



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

2 de junio 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

Declaración firmada sobre la originalidad del trabajo	4
Resumen	5
Extended abstract	7
1 Introducción	11
2 Estado del arte	13
3 Análisis de objetivos y metodología	17
3.1 Objetivos	17
3.2 Algoritmos de Machine Learning	17
3.2.1 Tratamiento de los datos	18
3.2.2 Algoritmos de aprendizaje supervisado	18
4 Diseño y resolución	22
4.1 Elección del protocolo	22
4.2 Obtención de datos	22
4.3 Análisis de los datos	24
4.3.1 Experimento 1: Muestra secuencial	24
4.3.2 Experimento 2: Muestra paralela	25
4.4 Elección de la muestra de datos	26
4.5 Reducción de la dimensionalidad	27
4.6 Entrenamiento de los modelos	27
5 Conclusiones y vías futuras	33
Bibliografía	35

Índice de figuras

2.1	Desbordamiento del contador [1]	14
2.2	Modelo del sistema de identificación [2]	15
2.3	Clasificador de dos niveles [3]	16
3.1	Funcionamiento del algoritmo KNN [12]	20
3.2	Separación por hiperplanos SVM [13]	21
4.1	Topología de la red	23
4.2	Offset acumulado muestra secuencial	24
4.3	Diferencias entre offsets de dispositivos	24
4.4	Diagrama de cajas muestra secuencial	25
4.5	Offset acumulado muestra paralela	26
4.6	Diagrama de cajas muestra paralela	27
4.7	Correlación entre las variables estadísticas	28
4.8	Particiones de los datos	29
4.9	Comparativa hiperparámetros Random Forest	30
4.10	Comparativa de resultados entre modelos	30
4.11	Matrices de confusión con datos de la muestra paralela	31
4.12	Matriz de confusión del modelo final	32

Índice de tablas

2.1	Resultados en el estado del arte	16
4.1	Ejemplo de los datos obtenidos de cada dispositivo	23
4.2	Datos estadísticos muestra paralela	27
4.3	Equivalencia entre algoritmo e implementación	28

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 de los 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 2 de junio de 2022

Fdo.: Sergio Marín Sánchez
Autor del TF

Resumen

En este proyecto hemos 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 hemos creado un mecanismo que permitirá realizar conexiones más seguras.

Este trabajo se puede dividir en tres partes. Por un lado tenemos la obtención de los datos, para lo que usamos 5 dispositivos iguales en hardware y software (Raspberry Pi). De estos dispositivos obtenemos marcas de tiempo cada segundo de dos formas distintas. Por otro lado, tenemos el análisis estadístico de los datos obtenidos, que generará nuevos datos a partir de los previos. Por último tenemos 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 hemos obtenido previamente.

Para obtener los datos hemos tenido que realizar 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. Por secuencial nos referimos a que primero obtenemos todas las marcas de tiempo de un dispositivo y posteriormente pasamos al siguiente. Por otro lado, en paralelo nos referimos a que se obtienen marcas de tiempo de todos los dispositivos simultáneamente.

Una vez tenemos todas las marcas de tiempo, debemos obtener sus desviaciones respecto al dispositivo que tomamos de 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 obtenemos variables estadísticas.

Una vez tenemos las variables estadísticas procedemos a analizar los modelos de machine learning que vamos a usar. 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 usaremos el modelo entrenamiento/validación/test como forma de dividir nuestros datos. Entrenamos todos nuestros modelos con el conjunto de entrenamiento y comprobamos su capacidad de generalización con el conjunto de validación. A la hora de entrenar los modelos comprobaremos diferentes parrillas de hiperparámetros para ajustar nuestros modelos lo más posible a los datos de entrenamiento.

Al realizar estos entrenamientos vemos que los modelos basados en árboles son los mejores, tanto árboles de decisión (97.57 %) como Random Forest (99.51 %). Nos quedamos finalmente

con el modelo de Random Forest como algoritmo final y lo entrenaremos con la totalidad de la base de datos de entrenamiento.

Finalmente el modelo de Random Forest con los hiperparámetros ajustados obtiene un valor de accuracy de 99.44 %.

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, we will create a system that will have the ability to identify these differences and as a consequence, between many devices.

As I have mentioned, in this article we will create 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. [1] 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. [2] 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. [3] 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. [4] 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 our case, to identify different devices we will use 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, we will use these values to train a machine learning model that is capable of performing this process automatically.

Our test scenario 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)”. We will take the client’s time as a reference, and we will analyze the differences between the other devices and the client.

The first problem appeared at this time. Taking into account that we are going to analyze very small differences, we 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 we can only represent times in milliseconds, which does not provide enough information. For this reason, we decided to use the body section of the packets; to be able to send data of any length. In our case, it will send absolute timestamps in nanoseconds, which will be encoded with 64 bits. Finally, we chose the TCP protocol because in this way, we do 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 we 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, we 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, we will obtain the increase in the deviation between each sample of a device. Due to these increases, we will generate 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 we work with such small values, the outliers hide completely the results we want to see, so we will not show them.

The median is expected to be approximately 0 and the interquartile range is expected to be very similar on each device. We expect these results since we are analyzing clonic devices and they do not suffer from clock skew, at least theoretically.

If we look at 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, we will not take into account this sample; we will lay the focus on the parallel sample.

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

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

- **Decision trees:** starting from a set of elements in the parent node, we ask binary questions (yes or no) such that we can divide the set into two purer ones.
- **Random forest:** set of decision trees working in parallel. Each tree will perform the divisions randomly, with which we achieve variability in the results of each one. Finally, we take as the output 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, we must set their hyperparameters. To make this adjustment, we will use 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. We will use a training/validation/test model.

We will divide the set of all the data into two, one with 70% of the data and another

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

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

The process of adjusting an algorithm takes time since we must train each algorithm with each combination of hyperparameters that we want to test. For this reason, we will reduce the volume of data we have (70 % of the total) and we will keep 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, we took random samples, but as our data had temporal correlation, we needed to reorder them so that we can keep this correlation and the models are able to recognize it.

To adjust the hyperparameters of an algorithm we will use an object called grid that allows us to specify all the hyperparameters that we want to adjust with their respective values. 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, we 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, we will choose this random forest model as the final model.

Finally, we only have to train the random forest model with the hyperparameters we have chosen and the training set in its completeness. Once this has been done we check its ability to generalize with the test set and obtain a final accuracy of 99.44 %.

As a conclusion we have 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 we had done the same study 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, we would expect to see that the graph of the deviation was linear, but this is not the case even if we are using a clock that doesn't decrease its value 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 [5]. 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 únicamente 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.

Los fines de esto son, por ejemplo, introducir virus en sus equipos, para realizar ataques DDoS (ataques de denegación de servicio distribuidos) mediante miles de equipos infectados, para cifrar los datos de dicho equipo (ataque de ransomware), para realizar ataques de phishing, enviando páginas visualmente idénticas a las que consulte el equipo, pero los datos sensibles de los formularios (contraseñas) pasen a disposición del atacante. Otro fin posible es el mero espionaje de los datos.

Es en este punto en el que se requieren técnicas de comunicación seguras entre los dispositivos. Desde el punto de vista de las redes existen protocolos de comunicación segura como SSL, TLS, IPsec, etc. Pero estos dispositivos son inútiles si el equipo se quiere conectar voluntariamente al equipo atacante (esto debido al engaño que se ha comentado anteriormente).

Por estos motivos, nos preguntamos cómo podemos saber a qué equipos nos estamos conectando, 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.

En el panorama actual del big data y el machine learning podemos explotar estas diferencias 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.

CAPÍTULO 2

Estado del arte

En este capítulo 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.

Comenzamos la revisión bibliográfica por el trabajo desarrollado por Pascal Oser et al. [1]. En este trabajo se desarrolla un sistema para identificar dispositivos entre un total de 562 en el centro del CERN. Para este cometido se toman 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 [6] 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. 2.1 con dos ejemplos.

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. [2]. En este trabajo se desarrolla un sistema que identifique a un dispositivo que quiere

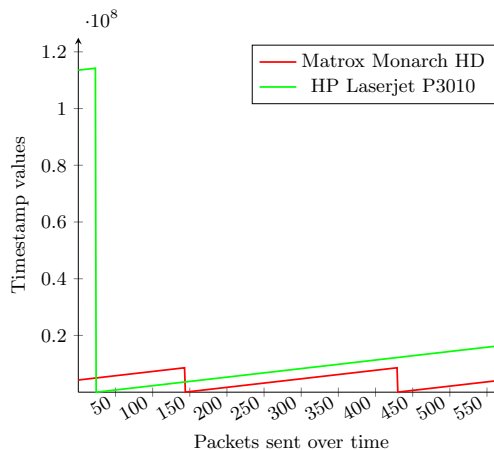


Figura 2.1: Desbordamiento del contador [1]

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. 2.2), 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 conectarse 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.

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. [3]. 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.

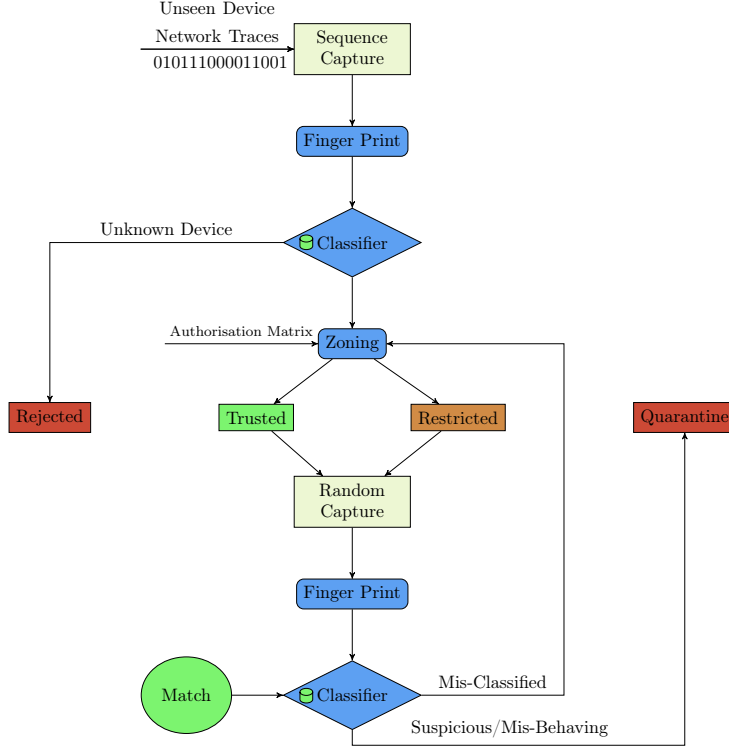


Figura 2.2: Modelo del sistema de identificación [2]

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

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 [7], de donde se obtienen 20 medidas de 23 dispositivos IoT.

Para la clasificación se han usado diferentes algoritmos de la herramienta WEKA [8] como son tablas de decisión, árboles de decisión J48, OneR y PART. Se decide usar una clasificación en dos niveles (Fig. 2.3), 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.

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. [4]. 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.

Para cada uno de los dispositivos se realiza una captura de 5 minutos. Esta captura será

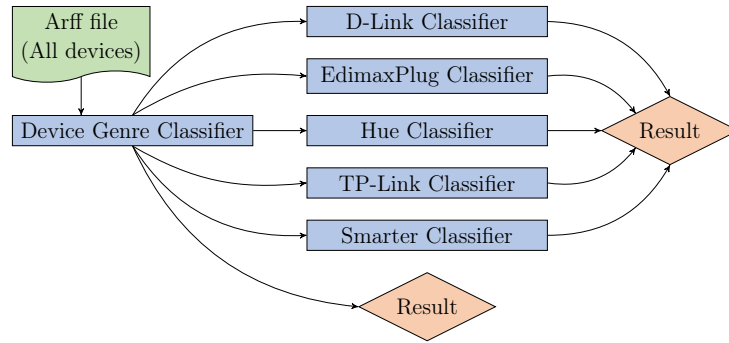


Figura 2.3: Clasificador de dos niveles [3]

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 [9]), con los que obtiene unos resultados de 96.3 %, 94.7 % y 76 % respectivamente.

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

Propuesta	Algoritmos usados	Resultados
Pascal Oser et al. [1]	MLP, SVM, Random Forest	99.76 % de accuracy y 97.03 % de precisión
Salma Hamad et al. [2]	AdaBoost, LDA, KNN, Árboles de decisión, Naive Bayes, SVM, Random Forest y GBoost	89 % de accuracy
Ahmet Aksoy et al. [3]	Tablas de decisión, Árboles de decisión J48, OneR y PART	Entre 42.2 % y 100 % de accuracy, con un promedio de 82 %
Hossein Jafari et al. [4]	DNN, CNN y LSTM	96.3 % de accuracy en DNN, 94.7 % de accuracy en CNN y 76 % de accuracy en LSTM

Tabla 2.1: Resultados en el estado del arte

CAPÍTULO 3

Análisis de objetivos y metodología

En esta sección definiremos los objetivos del trabajo y se realizará un análisis sobre las decisiones tomadas.

3.1 Objetivos

El objetivo de este trabajo será entrenar un modelo de machine learning que sea capaz de identificar cada dispositivo respecto de los otros que se encuentran bajo análisis. Para realizar esto mediremos la desviación de los respectivos relojes (*clock skew*).

Esta desviación se verá como la diferencia entre un reloj teóricamente exacto c_{ex} y el reloj del dispositivo a analizar c en un tiempo t . Con esto definimos la desviación del reloj c en el tiempo t como:

$$offset(c(t)) = c(t) - c_{ex}(t)$$

Una vez tengamos esta desviación de reloj para cada uno de los dispositivos, estudiaremos como varía esta desviación a lo largo del tiempo. Para esta labor definimos:

$$\Delta offset(c(t)) = offset(c(t)) - offset(c(t-1))$$

Una vez tenemos los datos recogidos, estudiaremos distintos valores estadísticos que, presumiblemente, nos aportarán información para distinguir los distintos dispositivos.

Por último analizaremos un conjunto de algoritmos de Machine Learning entrenados con un subconjunto estos datos, para evaluar su capacidad de identificar los dispositivos.

3.2 Algoritmos de Machine Learning

El Machine Learning se trata al fin y al cabo 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 vemos 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.

3.2.1 Tratamiento de los datos

Para obtener un resultado de cuan bueno es un modelo (algoritmo) que queremos usar debemos 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 datos, y con los datos de test comprobamos los resultados que obtenemos del algoritmo ante datos que no ha visto antes.

Para realizar una comparación entre distintos modelos y queremos ver cuál es el que mejor se ajusta a nuestros datos debemos dividir el conjunto de datos de entrenamiento en otros dos conjuntos: **conjunto de entrenamiento** y **conjunto de validación**. Con este último podemos evaluar la capacidad de generalizar de nuestro algoritmo.

Con esta segunda división obtenemos el algoritmo que mejor se adapta a nuestros datos, pero no un modelo final. Para tener un modelo final lo entrenaremos con el conjunto de datos de entrenamiento en su totalidad y lo evaluaremos con los datos de test.

3.2.2 Algoritmos de aprendizaje supervisado

Dentro de los algoritmos de aprendizaje supervisado encontramos 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 tenemos ejemplos **positivos** (pertenecen a la clase) o **negativos** (no pertenecen). Esto nos

lleva a 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 nos interesan para este trabajo.

3.2.2.1 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 podamos 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 podemos crear una partición diciendo si pertenecen o no a la clase. Por contra, si los atributos son ordinales podemos tener preguntas del tipo $x \leq x_c$.

3.2.2.2 Random Forest

Este algoritmo no es más que 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 obtenemos variabilidad en los resultados de los árboles. Por último, el algoritmo toma como salida aquella clase que haya sido el resultado de más árboles. [10] [11]

3.2.2.3 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.

3.2.2.4 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, \dots, a_n . Ahora este proceso se trata de conocer la probabilidad condicionada de que este conjunto de atributos pertenezca a una determinada clase. Esto es muy costoso y por tanto hay que

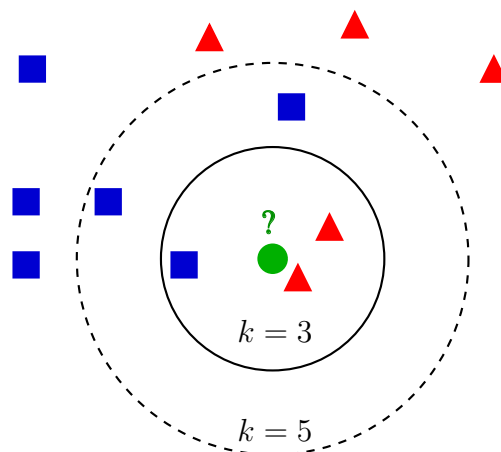


Figura 3.1: Funcionamiento del algoritmo KNN [12]

relajar esta hipótesis. Para ello se asume independencia entre los atributos de ahí el nombre de *naive*.

3.2.2.5 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. 3.1 se quiere clasificar el dato ● podemos usar múltiples valores de k y con ello obtendremos diferentes resultados:

- Si usamos $k = 3$ sólo nos fijaremos en 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 ▲.
- Si usamos $k = 5$ sólo nos fijaremos en 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 ■.

3.2.2.6 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, nos centraremos 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. 3.2 podemos ver como hay distintos hiperplanos ($\mathcal{H}_1, \mathcal{H}_2, \mathcal{H}_3$) que pueden dividir el espacio.

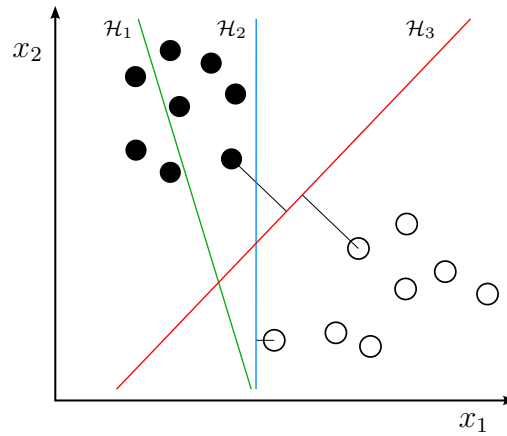


Figura 3.2: Separación por hiperplanos SVM [13]

- \mathcal{H}_1 no divide de forma correcta todos los puntos de la entrada, puesto que hay puntos negros juntos con blancos.
- \mathcal{H}_2 sí que divide a todos los puntos en dos clases de la forma correcta, pero no están separados 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.

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 podemos ver que tenemos un observador (modelo RockPro64) y 5 dispositivos a analizar (Raspberry Pi).

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 podamos obtener una marca de tiempo en 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 la mantendremos hasta el final.

4.2 Obtención de datos

Para obtener los datos usaremos una estructura cliente-servidor. En nuestro los dispositivos a analizar serán los que tomen el rol de servidores y el observador será el cliente.

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

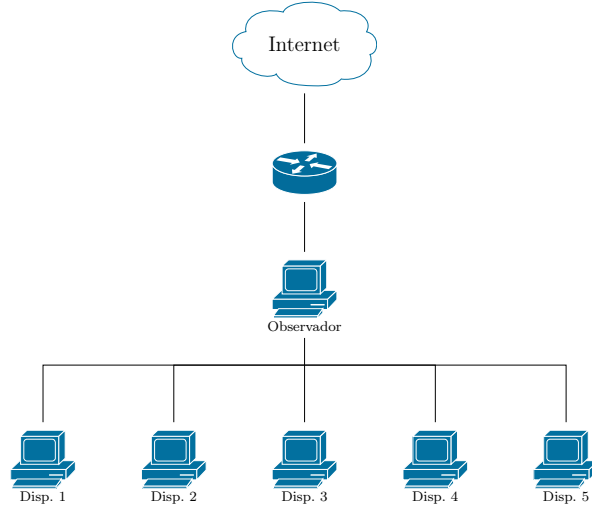


Figura 4.1: Topología de la red

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`[14]).

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 conseguimos 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.

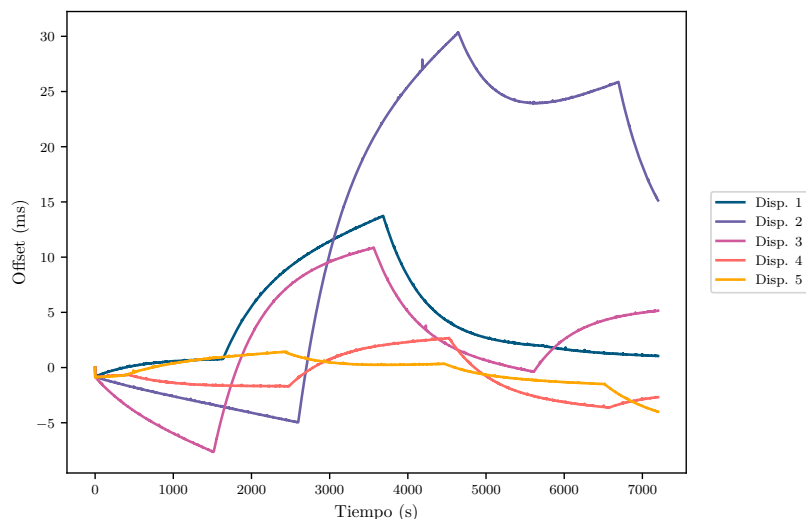


Figura 4.2: Offset acumulado muestra secuencial

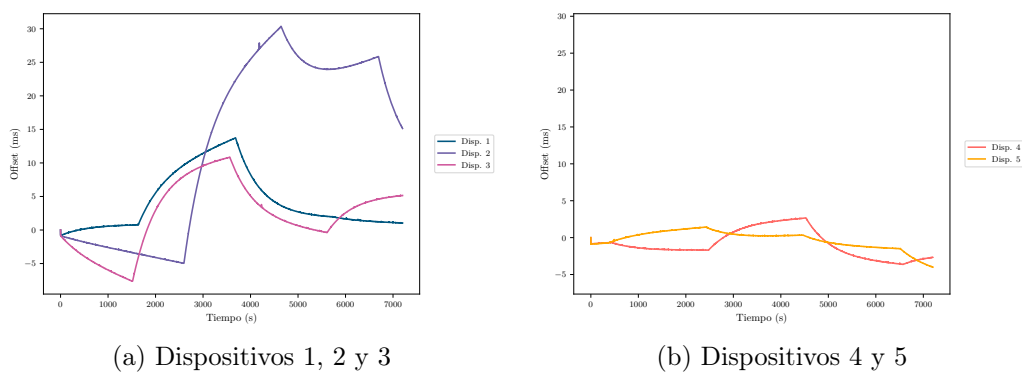


Figura 4.3: Diferencias entre offsets de dispositivos

4.3 Análisis de los datos

En esta sección analizaremos 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 observamos 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.

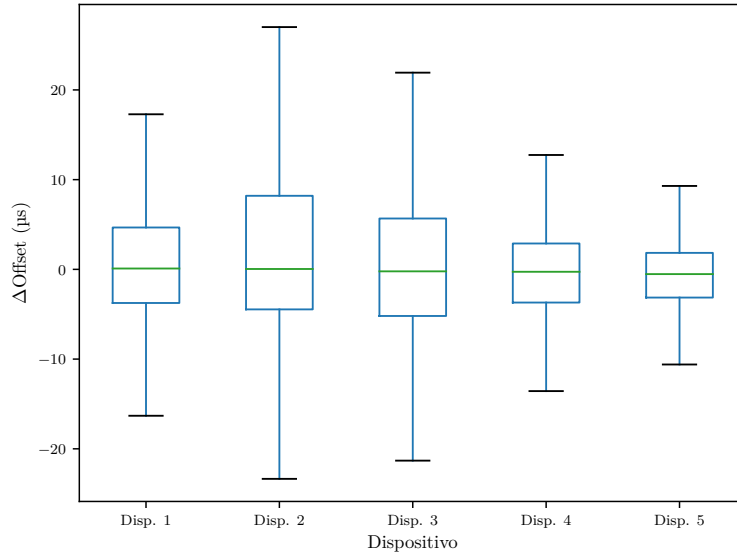


Figura 4.4: Diagrama de cajas muestra secuencial

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

Con este diagrama podemos ver 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 vemos 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 nos hacemos una idea de los dispositivos son distinguibles estadísticamente, lo que nos confirma que podremos 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.

En la Fig. 4.5 podemos ver 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

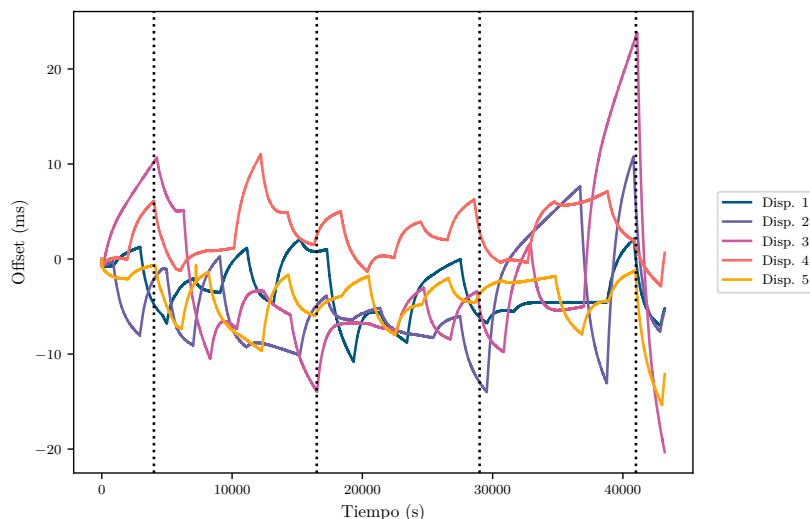


Figura 4.5: Offset acumulado muestra paralela

12 000 segundos).

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.

Si consultamos otro diagrama de cajas con el incremento de la desviación en cada instante (Fig. 4.6) podemos ver como todos los dispositivos están mucho a la par que en la muestra secuencial (Fig. 4.4). Siguen teniendo ciertas diferencias, por lo cual concluimos que son estadísticamente diferenciables también.

4.4 Elección de la muestra de datos

Analizando los incrementos de la desviación en ambas muestras vemos que la muestra paralela tiene valores más parecidos a los que esperaríamos. Estos valores de incremento de la desviación son más pequeños y más parecidos entre todos los dispositivos, es por ello que entrenaremos los modelos 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. Usaremos las siguientes variables estadísticas:

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

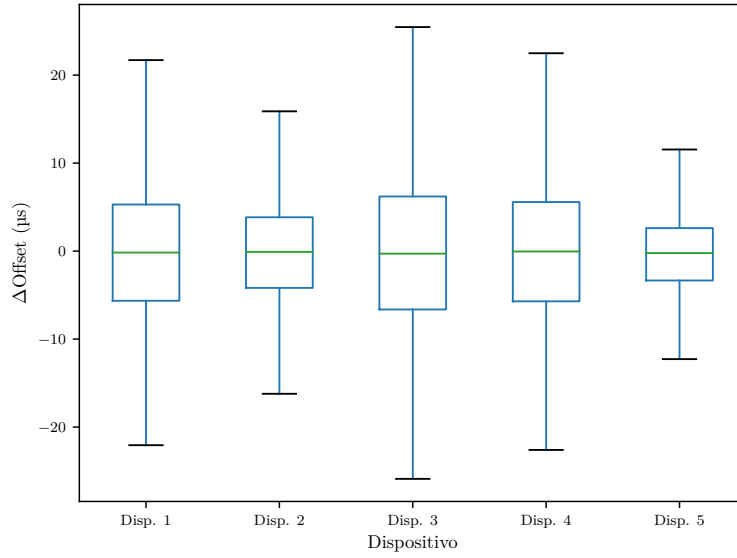


Figura 4.6: Diagrama de cajas muestra paralela

	Sum	Mean	Median	Mode	Std	IQR	Kurtosis	Skew	Max	Min	Device
1	21773.0	362.88333333333333	-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	19.783333333333333	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

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

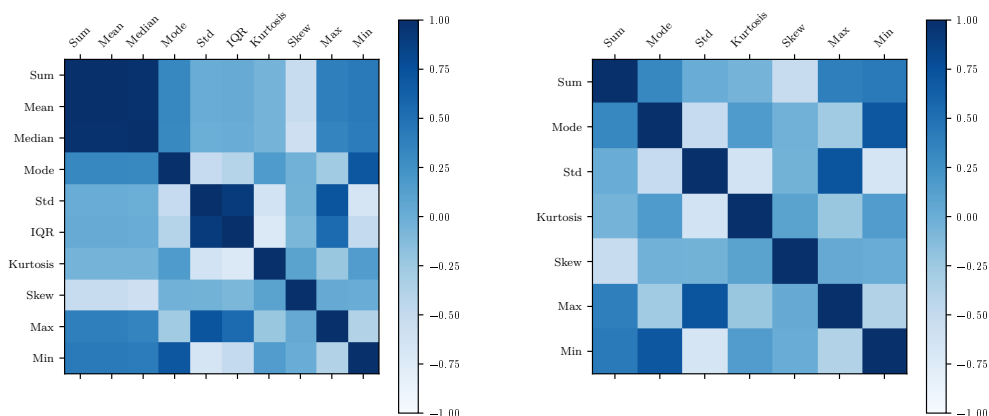
4.5 Reducción de la dimensionalidad

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. Vemos 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 analizaremos 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	<code>DecisionTreeClassifier</code> [15]
Random Forest	<code>RandomForestClassifier</code> [16]
MLP	<code>MLPClassifier</code> [17]
Naive Bayes	<code>GaussianNB</code> [18]
KNN	<code>KNeighborsClassifier</code> [19]
SVM	<code>LinearSVC</code> [20]

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 usemos (Sec. 3.2.2). 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 debemos 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 que aprende este modelo también para adaptarse a estos datos.

Tanto 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 lo haremos tomando muestras aleatorias, pero ordenadas para mantener la correlación temporal de los datos. Por ejemplo, si tuviéramos los datos

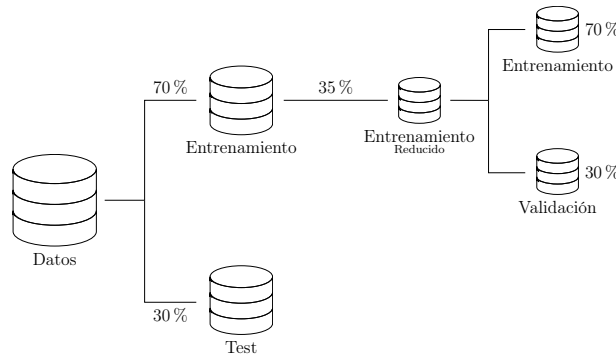


Figura 4.8: Particiones de los datos

x_1, \dots, 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 correlación temporal, como es nuestro caso, esta se pierde. Por tanto, reordenamos los datos 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 usaremos un conjunto reducido de este, un 35 % de los datos de entrenamiento.

A este subconjunto de los datos de entrenamiento también será dividido en un conjunto de entrenamiento/test, aunque en este caso al conjunto test lo llamaremos conjunto de validación. Con estos conjuntos entrenaremos los algoritmos para ajustar los hiperparámetros y decidir el modelo.

Para comprobar qué hiperparámetros son los que mejor se ajustan se usará la función `GridSearchCV` [21] que nos permite 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 podremos compararlos y quedarnos con los mejores. Esta función nos permite usar validación cruzada. La validación cruzada

Vamos a ajustar los hiperparámetros de un modelo como ejemplo, en este caso elegiremos el algoritmo de Random Forest sobre los datos la muestra paralela. Los hiperparámetros que se ha decidido 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 (hiperparámetro **max_depth**) a un valor de 1000.

Si nos fijamos primero en los hiperparámetros **criterion** y **max_features** (Fig. 4.9a) vemos que en promedio el mejor valor de **criterion** es **gini**. Fijaremos este parámetro y compararemos la totalidad de los valores entre **max_features** y **n_estimators** (Fig. 4.9b).

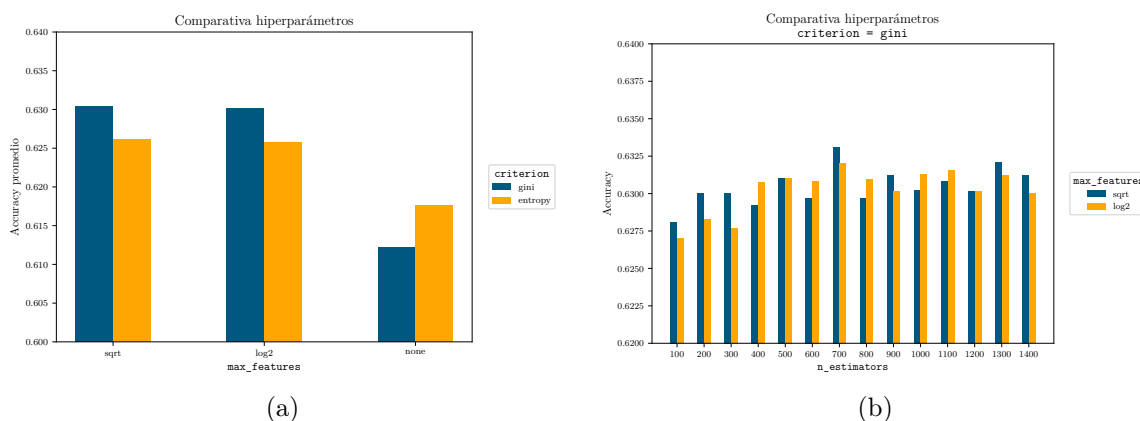


Figura 4.9: Comparativa hiperparámetros Random Forest

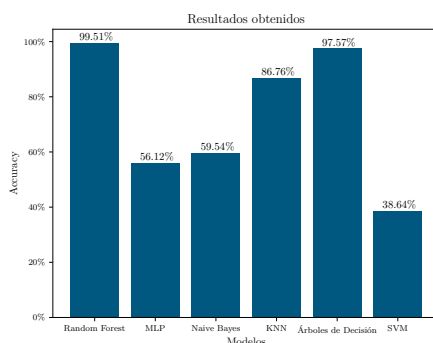


Figura 4.10: Comparativa de resultados entre modelos

En este caso vemos que los mejores valores son `criterion = gini`, `n_estimators = 700` y `max_features = sqrt`. Con estos hiperparámetros entrenamos un modelo sobre el conjunto de entrenamiento menor y después realizamos una predicción con el conjunto de validación.

Una vez entrenados todos los modelos con sus mejores hiperparámetros y con un valor de generalización obtenido de la predicción tenemos los resultados que se pueden ver en la Fig. 4.10.

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. 4.11 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 entrenaremos 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. 4.12). Con estos resultados obtenemos un valor final de accuracy de 0.9944 (99.44%).

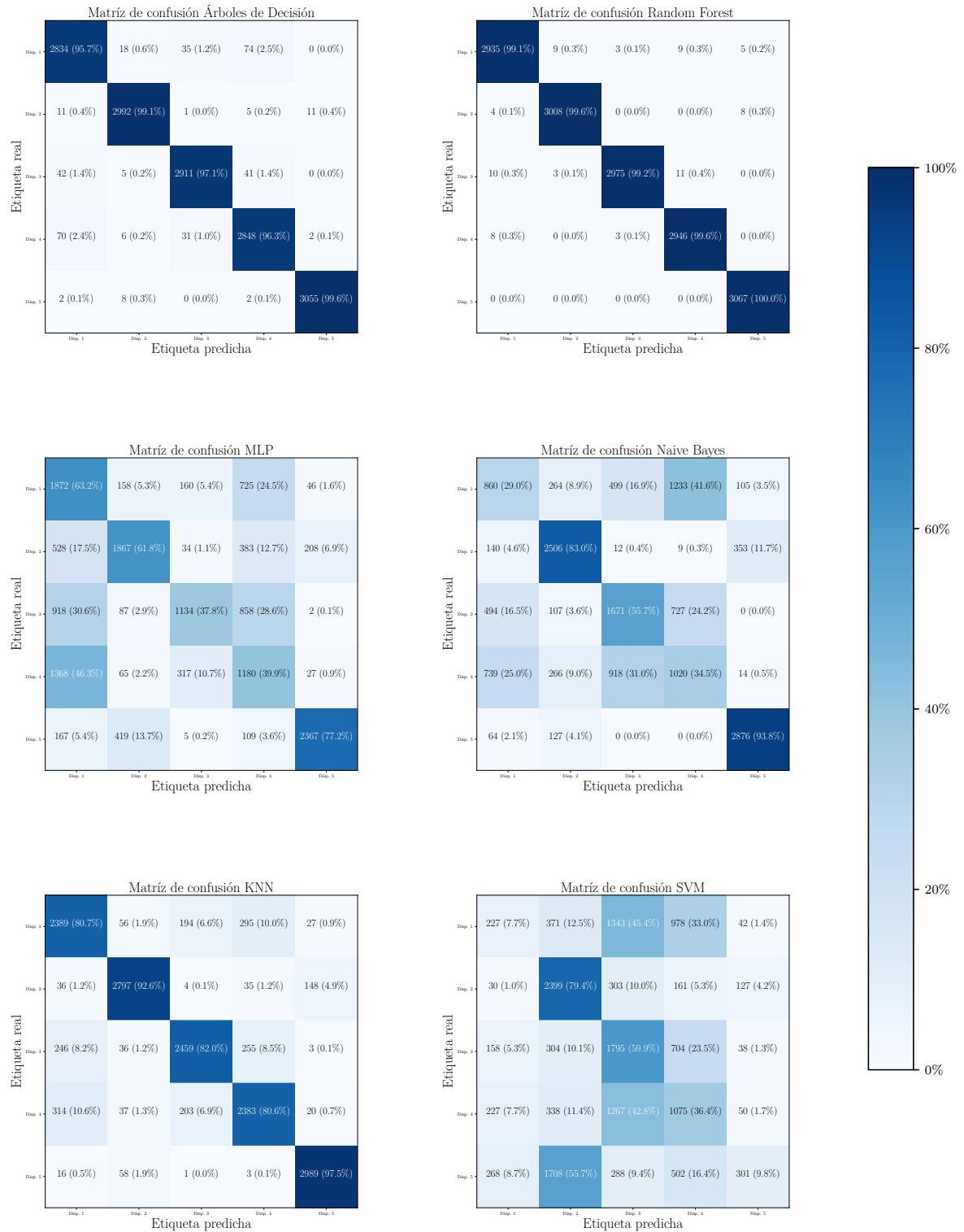


Figura 4.11: Matrices de confusión con datos de la muestra paralela

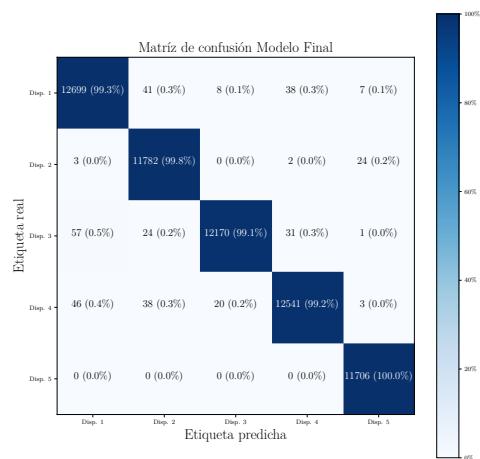


Figura 4.12: Matriz de confusión del modelo final

Conclusiones y vías futuras

En este proyecto hemos diseñado un modelo capaz de clasificar dispositivos con los que nos estamos comunicando en base a pequeñas diferencias en la fabricación de los componentes, que altera el tiempo que tardan en ejecutar una cierta tarea.

En la primera parte del proyecto hemos visto como para obtener la precisión en los tiempos que queríamos hemos tenido que usar el protocolo TCP y enviar las marcas de tiempo en el cuerpo del paquete.

En este momento se vio que el interno es susceptible de ser alterado por procesos externos para 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 captura de paquetes, pues las diferencias entre tiempos no serían las propias del 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 entrenamos el modelo 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 obtenemos unos resultados finales de 99.44 % en el valor de accuracy.

Como posibles vías futuras de este trabajo estaría 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.

Bibliografia

- [1] 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.
- [2] 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/Big-DataSE)*, IEEE, 2019, pp. 103–111.
- [3] 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.
- [4] 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.
- [5] 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.
- [6] 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>.
- [7] 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>.
- [8] 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, 10–18, 2009, ISSN: 1931-0145. [Online]. Available: <https://doi.org/10.1145/1656274.1656278>.
- [9] M. Abadi *et al.*, *TensorFlow: Large-scale machine learning on heterogeneous systems*, Software available from tensorflow.org, 2015. [Online]. Available: <https://www.tensorflow.org/>.
- [10] J. Hatwell, M. M. Gaber, and R. M. A. Azad, “Chirps: Explaining random forest classification”, *Artificial Intelligence Review*, vol. 53, pp. 5747–5788, 2020.

- [11] L. Breiman, “Random forests”, *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.
- [12] W. Commons. “File:knnclassification”. (2010), [Online]. Available: <https://commons.wikimedia.org/wiki/File:KnnClassification.svg>.
- [13] W. Commons. “Svm separating hyperplanes (svg)”. (2012), [Online]. Available: [https://commons.wikimedia.org/wiki/File:Svm_separating_hyperplanes_\(SVG\).svg](https://commons.wikimedia.org/wiki/File:Svm_separating_hyperplanes_(SVG).svg).
- [14] “std::chrono::steady_clock”, [Online]. Available: https://en.cppreference.com/w/cpp/chrono/steady_clock.
- [15] “DecisionTreeClassifier”, [Online]. Available: <https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier>.
- [16] “RandomForestClassifier”, [Online]. Available: <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html#sklearn.ensemble.RandomForestClassifier>.
- [17] “MLPClassifier”, [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html#sklearn.neural_network.MLPClassifier.
- [18] “GaussianNB”, [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html#sklearn.naive_bayes.GaussianNB.
- [19] “KNeighborsClassifier”, [Online]. Available: <https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html>.
- [20] “LinearSVC”, [Online]. Available: <https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html>.
- [21] “GridSearchCV”, [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html.