UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

TFG del Grado en Ingeniería Informática

AI for IoT security

Presentado por Eduardo Manuel Cabeza López
en Universidad de Burgos — 3 de julio de 2024
Tutores: Rubén Ruiz González y Alejandro Merino Gómez

UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

D. Rubén Ruiz González y Alejandro Merino Gómez, profesores del departamento de Digitalización, área de Ingeniería de Sistemas y Automática.

Exponen:

Que el alumno D. Eduardo Manuel Cabeza López, con DNI 47580573J, ha realizado el Trabajo final de Grado en Ingeniería Informática titulado AI for IoT security.

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección del que suscribe, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, 3 de julio de 2024

Vº. Bº. del Tutor:

D. Rubén Ruiz González

Vº. Bº. del co-tutor:

D. Alejandro Merino Gómez

Resumen

El presente trabajo consiste en el estudio de la posibilidad de proteger redes de dispositivos IoT mediante técnicas de *machine learning*. Para ello se lleva a cabo un estudio y análisis de trabajos relacionados donde los autores aplican diferentes técnicas para predecir posibles ataques a estos dispositivos mediante la clasificación del tráfico de la red. En esta fase se identifican las principales técnicas empleadas y los resultados obtenidos con diferentes modelos de inteligencia artificial y diferentes conjuntos de datos.

Se continua el trabajo tratando de replicar algunos de los resultados obtenidos por los autores, tanto para refutar estos resultados como para determinar la viabilidad de detectar ataques de ciberdelincuentes sobre estos dispositivos en los inicios del ataque y reducir al mínimo los daños ocasionados por el ataque producido.

Con los resultados obtenidos se trata de definir los siguientes trabajos o líneas de investigación que se deberían de llevar a cabo para la implantación de sistemas que utilicen inteligencia artificial para clasificar tráfico de red y detectar ciberataques en tiempo real desde que este comienza a producirse y tomar, en consecuencia, las medidas oportunas.

Descriptores

IA, *machine learning*, ciberseguridad, IoT, detección de anomalías e intrusiones, seguridad en redes.

Abstract

This work consists of a study of the possibility of protecting IoT device networks using machine learning techniques. To this end, a study and analysis of related work are carried out in which the authors apply different techniques to predict possible attacks on these devices by classifying network traffic. In this phase, the main techniques used and the results obtained with different artificial intelligence models and different datasets are identified.

The work continues by trying to replicate some of the results obtained by the authors, both to refute these results and to determine the feasibility of detecting attacks by cybercriminals on these devices at the beginning of the attack and to minimise the damage caused by the attack.

With the results obtained, the aim is to define the following work or lines of research that should be carried out for the implementation of systems that use artificial intelligence to classify network traffic and detect cyber-attacks in real time as soon as they begin to occur and to take the appropriate measures accordingly.

Keywords

AI, machine learning, cybersecurity, IoT, anomaly and intrusion detection, network security.

Índice general

Índice de figuras

Índice de tablas

1. Introducción

Este trabajo de fin de grado (TFG) consiste en la investigación y estudio de la viabilidad de implantar sistemas que, mediante técnicas de *machine learning*, sean capaces de clasificar el tráfico de redes de dispositivos IoT detectando ciberataques en las fases más tempranas y minimizar los daños ocasionados por estos. Para ello se implementarán algunos modelos de clasificación, los cuales se entrenarán y validarán para obtener sus métricas de rendimiento y, en función de los resultados obtenidos, se determinarán los siguientes pasos necesarios para la implantación de este tipo de sistemas de detección de ataques.

1.1. Motivación

El internet de las cosas o IoT ha experimentado un importante crecimiento en los últimos años y se espera que esta tendencia continúe en aumento en el futuro. Además, cada día es más frecuente utilizar este tipo de dispositivos, tanto de forma activa como pasiva, por casi cualquier persona que viva en un país medianamente desarrollado. Estos dispositivos nos hacen la vida más fácil de formas que hace algunos años solo podíamos esperar ver en el cine o en la televisión. El rápido crecimiento de este tipo de tecnologías no solo nos ha hecho las cosas más sencillas o cómodas, también introducen nuevos problemas y desafíos que se deben ser abordados de forma urgente, como lo es el problema de mantener la seguridad y la privacidad de los usuarios que hacen uso de estos dispositivos.

Las redes IoT, debido a su rápido crecimiento, son objetivos cada vez más llamativos para los ciberdelincuentes. Además, estos dispositivos son vulnerables a un amplio abanico de ataques debido al uso de sistemas operativos propietarios y disponer de recursos bastante limitados o ser

dispositivos móviles que de forma recurrente se conectan y desconectan de diferentes redes, entre otras causas.

En este contexto, la detección en fases tempranas del ciberataque se debe presentar como un problema de relativa urgencia, y teniendo en cuenta los avances realizados en el área de la inteligencia artificial en los últimos años, debería considerarse la puesta en marcha de sistemas de detección de ataques mediante técnicas de inteligencia artificial de forma masiva para mantener la seguridad y la privacidad de los millones de usuarios que hacen uso de este tipo de dispositivos constantemente. De este modo se podrían identificar y mitigar los daños ocasionados por estos ataques desde etapas tempranas, ayudando a prevenir daños mayores y protegiendo tanto la información como la integridad y disponibilidad de los dispositivos IoT. La resolución de este problema plantea ciertos desafíos, como la necesidad de procesar grandes volúmenes de datos en tiempo real y la adaptación a entornos dinámicos y altamente cambiantes.

Debido a estas consideraciones me he sentido motivado para realizar este trabajo, ya que la aplicación de técnicas de *machine learning* para la detección de ciberataques en redes de dispositivos IoT parece tener un potencial prometedor para mejorar la privacidad, integridad y disponibilidad de estos dispositivos y la información que procesan. Aprovechándonos de los avances de las técnicas del aprendizaje automático se pueden desarrollar sistemas capaces de identificar patrones de tráfico malicioso permitiéndonos anticiparnos a posibles amenazas.

La investigación en este campo contribuye tanto al avance científico-tecnológico como a la sociedad debido al impacto positivo que tiene sobre esta el mantener la seguridad y la privacidad de todos los usuarios, tanto activos como pasivos, de estos dispositivos.

1.2. Principales problemas y desafíos

La implantación de este tipo de sistemas de seguridad en redes IoT presenta algunos problemas y desafíos que habrá que resolver o minimizar en la medida de lo posible. De entre ellos, destacamos los siguientes:

Diversidad de dispositivos y protocolos La gran diversidad de dispositivos IoT junto a los múltiples protocolos que utilizan (algunos de ellos propietarios) presenta un importante desafío ya que podemos encontrar desde sensores de aparatos de medición hasta cámaras de

seguridad, pasando por electrodomésticos, coches y hasta bombillas inteligentes. Esta amplia variedad podría dificultar la implementación de medidas de seguridad estandarizadas y la gestión centralizada. A la hora de implementar estos sistemas de seguridad se debe tener en cuenta la alta variedad de dispositivos que podremos encontrarnos, funcionando en ocasiones con diferentes protocolos de comunicación.

Limitaciones de recursos La mayor parte de los dispositivos IoT tienen una disponibilidad limitada de recursos en términos de procesamiento, memoria, almacenamiento y energía, lo cual puede dificultar la implementación de sistemas de seguridad robustos en los propios dispositivos. A la hora de implementar sistemas para proteger este tipo de dispositivos habría que tener claro qué tipos de dispositivos se tendrán que proteger y si podrán ejecutar estos sistemas los propios dispositivos o si se deberá ejecutar de forma distribuida evitando pesadas cargas de procesamiento a los propios dispositivos.

Privacidad y gestión de datos Buena parte de los dispositivos IoT recopilan datos de forma masiva. Estos datos, en ocasiones, podrían ser datos personales o sensibles, lo que presenta el problema de mantener la privacidad de las personas que hagan uso de este tipo de dispositivos.

1.3. Impacto del aprendizaje automático en la seguridad informática

La ciberseguridad se ha convertido en un campo de investigación y aplicación indispensable en la sociedad digital actual. El aprendizaje automático se presenta como una herramienta útil y versátil para abordar diferentes desafíos de seguridad en diversas áreas, incluidas las redes de dispositivos IoT. Se destacan los siguientes puntos que remarcan la importancia de las técnicas de aprendizaje automático en la ciberseguridad:

Identificación de patrones maliciosos Mediante técnicas de aprendizaje automático se tiene la capacidad de analizar grandes cantidades de datos detectando patrones que podrían indicar que se está llevando a cabo una actividad maliciosa por parte de un ciberdelincuente. Mediante estas técnicas se podrían identificar comportamientos o patrones anómalos que nos indiquen que existe la posibilidad de que se esté produciendo un ataque y actuar en consecuencia.

Adaptabilidad Las redes IoT son altamente dinámicas y están en constante crecimiento y evolución. Cada nuevo dispositivo que conectemos a la red va a introducir cambios en los patrones de tráfico. Algunas técnicas de *machine learning* permiten que el sistema se adapte a los cambios sin necesidad de realizar grandes cambios.

Automatización Mediante estas técnicas se puede automatizar buena parte del proceso de detección de amenazas y respuesta ante el incidente, lo que permite ofrecer respuestas rápidas ante un ciberataque minimizando los daños ocasionados por este.

Mejora continua Algunos algoritmos de aprendizaje automático son capaces de mejorar mediante retroalimentación con los datos de entrada y los resultados obtenidos, volviéndose más efectivos con el paso del tiempo y haciéndolos más adaptables a los cambios en la red de dispositivos y a las técnicas utilizadas por los ciberdelincuentes.

2. Objetivos del proyecto

2.1. Objetivos principales

El propósito principal de este trabajo es investigar y estudiar la viabilidad de la implementación de sistemas que, mediante técnicas de *machine learning*, sean capaces de detectar ciberataques, a ser posible en las fases más tempranas, en redes IoT. Se establecen los siguientes objetivos para tratar de cumplir el propósito del trabajo:

Análisis del estado del arte Se lleva a cabo un exhaustivo estudio de los trabajos más recientes en materia de seguridad en redes IoT empleando técnicas de *machine learning*, clasificación de tráfico de red y análisis de conjuntos de datos de tráfico en redes de dispositivos IoT.

Identificar las técnicas aplicables Se comparan los resultados de diferentes trabajos identificando las técnicas y algoritmos que ofrecen mejores resultados para la clasificación del tráfico de red para determinar los modelos que se implementarán para estudiar su viabilidad.

Búsqueda y análisis de conjuntos de datos Se realiza una búsqueda y análisis de los diferentes conjuntos de datos disponibles con tráfico de redes IoT y se toma la decisión del conjunto de datos que se utilizará para entrenar y validar los modelos que se implementen.

Limpieza y procesado de los datos Se realizan diversas operaciones sobre el conjunto de datos la eliminación del ruido y limpieza de datos inválidos y filtrar y codificar correctamente los diferentes atributos del conjuntos de datos.

Implementación de clasificadores Se implementan algunos modelos para clasificación del tráfico de red teniendo en cuenta una gran variedad de ataques que se pueden producir sobre este tipo de redes y dispositivos.

Evaluar y ajustar los clasificadores Se realiza la evaluación y ajuste de los modelos con el conjunto de datos resultante del preprocesado realizado y se comparan los resultados entre los modelos implementados y con los resultados obtenidos por los autores de trabajos similares.

Evaluación de los resultados Se realiza una evaluación de los resultados obtenidos para extraer las conclusiones sobre la viabilidad de clasificar el tráfico de red, de forma que se puedan detectar posibles ataques a los dispositivos conectados a la red.

Determinar los trabajos futuros En base a los resultados y las conclusiones extraídas de los mismos, se determinan los siguientes pasos lógicos y trabajos futuros necesarios para mejorar e implementar este tipo de técnicas en redes reales.

Al alcanzar estos objetivos se espera contribuir al avance en materia de seguridad de las cada vez más numerosas redes IoT, proporcionando las bases para la implantación de técnicas de inteligencia artificial en sistemas reales protegiendo eficazmente este tipo de dispositivos.

2.2. Objetivos personales

Se realizó este trabajo orientado a la investigación persiguiendo varios objetivos personales, además de por el interés que suscito la propuesta recibida. Entre los diversos objetivos personales perseguidos en la realización de este tipo de trabajo destaco los siguientes:

Ampliar mi visión sobre las técnicas de inteligencia artificial Se buscó obtener una amplia visión del alcance y posibilidades que brindan las técnicas de inteligencia artificial, teniendo la posibilidad de documentarme ampliamente a lo largo del proyecto de las diferentes técnicas utilizadas hoy en día, así como afianzar conceptos vistos en el grado y extender estos en la medida de lo posible.

Ampliar mis conocimientos en seguridad informática Una de las razones por la que me inicié en el grado fue el de obtener conocimientos

avanzados en seguridad informática. Dado que en el grado no se lleva a cabo un estudio en profundidad de la materia, me pareció una buena oportunidad de ampliar mis conocimientos en este área.

Desarrollar habilidades de resolución de problemas Este trabajo, debido a los problemas y desafíos que conlleva, me permitirá desarrollar mis habilidades para resolución de problemas, lo que me será de gran utilidad en el futuro.

Adquirir experiencia práctica Este trabajo me ofrece la posibilidad de obtener experiencia práctica en el campo de la ciberseguridad y la inteligencia artificial, lo cual podrá ser muy beneficioso a nivel personal, ya que son los campos por los que más me he interesado, tanto a lo largo de la carrera como en etapas anteriores. Este interés fue el que me animó a comenzar mis estudios en el grado.

3. Conceptos teóricos

En esta sección se presentan los conceptos y fundamentos teóricos necesarios para la correcta comprensión del trabajo realizado. Se describirán de forma teórica el funcionamiento de los modelos de *machine learning* utilizados, las técnicas de procesamiento de datos y codificación de atributos, así como los conceptos sobre seguridad en redes IoT que se consideren necesarios y sobre los ataques considerados para su detección en este trabajo.

3.1. Aspectos relevantes sobre la seguridad en redes IoT

Las redes IoT son redes muy extensas, normalmente con dispositivos con una disponibilidad limitada de recursos, por lo que necesitan apoyarse de lo que se conoce como computación en la nube. Es decir, los dispositivos IoT suelen limitarse a mostrar o recoger información, que procesan de forma ligera que envían o reciben desde un servidor para realizar procesos computacionalmente más costosos, haciendo que la mayor parte de esta carga recaiga sobre el propio servidor, liberando los recursos de los dispositivos para realizar las operaciones para las que se destinan estos dispositivos con la suficiente solvencia a pesar de las limitaciones de estos dispositivos. Además, este tipo de redes deben ser altamente escalables, ya que suele ser frecuente el aumentar el número de dispositivos que se conectan en una determinada red. Por otro lado, multitud de dispositivos IoT se dedican a la recogida de información en el medio en el que se encuentran para ser enviada a un servidor u otro equipo para su procesamiento y/o almacenamiento. Estos datos que recogen los dispositivos, pueden ser de diferente clase. Podemos encontrar dispositivos que manejan desde datos de temperatura

ambiental hasta datos sensibles o personales, que posteriormente son enviados a través de la red, o incluso podrían ser datos telemétricos de maquinaría automatizada, que de no funcionar adecuadamente podría suponer grandes riesgos, tanto para los propios sistemas y los datos que manejan como para las personas. Por ejemplo, imaginemos que sucedería si un atacante falsea las lecturas de algunos sensores que activan ciertos sistemas de seguridad en una fábrica o acceden a cámaras de seguridad que vigilan propiedades privadas. Debido a las limitaciones de los dispositivos, se implementan técnicas criptográficas ligeras y protocolos sencillos que no introduzcan una carga computacional significativa para los dispositivos, por lo que surge la necesidad de implementar sistemas de protección externos a los propios dispositivos IoT. Todos estos aspectos se tratan de forma mucho más extendida y en detalle en [14].

Debido a estas características, las redes IoT son objetivos atractivos para los ciberdelincuentes y es necesario protegerlos, pero también es necesario que puedan operar adecuadamente, manteniendo la escalabilidad en las redes, el buen funcionamiento de los dispositivos, la privacidad de los datos y la confianza de los usuarios, entre otros aspectos.

3.2. Visión general sobre el aprendizaje supervisado

El aprendizaje supervisado es una técnica de *machine learning* que emplea algoritmos que son entrenados utilizando conjuntos de datos etiquetados. Estos datos y etiquetas se corresponden con las entradas y las salidas esperadas por parte del algoritmo. Los datos de entrada representan características o atributos del sistema que estamos analizando y la salida o etiqueta se corresponde con el resultado o la clase asociada a la instancia concreta del sistema que se clasifica o predice.

Durante el proceso de entrenamiento el algoritmo trata de identificar patrones en los datos que le permitan predecir el resultado esperado, de modo que después sea capaz de generalizar y realizar las predicciones correctamente sobre registros de datos sin etiquetar y que no han sido usados para entrenar el algoritmo.

Se tienen dos tipos principales de problemas en el aprendizaje supervisado, que son la clasificación y la regresión. En el caso que nos ocupa nos centraremos en la clasificación, donde el objetivo es predecir las clases o

categorías a las que pertenece la instancia de datos concreta que se está clasificando.

Se puede dar una definición más formal, como la que dan Emilio Soria et al. [13]:

"El aprendizaje supervisado es una familia de algoritmos que utiliza un conjunto de patrones etiquetados, llamado conjunto de entrenamiento y denotado como $\{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$, para aprender a hacer predicciones sobre nuevos datos.

Los elementos $\mathbf{x}_i$ son vectores de características. La expresión y_i se corresponde con la etiqueta asignada a cada patrón.

El objetivo es predecir la etiqueta y asociada a un nuevo patrón del que solo conocemos su vector de características σ. Para ello se genera un modelo cuya entrada sea un vector de características y su salida alguno de los posibles valores de las etiquetas. En el proceso de entrenamiento, se emplea el conjunto $\{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$ para que el modelo "aprenda" la relación entre los valores de entrada/salida y, con ello, sea capaz de estimar y conociendo σ".

En las siguientes subsecciones se presentará una breve descripción de las bases teóricas de los modelos de clasificación utilizados en el trabajo, así como los conceptos relacionados que puedan ser necesarios describir por mencionarse o utilizarse en el trabajo.

Ensemble

El aprendizaje por conjunto o sistema de clasificadores múltiples y aprendizaje basado en comités, también conocido como *ensemble*, se basa en entrenar y combinar múltiples clasificadores aumentando la capacidad de generalización del modelo, tal y como describe Zhi-Hua Zhou en [18]. Existen diferentes tipos de *ensembles*, como *ensembles* de árboles de decisión (como *random forest*) o de redes neuronales. A los clasificadores que componen el modelo, se les suele denominar clasificadores débiles y al *ensemble* o conjunto de clasificadores combinados se le denomina clasificador fuerte. La idea general es la combinación de múltiples clasificadores débiles para construir un clasificador fuerte. Dentro de este grupo se encuentran métodos como el *boosting* o el *bagging*.

Árboles de decisión

Un árbol de decisión es un modelo de aprendizaje supervisado que, una vez entrenado, proporciona un grafo con estructura de árbol para realizar

las predicciones, tal y como detallan Emilio Soria et al. en [13]. Cada nodo de ramificación del árbol se examina con una única característica del vector de características. Si el valor de la característica está por debajo de cierto umbral, se sigue por el hijo izquierdo en el árbol y en caso contrario por el derecho. Se sigue este proceso hasta llegar a un nodo hoja, donde se determina la clase a predecir por el árbol.

Durante el entrenamiento, en cada nodo de ramificación se realiza una división del conjunto de datos de entrenamiento que se tiene en ese nodo (ya que habrá sido dividido en nodos previos, salvo en la raíz) en dos partes, los que verifican la condición y los que no la verifican, tratando de situar estos puntos de división de la mejor forma posible, tratando de que en los nodos hoja solo queden patrones pertenecientes a una misma clase o, al menos, que haya una clase mayoritaria bien diferenciada. Por último, se asigna a cada región o nodo hoja, la clase mayoritaria de las etiquetas asociadas a las instancias del subconjunto de datos resultante en esa región al finalizar el entrenamiento. De este modo, el árbol es capaz de predecir la clase de un vector de características del que, en principio, desconocemos su etiqueta.

Clasificador *Random Forest*

El clasificador *Random Forest* es un modelo de inteligencia artificial que utiliza técnicas de aprendizaje supervisado, proporcionando buenos resultados, incluso sin ajustes de los parámetros, además de ser altamente adaptable, tal y como describen Abdallah et al.[10]. Estas técnicas son aplicables para problemas de clasificación combinando múltiples árboles de decisión (*ensembles*), produciendo un modelo capaz de predecir la clase de datos con mayor precisión y coherencia que los árboles de decisión por sí solos. Además, los árboles de decisión tienen problemas de sobreajuste que se solucionan con esta técnica de ensamblado, ya que se entrena cada árbol individual con un subconjunto aleatorio de características, por lo que cada árbol será diferente y dada una instancia de datos, estos árboles podrán predecir clases diferentes. Finalmente se clasifica la instancia de datos por votación entre todos los árboles del bosque, asignando la clase que más árboles hayan predicho para esa instancia en particular.

XGBoost o *Extreme Gradient Booster*

Usamos el término *boosters* para referirnos a una familia de algoritmos que en base a un modelo de aprendizaje "débil" crea un modelo "fuerte", mejorando el rendimiento de los modelos de aprendizaje automático. A

diferencia de otras técnicas (como los *ensembles*), los *boosters* no entrenan modelos del mismo tipo de forma aleatoria como ocurría con el modelo *random forest*, si no que se realiza un entrenamiento secuencial teniendo en cuenta los errores del modelo que se entreno anteriormente. En cada iteración o ronda de *boosting* se va aumentando la ponderación de las instancias que fueron predichas incorrectamente por el modelo recién entrenado, haciendo que el siguiente modelo se centre en identificar los patrones de las instancias difíciles de predecir. Finalmente, todos estos modelos "débiles" se combinan para formar el modelo "fuerte". Otra diferencia significativa con los *ensembles* es que no usa un sistema de votación "democrático" donde cada voto tiene el mismo peso. Estos modelos ponderan la decisión de cada modelo en función de la precisión de las predicciones que ha realizado, teniendo mayor peso los votos o decisiones de los modelos con mejores precisiones.

3.3. Métricas para la evaluación de los modelos

La evaluación de los modelos es una parte fundamental en el proceso de desarrollo de modelos de aprendizaje automático como se describe en [11]. Existen varios métodos de evaluación de modelos que nos ayudan a determinar qué modelo de aprendizaje automático funciona mejor para un conjunto de datos determinado. Entre las técnicas técnicas disponibles, podemos destacar la técnica conocida como *N-Fold Cross-Validation*, usada para el ajuste y evaluación de los modelos que se han probado en este trabajo en algunos de los experimentos realizados.

Por otro lado se tienen diferentes métricas utilizadas para evaluar el rendimiento del modelo, como *accuracy*, *precision*, *recall*, *F1score*, entre otras. Estas métricas, que se describen a continuación, son diferentes medidas del desempeño del modelo al ajustarse a los datos en el proceso de entrenamiento.

Accuracy

Se trata de una métrica de evaluación, cuya traducción podría darse como exactitud, que mide la tasa de aciertos del modelo. Esta métrica se calcula como el número de instancias bien clasificadas por el modelo dividido entre el número total de instancias de datos. Es decir, se obtiene mediante la formula (en clasificación binaria):

$$\frac{TP + TN}{TP + TN + FP + FN}$$

En la fórmula anterior se tiene que TP se corresponde con el número de verdaderos positivos, TN con el número de verdaderos negativos, FP con los falsos positivos y FN con los falsos negativos. Podemos extender esta formula para la clasificación con más de dos clases del siguiente modo:

$$\frac{\sum TC_i}{\sum TC_i + \sum NC_i}$$

Siendo en la fórmula anterior TC_i el número de instancias de la clase C_i bien clasificadas y NC_i el número de instancias de la clase C_i clasificadas de forma incorrecta.

Esta medida nos da la fracción de instancias clasificadas correctamente en tanto por uno. Se puede multiplicar por 100 para obtener el porcentaje de acierto del modelo.

Precision

Esta métrica, como su nombre indica (precisión en español), nos da la precisión del modelo. La precisión se obtiene, para cada una de las clases, de la siguiente fórmula:

$$\frac{TP}{TP + FP}$$

En la fórmula anterior TP representa el número de instancias de la clase positiva que han sido clasificadas correctamente y FP el número de instancias de la clase positiva que se clasifican erróneamente. [17]. Esto nos indica la precisión con la que se clasifica la clase positiva. Puede extenderse a la clase negativa sin más que sustituir los valores de la clase positiva por los de la negativa.

Recall

Esta métrica nos indica la proporción de instancias de una clase que se han clasificado correctamente, permitiendo que podamos decidir si un modelo es mejor que otro cuando los falsos positivos y los falsos negativos no son igual de costosos. Por ejemplo, en caso de querer predecir o clasificar diagnósticos médicos, un falso positivo podría ser más costoso debido a que se invertiría en medicación para un paciente sano, sin embargo, en detección de fraudes un falso negativo implicaría un fraude no detectado con la consecuente perdida económica [17]. El recall se puede obtener mediante la siguiente fórmula:

$$\frac{TP}{TP + FN}$$

Como podemos ver el *recall* es básicamente el número de instancias de la clase positiva predichas correctamente dividido entre todas las instancias de esa misma clase, tanto las clasificadas correctamente como las incorrectas. Tanto el *recall* como la *precision* nos ayudan a realizar la evaluación del modelo según si los errores en la clasificación en una clase acarrean mayor coste (no necesariamente económico) que en otra.

F1-Score

Esta métrica, llamada a veces *F-score* o *F-measure*, se puede interpretar como la media armónica de la *precision* y el *recall* [5]. Para calcular el *F1-score*, se asigna el mismo peso tanto para la *precision* como para el *recall* (en F_β*-score* se da más peso a la *precision* según crece el valor de β) [16]. La fórmula para el cálculo de esta métrica es la siguiente:

$$2\frac{precision \cdot recall}{precision + recall} = \frac{2TP}{2TP + FP + FN}$$

Al igual que la *precision* y el *recall*, esta métrica devolverá valores entre 0 y 1. Cuando vale 1 indica que tanto la *precision* como el *recall* son perfectos, es decir, ambas valen 1, y cuando vale 0 indica que al menos una de ellas tiene valor 0.

Curvas PR

Las curvas PR o curvas *precision-recall* nos sirven para evaluar y comparar modelos de forma rápida e intuitiva. Se construye representando en una gráfica la *precision* frente al *recall*. En esta curva se representa como varía la *precision* del modelo cuando varía el *recall* y viceversa. Una curva PR será mejor que otra en cuanto más se acerque al punto (1,1) del espacio de coordenadas del gráfico. Por ello, como se comenta en [18], una curva PR será mejor que otra si la primera curva envuelve completamente a la segunda, es decir, será mejor cuanto mayor sea el área bajo la curva PR. Esta métrica también es de utilidad, junto a la curva ROC que se describe más adelante, para evaluar el ajuste de los umbrales de las predicciones, si tratamos con probabilidades, y/o el ajuste de los pesos de las clases, para tratar de ajustar esto en función de los costes que puedan tener los distintos errores que cometa el modelo, pues es probable que un falso positivo no sea igual de costoso que un falso negativo.

Curvas ROC

Se trata de otra métrica gráfica para evaluar y comparar los modelos. Se construye representando la tasa de verdaderos positivos frente a la tasa de falsos positivos. Nos permite ver como aumenta la tasa de falsos positivos según se aumenta la tasa de los verdaderos positivos. En este caso, el modelo será mejor cuanto más cerca se encuentre la curva del punto ideal (0,1), donde se tiene una tasa de verdaderos positivos de 1 y una tasa de falsos positivos de 0, lo que sería un modelo perfecto, como se describe en [18]. Por esto, nuestro modelo también será mejor cuanto mayor área se tenga bajo la curva, lo que se conoce como AUC (area under curve). Esta gráfica es de gran utilidad, junto a la curva PR, para realizar comparaciones de varios modelos que tratan de resolver el mismo problema.

3.4. *N-fold Cross-Validation*

Como ya hemos mencionado, se trata de una técnica para la evaluación y ajuste de los modelos. Esta técnica es bastante sencilla y resuelve, en parte, el problema de la aleatoriedad de los resultados en el entrenamiento y validación de algunos algoritmos de aprendizaje automático al utilizar subconjuntos de datos diferentes en entrenamiento y validación. Básicamente la idea consiste en dividir el conjunto de entrenamiento en tantas particiones como *folds* o iteraciones se vayan a realizar. En cada iteración se usan $n-1$ particiones para entrenar el modelo y la restante para validación. Una vez finalizado el proceso se obtienen los valores medios de las diferentes métricas de rendimiento. Esta técnica es útil para validar y ajustar los hiper-parámetros de los modelos, obteniendo una visión más realista de como de bien (o mal) funcionará el modelo.

3.5. LoRaWAN y NB-IoT

LoRaWAN(*Long Range Wide Area Network*) es una especificación para redes de baja potencia y área amplia. Utiliza una técnica patentada, capaz de modular la red basándose en técnicas de modulación del espectro, permitiendo las comunicaciones a largas distancias a costa de utilizar un ancho de banda estrecho. Además, las ondas de banda estrecha lo hace resistente a las interferencias. Se puede consultar toda la documentación sobre este estándar de comunicaciones en la especificación técnica oficial [2]. Esta tecnología cubre grandes áreas de forma bidireccional para la transmisión de bajos volúmenes de datos.

NB-IoT (Narrowband Internet of Things) es otro estándar de comunicaciones definido por el 3GPP [1]. se trata de una tecnología de comunicaciones inalámbrica, enfocada en la conectividad de baja velocidad y bajo consumo para los dispositivos IoT. Se basa en la tecnología LTE ya existente, ofreciendo conexiones con bajas interferencias. Se diseño para facilitar la conexión de dispositivos que realizan transferencias con bajos volúmenes de datos durante tiempos prolongados, incluso en zonas con difícil acceso a la red. Además tiene la capacidad de conectarse a las redes móviles ya establecidas.

En resumen, estos estándares se basan en realizar transferencias de bajos volúmenes de datos de forma eficiente a multitud de dispositivos, minimizando el consumo de energía de los mismos a la par que se incrementa el área o rango de cobertura para llegar incluso a zonas de difícil acceso a la red por estar demasiado aisladas.

3.6. One-hot encoding

Se trata de una técnica de codificación de atributos o características categóricas, convirtiéndolas en numéricas. Esta técnica crea un nuevo atributo por cada categoría contenida en el atributo a codificar, indicando con el valor 1 o *true* si esa instancia contiene esa categoría para el atributo codificado. El resto de atributos de esa misma característica, generados en la codificación contendrán el valor 0 o *false*. Tras la codificación, es común eliminar una de las características generadas, ya que pueden ser inferidas a través del resto. Con esto se trata de resolver el problema de multicolinealidad, que afecta a ciertos problemas y modelos, como puede ser la regresión lineal. Sin embargo, al utilizar métodos basados en árboles de decisión, los modelos no se ven afectados por este problema, ya que pueden manejar características altamente correlacionadas sin problemas, aunque sigue siendo recomendable codificar estas características para, por ejemplo, poder usar el mismo conjunto de datos en diferentes tipos de modelos.

3.7. Optimización de hiper-parámetros

La optimización de los hiper-parámetros es un paso crucial para el buen rendimiento del modelo, permitiendo que tras el entrenamiento tengan una buena capacidad de generalización al tratar datos diferentes a los usados para el entrenamiento. Además estos hiper-parámetros definen como será nuestro modelo. En el caso de modelos *Random Forest*, podemos ajustar el número de árboles en el bosque o la profundidad máxima que tendrán estos

árboles, entre otros hiper-parámetros. Existen diferentes formas de realizar el ajuste de los hiper-parámetros. Una es mediante experimentación. Se realizan sucesivos entrenamientos variando el valor de los hiper-parámetros, estudiando como afecta la variación de cada uno de estos al rendimiento del modelo, hasta obtener el mayor rendimiento posible. Esto, en ciertos casos puede ser una tarea larga y tediosa, por lo que se puede recurrir a técnicas de optimización de los hiper-parámetros. En este trabajo se han tratado de aplicar dos técnicas para optimizar los hiper-parámetros, las cuales se describen a continuación.

Grid Search

La técnica de *Grid Search* o búsqueda en mallas, realiza una búsqueda exhaustiva entre todas las combinaciones de hiper-parámetros que se definen en las mallas de hiper-parámetros. Para cada uno de los hiper-parámetros que se tratan de optimizar, se define una serie de valores que podrá tomar este hiper-parámetro y se entrena al modelo con cada combinación posible de entre los valores para cada hiper-parámetro en la malla. Finalmente nos quedamos con la combinación que otorga un mayor rendimiento al modelo. En caso de tener mallas de hiper-parámetros grandes, puede no ser muy eficiente ya que debe realizar un gran número de entrenamientos sobre el modelo y estos, dependiendo del tamaño de los datos, el algoritmo escogido y el hardware disponible, podrían ser muy largos. Esta técnica suele combinarse con validación cruzada, para tener una visión más realista del rendimiento que se alcanza con cada combinación de atributos, pero por contra, añade más rondas de entrenamiento incrementando más aún el tiempo de búsqueda.

Randomized Search

Se trata de una técnica muy similar a la anterior, pero con la diferencia de que la búsqueda sobre las mallas no se realiza de forma exhaustiva, sino de forma aleatoria. Se define una malla de hiper-parámetros como la descrita en la subsección anterior y busca de forma aleatoria, durante el número de iteraciones que se le indique, la mejor combinación de hiper-parámetros entre las combinaciones exploradas. Esta técnica no es tan óptima como la búsqueda en malla en el sentido de que podría no explorar la mejor combinación de hiper-parámetros en las iteraciones realizadas, pero tiene como ventaja, que al no explorar todas las combinaciones el tiempo de búsqueda será mucho menor si se ajusta el número de iteraciones adecuadamente. Esta técnica, aunque no encuentre la combinación óptima, es altamente

probable que encuentre otra combinación parecida que otorgue un buen rendimiento al modelo sin necesidad de explorar todas las combinaciones, siempre y cuando la malla este definida en rangos de valores adecuados para cada hiper-parámetro y se configure un número de iteraciones suficiente, sin llegar a ser excesivo.

Ataques considerados

El conjunto de datos utilizado incluye datos de nueve escenarios de ataques e infección de *malware* distintos, los cuales se describen brevemente en esta sección basándose en las descripciones dadas en el trabajo de Nour Moustafa [12], donde el autor, el mismo que diseño y elaboró los conjuntos de datos utilizados en este trabajo, realiza un análisis exhaustivo de los diferentes conjuntos de datos y como se obtuvieron estos. Aunque en el presente trabajo no ha sido necesario profundizar en estos temas, considero de interés añadir unos breves comentarios sobre cada tipo de ataque considerado para tener una visión más clara del alcance que podrían tener este tipo de sistemas de detección de intrusiones.

Scanning

Este tipo de ataques, a veces llamados ataques de sondeo o de monitorización, constituyen la primera etapa de otros ataques o pruebas de penetración. Tiene como finalidad obtener información del sistema objetivo, como direcciones IP, puertos abiertos, etc. tratando de buscar canales de comunicación susceptibles de ser explotados por el atacante.

DoS (Denial of Service)

Se basa en inundar un sistema con múltiples peticiones, tratando de bloquear los servicios proporcionados por el sistema que se encuentra bajo el ataque al superar el número de peticiones simultaneas que este puede manejar, haciendo que el sistema se ralentice o incluso deje de funcionar por completo.

DDoS (Distributed Denial of Service)

Sigue la misma idea que los ataques DoS, pero con una diferencia importante, este ataque se lanza de forma distribuida desde un gran número de dispositivos, previamente infectados con *malware*. A estos dispositivos infectados se les suele denominar *bots* o *zombies*, y a la red de dispositivos infectados suele ser llamada *botnet*. El atacante tiene el control sobre todos

los dispositivos infectados, por lo que cuando está listo para lanzar el ataque, hace que todos los dispositivos, de forma simultanea, lancen ataques de denegación de servicio contra el sistema objetivo, siendo los efectos mucho más notables que en ataques DoS.

Ransomware

Se trata de una infección con un complejo *malware* que cifra todos los archivos de un sistema, impidiendo el acceso a los mismos a sus usuarios legítimos hasta que paguen un rescate para obtener la clave que les permita descifrar sus archivos, dejando el sistema totalmente inoperativo. Por esto también se conoce como ”secuestro de datos”.

Backdoor

Se trata de técnicas mediante las cuales los atacantes tratan de eludir las medidas de seguridad implementadas en un sistema, obteniendo acceso y escalada de privilegios, pudiendo infectar el sistema, por ejemplo, con *ransomware*, robar o modificar información, o incluso eliminar todos los ficheros del sistema dejándolo totalmente inoperativo. Una vez el atacante consigue el acceso y la escalada de privilegios, podría realizar cualquier acción maliciosa desde dentro del propio sistema.

Injection

Son técnicas de ataque que inyectan datos de entrada falsos desde los clientes hacia las aplicaciones para explotar sus vulnerabilidades. Existen diferentes técnicas de inyección como la inyección SQL, donde se trata de introducir parte de una sentencia SQL en lo que será un campo en dicha consulta, tratando de modificar el resultado de la misma y obtener, modificar o eliminar información contenida en las bases de datos. Existen otros tipos de inyección, como lo que se conoce como *client-side injection*, donde se trata de inyectar código malicioso desde el cliente a la aplicación, haciendo que se procese y ejecute el código inyectado en los servidores de la aplicación.

Cross-site Scripting

Se basa en inyectar *scripts* maliciosos en sitios o aplicaciones web, que originalmente eran de confianza, tratando de infectar con estos *scripts* maliciosos a los usuarios finales del sitio o aplicación web usado por el atacante.

Password cracking

Este tipo de ataques incluye cualquier tipo de técnica de pirateo de contraseñas, tales como ataques de fuerza bruta o ataques de diccionario, donde el objetivo es obtener o robar la contraseña de acceso al sistema objetivo.

MIMT (an-In-The-Middle)

Este ataque se produce cuando el atacante se interpone entre los usuarios y los servidores de aplicaciones, donde puede mantenerse a la escucha o hacerse pasar por una de las dos partes, donde se crea la apariencia de que se trata de un intercambio ordinario de información.

4. Técnicas y herramientas

En este capitulo se presentan las técnicas y herramientas utilizadas o consideradas para el desarrollo de este trabajo, resumiendo los aspectos más destacados de cada una y presentando una breve justificación de las opciones elegidas.

4.1. Python3

Como lenguaje de programación, para llevar a cabo la creación y evaluación de los modelos así como para el procesado de los datos, escogido fue, indiscutiblemente, Python. Python es el lenguaje de programación de uso más extendido en ciencia de datos e inteligencia artificial, donde podemos encontrar multitud de bibliotecas especificas para aprendizaje automático y ciencia de datos. Se usó la versión 3.9.10 de Python3, disponible en su página web oficial.

4.2. Entorno de programación

Jupyter Notebook

Inicialmente, se barajó la posibilidad de utilizar los cuadernos Jupyter o Jupyter Notebooks, los cuales nos permiten la edición y ejecución de código Python. Estos cuadernos nos permiten ejecutar código de forma independiente en diferentes celdas de código, lo cual puede ser lo indicado para ciertas tareas, pero en este caso, tras comenzar usando estos cuadernos, me resultaban poco cómodos, por lo que se pasó a trabajar con módulos de

Python o ficheros .py. Se puede consultar más información en la página web del proyecto.

Anaconda

Se pretendía inicialmente usar la herramienta Anaconda para la ejecución de los ficheros Jupyter Notebook, ya que es la herramienta que se había usado en asignaturas del grado para la programación en Python. Esta opción se termino desechando a favor de VS Code, ya que muchas de las características que incluye Anaconda no iban a ser usadas y me sentía más cómodo trabajando con VS Code al ser más ligero y ofrecer una interfaz mucho más simplificada. Para más información sobre esta herramienta consultar la página web oficial.

VS Code

Se trata de un editor de código ligero que soporta la edición y depuración de código Python, entre otros lenguajes. Se escogió esta alternativa, como ya se ha comentado, al sentirme más cómodo con su interfaz principalmente y contar con algo de experiencia con el al ser el IDE que usábamos en la empresa donde realicé mis prácticas en desarrollo de ciberinteligencia. Se puede consultar más información o descargar la herramienta en la página web oficial.

4.3. APIs y bibliotecas

Pandas

Se escogió utilizar la biblioteca Pandas para el manejo y preparación de los datos principalmente por dos motivos, es una biblioteca de uso muy extendido y ya había trabajado con ella anteriormente y conocía el alcance de algunas de las funciones y estructuras de datos que implementa, aunque algunas otras han sido totalmente nuevas para mí. No se llegó a estudiar ninguna alternativa en profundidad ya que los resultados que se obtuvieron fueron buenos, al ser capaz de procesar conjuntos de datos con más de 22 millones de instancias y más de 40 atributos en tiempos del orden de minutos. En caso de no haber conseguido realizar estas tareas en tiempos aceptables habría que haber estudiado la posibilidad de usar otras bibliotecas más optimizadas como Dask, Vaex o cuDF, las cuales introducen un alto grado de paralelismo al realizar las diferentes operaciones sobre el *DataFrame*

empleando la potencia de la GPU. Para consultar información adicional, así como para acceder a la documentación oficial de esta biblioteca, se puede acceder a su página web oficial.

Sklearn/Scikit-Learn

Se ha utilizado esta biblioteca para realizar algunas operaciones en el preprocesado de los conjuntos de datos, así como para implementar el modelo *Random Forest* y obtener las métricas de rendimiento de los modelos, tanto este que menciono como el implementado con la biblioteca XGBoost. Se escogió esta alternativa debido a que es una biblioteca de las más utilizadas para el aprendizaje automático en Python, la cual añade funciones bastante completas para el preprocesado de los datos. Además, como comentaba con Pandas, esta biblioteca se ha ido usando en algunas de las asignaturas del grado, por lo que ya estaba ligeramente familiarizado con ella. Con esta biblioteca, además de realizar el procesado de los datos permite la creación de una gran variedad de modelos, incluye herramientas para la optimización de hiper-parámetros, además de ser bastante más sencilla de utilizar que otras alternativas. Se puede consultar toda la documentación de la biblioteca en su página web oficial.

XGBoost

Se trata de otra biblioteca utilizada para el aprendizaje automático. Se escogió esta biblioteca, ya que buscando alguna opción diferente a Sklearn, encontré esta y me pareció una buena opción debido a su facilidad de uso y a que incluye soporte para GPU, haciendo que los tiempos de entrenamiento sean muy pequeños. Además, tras las primeras pruebas arrojó resultados de rendimiento bastante buenos. Para más información y acceder a la documentación, consultar su página web oficial.

Keras/Tensorflow

Se estudió el usar esta biblioteca a través de la API de Keras, pero termino siendo desechada esta opción frente a Sklearn debido a que esta otra era mucho más sencilla de usar y ya estaba familiarizado con ella, además de que algunas operaciones sobre los *DataFrames* erán mucho más sencillas y rápidas de realizar directamente con Sklearn, ya que TensorFlow maneja estructuras de tipo tensor y tipos de datos propios y se debían realizar demasiadas operaciones al pasar de un tipoo de estructura a otra. Esta biblioteca también es ampliamente utilizada para el aprendizaje automático,

la cual es famosa por el alto rendimiento y velocidad de ejecución, además de ser compatible con la plataforma CUDA. Para más información y acceder a la documentación oficial consultar las páginas web oficiales de TensorFlow y Keras.

4.4. Gestión del proyecto

GitHub

Se escogió la plataforma GitHub para la creación del repositorio para el proyecto. Esta decisión se tomo debido a ser la plataforma utilizada en las asignaturas del grado, por lo que ya me sentía familiarizado con ella además de ser una de las plataformas más populares. Para más información acceder a la web de la plataforma GitHub.

GitHub Desktop

Para la gestión del repositorio en GitHub se ha usado la herramienta GitHub Desktop, la cual permite realizar diversas operaciones sobre nuestro repositorio a través de su intuitiva interfaz gráfica. Además esta diseñada para su uso en GitHub, la plataforma de desarrollo colaborativo indicada para la realización de los trabajos de fin de grado. Se puede descargar y consultar información adicional en su página web oficial

Gestión ágil

Para la realización de este trabajo se ha utilizado una aproximación a la metodología de gestión ágil. Al no ser un proyecto de desarrollo, ni estar completamente determinado desde el inicio (salvo los objetivos perseguidos) siendo un proyecto más orientado a la investigación, no se ha usado ninguna metodología concreta como Kanban o Scrum al no encajar con el tipo de proyecto pero, si se han tratado de adaptar las bases de la gestión ágil, estableciendo *sprints* de dos semanas de duración y realizando revisiones periódicas sobre estos, adaptando la gestión del proyecto a como iba evolucionando este. Inicialmente se establecieron unos objetivos junto a una descripción inicial, la cual se ha ido modificando según como se iba avanzando y los resultados que se obtenían en cada paso.

4.5. LaTeX

Para la elaboración de la memoria y de los anexos se ha utilizado el sistema de composición de textos LaTeX, el cual nos permite elaborar documentos con una alta calidad tipográfica. Este sistema es ampliamente usado en la elaboración de artículos científico-técnicos y otros trabajos similares. Concretamente se uso este sistema a través de la plataforma de gestión de documentos LaTeX, Overleaf.

4.6. Análisis de la calidad del código

Para el análisis del código implementado en el trabajo se han considerado algunas alternativas, las cuales se describen a continuación.

SonarQube

SonarQube es una herramienta para el análisis de la calidad del código, la cual, junto al resto de herramientas, se consideró para analizar el código que se ha ido generando a lo largo del trabajo. Esta alternativa fue rechazada en favor de Codacy, la cual se describe más adelante. Uno de los motivos por el cual se rechazo el uso de esta herramienta fue el que estaba enfocada en el análisis de proyectos de código orientado a objetos, no encajando con el tipo de programación que se ha utilizado en este trabajo. Se puede consultar toda la información relacionada con la herramienta en su página web oficial.

Codacy

Se trata de otra herramienta para la revisión y análisis del código. Esta herramienta puede ayudarnos a aplicar buenas prácticas de programación, evitar problemas de seguridad conocidos, reducción de la duplicidad de código, etc. Además, puede integrarse con el repositorio GitHub del proyecto, haciendo que el proceso de análisis y revisión sea extremadamente sencillo. Se escogió esta herramienta por su facilidad de uso, por su interfaz gráfica sencilla e intuitiva y por estar enfocada al análisis de cualquier fichero de código en diversos lenguajes de programación. Para más información sobre la herramienta visitar su página web oficial.

4.7. Herramientas de IA generativa

Microsoft Copilot

Se utilizó la versión *Designer* de la herramienta Microsoft Copilot para la generación del logo utilizado para la portada del fichero readme del repositorio GitHub.

ChatGPT 3.5

Se utilizó la versión gratuita de ChatGPT para obtener algunas ideas para la elaboración del anexo F. De los resultados que se obtuvieron con la herramienta, se seleccionaron las ideas que se consideraron más acertadas y, a partir de estas, se fue completando el anexo con ideas propias basadas en las ideas propuestas por la herramienta.

5. Aspectos relevantes del desarrollo del proyecto

5.1. Definición inicial del proyecto

En un comienzo se pretendía seguir parte de la línea de investigación del proyecto financiado por el INCIBE que se va a llevar (o se está llevando) a cabo en la UBU, para la detección de ataques sobre redes de dispositivos IoT aplicando técnicas de *machine learning*. Como el proyecto estaba tardando en iniciarse, y ya habían pasado algunas semanas, se decidió comenzar un trabajo de investigación basado en esta idea inicial.

Se comenzó planteando la posibilidad de investigar sobre cómo proteger redes IoT que usen los protocolos LoRaWAN y NB-IoT usando técnicas de *machine learning*. Para ello, se requería buscar y analizar diferente documentación y trabajos relacionados con estas tecnologías, la búsqueda y análisis de un conjunto de datos con tráfico de red, tanto benigno como malicioso, y la selección de un modelo inicial que pudiese dar buenos resultados para la clasificación del tráfico de red y comenzar a realizar pruebas. Tras los pasos de búsqueda y análisis de documentación y trabajos relacionados, nos encontramos con el problema de no encontrar conjuntos de datos que se ajustaran al trabajo que se pretendía realizar ya que o los datos no estaban etiquetados o no contenían tráfico malicioso o no se publicaban por temas relacionados con la protección de datos. Por ello, se decidió modificar ligeramente el trabajo a desarrollar.

Se decidió seguir con la idea de proteger redes de dispositivos IoT aplicando técnicas de *machine learning*, pero eliminando la restricción de que se utilicen protocolos específicos. Se pretende con esta idea, realizar una

nueva búsqueda de documentación y trabajos relacionados para su lectura y análisis, búsqueda de un conjunto de datos de tráfico de redes IoT que se ajuste al trabajo y continuar con el mismo. Tras elaborar una definición inicial del trabajo, estableciendo los objetivos de este, los siguientes pasos se han ido definiendo en función de los avances y los resultados que se iban obteniendo en pasos previos.

5.2. Búsqueda y análisis de documentación

Los primeros pasos del trabajo consistían en la búsqueda de documentación técnica y trabajos relacionados con las tecnologías y técnicas que se pretenden trabajar. Inicialmente se realizó la búsqueda de diversas publicaciones relacionadas con la aplicación de técnicas de *machine learning* para proteger dispositivos IoT conectados a la red, análisis de vulnerabilidades de protocolos IoT y redes 5G, documentación sobre los protocolos LoRaWAN y NB-IoT, entre otra documentación. Tras la lectura de esta documentación y tener una visión algo más clara de como funcionan este tipo de dispositivos se procedió a la búsqueda del conjunto de datos, encontrando los problemas mencionados en la sección anterior.

Tras la modificación de la definición inicial del trabajo se realizó una búsqueda adicional de documentación, donde se destacan los trabajos realizados por Nour Moustafa [12], donde realiza un análisis de los conjuntos de datos de “The TON_IoT datasets” junto a una comparación con otros conjuntos de datos similares que se tienen publicados, o el trabajo de Abdallah et al. [9], donde obtienen muy buenos resultados con diferentes modelos y distintas formas de procesar los datos.

Después de varias semanas de búsqueda de documentación, lectura y de realizar alguna modificación sobre la definición inicial del trabajo, se estaba en posición de continuar con las siguientes fases del trabajo.

5.3. Búsqueda y selección del conjunto de datos

El siguiente paso en el trabajo era la búsqueda y selección de un conjunto de datos. Se aprovechó el encontrar diversos trabajos que utilizaban los conjuntos de datos de ”The TON_IoT datasets” así como el trabajo de Nour [12], donde realiza una comparativa con otros conjuntos similares. Entre estos conjuntos de datos, se seleccionó el conjunto de tráfico de red de ”The

TON_IoT" tanto por la disponibilidad de diversos trabajos que lo utilizan, avalado además por lo buenos resultados que obtienen, como por ser el más actual, contener tráfico clasificado con 9 tipos diferentes de ataques a dispositivos IoT, contener un número de instancias lo suficientemente grande y, además, obtenerse los datos de fuentes heterogéneas, con diversidad de dispositivos y protocolos. No se tardo mucho en decidirse en seleccionar este conjunto de datos por lo anteriormente mencionado, destacando entre todos los que se presentan en el trabajo de Nour.

5.4. Preparación inicial del conjunto de datos

El siguiente paso tras la selección del conjunto de datos era la preparación inicial de los datos. En un principio, debido a que no conocía algunos tipos de técnicas de procesado y otras no las tenía muy familiarizadas todavía, se trató de mantener el máximo número de características, eliminando solo las que tenían una varianza muy baja. En este caso, se tenían algunos atributos que se mantenían vacíos en la inmensa mayoría de los registros de datos. Se elaboró un *script* en Python para el procesado de datos y otro para realizar las particiones. En este punto se utilizaron todas las características exceptuando las anteriormente mencionadas y se trato de codificar las direcciones IP por octetos y considerando las instancias que usaban direcciones IPv4 solamente, eliminando las instancias con direcciones IPv6 para tratar de codificarlas en un futuro. Como en este punto todavía no tenía muchos conocimientos sobre el tema del preprocesado, me dejé guiar por la selección de atributos realizada en el trabajo de Abdallah et al. [9], donde eliminaban las direcciones IP, las marcas de tiempo o *timestamp* y los números de puerto, tanto de origen como de destino. Según comentan en este trabajo, estas características daban problemas de sobreentrenamiento u *overfiting*, por lo que se decidió eliminarlas y se dejo la idea de codificar las direcciones IP. Los atributos no numéricos, los campos protocolo, servicio y estado de la conexión, debido a que no tenían muchos valores diferentes se aplico la técnica de *one hot encoding* y cambiando los booleanos resultantes por unos y ceros. En este proceso, debido a la naturaleza de los datos, los campos con valores nulos o desconocidos se trataron como un valor más, ya que algunas capturas, según el tipo de tráfico, no tendrán información en esos campos, y sería interesante que el modelo pueda utilizar la información de que campos están vacíos para clasificar el tráfico resultante en la red. Los atributos numéricos se reescalaron mediante normalización, ya que algunos modelos pueden

ser susceptibles a la escala de valores de los atributos. Con esto se tenía el conjunto de datos preparado para las primeras pruebas. Tras realizar algunas pruebas con este conjunto de datos resultante y ver como algunos modelos que se probaron inicialmente predecían la clase mayoritaria en todas las predicciones, se creó un script para balancear el conjunto de datos, ya que se tenían muchas más instancias de tráfico malicioso que de tráfico benigno o normal. Como se tiene un conjunto de datos bastante grande, se decidió utilizar la técnica de *undersamplig*, eliminando aleatoriamente instancias de la clase mayoritaria hasta que estuviesen equilibradas ambas clases.

En posteriores fases, y teniendo nuevos conocimientos y conceptos afianzados en la asignatura minería de datos, se parte de la selección de atributos y el procesado inicial, pero esta vez realizando una selección óptima de las características usando la función de selección selectKBest con la función de puntuación Chi^2. Tras probar esta selección usando un *pipeline* junto a la selección óptima de atributos usando la función RandomizedSearchCV de Scikit-Learn, se determino que la selección óptima de atributos es la selección de las 47 características que se tienen tras el filtrado de características inicial y el codificado de las características categóricas, ya que en casi todos los experimentos con estas funciones de optimización se escogían las 47 características, salvo en algunas ocasiones donde se escogía un valor muy cercano, probablemente por no haber explorado una selección mayor.

Después de realizar algunas pruebas y el ajuste inicial de los modelos se detectó un problema. Se estaban normalizando los valores de las características antes de realizar las particiones de entrenamiento y test. Al detectar esto, se corrigió la forma de procesar los datos, añadiendo un *script* para normalizar las características numéricas tras dividir el conjunto de datos.

Finalmente se tienen tres conjuntos de datos para entrenar a los modelos.

Conjunto incorrectamente procesado Conjunto balanceado, codificado mediante *one-hot encoding*, normalizado antes de dividir los datos en proporciones 80-20 y estratificando por la columna *label*.

Conjunto normalizado Conjunto balanceado, codificado mediante *one-hot encoding*, normalizado tras generar las particiones en proporciones 80-20 y estratificando por la columna *label*.

Muestra para pruebas Pequeño conjunto de datos usado para realizar pruebas rápidas.

5.5. Implementación de un primer modelo

Como primer modelo se intentó utilizar una red neuronal sencilla implementada en TensorFlow, pero aun aumentando la complejidad de la red neuronal, y realizando diversos ajustes de los hiper-parámetros, no se consiguieron buenos resultados. El modelo siempre predecía la clase mayoritaria o positiva, incluso cuando se balanceó el conjunto de datos los resultados obtenidos no eran nada buenos. Por ello se descartó este tipo de modelos y se probó con un *ensemble*, concretamente con *Random Forest* (RF) de la biblioteca Scikit-Learn. Al probar este modelo, tanto con el conjunto de datos balanceado como el inicial dieron buenos resultados, tanto añadiendo pesos a las clases en función del número de instancias de la clase como sin realizarlo, siendo ligeramente mejores los resultados que utilizan el conjunto balanceado. Se continuó usando el conjunto de datos balanceado, ya que esto podrá evitarnos problemas al usar otros modelos. El modelo utilizado, al no ser muy susceptible del ajuste de hiper-parámetros y dar buenos resultados con los parámetros por defecto, se mantuvieron así en un principio hasta tener otro modelo con el que comparar. Después de implementar un segundo modelo y realizar algunas comparativas, se trató de ajustar el modelo mediante prueba y error, viendo como afectaba el ajuste de cada parámetro individual al modelo y tratando de escoger los mejores valores. Esto se realizo así en un principio por falta de conocimiento de algunas técnicas de optimización de hiper-parámetros, las cuales se pretende aplicar más adelante y comparar con los resultados obtenidos en este punto.

Finalmente, se consiguió mejorar ligeramente los resultados del modelo, pero de forma mínima, poniendo de manifiesto la baja dependencia que tienen los resultados de este modelo con el ajuste de los hiper-parámetros. Donde más se notó el ajusté fue en los tiempos de entrenamiento, notando un incremento notable al ir aumentando el número de estimadores o las profundidades máximas permitidas para cada árbol individual.

5.6. Selección e implementación de un segundo modelo

El siguiente paso fue seleccionar otro modelo para poder realizar alguna comparativa con el modelo anterior. Debido a los resultados obtenidos con *Random Forest* por parte de Abdallah et al. [9], se decidió usar un modelo basado en *boosters*, concretamente se decidió usar la biblioteca XGBoost, usando como *booster* árboles equilibrados o *general balanced trees* como lo

especifican en la biblioteca usada. Utilizando los mismos datos que para el conjunto anterior se obtuvo unos resultados casi tan buenos, incluso sin realizar el ajuste de los hiper-parámetros. Ajustando los hiper-parámetros, al igual que con el anterior clasificador, se mejoró ligeramente la tasa de acierto del modelo, pero en este caso tampoco se llegó a notar en los tiempos de entrenamiento, ya que a diferencia de Scikit-Learn, XGBoost si tiene soporte para GPU, reduciendo los tiempos de entrenamiento considerablemente. El ajuste de parámetros se realizó siguiendo la misma metodología que con el clasificador *Random Forest*, dejando para más adelante el ajuste optimizado del modelo mediante técnicas como *grid search*, la cual en este punto del trabajo aún desconocía.

5.7. Selección óptima de atributos e hiper-parámetros

Sobre el modelo *random forest*, se realizan algunas modificaciones en otro fichero, para mantener el anterior, añadiendo un *pipeline* para selección de atributos con la función selectKBest de Sklearn. Se utiliza este *pipeline* junto con un *Grid search* para obtener los mejores parámetros dentro de los valores que se tengan en la malla, entre los que se incluye el número de características a seleccionar del conjunto de datos. Al hacer uso de la función gridSearch, se está realizando implícitamente una validación cruzada con cada combinación de parámetros, por lo que para explotar todo el potencial de estas técnicas se necesita hardware más potente que el disponible, ya que estuvo más de 24 horas en ejecución, sin haber creado una malla de parámetros demasiado ambiciosa y usando 5 *folds* para la validación cruzada. Debido a esto se escogió usar la técnica de búsqueda aleatoria mediante la función RandomizedSearchCV de Scikit-Learn, en el cual se le pasa una malla de valores para los hiper-parámetros y realiza una búsqueda aleatoria entre las combinaciones disponibles usando un número determinado de iteraciones. Tras realizar esta búsqueda, obtenemos los mejores parámetros encontrados para entrenar el modelo.

Sobre los modelos XGB solo se aplicó la idea de la búsqueda aleatoria de hiper-parámetros debido a que no se obtuvieron mejoras notables al aplicar selección de atributos sobre los modelos RF y, en la mayor parte de los experimentos realizados a modo de pruebas, se escogían todas las características.

Se resumen los parámetros usados en cada modelo en las tablas 5.1 y 5.2. Todos los modelos XGB usan como parámetro *booster* los gbtree (*General*

Balanced Tree) y como *objective* se uso *binary:logistic* indicando que se trata de un clasificador binario. Los modelos XGB ajustados manualmente devuelven una probabilidad de pertenencia a la clase positiva o la clase 1, pudiéndose ajustar esta en función del coste de los errores de clasificación del modelo. Para este trabajo se ha considerado una buena opción mantener el umbral de probabilidad en 0.5, es decir, cualquier registro con una probabilidad mayor al 50 % de pertenecer a la clase positiva (ataque) será clasificado como tal y, en caso contrario, se clasificará como de la clase negativa.

Tabla 5.1: Parámetros usados en los modelos XGB

Model	colsample bytree	learning rate	max depth	eval metric	boost rounds
XGB-MOPT	0.6	0.025	20	log_loss	300
XGB-RSOPT	0.5	0.255	30	log_loss	257
XGB-RS	0.7	0.1325	10	error	242
XGB-LOGL	0.6	0.025	20	log_loss	300
XGB-ERR	0.6	0.025	20	error	300

Tabla 5.2: Parámetros usados en los modelos RF

Model	num estimators	max depth	criterion	num atts
RF-MOPT	241	49	gini	47
RF-RSOPT	285	40	log_loss	35
RF-RS	271	10	gini	45
RF-GINI	250	30	gini	47
RF-ENTR	250	30	entropy	47
RF-LOGL	250	30	log_loss	47

5.8. Resultados y comparativa

En esta sección se resumen los resultados que se han obtenido con los modelos implementados usando diferentes configuraciones de hiper-parámetros y con conjuntos de datos procesados de forma ligeramente diferente.

En un comienzo, se tuvo el error de normalizar los datos antes de obtener las particiones de entrenamiento y test, lo cual es incorrecto al estar utilizando información del conjunto de entrenamiento para procesar el conjunto de test que se usa posteriormente para validar el modelo, lo cual pueda arrojar resultados demasiado optimistas. Aun así se presentan los resultados obtenidos de este modo para comparar posteriormente estos resultados incorrectos u optimistas con los obtenidos con los conjuntos de

datos normalizados después de realizar las particiones. Se resumen estos resultados, indicando el tipo de procesado que se llevó a cabo sobre el conjunto de datos y los parámetros escogidos para el modelo.

Se presentan los resultados obtenidos con los diferentes conjuntos de datos, con los dos modelos implementados (*Random Forest Classifier* y *XGB Classifier*), tanto durante el ajuste manual de parámetros como con el ajuste de búsqueda aleatoria de parámetros. Las métricas *precision*, *recall* y *F1-score*, son relativas a la clase positiva (ataque). El *booster* utilizado con XGB es *gbtree* (*general balanced trees* y *objective* se establece como *binary:logistic* para realizar la clasificación binaria.

Para el ajuste manual de los modelos se partió de los parámetros por defecto y se fueron variando uno a uno los parámetros disponibles para determinar como afecta cada uno al modelo y se fueron ajustando hasta obtener los mejores resultados posibles sin que se produzca sobreentrenamiento. Para ello se observaba el valor de la función de perdida tanto en entrenamiento como en validación que se nos proporcionan activando la verbosidad, ya que cuando se produce el sobreentrenamiento vemos que la función de perdida en entrenamiento sigue disminuyendo en cada *epoch* o iteración del algoritmo, mientras que el valor de la función de perdida en evaluación comienza a estancarse y posteriormente a crecer.

Se presentan los resultados obtenidos con cada modelo usando diferentes configuraciones de hiper-parámetros y funciones de evaluación o perdida. Se muestran los resultados para el clasificador *Random Forest* (RF), con el conjunto de datos optimista tanto con ajuste manual (RF-MOPT) como con la selección aleatoria usando RandomizedSearchCV de Scikit-Learn (RF-RSOPT). Los mismo para el modelo XGB, se tiene el ajustado manualmente entrenado con el conjunto de datos optimista (XGB-MOPT), como el ajustado con RandomizedSearchCV (XGB-RSOPT). Luego se tienen los resultados de rendimiento de los modelos entrenados con el conjunto de datos correctamente procesado, haciendo que no se obtengan resultados tan optimistas. Para el *Random Forest* se tiene el ajustado con la función RandomizedSearchCV (RF-RS) y el ajustado manualmente con diferentes criterios para medir la impureza de la división al construir los árboles. Con el índice de Gini (RF-GINI), con la entropía (RF-ENTR) y con la perdida logarítmica o *log-loss* (RF-LOGL). Para el modelo XGB se utilizan el ajustado con al función de búsqueda aleatoria (XGB-RS), y los ajustados manualmente usando como criterios o funciones de evaluación la perdida logarítmica o log-loss (XGB-LOGL) o usando la tasa de error del modelo (XGB-ERR). Los resultados obtenidos se resumen en la tabla 5.3.

Tabla 5.3: Resultados de los diferentes ajustes sobre los modelos

Model	Accuracy	Precision	Recall	F1-score
RF-MOPT	0.9851	0.9943	0.9758	0.985
RF-RSOPT	0.9852	0.9944	0.9759	0.985
XGB-MOPT	0.9855	0.995	0.9759	0.9854
XGB-RSOPT	0.9847	0.9941	0.9752	0.9846
RF-RS	0.9378	0.9325	0.9439	0.9382
RF-GINI	0.9573	0.969	0.9448	0.9568
RF-ENTR	0.9568	0.9684	0.9445	0.9563
RF-LOGL	0.9568	0.9684	0.9445	0.9563
XGB-RS	0.9861	0.9939	0.9782	0.986
XGB-LOGL	0.9866	0.9943	0.9787	0.9865
XGB-ERR	0.9865	0.9942	0.9786	0.9863

A la hora de realizar los ajustes manuales y tras comprobar que, como se comenta en parte de la bibliografía disponible, los métodos basados en árboles de decisión son poco susceptibles del ajuste de los parámetros, al contrario de lo que ocurre con otro tipo de modelos, por lo que se puede simplificar bastante el ajuste de los hiper-parámetros realizando un ajuste sobre la cantidad y tamaño de los árboles generados de forma inicial, ya que son estos los parámetros que más pueden afectar al modelo y al tiempo de entrenamiento. Tras ajustar estos parámetros se estudia como se comporta el modelo al variar el resto de parámetros disponibles, descartando aquellos que no producen cambios significativos o incluso que empeoran el rendimiento del modelo al cambiar su valor por defecto. Tras obtener este ajuste inicial, se puede ir reduciendo la tasa de aprendizaje y aumentando el número de estimadores o clasificadores tratando de aumentar el rendimiento a costa de aumentar los tiempos de entrenamiento, por lo que se debe tener cuidado de mantener estos tiempos en valores aceptables.

Además de presentar la tabla de resultados, se presentan algunas matrices de confusión de los modelos que utilizan el conjunto de datos procesado de forma correcta, discriminando las de los modelos entrenados con el conjunto "optimista" ya que simplemente se han presentado sus resultados en la tabla para poder realizar una comparativa de como se comportan los modelos al usar un conjunto de datos procesado de forma incorrecta. Estos modelos deben ser rechazados independientemente de las métricas de rendimiento que arrojen. Podemos ver las matrices de confusión de los modelos en las figuras 5.1, 5.2, 5.3 y 5.4.

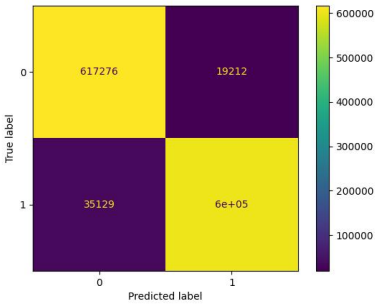

Figura 5.1: Matriz de confusión del modelo RF-GINI

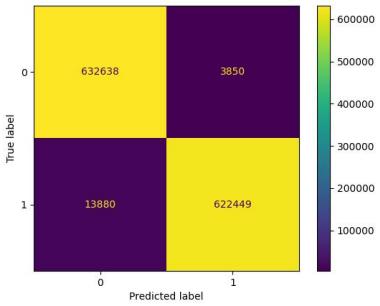

Figura 5.2: Matriz de confusión del modelo XGB-RS

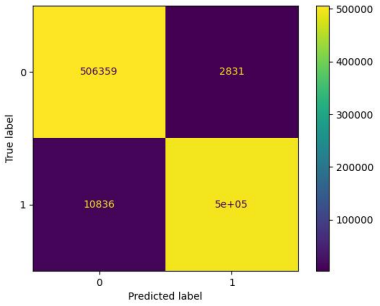

Figura 5.3: Matriz de confusión del modelo XGB-LOGL

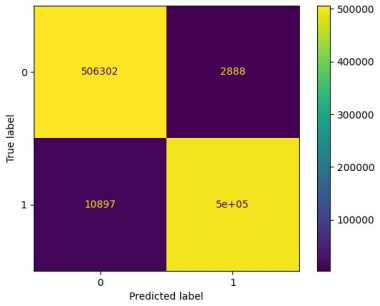

Figura 5.4: Matriz de confusión del modelo RF-RS

Se observa que para los modelos *Random Forest* se obtienen métricas muy optimistas al entrenarse con el conjunto de datos "optimista", observándose una diferencia aproximada del 3 % en el *accuracy*. Sin embargo en los modelos XGB se puede observar todo lo contrario. Las métricas de rendimiento son mejores usando el conjunto de datos procesado de forma correcta, mostrando la importancia de realizar un procesado correcto sobre los conjuntos de datos, ya que se haya observado una pérdida de rendimiento en los modelos RF a la hora de poner el modelo a trabajar con datos distintos a los de entrenamiento, por lo que tendrá una menor capacidad de generalización. Por otro lado, entre los modelos RF entrenados con el conjunto de datos correctamente procesado se observa que funciona ligeramente mejor al utilizar el índice de Gini frente a la entropía o la pérdida logarítmica. En cuanto a los XGB, la diferencia es mínima obteniendo los mejores resultados usando la tasa de error como función de evaluación.

Si comparamos los modelos que se han entrenado con el conjunto correctamente procesado, vemos que claramente XGB es superior a RF independientemente de la configuración de hiper-parámetros y función de evaluación utilizada. Además, podemos observar como al usar la búsqueda aleatoria para ajustar los modelos, los resultados no son tan buenos como con el ajuste manual debido a las limitaciones en términos de capacidad de computación. Para obtener mejores resultados habría que realizar exploraciones más extensas sobre mallas de parámetros algo más ambiciosas, pero el tiempo de ejecución es muy largo si no se tienen los medios adecuados. En este caso se puede tomar una búsqueda como situación de partida inicial si los resultados de la búsqueda son satisfactorios, realizando un ajuste fino de los hiper-parámetros de forma manual tras la búsqueda aleatoria.

En general, los resultados son buenos, siendo superiores usando XGB. El mejor modelo de los estudiados para el problema que nos ocupa es el modelo XGB, usando como función de perdida la perdida logarítmica y el ajuste de parámetros obtenido de forma manual, si tenemos en cuenta la tasa de acierto o *accuracy*. Como podría ser posible que, según como se vayan a gestionar las alertas generadas por el modelo, los falsos positivos y los falsos negativos podrían no ser igual de costosos Por ello, se presentan dos métricas gráficas, las curvas PR y ROC, que nos pueden ayudar a tener una visión más clara del rendimiento que podría tener el modelo y como podría ajustarse el modelo para tener una tasa menor de falsos positivos o falsos negativos, según el coste o el impacto que tengan estos. Se presentan estas curvas, en las figuras 5.5- 5.16, para cada uno de los modelos, de donde se podrán extraer algunas conclusiones de interés.

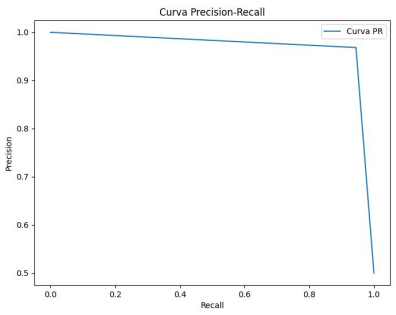

Figura 5.5: Curva PR del modelo RF-ENTR

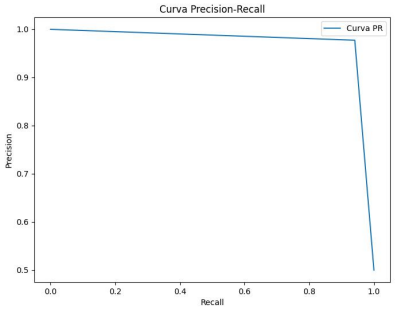

Figura 5.6: Curva PR de los modelos RF-GINI y RF-LOGL

Las curvas PR nos indican a partir de que *recall* se empieza a perder *precision* de forma considerable o viceversa. Esto nos puede ayudar a ajustar

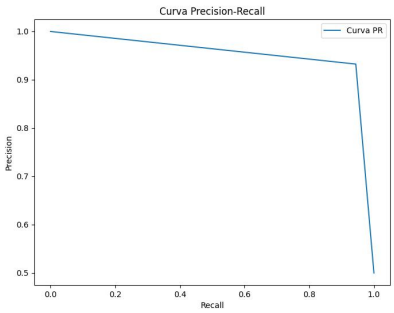

Figura 5.7: Curva PR del modelo RF-RS

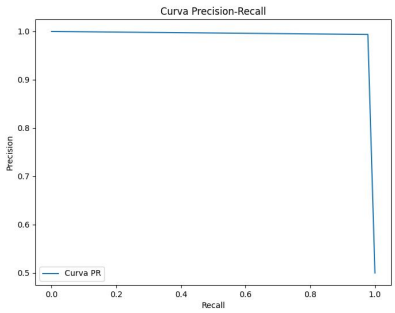

Figura 5.8: Curva PR del modelo XGB-RS

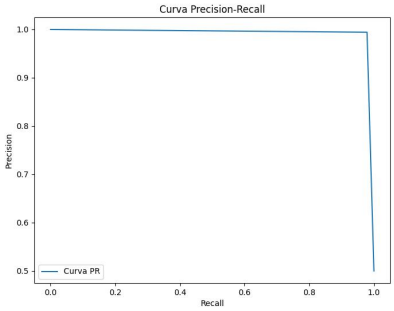

Figura 5.9: Curva PR del modelo XGB-LOGL

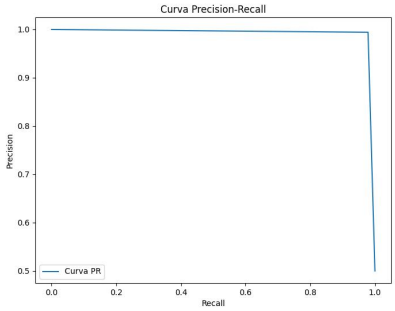

Figura 5.10: Curva PR del modelo XGB-ERR

los modelos en función de que sean más costosos los falsos positivos o falsos negativos. Esto dependerá en gran medida del tratamiento que se vaya a realizar de las alertas generadas por los modelos. Si, por ejemplo, se pretende que una persona gestione manualmente las alertas, se aumentará el costo de los falsos positivos, por lo que será necesario en este caso ajustar los pesos de las clases para aumentar en la medida de lo posible la métrica de *precision*. En cambio, si se van a tratar de forma automatizada, sin que una alerta suponga una interrupción del servicio, se podría considerar que se reduce el coste de los falsos positivos y se aumenta el de los falsos negativos, tomando un mayor peso el *recall* a la hora de escoger el mejor modelo. En términos generales, un modelo será mejor cuanto más se acerque la curva PR a la esquina superior derecha. En caso de alcanzar ese punto, se trataría de un modelo con *precision* y *recall* de 1, lo que sería la situación ideal, aunque lamentablemente no se suele dar esto en problemas reales. En el

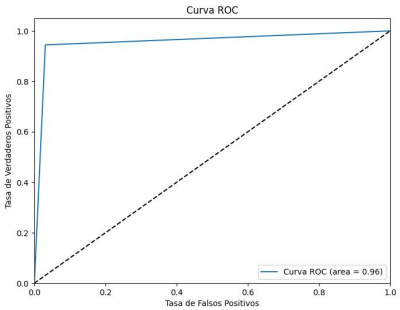

Figura 5.11: Curva ROC del modelo RF-ENTR

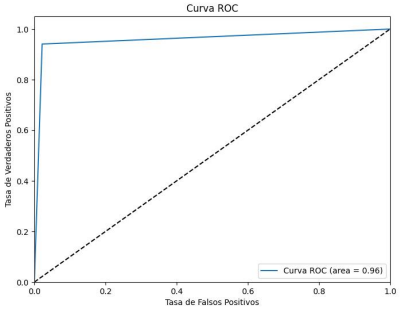

Figura 5.12: Curva ROC de los modelos RF-GINI y RF-LOGL

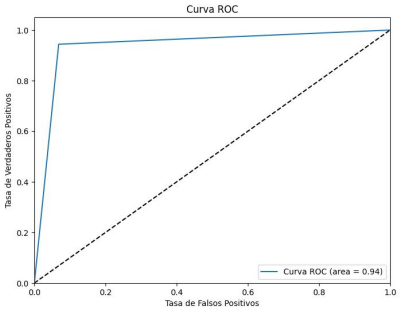

Figura 5.13: Curva ROC del modelo RF-RS

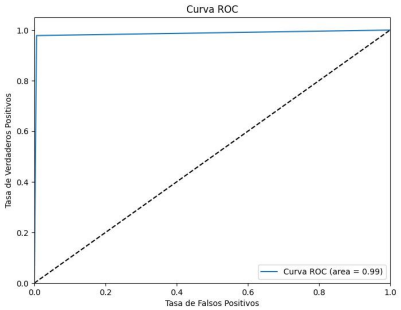

Figura 5.14: Curva ROC del modelo XGB-RS

caso de los modelos RF, se obtiene la misma curva PR al usar las funciones de evaluación índice de Gini y *logloss*. Si lo comparamos con la obtenida usando la función de entropía vemos que al usar la entropía el modelo es ligeramente menos eficaz, ya que en el vértice que muestra la curva en la parte superior derecha, se mantiene el *recall* aproximadamente igual, pero la precisión se puede ver que disminuye respecto a usar las otras dos funciones de evaluación. Por otro lado, podemos ver que para los modelos XGB se obtienen curvas idénticas tanto para el modelo que utiliza la función de evaluación *logloss* como para el que usa la tasa de error. Entre estas dos y la obtenida para el modelo con optimización mediante búsqueda aleatoria, las diferencias son mínimas, siendo ligeramente mejor para los modelos que utilizan las funciones de evaluación *logloss* y *error*, al estar el vértice que se ve en la curva más cercano a la esquina superior derecha. Por otro lado, entre los diferentes modelos RF y los modelos XGB, claramente son superiores los

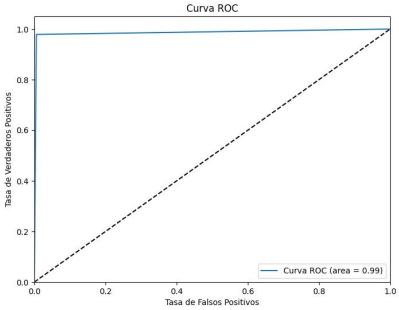

Figura 5.15: Curva ROC del modelo XGB-LOGL

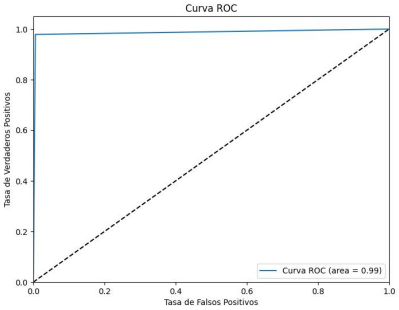

Figura 5.16: Curva ROC del modelo XGB-ERR

modelos XGB tanto si nos fijamos en las métricas presentadas anteriormente (*accuracy, precision, recall,f1-score*) como si utilizamos las curvas PR.

Si nos fijamos en las curvas ROC de los modelos podemos ver la representación de las tasas de verdaderos positivos (TVP), que representan capturas de tráfico maligno detectadas, frente a la tasa de falsos positivos (TFP), que representan capturas de tráfico normal o benigno que han sido clasificadas como tráfico maligno. En esta gráfica se puede ver como aumenta la TFP en función de la TVP. Se tienen dos regiones claramente diferenciadas. En la primera vemos como aumenta rápidamente la TVP sin que esto suponga un incremento significativo de la TFP. Al llegar a un determinado punto, ocurre totalmente lo contrario, un pequeño aumento en la TVP implica un gran aumento de la TFP, lo cual no es conveniente que ocurra. Teniendo esto en cuenta podemos ver como claramente los modelos XGB ofrecen un mejor rendimiento en estos términos frente a los modelos RF al tener curvas más cercanas al punto (0,1) del gráfico y con un mayor área bajo la curva ROC. Por otro lado, entre los modelos RF no se ven apenas diferencias entre los ajustados manualmente. Si comparamos estos con el optimizado mediante búsqueda aleatoria, vemos que los ajustados manualmente ofrecen un mayor rendimiento independientemente de la función de evaluación empleada con un área bajo la curva de 0.96 frente a los 0.94 del modelo que implementa la optimización aleatoria. Entre los XGB ocurre lo mismo, todas sus curvas ROC son muy similares, arrojando valores de área bajo la curva de 0.99 en todos los casos, incluido el optimizado mediante búsqueda aleatoria, pasando muy cerca del punto ideal (0,1).

Haciendo un balance general con todas las métricas de rendimiento consideradas hasta el momento, los modelos XGB han sido claramente

superiores en todas las comparativas realizadas a los modelos RF. De entre los diferentes XGB, Los modelos ajustados manualmente han demostrado ser ligeramente superiores al optimizado mediante búsqueda aleatoria. Entre estos dos, se podría decir que el que utiliza la función de evaluación *logloss* ofrece mejor rendimiento si nos fijamos en los resultados de la tabla aunque la diferencia es mínima.

5.9. Análisis de la calidad del código

Para el análisis de la calidad del código se ha usado la herramienta Codacy, la cual nos permite realizar revisiones sobre el código generado, indicando código duplicado, problemas de seguridad conocidos, etc. Para usar esta herramienta es tan sencillo como registrarse, acceder con la cuenta de GitHub y seleccionar el repositorio a añadir. Una vez hecho esto veremos nuestro repositorio en la plataforma de Codacy como se muestra en la figura 5.17.

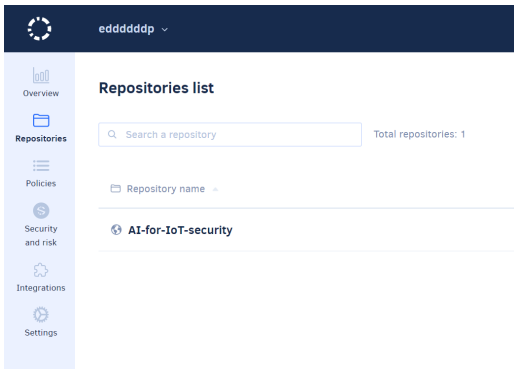

Figura 5.17: Repositorio GitHub integrado en Codacy

Al acceder al repositorio realizará un análisis del código alojado en el repositorio, tras el cual podremos ver el reporte generado. En la sección *Dashboard* se muestra un resumen de los resultados del análisis. En la figura 5.18 podemos ver el resumen del reporte obtenido. Como vemos se tienen algunos problemas de seguridad y de duplicidad de código. Podemos ver más en detalle estos problemas donde nos encontramos con que se tienen algunas líneas de código duplicadas. En este proyecto es normal tener algo de código duplicado ya que en diferentes módulos se realizan operaciones muy similares. Aun así se trató de reducir esto implementando funciones para la carga y guardado de los modelos.

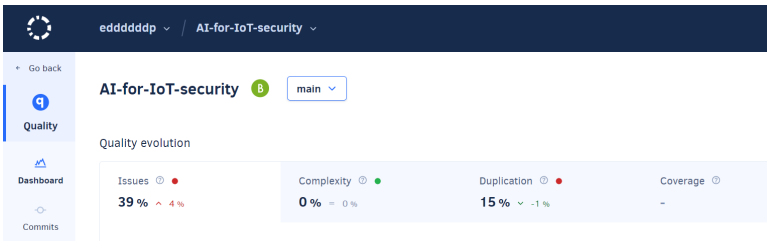

Figura 5.18: Resumen del análisis de la calidad del código

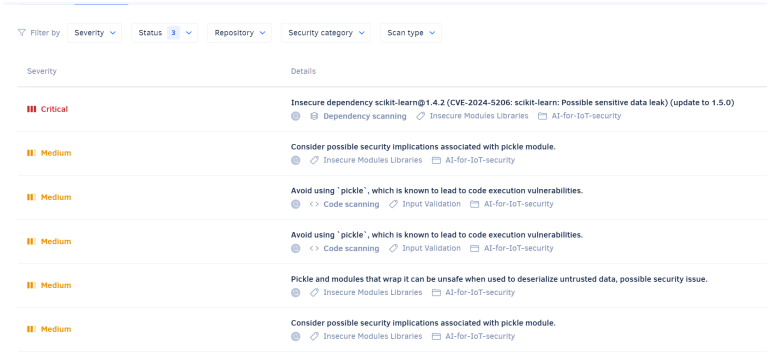

Figura 5.19: Algunos de los problemas de seguridad detectados por Codacy

En cuanto a los problemas de seguridad, se tienen 18 avisos. En los requerimientos se añadió la versión que se ha usado a lo largo del trabajo, la 1.4.2, la cual ha sido detectada como insegura al tener problemas de seguridad conocidos. Se soluciona este problema actualizando a la versión 1.5.0 y modificado el fichero de requerimientos. Cuando se configuró el entorno, esta era la última versión pero, en abril de 2024 se lanzó la versión 1.5.0 que corrige estos problemas de seguridad. El resto de problemas de seguridad tienen que ver con el uso de la biblioteca Pickle, como podemos ver en la figura 5.19. El uso de esta biblioteca debe hacerse con precaución ya que se corre el riesgo de deserializar código malicioso que pueda ser ejecutado en nuestro equipo. Las alternativas conocidas como la biblioteca Joblib, introduce los mismos problemas, por lo que se debe proceder con precaución a la hora de cargar los modelos y solo tratar de cargar modelos guardados en ficheros cuyo origen sea conocido y de confianza.

Tras corregir el problema de seguridad, podemos ver como desaparece este problema al recargar el reporte como se aprecia en la figura 5.20.

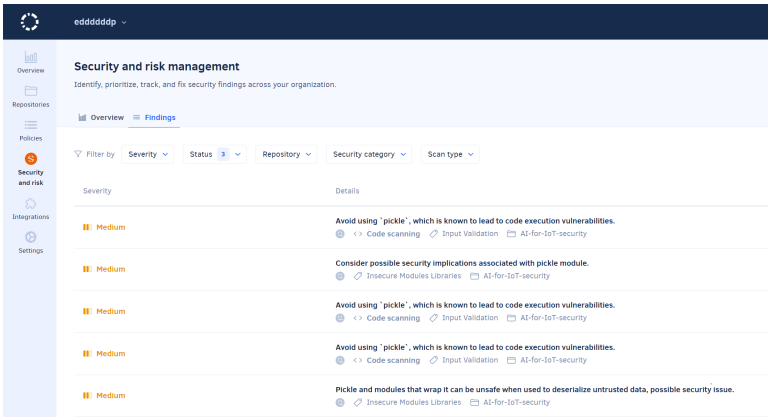

Figura 5.20: Problemas de seguridad tras resolver el problema de versión de Scikit-Learn

6. Trabajos relacionados

En esta sección se presenta un breve resumen del estado del arte actual en materia de aplicación de técnicas de aprendizaje automático para la protección de dispositivos IoT. Se dará un vistazo general a algunos de los trabajos realizados en los últimos años para ponernos en contexto sobre el panorama actual de la literatura disponible en la materia.

En septiembre de 2020 se publica el trabajo de Abdullah Alsaedi et al. [3], donde ponen de manifiesto la carencia de conjuntos de datos sobre datos de rendimiento y tráfico de red de dispositivos IoT, proponiendo algunos de los más utilizados debido a su completitud y calidad. También ponen a prueba algunos de estos conjuntos de datos analizados entrenando algunos modelos con estos, obteniendo buenos resultados en algunos casos, pero no así en otros. Con este trabajo pretendían allanar el camino para el diseño de este tipo de sistemas de detección en el futuro.

Unos meses después se publica el trabajo realizado por Nour Moustafa [12], concretamente en mayo de 2021. El autor reafirma la opinión de los autores del anterior trabajo comentando sobre la falta de arquitecturas distribuidas para generar conjuntos de datos de calidad, que mantengan los comportamientos reales de las redes IoT en escenarios complejos. Posteriormente se validan los conjuntos de datos "The TON_IoT datasets", de los que se extraen los datos usados en este trabajo de fin de grado, utilizando cuatro algoritmos de aprendizaje automático, en los cuales se obtienen resultados prometedores.

En octubre de 2021 se publica en el IEEE el trabajo de Abdallah R. Gad et al. [10]. En este trabajo se propone mejorar la seguridad de los dispositivos empleados para el control del transporte inteligente, haciendo énfasis en la falta de trabajos que hagan uso de conjuntos de datos más actualizados que

recojan datos de los ataques más recientes, proponiendo los conjuntos de datos de "The TON_IoT datasets" para ampliar las capacidades de este tipo de sistemas de detección de intrusiones, aplicando técnicas como Chi^2 para la selección de características o SMOTE para el balanceado de las clases, obteniendo buenos resultados con algunos de los algoritmos de clasificación utilizados.

Un año después, en octubre de 2022, Mohammad Shahin et al. [15] presentan un trabajo dando un enfoque ligeramente distinto a los anteriores trabajos al utilizar redes neuronales totalmente convolucionales para la protección de dispositivos IoT en entornos industriales y de fabricación inteligente obteniendo resultados muy buenos con diferentes conjuntos de datos, entre los que se encuentra "The TON_IoT datasets".

En octubre de ese mismo año se presento otro trabajo elaborado por Abdallah R. Gad et al. [9] en el cual se utilizan los conjuntos de datos de "The TON_IoT datasets" para clasificación multiclase usando los conjuntos de datos sobre uso de memoria y el estado de los diferentes procesos en ejecución, tanto en sistemas Linux como en Windows 7 y Windows 10, obteniendo en algunos casos tasas de acierto del 100 %.

En diciembre de 2022 se publica el trabajo de Rania Elsayed et al. [7], donde tratan este problema tratando de adaptarlo a las actuales ciudades inteligentes habilitadas para IoT utilizando potentes redes neuronales, donde consiguieron resultados superiores al 98 %.

En julio de 2023 se acepta y publica el trabajo elaborado por Abdulmuneem BasHaiwth et al. [4] donde utilizan redes LSTM (Long Short-Term Memory) para clasificar ataques DDoS, utilizando diferentes métodos de interpretación para las predicciones del modelo, obteniendo tasas de acierto superiores al 97 %.

En octubre de 2023 publican el trabajo de Rania Elsayed et al. [8], donde continúan trabajando en el estudio de sistemas basados en aprendizaje automático para proteger redes de dispositivos IoT y redes definidas por software (SDN), haciendo énfasis en la baja precisión y escalabilidad que tienen algunas de las técnicas empleadas anteriormente. En este trabajo utilizan una red mejorada LSTM capaz de distinguir entre tráfico malicioso y benigno, identificando la categoría del ataque y prediciendo el ataque concreto con precisiones superiores al 96 %.

Por último se presenta el trabajo de Ramkumar Devendiran et al. [6], previsto para su publicación en *Expert System whit Applications* el 1 de julio de 2024. En este trabajo se propone utilizar redes neuronales utilizando

una estrategia de optimización estocástica o caótica, utilizando el algoritmo *Chaotic Honey Badger* para la selección óptima de características y usando para la clasificación el algoritmo Dugat-LSTM (Gated Attention Dual Long Short-Term Memory), donde obtienen una tasa de acierto cercana al 99 %.

7. Conclusiones y Líneas de trabajo futuras

Se presentan en esta sección las conclusiones más relevantes que se han extraído a lo largo de la realización del trabajo, así como presentar próximos pasos o futuras líneas de trabajo que complementen a este.

7.1. Conclusiones

En la realización de este trabajo se ha tenido la oportunidad de estudiar los trabajos más recientes en cuanto a la aplicación de técnicas de aprendizaje automático a la ciberseguridad, permitiéndome conocer el punto en el que se encuentra este área de la informática actualmente. Por otro lado, he podido experimentar con algunos modelos de inteligencia artificial y con el tratamiento de los datos, donde se ha podido comprobar la importancia de realizar un correcto tratamiento de los mismos para la obtención de resultados realistas como se pudo comprobar al no procesar correctamente los datos en un inicio, haciendo que se obtuviesen resultados demasiado optimistas. En este sentido me pude encontrar con varios problemas a la hora de codificar algunos atributos, a los que se trato de poner diferentes soluciones, como fue el caso cuando se trató de codificar las direcciones IP de diversos modos (Por valor, por octetos, etc.), aunque esto se terminó descartando al ver que no aportaba información útil y se terminó por eliminar estas características junto a los números de puerto de origen y destino así como varias características con varianzas muy bajas, lo que hizo que se obtuviesen mejores resultados al tener el conjunto de datos más limpio.

Realizar este trabajo también me ha permitido profundizar en algunos conceptos relativos a los modelos de clasificación, así como conocer modelos más complejos y avanzados que los vistos en el grado, aunque no se terminaron de utilizar debido a la complejidad en su ajuste, como ocurrió con las redes LSTM, obteniendo resultados bastante pobres en el mejor de los casos. Como la disponibilidad de tiempo es limitada, se siguió experimentando con otros modelos más sencillos, como son los métodos basados en árboles de decisión, que no son tan susceptibles del ajuste de los parámetros y al ruido en los datos entre otros inconvenientes, que debido a la naturaleza de los datos, se presenta como una ventaja considerable ya que es inevitable introducir ruido al provenir de capturas de tráfico en entornos controlados y no de tráfico de redes reales.

En base a los resultados obtenidos con el modelo implementado con la biblioteca XGBoost, se podría decir que se han conseguido alcanzar los objetivos perseguidos al obtener tasas de acierto superiores al 98 % en la clasificación de tráfico malicioso y benigno, mostrando que es viable la clasificación del tráfico de red para proteger redes IoT. Esto permite definir futuros trabajos que mejoren los resultados obtenidos y que nos acerquen un poco más a la implementación de sistemas de detección de intrusiones en redes reales capaces de detectar ataques en tiempo real, o al menos detectarlos en el menor tiempo posible tras su materialización. Además, en caso de utilizar el *booster* con la biblioteca XGBoost, se podría realizar un nuevo procesado de datos manteniendo las características categóricas convirtiendo estas columnas de características al tipo de datos de pandas *category*, y ver si se obtienen los mismos resultados, ya que esto podría reducir la carga del sistema a la hora de procesar capturas en tiempo real, a la par que se aceleraría el proceso de predicción al utilizar un menor número de características. Además, al devolvernos estos modelos una probabilidad de pertenencia a la clase positiva, nos permite poder ajustar este valor según el coste que se considere para los falsos positivos y falsos negativos.

En definitiva, el realizar este trabajo, a parte de permitirme afianzar conceptos importantes en materia de ciberseguridad y aprendizaje automático, me ha devuelto mi interés por la inteligencia artificial, el cual había ido perdiendo poco a poco al ver que no se profundizaba mucho en el grado en estos temas además de verlo desde un punto de vista excesivamente teórico (aunque mi visión cambio un poco en las últimas sesiones prácticas de la asignatura minería de datos), y el acercarme un poco más al mundo de la ciberseguridad el cual me llevaba llamando la atención desde antes de emprender mis estudios de ingeniería informática.

7.2. Líneas de trabajo futuras

Tras la realización de este trabajo se pueden proponer algunos trabajos que podrían desarrollarse para mejorar los resultados obtenidos o seguir avanzando en la persecución de sistemas de detección de intrusiones basadas en la clasificación automática del tráfico de red. Entre las ideas que me han surgido en el desarrollo del presente trabajo, me gustaría destacar las siguientes:

Implementar clasificación multiclase Se podría continuar este trabajo tratando de buscar una solución para implementar la clasificación multiclase para, tras detectar un ataque, clasificar este dentro de alguna de las categorías de ataques contemplados en el conjunto de datos, acotando las medidas a tomar, las cuales en función del tipo de ataque detectado podrían incluso aplicarse de manera automática (regeneración de claves, modificación automática de las configuraciones de seguridad, etc.). Para esto debería estudiarse la posibilidad de emplear redes LSTM, por ejemplo, viendo que en la literatura disponible se han logrado buenos resultados en esta tarea aplicando diferentes enfoques en la interpretación de los resultados.

Estudio de la viabilidad de la clasificación en tiempo real Se tendría que estudiar la viabilidad de clasificar todo el tráfico de una red IoT, teniendo en cuenta que pueden estar conformadas por un elevado número de dispositivos, en tiempo real e implementar un sistema capaz de realizar esta tarea, solventando problemas que pudiesen aparecer, como podría ser el caso de necesitar almacenar parte del tráfico de la red hasta que sea clasificado en momentos puntuales con un tráfico elevado, sin que se produzca acumulación continua de tráfico que impida que se clasifique en un tiempo aceptable.

Implementación de un sistema de codificación Para continuar con la idea de construir un sistema de detección de intrusiones, sería necesario implementar un sistema capaz de recoger todo el tráfico de la red y codificarlo adecuadamente para que nuestro modelo sea capaz de clasificarlo. Como en las redes IoT es frecuente encontrarnos con situaciones de elevado tráfico, podría ser conveniente añadir un alto grado de paralelismo en el sistema destinado a estos fines.

Búsqueda de modelos de clasificación rápida Se podría realizar un estudio utilizando diferentes algoritmos de clasificación, donde el objetivo sea medir la capacidad de clasificación de cada algoritmo conside-

rado para el problema que nos ocupa y realizar una comparación de los resultados de clasificación para tener una visión más amplia del comportamiento que podemos esperar de cada modelo, lo cual ayudará a escoger los mejores algoritmos para el fin perseguido. Aunque se ha tratado de utilizar algoritmos en los que la clasificación se realice con un bajo coste, puede ser interesante el realizar un estudio centrado en este aspecto.

Poner los modelos a prueba Se podría construir o emular una red IoT, donde se comenzaría a generar tráfico mientras ponemos a trabajar el clasificador entrenado. A su vez se ejecutarían o emularían ataques de diferentes tipos sobre esta red para obtener una evaluación más precisa de la capacidad de generalización de nuestros modelos.

Creación de nuevos conjuntos de datos Un problema que puntualizan diversos autores de trabajos relaciones es la falta de conjuntos de datos de redes IoT que abarquen los posibles ataques a los que los dispositivos IoT son susceptibles y que además se ajusten a entornos reales, ya que los que se tienen disponibles han sido generados en entornos controlados. Otros conjuntos de datos que si utilizan tráfico real, como los generados y usados en algunos trabajos disponibles en la literatura, no están disponibles públicamente debido a que las leyes de protección de datos lo impiden. Por ello, sería interesante buscar un modo en el cual se pueda generar un conjunto de datos lo suficientemente completo y que, además, se ajuste al tráfico real que podamos encontrar en este tipo de redes.

Estudiar requisitos y costes de hardware Una vez finalizados con éxito algunos de los trabajos anteriormente mencionados, se deberían estudiar los requisitos y los costes del hardware necesario para ejecutar todas las tareas necesarias en tiempos aceptables, para poder desplegar este tipo de sistemas de modo que sean capaces de proteger las redes IoT que se deseen. Para ello se debería de estudiar cada caso concreto, ya que dependerá del tamaño y trafico habitual en cada red, y buscando además la máxima escalabilidad posible, ya que este tipo de redes suelen tener una tendencia de crecimiento cada vez mayor.

Bibliografía

[1] 3GPP. Standardization of NB-IOT completed. https://www.3gpp.org/news-events/3gpp-news/nb-iot-complete, 2016.

[2] LoRa Alliance. TS001-1.0.4 LoRaWAN® L2 1.0.4 Specification. https://resources.lora-alliance.org/technical-specifications/ts001-1-0-4-lorawan-l2-1-0-4-specification, 2020.

[3] Abdullah Alsaedi, Nour Moustafa, Zahir Tari, Abdun Mahmood, and Adnan Anwar. TON_IoT Telemetry Dataset: A New Generation Dataset of IoT and IIot for Data-Driven Intrusion Detection Systems. *IEEEAccess*, 8, 2020.

[4] Abdulmuneem Bashaiwth, Hamad Binsalleh, and Basil AsSadhan. An explanation of the LSTM Model Used for DDoS Attacks Classification. *MDPI*, 2023.

[5] Scikit-Learn Developers. Sklearn.metrics.f1_score — Scikit-Learn User Guide. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html, 2024. [Internet; Consultado el 10/05/2024].

[6] Ramkumar Devendiran and Anil V Turukmane. Dugat-LSTM: Deep learning based network intrusion detection system using chaotic optimization strategy. *Expert Systems with Applications*, 245, 2024.

[7] Rania Elsayed, Reem Hamada, Mohammad Hammoudeh, Mahmoud Abdalla, and Shaimaa Ahmed Elsaid. A hierarchical Deep Learning-Based Intrusion Detection Architecture for Clusteres Internet Of Thing. *Journal of Sensor and Actuator Networks*, 2022.

[8] Rania A. Elsayed, Reem A. Hamada, Mahmoud I. Abdalla, and Shaimaa Ahmed Elsaid. Securing IoT and SDN systems using deep-learning based automatic intrusion detection. *Ain Shams Engineering Journal*, 14(10), 2023.

[9] Abdallah R. Gad, Mohamed Haggag, Ahmed Nashat, and Tamer M. Barakat. A Distributed Intrusion Detection System using Machine Learning for IoT based on ToN_IoT Dataset. *International Journal of Advanced Computer Science and Applications*, 16:548–563, 2022.

[10] Abdallah R. Gad, Ahmed A. Nashat, and Tamer M. Barkat. Intrusion Detection System Using Machine Learning for Vehicular Ad Hoc Networks Based on ToN_IoT Dataset. *IEEEAccess*, 9, 2021.

[11] Xiaowei Huang, Gaojie Jin, and Wenjie Ruan. *Machine Learning Safety*. Springer, 2023.

[12] Nour Moustafa. A new distributed architecture for evaluating AI-based security systems at the edge: Network TON_IoT datasets. *ResearchGate*, 2021.

[13] Emilio Soria olivas, Manuel Antonio Sanchez montañés Isla, Ruth Gamero Cruz, Borja Castillo Caballero, and Pedro Cano Michelena. *Sistemas de Aprendizaje Automático*. Ra-Ma, 2023.

[14] Sandeep Saxena and Ashok Kumar Pradhan. *Internet of Things: Security and Privacy in Cyberspace*. Springer, 2022.

[15] Mohammad Shahin, F. Frank Chen, Hamed Bouzary, Rasoul Rashidifar, and Ali Hosseinzadeh. A novel fully convolutional neural network to approach for detection and classification of attacks on industrial IoT devices in smart manufacturing systems. *International Journal of Advanced Computer Science and Applications*, 2022.

[16] Wikipedia. F-score — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/F-score, 2024. [Internet; Consultado el 10/05/2024].

[17] Wikipedia. Precision and recall — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Precision_and_recall, 2024. [Internet; Consultado el 10/05/2024].

[18] Zhi-Hua Zhou. *Machine Learning*. Springer, 2021.