eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Grado en Ingeniería Informática

Computación

Trabajo de Fin de Grado

Detección de anomalías en redes IoT mediante Stream Machine Learning

Autor

Asier Zubia Garea

2021

eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Grado en Ingeniería Informática

Computación

Trabajo de Fin de Grado

Detección de anomalías en redes IoT mediante Stream Machine Learning

Autor

Asier Zubia Garea

Director

Jose Miguel Alonso

Resumen

Internet ha evolucionado rápidamente y esto ha permitido que las redes IoT (Internet of Things) sean ya una realidad y no sólo una visión de futuro. Las redes IoT integran dispositivos muy sencillos, con frecuencia insuficientemente protegidos, que las hace muy vulnerables a ataques. La detección de anomalías en dichas redes es crítica ya que permite minimizar el posible impacto de las mismas e incrementa la fortaleza de los sistemas.

En este trabajo se han desarrollado varios modelos basados en aprendizaje automático en streaming, que permiten la detección de anomalías en tiempo real en una red IoT de sensores de temperatura. Para ello hemos utilizado el conjunto de datos etiquetados DAD de una red IoT que contiene diferentes tipos de ataques: duplicaciones, intercepciones y modificaciones. Se han generado dos tipos de modelos, uno a nivel de aplicación y otro a nivel de red para detectar los ataques en el momento en el que se producen y así mantener la seguridad de la red IoT.

El trabajo también incluye las pruebas realizadas que permiten evaluar el comportamiento de los modelos.

Summary

The Internet has evolved rapidly and this has allowed IoT (Internet of Things) networks to become a reality and not just a vision of the future. IoT networks integrate very simple devices, often insufficiently protected, which makes them very vulnerable to attacks. The detection of anomalies in such networks is critical as it makes it possible to minimize the possible impact of anomalies and increase the robustness of the systems.

In this work we have developed several models based on streaming machine learning, which allow the detection of anomalies in streaming, in an IoT network of temperature sensors. For this purpose, we have used the dataset labeled DAD of an IoT network containing different types of attacks: duplications, interceptions and modifications. Two types of models have been generated, one at application level and one at network level to detect attacks as they occur in order to maintain the security of the IoT network.

The work also includes the tests performed to evaluate the behavior of the models.

Índice general

Índice de figuras

1. CAPÍTULO

Introducción

Desde hace un tiempo, el término Internet de las cosas, Internet of Things o simplemente IoT está muy presente, y no son pocos los fabricantes que lanzan al mercado dispositivos orientados a este sector [IoT, 2021].

Internet de las cosas no es sino una red de interconexión digital entre dispositivos, personas y la propia Internet que permite el intercambio de datos entre ellos, posibilitando que se pueda capturar información clave sobre el uso y el rendimiento de los dispositivos y los objetos para detectar patrones, hacer recomendaciones, mejorar la eficiencia y crear mejores experiencias para los usuarios.

Internet de las cosas es un concepto un tanto intangible o abstracto; es la conexión, por ejemplo, entre nuestro smartphone y los dispositivos smart que tenemos en casa para controlar la iluminación o el aire acondicionado. Es, por poner otro ejemplo, una Raspberry Pi que controla la programación del televisor, o un robot aspiradora que configuramos desde el móvil para que nos limpie la casa automáticamente cuando no estamos. Según el Worldwide Global DataSphere IoT Devices and Data Forecast, para el año 2025 tendremos en torno a 41.600 millones de dispositivos conectados [CTO, 2021].

Los dispositivos IoT se conectan mediante un proceso llamado M2M (machine to machine, o máquina a máquina) utilizando cualquier tipo de conectividad (que puede ser cable, WiFi, Bluetooth, etc.), haciendo su trabajo sin la necesidad de que un humano intervenga. Este proceso se realiza con emisores, receptores y chips específicos que están integrados en los dispositivos (por ejemplo, un smartphone habitualmente tiene Bluetooth, GPS, etc.).

Estos dispositivos conectados generan una gran cantidad de datos que llegan a una plataforma IoT que recolecta, procesa y analiza dichos datos. Dicha información permite al usuario o a otro actor (empresa) sacar conclusiones de los hábitos y preferencias del usuario, así como facilitarle la vida. Por ejemplo, si se produce algún fallo en tu coche, se puede generar un aviso para que lo lleves al taller; o si tienes un accidente con el coche, el coche activa automáticamente una señal/llamada de emergencia [IDC, 2021].

Los patrones de datos son bastante estables, pero a veces se producen anomalías. En este contexto se entiende por anomalía una situación inusual o estadísticamente improbable en los patrones de datos que se desvía de las expectativas. Su reconocimiento y detección es muy importante en redes IoT ya que nos proporciona información sobre la seguridad y/o sobre los errores producidos en dichas redes. La detección de anomalías ha permitido la identificación y prevención de actividades maliciosas como fraude e intrusiones, entre otros. Para la detección de anomalías se han utilizado técnicas de aprendizaje automático con buenos resultados.

1.1. Objetivos

El objetivo de este proyecto es analizar los datos obtenidos de un entorno IoT real para la detección de posibles anomalías, utilizando para ello técnicas de Streaming Machine Learning (SML). Se ha tomado como base un dataset con capturas de paquetes correspondientes a una red de sensores de temperatura de un datacenter. Dichos sensores envían muestras de temperatura a un broker centralizado. Las anomalías pueden producirse tanto a nivel de aplicación (manipulación de las temperaturas) como de red (eliminación, inyección de paquetes). Se han generado dos modelos de detector, uno a nivel de aplicación y otro a nivel de red, para la identificación inmediata de diferentes tipos de ataques. También se ha realizado un un proceso de evaluación seguido de un análisis de los resultados obtenidos.

El proyecto está dividido en varías fases con el fin de conseguir el objetivo final.

1.2. Organización del trabajo

Para llevar a cabo este trabajo se han tenido que realizar diferentes tareas, no necesariamente secuenciales.

1. La primera fase consiste en entender a la perfección el dataset utilizado. Verificar qué se puede analizar, y examinarlo manualmente para validar su contenido, incluyendo las anomalías. Para ello se ha realizado un análisis exhaustivo de los paquetes con los que se va a trabajar, es decir, se han estudiado los protocolos, puertos, direcciones IP, capas, información enviada, flujos etc. En esta primera etapa se han elegido las herramientas y librerías más adecuadas para dicho propósito.

2. Extraer del dataset diferentes series temporales, tanto a nivel de aplicación (temperaturas) como de red (protocolos, paquetes, bytes y otras medidas derivadas). Las series temporales son una colección de observaciones (muestras) de una variable o variables recogidas secuencialmente en el tiempo. Las series temporales sirven para entrenar los modelos y para obtener, dada una nueva muestra, un "anomaly score" que permite identificar las posibles anomalías.

3. Analizar dichas series temporales usando detectores de anomalías basados en SML. Comprobar hasta qué punto los detectores son capaces de identificar las anomalías presentes en el dataset.

4. Crear y poner en marcha un entorno de trabajo que recoja todas las funcionalidades que se han indicado para así lograr el objetivo final.

La realización de este trabajo ha sido necesario elegir un conjunto de herramientas y librerías:

- Python como lenguaje ya que dispone de un gran número de librerías para todo tipo de aplicaciones diferentes.
- Scapy como librería para el análisis de paquetes capturados de la red puesto que ofrece una amplia gama de funcionalidades para la extracción de información de los paquetes.
- River como librería de SML por su potente algoritmo de detección de anomalías, HST (Half-Space Trees).
- Wireshark como analizador de protocolos con interfaz gráfica.
- Pycharm y Jupyter Notebook como aplicaciones para la implementación del código.
- Google Colaboratory como plataforma para la ejecución del código.

1.3. Organización de esta memoria

En el capítulo 2 estudiamos los conceptos básicos de este trabajo:

1. El entorno físico de captura de los datos, es decir, la estructura física de una red de sensores ubicados en un centro de datos real, la relación entre los sensores, su distribución y su ubicación.
2. Los protocolos empleados para el intercambio de datos con medidas de temperatura, en particular, el protocolo MQTT.
3. El dataset utilizado para el análisis, explicando cómo se ha generado.
4. También se introduce el Streaming Machine Learning, resaltando por qué es imprescindible en el contexto de este TFG (Trabajo Fin de Grado) y a qué se va a aplicar.
5. Finalmente se describe la detección de anomalías y su relación con el uso de técnicas de aprendizaje automático.

En el capítulo 3 explicamos el proceso seguido para la selección de herramientas y librerías, priorizando en la medida de lo posible aquellas de código abierto.

El capítulo 4 detalla la elaboración de los modelos, su implementación y un análisis de los resultados obtenidos. Se ha dividido en los siguientes puntos.

1. Implementación
    - Generación de series temporales a nivel de aplicación y de red, para poder probar y entrenar los modelos.
    - Creación de los modelos, a nivel de aplicación y a nivel de red.
2. Análisis de los resultados

El capítulo 5 resume las principales conclusiones y aportaciones de este trabajo.

Por último, el capítulo 6 recoge la gestión del trabajo, desde la planificación y descripción de tareas, hasta el análisis de riesgos. En concreto se ha organizado de la siguiente forma:

1. Descripción de las tareas realizadas.
2. Cronograma que especifica el periodo de realización de las tareas.
3. Análisis de riesgos a los que se ha visto expuesto el proyecto.
    - Identificación de los riesgos y medidas de contingencia.
    - Matriz probabilidad-impacto para establecer prioridades.

2. CAPÍTULO

Contexto

Este capítulo se ha dividido en dos secciones. En la primera sección se presenta la estructura física de una red de sensores ubicados en un centro de datos real, la relación entre los sensores, su distribución y su ubicación. A continuación, se introduce el protocolo *MQTT* y finalmente se explica cómo se ha generado el dataset que se utiliza en este proyecto. En la segunda sección se introduce el *Streaming Machine Learning*, resaltando por qué es imprescindible y a qué se va a aplicar.

2.1. Arquitectura física y generación de datos

2.1.1. Captura de datos de sensores en un datacenter

Para aproximar el conjunto de datos a un entorno real, se obtuvieron datos de sensores de temperatura en el centro de datos del Centro de Investigaciones en Tecnologías de la Información y las Comunicaciones (CITIC de A Coruña) [CIT, 2019]. Este trabajo fue realizado por los investigadores Laura Vigoya [DAD, 2021], Diego Fernandez y Victor Carneiro, del grupo de telemática liderado por el Profesor Fidel Cacheda adscrito al Departamento de Ciencias de la Computación y Tecnologías de la Información de la Universidade da Coruña.

Las temperaturas de varios elementos del centro de datos se monitorizan mediante el uso de sensores pasivos inteligentes basados en tecnología NFC (Near Field Communication).

Hay tres elementos con sensores: los racks, las regletas de enchufes (PDU) y las máquinas de refrigeración (InRow). Solo consideramos los sensores de los InRows, porque los valores del resto de sensores varían muy poco y, por tanto, no son significativos. Dado que el posicionamiento de los sensores es importante para determinar su función, la estructura del centro de datos se muestra en la Figura 2.1.

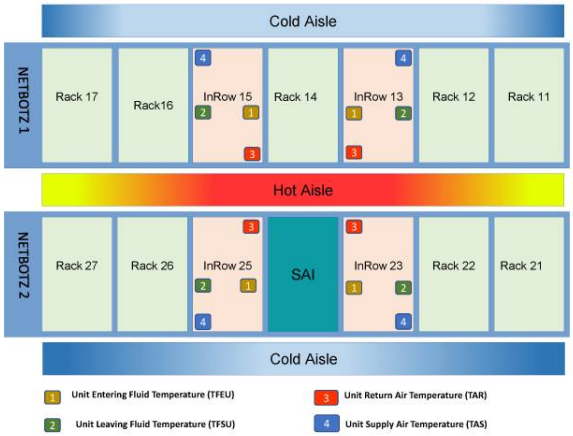

Figura 2.1: Arquitectura física

Los InRows (13, 15, 23 y 25) son los dispositivos encargados de enfriar el aire del centro de datos mediante un sistema de refrigeración líquida. Los InRows tienen 4 tipos de sensores relevantes:

- **Temperatura del aire de suministro de la unidad (TAS, marcados con un 4 azul en la figura):** Este detecta la temperatura del aire proveniente del pasillo frío. Normalmente, la temperatura medida por este sensor está relacionada con la medida obtenida por los sensores frontales de los racks.
- **Temperatura del aire de retorno de la unidad (TAR, marcados con un 3 rojo en la figura):** Este detecta la temperatura del aire ubicado frente al pasillo caliente. La medida dada por el sensor es equivalente a la medida por los sensores traseros de los racks.
- **Unidad que mide la temperatura del fluido (TFEU, marcados con un 1 amarillo en la figura):** Este sensor mide la temperatura del fluido de refrigeración del sistema antes del proceso de enfriamiento.
- **Unidad de temperatura del fluido de salida (TFSU, marcados con un 2 verde**

en la figura): Este sensor mide la temperatura del fluido de refrigeración después del proceso de enfriamiento.

Estos sensores envían datos cada cinco minutos, es decir, 288 muestras por día, para un total de 4032 muestras por sensor durante los catorce días de actividad de cada sensor.

Las señales obtenidas de los sensores en los InRows del centro de datos se muestran en la Figura 2.2. El comportamiento de las señales indica la carga de trabajo presentada en los procesadores, ya que la temperatura detectada depende de la carga de trabajo de la máquina. Algunos eventos externos, como abrir la puerta de un pasillo, también pueden cambiar el valor de los sensores.

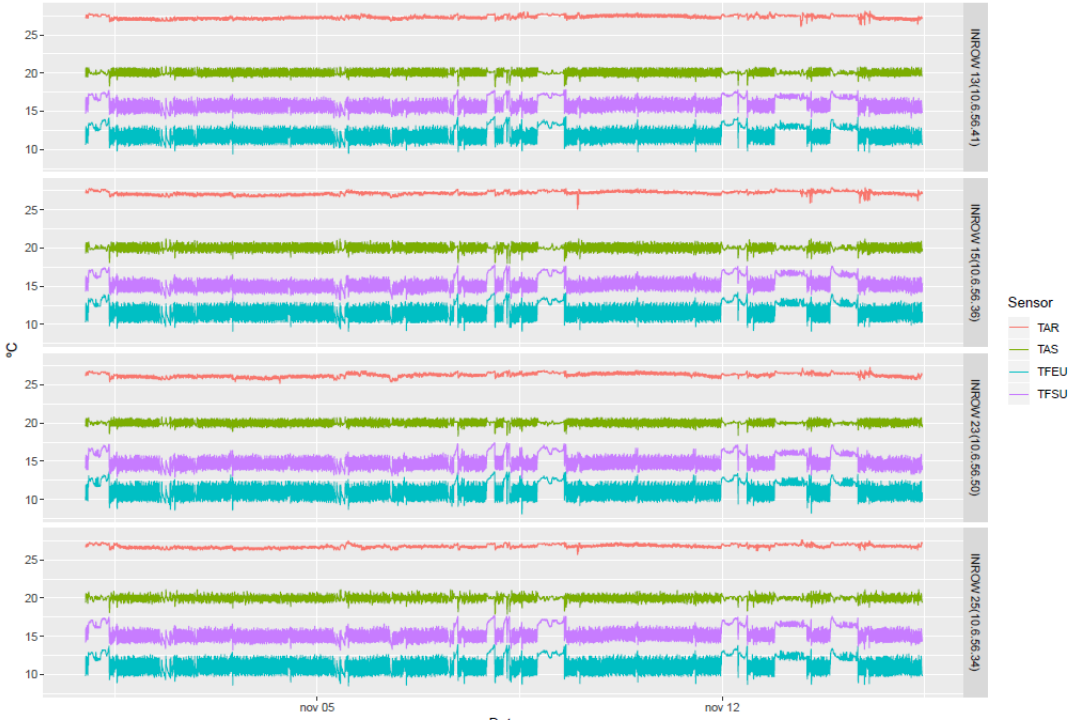

Figura 2.2: Medidas en un entorno real

2.1.2. Introducción a MQTT

"MQTT es un protocolo de transporte de mensajería cliente-servidor de publicación/suscripción. Es ligero, abierto, sencillo y está diseñado para ser fácil de implementar. Estas características lo hacen ideal para su uso en muchas situaciones, incluyendo entornos restringidos como la comunicación en contextos de Máquina a Máquina (M2M) e Internet de las Cosas (IoT) donde se requiere un ancho de banda pequeño" [MQT, 2020].

El resumen anterior de la especificación MQTT describe muy bien en qué consiste MQTT. Es un protocolo muy ligero y binario, y debido a su mínima sobrecarga de paquetes (la sobrecarga de protocolo en las redes informáticas se refiere a la información que debe enviarse con los datos que se transmiten a través de la red hacia un destino), MQTT destaca por ser un protocolo mucho más ligero que HTTP. Otro aspecto importante del protocolo es que MQTT es extremadamente fácil de implementar en el lado del cliente. La facilidad de uso fue una de las principales preocupaciones en el desarrollo de MQTT y se ajusta perfectamente a los dispositivos con recursos limitados [Richard Coppen, 2020].

El protocolo MQTT fue inventado en 1999 por Andy Stanford-Clark (IBM) y Arlen Nipper (Arcom, ahora Cirrus Link) [Stanford-Clark, 2021]. Necesitaban para una aplicación particular, conectar con oleoductos vía satélite, un protocolo que requiriera a la vez un consumo mínimo de batería y un ancho de banda pequeño. Los dos inventores especificaron varias condiciones para el futuro protocolo:

- Entrega de datos con calidad de servicio
- Eficiencia del ancho de banda
- Indiferente al formato de los datos
- Seguimiento continuo de la sesión

Estos objetivos siguen siendo el núcleo de MQTT. Sin embargo, el enfoque principal del protocolo ha cambiado de los sistemas integrados propietarios a los casos de uso abierto del Internet de las cosas (IoT), como es el caso de este proyecto. Este cambio de enfoque ha creado mucha confusión sobre el significado de las siglas MQTT. La respuesta corta es que MQTT ya no se considera un acrónimo. MQTT es simplemente el nombre del protocolo.

MQTT está basado en TCP/IP como base para la comunicación. Una característica de este protocolo es que las conexiones se mantienen abiertas y se “reutilizan” en cada comunicación.

El funcionamiento de MQTT se basa en un servicio publicador/suscriptor. Por lo tanto el cliente y el broker necesitan tener una pila (es un conjunto ordenado de protocolos en capas ficticias que se ponen unas encima de otras) TCP/IP como se puede observar en la Figura 2.3.

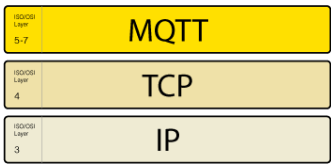

Figura 2.3: Pila TCP/IP del cliente y broker [Team, 2020]

Los clientes se conectan con un servidor central denominado broker. El broker se encarga de recibir todos los mensajes, filtrarlos, determinar quién está suscrito a cada mensaje (topic) y enviar el mensaje a los clientes suscritos. La representación gráfica del funcionamiento del protocolo se puede ver en la Figura 2.4. Dependiendo de la implementación el broker puede manejar millones de clientes MQTT conectados simultaneamente.

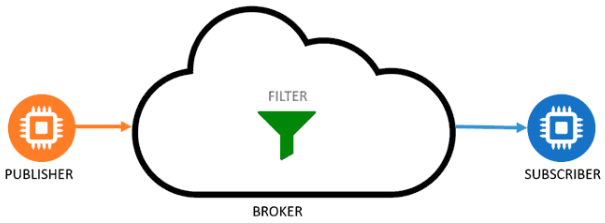

Figura 2.4: Funcionamiento del protocolo MQTT [MQT, 2021]

Primeramente los clientes establecen una conexión TCP/IP con el broker. El broker almacena en un registro todos aquellos clientes conectados. El puerto que utiliza el protocolo por defecto es el 1883 y si se utiliza sobre TLS, el puerto 8883.

Tal como se muestra en la Figura 2.5, la estructura de un mensaje MQTT es:

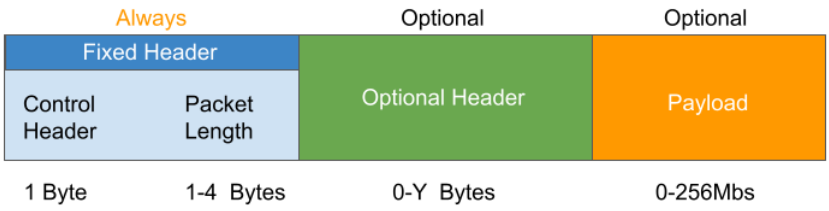

Figura 2.5: Estructura mensaje MQTT [MQT, 2021]

- **Cabecera fija:** Tiene un tamaño comprendido entre 2 y 5 bytes. Este es un campo obligatorio que debe contener un mensaje MQTT. Este campo contiene un código de control para identificar el tipo de mensaje enviado y la longitud del mensaje.

- **Cabecera variable:** Esta campo contiene información adicional dependiendo del tipo de mensaje MQTT. Es un campo opcional.
- **Contenido (payload):** Este campo incluye el contenido real del mensaje. Es un campo opcional.

Los tipos de mensajes MQTT son: CONNECT, CONNACK, PUBLISH, PUBACK, PUBREC, PUBREL, PUBCOMP, SUBSCRIBE, SUBACK, UNSUBSCRIBE, UNSUBACK, PINGREQ, PINGRESP, DISCONNECT.

A la hora de establecer el cliente una conexión con el broker, el cliente envía un mensaje *MQTT CONNECT*. Los clientes nunca se conectan entre sí directamente. Una vez que el broker recibe el mensaje de conexión, le responde con un mensaje *CONNACK*. El mensaje *CONNACK* contiene un atributo denominado *sessionPresent*. Dicho atributo puede tener valores desde 0 hasta el 5, e indica si la conexión se ha establecido correctamente y en caso contrario la razón por la que no se ha podido establecer la conexión.

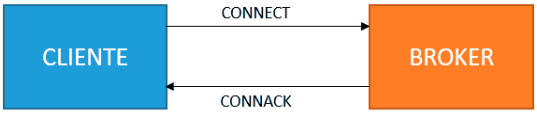

Figura 2.6: Conexión cliente-broker MQTT [MQT, 2021]

En caso de que el cliente quiera recibir mensajes de un *topic*, deberá suscribirse a dicho *topic*. El cliente envía un mesanje *SUBSCRIBE* indicando el *topic* y el broker le responderá con un mensaje *SUBACK*. Una vez establecida la conexión el broker la mantiene abierta hasta que el cliente envíe un mensaje de desconexión (*DISCONNECT*) o se interrumpa dicha conexión.

La estructura del mensaje enviado por el cliente para establecer la conexión se muestra en la Figura 2.7:

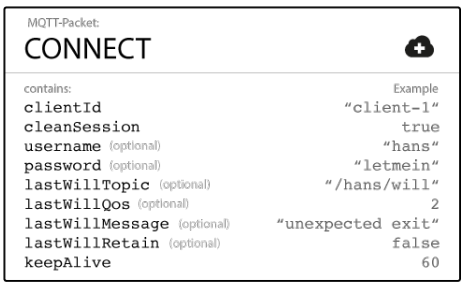

Figura 2.7: Contenido de un mensaje de conexión MQTT [Team, 2020]

Cuando el cliente quiera enviar un mensaje con información enviará un mensaje MQTT de tipo PUBLISH. Dicho mensaje contiene el *payload* y el *topic*. Más tarde será el broker el encargado de hacer llegar dicha información a los clientes suscritos al *topic*.

Con el fin de mantener la conexión abierta, los clientes envían periódicamente un mensaje PINGREQ para indicar que quieren seguir manteniendo la conexión abierta y el broker les responderá con un mensaje PINGRESP para comunicar a cada cliente que lo ha recibido.

2.1.3. Dataset

Generar conjuntos de datos buenos es una tarea que requiere mucho tiempo. El conjunto de datos empleado en este trabajo, DAD [Vigoya et al., 2020], se generó a través del seguimiento durante siete días de la situación que se da a diario en un entorno real (definido en la sección 2.1.1). El proceso tiene tres fases: configuración de los datos, captura de los mismos y etiquetado. DAD presenta diversos conjuntos de escenarios de anomalías donde el tráfico anormal es estadísticamente diferente de tráfico normal, cuando la mayoría de las instancias de tráfico de red son normales.

Analizados los datos reales del datacenter anterior, se crea un entorno virtual con máquinas que generan datos similares a los observados. En dicho entorno se pueden insertar diferentes tipos de anomalías:

- **Interceptación:** eliminar aleatoriamente algunos paquetes.
- **Modificación:** cambiar la temperatura a enviar.
- **Duplicación:** enviar más paquetes de lo planeado inicialmente.

La infraestructura virtual necesaria para la creación del conjunto de datos fue realizada por el entorno de virtualización VMWare ESXi6.5 en el que se ha definido una red virtual, denominada IoT interna. Esta red virtual está aislada del resto de redes de la infraestructura, utilizando la capacidad de VSwitch que proporciona esta herramienta. Sobre el entorno virtualizado, se han creado cinco máquinas virtuales con el sistema operativo del servidor Ubuntu servidor 18.04.

Cada uno de ellas está conectada a la red interna de IoT, para que el tráfico entre ellas esté totalmente aislado. En una de estas máquinas, se ha instalado el broker mosquitto [mos, 2018], que centraliza la suscripción de todos los clientes MQTT y es el lugar donde se realiza la captura de tráfico con tcpdump [tcp, 2010].

Este proceso inicializa a los clientes, distribuye el código asociado con cada nodo que implementa la serie temporal asignada, inicializa cada nodo y controla su ejecución hasta finalizar el envío de paquetes al broker. En cada uno de los cuatro nodos cliente, se lanza un proceso que simula cada uno de los cuatro sensores que componen cada una de las cuatro unidades frigoríficas. Estos procesos reciben un identificador correlacionado con el identificador de la unidad de frío (InRow); por ejemplo, la unidad de frío 13 tiene los procesos 131, 132, 133 y 134, cada uno de los cuales simula el funcionamiento de la temperatura del aire de suministro de la unidad (TAS), temperatura del aire de retorno de la unidad (TAR), temperatura del fluido de entrada de la unidad (TFEU) y temperatura del fluido a salida de la unidad (TFSU). Cada una de las muestras generadas por los sensores se envía cada cinco minutos al broker por medio de un mensaje MQTT que contiene su identificador de nodo como ClientId, una vez que el sensor se ha conectado para transmisión del mensaje correspondiente.

En el broker, a través de *tcpdump*, se realiza una captura del tráfico intercambiado con los nodos cliente. Posteriormente se anota como parte de la información que tienen los clientes sobre el envío de los tokens (mensajes a enviar) en la conexión, lo que les permite marcar aquellos tokens que pertenecen a situaciones anómalas. El escenario virtual se puede observar en la Figura 2.8:

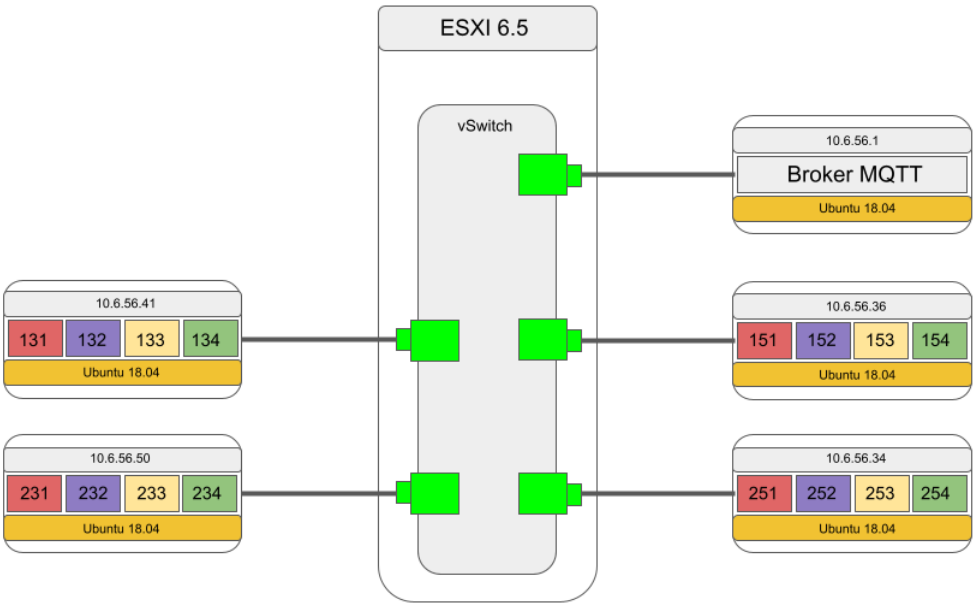

Figura 2.8: Escenario virtual

El dataset está formado por ficheros *.pcap*, *.csv* y *.xml*. Dentro de los ficheros *.pcap* se

encuentra la captura de tráfico correspondiente a un día concreto. En cuando a los ficheros *.csv* contienen características sobre cada paquete de un *.pcap* específico. Por último los ficheros *.xml* únicamente contienen información acerca de si un paquete es anómalo o no, de cada uno de los ficheros *.pcap*. El etiquetado del tráfico como benigno o malicioso se realiza tanto a nivel de paquete como de flujo para cada pieza de tráfico y también permite identificar qué paquetes forman parte de flujos con un comportamiento anormal o inusual. El proceso de anotación se realiza con la herramienta Scapy, que es un potente programa interactivo de manipulación de paquetes que será explicado en el capítulo 3. Dicha herramienta permite obtener los datos necesarios para registrar el conjunto de datos, la dirección IP y el puerto TCP del cliente MQTT. Para un paquete MQTT, se obtiene el identificador del mensaje o identificador del token, y el identificador del cliente indicado en el paquete. Por último, los paquetes que forman parte de flujos que han sido modificados se marcan en el campo de la etiqueta.

A continuación, se muestra un ejemplo de anotación en el archivo CSV:

$$frame.number; ip.src; tcp.srcport; mqtt.clientid; mqtt.msgid$$

El dataset utilizado no proporciona las etiquetas de aquellos datos que han sido interceptados pero si de aquellos que han sido modificados o duplicados.

Los ataques se encuentran divididos a lo largo de los 7 días de capturas de tráfico:

- Lunes 21: no hay ataques.
- Martes 22: se han eliminado algunos paquetes.
- Miércoles 23: se produce una modificación de paquetes entre las 4h y las 6h.
- Jueves 24: inserción de paquetes a las 3h.
- Viernes 25: se realiza una mezcla de interceptación, duplicación y modificación a las 6h y entre las 14-16h.
- Sábado 26: se realiza una mezcla de interceptación, duplicación y modificación a las 6h y entre las 14-16h.
- Domingo 27: no hay ataques.

2.2. Streaming Machine Learning

En una aplicación como la expuesta en la sección anterior, los datos cambian de forma dinámica. Tradicionalmente, los algoritmos de Machine Learning (ML) utilizan conjuntos de datos masivos que se adquieren de manera progresiva a lo largo del tiempo, lo que requiere infraestructura de memoria y un largo tiempo de procesado. Además, ocurre a menudo que los datos que hemos utilizado para entrenar un sistema hace unos meses, ya no son útiles en la actualidad, puesto que los datos y/o las anomalías en cuanto a sus propiedades, distribución y patrones han cambiado.

Así pues, el modelo estándar de ML no puede ser utilizado en contextos en los que no se dispone a priori de todos los datos a analizar. Los recursos disponibles de almacenamiento y procesamiento de los datos son los elementos claves a la hora de diferenciar entre el modelo estándar y el Streaming Machine Learning [Lobo, 2020]. Además, todo el tiempo que se ha invertido en entrenar el modelo con los diferentes conjuntos de datos no se puede recuperar y se ha perdido la oportunidad de detectar posibles patrones con los datos no utilizados, otro inconveniente si se quiere entrenar con frecuencia para mantener los modelos actualizados.

Alternativamente, los modelos en streaming (Streaming Machine Learning-SML) en lugar de esperar a conseguir y recoger dichos datos, nos permiten identificar patrones y tomar decisiones en base a los datos que están entrando constantemente. Además, aunque los patrones de los datos cambian dinámicamente, los modelos en streaming se pueden adaptar. De esta manera, nos permiten obtener análisis más actualizados de los resultados del modelo en tiempo real. Por dicha razón los modelos en streaming no necesitan almacenar datos antiguos ni requieren largos tiempos de procesado, lo que optimiza los recursos asociados a la detección de anomalías.

Otra de las ventajas de los SML es que mantiene los modelos de manera online. ¿Cómo?. Incorporando los datos constantemente, como ya hemos comentado. Esto permite que los conjuntos de datos para entrenamiento sean “infinitos”. A su vez, provoca que la máquina utilice de manera eficiente los recursos disponibles, detecte los cambios y se adapte fácilmente. Estas adaptaciones hacen que se creen modelos dinámicos.

Entre las muchas aplicaciones de SML se pueden destacar la detección de intentos de fraude en un banco, o la detección de ataques en una red, como es el objetivo de este proyecto.

2.2.1. Detección de anomalías

La detección de anomalías es la identificación de elementos raros, eventos u observaciones que generan sospechas al diferenciarse significativamente de la mayoría de los datos [López-Avila et al., 2019]. El uso de técnicas de aprendizaje automático ha permitido en los últimos años obtener resultados de gran relevancia en la detección de anomalías.

Sus aplicaciones son de amplio espectro entre las que podemos destacar: el análisis de redes informáticas y sociales, análisis de transacciones bancarias, y análisis de datos sensoriales, entre otros. En una red informática, patrones de comportamiento inusual podrían significar que un ordenador pirateado está enviando datos confidenciales a un destino no autorizado o, por citar otro ejemplo, las lecturas poco frecuentes de un sensor de nave espacial podrían significar un error en algún componente de la nave. Todos estos eventos que no siguen el patrón esperado son conocidos como anomalías y su detección permite la prevención de nuevos ataques o funcionamientos deficientes. Gracias al aprendizaje automático, se pueden generar modelos con dichas técnicas que ayuden a encontrar una "aguja en un pajar".

El Streaming Machine Learning conecta perfectamente con la detección de anomalías. Dependiendo de las situaciones, es muy importante la detección instantánea de las mismas, para proceder con la correspondiente reacción que evite un desafortunado evento. Por ejemplo, monitorizando los cardiogramas de una persona en tiempo real, la detección de una posible anomalía podría predecir el futuro infarto de dicha persona y lograr evitarlo o tratarlo cuanto antes. Al no monitorizarlo en tiempo real sino una vez pasado el tiempo, la detección se haría de la misma manera, salvo que cabe la posibilidad de que no se reaccione a tiempo. Otro ejemplo claro podría ser el de los ataques informáticos a las entidades financieras. Monitorizando todas las transacciones en tiempo real se podría evitar una pérdida masiva de dinero.

3. CAPÍTULO

Herramientas y librerías

En este capítulo se describen brevemente las herramientas y librerías más importantes que hemos utilizados para la ejecución de este proyecto. En la medida de lo posible se han utilizado herramientas y librerías de código abierto.

3.1. Introducción

A la hora de elaborar el proyecto es fundamental hacer una buena elección de las herramientas con las que se va a trabajar. Durante la primera fase del proyecto se investigó sobre las herramientas más adecuadas entre las disponibles. Esta fase incluyó la prueba de dichas herramientas antes de tomar una decisión. Previamente a la búsqueda de las herramientas se eligió el lenguaje de programación a utilizar. El lenguaje escogido fue Python, ya que actualmente es uno de los más usados en el contexto de ciencia de datos, y dispone de un gran número de librerías para todo tipo de aplicaciones diferentes.

Dado que se va a trabajar con capturas de tráfico realizadas en un datacenter, se buscaron y analizaron varias herramientas diferentes para la extracción y análisis de los paquetes. Entre ellas están Tshark, PyShark y Scapy. Son todas librerías Open Source de Python que se actualizan periódicamente y que añaden a su gran funcionalidad el respaldo que dan muchos profesionales dentro del campo de la extracción de datos de capturas de tráfico.

Previamente a la instalación, se indagó en las funcionalidades que ofrecían cada una de las librerías y en cómo aplicarlas a las capturas. Tras leer la documentación necesaria se

procedió con el siguiente paso, la instalación. Una vez instaladas las librerías, se realizaron las pruebas correspondientes para determinar cuál debía ser la herramienta con la que trabajar. Al final se optó por la librería Scapy. La elección fue clara debido a la gran información que se puede extraer de los paquetes, algo muy importante para las siguientes fases del proyecto.

El siguiente paso era seleccionar los programas en los que se iba a desarrollar el proyecto. Los programas escogidos para la implementación fueron el IDE Pycharm, Google Colaboratory y los cuadernos Jupyter (Jupyter Notebooks).

Después, se realizó una investigación más a fondo sobre los diferentes algoritmos de detección de anomalías en Stream Machine Learning. Debido al aumento de la investigación y del uso de la Inteligencia Artificial, el número de librerías desarrolladas es enorme. Sin embargo, al ser el Stream Machine Learning un campo relativamente novedoso, las librerías desarrolladas y avaladas por profesionales en el campo no son demasiadas. Durante este proyecto se descubrieron librerías como Scikit-multiflow, Creme y River. A continuación, se estudiaron los algoritmos que disponían para la detección de anomalías. Todos ellos cumplían con los requisitos básicos: trabajar con datos a los que le faltan características y trabajar de forma eficiente con pocos datos para el entrenamiento. Dichos requisitos se impusieron tras un análisis previo de los datos de las capturas de tráfico. Finalmente se determinó utilizar la librería River para desarrollar los diferentes modelos de Stream Machine Learning. Las razones por las que se optó por esta herramienta fueron que era una mezcla entre las dos librerías anteriormente comentadas, recogiendo lo mejor de ambas y añadiendo nuevas funcionalidades que parecían interesantes para el correcto desarrollo de la creación de los modelos.

3.2. Herramientas

3.2.1. Jupyter Notebook

Jupyter Notebook es una aplicación de código abierto que permite crear y compartir documentos que contienen código, ecuaciones, métricas, gráficos, visualizaciones y texto [Jup, 2021]. La principal característica de esta aplicación es que dispone de la posibilidad de ejecutar el código línea a línea en tiempo real. Jupyter soporta más de 40 lenguajes de programación diferentes, entre los que se encuentran Python, R, Julia y Scala.

3.2.2. Wireshark

Wireshark es el analizador de protocolos con interfaz gráfica más famoso y utilizado del mundo [Wir, 2021]. Esta aplicación permite observar y analizar de manera pormenorizada todo lo que está ocurriendo dentro de una red.

Esta herramienta ha sido fundamental a la hora de estudiar el contenido de los paquetes de las capturas de tráfico y su correspondiente funcionamiento. Su gran utilidad ha sido a la hora de filtrar los paquetes de las capturas de tráfico o mirar el contenido de paquetes específicos.

3.2.3. Pycharm

Pycharm es un IDE (Entorno de Desarrollo Integrado) específico para el lenguaje de programación Python. Un IDE [Pyc, 2021] es un editor y compilador que tiene como fin escribir y compilar código para la creación de programas. Esta aplicación proporciona al usuario análisis de código, debugger (depurador) gráfico, compatibilidad con el desarrollo web Django además de muchas otras características. Pycharm fue creado por los desarrolladores de la compañía Jetbrains.

Mediante esta herramienta se ha escrito todo el código del proyecto en el lenguaje Python.

3.2.4. Google Colaboratory

Google Colaboratory, también llamado Google Colab, es la herramienta de Google en la nube para ejecutar código Python y crear modelos de Machine Learning a través de la nube de Google y con la posibilidad de hacer uso de sus GPUs. “Google Colaboratory es un entorno gratuito de Jupyter Notebook que no requiere configuración y que se ejecuta completamente en la nube” [Sanz, 2019].

La principal ventaja que ofrece esta herramienta es que libera a nuestro ordenador de tener que llevar a cabo un trabajo demasiado costoso tanto en tiempo como en potencia o incluso nos permite realizar ese trabajo si nuestro equipo no cuenta con recursos suficientemente potentes. Y todo de forma gratuita. Otro de los beneficios que tiene lo indica el propio nombre, “Colaboratory”, es decir, colaborativo; nos permite realizar tareas en la nube y compartir nuestros cuadernos si necesitamos trabajar en equipo.

Para utilizarlo basta con acceder a nuestra cuenta de Google o bien entrar directamente al enlace de Google Colab o ir a nuestro Google Drive, pulsar el botón “Nuevo” y desplegar el menú “Más” para seleccionar “Colaboratory”, lo que creará un nuevo cuaderno (notebook).

Para el desarrollo del proyecto, Google Collaboratory ha posibilitado junto con los cuadernos Jupyter, la ejecución del programa en situaciones en las que no se disponía de recursos hardware. Así mismo también ha permitido obtener los resultados más rápidamente.

3.3. Librerías

3.3.1. River

River es una librería Python de código abierto de Streaming Machine Learning. Es el resultado de la fusión de dos librerías de Python, Creme [Halford, 2021] y Scikit-multiflow [Sci, 2021]. El objetivo de los desarrolladores es conseguir que sea la librería de referencia para programar Machine Learning con datos en tiempo real [Montiel et al., 2020].

Con ayuda de esta librería se ha realizado la implementación de los modelos. Para ello, las funciones más destacables han sido:

- HalfSpaceTrees(n_trees,height,window_size,seed): esta función corresponde a la implementación del algoritmo de detección de anomalías *Half-Space Trees*. Este algoritmo basado en árboles es una variante online del conocido Isolation Forest [Liu et al., 2008]. Además, trabaja con datos numéricos comprendidos entre 0 y 1. Los parámetros de entrada para la función son los siguientes:

  - **Número de árboles:** Indica el número de árboles a utilizar.
  - **Altura:** Indica la altura de cada árbol. Por ejemplo,un árbol de altura *h* está formado por *h + 1* niveles y por lo tanto contiene *2 ** (h +1) - 1* nodos.
  - **Tamaño de la ventana:** Indica el número de observaciones a utilizar para calcular la masa de cada nodo de cada uno de los árboles.
  - **Semilla:** Número aleatorio para la semilla.

  Esta función contiene tres métodos:

- **learn_one(dict):** Actualiza el modelo. Se aceptan múltiples variables organizadas en un diccionario que recibe como paramámetro de entrada. El diccionario es una estructura donde se almacenan las características de una muestra de la serie temporal. Los rangos de dichas variables deberán estar comprendidos entre 0 y 1. Existe la posibilidad de que haya datos vacíos, pero esto no supone un problema porque el algoritmo trabaja bien con diccionarios incompletos.
- **score_one(dict):** Devuelve una puntuación de anomalía comprendida entre 0 y 1. Recibe un diccionario como parámetro de entrada. Una puntuación baja indica una observación normal y una puntuación alta es indicativa de una anomalía.
- **clone:** Clona el modelo actual con los mismos parámetros.

- MinMaxScaler(): esta función se ocupa de normalizar (transformar) los valores de los datos, de manera que escala los datos a un rango fijo entre 0 y 1.
- Pipeline(): esta función permite concatenar diferentes pasos en una secuencia. Se ha utilizado principalmente para crear una cadena con las dos funciones anteriormente comentadas. De esta manera, el modelo se encargará él mismo de escalar los datos.

3.3.2. Scapy

Scapy es un potente programa interactivo de manipulación de paquetes de red. Es capaz de generar o decodificar paquetes de un amplio número de protocolos, enviarlos por el cable, capturarlos, emparejar peticiones y respuestas, y mucho más. Puede manejar fácilmente la mayoría de las tareas clásicas como el escaneo, el tracerouting, el sondeo, las pruebas unitarias, los ataques o el descubrimiento de la red (puede sustituir a hping, el 85 % de nmap, arpspoof, arp-sk, arping, tcpdump, tethereal, p0f, etc.). También realiza muy bien otras tareas específicas que la mayoría de las otras herramientas no pueden manejar, como el envío de tramas no válidas, la inyección de sus propias tramas 802.11, la combinación de técnicas (VLAN hopping+ARP cache poisoning, decodificación de VOIP en el canal encriptado WEP, ...), etc... [Biondi, 2018].

Scapy se ejecuta de forma nativa en Linux, Windows, OSX y en la mayoría de los Unix con libpcap. El mismo código base se ejecuta tanto en Python 2 como en Python 3.

La principal ventaja de Scapy es que, a diferencia de otras herramientas, nos proporciona la capacidad de extraer información acerca de los paquetes en cualquiera de sus capas.

Scapy ha sido de gran utilidad para la extracción de la información de los paquetes de los ficheros PCAP que forman las trazas DAD con las que se ha trabajado. Se ha hecho uso de muchas funciones diferentes. La cantidad de información a extraer de un paquete es enorme, por lo que se van a describir a continuación las funciones más utilizadas:

- **hasLayer(cls):** devuelve *true* si el paquete tiene una capa que es una instancia del parámetro cls. Las capas de un paquete más utilizadas son:
  - **IP:** Indica que el paquete contiene un mensaje IP.
  - **UDP:** Indica que el paquete contiene un mensaje UDP.
  - **TCP:** Indica que el paquete contiene un mensaje TCP.
  - **MQTT:** Indica que el paquete contiene un mensaje MQTT.
  - **MQTTConnect:** Indica que el paquete contiene un mensaje de conexión MQTT.
  - **MQTTPublish:** Indica que el paquete contiene un mensaje con una carga útil que contiene los datos a transmitir. Las temperaturas están codificados en formato decimal, carácter a carácter. Por ejemplo, una temperatura de 23 grados se enviaría con dos bytes: el primero con el código ASCII correspondiente al símbolo "2", y el segundo con el código ASCII correspondiente al símbolo "3".
  - **MQTTDisconnect:** Indica que el paquete contiene un mensaje de desconexión MQTT.
- **rdcap(nombre_fichero):** lee el fichero con extensión *.pcap* y devuelve una lista de paquetes.

  La información que se puede extraer acerca de un paquete es muy grande. A continuación se muestran los atributos más utilizados:

- **packet.sport:** Indica el puerto del que proviene el paquete.
- **packet.dport:** Indica el puerto al que se va a enviar el paquete.
- **packet[cls]:** Esta estructura permite extraer atributos específicos que formen parte de una capa específica del paquete. Para ello hay que pasarle como parámetro la capa escogida. Dentro de dichas capas (IP, UDP, TCP, MQTT, MQTTConnect, MQTTPublish y MQTTDisconnect) los atributos que se han manejado han sido los siguientes.

- **src:** Indica la dirección IP de la que proviene el paquete.
- **dst:** Indica la dirección IP a la que se ha enviado el paquete.
- **clientId:** Indica el identificador del cliente MQTT que envía el paquete.
- **value:** Indica el valor de la carga útil (valor de temperatura del sensor en bytes) de un mensaje enviado por un cliente MQTT.

Además de Scapy se ha utilizado Wireshark como herramienta visual para entender la estructura de los paquetes, facilitando así su análisis con Scapy.

4. CAPÍTULO

Implementación y evaluación

Este capítulo aborda la elaboración de los modelos y su implementación. Parte esencial es la extracción de series temporales ya que serán estas con la que se entrenarán y evaluarán los modelos. Para el desarrollo del proyecto se concretó implementar dos tipos distintos de modelos de SML: un primer modelo a nivel de aplicación y un segundo modelo a nivel de red. A nivel de aplicación, se trabajará con los datos de temperatura de los sensores. En cuanto a nivel de red, se actuará con el número de paquetes recibidos, enviados, número de puertos distintos utilizados, las diferentes direcciones IP, el número de bytes enviados y recibidos, y también flujos y protocolos. El capítulo finaliza con un análisis de los resultados.

4.1. Implementación

Los datos necesarios para entrenar el modelo a nivel de aplicación se obtienen con secuencias de medidas de temperaturas de los sensores y su correspondiente análisis multi-mono variado. Sin embargo, los datos para entrenar el modelo a nivel de red se obtienen a partir de datos estadísticos de ventanas de tiempo, exactamente de 5 minutos. Dicho intervalo se debe a que todos los sensores envían datos cada 5 minutos.

4.1.1. Generación de series temporales

Series temporales a nivel de aplicación

Se han generado tres tipos diferentes de series temporales, una para cada sensor, otra para cada máquina (4 sensores) y por último una para los 16 sensores (todos).

La extracción del valor de las temperaturas es la única información que interesa para la generación de las muestras de las series temporales, por lo que a pesar de haber generado tres tipos de muestras diferentes, la extracción se realiza de igual manera.

En concreto, a la hora de generar las muestras de las series temporales para entrenar al modelo de nivel de aplicación se han seguido los siguientes pasos:

```
1
2  while (llegan paquetes) loop
3
4      if paquete hasLayer(MQTTPublissh) entonces
5
6          insertar(Diccionario[sensor], valor de temperatura)
7
8      if (se han recibido datos de los 1 / 4 / 16 sensores) entonces
9
10         generarTimeSeries (4 valores de temperaturas de los 4 sensores)
```

Como ya se ha mencionado anteriormente, a pesar de trabajar con capturas de tráfico, hay que conseguir simular que se está escuchando directamente al broker, como si no se tuvieran dichas capturas. Todo esto para lograr que el proyecto tenga una aplicabilidad real. De ahí que el bucle principal sea un bucle infinito, dado que hay que suponer que los datos llegan constantemente, sin parar, a no ser que algún elemento hardware o software falle.

Sin embargo, como para este proyecto no se dispone de capturas de tráfico infinitas, en cuanto termina de analizar todos los paquetes, se fuerza la finalización de la generación de series temporales.

La única diferencia notable en lo que se refiere a la implementación de la generación de las muestras de las series temporales, viene dada por el momento en que se tienen que generar las mismas, es decir, el intervalo que hay que definir para saber cuándo generarlas. En el caso de un único sensor, tan pronto como se recibe el valor de temperatura del mismo en el broker, se genera una muestra de las series temporales que se pasa directamente al modelo correspondiente.

Si se consideran 4 sensores (uno por cliente MQTT, identificado por su dirección IP), hay que esperar a recibir los valores de temperatura de los cuatro sensores que interesan antes de generar la muestra de las series temporales y pasarlo al modelo. Ocurre lo mismo si se trabaja con datos de los 16 sensores; habrá que esperar a que se reciban los valores de todos los sensores antes de generar una nueva muestra de la serie temporal. Se ha observado el caso en el que ciertos sensores no envían ningún valor de temperatura al broker en el intervalo de tiempo de 5 minutos, que es el tiempo establecido para que los sensores envíen los datos al broker. Esta situación se ha considerado y se ha decidido que aunque no lleguen los valores de temperatura de algún sensor, para que el programa no se quede esperando infinitamente, tras un tiempo de 5 minutos, se generará la muestra de la serie temporal con los datos que se hayan obtenido. Es decir, se entregará un diccionario incompleto. Esto no ha supuesto ningún problema a la hora de entrenar el modelo con muestras incompletas, ya que la implementación de HST en River acepta muestras incompletas.

Series temporales a nivel de red

A la hora de implementar el código para la generación de las series temporales del modelo a nivel de red, se extrajeron características diferentes a los valores de temperatura; se extrajeron características referidas al tráfico de datos generados en el entorno IoT utilizado.

Las características escogidas para la generación de las muestras de la serie temporal son las siguientes:

- **pck_in:** Indica el número de paquetes entrantes al broker.
- **pck_out:** Indica el número de paquetes salientes del broker.
- **num_bytes:** Indica el número de bytes que ocupan todo el conjunto de paquetes enviados y recibidos por el broker.
- **prot:** Indica el número de protocolos diferentes utilizados para el envío y recepción de paquetes.
- **num_puertos_diferentes:** Indica el número de puertos diferentes utilizados para el envío y recepción de paquetes.
- **num_ips_diferentes:** Indica el número de direcciones IP diferentes utilizados para el envío y recepción de paquetes.

- **min(tiempo_entre_paquetes):** Indica el tiempo mínimo que ha transcurrido entre un paquete saliente del broker y el siguiente.
- **max(tiempo_entre_paquetes):** Indica el tiempo máximo que ha transcurrido entre un paquete saliente del broker y el siguiente.
- **media(tiempo_entre_paquetes):** Indica el tiempo medio que ha transcurrido entre un paquete saliente del broker y el siguiente.

En comparación con el modelo a nivel de aplicación, esta vez no hay que esperar a que lleguen los paquetes de los sensores correspondientes, sino que cada cada 5 minutos exactos se generan las muestras de las series temporales. De esta manera se optiene información útil para la generación de las muestras. El procedimiento se resume en los siguientes pasos:

```
1  while (llegan paquetes) loop:
2
3      if (han pasado 5 minutos) entonces
4
5          generarTimeSeries( pck_in, pck_out, num_bytes, prot, num_puertos_diferentes,
       num_ips_diferentes, min(tiempo_entre_paquetes), max(tiempo_entre_paquetes) ,media(
       tiempo_entre_paquetes))
6
7      else
8
9          num_bytes += bytes(paquete)
10
11         if hasLayer(TCP) and not hasLasyer(MQTT) entonces
12             prot++
13         else if hasLayer(UDP) and not hasLayer(NTP) entonces
14             prot++
15
16         if (ip_destino == ip_broker) entonces
17
18             pck_in++
19
20         else
21             pck_out++
22
23         insertar(tiempo_entre_paquetes, tiempo_paq_recibido - tiempo_paq_anterior_recibido)
```

Se parte de un bucle infinito, que únicamente acabará cuando se dejen de recibir mensajes. Partiendo de esta base, la condición principal para generar una muestra, es conocer en todo momento si han pasado 5 minutos desde la última vez que se generó la última muestra.

Si todavía no han transcurrido los 5 minutos fijados, entonces se procede a la recogida de la información necesaria sobre los paquetes. Primeramente aumentamos en uno el

número de paquetes totales. Más tarde acumulamos el número total de bytes sumándole los bytes que ocupa el nuevo paquete. A continuación, comprobamos si se debe aumentar la variable que indica el número de protocolos utilizados en dicha ventana de 5 minutos. Al analizar los datos, resultó que únicamente se utilizaban los protocolos MQTT dentro de TCP y NTP dentro de UDP; por lo que la utilización de cualquier otro protocolo podría indicar una anomalía. Posteriormente debemos comprobar si dicho paquete es un paquete que entra o sale del broker, mirando su dirección IP destino. Para finalizar, se guardan los intervalos de tiempo que han transcurrido entre la llegada del paquete actual y del paquete anterior.

4.1.2. Creación de los modelos

Ahora que se dispone de las muestras de las series temporales necesarias, el siguiente paso lógico es la creación de los modelos. Se crearán dos modelos diferentes, uno a nivel de aplicación y otro a nivel de red. Estos se generan una única vez al principio del programa y luego se van actualizando a medida que van recibiendo las correspondientes muestras de las series temporales.

Todos los modelos utilizan el mismo algoritmo de detección de anomalías, concretamente HST (Half-Space Trees). Como ya se ha comentado en el capítulo 3, dicho algoritmo trabaja con valores entre 1 y 0. Sin embargo, los valores de las series temporales superan claramente dichos valores. Por dicha razón, la primera parte de la creación de los modelos es normalizar los valores de las series temporales para que el algoritmo funcione correctamente. Existe una función incluida dentro de la librería River que se encarga de normalizar los valores de manera automática. Este proceso se detallará más adelante.

Para ajustar los valores de la parametrización se realizaron diferentes pruebas variando el número de árboles, la altura de los árboles, el tamaño de la ventana y la semilla. Tras analizar los resultados se observó que se obtenían buenos resultados con la parametrización siguiente:

- **Número de árboles:** 10.
- **Altura:** 8.
- **Tamaño de la ventana:** 30.
- **Semilla:** 42.

Una vez establecida la parametrización se puede proceder a dar el siguiente paso, la creación de los detectores de anomalías. Para ello simplemente se utiliza la función HST que actúa sobre los valores de los parámetros que se han descrito en el apartado anterior. Sin embargo, llamar a la función de la librería River no es suficiente. Se necesitará de la creación de un *PipeLine* mediante la función *Pipeline* y llamar a la función *MinMaxScaler*, para que el detector normalice directamente entre 0 y 1 las características que se le van a pasar.

Tanto para el modelo a nivel de aplicación como a nivel de red, disponemos de muestras mono-variadas y multi-variadas. Por lo que para ambos casos se necesitará la creación de un diccionario, donde cada una de las características sea una clave del mismo. En particular, para las muestras mono-variadas se creará un diccionario con una única clave ya que solo disponen de una característica. En el caso de las muestras multi-variadas se creará un diccionario que tenga el mismo número de claves que características. El valor asociado a cada clave coincidirá con el valor de dicha característica.

Tras generar el modelo, se debe entrenar a medida que las muestras de las series temporales van llegando. Primeramente se intenta predecir la puntuación de anomalía que contiene la muestra que se le ha pasado al modelo, y acto seguido se entrena con la misma. Un resumen del procedimiento se recoge en el siguiente pseudocódigo:

```
1
2  model = compose.Pipeline( MinMaxScaler(), HalfSpaceTrees(n_trees=10,height=8,window_size=30,seed
       =42))
3
4  while sigan llegando muestras:
5
6      #Creación del diccionario
7      features = {}
8      for valor_caracteristica in muestra_serie_temporal:
9          features[f'x{idx:02d}'] = float(valor_caracteristica)
10     anomaly_scores.append(model.score_one(features))
11     model = model.learn_one(features)
```

Es importante establecer dicho orden ya que en caso contrario se estaría haciendo “trampa”. Como el modelo todavía no se ha entrenado con las muestras de las series temporales, el modelo devolverá una puntuación de anomalía llamando a la función de la librería River denominada *score_one* pasándole como parámetro el diccionario creado. Entonces devolverá una puntuación de anomalía realista. Después se entrena al modelo y se actualiza el mismo. De manera que el modelo no para de aprender y cada vez que entrena con muestras se genera uno nuevo mediante la función *learn_one* y pasándole como paráme-

tro el diccionario creado. Sin embargo, si se hiciera al revés, es decir; entrenar primero y luego intentar predecir la anomalía mediante una puntuación, al conocer el modelo y las muestras (diccionario con las características), la puntuación no sería realista, sino ventajista.

4.2. Análisis de los resultados

Antes de comenzar con el análisis, cabe destacar que el algoritmo de detección de anomalías utilizado aprende de los valores que son normales de un conjunto pequeño de muestras (200- 300). Como se ha visto, al no existir ningún tipo de anomalía etiquetada en el dataset los dos primeros días (lunes y martes), el modelo aprende eficazmente del espectro de valores normales. Además este algoritmo, como todos los de detección de anomalías, te devuelven una puntuación de anomalía. Dicha puntuación está comprendida entre 0 y 1. Por lo tanto, hay que establecer un umbral a partir del cual se va a considerar que una muestra de la serie temporal es una anomalía. Para ambos modelos, tanto a nivel de aplicación como a nivel de red, se ha establecido el umbral en 0.95. Sin embargo, en las tablas que se mostrarán más adelante también se incluyen los resultados del detector con umbrales de 0.85, 0,90 y 0.95, con objeto de facilitar el análisis de los resultados. Aquellas muestras cuya puntuación de anomalía supere el umbral establecido se considerarán anómalas y el resto se tomarán como muestras normales.

Es conocido que todos los algoritmos de detección de anomalías tienen mayor eficacia cuando las anomalías vienen separadas y son infrecuentes. En el caso del dataset empleado en este proyecto, las anomalías vienen agrupadas. Esto provoca que el algoritmo sea capaz de detectar correctamente las primeras anomalías. Como el modelo es un modelo dinámico, va aprendiendo con cada muestra, por lo que si recibe muchas muestras anómalas juntas, el modelo entenderá ahora que dichas muestras son cada vez más normales y las puntuaciones asociadas disminuirán. Después, al volver a encontrarse con muestras normales, volverán a subir las puntuaciones de anomalías de las muestras, ya que difieren de las últimas 15 o 20 muestras con las que estaba aprendiendo el modelo y que eran anómalas. Esta situación de falsos positivos se va corrigiendo a medida que van entrando más y más muestras normales, y el modelo se vuelve otra vez eficaz, otorgando valores altos únicamente a las anomalías.

Como se ha comentado en el capítulo 2, el dataset con el que se ha trabajado en el desarrollo de este proyecto, es un dataset etiquetado. Aunque se ha optado por desarrollar

modelos no supervisados, al realizar el análisis es una gran ventaja contar con los datos etiquetados para verificar el correcto funcionamiento del modelo.

Los datos pueden, de manera normal, tener ruido (una muestra solitaria o dos, o las que sean que se desvían mucho de las anteriores), que no es necesariamente una anomalía. Este ruido se pudo observar claramente en los resultados de los modelos por lo que se procedió a realizar un filtrado (post-processing). Dado que todas las anomalías incluídas en el dataset vienen en conjuntos de 7 ó más muestras, se decidió considerar como ruido y eliminar aquellas muestras con un valor de anomalía superior a 0.95 que no forme parte de una secuencia de 5 muestras anómalas.

A continuación se van a definir varios conceptos que ayudarán a analizar los resultados: sensibilidad, especificidad y tasas de falsos positivos y falsos negativos.

Se definen como verdaderos positivos VP, las anomalías reales detectadas y como falsos negativos FN, anomalías reales que no han sido detectadas. Analogamente se definen VN como los verdaderos negativos (muestras normales) y FP como falsos positivos, que son muestras normales detectadas como anomalías. Las anomalías totales serán las reales detectadas más las no detectadas VP+FN. De acuerdo con esto se define:

1. Sensibilidad (equivalente a la tasa de positivos verdaderos): Proporción de muestras positivas VP que están bien detectados por la prueba (anomalías reales detectadas). La definición matemática es:

$$S = VP/(VP+FN)$$

2. Especificidad (también llamada tasa de verdaderos negativos): proporción de muestras negativas VN (casos normales) que son bien detectados por la prueba. La definición matemática es:

$$E = VN/(VN+FP)$$

3. Tasa de falsos positivos (TFP): Proporción de muestras negativas que la prueba detecta como positivos:

$$TFP = 1 - Especificidad.$$

4. Tasa de falsos negativos (TFN): Proporción de muestras positivas que la prueba detecta como negativos.

$$TFN = 1 - Sensibilidad$$

4.2.1. Modelos a nivel de aplicación

Modelo multi-variado completo

Con el fin de crear un modelo eficiente, se han desarrollado tres detectores diferentes a nivel de aplicación. El primer detector de anomalías a desarrollar fue uno para todos los sensores existentes, es decir, uno que utiliza una serie temporal multi-variada, donde las características son los valores de temperatura de todos los sensores. Una muestra incluye datos de 16 sensores aunque puede darse el caso en el que falte algún dato de alguno de ellos. Pero como ya se ha indicado anteriormente, esto no supone un problema. Los resultados obtenidos están representados en la Figura 4.1:

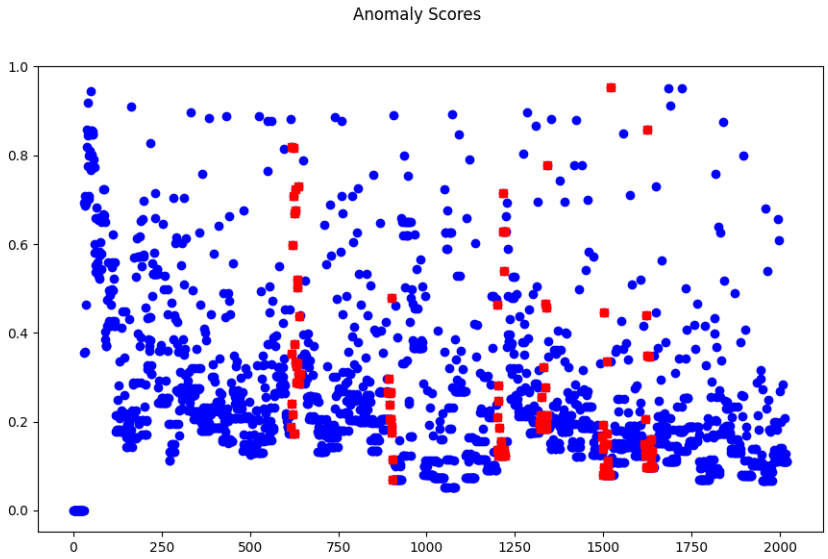

Figura 4.1: Gráfico de puntuaciones de anomalía para el caso del modelo multi-variado completo

Los puntos azules indican los valores de temperaturas que provienen de muestras normales y los puntos rojos aquellos valores de temperaturas que provienen de muestras que sabemos que son anómalas porque lo indica el fichero *CSV* que forma parte del dataset. El eje Y indica la puntuación de anomalía de 0 a 1 y el eje X indica el número de la muestra

en la serie temporal (tiene 2021 muestras). Claramente se puede observar que la mayor parte de las puntuaciones de anomalías se encuentran en franjas de valores bajos, entre 0.15 y 0.4. Sin embargo, lo que realmente interesa es que detecte las anomalías reales (puntos rojos), asociándolas una puntuación alta (mayor o igual que 0.95). La tabla de resultados se muestra en la Figura 4.2:

	Umbral 0.95	**Umbral 0.90**	**Umbral 0.85**
Número de anomalías totales detectadas	0	25	73
Número de anomalías reales	136	136	136
Número de anomalías reales detectadas-VP	0	1	2
Número de anomalías reales no detectadas-FN	136	135	134
Número de muestras totales detectadas como normales	1885+136	1861+135	1814+134
Número de muestras normales	1885	1885	1885
Número de muestras normales detectadas como normales-VN	1885	1861	1814
Número de muestras normales detectadas como anómalas-FP	0	24	71
Número de muestras totales	2021	2021	2021
Sensibilidad S=VP/(VP+FN)	0	0.01	0.01
Especificidad E= VN/(VN+FP)	1	0.98	0.96
TFP: 100-Especificidad Tasa de falsos positivos	0	0.02	0.04
TFN: 100-Sensibilidad Tasa de falsos negativos	1	0.99	0.99

Figura 4.2: Tabla resultados del análisis para el modelo multi-variado completo

Sin embargo, mediante este modelo hemos podido observar que la cantidad de falsos positivos era superior a la de positivos verdaderos. Se concluyó que el modelo funcionaba mal con las muestras de la serie temporal utilizada. Antes de proseguir, había que identificar la base del problema, qué estaba pasando con las muestras de la serie temporal para que el detector no fuera capaz de descubrir casi ninguna de las anomalías reales. Mirando bien todas las muestras, y comparando muestras normales con muestras anómalas reales, se vio que los valores de las temperaturas de todos los sensores apenas diferían entre sí, y tampoco diferían con los rangos de los valores salvo en dos o tres sensores por muestra.

Esto explica por qué el modelo era incapaz de detectar las anomalías reales, aunque todavía quedaba entender la razón de la cifra tan alta de falsos positivos. La gran mayoría de

estos se encuentran en las primeras 300 muestras de la serie temporal. Algo entendible, ya que el modelo está tratando de entender los datos con los que trabaja, y cuando alguno de los valores de la muestra cambia ligeramente, la puntuación de anomalía aumenta. Por lo que se puede decir que el modelo realmente está siendo eficaz. A partir de las 300 muestras, se puede ver como los valores van bajando drásticamente hasta mantenerse por debajo de una puntuación de anomalía de 0.6 (79 %). Esta es otra indicación de que el modelo está trabajando eficazmente, y el problema no es el algoritmo de detección si no la serie temporal y los datos que hay en ella.

Modelo multi-variado con muestras por cliente

Por todas las razones anteriores se procedió a desarrollar un nuevo modelo, en el que las series temporales iban a contener otros datos. En este segundo modelo, la serie temporal multi-variada contiene los valores de temperatura de cada uno de los cuatro sensores incluidos en cada máquina del escenario virtual. El análisis de los resultados de este modelo se realizó en base a los 4 sensores (131,132,133,134) incluidos en la máquina con dirección IP 10.6.56.41 debido a que era la máquina que contenía el mayor número de anomalías. Los resultados se muestran en la Figura 4.3:

Figura 4.3: Gráfico de puntuaciones de anomalía del modelo multi-variado con muestras por cliente

Aunque a primera vista no se aprecia gran diferencia, el modelo con la nueva serie temporal es capaz de detectar mucho mejor las anomalías reales. La media de puntuación de anomalía de los paquetes normales es 0.39, un valor aceptable. Como ya hemos mencionado al principio del capítulo, se considera anomalía cuando la puntuación de anomalía devuelta por el modelo es igual o superior a 0.95. Es por ello que si se analiza detalladamente ese rango de valores, se puede observar que el modelo funciona mejor. Eso se ve claramente reflejado en la Figura 4.4, donde se observa que la gran mayoría de las muestras puntuadas con valores superiores a 0.95 son anomalías reales. La Figura 4.4 contiene sólo las muestras con puntuaciones iguales o superiores a 0.95.

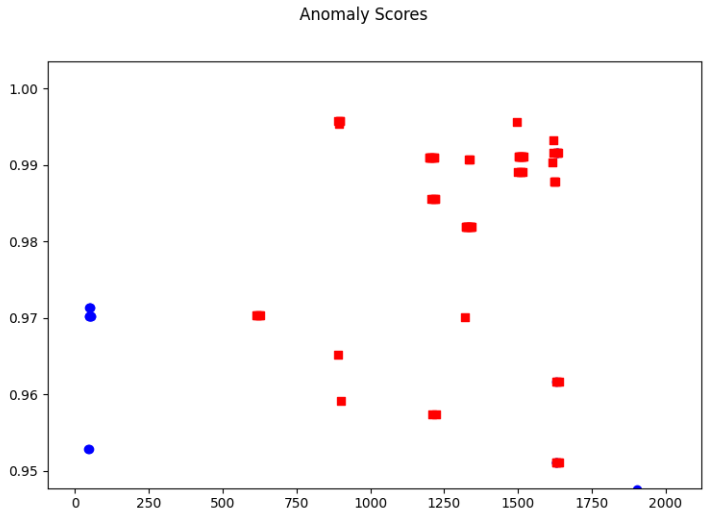

Figura 4.4: Ampliación del rango de valores del gráfico de puntuaciones de anomalía del modelo multi-variado con muestras por cliente

El número de falsos positivos ha disminuido claramente (a 10 en la figura), a la vez que el número de positivos verdaderos ha aumentado considerablemente. Los resultados se resumen en la tabla de la Figura 4.5:

	Umbral 0.95	**Umbral 0.90**	**Umbral 0.85**
Número de anomalías totales detectadas	74	110	132
Número de anomalías reales	136	136	136
Número de anomalías reales detectadas-VP	64	85	90
Número de anomalías reales no detectadas-FN	72	51	46
Número de muestras totales detectadas como normales	1875+72	1860+51	1843+46
Número de muestras normales	1885	1885	1885
Número de muestras normales detectadas como normales-VN	1875	1860	1843
Número de muestras normales detectadas como anómalas-FP	10	25	42
Número de muestras totales	2021	2021	2021
Sensibilidad S=VP/(VP+FN)	0.47	0.62	0.66
Especificidad E= VN/(VN+FP)	0.99	0.98	0.97
TFP: 100-Especificidad Tasa de falsos positivos	0.01	0.02	0.03
TFN: 100-Sensibilidad Tasa de falsos negativos	0.53	0.38	0.34

Figura 4.5: Tabla de los resultados del análisis del modelo multi-variado con muestras por cliente

El modelo detecta correctamente la mitad de las anomalías reales, 64 de 136. La razón es que los valores de los sensores de temperatura de cada máquina están correlados entre sí pero no con los sensores de otras máquinas. En la Figura 4.6 se observa este comportamiento para dos muestras diferentes en la que los sensores de cada máquina están agrupados de cuatro en cuatro.

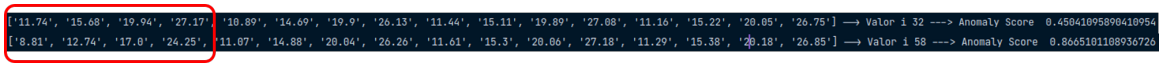

Figura 4.6: Muestras de la serie temporal

En el modelo anterior se entrenaba con las temperaturas de todos los sensores y en este sólo con los de una máquina, los marcados en rojo. Se observa que todos los sensores cambian a la vez.

Modelo con muestras mono-variadas

Por último se desarrolló un nuevo modelo, uno por cada sensor, buscando confirmar las conclusiones sacadas con los dos modelos anteriores. En este caso las series temporales iban a ser mono-variadas ya que solo disponen de una única característica, el valor de temperatura del sensor correspondiente. El sensor escogido para realizar el análisis ha sido aquel identificado como 134. Esto se debe a que los cuatro sensores (131,132,133,134) en la máquina con la dirección IP 10.6.56.41 tienen un número de anomalías muy superior a cualquier otro sensor de cualquier otra máquina. Tras entrenar y probar el modelo con la serie temporal, se obtuvieron los resultados mostrados en la Figura 4.7:

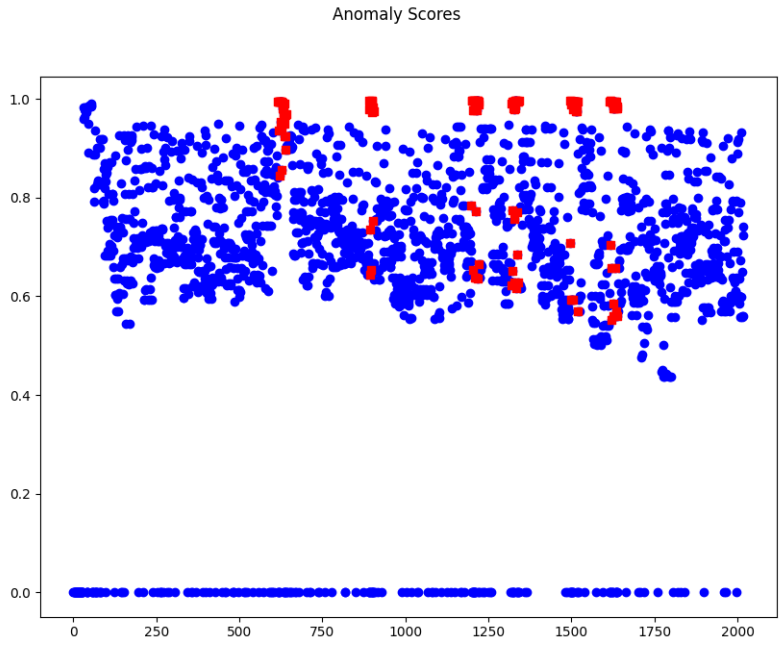

Figura 4.7: Gráfico de puntuaciones de anomalía del modelo con muestras mono-variadas

Estos resultados indican que el modelo trabaja bien y es capaz de detectar la gran mayoría de anomalías reales. De hecho, si excluimos el periodo de entrenamiento del comienzo de cada serie, el modelo no da ningún falso positivo; se puede observar claramente en la Figura 4.8 donde se ha ampliado la zona de valores superiores a 0.95.

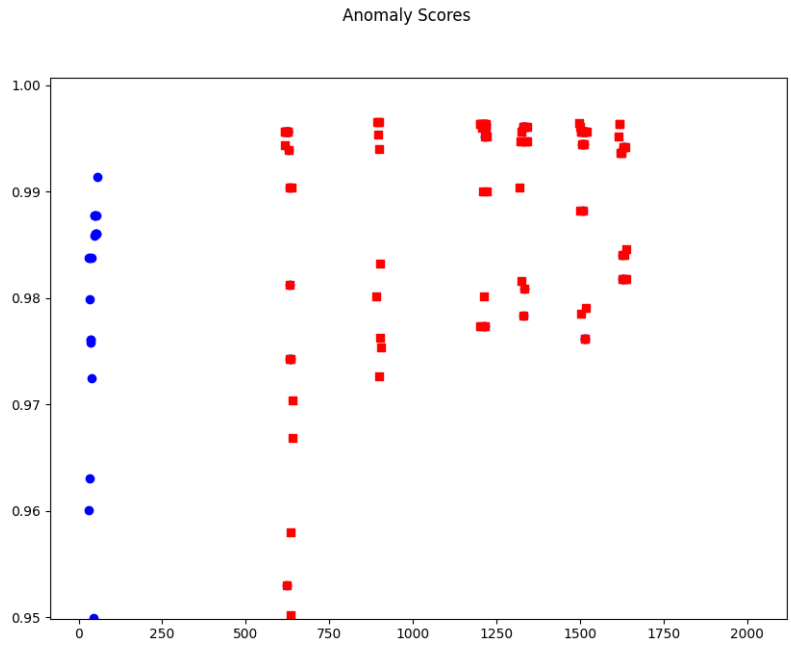

Figura 4.8: Ampliación del rango de valores del gráfico de puntuaciones de anomalía del modelo con muestras mono-variadas

Uno de los cambios notables frente al modelo anterior es que la media de las puntuaciones de anomalía de las muestras normales se sitúa alrededor de 0.74. Un valor bastante alto en comparación con el anterior modelo, 0.39. La razón estriba en que las diferencias de las temperaturas del sensor entre muestras normales y anómalas son pequeñas y al modelo le cuesta distinguir, asignando valores medio-altos a muestras tanto normales como anómalas aunque claramente reserva los valores más altos para las muestras realmente anómalas. La tabla de resultados se recoge en la Figura 4.9:

	Umbral 0.95	Umbral 0.90	Umbral 0.85
Número de anomalías totales detectadas	116	271	396
Número de anomalías reales	136	136	136
Número de anomalías reales detectadas-VP	96	100	102
Número de anomalías reales no detectadas-FN	40	36	34
Número de muestras totales detectadas como normales	1863+40	1712+36	1589+34
Número de muestras normales	1883	1883	1883
Número de muestras normales detectadas como normales-VN	1863	1712	1589
Número de muestras normales detectadas como anómalas-FP	20	171	294
Número de muestras totales	2019	2019	2019
Sensibilidad S=VP/(VP+FN)	0.70	0.73	0.75
Especificidad E= VN/(VN+FP)	0.98	0.90	0.84
TFP: 100-Especificidad Tasa de falsos positivos	0.02	0.10	0.16
TFN: 100-Sensibilidad Tasa de falsos negativos	0.30	0.27	0.25

Figura 4.9: Tabla resultados del análisis del modelo con muestras mono-variadas

Por otra parte, el modelo anterior es más estable porque son cuatro las lecturas de los sensores las que varían a la vez en un episodio anómalo. En el último modelo con un único sensor aumenta la especificidad aunque también la desviación estándar y por tanto los errores.

Una vez analizados los resultados de los modelos creados, se decidió por generar un modelo por cada sensor. Esta conclusión viene dada por los datos de las tablas asociadas a los modelos. En ellas se puede observar con nitidez, que la sensibilidad del tercer modelo (1 por sensor) es superior a la del segundo modelo (uno por máquina). El segundo modelo tiene una especificidad que varía poco con el umbral y una tasa de falsos positivos claramente menor. Sin embargo, el segundo modelo tiene un tasa de falsos negativos alta, esto implica que es capaz de detectar el 47 % de las anomalías reales frente al 70 % que es capaz de detectar el tercer modelo. Concluimos que el tercer modelo identifica la gran mayoría de las anomalías reales y de todas aquellas detectadas como anomalías, el 98 % son reales.

4.2.2. Modelo a nivel de red

En el caso del modelo a nivel de red, se ha tenido que incluir la hipótesis del correcto funcionamiento del modelo. Esto se debe a que no se dispone de información sobre los paquetes interceptados (eliminados), por lo que no se puede validar la efectividad del modelo a la hora de detectar dicho ataque. Sin embargo, sí se dispone de la información necesaria sobre los paquetes duplicados. Respecto a la modificación de temperaturas de los sensores, es algo de lo que se encarga el modelo a nivel de aplicación ya que trabaja con valores de temperatura. En el caso del modelo a nivel de red, es imposible detectar dichos paquetes. Por lo que los ataques que se pueden validar mediante este modelo son aquellos que han duplicado los paquetes.

Después de entrenar y probar el modelo con la muestras multi-variadas de la serie temporal correspondiente se obtuvieron los resultados mostrados en la Figura 4.10:

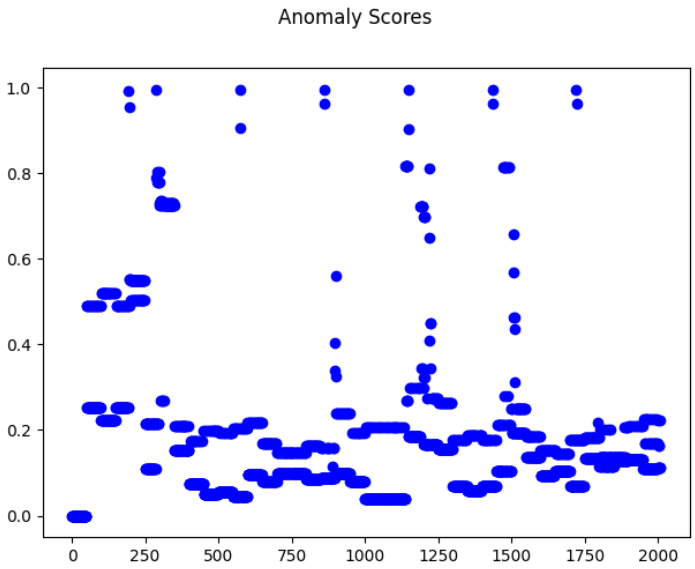

Figura 4.10: Gráfico de puntuaciones de anomalía del modelo a nivel de red

Para validar el correcto funcionamiento del modelo, se analizaron una a una y manualmente todas aquellas muestras que superaban la puntuación de anomalía 0.75 (exactamente 80 muestras).

El análisis revela que el 75 % de las muestras tenían en común un incremento/decremento en el valor de varias características de las muestras de la serie temporal. Dichas caracte-

rísticas eran el número de bytes totales de la ventana, y la suma de los paquetes entrantes y salientes del broker.

El restante 25 % de las muestras tenían en común un decremento en el valor de la característica que indica el tiempo transcurrido entre la recepción de paquetes en el broker junto con un pequeño incremento en el número de bytes por ventana y la suma del número de paquetes entrantes y salientes del broker.

Se vió de los datos que aquellas muestras que presentaban un incremento del valor de las características correspondían a un ataque de duplicación.

El 85 % de las muestras que han recibido un valor de anomalía superior a 0.75 son anomalías reales. Dentro del 85 % de las muestras, el 80 % son anomalías relacionadas con ataques de duplicación y el 20 % con ataques de interceptación.

Al no disponer de los ataques de interceptación etiquetados en el dataset, no se ha podido realizar una evaluación automatizada del comportamiento real de este modelo. Sin embargo, para comprobar los paquetes interceptados, se procedió a realizar un gráfico del número de paquetes enviados por cada uno de los clientes MQTT el martes, ya que es el único día el que solo se realizan este tipo de ataques. El resto de días contienen tanto ataques de duplicación como de intercepción, lo que dificultan mucho la detección de ataques de interceptación, ya que no están etiquetados. Con objeto de buscar anomalías no etiquetadas se recurrió a observar la falta de uniformidad en el tráfico de los datos en las capturas, como se puede ver en la Figura 4.11.

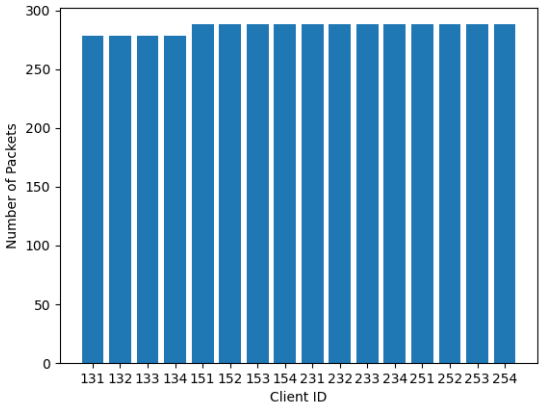

Figura 4.11: Gráfico del número de paquetes enviados por cada cliente MQTT el martes

Se pudo verficar manualmente mediante Wireshark que el 90 % de las muestras con una

puntuación de anomalía por encima de 0.75 del tráfico del martes, eran anomalías relacionadas con ataques de interceptación.

Visualizar las muestras y los resultados obtenidos por el modelo, permite establecer la siguiente hipótesis: El modelo trabaja correctamente para detectar ataques relacionados con duplicación o con interceptación de paquetes. Sin embargo, no es capaz de detectar ataques relacionados con la modificación de los paquetes.

Al disponer de estos dos modelos diferentes, a nivel de aplicación y a nivel de red, el programa desarrollado permite identificar los tres tipos de ataques que buscamos, alteraciones, duplicaciones e interceptaciones. Los resultados de ambos modelos muestran que cuando avisan por alerta de anomalía, dicha anomalía tiene una gran probabilidad de ser una anomalía real, siendo la tasa de falsos negativos muy baja.

4.2.3. Conclusiones técnicas del análisis

- El modelo creado a nivel de aplicación detecta las anomalías ligadas a las modificaciones en las lecturas de los valores de temperatura de los sensores de la red IoT (datacenter).

- El modelo escogido a nivel de aplicación es aquel que trabaja con datos monovariados. Tiene un tasa de falsos positivos inferior al 2 % y una tasa de falsos negativos del 30 %. Lo anterior indica que detecta muchas de las anomalías relacionadas con modificaciones de temperatura además de tener una sensibilidad alta, del 70 % y una especificidad del 98 %. La gran mayoría de las anomalías que detecta son reales.

- Los modelos a nivel de aplicación muestran un comportamiento ambivalente. La sensibilidad disminuye a medida que aumentamos el número de sensores en consideración, es decir, disminuye el número de anomalías reales detectadas, desde un 75 % en el caso de un único sensor hasta un 47 % para 4 sensores. Por el contrario, la tasa de falsos positivos es mayor cuando analizamos cada sensor independientemente, es decir uno a uno.

- El modelo creado a nivel de red detecta las anomalías ligadas a intrusiones de red, como la interceptación o duplicación de paquetes. El 85 % de las muestras con un valor de anomalía superior a 0.75 son anomalías reales, por lo que tiene una tasa

de falsos positivos baja. Respecto a las anomalías relacionadas con ataques de interceptación es capaz de detectarlas en un 90 %. Sin embargo la detección de las mismas se dificulta cuando hay tanto ataques de interceptación como de duplicación. No se ha podido validar la eficacia del modelo en la detección de ataques de interceptación cuando se producen a la vez ataques de duplicación.

5. CAPÍTULO

Conclusiones

El desarrollo de este TFG me ha permitido tanto profundizar sobre los conocimientos impartidos en la carrera como adquirir nuevos concocimientos sobre diversos temas de interés. Entre ellos me gustaría destacar:

- Stream Machine Learning junto con algunos de los algoritmos que utiliza como por ejemplo el HST.
- La creación de modelos de aprendizaje automático junto al preprocesado y postprocesado de los datos, así como la generación de las series temporales.
- Redes IoT, su funcionamiento, sus protocolos (MQTT) y la estructura de los paquetes que utilizan dichos protocolos.
- Detección de anomalías e investigación de diferentes ataques.
- Herramientas para la extracción de datos contenidos en paquetes (Scapy).

El trabajo realizado en este proyecto ha permitido el desarrollo de dos tipos de modelos, uno a nivel de aplicación y otro a nivel de red que permiten la detección de anomalías en una red IoT mediante algoritmos basados en aprendizaje automático no supervisado en streaming. Los modelos elaborados permiten la detección de ataques como duplicaciones, intercepciones y modificaciones en un red IoT de sensores. El dataset utilizado tiene dos limitaciones importantes; por una parte el conjunto de datos es pequeño y por otra solo tiene etiquetados dos de los tres ataques (duplicación e interceptación). A pesar de ello se

han obtenido buenos resultados. El modelo propuesto a nivel de aplicación tiene un tasa de falsos positivos inferior al 2 % y una tasa de falsos negativos del 30 %.

Finalmente me gustaría destacar que hay algunos aspectos pendientes de abordar:

- Probar los detectores de anomalías con un dataset más diverso, con diferentes tipos de ataques y mucho más grande.
- Realizar Grid Search para la optimización de los parámetros del detector de anomalías y así lograr los mejores resultados posibles.
- Investigar sobre otros detectores de anomalías diferentes de los HST.
- Realizar un proceso de selección de características para el modelo basado en datos de red.

6. CAPÍTULO

Planificación

Este capítulo sirve para describir como ha sido la gestión del trabajo, desde la planificación y descripción de tareas, hasta el análisis de riesgos.

Me he apoyado en las directrices del Project Management Institute que figuran en la tabla 4.1 de la sexta edición del libro "A guide to the Project managements body of knolwledge (PMBOK guide)" [PMB, 2017].

6.1. Descripción de las tareas

Las tareas se han distribuido en cuatro bloques principales donde cada uno de ellos se divide en otros dos bloques específicos como se puede ver en el diagrama de bloques recogido en la Figura 6.1. Cada bloque específico agrupa diferentes tareas a realizar.

En la siguiente lista se muestran las tareas que contiene cada bloque:

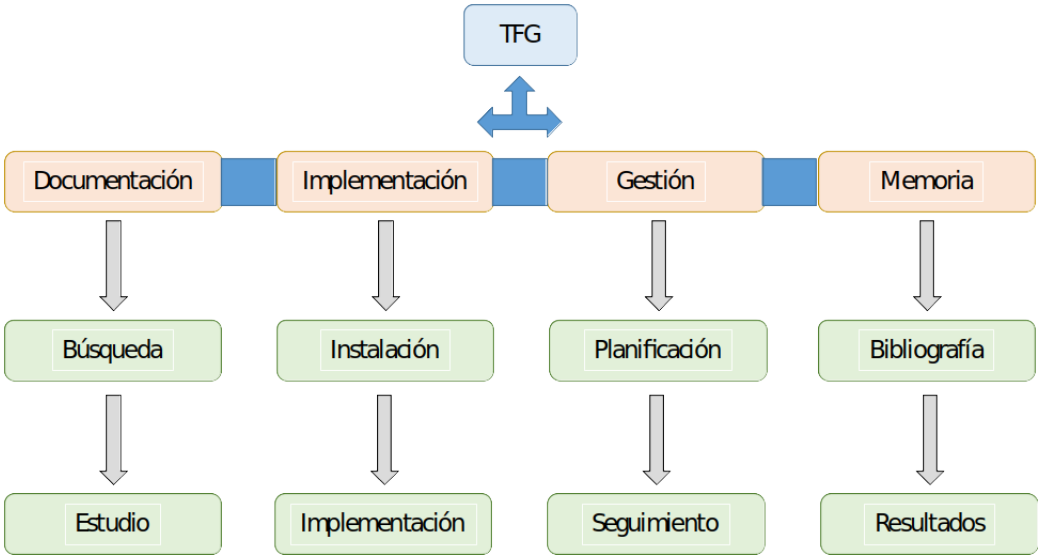

Figura 6.1: Diagrama de bloques de las tareas del proyecto

Bloque de Documentación.

1. **Búsqueda**
   - B1. Crear un listado de los temas a buscar.
   - B2. Buscar libros y artículos relacionados con el propósito del proyecto.
   - B3. Leer, estudiar, catalogar y/o desechar los artículos y libros encontrados.
2. **Estudio**
   - L1. Repaso de conceptos necesarios para la realización del proyecto.
   - L2. Leer y comprender la idea general de cada artículo o libro.
   - L3. Leer y comprender las técnicas necesarias para la implementación.

Bloque de Implementación.

1. **Instalación de librerías**
   - I1. Instalar y leer la documentación de Wireshark.
   - I2. Instalar y leer la documentación de Scapy.
   - I3. Instalar y leer la documentación de River.
   - I4. Realizar algunas pruebas con las librerías.

2. **Implementación**

    - M1. Implementación del programa con Python utilizando las librerías mencionadas.
    - M2. Análisis de los paquetes con Wireshark.
    - M3. Tratamiento de los datos con Scapy.
    - M4. Definición de figuras de mérito del algoritmo de detección

Bloque de Gestión.

1. **Planificación**

    - P1. Identificación de las tareas a realizar, tiempos y fechas clave.
    - P2. Actualización de la planificación, en caso necesario.

2. **Seguimiento y control**

    - S1. Reuniones con el tutor y control del progreso del trabajo.
    - S2. Modificación de tareas, herramientas y/o técnicas si es necesario.
    - S3. Entrega en plazo de la memoria del TFG.

Bloque de Memoria.

1. **Bibliografía**

    - D1. Documentación y aprendizaje.

2. **Resultados**

    - R1. Escribir el capítulo de implementación.
    - R2. Escribir el capítulo de resultados y conclusiones.
    - R3. Revisión y corrección de la memoria.

6.2. Cronograma

El periodo de realización de las tareas abarca 24 semanas, desde el día 11 de enero al 20 de junio, y se estiman unas 350 horas de dedicación total. La génesis del proyecto comenzó un poco antes, en julio de 2020. Durante los meses de verano profundicé en la lectura y estudio de conceptos de seguridad de los paquetes. Este periodo de formación no se recoge en la planificación. En el siguiente diagrama se muestra la planificación semanal de tareas.

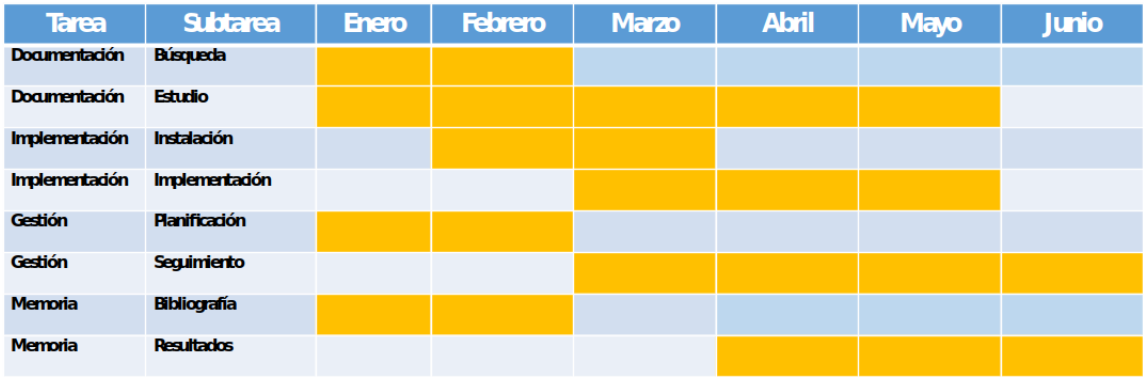

Figura 6.2: Diagrama de Gantt del proyecto

En la siguiente tabla se muestra el tiempo estimado en horas para cada bloque de tareas:

Tarea	Subtarea	Dedicación Subtarea (horas)	Dedicación tarea (horas)
Documentación	Búsqueda	20	75
Documentación	Estudio	55	
Implementación	Instalación	25	100
Implementación	Implementación	75	
Gestión	Planificación	15	35
Gestión	Seguimiento	20	
Memoria	Bibliografía	65	140
Memoria	Resultados	75	
Total			350

Figura 6.3: Horas dedicadas a cada tarea-subtarea

6.2.1. Hitos y entregables

Hito	Descripción	Fecha
H.1	Proyecto iniciado y definido	18/01/2021
H.2	Librerías de Machine Learning seleccionadas	04/02/2021
H.3	Obtención de bases de datos	22/03/2021
H.4	Desarrollo básico del algoritmo finalizado	31/05/2021
H.5	Algoritmo de detección de anomalías	24/05/2021
H.6	Proyecto finalizado	21/06/2021

Figura 6.4: Hitos del proyecto

Entregable	Descripción	Fecha
E.1	Programas instalados	28/02/2021
E.2	Bases de datos ordenada	04/04/2021
E.3	Clasificador de anomalías	01/05/2021
E.4	Optimización del clasificador de anomalías	31/05/2021
E.5	Valores de las estadísticas de la anomalías obtenidos	24/05/2021
E.6	Código final del algoritmo	30/05/2021
E.7	Memoria del proyecto	21/06/2021

Figura 6.5: Entregables del proyecto

6.3. Análisis de riesgos

En este apartado en primer lugar, se analizarán los riesgos a los que se verá expuesto el proyecto, clasificándolos posteriormente mediante la matriz Probabilidad-Impacto. Además, se elaborarán medidas para evitar y minimizar el impacto de los riesgos identificados.

6.3.1. Identificación de los riesgos y medidas de contingencia

- **Demoras (R1)** Los retardos a la hora de entregar las tareas correspondientes podrían deberse a diversos factores. En primer lugar, podría darse el caso en el que el proyectista no asimilara del todo bien los conocimientos explicados por el director del proyecto, teniendo que dedicarle más tiempo del debido a realizar un estudio

más exhaustivo por su cuenta. Además, debido a factores externos en el ámbito académico del proyectista, podría darse la situación en la que éste no pudiera dedicar las horas requeridas a las tareas indicadas en la planificación, posponiendo, en consecuencia, su fecha de entrega. Un ejemplo de riesgo a priori es el causado por la pandemia del COVID-19. Podría darse el caso de que cerraran la universidad, con lo que el acceso a los recursos bibliográficos o a las reuniones de seguimiento con el tutor estaría parcialmente comprometido. Por otra parte, bien el proyectista bien el tutor o ambos, podrían enfermar con el consiguiente retraso en la ejecución del proyecto. Ante estos imprevistos, la solución sería llevar una estricta planificación de las entregas, indicando también las fechas límite en caso de existir una demora.

- **Pérdida de datos (R2)** Podría darse el caso en el que, a causa tanto de errores internos del programa a utilizar como de no actualizar las modificaciones realizadas, el trabajo hecho se pierda parcial o totalmente. Para ello, se considera necesario disponer de varias copias de seguridad en diferentes unidades físicas y en la nube, además de guardar cada poco tiempo todo avance realizado.

- **Dificultades técnicas (R3)** Uno de los posibles riesgos puede residir en que las herramientas utilizadas de *Machine Learning* no aprendan bien a identificar en tiempo real, las anomalías en los paquetes. Esto puede deberse a que no se utilicen correctamente o a las propias limitaciones de las librerías. Es por ello que quizás algunas de las librerías seleccionadas al principio no den el resultado requerido y deban ser sustituidas por otras más eficientes o compatibles con nuestro problema. Otro problema técnico es que las bases de datos de anomalías sean pequeñas, por lo que el proceso de aprendizaje deba realizarse en una muestra pequeña con el consiguiente deterioro de los resultados de predicción y clasificación.

6.3.2. Matriz probabilidad-impacto

La matriz probabilidad-impacto es una herramienta que permite establecer prioridades en cuanto a la atención que se debe prestar a cada riesgo en cuestión, estableciendo una relación entre su probabilidad de aparición y el impacto que supondría el hecho de darse. En la figura 6.6 puede apreciarse que no hay riesgos ante los que se deba actuar de antemano urgentemente. Sin embargo, los riesgos R1 y R2 podrían llegar a suponer algún problema en cuanto a la finalización del proyecto se refiere.

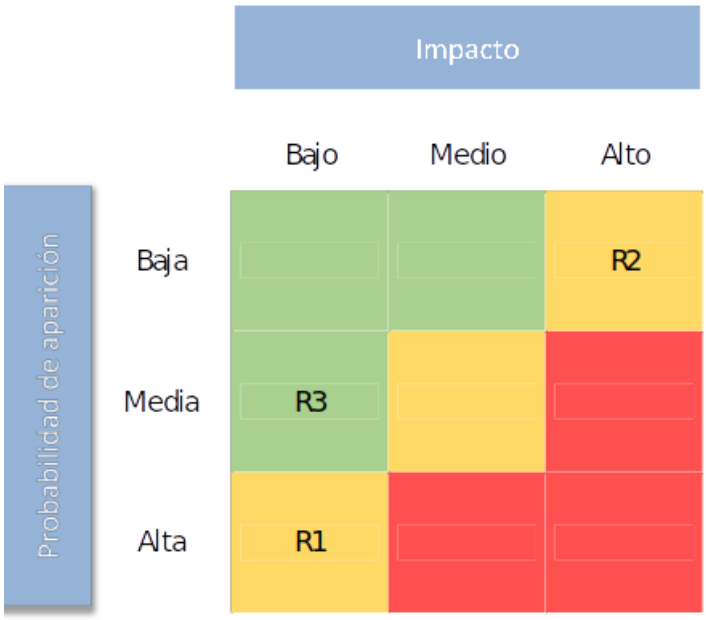

Figura 6.6: Matriz probalidad-impacto

Bibliografía

[tcp, 2010] (2010). TCPDUMP/LIBPCAP public repository. Última vez visitado el 08/05/2021. https://www.tcpdump.org/.

[PMB, 2017] (2017). A guide to the project management body of knowledge / Project Management Institute. Última vez visitado el 22/04/2021. https://book.akij.net/eBooks/2018/March/5abcc35b666f7/a%20guide%20to%20the%20project%20management%20body%20of%20knowledge%206e.pdf.

[mos, 2018] (2018). Eclipse Mosquitto. Última vez visitado el 08/05/2021. https://mosquitto.org/.

[CIT, 2019] (2019). CITIC – Centro de Investigación en Tecnologías de la Información y las Comunicaciones. Última vez visitado el 12/05/2021. https://citic.udc.es/en/home-english-2/.

[MQT, 2020] (2020). MQTT - The Standard for IoT Messaging. Última vez visitado el 25/05/2021. https://mqtt.org/.

[CTO, 2021] (2021). Arlen Nipper, President and CTO. Última vez visitado el 14/05/2021. https://iiot-world.com/our-team/our-contributors/arlen-nipper-president-and-cto/.

[DAD, 2021] (2021). Dataset for anomaly detection in iot networks. Última vez visitado el 04/07/2021. https://github.com/dad-repository/dad. original-date: 2020-04-14T17:37:56Z.

[IoT, 2021] (2021). Internet of things. Última vez visitado el 11/05/2021. https://www.ibm.com/cloud/internet-of-things.

[Jup, 2021] (2021). The jupyter notebook is a web-based interactive computing platform. the notebook combines live code, equations, narrative text, visualizations, interactive dashboards and other media. Última vez visitado el 13/05/2021. https://www.jupyter.org.

[Pyc, 2021] (2021). PyCharm: the Python IDE for Professional Developers by JetBrains. Última vez visitado el 07/05/2021. https://www.jetbrains.com/pycharm/.

[Sci, 2021] (2021). Scikit-multiflow: Machine learning package for streaming data in python. Última vez visitado el 25/05/2021. https://scikit-multiflow.github.io/.

[Wir, 2021] (2021). Wireshark. Última vez visitado el 27/04/2021. https://www.wireshark.org/.

[IDC, 2021] (2021). Worldwide Global DataSphere IoT Device and Data Forecast, 2020–2024. Última vez visitado el 01/05/2021. https://www.idc.com/getdoc.jsp?containerId=US46718220.

[MQT, 2021] (2021). ¿Qué es MQTT? Su importancia como protocolo IoT. Última vez visitado el 02/06/2021. https://www.luisllamas.es/que-es-mqtt-su-importancia-como-protocolo-iot/.

[Biondi, 2018] Biondi, P. (2018). Scapy: Packet crafting for python2 and python3. Última vez visitado el 27/06/2021. https://scapy.net/.

[Halford, 2021] Halford, M. (2021). creme: Online machine learning in Python. Última vez visitado el 15/05/2021. https://github.com/creme-ml/creme.

[Liu et al., 2008] Liu, F. T., Ting, K. M., and Zhou, Z.-H. (2008). Isolation Forest. In *2008 Eighth IEEE International Conference on Data Mining*, pages 413–422, Pisa, Italy. IEEE.

[Lobo, 2020] Lobo, J. L. (2020). How to become an expert in Machine Learning for Data Streams?. Última vez visitado el 25/05/2021. https://towardsdatascience.com/how-to-become-an-expert-in-machine-learning-for-data-streams-3ecbb612b641.

[López-Avila et al., 2019] López-Avila, L., Acosta-Mendoza, N., gago Alonso, A., López-Avila, L., Acosta-Mendoza, N., and gago Alonso, A. (2019). Detección de anomalías basada en aprendizaje profundo: Revisión. Última vez visitado el

20/06/2021. http://scielo.sld.cu/scielo.php?script=sci_abstract&pid=S2227-18992019000300107&lng=es&nrm=iso&tlng=es. Publisher: Universidad de las Ciencias Informáticas.

[Montiel et al., 2020] Montiel, J., Halford, M., Mastelini, S. M., Bolmier, G., Sourty, R., Vaysse, R., Zouitine, A., Gomes, H. M., Read, J., Abdessalem, T., and Bifet, A. (2020). River: machine learning for streaming data in python. Última vez visitado el 29/06/2021. https://riverml.xyz/.

[Richard Coppen, 2020] Richard Coppen, I. C. (2020). OASIS Message Queuing Telemetry Transport (MQTT) TC | OASIS. Última vez visitado el 21/05/2021. https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=mqtt.

[Sanz, 2019] Sanz, M. d. l. F. (2019). Google Colab: Python y Machine Learning en la nube. Última vez visitado el 24/04/2021. https://www.adictosaltrabajo.com/2019/06/04/google-colab-python-y-machine-learning-en-la-nube/.

[Stanford-Clark, 2021] Stanford-Clark, A. J. (2021). Andy Stanford-Clark. Última vez visitado el 17/06/2021. https://en.wikipedia.org/w/index.php?title=Andy_Stanford-Clark&oldid=1027754854. Page Version ID: 1027754854.

[Team, 2020] Team, T. H. (2020). MQTT Client and Broker and MQTT Server and Connection Establishment Explained - MQTT Essentials: Part 3. Última vez visitado el 19/06/2021. https://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment.

[Vigoya et al., 2020] Vigoya, L., Fernandez, D., Carneiro, V., and Cacheda, F. (2020). Annotated Dataset for Anomaly Detection in a Data Center with IoT Sensors. Última vez visitado el 30/06/2021. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7374380/.