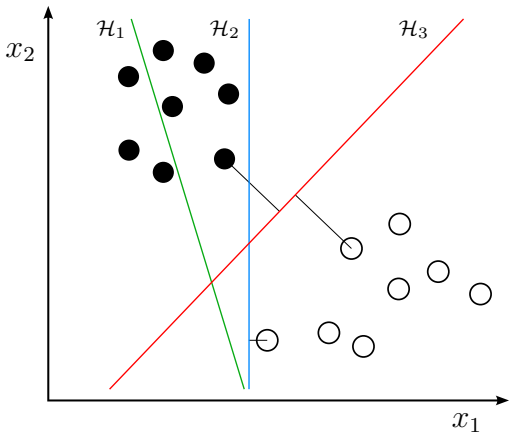


Figura 2.2: Separación por hiperplanos SVM [6]

- $\mathcal{H}_2$ sí que divide a todos los puntos en dos clases de la forma correcta, pero no están seraparas lo máximo posible de este hiperplano.
- $\mathcal{H}_3$ sí que cumple con las restricciones de división y distancia máxima.

En muchas ocasiones no se podrán dividir los conjuntos de forma correcta usando únicamente divisiones lineales del espacio, por ello, habrá que usar divisiones no lineales.

2.3 Revisión bibliográfica

En esta sección se realiza un estudio del estado del arte relativo a la aplicación de distintas técnicas que tengan como objetivo identificar a un dispositivo a través de la red independientemente de un identificador susceptible de ser identificado. Para estos cometidos son muy utilizados los algoritmos de aprendizaje automático o Machine Learning.

Para comenzar la revisión bibliográfica, se analizará el trabajo desarrollado por Pascal Oser et al. [19]. En este trabajo se desarrolla un sistema para identificar dispositivos entre un total de 562 en el centro del CERN. Para este cometido se tomas muestras periódicas de las marcas de tiempo (timestamps) contenidas en paquetes TCP que se envían y reciben de estos dispositivos.

De estas marcas se obtienen las variables que se usarán para entrenar los distintos modelos de machine learning. Las variables que obtiene son el incremento entre dos marcas consecutivas, el desbordamiento del contador de la marca de tiempo, la mediana de todas las marcas, etc.

Una variable interesante es la del desbordamiento del contador. En el RFC 1323 [3] se define que el contador del timestamp (campo TSval) tiene una longitud de 4 bytes (32 bits), por lo tanto, el mayor valor que podrá tener asignado es 2^{32}, puesto que es un entero sin signo. Se podría pensar que todos los dispositivos desbordan a la vez, pero cada fabricante establece el valor al que el contador se desborda. Esto está ilustrado en la Fig. 2.3 con dos ejemplos.

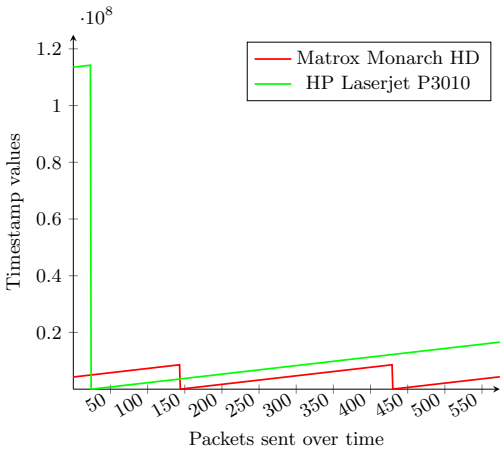


Figura 2.3: Desbordamiento del contador [19]

Una vez tiene todas las variables registradas para todos los dispositivos, compara varios modelos de machine learning, para clasificar cada uno de los dispositivos. Los resultados que obtiene son de un accuracy del 99.22 % con MLP, un 99.14 % con SVM y por último 99.67 % con Random Forest.

Con estos datos decide quedarse con el modelo de Random Forest, el cual entrena con la totalidad de los datos de entrenamiento, obteniendo finalmente una precisión del 97.03 % y un accuracy del 99.76 %.

Otro trabajo relacionado con la temática ha sido desarrollado por Salma Abdalla Hamad et al. [12]. En este trabajo se desarrolla un sistema que identifique a un dispositivo que quiere conectarse a la red y decida si se le permite realizar esta acción o se le deniega. Además este sistema comprobará periódicamente los dispositivos que están ya en la red para detectar comportamientos malintencionados.

Este sistema (Fig. 2.4), una vez se tiene una traza del dispositivo que se quiere conectar por primera vez a la red, crea una huella del dispositivo que será analizada por un modelo (previamente entrenado) que lo clasificará como desconocido o no, en caso de que esté en una lista blanca. En caso de ser un dispositivo desconocido se le denegará el acceso a la red. Por contra, si está en la lista, se consultará una matriz de autorización (que contiene la confianza de ese dispositivo según sus vulnerabilidades) para saber que privilegios recibirá ese dispositivo. Con esto el dispositivo puede contectarse como un dispositivo “confiable”, teniendo acceso a comunicarse con todos los dispositivos de la red, o por contra, de “acceso restringido” donde solo podrá comunicarse con dispositivos de ese mismo nivel de acceso.

Para los dispositivos que ya están conectados a la red, se toma una muestra aleatoria que comprobará si se han clasificado de forma correcta. En el caso de que se haya clasificado mal pero siga estando en la lista blanca se actualizará su matriz de autorización. Por otra parte, en caso de que no esté en dicha lista se pondrá en cuarentena para ser analizado más adelante.

Las huellas de los dispositivos se obtienen del payload de los mensajes enviados por ese dispositivo. De ellos se obtienen 67 características como el tamaño del paquete Ethernet, el tamaño de las cabeceras, la IP de destino, el TTL, etc.

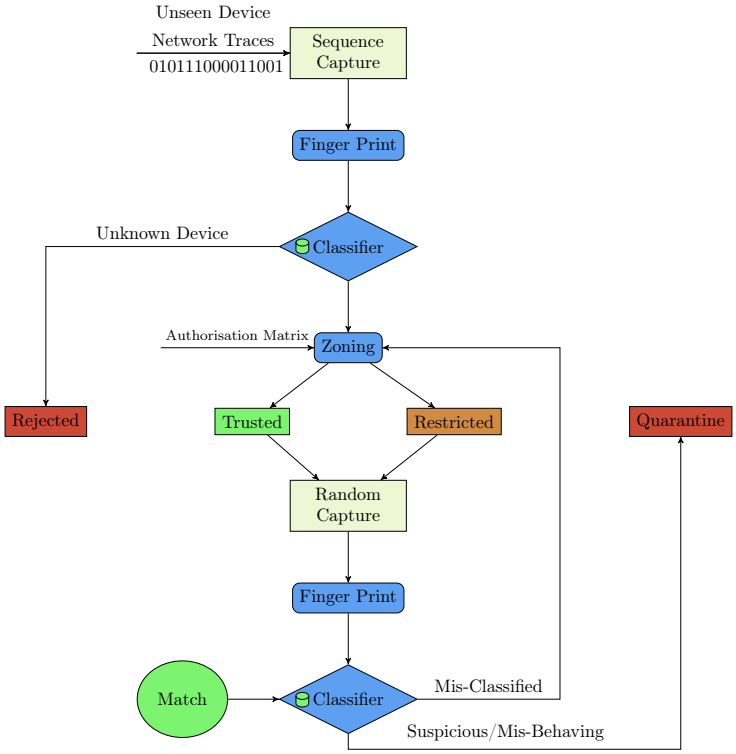


Figura 2.4: Modelo del sistema de identificación [12]

Con estas huellas entrena distintos modelos como son AdaBoost, LDA, KNN, Árboles de decisión, Naive Bayes, SVM, Random Forest y GBoost. Obtuvo un accuracy final del 89 %.

El siguiente artículo que se analizará es el realizado por Ahmet Aksoy et al. [2]. Este trabajo diseña un sistema (llamado SysID) que es capaz de identificar dispositivos a través del tráfico de red.

Este sistema únicamente necesita un paquete de red para identificar el dispositivo. Dado un paquete de la traza se seleccionan n cabeceras del paquete mediante un algoritmo genético, que implementa una función fitness (Eq. 2.1) que trata de reducir este número de cabeceras.

$$Fitness = 0.9 \cdot Accuracy + 0.1 \cdot \left(1 - \frac{|SelectedFeatures| - 1}{|AllFeatures| - 1}\right) \tag{2.1}$$

donde $n = |SelectedFeatures|$ y $Accuracy$ es el resultado que se obtiene del modelo de machine learning entrenado con esas cabeceras.

Las trazas que se han usado para clasificar estos dispositivos provienen de una base de datos de otro artículo [18], de donde se obtienen 20 medidas de 23 dispositivos IoT.

Para la clasificación se han usado diferentes algoritmos de la herramienta WEKA [11] como son tablas de decisión, árboles de decisión J48, OneR y PART. Se decide usar una clasificación en dos niveles (Fig. 2.5), se clasifican primero entre su proveedor o el propio dispositivo, dependiendo de si hay más de un mismo dispositivo del mismo proveedor. Después se clasifican los dispositivos del mismo proveedor entre sí. Este esquema ayuda a tener mejores resultados pues cada proveedor puede ser identificado de mejor forma por un algoritmo distinto.

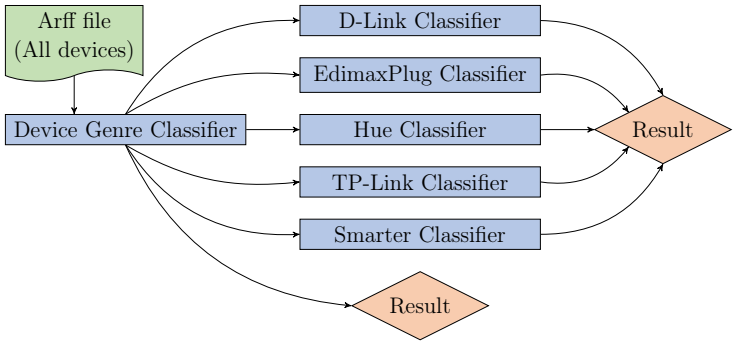


Figura 2.5: Clasificador de dos niveles [2]

Los resultados que obtiene son de un 82 % de accuracy promedio entre todos los clasificadores de cada proveedor. Los valores individuales están entre el 42.2 % y el 100 %.

El siguiente trabajo que se verá es el realizado por Hossein Jafari et al. [14]. En este artículo no se utilizan paquetes de la capa de red o transporte, sino de la capa física. Para realizar esta labor se obtendrá la huella de cada dispositivo mediante radio frecuencias, por esta razón, este estudio sólo se centrará en conexiones inalámbricas, en este caso 6 dispositivos ZigBee [9].

Para cada uno de los dispositivos se realiza una captura de 5 minutos. Esta captura será del SNR en 5 niveles distintos, con lo que se obtiene 10 GB por dispositivo y nivel de SNR, lo que en total serían 300 GB de datos.

A continuación procede a entrenar tres modelos de deep learning, DNN, CNN y LSTM (los tres modelos pertenecen a la librería `TensorFlow` programada para Python [1]), con los que obtiene unos resultados de 96.3 %, 94.7 % y 76 % respectivamente.

Fabian Lanze et al. [15] han desarrollado un trabajo en el que crean un sistema de identificación de dispositivos basado en el clock skew. Su objetivo es generar huellas para cada punto de acceso, de forma que si un dispositivo pueda saber si ese punto de acceso es legítimo, o por contra si puede ser un potencial atacante.

Lo primero que hacen es definir el skew de un reloj como la primera derivada del offset de dicho reloj. De forma práctica usarán la desviación que almacena el demonio del servicio NTP para aproximar el skew del dispositivo conectado al punto de acceso. Usará conexión inalámbrica y recogerá hasta 6000 paquetes por dispositivo en cada muestra (10 minutos). Si en una muestra no se alcanzan los 500 paquetes, por problemas de conexión o cualquier otro motivo, esa muestra se elimina. Con este proceso se consiguen medir 388 puntos de acceso.

Finalmente, trata de generar una huella para cada punto de acceso de forma estadística, basándose sobre todo en la distribución del clock skew. Como resultado obtiene que su método no es suficientemente preciso y que muchos dispositivos presentan una huella similar, por lo tanto, concluye que no es posible identificarlos únicamente de esta forma.

En el trabajo desarrollado por Yair Meidan et al. [16] se desarrolla un sistema que es capaz de identificar dispositivos IoT a través de su tráfico de red. Las pruebas se realizan con trazas sobre conexión inalámbrica y tanto con dispositivos IoT como con PCs, smartphones, etc.

De estas trazas se centran específicamente en las sesiones TCP, donde cada una se representa como un conjunto de características de cada paquete.

Primeramente, ejecutan un clasificador binario con el que obtienen la probabilidad de que un dispositivo sea IoT, usando solo información de una sesión TCP de cada dispositivo. En este apartado obtienen un 100 % de acierto.

Por último, entrenan distintos modelos de clasificación (GBM, Random Forest, XGBoost), únicamente sobre los dispositivos previamente etiquetados como IoT, obteniendo finalmente un resultado de un 99.281 % de accuracy en la identificación del modelo y marca de cada uno de ellos.

Como último artículo se analizará la propuesta realizada por Loh Chin Choong Desmond et al. [7]. En este trabajo han construido un sistema de identificación de dispositivos basado en los mensajes *probe request* de los distintos dispositivos. Para ello, distinguirán como dispositivos distintos las distintas combinaciones de la tupla: ⟨"Machine", "Wireless Network Interface Card (NIC) Driver", "Operating System"⟩.

A continuación, comprueban que en el tiempo de emisión de estos mensajes se ven varios comportamientos que se repiten. Luego, observan que la latencia entre ráfagas de estos mensajes también presenta estos comportamientos distintivos. Con estas latencias son con las que entrenarán un modelo que sea capaz de distinguir los dispositivos.

El modelo que eligen se basa en aprendizaje no supervisado, Maximum Variance Clustering [29]. Este modelo es similar al algoritmo de k-medias, la distinción es que k-medias necesita conocer previamente el número de clústeres k previamente, mientras que Maximum Variance Clustering necesita conocer la varianza máxima dentro de un clúster σ^2_{max}. Con este algoritmo obtienen unos resultados de accuracy entre el 70 % y el 80 %.

2.3.1 Resultados de la revisión bibliográfica

Por último se realizará un resumen en forma de tabla de los artículos que han sido revisados.

Referencia	Tipo de identificación	Fuente de los datos	Método de clasificación	Tipo de aprendizaje	Algoritmos usados	Resultados
[19]	Tipo de dispositivo	Cabeceras TCP	Machine Learning	Supervisado	MLP, SVM, Random Forest	99.76 % de accuracy y 97.03 % de precisión
[12]	Individual	Características del payload	Machine Learning	Supervisado	AdaBoost, LDA, KNN, Árboles de decisión, Naive Bayes, SVM, Random Forest y GBoost	89 % de accuracy
[2]	Tipo y modelo del dispositivo	Cabereceras TCP	Machine Learning	Supervisado	Tablas de decisión, Árboles de decisión J48, OneR y PART	Entre 42.2 % y 100 % de accuracy, con un promedio de 82 %
[14]	Individual	Radio frecuencias	Machine Learning	Supervisado	DNN, CNN y LSTM	96.3 % de accuracy en DNN, 94.7 % de accuracy en CNN y 76 % de accuracy en LSTM
[15]	Modelo del dispositivo	Registros NTP	Estadístico	-	-	Inconcluyentes
[16]	Tipo y modelo	Sesiones TCP	Machine Learning	Supervisado	GBM, Random Forest y XGBoost	99.281 % de accuracy
[7]	Individual	Mensajes *probe request*	Machine Learning	No supervisado	Maximum Variance Clustering	Entre un 70 % y 80 % de accuracy

Tabla 2.1: Resultados en el estado del arte

CAPÍTULO 3

Análisis de objetivos y metodología

En esta sección se analizarán los objetivos del trabajo y se realizará un análisis sobre las decisiones tomadas en el proceso de identificar dispositivos.

3.1 Análisis de objetivos

El principal objetivo de este proyecto es conseguir identificar dispositivos idénticos. La solución que se propone es capaz de identificar los patrones relativos a la desviación de sus relojes internos y usarlos para reconocer a cada dispositivo mediante un modelo de Machine Learning. Para lograr dicho objetivo han sido establecidas diversas metas intermedias:

- Usar lo visto en el estado del arte, para conocer y seleccionar los desafíos de este proyecto.
- Conseguir un registro de los relojes internos de cada uno de los dispositivos que se van a participar en el análisis. Para lo que definiremos una estructura cliente-servidor que recogerá un registro de los datos.
- Generar una huella estadística de estos registros. Para desarrollar este objetivo se hace uso de una ventana deslizante que permite agrupar los datos y extraer estadísticas comunes.
- Analizar distintas propuestas de algoritmos de Machine Learning que sean capaces de tomar como entrada las huellas estadísticas y, con ellas, identificar a cada dispositivo.

3.2 Metodología de trabajo

Para poder alcanzar los objetivos propuestos, se presenta una metodología de trabajo.

1. **Revisión bibliográfica**. Revisión de los trabajos relacionados con el tema de este trabajo. En concreto, se busca comprender cómo obtienen sus datos y su método de clasificación de dispositivos.
2. **Recolección de los datos**. Generación de un dataset que contiene las marcas de tiempo tanto del dispositivo de referencia como del que se encuentra bajo análisis. Este proceso se realiza por cada dispositivo a analizar.
3. **Generación de huella estadística**. Por cada dispositivo se hace uso de una ventana deslizante que agrupa los datos en grupos de n muestras. De cada grupo se obtienen un conjunto de variables estadísticas que son etiquetadas con el dispositivo del que se han obtenido.
4. **Búsqueda de variables correlacionadas**. Se realiza un test de correlación a los datos con el fin de reducir la dimensionalidad de los mismos. Las variables correlacionadas no aportan información y por ello han de ser eliminadas.
5. **Particionamiento del conjunto de datos**. Con el fin de entrenar algoritmos de Machine Learning, se divide el conjunto de datos en dos de menor tamaño (entrenamiento y test).
6. **Ajuste de hiperparámetros**. Para decidir cuál de los algoritmos es mejor para clasificar los datos de este trabajo se comparan distintos algoritmos de Machine Learning. Estos algoritmos han de ser ajustados correctamente para obtener los mejores resultados. Para realizar este ajuste se realiza una partición similar a la realizada con el conjunto de todos los datos, pero con un conjunto reducido de forma que los entrenamientos duren menos tiempo.
7. **Evaluación de los distintos modelos**. Una vez entrenados todos los algoritmos que se quieren comparar, se comparan sus resultados identificando dispositivos para elegir al mejor candidato.
8. **Entrenamiento del modelo final**. Decididos el algoritmo que se va a usar y sus hiperparámetros, se realiza un entrenamiento final con el conjunto completo de los datos de entrenamiento.
9. **Evaluación del modelo final**. Análisis de los resultados finales obtenidos.

CAPÍTULO 4

Diseño y resolución

En este capítulo se realiza la descripción de los experimentos realizados, su análisis y la búsqueda de un modelo de Machine Learning, así como los resultados finales del trabajo.

En este trabajo se usará una topología de red como la que ilustra la Fig. 4.1. En dicha topología se puede ver que tenemos un observador (modelo RockPro64) y 5 dispositivos a analizar (Raspberry Pi 4 Model B).

La idea es mandar mensajes desde el observador a cada uno de los dispositivos y que cada uno de los dispositivos conteste a esos mensajes y obteniendo así una marca de tiempo de ese dispositivo.

4.1 Elección del protocolo

Las opciones barajadas han sido TCP e ICMP. En un principio se pensaba usar las marcas de tiempo contenidas en las cabeceras de estos protocolos, pero esta idea se descartó puesto que únicamente permitían obtener los tiempos en milisegundos. El objetivo era obtener los tiempos en nanosegundos por tanto, se enviaron los datos en el cuerpo del paquete que no tiene una limitación tan corta de espacio.

Con el fin de mantener la conexión persistente en toda la muestra se escogió el protocolo TCP, ya que una vez establecida la conexión se mantendrá hasta el final.

4.2 Obtención de datos

Para obtener los datos se usará una estructura cliente-servidor. Los dispositivos a analizar serán los que tomen el rol de servidores y el observador será el cliente.

Con esta idea en mente se crea un programa servidor que escuche a cualquier dirección IP en un determinado puerto y responda con una marca de tiempo en nanosegundos. Este

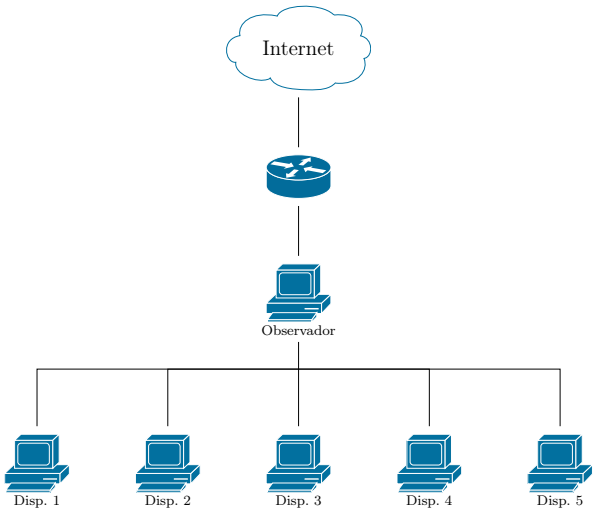


Figura 4.1: Topología de la red

time	TSrock	TSrasp	offset	device
292	119238112796030	104592709716803	-14645403079227	192.168.1.111
1001191222	119239113986960	104593710167425	-14645403819535	192.168.1.111
2001485862	119240114281600	104594710453699	-14645403827901	192.168.1.111
⋮	⋮	⋮	⋮	⋮

Tabla 4.1: Ejemplo de los datos obtenidos de cada dispositivo

programa se ejecutará en cada uno de los dispositivos bajo análisis.

A la hora de capturar los datos se usó un reloj interno que no debería sufrir alteraciones que disminuyeran su valor (`steady_clock`[27]).

Una vez los dispositivos estén todos escuchando, el observador ejecutará un programa cliente que es el que se encargará de enviar los mensajes al servidor correspondiente. Este programa guarda una marca de tiempo al comienzo de la ejecución t_{start}, que será nuestro punto de referencia. Después mandará n mensajes equiespaciados en intervalos de 1 segundo.

En cada ejecución del bucle se obtendrá una marca de tiempo en el observador t_i, y una marca de tiempo del dispositivo t'_i. Con esto se consiguen tener varios datos:

- La marca de tiempo relativa a cada mensaje desde el inicio, $t_i - t_{start}$. Teóricamente debería de dar valores exactos, ya que se manda un mensaje cada segundo, pero existe cierto retraso.
- La marca de tiempo absoluta del observador t_i.
- La marca de tiempo absoluta del dispositivo t'_i.
- La desviación del reloj del dispositivo respecto al del observador, $t_i - t'_i$.

Un ejemplo de como se guardan estos datos se puede ver en la Tabla 4.1.

Este proceso se realizará para cada dispositivo tanto en una muestra secuencial, como en una muestra en paralelo.

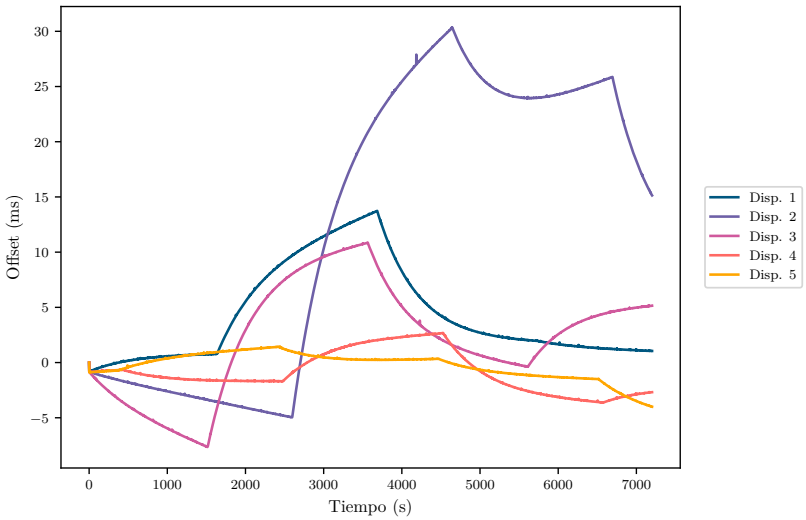
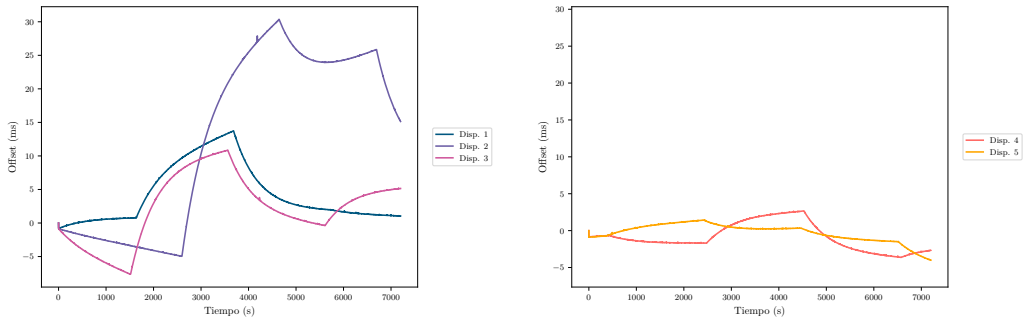


Figura 4.2: Offset acumulado muestra secuencial



(a) Dispositivos 1, 2 y 3

(b) Dispositivos 4 y 5

Figura 4.3: Diferencias entre offsets de dispositivos

4.3 Análisis de los datos

En esta sección se estudiará el incremento de la desviación del reloj en cada una de las muestras, así como la desviación acumulada en cada punto.

4.3.1 Experimento 1: Muestra secuencial

En este primer experimento se ha tomado una muestra de 7200 segundos, es decir, 2 horas por dispositivo, lo que en total suman 10 horas de muestras.

En la Fig. 4.2 se muestra la desviación (offset) acumulada en cada dispositivo durante esas 2 horas.

Como se puede ver hay 2 tipos de comportamiento. Los dispositivos 1, 2 y 3 se mantienen monótonos al comienzo para después incrementar mucho su desviación. Por contra los dispositivos 4 y 5 se mantienen sin grandes cambios en toda la muestra. Esto se puede ver más claramente en la Fig. 4.3.

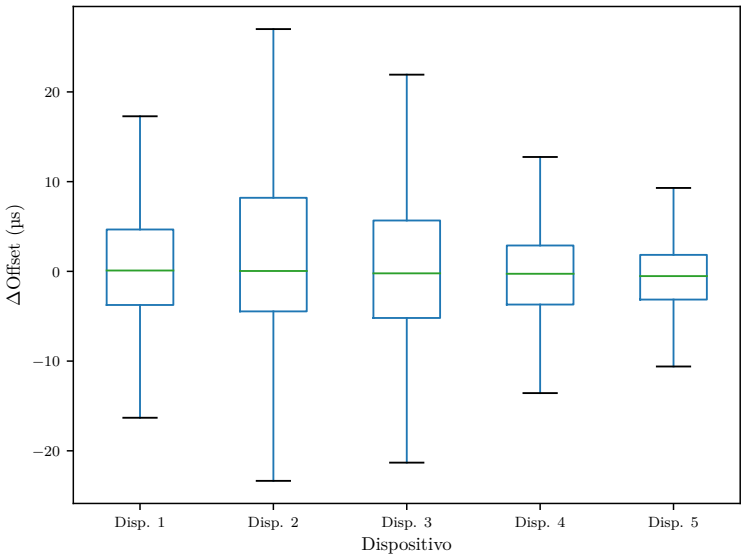


Figura 4.4: Diagrama de cajas muestra secuencial

Lo que interesa al final es obtener una forma de distinguir estadísticamente los datos, para ello se genera un gráfico que muestre entre que valores se mueven estos datos en cada dispositivo, en definitiva, se puede ver con un diagrama de cajas (Fig. 4.4). Se han eliminado los valores atípicos ya que a tan pequeña escala no dejan ver los verdaderos resultados.

Este diagrama ilustra lo comentado anteriormente. Los dispositivos 4 y 5 son los que menos varían, esto se puede ver en que las cajas son pequeñas y la mediana está centrada entre los cuartiles (Q1 y Q3).

En los otros 3 dispositivos se observa como entre la mediana y el tercer cuartil hay más espacio que entre la mediana y el primer cuartil. Esto se debe a que tienen muchos incrementos grandes, por eso crecen tan bruscamente.

Con esto se tiene una idea de los dispositivos son distinguibles estadísticamente, lo que confirma que se podrá entrenar un modelo que los identifique.

4.3.2 Experimento 2: Muestra paralela

Para el segundo experimento se han capturado datos de todos los dispositivos simultáneamente durante un periodo de 43 200 segundos, es decir, 12 horas.

La Fig. 4.5 muestra la desviación acumulada en cada dispositivo en este periodo de tiempo. En este gráfico también se han marcado ciertos puntos en los que los dispositivos parecen "sincronizarse". Con esto nos referimos a que alrededor de estas marcas de tiempo todos los dispositivos cambian de tendencia bruscamente y además parecen repetirse con cierta periodicidad. Las marcas de tiempo de este gráfico corresponden a los puntos $\{4000, 16\,500, 29\,000, 41\,000\}$ que están aproximadamente equidistantes (aproximadamente 12 000 segundos).

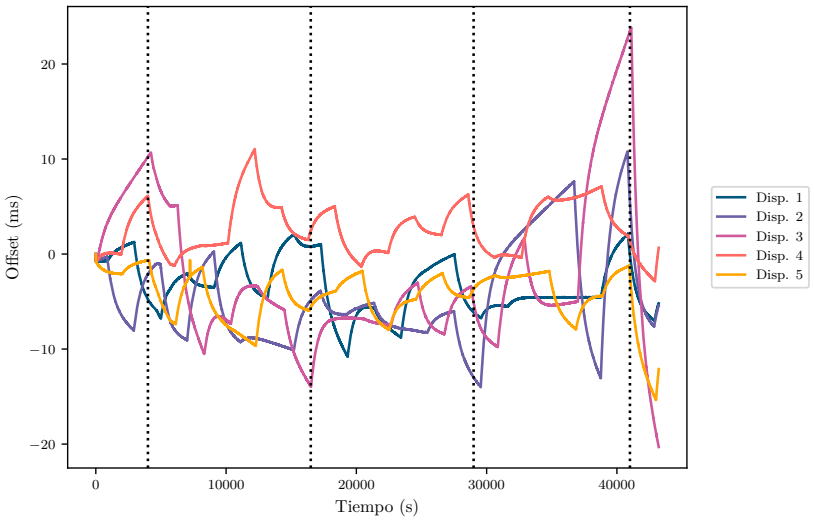


Figura 4.5: Offset acumulado muestra paralela

Esta "sincronización" entre dispositivos parece deberse a algún tipo de actualización del reloj interno del dispositivo mediante algún demonio del sistema operativo. El principal demonio que se encarga de esta tarea es el servicio NTP, el cual fue desactivado para realizar estas pruebas.

Consultando otro diagrama de cajas con el incremento de la desviación en cada instante (Fig. 4.6) se observa como todos los dispositivos están más a la par que en la muestra secuencial (Fig. 4.4). Aún así, siguen teniendo ciertas diferencias, con lo que se puede concluir que también son estadísticamente diferenciables.

4.4 Elección de la muestra de datos

Analizando los incrementos de la desviación en ambas muestras se puede ver que la muestra paralela tiene valores más parecidos a los esperados. Estos valores de incremento de la desviación son más pequeños y más parecidos entre todos los dispositivos, es por ello que se entrenarán los algoritmos con los datos de la muestra paralela.

Estos datos se obtendrán mediante una ventana deslizante de 1 minuto sobre el incremento de la desviación del reloj del dispositivo. Se obtendrán las siguientes variables estadísticas:

- Suma
- Media
- Mediana
- Moda
- Desviación típica
- Rango intercuartílico
- Curtosis
- Coeficiente de asimetría (skewness)
- Máximo
- Mínimo

Generando estos datos estadísticos de la muestra paralela mediante la ventana deslizante,

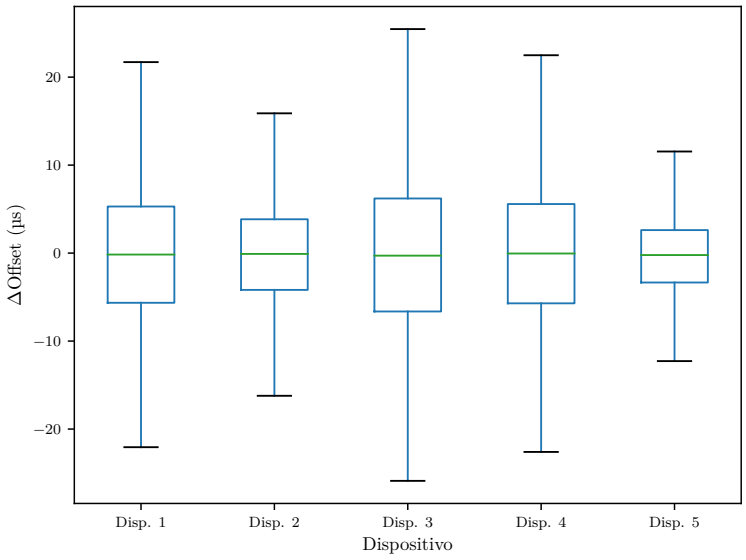


Figura 4.6: Diagrama de cajas muestra paralela

	Sum	Mean	Median	Mode	Std	IQR	Kurtosis	Skew	Max	Min	Device
1	21773.0	362.8833333333333	-306.5	-17885.0	8212.584818005707	12095.5	0.0567239505757037	0.4351150677774124	20799.0	-17885.0	Disp. 1
2	4059.0	67.65	239.5	-15862.0	5615.809629134339	3012.25	1.400480680742287	-0.1242668154207108	14194.0	-15862.0	Disp. 2
3	1187.0	19.78333333333333	503.5	-25009.0	7760.832728977938	8467.0	1.231550787741679	-0.2621794660161829	20510.0	-25009.0	Disp. 3
4	74154.0	1235.9	2016.5	-21616.0	8391.731753359314	9663.75	0.4104874852336051	-0.4087239664161064	19992.0	-21616.0	Disp. 4
5	-127078.0	-2117.9666666666667	-2385.0	-11894.0	4354.015072678016	2737.0	1.9665757630194	0.6070065306393123	10383.0	-11894.0	Disp. 5
6	20147.0	335.78333333333336	-306.5	-17885.0	8244.497450208664	12095.5	0.0362682762350909	0.4263535051366743	20799.0	-17885.0	Disp. 1
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Tabla 4.2: Datos estadísticos muestra paralela

se obtienen los resultados que se pueden ver en la Tabla 4.2. Estos datos estadísticos serán los que posteriormente se usarán para entrenar un modelo de Machine Learning.

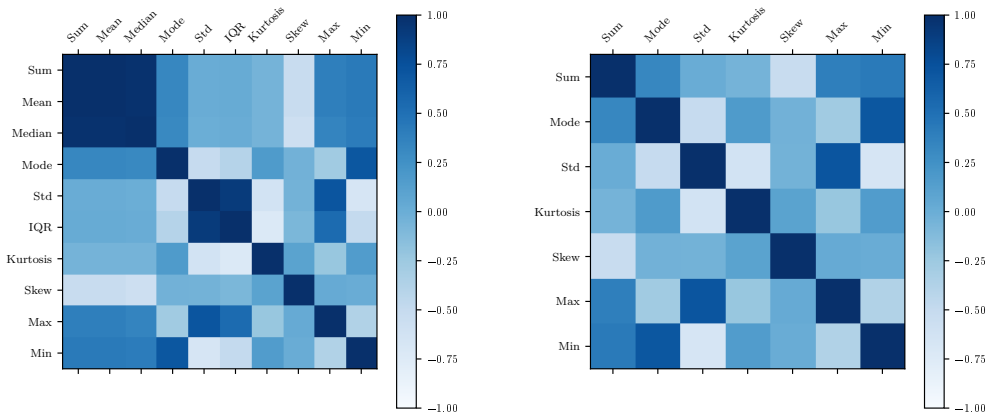
4.5 Reducción de la dimensionalidad

En esta sección buscaremos valores estadísticos correlacionados entre sí. En caso de existir habrá que eliminarlos antes de entrenar los modelos, puesto que no aportan información y provocarán que el algoritmo tarde aún más tiempo en ser entrenado.

La correlación entre las variables estadísticas se puede ver en la Fig. 4.7. Se puede ver como suma, media y mediana son las variables más correlacionadas entre sí, por tanto serán eliminadas 2 de ellas.

4.6 Entrenamiento de los modelos

En esta sección se analizarán los resultados que hemos obtenido de los modelos sobre los datos de la muestra paralela, como hemos concluido en la sección anterior.



(a) Correlación entre las variables iniciales (b) Correlación entre las variables finales

Figura 4.7: Correlación entre las variables estadísticas

Algoritmo	Implementación
Árboles de decisión	`DecisionTreeClassifier` [20]
Random Forest	`RandomForestClassifier` [26]
MLP	`MLPClassifier` [25]
Naive Bayes	`GaussianNB` [21]
KNN	`KNeighborsClassifier` [23]
SVM	`LinearSVC` [24]

Tabla 4.3: Equivalencia entre algoritmo e implementación

Para realizar este entrenamiento se ha usado la librería `scikit-learn` sobre el lenguaje de programación Python. Esta librería tiene clases que implementan los algoritmos vistos anteriormente que serán los que se usarán (Sec. 2.2.3). Las clases que corresponden a cada modelo son las que se ven en la Tabla 4.3.

A la hora de entrenar modelos de machine learning lo primero que hay hacer es ajustar sus hiperparámetros. Los hiperparámetros de un modelo son parámetros del mismo que se pueden modificar para que este se adapte mejor a los datos con los que se está trabajando o para cambiar la forma en la aprende este modelo también para adaptarse a estos datos.

Para ajustar los hiperparámetros no se usa la completitud de los datos, se usa un modelo entrenamiento/validación/test. Los datos se han dividido en un conjunto de entrenamiento y otro de test, 70 % para entrenamiento y 30 % para test, lo habitual es tener en torno a una tercera parte de los datos para entrenamiento y el resto para test. Este proceso (Fig. 4.8) se hace para comprobar la capacidad de generalización del modelo con datos que no haya visto nunca.

Para particionar los datos se tomarán muestras aleatorias, pero ordenadas para mantener la correlación temporal de los datos. Por ejemplo, ante el conjunto de datos $x_1, \ldots, x_{20}$, una muestra aleatoria podría ser $x_{15}, x_8, x_3, x_{17}, x_5, x_{12}$ pero si estos datos presentan una

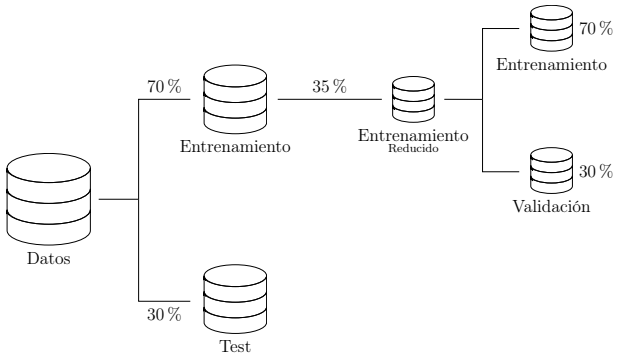


Figura 4.8: Particiones de los datos

correlación temporal, como es el caso de este trabajo, esta se pierde. Por tanto, los datos han de ser devueltos a su orden natural $x_3, x_5, x_8, x_{12}, x_{15}, x_{17}$.

El conjunto de entrenamiento se usará únicamente con el modelo final y con los hiperparámetros ya ajustados, puesto que contiene una gran cantidad de datos. Para ajustar los hiperparámetros de cada modelo y comparar los modelos entre sí se usará un conjunto reducido de este, un 35 % de los datos de entrenamiento.

Este subconjunto de los datos de entrenamiento también será dividido en un conjunto de entrenamiento/test, aunque en este caso el conjunto test recibirá el nombre de conjunto de validación. Se entrenarán los distintos algoritmos con estos conjuntos con el fin de ajustar los hiperparámetros y decidir el mejor entre todos ellos.

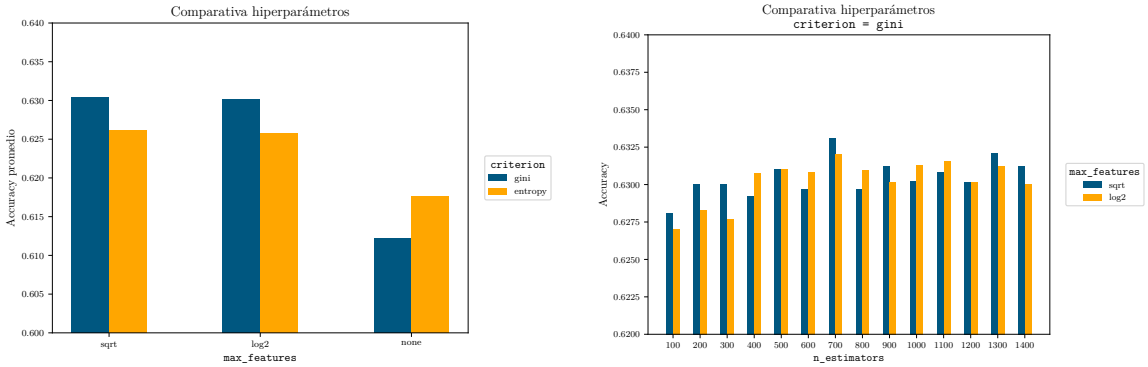
Para comprobar qué hiperparámetros son los que mejor se ajustan a los datos, se usará la función `GridSearchCV` [22] que permite que dados un algoritmo y un conjunto de valores para los hiperparámetros probar todas las combinaciones y obtener así un accuracy de cada uno, con lo que se pueden comparar y filtrar únicamente a los mejores. Esta función permite usar la validación cruzada. La validación cruzada

A continuación se mostrará un ejemplo del proceso de ajustar los hiperparámetros de un modelo, en este caso se usará el algoritmo de Random Forest sobre los datos la muestra paralela. Los hiperparámetros que se han decido ajustar son:

- `criterion`: función que mide la pureza de los nodos hijos.
- `max_features`: número de características que se tienen en cuenta para realizar la división de un nodo.
- `n_estimators`: número de árboles de decisión que participan en el algoritmo.

Para evitar que una rama entre en un bucle infinito debido a que no es capaz de dividir correctamente los nodos se ha fijado la profundidad máxima de cada rama (hiperpárametro `max_depth`) a un valor de 1000.

Hablando en primer lugar sobre los hiperparámetros `criterion` y `max_features` (Fig. 4.9a) se puede ver que en promedio el mejor valor de `criterion` es `gini`. Este parámetro será fijado y, a continuación, se comprobarán todos los pares de valores que generan los hiperparámetros `max_features` y `n_estimators` (Fig. 4.9b).



(a) (b)

Figura 4.9: Comparativa hiperparámetros Random Forest

Los mejores valores obtenidos son `criterion = gini`, `n_estimators = 700` y `max_features = sqrt`. Con estos hiperparámetros se entrenará un modelo sobre el conjunto de entrenamiento menor y después se realizará una predicción con el conjunto de validación.

CAPÍTULO 5

Resultados

Una vez entrenados todos los algoritmos con sus mejores hiperparámetros y con un valor de generalización obtenido de la predicción, tenemos los resultados que se pueden ver en la Fig. 5.1.

Como se puede ver en ambas muestras los modelos que presentan mejores resultados son los que se basan en árboles de decisión, que son el propio algoritmo de árboles de decisión y random forest. Otra forma de ver estos resultados es mediante las matrices de confusión que nos genera cada algoritmo.

Es fácil ver en la Fig. 5.2 que los modelos basados en árboles aciertan prácticamente en la totalidad de las ocasiones, en particular, el algoritmo de random forest es el que mejores resultados consigue ($\sim 99.5\,\%$). Por esta razón se entrenará un modelo de random forest con los hiperparámetros ajustados anteriormente con la totalidad de los datos de entrenamiento. Los resultados obtenidos se pueden ver en la matriz de confusión resultante (Fig. 5.3). Con estos resultados se tiene un valor final de accuracy de $99.44\,\%$.

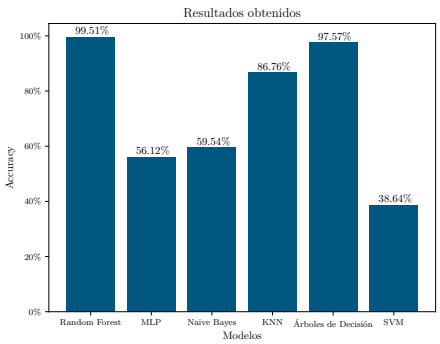


Figura 5.1: Comparativa de resultados entre modelos

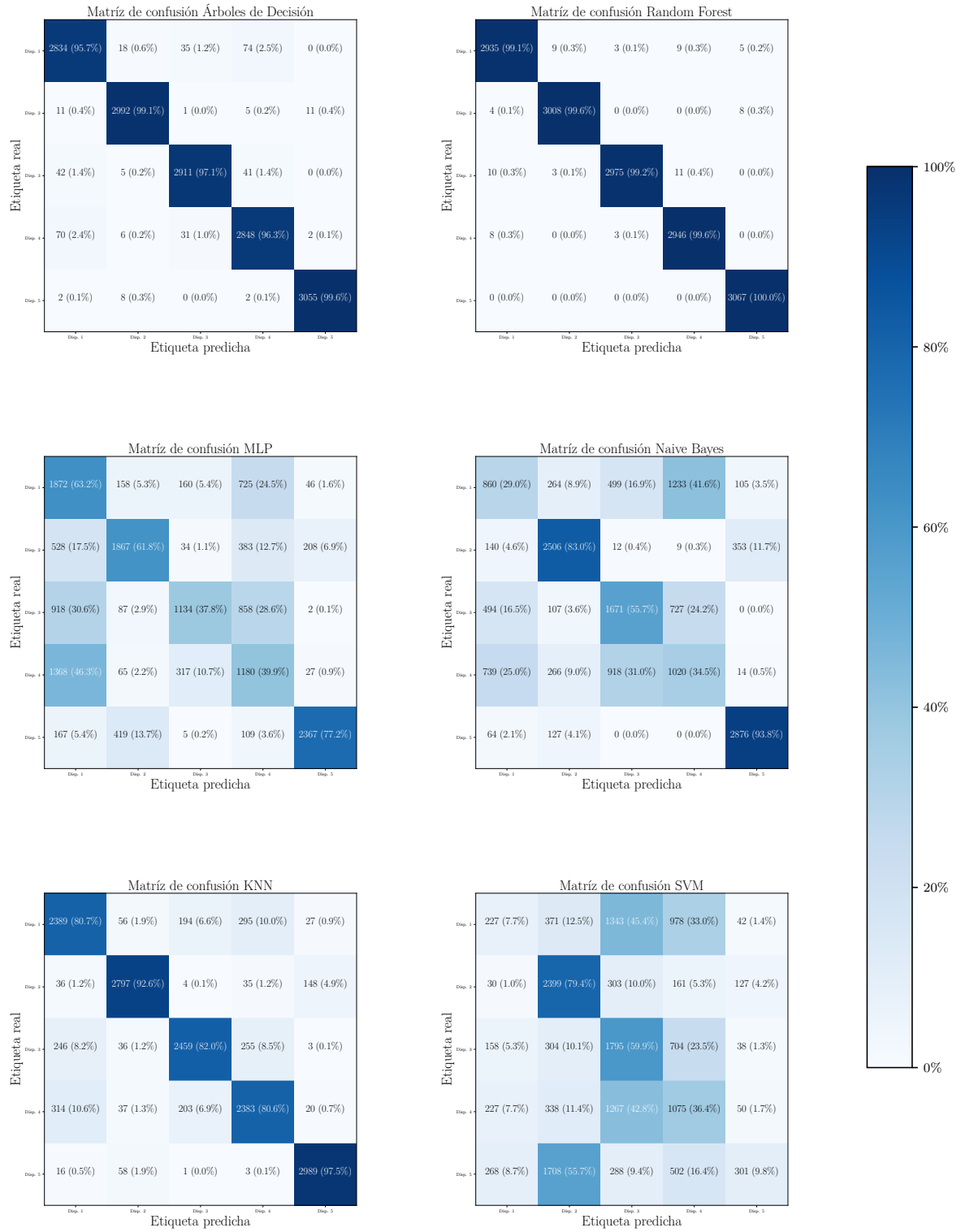


Figura 5.2: Matrices de confusión con datos de la muestra paralela

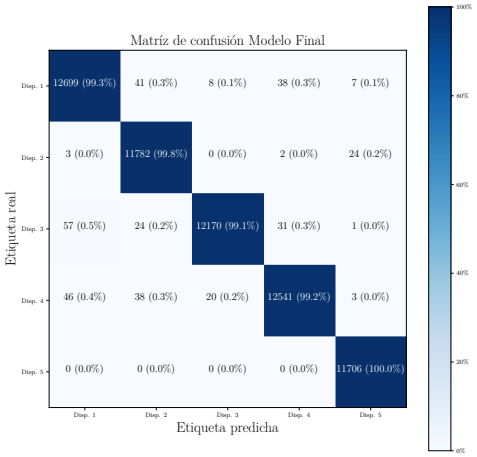


Figura 5.3: Matríz de confusión del modelo final

CAPÍTULO 6

Conclusiones y vías futuras

En este proyecto se ha diseñado un modelo capaz de clasificar dispositivos con los que se ha establecido una comunicación en base a pequeñas diferencias en la fabricación de los componentes, que alteran el tiempo que tardan en ejecutarse ciertas tarea.

En la primera parte del proyecto se ha visto que para obtener la precisión en los tiempos deseada, ha sido necesario el uso del protocolo TCP y el envío de las marcas de tiempo en el cuerpo del paquete.

Durante el desarrollo de esta parte de la investigación se apreció que el reloj interno es susceptible de ser alterado por procesos externos con el cometido de que esté sincronizado en todo momento con el resto de los dispositivos (protocolo NTP). Por este motivo este servicio tuvo que ser desactivado antes de realizar ninguna recolección de datos, pues los diferencias entre tiempos no serían las propias de cada dispositivo.

Una vez desactivado el servicio, aún se obtenían datos que no eran correctos debido a que se estaba usando un reloj del sistema que podía ser modificado. Este reloj fue cambiado por un reloj que no fuera modificable (`steady_clock`) y con eso los datos fueron más precisos.

Se realizaron capturas tanto en secuencial como en paralelo de la desviación de los relojes de los dispositivos, de las cuales se obtuvieron sus incrementos en cada momento. Con estos incrementos y una ventana deslizante de 1 minuto se obtienen variables estadísticas que servirán para entrenar los modelos.

Después de ver los resultados de los modelos con los conjuntos de entrenamiento/validación y analizando los posibles usos del sistema implementado se considera que es mejor quedarse con la muestra paralela.

Por último, se entrenó el algoritmo elegido, Random Forest, con los datos de la muestra paralela y los hiperparámetros que se consideraron mejores cuando se realizó el entrenamiento los conjuntos de entrenamiento/validación. De este modelo se obtuvieron unos resultados finales de 99.44 % en el valor de accuracy.

Una de las posibles vías futuras de este trabajo sería el desarrollo de un modelo a tiempo

real de este sistema. Para este cometido se debería tener una copia local de las huellas que generan ciertos dispositivos para poder compararlos con los que estamos recibiendo en ese momento y así comprobar si se trata de un atacante.

Para realizar este sistema a tiempo real también habría que crear mecanismos que permitan al sistema actualizarse con nuevos datos, y con ello generar nuevas huellas para los dispositivos. También habría que modificar los modelos de Machine Learning debido a que para que el sistema funcione a tiempo real, estos deberían actualizarse.

Bibliografía

[1] M. Abadi *et al.*, *TensorFlow: Large-scale machine learning on heterogeneous systems*, Software available from tensorflow.org, 2015. [Online]. Available: https://www.tensorflow.org/.

[2] A. Aksoy and M. H. Gunes, "Automated iot device identification using network traffic", in *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, 2019, pp. 1–7.

[3] D. A. Borman, R. T. Braden, and V. Jacobson, "TCP Extensions for High Performance", Tech. Rep. 1323, May 1992, 37 pp. [Online]. Available: https://rfc-editor.org/rfc/rfc1323.txt.

[4] L. Breiman, "Random forests", *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[5] W. Commons. "File:knnclassification". (2010), [Online]. Available: https://commons.wikimedia.org/wiki/File:KnnClassification.svg.

[6] W. Commons. "Svm separating hyperplanes (svg)". (2012), [Online]. Available: https://commons.wikimedia.org/wiki/File:Svm_separating_hyperplanes_(SVG).svg.

[7] L. C. C. Desmond, C. C. Yuan, T. C. Pheng, and R. S. Lee, "Identifying unique devices through wireless fingerprinting", in *Proceedings of the First ACM Conference on Wireless Network Security*, ser. WiSec '08, Alexandria, VA, USA: Association for Computing Machinery, 2008, pp. 46–55, ISBN: 9781595938145. [Online]. Available: https://doi.org/10.1145/1352533.1352542.

[8] M. U. Farooq, M. Waseem, S. Mazhar, A. Khairi, and T. Kamal, "A review on internet of things (iot)", *International journal of computer applications*, vol. 113, no. 1, pp. 1–7,

[9] D. Gislason, *Zigbee Wireless Networking*, Pap/Onl. USA: Newnes, 2008, ISBN: 978-0-7506-8597-9.

[10] P. Gokhale, O. Bhat, and S. Bhat, "Introduction to iot", *International Advanced Research Journal in Science, Engineering and Technology*, vol. 5, no. 1, pp. 41–44, 2018.

[11] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: An update", *SIGKDD Explor. Newsl.*, vol. 11, no. 1, pp. 10–18, 2009, ISSN: 1931-0145. [Online]. Available: https://doi.org/10.1145/1656274.1656278.

[12] S. A. Hamad, W. E. Zhang, Q. Z. Sheng, and S. Nepal, "Iot device identification via network-flow based fingerprinting and learning", in *2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, IEEE, 2019, pp. 103–111.

[13] J. Hatwell, M. M. Gaber, and R. M. A. Azad, "Chirps: Explaining random forest classification", *Artificial Intelligence Review*, vol. 53, pp. 5747–5788, 2020.

[14] H. Jafari, O. Omotere, D. Adesina, H.-H. Wu, and L. Qian, "Iot devices fingerprinting using deep learning", in *MILCOM 2018 - 2018 IEEE Military Communications Conference (MILCOM)*, 2018, pp. 1–9.

[15] F. Lanze, A. Panchenko, B. Braatz, and A. Zinnen, "Clock skew based remote device fingerprinting demystified", in *2012 IEEE Global Communications Conference (GLOBECOM)*, 2012, pp. 813–819.

[16] Y. Meidan *et al.*, "Profiliot: A machine learning approach for iot device identification based on network traffic analysis", in *Proceedings of the Symposium on Applied Computing*, ser. SAC '17, Marrakech, Morocco: Association for Computing Machinery, 2017, pp. 506–509, ISBN: 9781450344869. [Online]. Available: https://doi.org/10.1145/3019612.3019878.

[17] R. van der Meulen, *8.4 billion connected things will be in use 2017 | gartner*, https://www.gartner.com/en/newsroom/press-releases/2017-02-07-gartner-says-8-billion-connected-things-will-be-in-use-in-2017-up-31-percent-from-2016, 2017.

[18] M. Miettinen, S. Marchal, I. Hafeez, N. Asokan, A.-R. Sadeghi, and S. Tarkoma, "Iot sentinel: Automated device-type identification for security enforcement in iot", *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, 2017. [Online]. Available: http://dx.doi.org/10.1109/ICDCS.2017.283.

[19] P. Oser, F. Kargl, and S. Lüders, "Identifying devices of the internet of things using machine learning on clock characteristics", in *Security, Privacy, and Anonymity in Computation, Communication, and Storage*, G. Wang, J. Chen, and L. T. Yang, Eds., Cham: Springer International Publishing, 2018, pp. 417–427, ISBN: 978-3-030-05345-1.

[20] "DecisionTreeClassifier", [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier.

[21] "GaussianNB", [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html#sklearn.naive_bayes.GaussianNB.

[22] "GridSearchCV", [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html.

[23] "KNeighborsClassifier", [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html.

[24] "LinearSVC", [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html.

[25] "MLPClassifier", [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html#sklearn.neural_network.MLPClassifier.

[26] "RandomForestClassifier", [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html#sklearn.ensemble.RandomForestClassifier.

[27] "std::chrono::steady_clock", [Online]. Available: https://en.cppreference.com/w/cpp/chrono/steady_clock.

[28] "The "only" coke machine on the internet". (1982), [Online]. Available: https://www.cs.cmu.edu/~coke/history_long.txt.

[29] C. Veenman, M. Reinders, and E. Backer, "A maximum variance cluster algorithm", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 9, pp. 1273–1280, 2002.

[30] I. Yaqoob *et al.*, "Internet of things architecture: Recent advances, taxonomy, requirements, and open challenges", *IEEE Wireless Communications*, vol. 24, no. 3, pp. 10–16, 2017, ISSN: 1558-0687.