Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Grado en Ingeniería Informática

Trabajo Fin de Grado

Creación de un Modelo ML para el Despliegue de IDS en Entornos Empresariales

Autor: Alejandro Ortega Hernández
Tutor: Antonio Jesús Díaz Honrubia

Madrid, julio 2025

Este Trabajo Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Grado
Grado en Ingeniería Informática

Título: Creación de un Modelo ML para el Despliegue de IDS en Entornos Empresariales

julio 2025

Autor: Alejandro Ortega Hernández
Tutor: Antonio Jesús Díaz Honrubia
Departamento de Lenguajes y Sistemas Informáticos e Ingeniería de Software
Escuela Técnica Superior de Ingenieros Informáticos
Universidad Politécnica de Madrid

Resumen

La seguridad de las tecnologías de la información es un factor clave en un mundo en el que todo está digitalizado. Bancos, colegios, gobiernos, servicios sanitarios y otras muchas entidades manejan a diario datos sensibles. Para protegerlos se puede recurrir a tecnologías de cifrado, autenticación y otro tipo de medidas para conservar la integridad y evitar que aquel que no deba, no pueda acceder a los mismos. No obstante estos sistemas no sirven de nada si una máquina con acceso a estos datos en claro se encuentra expuesta o esta siendo objetivo de un ataque a través de una red. Es por eso que la detección temprana de este tipo de ataques es clave en la mitigación de daños por parte de todos los organismos previamente citados.

Este Trabajo de Fin de Grado se centra en el desarrollo de un modelo de machine learning (ML) orientado a la detección de intrusiones en redes empresariales. Partiendo del uso de dos datasets ampliamente utilizados en ciberseguridad, CICIDS2017 y CICIDS2018, se lleva a cabo un proceso completo de preparación de datos que incluye limpieza, normalización, categorización de puertos y codificación de variables. Se estudian varios algoritmos de aprendizaje supervisado, entre ellos Random Forest y XGBoost, seleccionando este último por su alto rendimiento y eficiencia en el entrenamiento.

El modelo final se integra en una API desarrollada en Python y desplegada mediante Docker, permitiendo su uso en distintos entornos de forma sencilla y escalable. Para validar su efectividad, se diseña un entorno simulado que emula una red empresarial con tráfico legítimo y malicioso, obteniendo resultados muy satisfactorios: una precisión del 98,5 % y un recall del 98,49 %.

Además del desarrollo técnico, se plantean estrategias de despliegue y escalabilidad, facilitando su adopción en entornos reales. El trabajo demuestra que es posible construir una base sólida para sistemas IDS más eficientes y adaptables mediante el uso de técnicas de ML aplicadas a datos de red bien procesados.

Abstract

IT security is a key factor in a world where everything is digitalised. Banks, schools, governments, health services and many other organisations handle sensitive data on a daily basis. To protect this data, encryption technologies, authentication and other measures can be used to preserve its integrity and prevent access to it by those who should not have access to it. However, these systems are useless if a machine with access to this data is exposed or targeted by a network attack. That is why early detection of such attacks is key to the mitigation of damage by all the organisations mentioned above.

This Final Degree Project focuses on the development of a machine learning (ML) model for intrusion detection in enterprise networks. Using two widely recognized cybersecurity datasetsCICIDS2017 and CICIDS2018 complete data preparation pipeline was implemented, including cleaning, normalization, port categorization, and variable encoding. Several supervised learning algorithms were evaluated, with XGBoost ultimately selected due to its strong performance and training efficiency.

The final model was integrated into a Python-based API and deployed using Docker, enabling easy and scalable use in different environments. To validate its effectiveness, a simulated enterprise network environment was created with both benign and malicious traffic, yielding highly satisfactory results: 98.5% precision and 98.49% recall.

Beyond the technical implementation, the project proposes deployment and scalability strategies to support real-world adoption. The results demonstrate that it is possible to build a solid foundation for more efficient and adaptable IDS solutions by leveraging ML techniques applied to well-processed network traffic data.

Tabla de contenidos

Índice de figuras

1. Introducción

La creciente dependencia de la tecnología en empresas e industrias ha aumentado mucho el riesgo de sufrir ataques cibernéticos. Esto no solo pone en peligro información importante, sino también infraestructuras críticas necesarias para el día a día de estas organizaciones. Por eso, los Sistemas de Detección de Intrusos (IDS, por sus siglas en inglés) se han vuelto esenciales para proteger las redes, ya que supervisan y analizan constantemente el tráfico para detectar actividades sospechosas o potencialmente peligrosas.

Los ataques de denegación de servicio (DoS) y denegación de servicio distribuido (DDoS) constituyen amenazas críticas para la disponibilidad de servicios en red. Los DDoS, en particular, han mostrado un aumento significativo en frecuencia y escala, con incidentes que superan los 2 Tbps reportados en los últimos años [1]. Estos ataques, al provenir de múltiples orígenes simultáneamente, dificultan su mitigación mediante técnicas convencionales. Por otro lado, los ataques por fuerza bruta siguen siendo ampliamente utilizados, especialmente para comprometer servicios expuestos como SSH o RDP, donde los atacantes automatizan intentos masivos de autenticación utilizando diccionarios de contraseñas [2].

Las botnets, redes de dispositivos comprometidos utilizados para orquestar actividades maliciosas, son una de las principales fuentes de tráfico DDoS y de ataques coordinados [3]. Su capacidad de operar de forma distribuida y encubierta las convierte en una amenaza persistente. En el ámbito de la seguridad web, los ataques de *cross-site scripting* (XSS) permiten a los atacantes inyectar código malicioso en aplicaciones vulnerables, comprometiendo la integridad de los datos y la privacidad de los usuarios [4]. La detección eficaz de estos ataques en tiempo real requiere modelos capaces de identificar patrones de tráfico anómalos y adaptarse dinámicamente a nuevas variantes, lo que justifica la aplicación de técnicas de *machine learning* en soluciones IDS modernas [5].

Aunque son herramientas fundamentales, los sistemas IDS enfrentan problemas importantes. Uno de los principales es la gran cantidad de falsos positivos, es decir, alertas erróneas que pueden saturar a los equipos de seguridad y dificultar que se identifiquen las amenazas reales a tiempo. Otro problema común es que estos sistemas deben adaptarse a diferentes entornos, y su desempeño puede variar mucho según la calidad de los datos utilizados para entrenarlos.

Este Trabajo de Fin de Grado busca resolver estos problemas utilizando técnicas de Machine Learning (ML). La idea es mejorar cómo se preparan y seleccionan

los datos que se usan para entrenar los IDS, buscando así reducir los falsos positivos y mejorar la eficacia operativa. También se busca hacer estos sistemas más adaptables y escalables para que puedan funcionar correctamente en distintos tipos de empresas.

Para lograrlo, el proyecto se divide en varias fases. Primero, se realizó estudio sobre IDS y ML, y se analizaron soluciones comerciales y de código abierto ya existentes. Además, se eligieron datasets conocidos y representativos del tráfico real de redes, como CICIDS2017 y CICIDS2018. En la fase de preparación de los datos, se llevaron a cabo tareas como limpieza, normalización y segmentación del tráfico.

En la fase de desarrollo del modelo, se utilizaron algoritmos como Random Forest y XGBoost comparando el rendimiento de cada uno de los modelos junto con el tiempo necesario para su entrenamiento. Tras esto se puso a prueba el modelo resultante prediciendo si diferentes patrones de tráfico en una red simulada son o no tráfico malicioso.

Finalmente este trabajo ofrece una comparativa del rendimiento de los modelos obtenidos con diferentes algoritmos así como los resultados del modelo final tras su prueba con una simulación de tráfico en una red simulada, y puede ofrecer un modelo base para implementar en una empresa como capa base de un IDS que integre ML.

1.1. Objetivos

El objetivo principal de este Trabajo de Fin de Grado es generar, probar e implementar un modelo de machine learning optimizado para la detección de intrusiones en redes. El modelo no se enfoca en detectar el tipo de ataque al que se ve expuesta la red sino que en detectar si el tráfico es anómalo o no.

Los objetivos necesarios para completar el objetivo principal son:

1. Preparar y procesar un conjunto de datos de tráfico de red para su uso en el entrenamiento del modelo.
2. Estudiar y evaluar algoritmos de aprendizaje supervisado adecuados para la detección de intrusiones.
3. Entrenar un modelo de machine learning optimizado para la detección de intrusiones.
4. Evaluar el rendimiento del modelo utilizando métricas adecuadas.
5. Probar el modelo con un entorno simulado en el que se simule un ataque.
6. Proporcionar recomendaciones sobre escalabilidad.

1.2. Planificación del trabajo

Estudio previo

- T1. Revisión de literatura sobre IDS y machine learning en ciberseguridad.
- T2. Análisis de soluciones IDS comerciales y opensource.
- T3. Busqueda y estudio de datasets de tráfico de red.

Procesamiento y preparación de datos

- T4. Definir criterios para la selección de datos relevantes.
- T5. Implementación de procesos de limpieza.
- T6. Etiquetado y segmentación de tráfico.
- T7. Extracción de características para optimizar el modelo.

Deasarrollo del modelo y optimización

- T8. Selección de algoritmos de machine learning adecuados para soluciones IDS.
- T9. Implementación de un modelo inicial.
- T10. Fine-tuning del modelo incial.
- T11. Tests sobre el modelo inicial.

Entorno de pruebas

- T12. Configuración de un entorno empresarial simulado.
- T13. Inyección de tráfico malicioso y evaluación del modelo.

Escalabilidad

- T14. Plan de despliegue y escalabilidad en el entorno.

Documentación y entrega

- T15. Redacción de la memoria.
- T16. Modificaciones por recomendaciones del tutor.
- T17. Preparación de la presentación y defensa del TFG.

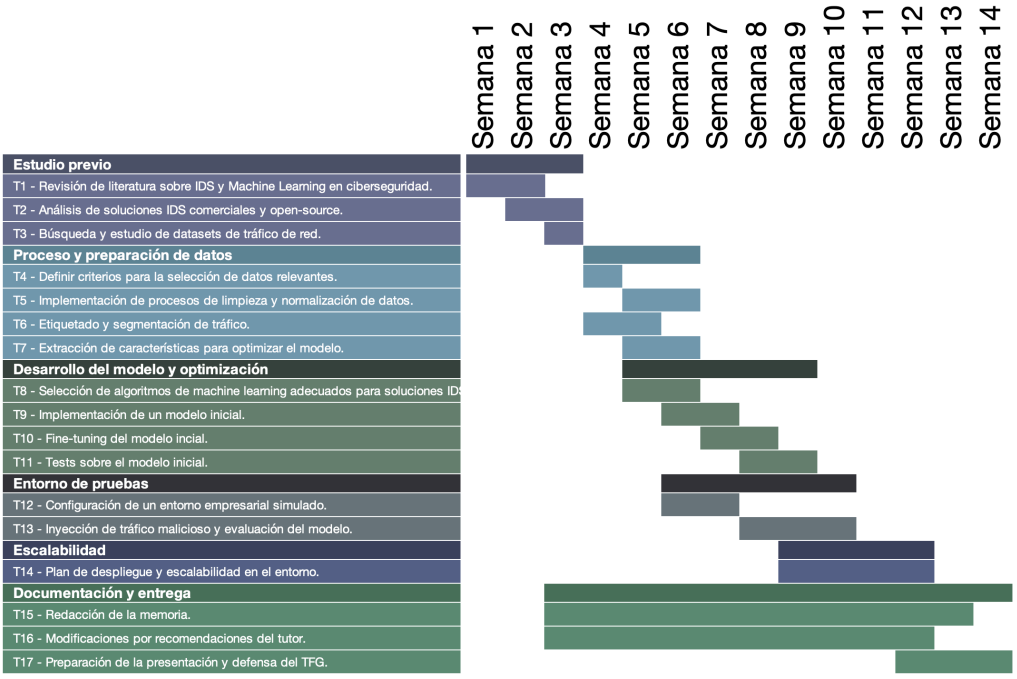

Figura 1.1: Diagrama de Gantt.

1.3. Estructura del documento

El resto de la memoria se ve dividida en los siguientes capítulos.

- **Capítulo 2. Trabajos previos**: trabajos previos relacionados con el tema del trabajo enfocándose en técnicas de machine learning e IDSs.
- **Capítulo 3. Fundamentos técnicos**: descripción de las principales herramientas utilizadas así como del algoritmo escogido junto con algunos conceptos básicos sobre el machine learning.
- **Capítulo 4. Metodología y diseño del modelo**: propuesta, metodología y el diseño necesarios para el desarrollo del modelo.
- **Capítulo 5. Preparación del modelo, entorno de priebas y resultados**: pasos seguidos para desarollar el modelo, descripción del entorno de pruebas y resultados del modelo en las pruebas.
- **Capítulo 6. Propuesta de despliegue y escalabilidad**: integración del modelo final en un formato escalable y propuestas de despliegue del mismo.
- **Capítulo 7. Conclusiones y trabajo futuro**: análisis de los resultados finales del trabajo y futuros pasos a seguir.
- **Capítulo 8. Análisi de impacto**: posible impacto del trabajo en con diferentes enfoques.

2. Trabajos previos

En los últimos años se han publicado numerosos estudios sobre la aplicación de técnicas de Machine Learning (ML) y Deep Learning (DL) en los sistemas de detección de intrusos en redes (IDS), tanto en contextos empresariales como industriales. Uno de los trabajos más completos en este sentido es el de Ahmad [6], que ofrece una revisión sistemática muy detallada sobre los distintos métodos empleados en este campo. El artículo compara algoritmos clásicos como Support Vector Machines (SVM), Random Forest (RF) y XGBoost, así como métodos no supervisados como K-Means e Isolation Forest. También analiza el uso de redes neuronales como las CNN, RNN y LSTM. Una aportación destacada del trabajo es la tabla comparativa entre datasets como NSL-KDD, KDD99, UNSW-NB15 y CICIDS2017, donde se resaltan fortalezas y limitaciones, como la obsolescencia de algunos conjuntos o la falta de equilibrio entre clases. En algunos de los modelos revisados, por ejemplo, Random Forest aplicado al dataset CICIDS2017, se alcanzan precisiones superiores al 99 %, lo cual ilustra el potencial de estas técnicas, aunque también se señala que un alto rendimiento en entornos controlados no garantiza un buen comportamiento en producción.

Siguiendo en esta línea, el trabajo de Sarobin [7] se centra en la implementación práctica de un IDS basado en ML, orientado tanto a entornos empresariales como a dispositivos IoT. El sistema se divide en cinco fases: recogida de datos, preprocesado, selección de características, entrenamiento y detección. Utilizan múltiples algoritmos, entre ellos SVM, KNN, RF y XGBoost, siendo estos dos últimos los que ofrecen mejor rendimiento. En sus pruebas, Random Forest alcanzó un 98.7 % de precisión y XGBoost un 99.1 %, con tasas de falsos positivos por debajo del 1.2 %. Lo interesante es que el estudio simula escenarios reales, incluyendo tráfico de red de dispositivos IoT, y demuestra que el sistema puede escalar manteniendo un rendimiento alto. Este enfoque resulta especialmente relevante para soluciones que buscan equilibrio entre precisión y eficiencia en tiempo real.

En contextos más industriales, el estudio de Chen [8] describe un caso real de despliegue de Snort, un IDS basado en reglas, en una red SCADA. Aunque Snort no incorpora ML, los autores lo adaptan mediante la creación de reglas específicas para entornos industriales. El trabajo analiza los retos de este tipo de implementación, como la elevada tasa de falsos positivos inicial (por encima del 15 %), que se consiguió reducir al 4.6 % tras afinar las reglas. También se aborda la dificultad para detectar ciertos ataques lentos y persistentes, propios

de los entornos industriales, lo que pone de relieve la necesidad de integrar modelos ML que complementen las reglas tradicionales.

Por último, la revisión de Umer [9] se centra en la aplicabilidad del ML en infraestructuras críticas, como plantas de energía, sistemas de agua o fábricas automatizadas. Aparte de técnicas supervisadas y no supervisadas, también discuten enfoques de aprendizaje por refuerzo y sistemas híbridos. Uno de los aspectos más valiosos del trabajo es la identificación de desafíos como la escasez de datos de calidad y etiquetados, el problema de clase desbalanceada y la importancia de minimizar los falsos positivos, ya que en estos entornos cada alerta incorrecta puede traducirse en costes económicos y operativos importantes. Proponen combinar reglas heurísticas con modelos ML para lograr una detección robusta sin comprometer el rendimiento del sistema.

En resumen, estos trabajos ofrecen una visión bastante completa sobre el estado del arte en el uso de técnicas de ML y DL aplicadas a IDS. Desde enfoques teóricos hasta pruebas reales en entornos empresariales e industriales, permiten entender no solo qué técnicas se utilizan, sino también por qué, cómo y con qué limitaciones. Esta base ha sido clave para orientar el diseño del entorno experimental y del modelo propuesto en este trabajo, con el fin de mejorar la detección de amenazas manteniendo un buen equilibrio entre precisión, escalabilidad y coste computacional.

3. Fundamentos técnicos

En esta sección se va a paortar información sobre las principales herramientas utilizadas durante el desarrollo de este trabajo así como conceptos importantes para comprender correctamente el mismo y una descripción algo más completa sobre el principal algoritmo de aprendizaje automático utilizado.

3.1. Machine Learning

El *Machine Learning*, en castellano aprendizaje automático es una rama de inteligencia artificial la cual nos da la posibilidad de enseñar a los sistemas mediante conjuntos de datos y que estos lleven a cabo predicciones o tomen decisiones sin que hayan sido programados explicitamente para ello [10]. Dentro del machine learnign existen tres subcategorías, el aprendizaje supervisado, el parendizaje no supervisado y el aprendizaje por refuerzo.

- **Aprendizaje supervisado**: el *Aprendizaje Supervisado* trabaja con un conjunto de datos el cual ha sido procesado previamente y contiene etiquetas, estas son utilizadas para entrenar al modelo para que posteriormente pueda hacer predicciones sobre nuevos datos. Algunos los algoritmos más utilizados dentro de esta categoría son los árboles de decisión, redes neuronales y regresión logística. Este es utilizado por ejemplo en el reconocimiento de imagenes.
- **Aprendizaje no supervisado**: el *Aprendizaje No Supervisado* trabaja con conjuntos de datos que carecen de etiquetas y su enfoque no es tanto en hacer prediccione, sino que se enfoca en encontrar patrones y agrupaciones en un conjunto de datos. Dentro de esta categoría encontramos las técnicas de clústering y de reducción de dimensionalidad. Este tipo de aprendizaje es utilizado para segmentar clientes en marketing.
- **Aprendizaje por refuerzo**: el *Aprendizaje por Refuerzo* utiliza un sistema de recompensas y penalizaciones sobre el modelo guiando a este hacia una solución óptima mediante la prueba y error. Un ejemplo de este tipo de aprendizaje es AlphaZero, un modelo de inteligencia artificial capaz de juguar al ajedrez.

3.2. Sistema de Detección de Intrusiones (IDS)

Un *Sistema de Detección de Intrusiones* (*IDS* sus siglas en inglés) es una herramienta cuya función es analizar y monitorizar el tráfico de una red o un sistema su objetivo es detectar en tiempo real comportamientos anómalos o actividades sospechosas que pudieran ser indicativos de un ataque o intrusión en el sistema o la red. Al detectar e identificar una posible amenaza, reporta esta para que se puedan tomar las medidas pertinentes buscando reducir el posible impacto ante una intrusión. Estos pueden clasificarse en dos tipos:

- **IDS basados en anomalías**: Estos analizan el tráfico de la red o el comportamiento del sisitema y lo clasifican como normal o anómalo. Para esto necesitan un modelo preentrenado que les permita identificar patrones de comportamiento anómalos. Este tipo de IDS puede detectar ataques desconocidos, pero también puede generar falsos positivos si el modelo no está bien ajustado.

- **IDS basados en firmas**: estos se basan en firmas de ataques conocidos para detectar intrusiones. Utilizan una base de datos de firmas de ataques y comparan el tráfico de la red o el comportamiento del sistema con estas firmas. Este tipo de IDS es más preciso que el basado en anomalías, pero no puede detectar ataques desconocidos.

3.3. Python y librerías utilizadas

Python [11] es un lenguaje de programación de alto nivel y muy versátil, abarcando desde el desarrollo web hasta la ciencia de datos y la inteligencia artificial. Posee la capacidad de ser utilizado como lenguaje de scripting, lo que junto con *jupyter notebooks* [12] lo convierte en una herramienta ideal para el desarrollo de modelos de machine learning. Además, cuenta con una amplia variedad de librerías muy potentes que hacen sencillo el desarrollo de nuevos modelos así como la carga de modelos preentrenados.

3.3.1. NumPy

NumPy [13] es una librería numérica muy potente utilizada para todo el ecosistema científico de Python. Una de sus principales caracterísitcas más importantes es el uso del objeto ndarray; este es un contenedor multidimensional de elemntos del mismo tipo y tamaño y sobre los cuales se puede operar como vectores. Las operaciones sobre estos datos no se llevan a cabo en Python, sino que hay rutinas en C y Fortran optimizadas lo cual mejora el rendimiento de la librería significactivamente. Además de esto, dispone de una amplia gama de funciones de álgebra lineal y generación de números aleatorios entre otros. NumPy sirve de sustento para otras librerías como Pandas y scikit-learn entre otras por lo que es compatible con las mismas.

3.3.2. Scikit-learn

Scikit-learn [14] se trata de una de las librerías más importantes en lo que al machine learning se refiere en Python. Esta contiene una gran variedad de algoritmos y utilidades. La librería Scikit-learn está construida sobre NumPy y SciPy y aprovecha las optimizaciones de NumPy previamente mencionadas. Esta relega las operaciones matemáticas relacionadas con los diferentes modelos a las librerías sobre las que se compone por lo que su operación se basa en la elección de un modelo, su ajuste y la evaluación de este. Incluye algoritmos de regresión lineal, k-means y Random Forest entre otros. Con pocas líneas procedentes de un Dataframe de Pandas, se puede tener un modelo operativo. Como se muestra en el ejemplo, con 6 líneas de código se ha ganerado, entrenado y probado un modelo.

```
X, y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y,
 test_size=0.3, random_state=42)

clf = KNeighborsClassifier(n_neighbors=3)
clf.fit(X_train, y_train)
preds = clf.predict(X_test)

print(f"Accuracy: {accuracy_score(y_test, preds):.2%}")
```

3.3.3. Pandas

Pandas [15] se trata de una librería en Python que utiliza los objetos ndarray de NumPy en formato tabular, cada una de las fialas y columnas son etiquetadas ya sea con un índice numérico o con etiquetas propias en el caso de las columnas. Se trata de una librería muy versátil que tiene capacidad de migrar datos de o a formatos como SQL o Excel; además de permitir llevar a cabo transformaciones, y agrupaciones entre otras cosas sin la necesidad explícita de la declaración de bucles. Destaca por su integración con otras librerías como Matplotlib o Seaborn así como con ScikitLearn. Algunas de las características más destacadas es que se puede cargar una tabla a partir de un fichero CSV (siempre y cuando este esté bien estructurado) en una sola linea de codigo, la simplicidad de las operaciones que se pueden llevar a cabo para limpiar un conjunto de datos o la capacidad de obtener métricas de un dataset ya cargado con facilidad.

3.3.4. Matplotlib

Matplotlib [16] es una librería preparada para la visualización de datos en Python y en la cual se apoyan muchas otras librerías de este mismo lenguaje. Matplotlib permite además de representar cómodamente los datos sin demasiados ajustes previos, tiene también la capacidad de especificar multitud de variables para ajustar el grosor de líneas, el color de las representaciones, las escalas y otra multitud de elementos relevantes a la hora de generar gráficos a partir de datos en Python. Una característica importante de esta librería es su integración con otras ya mencionadas previamente como NumPy y Pandas, así como con Jupyter Notebooks. Junto con la generación de los gráficos, tienes la capacidad de exportar estos en diferentes formatos como SVG o PNG.

```
plt.plot([0, 1, 2, 3], [0, 1, 4, 9])
plt.title('y = x^2 en cuatro puntos')
plt.show()
```

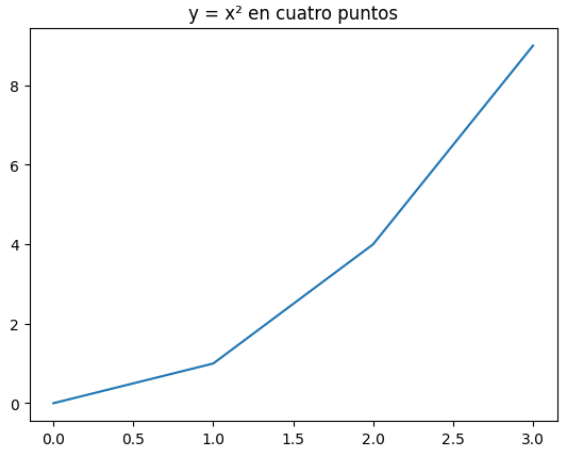

Figura 3.1: Gráfico de ejemplo de matplotlib.

3.3.5. TensorFlow

TensorFlow [17] es una librería de código abierto desarrollada por Google para el desarrollo de modelos de aprendizaje automático y redes neuronales. Proporciona una amplia gama de herramientas y recursos para la creación, entrenamiento y despliegue de modelos de aprendizaje profundo.

3.3.6. Seaborn

Seaborn [18] es una librería de visualización de datos basada en Matplotlib, que proporciona una interfaz simplificada para crear gráficos estadísticos atractivos y informativos. Seaborn facilita la creación de gráficos complejos y mejora la estética de las visualizaciones, permitiendo una mejor comprensión de los datos.

3.4. Árboles de decisión

Un *Árbol de decisión* [19] es un algoritmo de aprendizaje supervisado cuya estructura es jerárquica y con forma de árbol. Estos pueden ser binarios o multi-rama dependiendo del número de hijos que puede tener cada nodo interno. Los elementos de un árbol de decisión son los siguientes:

- *Nodo raíz*: es el nodo del cual parte el árbol de decisión y en el cual se genera la primea división.
- *Nodos internos o de decisión*: los nodos internos son los puntos del árbol en los que se aplica una regla por lo que son puntos de bifurcación del árbol.
- *Ramas*: las ramas son las aristas que conectan cada nodo padre con sus nodos hijo.
- *Nodos hoja*: son nodos terminales que contienen la predicción final. El total de nodos hoja son el total de resultados posibles del árbol de decisión.

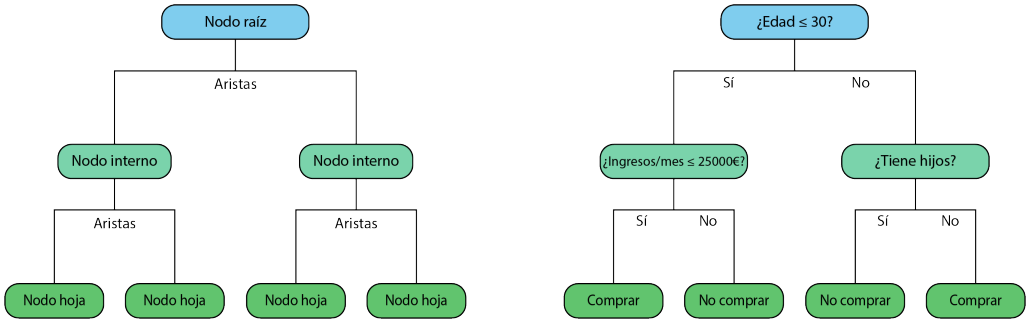

Figura 3.2: Ejemplo simple de árbol de decisión.

La construcción de un árbol busca la división óptima del conjunto y tratando de mejorar la función objetivo. Esta simplicidad permite el uso de variables tanto categóricas como numéricas discretas y contínuas. La técnica de generación se basa en la estratégia de divide y vencerás, en el que se va clasificando el árbol en subconjuntos homogéneos. Los árboles de decisión de grandes dimensiones tienden al sobreajuste por lo que se aplican técnicas de poda, validación cruizada y otro tipo de estratégicas para reducir esto.

Los principales algoritmos de árboles de decisión entorno a los cuales se estructuran el resto, son:

- **ID3 (Iterative Dichotomiser 3)**: se trata de un algoritmo basado en la entropía [1] Ecuación(3.1) y en la ganancia de información Ecuación(3.2) para evaluar las particiones en los nodos internos. Los árboles generados mediante este algoritmo tienden a ser algo profundos y bastante ajustados al conjunto de entrenamiento. Este fue propuesto por Ross Quinlan en la década de los 80.

$$H(S) = -\sum_{i=1}^{k} p_i \log p_i \tag{3.1}$$

$$IG(S, A) = H(S) \; - \sum_{v \in Valores(A)} \frac{|S_v|}{|S|} H(S_v) \tag{3.2}$$

- **C4.5**: es un algoritmo sucesor de ID3 y desarrollado también por el creador de este. Este utiliza la ganancia de información, pero normaliza esta dividiendo la ganancia de información por la información de la partición Ecuaciones(3.3 y 3.4). Mediante esto se reduce el peso de los atributos con muchos valores posibles. El resultado de este algoritmo es un árbol menos profundo y con menor tendencia al sobre ajuste.

$$SI(S, A) = - \sum_{v \in Valores(A)} \frac{|S_v|}{|S|} \log \frac{|S_v|}{|S|} \tag{3.3}$$

$$GR(S, A) = \frac{IG(S, A)}{SI(S, A)} \tag{3.4}$$

- **CART (Clasication and regression tree)**: este algoritmo utiliza la impureza de Gini [2] Ecuación (3.5) para decidir el atributo en el que hacer la partición en el caso de que la variable objetivo sea categórica. En el caso de la regresión lineal que se da cuando la variable objetivo es numércia contínua, el algoritmo trata de reducir la diferencia de errores Ecuación (3.6). Este algoritmo está contenido en librerías como ScikitLearn.

$$Gini(S) = \sum_{i=1}^{k} p_i \, (1 - p_i) = 1 - \sum_{i=1}^{k} p_i^2 \tag{3.5}$$

$$min\{\frac{1}{k} \sum_{i=1}^{k} (y_i - \hat{y}_i)^2\} \tag{3.6}$$

[1] La entropía de Shannon de una distribución es la cantidad de información que aporta la aparición de un evento en esa distribución [20].

[2] La impureza de Gini es la medida de la probabilidad promedio de que un ejemplo aleatorio sea clasificado de manera errónea.

3.5. XGBoost

XGBoost [21] (Extreme Gradient Boosting) es una implementación de árboles de decisión potenciados por gradiente que mejora la escalabilidad, la robustez y la velocidad del mismo. Este parte del gradient boosting [3] al que añade un nivel más de profundidad mediante el cálculo de las curvaturas (hessianas) de la función de pérdida, lo que permite saber la dirección en la que hacer la correción así como de cuánto tiene que ser ese ajuste. El resultado de esto es un conjunto de pequeños árboles CART ponderados a los cuales se les pasan los datos del elemento a predecir y que mediante una suma obtienen el resultado. XGBoost tiene otro tipo de optimizaciones que manejan de manera automática valores faltantes, así como la dispersión de las matrices. También tiene mejoras para su ejecución en diferentes tipos de dispositivos. Finalmente este incluye la poda temprana junto con shrinkage [4] y el column subsampling [5]. Esta combinación genera modelos robustos con buen rendimiento y buena precisión y por lo tanto es ampliamnete utilizado en labores de aprendizaje supervisado.

3.6. CICFlowMeter

CICFlowMeter [23] es una herramienta de libre uso desarrollada en Java por el Canadian Institute for Cybersecurity (CIC) la cual transforma capturas de tráfico de red (pcap) en flujos bidireccionales de los cuales calcula numerosas métricas de este tráfico. Esta herramienta en modo de escucha activa ha sido utilizada para generar los datasets utilizados en este trabajo.

3.7. Datasets utilizados

Los dos datasets utilizados en este trabajo de fin de grado han sido generados por el Canadian Institute for Cybersecurity (CIC) en diferentes laboratorioes en años distintos. Los dos contienen todos los datos etiquetados con diferentes tipos de ataque así como tráfico benigno. Los atributos de ambos datasets son en general los mismos ya que fueron generados con la misma herramienta, el CICFlowMeter y contien lo siguientes campos agrupados:

- Puerto de destino
- Proporción Entrada/Salida
- Métricas de tiempo de un flujo
- Métricas de envío y recepción de paquetes
- Conteo de flags URG, CWE, ECE, ACK, PUSH, RST, SYN, FIN
- Métricas de dimensiones de paquetes

[3] El gradient boosting se basa en la construcción de secuencia de árboles CART de pequeñas dimensiones que corrigen los errores del modelo anterior, cada iteración es una ronda.

[4] Shrinkage es una técnica que reduce el peso de los árboles ya presentes para dejar espacio para futuros árboles de cara a mejorar el modelo a futuro.[22]

[5] El column subsampling consiste en el uso de sólo una fracción de los atributos en cada uno de los árboles.

Los ataques llevados a cabo en ambas simulaciones son los siguientes:

- Ataque por fuerza bruta: se trata de un tipo de ataque en el que mediante una iteración sobre un conjunto de usuarios y contraseñas se trata de acceder a un sistema, en el caso de estas simulaciones se trató de acceder mediante los servicios SSH y FTP.

- Ataque DoS: este ataque trata de provocar una denegación de servicio en un sistema sobrecargando el proveedor con una gran cantidad de peticiones.

- Ataques web: los ataques web consistieron en pruebas sobre plataformas de testeo con multiples vulnerabilidades SQLi [6] y XSS [7].

- Ataque de infiltración: esta parte del ataque consiste en el estudio de la red, el primer paso consiste en vulnerar la seguridad de una máquina conectada a la red, en este caso mediante un enlace malicioso, tras esto el malware descargado ejecuta un escaneo de la red mediante la herramienta Nmap[8] y envía los datos de la exploración a una dirección externa.

- Ataque de botnet: este ataque consistión en infectar varios PCs de la red simulada. Estos PCs infectados abren un canal HTTP sin cifrar con el servidor de command and control al que notifican su estado de manera recurrente, mediante esta conexión se puede abrir una consola remota y ejecutar comandos de manera instantánea, así como cargar archivos y subir capturas de pantalla. Toda la información capturada se envía al servidor command and control.

- Ataque DDoS y escaneo de Puertos: este ataque consiste en uno parecido a el ataque DoS pero en este caso la fuente del ataque no es un sólo equipo sino que se trata de un ataque con múltiples atacantes simultáneos. Junto con este se lleva a cabo un escaneo de puertos abiertos en las máquinas de la red.

3.7.1. CICIDS 2017

CIC-IDS2017 [24] este dataset proviene de una simulación de ataque en los laboratorios del CIC entre el 3 de julio de 2017 y el 7 de julio de 2017. La red estuvo compuesta por equipos con multiples sistemas operativos (MacOSX, Ubuntu, Windows) así como diversos equipos de red como módems, firewalls, switches y routers. Los dispositivos atacantes ejecutaban los sistemas operativos Kali Linux y Windows.

[6] SQLi es la abreviatura de SQL injection que consiste en vulnerar la seguridad de una aplicación web introduciendo consultas aparentemente no permitidas a la base de datos asociada a esta.

[7] XSS significa cross-site scripting y consiste en la ejecución de código malicioso en páginas web.

[8] Nmap es una herramienta que permite escanear los dispositivos presentes en una red así como puertos o servicios que tiene abiertos a la misma.

3.7.2. CICIDS 2018

CIC-IDS2018 [25] se trata del dataset fruto de una simulación de ataque en los laboratorios del CIC entre el 14 de febrero de 2018 y el 2 de marzo de 2018. En esta intervinieron más de 500 máquinas como susceptibles víctimas, estas corrían diferentes versiones de Windows, así como Windows Server y equipos con Ubuntu. Por otro lado quedan los equipos atacantes cuyo sistema operativo era Kali Linux. La red utilizada en la simulación estña representada en la imagen(3.3).

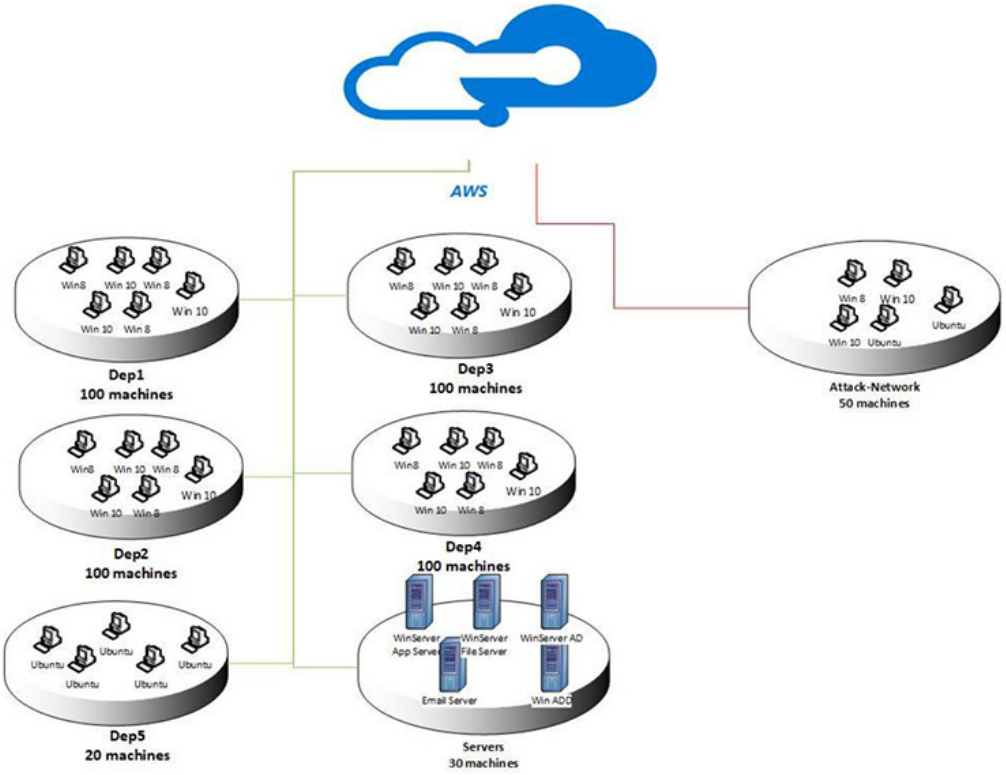

Figura 3.3: Red de CIC-IDS2018.
[26]

3.8. pfSense

pfSense [27] es una distribución de libre uso basada en el sistema operativo FreeBSD que permite convertir un equipo en un firewall/router. Las capacidades del equipo con este software son muy amplias, permite actuar como NAT, funciones de balanceo de carga, desplegar VPNs. En la edición pfSense Comunity Edition en la versión 2.8.0 existe la posibilidad de junto con las características embebidas en el sistema, instalar paquetes para añadir funcionalidades que extiendan la capacidad del sistema. La gestión de este se puede hacer mediante consola así como desde una interfaz web.

3.9. Proxmox

Proxmox [28] es un sistema operativo de virtualización de código abierto basado en Unix que agrupa en un mismo nodo la posibilidad de desplegar máquinas virtuales así como contenedores ligeros junto con almacenamiento dirigido por software y redes dirigidas por software. La administración del sistema se puede llevar a cabo desde el própio terminal de la máquina que corre el sistema cómo a través de la interfaz web.

3.10. Docker

Docker [29] es una plataforma open-source que permite empaquetar una aplicación junto con todas sus dependencias en una imagen que al ejecutarse con DOcker Engine pasa a ser un contenero aislado. A diferencia de las máquinas virtuales, el contenedor comparte el kernel del sistema con el de la máquina anfitriona.

3.11. Suricata

Suricata [30] es un motor de análisis y detección de amenazas de uso libre desarrollado por la Open Information Security Foundation. Este puede ser ejecutado como IDS, IPS o como sonda de monitorización de seguridad de red. Permite establecer multitud de reglas que pueden combinarse con detección medinate firmas o inspección profunda.

4. Metodología y diseño del modelo

El trabajo se enfoca en generar un modelo de machine learning optimizado para la detección de intrusiones en redes empresariales. El modelo no se enfoca tanto en detectar el tipo de ataque al que se ve expuesta la red sino que en detectar si el tráfico es anómalo o no.

El fin no es implementar un sistema de detección de intrusos, tan sólo un modelo base preentrenado con un importante conjunto de datos de tráfico de red y una API para su uso. El fin es que el modelo pueda ser integrado con diferentes sistemas a partir de esta API siempre y cuando se le transfieran los datos con la estructura adecuada.

Para el tratamiento de las capturas de tráfico de red, lo cual es necesario para la API que maneja el modelo, se ha utilizado una herramienta proporcionada por el Canadian Instituto for Cybersecurity (CIC) llamada *CICFlowMeter*.

4.1. Metodología

La metodología seguida para el desarrollo del modelo se ha dividido en varias fases. Debido a la naturaleza del trabajo no fueron llevadas a cabo estrictamente de manera secuencial, sino que se fueron iterando a medida que procesó el trabajo y en función de las necesidades y dificultades que se fueron presentando. Este enfoque es coherente con la metodología CRISP-DM (*Cross Industry Standard Process for Data Mining*), comunmente utilizada en proyectos de análisis de datos, que permite volver atrás o saltar entre fases según lo requiera el proceso [31].

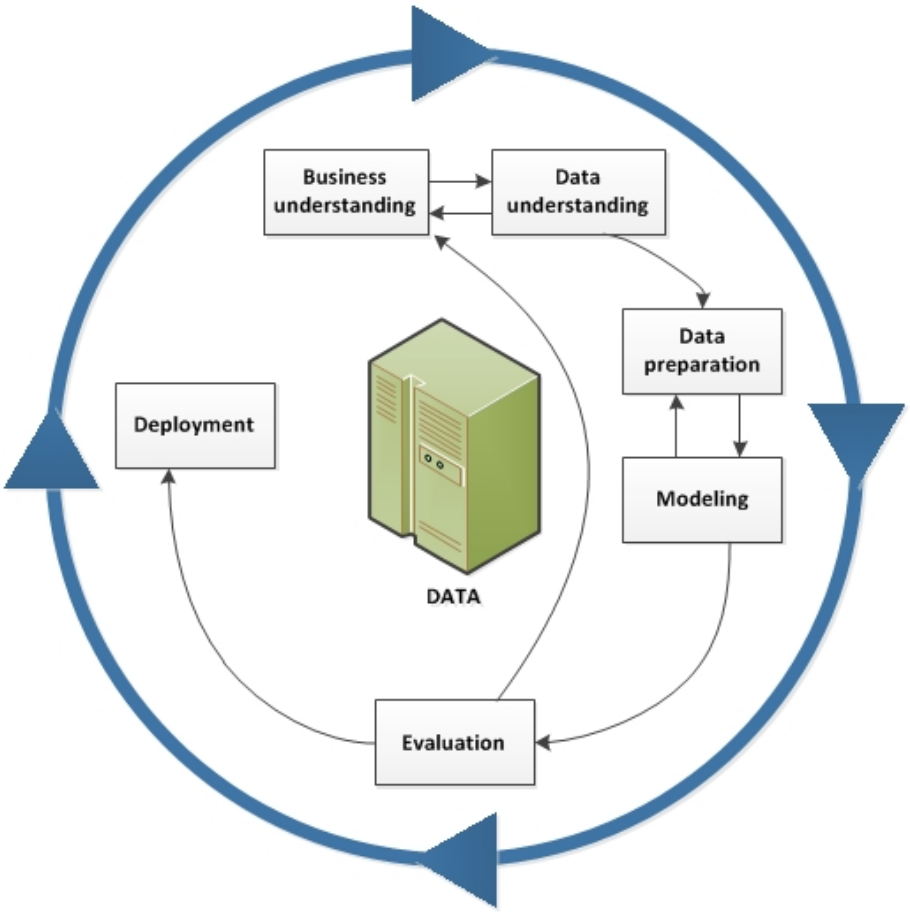

Figura 4.1: Diagrama CRISP-DM.
[32]

Se comenzó buscando un conjunto de datos con el cual se pudiese entrenar un modelo viable. Se encontraron diversos conjuntos de datos bastante relevantes, otros candidatos fueron rechazados por haber sido generados hace más de una década mientras que otros se consideró que no eran lo suficientemente genéricos como para desarrollar un modelo con el enfoque de este trabajo. Tras el análisis y pruebas sobre este conjunto se decidió buscar un segundo conjunto sobre el que se llevó a cabo el mismo análisis y pruebas. Con ambos conjuntos analizados y habiendo implementado diversos modelos sobre estos, se decidió unificar ambos conjuntos generando un único dataset con el total de los registros. Tras esto se generó un nuevo modelo basando su desarrollo en los resultados de las pruebas con los modelos anteriores. Ya con un modelo final con buenas métricas de rendimiento se procedió con la optimización y prueba del modelo recurriendo a datos de tráfico de red generados en un entorno simulado. Finalmente tras la validación del modelo con estas pruebas se presentaron los datos de rendimiento y se generó una API con la que obtener la estimación del modelo sobre nuevos datos de tráfico de red.

4.2. Diseño

Al ser un proceso de un estudio inicial sobre un conjunto de datos, tras esto su limpieza, preprocesado, partición entrenamiento y pruebas del modelo resultante, no se llevó a cabo con ficheros de código en Python; sino que se llevó a cabo mediante notebooks en jupyter para poder llevar a cabo el trabajo por pasos y si era necesario retroceder o introducir algún paso extra el uso de este tipo de simplifica estas labores. Otra de las ventajas del uso de notebooks es que puedes ver el progreso la limpieza de un conjunto de datos ejecutando tan solo una caja del mismo. Los ficheros presentes son los siguientes:

- Limpieza_y_modelos_v1_eda2017.ipynb
- Limpieza_y_modelos_v1_eda2018.ipynb
- evaluador_2017.py
- XGBoost_general.ipynb
- 2017_ids/
- 2018_ids/
- test.ipynb
- API_modelo.zip

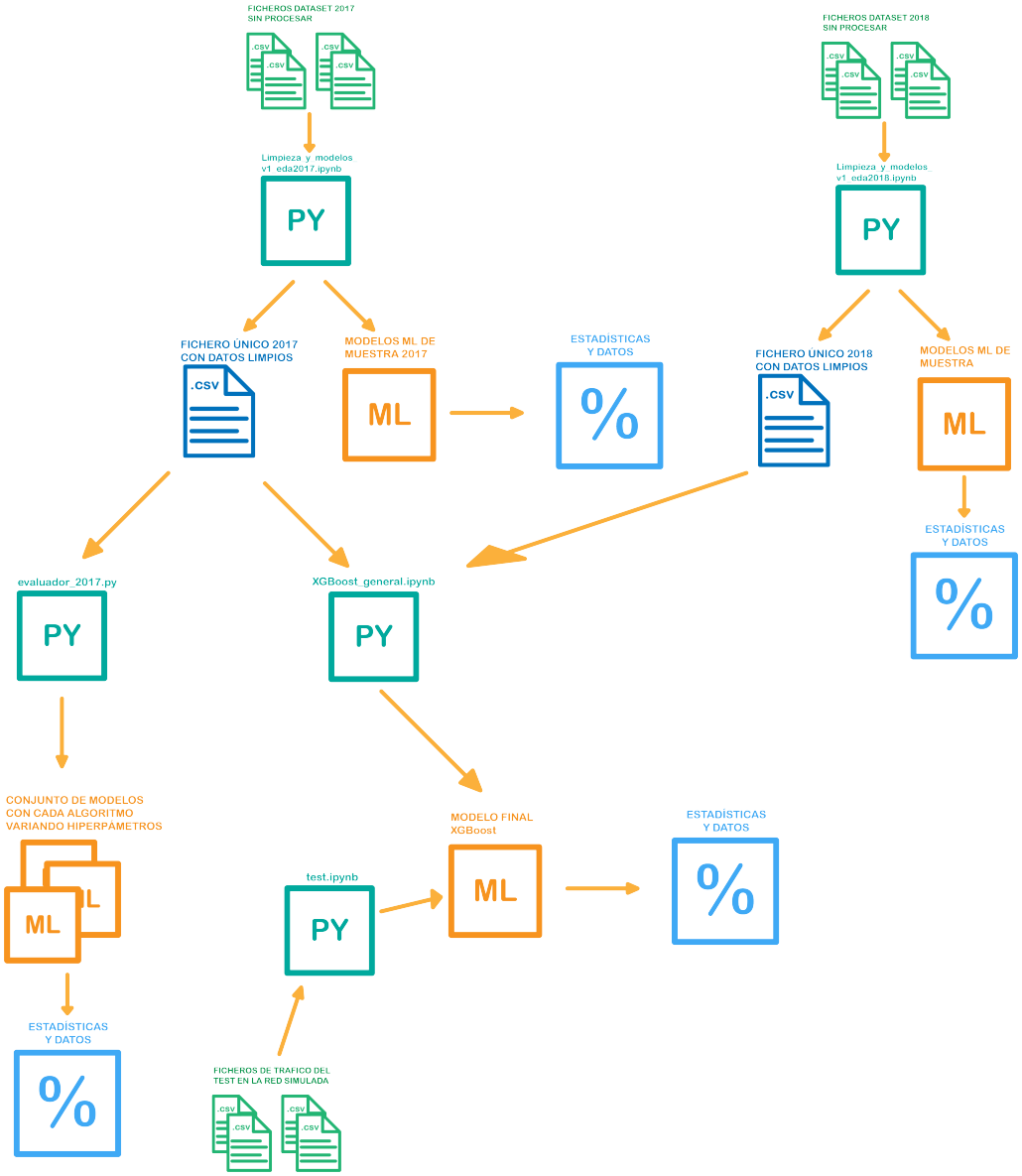

Figura 4.2: Flujo de trabajo en los datos modelos resultantes.

5. Preparación del modelo, entorno de pruebas y resultados

Las partes clave de este trabajo son el preproceso de datos, el entrenamiento del modelo y su validación.

El modelo de machine learning desarrollado para la detección de intrusiones se basa en un enfoque de aprendizaje supervisado, utilizando conjuntos de datos de tráfico de red para entrenar y evaluar su rendimiento. Las fases de trabajo en este se presentan a continuación.

5.1. Preparación del modelo

5.1.1. Estudio de los datos

El trabajo comenzó con el estudio de posibles fuentes de datos con lo que se puediese entrenar el modelo objetivo. Se procedió con una búsqueda de candidatos en repositorios públicos de datos y se encontró el *CICIDS 2017*, no obstante, no se consideró que este conjunto de datos fuese suficiente para el objetivo del trabajo ya que este estaría entrenado con la situación de la red del laboratorio con el que se generaron los datos, por ello se decidió buscar un segundo conjunto de datos con las mismas variables para buscar una mayor generalización del modelo. Se encontró el *CICIDS 2018* del mismo instituto. El conjunto de datos ascendió a un total de 10,8 millones de registros de red con una divesidad de tipos de tráfico y ataques, y lo que es más importante aún, con la casuística de dos laboratorios diferentes, lo que permite una mayor generalización del modelo.

Se escogieron estos dos conjuntos de datos por diversas razones. Principalmente la gran variedad de tráfico tanto legítimo como malicioso generado por dos laboratorios con una diferencia de aproximadamente un año entre sí. Esto junto con la variedad de ataques presentes en ambos conjuntos ha sido un factor determinante a la hora de seleccionarlos. Otro de los factores a destacar, es la cantidad de informacón que ofrecen del tráfico de red; recordemos que este a menos que sea procesado por una herramienta como *CICFlowMeter*, este es capturado a nivel de paquete y de este se pueden obtener datos como el emisor, el receptor, tamaño y campos como ciertas flags, pero si tan sólo se miran los paquetes de uno en uno no se puede saber cuál de los nodos de una conversación inició la misma, así como tampoco podemos saber el ancho

de banda de esta y muchos otros campos que si que están presentes en estos datasets sin tener que trabajar sobre los pcaps para obtenerlo. Por otro lado es importante resaltar el hecho de que los datos están etiquetados de manera exhaustiva habiendo sido cada uno de los flujos etiquetado por el equipo que generó el conjunto. Un factor importante es su relevancia a nivel científico formando parte de amplias investigaciones en el campo. Finalmente el volúmen de datos fruto de la combinación de ambos datasets ha sido clave para escogerlo.

5.1.2. Preproceso de los datos

En cada uno de los dos conjuntos se llevó a cabo la eliminación de las filas con valores nulos o no válidos. Simultáneamente se hizo una selección de los puertos con mayor numero de registros, tras esto aquellos que no se encontraban en la lista de puertos seleccionados fueron recalificados como *Other*. Tras esto se reemplazo la columna *Destination Port* por 21 columnas binarias cuyos nombres eran `Port Category_[port number]` donde `port number` era el número del puerto o en su defecto *other*, esta labor se llevó a cabo con *OneHotEncoder* [33]. Tras esto se normalizaron los datos de las columnas numéricas evitando errores de ajuste en el modelo por las diferentes escalas de las variables. Para esto se usó *StandardScaler* [33] al igual que *OneHotEncoder* de la librería *Sklearn*. Finalmente se convirtió la variable *Label* a un formato numérico, en este caso con dos posibles valores, *Normal* y *Attack*, donde *Normal* obtuvo el valor 0 y *Attack* a 1. Esta conversión se realizó porque el objetivo del modelo no es distinguir la naturaleza de un ataque, sino detectar de si se trata de algún intento de ataque o no.

5.1.3. Evaluación de algoritmos

Tras un estudio inicial de los datos, el siguiente paso fue el estudio de los posibles algoritmos de machine learning de aprendizaje supervisado que se podrían aplicar sobre los datos de cada dataset. Esta evaluación se hizo de manera independiente llevando a cabo iteraciones con el algoritmo de regresión logística *LogisticRegression* [33] de la librería de Python *Sklearn* [14], el algoritmo *RandomForestClassifier* [33] de la misma librería, el algoritmo *XGBoost* [34] y el algoritmo *LightGBM* [35]. Estos algoritmos fueron seleccionados por su popularidad.

Para obtener los conjuntos de entrenamiento y validación se utilizó la validación por separación simple. Este método se basa en dividir un conjunto original en dos subconjuntos siendo estos resultantes los subconjuntos de entrenamiento y test. Las proporciones de estos son modificables. Al hcerse una sola partición del conjunto el coste de ejecución es menor que al usar otros métodos. Al ser un conjunto de grandes dimensiones y con proporciones similares de flujos benignos y maliciosos, tras la reordenación aleatoria de las filas mediante la semilla aleatoria y no diferenciar entre tipos de ataque, las dos etiquetas quedarán representadas en ambos conjuntos de prueba y validación. Para estos modelos iniciales se utilizó la proporción 80% de entrenamiento y 20% de validación. El rendimiento de cada uno de estos algoritmos fue evaluado con las siguientes métricas:

- **Precisión:** la proporción de verdaderos positivos sobre el total de positivos predichos.

$$\text{Precision} = \frac{TP}{TP + FP} * 100$$

- **Recall:** la proporción de verdaderos positivos sobre el total de positivos reales.

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

- **F1-score:** la media armónica entre la precisión y la exhaustividad.

$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

Los resultados de las pruebas fueron los siguientes:

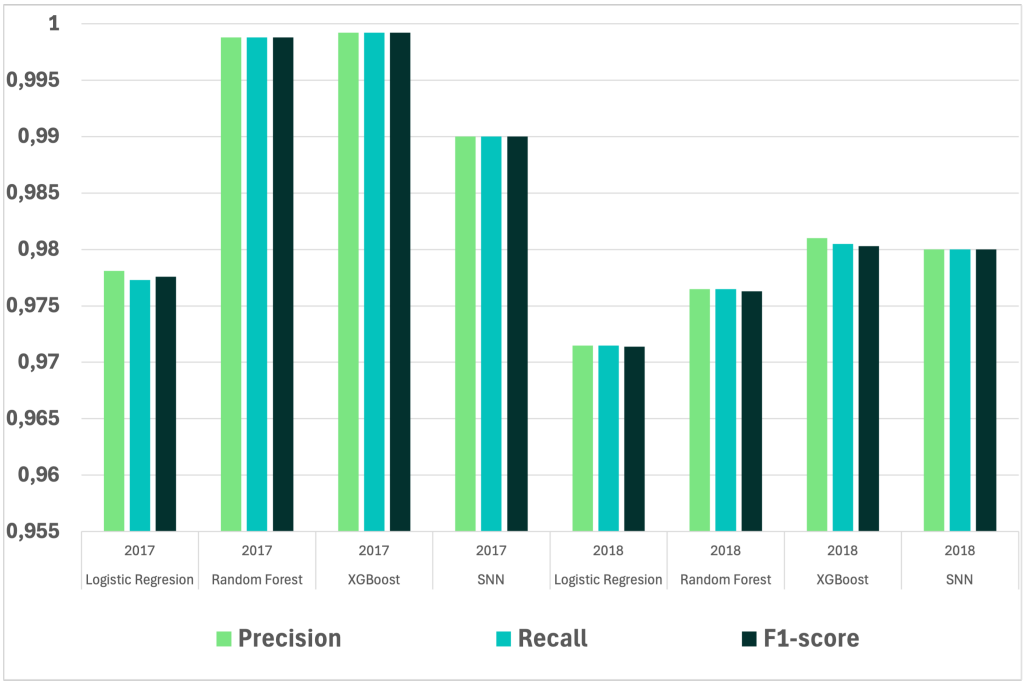

Figura 5.1: Resultados de los modelos de machine learning.

A partir de estos se se vió la superioridad de dos algoritmos sobre el resto, el *Random Forest Classifier* y el *XGBoost*. También se muestra que la calidad de los datos del conjunto CICIDS 2017 es ligeramente superior que el otro conjunto de datos.

5.1.4. Pruebas de algoritmos

Se desarrolló en Python un código que generaba 4 modelos, cada uno basado en uno de los algoritmos previamente mencionados en el trabajo, todos estos utilizaban la mmisma semilla aleatoria para la separación de los conjuntos, también se hacían iteraciones con las proporciones de datos de entrenamiento y validación. Se mantuvo ejecutando este código durante más de 1 día para comprobar si existía algún tipo de relación directa entre las proporciones de los subconjuntos. Este proceso también midió el tiempo de entrenamiento de cada uno de los modelos.

Los resultados finales de estas pruebas fueron determinantes para escoger el algoritmo *XGBoost* para el modelo final con todos los datos. Otra aportación importante de estas pruebas fue la ausencia de relación entre las dimensiones de los conjuntos y la precisión del modelo, ya que se observaba que las diferencias entre los modelos con subconjuntos de entrenamiento de mayores dimensiones ofrecían prestaciones similares a aquellos con dos subconjuntos de igual proporción.

5.1.5. Subconjuntos de variables predictorias

Antes de trabajar con los dos datasets juntos, se propuso comporbar posibles mejoras de optimización. Se extrajeron los pesos de cada una de las variables predictorias en el modelo *XGBoost* de ambos datasets para decidir la clase objetivo y se cotejaron ambos conjuntos. Sin embargo, en este punto no se encontró un consenso claro para hacer una selección de variables para ambos por lo que se procedió con los dos datasets completos al siguiente paso.

5.1.6. Preparación de los datos

En este punto se tenían dos datasets con un importante número de registros y variables los cuales tenían que ser unidos correctamente para que en un futuro sirviesen para entrenar el modelo. Todo este trabajo de preparación se llevó a cabo con las herramientas *Python* y *Jupyter Notebooks* junto con la librería de Python *Pandas*. Para la conversión de se hizo de manera manual una comparación de los nombres de las variables y se hizo un mapeado de conversión desde el conjunto de datos del CICIDS 2018 a la estructura de CICIDS 2017. Una vez se tuvo el mapeado, se procedió a la unión de los datasets.

5.1.7. Entrenamiento del modelo

Ya con los dos conjuntos de datos unidos y sin valores faltantes. Se procedió con el entrenamiento de un modelo final con cada uno de los algoritmos. Es cierto que en apartados anteriores del trabajo se había mencionado que *XGBoost* había presentado un mejor rendimiento y por lo tanto se había escogido como algoritmo para el modelo final. No obstante, al haber mezclado ambos conjuntos de datos, podría darse el caso de que otro modelo diese mejores prestaciones por lo que se procedió de esta manera. Se realizaron pruebas iterativas variando en dimensiones de los datos de entrenamiento y diferentes valores de estado aleatorio por las mismas razones.

Los resultados de estas pruebas fueron los siguientes:

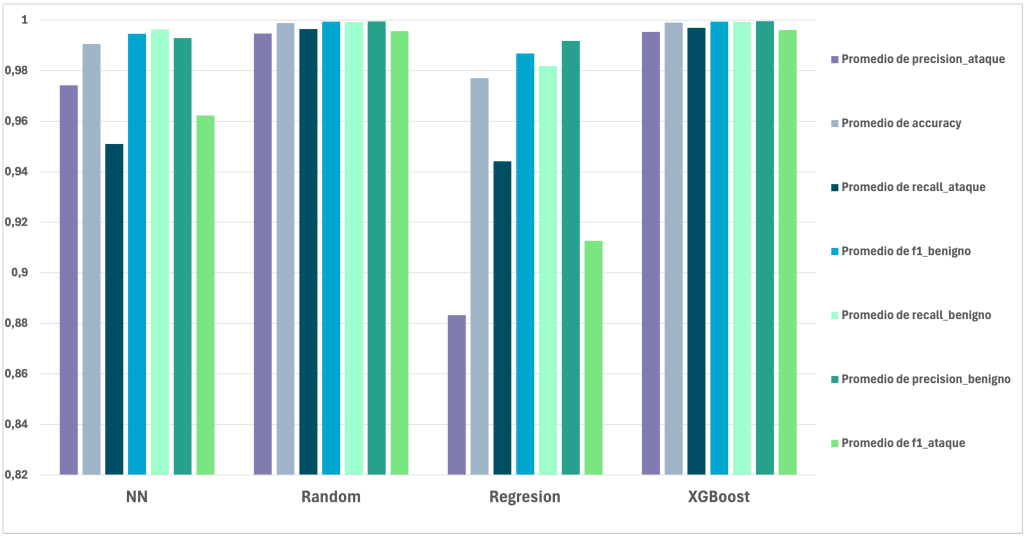

Figura 5.2: Resultados de los modelos sobre el dataset final.

Los resultados muestran que los modelos *XGBoost* y *Random Forest* se mantienen como los modelos con mejor rendimiento. Si a esto le sumamos que los tiempos de entrenamiento del modelo *XGBoost* son considerablemente inferiores a los del modelo *Random Forest*, se decidió continuar con el modelo *XGBoost* para el modelo final.

5.1.8. Metricas del modelo final

El modelo final se entrenó con un 50 % de los datos del dataset combinado y se evaluó con el otro 50 % con el mismo método de validación que el que se ha descrito anteriormente, el método de validación por separación simple. Los resultados del modelo final fueron los siguientes:

Cuadro 5.1: Resultados del modelo final.

	precision	recall	f1-score
0	0,9810	0,9994	0,9901
1	0,9980	0,9389	0,9675
modelo	0.9851	0.9849	0.9847

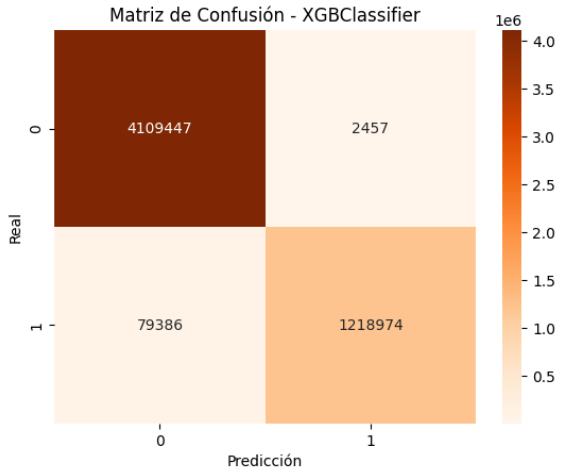

Figura 5.3: Matriz de confusión del modelo final.

Como se puede observar en la tabla 5.1, el modelo final tiene una precisión del 98,51 % y un recall del 98,49 %. Esto indica que el modelo es capaz de detectar el 98,49 % de los ataques y que el 98,51 % de las predicciones de ataque son correctas. Estos son unos resultados muy buenos, ya que el modelo es capaz de detectar la mayoría de los ataques y de no generar falsos positivos en la mayoría de los casos.

Tras la obtención de este modelo se podría haber optado por tratar de optimizar este reduciendo el número de variables predictorias por ejemplo eliminando aquellas con pesos ínfimos o haciendo una selección de variables wrapped. No obstante no se llevó a cabo porque se considera que el rendimiento del modelo ya era bueno por lo que no se requería y de hecho de haber hecho una selección de variables por el método wrapped, se hubiese dedicado un periodo de tiempo bastante grande a la evaluación de las variables teniendo en cuenta la naturaleza del mismo y las dimensiones del dataset.

5.2. Entorno de pruebas

Para comprobar el rendimiento del modelo final se generó un entorno de pruebas con el objetivo de simular tráfico en una posible red empresarial. Los elementos clave en este son:

- **Router pfSense**: se trata de un router virtualizado, en este caso se escogió el sistema operativo *pfSense* que además de cumplir las funciones de router, es un firewall. En el momento de ser creada la maquina virtual, se configuraron dos interfaces de red. Una de ellas se configuró como *WAN* y la otra como *LAN*. A nivel de la configuración inicial, es relevante qué interfaz se asigna a cada red, puesto que por defecto la interfaz *WAN* tiene restringido el acceso a la red *LAN*. Mientras tanto, la red *LAN* también tiene limitaciones, para su uso se tiene que haber configurado correctamente el gateway y si fuese necesario hacer que el propio *pfSense* actúe como servidor *DHCP* para la red *LAN*.

  Se escogió este sistema operativo para el router por ser de código abierto y tener una multitud de funciones relevantes para el sistema. La capacidad de establecer reglas complejas junto con que esté basado en Unix y todo lo que esto conlleva han sido determinantes para su elección. Por otro lado el hecho de que hubiese documentación del sistema accesible en internet también ha sido importante.

- **IDS Suricata**: se trata de un sistema de detección de intrusiones basado en firmas, que se ejecuta sobre el router *pfSense*. Este IDS se encarga de monitorizar el tráfico a través de las dos interfaces del router. Para su configuración se han utilizado no obstante el interés de este IDS no recae en las reglas que se pudiesen establecer para la detección, sino en la captura del tráfico que se genera en la red y su almacenamiento en ficheros *pcap*.

  La selección de este software ha sido por ser de uso libre y por la comunidad activa que tiene. Su versatilidad es bastante importante también. Un factor clave es el hecho de poder seleccionar los criterios con los que generar las capturas de red, pudiendo especificar que tipo de tráfico, en qué interfaz, con qué protocolo o en qué dirección ha de ser capturado.

- **Sensores simulados**: para generar tráfico en la red se han implementado varios sensores simulados, estos son contenedores de *Docker* que envían datos aleatorios supuestas centrales eléctricas de diferentes tipos, ya sea de energía solar, eólica, nuclear o gas. Los datos son enviados a través de *HTTP* a un servidor simulado que se encarga de recibirlos y almacenarlos. Estos sensores simulan el tráfico normal de una red empresarial, generando un flujo constante de datos que el modelo debe ser capaz de distinguir del tráfico malicioso.

  El uso de Docker es por ser de uso libre, además de por ser una herramienta muy versátil con la que con unos pocos comandos se pueden desplegar una gran cantidad de máquinas que no consumen demasiados recursos. Otro criterio importante es el hecho de poder usar

docker-compose.yml que permiten exportar las configuraciones de los contenedores a otro equipo y desplegar allí ese mismo servicio. Por esto si se hubiese querido y configurado todo bien, el numero de sensores simulados podría ser notablemente mayor.

Para el desarrollo de estos sensores simulados se generó un script en Python que en un bucle infinito en el que se envían los campos según el tipo de sensor simulado, estos son los casos:

Sensor de parque solar

- DEVICE_ID: id del sensor.
- timestamp: marca de tiempo del envío.
- temperature: temperatura generada aleatoriamente de una normal entre 15,0 y 25,0 grados.
- presure: presión generada aleatoriamente de una normal entre 1,0 y 2,0 atmósferas.

Sensor de central de gas

- DEVICE_ID: id del sensor.
- timestamp: marca de tiempo del envío.
- gas_flow: flujo de gas generado aleatoriamente de una normal entre 50,0 y 100,0 centímetros cúbicos.
- presure: presión generada aleatoriamente de una normal entre 1,5 y 3,0 atmósferas.

Sensor de central nuclear

- DEVICE_ID: id del sensor.
- timestamp: marca de tiempo del envío.
- radiation: radiación generada aleatoriamente de una normal entre 0,05 y 0,15 mSvs.
- coolant_temp: temperatura del refrigerante generada aleatoriamente de una normal entre 60,0 y 90,0 grados.

Sensor de aero generador

- DEVICE_ID: id del sensor.
- timestamp: marca de tiempo del envío.
- co2: dióxido de carbono generado aleatoriamente de una normal entre 300,0 y 800,0 partes por millón.
- noise: ruido generado aleatoriamente de una normal entre 30,0 y 70,0 decibelios.

El envío de los datos por parte de cada uno de los sensores es periódico con un timepo de espera entre cilo y cilo de 5 segundos. En total se desplegaron 10 sensores de los cuales eran

- 2 parques solares.
- 3 aerogeneradores.
- 4 centrales de gas.
- 1 central nuclear.

Se generaron estos sensores por la necesidad de que hubiese altas cantidades de tráfico benigno en la red para comprobar por ejemplo si el modelo era capaz de distinguir entre un ataque DoS o DDoS o de fuerza bruta cuyo objetivo fuese el servidor de datos o cualquier otro y el envío de datos a un servidor mediante HTTP y TCP.

- **Servidor de datos**: se trata de un servidor en docker simulado que recibe los datos de los sensores simulados. Este servidor se encarga de almacenar los datos recibidos y de responder a las peticiones de los sensores. El objetivo es simular un servidor real al que los sensores envían datos, generando tráfico normal en la red. También tiene la capacidad de consultar los datos almacenados mediante una interfaz web sencilla.

  Este servidor también ha sido desplegado en docker ya que es muy sencillo tener una base de datos operativa tan solo descargando una imagen de PostgreSQL y configurandola a tu gusto, sin generar archivos residuales en la máquina host.

  Esta máquina incluye también coódigo en Python para recibir los datos de los sensores así como para responder a consultas sobre los datos almacenados en la base de datos.

- **Generador de tráfico malicioso**: se trata de una máquina virtual de *Kali Linux* [36] que ejecuta un script mediante el cual inyecta tráfico malicioso de diferentes tipos en la red. Este tráfico es el que el modelo debe ser capaz de detectar como anómalo. El script inyecta tráfico de diferentes tipos, como *Brute Force*, *DDoS*, *Port Scanning*, entre otros. El objetivo es simular un ataque real contra el servidor de datos.

  La elección de Kali Linux ha sido por ser uno de los sistemas de código abierto más utilizados en actividades de ciberseguridad ofensiva y en todo ese entorno. Por esto, el sistema operativo tiene ya preinstaldas numerosas herramientas para llevar a cabo ataques de diversa índole y con la que comprobar la robustez y la seguridad de los sistemas y de la red.

- **Servidor victima**: se trata de un ordenador con sistema operativo ubuntu 22.04 LTS [37] que actúa como servidor víctima y contiene el servidor de datos. El objetivo es que haya una víctima para los ataques en la red y en esta máquina se ha instalado un servidor web y otro de bases de datos para simular un servidor contra el que hacer el ataque.

  Este servidor se ha basado en una máquina linux Ubuntu Server 24 ya que se buscaba un sistema fácil de desplegar, con una gran versatilidad y con la facilidad de instalar software mediante herramientas sencillas mediante linea de comandos, en este caso usando *apt*.

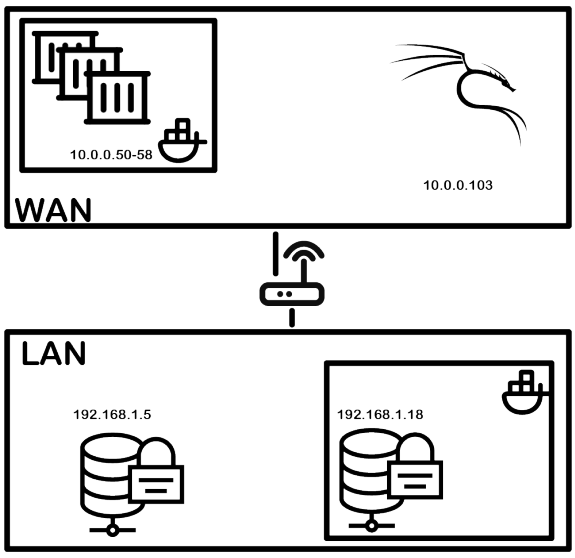

Figura 5.4: Arquitectura de red entorno de pruebas.

5.3. Pruebas

Tras el despliegue del entorno se ejecutaron scripts enfocados en ataques del tipo DoS, DDoS, mapeo de red, escaneo de puertos, fuerza bruta contra SSH y XSS contra la red y los equipos contenidos en esta. El entorno estuvo desplegado durante 2 horas en las que se estuvo capturando el tráfico de red. En este periodo estuvieron activos los sensores simulados, así como el servidor con el que se comunicaban, junto a esto estuvo activo también el servidor víctima.

Tras estas ejecuciones y teniendo en cuenta que de manera simultánea a los ataques, existían dispositivos no dañinos operando en este misma red, se extrageron las capturas de tráfico de red y se procedió a su tratamiento.

Primero se compiló el paquete con CICFlowMeter en el dispositivo con las capturas de red. Tras esto, mediante el comando

```
./cfm {directorio con los pcaps} {directorio de salida en csv}
```

se obtuvieron los ficheros .csv con los parámetros necesarios para el modelo generado previamente. EL conjunto de csv resultante de la conversión de estas capturas de tráfico contenía unas 140.000 entradas de flujos de la red simulada.

Antes de probar el modelo sobre estos datos en bruto, primero se procedió a la etiquetación de los datos. Todo el tráfico procedente de la máquina atacante fue etiquetado como maligno ya que esta estaba configurada de tal manera que exclusivamente accediese a esta red para los ataques, el resto de comunicaciones de la máquina eran dirigidas por una interfaz distinta y por lo tanto no se veían reflejadas en las capturas de tráfico de esta red.

Tras esto, se procedió a la eliminación de las columnas

- Flow ID
- Src IP
- Dst IP
- Src Port
- Timestamp
- Protocol

puesto que el modelo no utiliza estos datos. El siguiente paso fue convertir los nombres de las columnas para que coincidiesen con los parámetros que necesita el modelo. Se extrajo la columna con la variable a predecir y se almacenó para un posterior análisis del rendimiento del modelo. Los puertos presentes en la columna 'Dst Port' fueron categorizados según los 20 puertos más relevantes del modelo y aquellos que no se encontraban entre estos, fueron categorizados como 'other'.

Finalmente, tras todo el ajuste previo, se calculó la predicción del modelo sobre el dataset y los resultados fueron bastante satisfactorios. La precisión resultante es de un 99,23 % en el caso de los positivos, en el caso de los benignos, la precisión es de un 91,82 % por lo que es bastante aceptable. El recall de ambos casos es del 84,02 % en el caso de los malignos y de 99,64 % en el caso de los benignos.

6. Propuesta de despliegue y escalabilidad

Tras la prueba con los datos del entorno simulado, se ha desarrollado una API en Python mediante la cual, el modelo retorna la predicción de cada una de las filas de un csv que se le envíe.

Buscando que el despliegue de esta API sea lo más sencillo posible, se ha desarrollado un Docker compose que genera el contenedor con Python 3.11 en su versión ligera junto con las librerías necesarias para la API. Junto al fichero docker-compose.yml se encuentra el fichero .json que contiene el modelo final `XGBoost` y el fichero `API_modelo.py` que contiene el código en Python para la API.

6.1. Despliegue

El despliegue de la API requiere de lo siguiente:

- Docker y Docker Compose
- CPU x86-64
- 2GB de RAM disponible
- 300 MB de disco libre
- Puerto 8000 disponible (a menos que cambies el puerto externo en el docker-compose.yml)
- Contenido de API_modelo.zip en una carpeta

Tras tener esto, mediante la ejecución del comando

```
docker compose up
```

en el caso de tener instalado docker compose v2

```
docker-compose up
```

en el caso de tener instalado docker compose v1, Docker descargará lo necesario para que funcione el contenedor y tras esto ejecutará el código en python de la API. Tras el despliegue completo de esta, ya es posible hacer peticiones a la misma y obtener la predicción del modelo.

Las posibles respuestas de la API son:

- 200: OK, Petición correcta, "predictions": [0, 1, 0]
- 400: Bad Request, Entrada no válida (archivo no es CSV, columnas faltantes, error al leer el fichero) *detail* describe el problema., `"detail": "Se esperaba un archivo CSV."`
- 500: Internal Server Error, Fallo interno (modelo no disponible, error en la predicción, etc.) *detail* indica la causa., `"detail": .Error durante la predicción: ..."`
- 404: Not Found Endpoint inexistente., `"detail": "Not Found"`

Para probar la API, se puede ejecutar el siguiente comando en bash

```
curl <http://IP_API:PUERTO>
```

mediante este, se puede comprobar que está desplegada. Para obtener los resultados de la predicción de un csv, se puede ejecutar el siguiente comando en bash

```
curl -F "file=@{/Direccion/al/csv};type=text/csv" <http://IP_API:PUERTO>
```

6.2. Especificaciones de la API

La API es un código en Python que utiliza las librerías fastapi, uvicorn, pandas, scikit-learn y numpy. De aquñi algunas de las librerías ya han sido descritas previamente. Las que no se han utilizado hasta ahora son

- **fastapy**: es un framework de Python que permite desarrollar APIs web con versiones de Python 3.7. Con esta se ha creado un servicio RESTful sencillo que responde a dos endpoints.
- **uvicorn**: es un servidor web ASGI (Asynchronous Server Gateway Interface) de alto rendimiento para Python, diseñado para ejecutar aplicaciones asíncronas como las desarrolladas con FastAPI. Es ligero, rápido y compatible con asyncio, lo que permite gestionar múltiples conexiones concurrentes de manera eficiente. Uvicorn actúa como el servidor que escucha peticiones HTTP y las entrega a la aplicación, siendo ideal para entornos de producción y despliegue de APIs modernas.

La API da servicio a dos endpoints

- GET /: la operación get sobre el endpoint raíz retorna un archivo JSON con un mensaje de bienvenida y una descripción de las columnas que son requeridas en el csv que se mande como archivo a el otro endpoint.

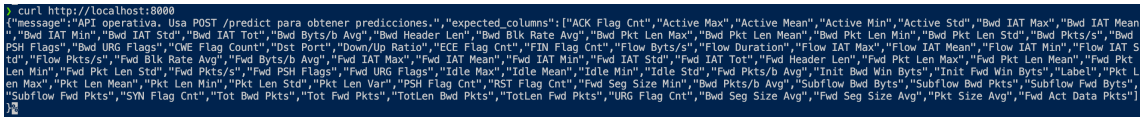

Figura 6.1: Respuesta del endpoint GET.

- POST /predict: la operación posr sobre el endpoint '/predict' requiere que se pase en el campo file un argumento que sea un fichero en formato csv. Si el fichero csv tiene las columnas correctas, el modelo generará una predicción para cada una de las filas del fichero y esta será retornada en formato JSON.

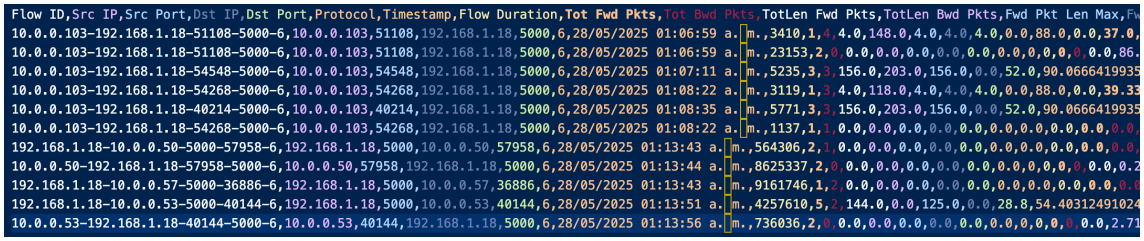

Figura 6.2: Petición endpoint predict.

Figura 6.3: Respuesta endpoint predict.

6.3. Escalabilidad

Debido a la implementación de un fichero docker-compose que con los ficheros necesarios de la API y el modelo, un usario cualquiera con conocimientos de informática podría desplegar una API que ejecute predicciones en cuestión de una hora como mucho y en el caso de que este tuviese que instalar el software previo necesario y se le presentase alguna complicación en el camino. Esto sumado a que tras el primer despliegue, se pueden hacer n despliegues más variando los puertos de escucha y siempre y cuando las capacidades de la máquina en la que se desplieguen sean suficientes para mantener a todos. Se trata de una solución de fácil despliegue y escalabilidad.

7. Conclusiones

A lo largo del desarrollo de este trabajo se ha podido constatar que el diseño y puesta en marcha de un modelo de *machine learning* para la detección de intrusiones depende en gran medida de la calidad del conjunto de datos, de su preparación adecuada y de la correcta elección y ajuste del modelo. El objetivo principal del proyecto, consistente en optimizar la preparación de datos y el *fine-tuning* de una solución IDS para reducir costes operativos y mejorar su escalabilidad en entornos reales, ha sido cumplido de forma satisfactoria. El modelo propuesto ha mostrado un rendimiento competitivo tanto en los conjuntos de datos originales como en un entorno simulado, lo que sugiere su aplicabilidad en escenarios reales con una validación adicional.

Respecto a los objetivos específicos:

- En relación con la preparación y procesamiento de datos de tráfico de red, se han aplicado técnicas de limpieza, normalización, etiquetado y extracción de características sobre los datasets CICIDS2017 y CICIDS2018. Este tratamiento ha permitido obtener una base sólida sobre la que entrenar modelos de detección de intrusiones.
- En cuanto al estudio y evaluación de algoritmos de aprendizaje supervisado, se ha realizado un análisis comparativo de varios algoritmos, destacando el buen rendimiento de Random Forest y XGBoost. Estos modelos han mostrado un equilibrio adecuado entre precisión, velocidad de entrenamiento y capacidad de generalización.
- El entrenamiento del modelo se ha llevado a cabo de forma iterativa, realizando ajustes en los hiperparámetros y técnicas de *fine-tuning*, lo que ha conducido a un modelo optimizado con buenos resultados en tareas de clasificación binaria del tráfico de red.
- Para la evaluación del rendimiento, se han utilizado métricas clásicas como *precision*, *recall* y *F1-score*. Los resultados obtenidos reflejan una buena capacidad del modelo para identificar patrones maliciosos y minimizar los falsos positivos, un aspecto clave en sistemas IDS reales.
- El modelo se ha puesto a prueba en un entorno simulado, distinto a los datos de entrenamiento, donde se ha comprobado su capacidad de detección en escenarios controlados con tráfico generado artificialmente. Esto ha permitido validar que el modelo no está sobreajustado y mantiene una capacidad de generalización aceptable.

- Finalmente, se han planteado recomendaciones para favorecer la escalabilidad del sistema, como el uso de contenedores ligeros, la modularización del entorno de pruebas y la adaptación del modelo a diferentes topologías de red. Estas propuestas buscan reducir los costes de despliegue y mantenimiento en entornos empresariales.

7.1. Trabajo futuro

Este trabajo ha demostrado que es posible entrenar modelos de *machine learning* eficaces para la detección de intrusiones utilizando datasets públicos y un entorno simulado que emula una red empresarial. No obstante, aún existen múltiples líneas de mejora y expansión que podrían abordarse en futuros desarrollos.

En primer lugar, una extensión natural sería aumentar la diversidad de los entornos de prueba. Aunque el modelo ha sido validado en un entorno simulado con tráfico generado artificialmente, sería de gran interés replicar el experimento en redes reales o con topologías más complejas, como redes segmentadas, entornos cloud híbridos o infraestructuras críticas. Esto permitiría evaluar la robustez del sistema frente a la heterogeneidad y dinamismo propios de entornos productivos.

Asimismo, sería relevante ampliar el repertorio de ataques simulados. En este trabajo se ha utilizado una base representativa proveniente de los datasets CICIDS2017 y CICIDS2018, pero incorporar otros vectores de ataque, como técnicas avanzadas de evasión, ataques de día cero o campañas persistentes, permitiría analizar el comportamiento del modelo ante amenazas más sofisticadas y actuales. También sería interesante estudiar el impacto de actores con distintos perfiles de comportamiento, incluyendo atacantes internos o automatizados mediante bots.

Otra línea de trabajo prometedora es la integración del sistema propuesto con plataformas SIEM (*Security Information and Event Management*). Esta integración permitiría centralizar la gestión de alertas, correlacionar eventos con otros sistemas de seguridad y proporcionar una visión más completa del estado de la red. Además, facilitaría la incorporación del modelo como una herramienta auxiliar en un flujo de trabajo de ciberseguridad ya existente.

Por otro lado, convendría explorar mecanismos de adaptación continua del modelo mediante técnicas de aprendizaje en línea o incremental. Esto permitiría que el sistema se actualizase a medida que cambia el comportamiento del tráfico de red o surgen nuevas amenazas, manteniendo su eficacia a lo largo del tiempo sin necesidad de un reentrenamiento completo.

Finalmente, sería interesante realizar un análisis más profundo sobre el coste computacional del modelo en dispositivos con recursos limitados, evaluando su viabilidad para el despliegue en edge devices o nodos periféricos de la red, lo cual abriría la puerta a arquitecturas distribuidas de detección con menor latencia y mayor escalabilidad.

8. Análisis de impacto

Este trabajo ha permitido al autor adquirir conocimientos sólidos en machine learning, ciberseguridad y gestión de grandes volúmenes de datos, así como experiencia práctica en la creación de entornos de prueba realistas.

Desde el punto de vista empresarial, la solución propuesta ofrece una mejora en la detección de intrusiones, reduciendo los falsos positivos y aumentando la eficacia general del sistema. Esto conlleva un ahorro en recursos humanos y técnicos, y facilita una mejor protección frente a amenazas reales sin necesidad de infraestructuras complejas.

A nivel social y económico, contribuir a la seguridad de los servicios digitales refuerza la confianza en las tecnologías actuales y ayuda a proteger infraestructuras críticas. Además, al optimizar el uso de recursos, las empresas pueden destinar sus inversiones a otras áreas estratégicas.

Aunque el impacto medioambiental no es directo, una gestión más eficiente de los recursos informáticos puede derivar en un menor consumo energético.

Finalmente, el trabajo fomenta la concienciación sobre la importancia de la ciberseguridad tanto en el ámbito empresarial como en el social, y se alinea con los Objetivos de Desarrollo Sostenible, especialmente el número 9 (industria, innovación e infraestructura) puesto que fomenta la mejora y el desarrollo de las tecnologías de la información y las infraestructuras que las sostienen mediante el uso de tecnologías nuevas, aportando así innovación en el ámbito de la ciberseguridad y mejorando la robustez de las redes industriales.

Asimismo, de forma complementaria, el trabajo se relaciona con el ODS número 7 (Energía asequible y no contaminante). Aunque no se aborda directamente la cuestión energética, la eficiencia del modelo es considerable, en cosas como la escalabilidad y posibilidad de implementación en infraestructuras distribuidas o de bajo consumo. Esto puede repercutir positivamente en la reducción del gasto energético asociado al procesamiento de grandes volúmenes de datos en sistemas IDS tradicionales, promoviendo una infraestructura digital sostenible.

En definitiva, este trabajo no solo responde a una necesidad técnica dentro del ámbito de la ciberseguridad, sino que también impulsa mejoras en sostenibilidad, resiliencia y eficiencia, contribuyendo a la construcción de una infraestructura digital más segura y alineada con los desafíos globales.

Bibliografía

[1] Cloudflare, *DDoS attack trends for 2024 Q1*, 2024. dirección: https://www.cloudflare.com/ddos/2024-q1-report/.

[2] OWASP, *Brute Force Attack*, 2023. dirección: https://owasp.org/www-community/attacks/Brute_force_attack.

[3] N. Moustafa y J. Slay, «The evaluation of network anomaly detection systems: Statistical analysis of the UNSW-NB15 dataset and the comparison with the KDD99 dataset», *Information Security Journal: A Global Perspective*, vol. 25, n.º 1-3, págs. 18-31, 2016.

[4] OWASP, *Cross Site Scripting (XSS)*, 2023. dirección: https://owasp.org/www-community/attacks/xss/.

[5] R. Sommer y V. Paxson, «Outside the closed world: On using machine learning for network intrusion detection», *IEEE Symposium on Security and Privacy*, págs. 305-316, 2010.

[6] Z. Ahmad, A. Shahid Khan, W. S. Cheah, J. Abdullah y F. Ahmad, «Network intrusion detection system: A systematic study of machine learning and deep learning approaches», *Transactions on Emerging Telecommunications Technologies*, vol. 32, n.º 1, e4150, 2021. DOI: 10.1002/ett.4150.

[7] M. V. R. Sarobin, P. R. Rukmani y E. M. Anita, *Machine LearningBased Intrusion Detection System*, https://www.researchgate.net/publication/391201550_Machine_Learning_Based_Intrusion_Detection_System, 2024.

[8] N. Y. Chen, P. W. Chou, J. S. Li e I. H. Liu, «A Case Study of NetworkBased Intrusion Detection System Deployment in Industrial Control Systems with Network Isolation», en *Proceedings of The 2024 International Conference on Artificial Life and Robotics (ICAROB 2024)*, ALife Robotics Corporation Ltd, 2024, págs. 30-33. DOI: 10.5954/icarob.2024.os1-5.

[9] M. A. Umer, K. N. Junejo, M. T. Jilani y A. P. Mathur, «Machine Learning for Intrusion Detection in Industrial Control Systems: Applications, Challenges, and Recommendations», *International Journal of Critical Infrastructure Protection*, vol. 38, pág. 100 516, 2022. DOI: 10.1016/j.ijcip.2022.100516.

[10] E. S. Olivas, M. A. S.-M. Isla, R. G. Cruz, B. C. Caballero y P. C. Michelena, *Sitemas de Aprendizaje Automatico*, R.-M. Editorial, ed. Ra-Ma, 2023. dirección: https://www.ra-ma.es/libro/sistemas-de-aprendizaje-automatico/.

[11] Python Software Foundation, *Python Programming Language*, https://www.python.org, Accedido: 10-05-2025, 2024.

[12] Project Jupyter, *Jupyter Notebooks*, https://jupyter.org, Accedido: 11-05-2025, 2024.

[13] NumPy Developers, *NumPy*, https://numpy.org, Accedido: 11-05-2025, 2025.

[14] Scikit-learn developers, *Scikit-learn: Machine Learning in Python*, https://scikit-learn.org, Accedido: 13-05-2025, 2025.

[15] The pandas development team, *pandas: Python Data Analysis Library*, https://pandas.pydata.org, Accedido: 13-05-2025, 2025.

[16] Matplotlib Development Team, *Matplotlib: Visualization with Python*, https://matplotlib.org, Accedido: 13-05-2025, 2025.

[17] TensorFlow Developers, *TensorFlow: An end-to-end open source platform for machine learning*, https://www.tensorflow.org, Accedido: 13-05-2025, 2025.

[18] Seaborn Developers, *Seaborn: statistical data visualization*, https://seaborn.pydata.org, Accedido: 13-05-2025, 2025.

[19] IBM, *Árboles de decisión*, 2024. dirección: https://www.ibm.com/es-es/think/topics/decision-trees.

[20] I. Goodfellow, Y. Bengio y A. Courville, *Deep Learning*. MIT Press, 2016, pág. 72, ISBN: 9780262035613. dirección: https://www.deeplearningbook.org/contents/prob.html.

[21] XGBoost Contributors, *XGBoost: A Scalable Tree Boosting System*, https://xgboost.readthedocs.io, Accedido: 13-05-2025, 2025.

[22] T. Chen y C. Guestrin, «XGBoost: A Scalable Tree Boosting System», *arXiv preprint arXiv:1603.02754*, jun. de 2016, Véase la Sección 2.3: Shrinkage and Column Subsampling. arXiv: 1603.02754 [cs.LG].

[23] Canadian Institute for Cybersecurity, *CICFlowMeter: Network Traffic Flow Generator*, https://github.com/UNBCIC/CICFlowMeter, Accessed: 2025-06-01, 2016.

[24] I. Sharafaldin, A. H. Lashkari y A. A. Ghorbani, «Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization», en *Proceedings of the 4th International Conference on Information Systems Security and Privacy (ICISSP)*, Portugal, 2018, págs. 108-116.

[25] I. Sharafaldin, A. H. Lashkari y A. A. Ghorbani, *CSECICIDS2018 on AWS: Canadian Institute for Cybersecurity Intrusion Detection Evaluation Dataset*, Dataset con tráfico benigno y siete escenarios de ataque; 80+ features extraídas con CICFlowMeter-V3., Canadian Institute for Cybersecurity, University of New Brunswick, 2018.

[26] C. I. for Cybersecurity. «CIC-IDS2018 Dataset Architecture Diagram». dirección: https://www.unb.ca/cic/_assets/images/cse-cic-ids2018.jpg.

[27] pfSense Foundation, *pfSense Software Router/Firewall*, Online, 2025. dirección: https://www.pfsense.org/.

[28] Proxmox Server Solutions GmbH, *Proxmox Virtual Environment*, Version 8.x, Proxmox Server Solutions GmbH, 2024. dirección: https://www.proxmox.com/en/proxmox-ve.

[29] Docker Inc., *Docker: Empowering App Development for Developers*, Version 24.x, Docker Inc., 2024. dirección: https://www.docker.com.

[30] Open Information Security Foundation, *Suricata: Open Source IDS/IPS/NSM Engine*, ver. 7.0.10, Open Information Security Foundation, 2025. dirección: https://suricata.io/download/.

[31] Chapman, P. and Clinton, J. and Kerber, R. and Khabaza, T. and Reinartz, T. and Shearer, C. and Wirth, R., *CRISP-DM 1.0: Step-by-step data mining guide*, https://www.the-modeling-agency.com/crisp-dm.pdf, Accessed July 1, 2025, 2000.

[32] IBM Corporation, *CRISP-DM Process Diagram*, https://www.ibm.com/docs/es/SS3RA7_sub/modeler_crispdm_ddita/clementine/images/crisp_process.jpg, Accedido el 1 de julio de 2025, 2023.

[33] F. Pedregosa et al., «Scikit-learn: Machine Learning in Python», *Journal of Machine Learning Research*, vol. 12, págs. 2825-2830, 2011. dirección: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html.

[34] Chen, Tianqi, Guestrin y Carlos, «XGBoost: A Scalable Tree Boosting System», ago. de 2016, págs. 785-794.

[35] G. Ke et al., «LightGBM: A Highly Efficient Gradient Boosting Decision Tree», en *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 30, 2017. dirección: https://lightgbm.readthedocs.io/en/stable/Python-API.html.

[36] Offensive Security, *Kali Linux*, Online, 2025. dirección: https://www.kali.org/.

[37] Canonical Ltd., *Ubuntu 25.04 Plucky Puffin Release Announcement*, https://canonical.com/blog/canonical-releases-ubuntu-25-04-plucky-puffin, abr. de 2025.

Recibo digital

Este recibo confirma quesu trabajo ha sido recibido por Turnitin. A continuación podrá ver la información del recibo con respecto a su entrega.

La primera página de tus entregas se muestra abajo.

Autor de la entrega:	ALEJANDRO ORTEGA HERNANDEZ
Título del ejercicio:	Turnitin Memoria Final
Título de la entrega:	Alejandro Ortega Hernández.pdf
Nombre del archivo:	9290_ALEJANDRO_ORTEGA_HERNANDEZ_Alejandro_Ortega_H...
Tamaño del archivo:	1.93M
Total páginas:	49
Total de palabras:	12,930
Total de caracteres:	69,098
Fecha de entrega:	02-jul-2025 12:03a. m. (UTC+0200)
Identificador de la entrega:	2709006302

Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Grado en Ingeniería Informática

Trabajo Fin de Grado

Creación de un Modelo ML para el Despliegue de IDS en Entornos Empresariales

Autor: Alejandro Ortega Hernández
Tutor: Antonio Jesús Díaz Honrubia

Madrid, julio 2025

Este documento esta firmado por

	**Firmante**	CN=tfgm.fi.upm.es, OU=CCFI, O=ETS Ingenieros Informaticos - UPM, C=ES
	Fecha/Hora	Wed Jul 02 02:45:59 CEST 2025
	Emisor del Certificado	EMAILADDRESS=camanager@etsiinf.upm.es, CN=CA ETS Ingenieros Informaticos, O=ETS Ingenieros Informaticos - UPM, C=ES
	Numero de Serie	561
	Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)