GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

Análisis de modelos de Machine Learning para la detección y prevención de amenazas en ciberseguridad de redes

Autor: Victoria Sánchez de León González

Director: Alfonso Vázquez Requejo

Victoria.SdL	

Madrid

Declaración de originalidad

Declaro bajo mi responsabilidad que el Proyecto presentado con el título **Análisis de modelos de Machine Learning para la detección y prevención de amenazas en ciberseguridad de redes** e la ETS de Ingeniería – ICAI de la Universidad Pontificia Comillas en el curso académico **2024/2025** es de mi autoría y no ha sido presentado con anterioridad a otros efectos. El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido tomada de otros documentos está debidamente referenciada.

Uso de Inteligencia Artificial[1]

Declaro bajo mi responsabilidad que (indicar la opción correcta):

☐ No he utilizado Inteligencia Artificial en la elaboración del presente documento.

■ He utilizado Inteligencia Artificial en la elaboración del presente documento y/o del Anexo B siempre en las condiciones permitidas por la Universidad Pontificia Comillas, es decir, aplicando el Nivel 2 de la Escala de Evaluación de Perkins et al. (2024): "*La IA puede utilizarse para actividades previas a la tarea, como la lluvia de ideas, la descripción y la investigación inicial. Este nivel se centra en el uso de la IA para la planificación, las síntesis y la generación de ideas, pero las evaluaciones deben hacer hincapié en la capacidad de desarrollar y refinar estas ideas de forma independiente*". En concreto, las Inteligencia Artificial ha sido empleada para:

La Inteligencia Artificial ha sido empleada como apoyo en las siguientes tareas:

- Perfeccionamiento de la redacción de secciones del documento, partiendo en todo momento de ideas, datos experimentales y decisiones técnicas propias.
- Revisión y mejora del estilo, la coherencia y la claridad del texto.
- Corrección de errores en el código Python desarrollado para el estudio experimental.
- Traducción y comprensión de fuentes bibliográficas en inglés.

En ningún caso se ha delegado en la IA el análisis de resultados, la interpretación de los datos, ni la elaboración de conclusiones. Todo el trabajo experimental, incluyendo el diseño del pipeline, la selección de modelos y la evaluación de resultados, ha sido realizado por mí.

Firmado (alumno):

[1] Esta declaración se refiere al uso de la Inteligencia Artificial generativa para realizar los documentos del Proyecto (Anexo B y Memoria). No aplica a Proyectos donde, por su naturaleza, deban emplear inteligencia artificial como parte de los mismos (aplicación de técnicas de aprendizaje automático, redes neuronales, análisis de datos…)

Autorización para la entrega del Proyecto

El Director del Proyecto	El co-Director del Proyecto (si aplica)
	
Fdo: ALFONSO VÁZQUEZ REQUEJO	Fdo:
Fecha: 12 Junio 2026	Fecha:

COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI

GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

Análisis de modelos de Machine Learning para la detección y prevención de amenazas en ciberseguridad de redes

Autor: Victoria Sánchez de León González

Director: Alfonso Vázquez Requejo

Madrid

ANÁLISIS DE MODELOS DE MACHINE LEARNING PARA LA DETECCIÓN Y PREVENCIÓN DE AMENAZAS EN CIBERSEGURIDAD DE REDES

Autor: Sánchez de León González, Victoria.
Director: Vázquez Requejo, Alfonso.
Entidad Colaboradora: ICAI – Universidad Pontificia Comillas

RESUMEN DEL PROYECTO

Este trabajo entrena y compara seis modelos de Machine Learning para la detección de intrusiones en redes sobre el conjunto de datos CICIDS2017, que contiene más de 1,6 millones de flujos de red etiquetados en diez categorías: tráfico benigno y nueve tipos de ataque. Los modelos evaluados son K-Nearest Neighbors (KNN), árbol de decisión CART, Random Forest, red neuronal MLP, Support Vector Machine (SVM) y el modelo de lenguaje Llama 3.2-1B El trabajo también incluye un análisis del modelo Claude Mythos Preview y su impacto en el ámbito de la ciberseguridad.

Palabras clave: detección de intrusiones, machine learning, ciberseguridad de redes, CICIDS2017, clasificación multiclase

1. Introducción

El número de dispositivos conectados a Internet ha crecido exponencialmente en las últimas décadas y con ello también la superficie expuesta a ataques. Hoy en día, prácticamente cualquier organización, ya sea una universidad o empresa, depende de una red para operar. Por lo tanto, asegurar la seguridad de esa red se ha vuelto una prioridad real y cotidiana. Los sistemas de detección de intrusiones (IDS) son una capa defensiva importante y el Machine Learning lleva años siendo estudiado con el objetivo de mejorar su capacidad para identificar amenazas de forma automática.

El interés por estos sistemas es cada vez más grande y se busca perfeccionarlos lo máximo posible. Un IDS que genera demasiadas falsas alarmas acaba siendo ignorado por los analistas. Uno que no detecta ciertos tipos de ataque pierde su utilidad. Por eso uno de los tipos de amenazas detecta bien cada uno y en cuáles falla.

A lo largo del proyecto se trabaja sobre el dataset CICIDS2017, un conjunto de datos que contiene tráfico real de red etiquetado con diez categorías: tráfico benigno, DDoS, PortScan, FTP-Patator, SSH-Patator, Bot, Web Attack–Brute Force, Web Attack–XSS, Web Attack–SQL Injection e Infiltration. Tiene más de 1,6 millones de instancias pero una distribución muy desigual. En efecto, más del 80 % del tráfico es benigno y algunas clases tienen menos de 25 instancias en todo el conjunto.

2. Definición del Proyecto

El objetivo principal es comparar varios clasificadores supervisados para la detección de ataques de red. Para ello se entrena y ajusta cada modelo sobre el mismo dataset y se evalúa su rendimiento mediante F1-macro. El trabajo se divide en una parte teórica y una parte práctica. Por un lado, la parte teórica cubre:

- los fundamentos de ciberseguridad: tipos de ataques, técnicas de ataque, evolución histórica.

- los fundamentos de Machine Learning necesarios para entender los modelos aplicados a continuación.

La parte práctica implementa los seis modelos en Python. Para ello, se aplica técnicas de preprocesamiento y balanceo de clases, una búsqueda y selección de hiperparámetros y finalmente se evalúa el rendimiento sobre un conjunto de test independiente.

Además, el trabajo incluye una sección sobre Claude Mythos Preview, el modelo más reciente de Anthropic. No es un IDS sino un modelo de lenguaje de propósito general, que ha demostrado grandes capacidades en detección de vulnerabilidades de red. Su mención en el trabajo permite contextualizar los resultados obtenidos en la situación actual de la IA aplicada a ciberseguridad.

3. Descripción del modelo/sistema/herramienta

La dinámica del trabajo sigue los mismos pasos para todos los modelos supervisados, como refleja la figura 1. Los seis archivos CSV del CICIDS2017 se cargan y concatenan en un único DataFrame. A continuación, se limpian los datos hasta obtener un dataset de 1.606.989 instancias y 78 variables numéricas además de la etiqueta de clase.

La división en entrenamiento y test se hace antes de aplicar. SMOTE se aplica solo sobre el entrenamiento, con un tope de 10.000 instancias por clase pues sobremuestrear hasta el nivel de BENIGN generaría un conjunto irreal y de tamaño inmanejable.

La búsqueda de hiperparámetros se hace con GridSearchCV o RandomizedSearchCV según el modelo, optimizando siempre el F1-macro. En el caso de que al modelo le suponga demasiado coste computacional trabajar sobre el conjunto completo, la búsqueda se hace sobre una submuestra. El modelo final siempre se evalúa sobre el test original sin tocar.

Este proceso, común a todos los clasificadores, queda resumido en la Figura 1.

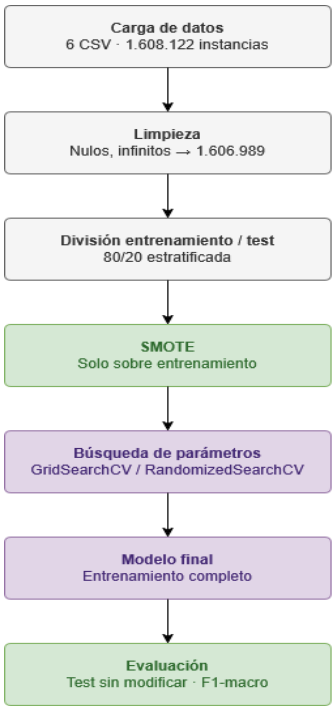

Figura 1- Pipeline de clasificación aplicado a los modelos supervisados

4. Resultados

La Tabla 1 siguiente resume los resultados globales de cada modelo:

MODELO	MACRO F1 (test)
Random Forest + SMOTE	0,83
KNN	0,85 (submuestra 5.000)
CART + SMOTE	0,71
MLP + SMOTE	0,70
SVM + SMOTE	0,45
Llama 3.2-1B (zero-shot)	0,02

Tabla 1- Resultados globales de F1-macro por modelo

Vemos que Random Forest obtiene el mejor resultado global sobre el test completo. KNN alcanza 0,85 pero sobre una submuestra de 5.000 instancias, lo que hace que la comparación directa no sea realmente justa. Esto se debe a que el coste computacional de KNN en predicción hace inviable evaluarlo sobre el test completo. CART y MLP quedan en posiciones intermedias. SVM, entrenada sobre solo 15.000 instancias por limitaciones computacionales, obtiene 0,45. El LLM clasifica prácticamente todo como tráfico benigno, es decir, que no detecta ningún ataque.

Cabe destacar algunos patrones claros. DDoS, FTP-Patator, SSH-Patator y PortScan los detectan bien todos los modelos gracias a sus características estadísticas constantes y a su gran volumen de datos de entrenamiento. Los ataques web son el punto débil generalizado: la confusión entre Brute Force y XSS aparece en todos los modelos porque tienen características de flujo muy similares. Bot varía bastante según el modelo y SQL Injection, con solo 4 instancias en el test, es prácticamente indetectable para todos.

5. Conclusiones

Random Forest con SMOTE es el modelo más adecuado para este problema ya que presenta el mejor F1 global y la mejor detección en las clases difíciles. CART sería preferible en entornos donde la interpretabilidad es prioritaria pero sacrificando rendimiento.

El experimento con Llama 3.2-1B confirma que un modelo de lenguaje pequeño sin entrenamiento específico no consigue clasificar flujos de red de forma útil. Se puede usar para otras tareas complementarias como la explicación de código o vulnerabilidades al analista. En cambio, existen otro modelos LLM más potentes, como Claude Mythos Preview, que muestran un comportamiento muy diferente con resultados mucho mejores y prometedores en detección de vulnerabilidades de red.

Las limitaciones más importantes del trabajo no son de los modelos sino del dataset. El CICIDS2017 presenta un gran desbalanceo de clases y las características son estadísticas de flujo sin información sobre el contenido de los paquetes.

6. Referencias

Las referencias completas se recogen en la bibliografía del proyecto. Las principales fuentes para esta sección son:

Buczak, A. L., y Guven, E. (2016). A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications Surveys & Tutorials*, 18(2), 1153–1176.

Kurose, J., & Ross, K. (2017). *Computer Networking: A Top-Down Approach*. Pearson.

Liao, H. J., Lin, C. H. R., Lin, Y. C., y Tung, K. Y. (2013). Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications*, 36(1), 16–24.

AI Security Institute (AISI) (2026). *Our evaluation of Claude Mythos Preview's cyber capabilities*. UK Department for Science, Innovation and Technology.

Fortune (2026, 7 de abril). Anthropic is giving companies, including Amazon, Apple, and Microsoft, access to its unreleased Claude Mythos model to prepare cybersecurity defense.

ArmorCode (2026). *The Claude Mythos Security Playbook: Operationalizing AI-Scale Vulnerability Discovery*.

ANALYSIS OF MACHINE LEARNING MODELS FOR THE DETECTION AND PREVENTION OF CYBERSECURITY THREATS IN NETWORKS

Author: Sánchez de León González, Victoria.
Supervisor: Vázquez Requejo, Alfonso.
Collaborating Entity: ICAI – Universidad Pontificia Comillas

ABSTRACT

This study trains and compares six machine learning models for network intrusion detection using the CICIDS2017 dataset, which contains over 1.6 million network flows labelled across ten categories: benign traffic and nine types of attack. The models evaluated are K-Nearest Neighbours (KNN), CART decision tree, Random Forest, MLP neural network, Support Vector Machine (SVM) and the Llama 3.2-1B language model. The work also includes an analysis of the Claude Mythos Preview model and its impact on the field of cybersecurity.

Keywords: intrusion detection, machine learning, network cybersecurity, CICIDS2017, multi-class classification

1. Introduction

The number of devices connected to the internet has grown exponentially in recent decades, and with it, the surface area exposed to attacks. Nowadays, virtually every organization, be it a university, a business or a public administration, relies on a network to operate. Consequently, ensuring the security of that network has become a real and daily priority. Intrusion detection systems (IDS) are an important layer of defense, and machine learning has been studied for years with the aim of improving their ability to identify threats automatically.

Interest in these systems is growing, and efforts are being made to refine them as much as possible. An IDS that generates too many false alarms ends up being ignored by analysts. One that fails to detect certain types of attack loses its usefulness. That is why one of the objectives of this work is to evaluate not only the overall performance metrics of each model, but also which types of threats each model detects well and where it falls short.

Throughout the project, work is carried out on the CICIDS2017 dataset, a dataset containing real network traffic labelled with ten categories: benign traffic, DDoS, PortScan, FTP-Patator, SSH-Patator, Bot, Web Attack – Brute Force, Web Attack–XSS, Web Attack–SQL Injection and Infiltration. It contains over 1.6 million instances but has a very uneven distribution. Indeed, over 80% of the traffic is benign, and some classes have

2. Project Definition

The main objective is to compare various supervised classifiers for the detection of network attacks. To this end, each model is trained and fine-tuned on the same dataset, and its performance is evaluated using the F1-macro score.

The work is divided into a theoretical section and a practical section. The theoretical section covers:

- the fundamentals of cybersecurity: types of attacks, attack techniques, and historical development.
- the fundamentals of machine learning necessary to understand the models applied below.

The practical part implements the six models in Python. To do this, pre-processing and class balancing techniques are applied, followed by a search for and selection of hyperparameters, and finally the performance is evaluated on an independent test set.

3. Description of the model/system/tool

The workflow follows the same steps for all supervised models, as shown in Figure 1. The six CSV files from CICIDS2017 are loaded and concatenated into a single DataFrame. The data is then cleaned to produce a dataset of 1,606,989 instances and 78 numerical variables, in addition to the class label.

The split into training and test sets is performed before applying SMOTE and always using stratified sampling. SMOTE is applied only to the training set, with a cap of 10,000 instances per class, as oversampling down to the BENIGN level would generate an unrealistic and unmanageably large dataset.

Hyperparameter tuning is performed using GridSearchCV or RandomizedSearchCV depending on the model, always optimising the F1-macro. In the event that working on the full dataset imposes too high a computational cost on the model, the search is performed on a subsample. The final model is always evaluated on the original, untouched test set.

Figure 1 summarizes the process followed by all classifiers.

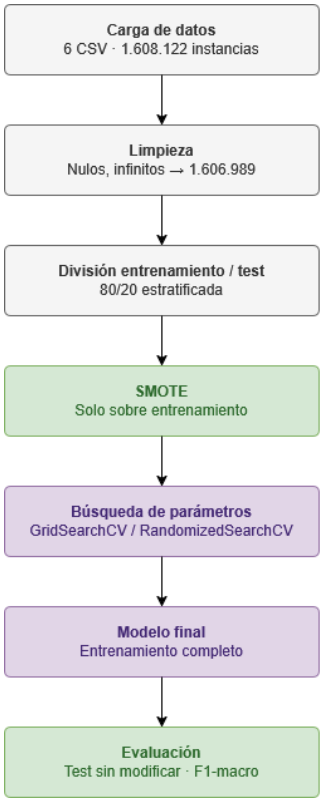

Figure 1- Classification pipeline applied to supervised models

4. Results:

The following Table 1 summarizes the overall results for each model:

MODELO	MACRO F1 (test)
Random Forest + SMOTE	0,83
KNN	0,85 (submuestra 5.000)
CART + SMOTE	0,71
MLP + SMOTE	0,70
SVM + SMOTE	0,45
Llama 3.2-1B (zero-shot)	0,02

Table 1-Overall F1-macro scores by model

We can see that Random Forest achieves the best overall result on the full test set. KNN reaches 0.85, but this is based on a subsample of 5,000 instances, which means a direct comparison is not really fair. This is because the computational cost of KNN for prediction makes it impractical to evaluate it on the full test set. CART and MLP occupy intermediate positions. SVM, trained on only 15,000 instances due to computational constraints, achieves 0.45. The LLM classifies practically everything as benign traffic, meaning it does not detect any attacks.

Some clear patterns are worth noting. DDoS, FTP-Patator, SSH-Patator and PortScan are detected well by all models thanks to their consistent statistical characteristics and the large volume of training data. Web attacks are the general weak point: confusion between Brute Force and XSS occurs in all models because they have very similar flow characteristics. Bot varies considerably depending on the model, and SQL Injection, with only 4 instances in the test, is practically undetectable by all.

5. Conclusions

Random Forest with SMOTE is the most suitable model for this problem, as it achieves the best overall F1 score and the best detection rates for difficult classes. CART would be preferable in environments where interpretability is a priority, albeit at the expense of performance.

The experiment with Llama 3.2-1B confirms that a small language model without specific training cannot classify network flows effectively. It can be used for other complementary tasks such as explaining code or vulnerabilities to the analyst. However, there are other more powerful LLM models, such as Claude Mythos Preview, which exhibit very different behaviour with much better and more promising results in network vulnerability detection.

The most significant limitations of the work lie not with the models but with the dataset. CICIDS2017 exhibits a significant class imbalance, and the features consist of flow statistics without information on packet content.

6. References:

Buczak, A. L., y Guven, E. (2016). A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications Surveys & Tutorials*, 18(2), 1153–1176.

Kurose, J., & Ross, K. (2017). *Computer Networking: A Top-Down Approach*. Pearson.

Liao, H. J., Lin, C. H. R., Lin, Y. C., y Tung, K. Y. (2013). Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications*, 36(1), 16–24.

AI Security Institute (AISI) (2026). *Our evaluation of Claude Mythos Preview's cyber capabilities*. UK Department for Science, Innovation and Technology.

Fortune (2026, 7 de abril). Anthropic is giving companies, including Amazon, Apple, and Microsoft, access to its unreleased Claude Mythos model to prepare cybersecurity defense.

ArmorCode (2026). *The Claude Mythos Security Playbook: Operationalizing AI-Scale Vulnerability Discovery*.

Índice de la memoria

Índice de figuras

Índice de tablas

Capítulo 1. INTRODUCCIÓN

1.1 CONTEXTO: CIBERSEGURIDAD EN LA ERA HIPERCONECTADA

Hoy en día, casi todas las entidades, incluyendo universidades, empresas privadas e instituciones públicas, están conectadas a Internet. Esta infraestructura conecta miles de millones de dispositivos, desde servidores hasta electrodomésticos o relojes inteligentes, que intercambian información constantemente (Kurose & Ross, 2017). El problema es que esta conectividad ha ampliado también la superficie expuesta a ataques, y las amenazas son cada vez más variadas y difíciles de anticipar. La seguridad de red se ha convertido en una preocupación real para cualquier organización.

Desde un enfoque estructural, los ataques se dividen en no estructurados, estructurados, directos e indirectos (Kurose & Ross, 2017; Kim & Solomon, 2016). Los no estructurados suelen venir de personas con conocimientos técnicos limitados y motivos variados, curiosidad, desafío, renombre, mientras que los estructurados los organizan actores con más recursos y objetivos concretos. Los directos implican una interacción directa entre atacante y sistema objetivo; los indirectos usan intermediarios para ocultar el origen del ataque.

Un ataque cibernético generalmente sigue un orden de etapas. Primero, el reconocimiento: el atacante recopila información sobre su objetivo. Por eso, en herramientas como SIEMs o EDRs, una de las alertas más útiles en la práctica es la que detecta escaneos de puertos desde una IP sospechosa. Después viene el acceso y la persistencia en el sistema. Por último, la fase de encubrimiento: borrar rastros, devolver archivos a su estado anterior y eliminar registros (Kurose & Ross, 2017; Kim & Solomon, 2016).

Los motivos detrás de un ataque pueden ser muy distintos: venganza, reconocimiento, ideología, espionaje o ciberguerra (Kim & Solomon, 2016). En todos los casos, el objetivo suele ser comprometer la confidencialidad, la integridad o la disponibilidad de la información.

1.2 EVOLUCIÓN HISTÓRICA DE LOS CIBERATAQUES

Los ciberataques no siempre fueron lo que son hoy. Los primeros virus y gusanos aparecieron en entornos académicos durante los años setenta y ochenta, creados más por curiosidad que por intención maliciosa (Netwrix, 2024). En los noventa, con la expansión de las redes locales y los ordenadores personales, estos programas empezaron a propagarse de forma masiva a través de disquetes y correos electrónicos. Ya en los años 2000, el dinero empezó a ser la motivación principal: ahí es cuando el phishing, el spyware y la ingeniería social se volvieron habituales (Netwrix, 2024).

La década de 2010 consolidó la ciberdelincuencia organizada. Casos como *CryptoLocker* en 2013 o WannaCry en 2017 mostraron que un solo ataque podía afectar a cientos de miles de sistemas en todo el mundo. Al mismo tiempo, fueron apareciendo ataques con objetivos más políticos o estratégicos, como el ciberespionaje entre estados (Netwrix, 2024).

Entender esta evolución ayuda a diseñar mejores defensas. Al final, muchos de los ataques actuales siguen patrones que ya se veían hace décadas, aunque con herramientas mucho más sofisticadas.

1.3 MOTIVACIÓN DEL PROYECTO

La motivación de este proyecto parte de una pregunta concreta: ¿hasta qué punto pueden los modelos de Machine Learning detectar amenazas en redes de forma fiable en un entorno real? Aunque hay muchos estudios que muestran resultados prometedores, la adopción en producción sigue siendo limitada, en parte porque un modelo que funciona bien en un papel no siempre es útil en un SOC. Si genera demasiadas falsas alarmas, los analistas acaban ignorándolas. Si es demasiado complejo, nadie entiende por qué clasifica como lo hace, y eso es un problema cuando hay que justificar una decisión de seguridad.

Esto lo pude ver de cerca durante mi experiencia en el departamento de ciberseguridad de Telefónica Tech, donde participé en el análisis manual de alertas en un SIEM. Para cada evento había que revisar variables como la IP de origen, el número de conexiones, la

duración o el volumen de tráfico, y decidir si era una amenaza real o no. Con un volumen alto de eventos diarios, ese proceso consume mucho tiempo. Automatizar una parte de ese análisis con modelos de ML permitiría liberar recursos y reducir tiempos de respuesta.

Este trabajo no pretende encontrar el modelo perfecto, sino hacer una comparativa honesta: entrenar varios algoritmos bajo las mismas condiciones, ver cómo se comportan con datos reales y desbalanceados, y sacar conclusiones sobre cuáles tienen más sentido en un contexto operativo.

1.4 OBJETIVOS DEL PROYECTO

Objetivo general:

Analizar y evaluar la aplicabilidad de distintos modelos de Machine Learning en la detección y prevención de amenazas en redes de comunicaciones, comparando su comportamiento, sus ventajas y sus limitaciones desde un punto de vista técnico y práctico.

Objetivos específicos

A partir de este objetivo general, se definen los siguientes:

1. Estudiar los fundamentos teóricos de los modelos de Machine Learning aplicables a la ciberseguridad de redes, con el fin de justificar la selección de algoritmos y entender su adecuación al tipo de datos de tráfico de red.

2. Diseñar una metodología experimental común a todos los modelos, que incluya la selección del dataset, el preprocesado, la extracción de características y la definición de métricas de evaluación, garantizando que los resultados sean comparables y reproducibles.

3. Entrenar y evaluar los modelos seleccionados sobre datos reales de tráfico de red, analizando su capacidad para identificar amenazas, su comportamiento frente a

clases desbalanceadas y su rendimiento en términos de precisión, *recall* y tasa de falsas alarmas.

4. Comparar los modelos más allá de los números: analizar también su interpretabilidad, su coste computacional y su viabilidad para integrarse en un sistema de seguridad real.

5. Extraer conclusiones sobre qué enfoques funcionan mejor en este contexto e identificar posibles mejoras o líneas de trabajo futuro.

1.5 ESTRUCTURA DEL DOCUMENTO

El documento se organiza en dos partes. La primera recoge el marco teórico: el capítulo 2 describe los principales tipos de ciberataques con ejemplos reales; el capítulo 3 introduce el Machine Learning y su aplicación a la detección de amenazas en redes; y el capítulo 4 explica en detalle los modelos utilizados en la parte experimental.

La segunda parte recoge el estudio experimental. El capítulo 5 describe el dataset, el preprocesado y la metodología común a todos los modelos. El capítulo 6 desarrolla el entrenamiento y la evaluación de cada modelo por separado. El capítulo 7 compara los resultados de forma conjunta y analiza qué modelo se comporta mejor en cada escenario. El capítulo 8 recoge las conclusiones y las posibles líneas de trabajo futuro.

El documento incluye también dos anexos: el primero recoge la alineación del proyecto con los Objetivos de Desarrollo Sostenible, y el segundo los recursos y el plan de trabajo seguido durante el desarrollo del proyecto.

Capítulo 2. ESTADO DEL ARTE

En 2024 se registraron más de 2.200 ciberataques al día a nivel global, lo que equivale a un incidente cada 39 segundos (Netwrix, 2024). No todos tienen el mismo impacto, pero el volumen ya da una idea de la escala del problema.

La ciberseguridad es el conjunto de prácticas, tecnologías y procesos orientados a proteger sistemas informático y redes frente a accesos no autorizados, ataques y daños. Durante décadas fue una disciplina aplicada sobre todo en entornos militares o grandes corporaciones. Pero esto ha cambiado radicalmente con la digitalización masiva: hoy casi cualquier organización depende de su infraestructura digital para operar y esa dependencia la expone a posibles ataques.

Las amenazas también han cambiado. Los primeros virus de los años ochenta circulaban en entornos académicos, más por curiosidad que por intención maliciosa. En los noventa, con la expansión de las redes e internet, empezaron a propagarse de forma masiva. Ya en los 2000 el dinero se convirtió en la motivación principal: phishing, spyware, troyanos bancarios. La siguiente década ya apareció la ciberdelincuencia organizada y los ataques organizados por estados. WannaCry, en 2017, afectó a más de 200.000 sistemas en 150 países en cuestión de horas (Netwrix, 2024). Desde entonces los ataques son más rápidos, más automatizados y más difíciles de identificar.

Para hacer frente a esta evolución se han ido desarrollando distintas capas de defensa. Los cortafuegos y los sistemas de control de acceso fueron los primeros. Después llegaron los antivirus, los sistemas de detección de intrusiones, las soluciones de gestión de eventos de seguridad (SIEM) y, más recientemente, las plataformas de respuesta automatizada. En la práctica, todo esto no funciona de forma aislada sino que la seguridad de una red depende de cómo se combinan y de quién las gestiona.

Hoy en día, la ciberseguridad se ha vuelto una prioridad. El volumen de tráfico de red crece, los ataques se vuelven más sofisticados y los equipos de seguridad no siempre tienen

capacidad para analizar manualmente todas las alertas que generan sus sistemas. Por ello, los enfoques basados en Machine Learning han ido ganando peso, primero como complemento a los sistemas de detección tradicionales y, últimamente, como componente central de algunas arquitecturas de seguridad.

2.1 AMENAZAS EN CIBERSEGURIDAD DE REDES

Antes de ver que técnicas existen actualmente para detectar ataques, conviene tener claro qué tipos de amenazas existen hoy en día y cómo funcionan. Las amenazas de red han evolucionado considerablemente desde los primeros virus de los años ochenta, y hoy abarcan desde ataques de denegación de servicio hasta técnicas de infiltración que pueden pasar desapercibidas durante semanas. En los apartados siguientes se describen las principales categorías conocidas con algunos ejemplos históricos que ilustran su impacto real.

2.1.1 INGENIERÍA SOCIAL

La ingeniería social no es un ataque técnico: es manipulación. El objetivo es persuadir a una persona para que revele información confidencial, comparta credenciales o dé acceso a un sistema, usando tácticas psicológicas como la urgencia, el miedo o la empatía (CrowdStrike, s.f.). En la práctica, muchos ataques complejos empiezan por aquí, porque es más fácil engañar a una persona que vulnerar un sistema bien configurado (Imperva, s.f.).

El proceso suele seguir tres pasos: investigar a la víctima para encontrar vulnerabilidades, establecer una relación de confianza y, finalmente, provocar una acción concreta como hacer clic en un enlace, descargar un archivo o revelar una contraseña. Lo que hace difícil defenderse de esto es que el componente explotado es humano, y eso no se puede solucionar con una actualización de software.

PHISHING

El phishing consiste en suplantar la identidad de una entidad legítima, un banco, una empresa, un servicio online, para engañar al usuario y obtener datos como contraseñas o

números de tarjeta. Los métodos más habituales son correos electrónicos fraudulentos, sitios web falsos que imitan a los reales y mensajes de texto engañosos (Kim & Solomon, 2021).

Dentro del phishing hay variantes con distintos niveles de sofisticación. El *phishing masivo* o *bulk email phishing* consiste en enviar correos a gran escala simulando provenir de empresas conocidas, con asuntos que apelan a la urgencia, "problema con su cuenta", "factura pendiente", para redirigir al usuario a un portal falso (IBM, s.f.). El *spear phishing* es más peligroso: está personalizado y se construye con información real sobre la víctima, frecuentemente obtenida de redes sociales. El *whaling* es una variante del anterior dirigida específicamente a altos ejecutivos con acceso a información sensible o capacidad para autorizar transferencias (IBM, s.f.; Fortinet, s.f.). En los últimos años también ha aparecido el phishing generado con inteligencia artificial, que produce mensajes más convincentes y sin los errores típicos que antes ayudaban a identificarlos (IBM, s.f.).

SPAM

El spam es el envío masivo de correos no solicitados, habitualmente con fines comerciales, aunque también se usa para distribuir malware o enlaces maliciosos (Cisco, s.f.). A diferencia del phishing, no busca un engaño directo sino la saturación o la promoción indiscriminada. Su impacto real en seguridad viene del consumo de recursos, ancho de banda, tiempo de CPU, atención del equipo técnico, que puede desviar esfuerzos de tareas más críticas (Kim & Solomon, 2021).

2.1.2 ATAQUES DE DENEGACIÓN DE SERVICIO (DOS/DDOS)

Un ataque de denegación de servicio (DoS) representa una de las modalidades más frecuentes y perturbadoras de agresión en el contexto de la ciberseguridad. El objetivo principal consiste en saturar los recursos de un sistema, tales como el ancho de banda, la memoria o la capacidad de procesamiento, hasta alcanzar un nivel que impida la respuesta a solicitudes legítimas de los usuarios. En la práctica, este tipo de ataque tiene como objetivo agotar de manera deliberada los recursos de la red o del servidor, lo que puede resultar en la interrupción del servicio y, en numerosas ocasiones, en su colapso total. De acuerdo con

Kurose y Ross, la creciente interconexión de sistemas y dispositivos en Internet ha incrementado la vulnerabilidad de las redes a diversas amenazas (Kurose & Ross, 2017).

La evolución del ataque de denegación de servicio (DoS) ha conducido al desarrollo de una modalidad más sofisticada, denominada ataque de denegación de servicio distribuido (DDoS). En este contexto, el agresor utiliza un conjunto extenso de dispositivos infectados, denominados botnets, que operan de manera coordinada para generar un volumen significativo de tráfico dirigido hacia el objetivo. La distribución de las fuentes de ataque contribuye a que los ataques DDoS sean más efectivos, complicando su detección y mitigación. Esto se debe a que el volumen total de solicitudes falsas puede exceder con facilidad la capacidad de respuesta del servidor (Fortinet, 2024; Cloudflare, 2024).

Desde una perspectiva técnica, los ataques de Denegación de Servicio pueden manifestarse a través de múltiples métodos:

Una de las modalidades más comunes consiste en la explotación de vulnerabilidades en software o redes a través del envío de paquetes diseñados de manera específica, lo que provoca que el sistema tenga comportamientos anómalos, tales como bloqueos o reinicios inesperados. Un ejemplo de ataque basado en la explotación de vulnerabilidades es el denominado Ping of Death. Este ataque implica el envío de paquetes ICMP malformados, tales como aquellos que presentan fragmentación incorrecta o que superan los límites de tamaño permitidos. El propósito de esta acción es inducir un desbordamiento de buffers en el sistema objetivo. El resultado puede manifestarse en forma de bloqueo del servicio, de la memoria o reinicio inesperado del equipo.

Por otro lado, tenemos la inundación de ancho de banda, en la cual el atacante envía una gran cantidad de paquetes. Esta acción provoca la congestión del canal de comunicación, lo que impide que los paquetes legítimos alcancen su destino. Un ejemplo es el ataque Smurf, el cual se lleva a cabo en la capa de red y se basa en la explotación del Protocolo de Mensajes de Control de Internet (ICMP). El funcionamiento de este ataque se fundamenta en la incrementación del tráfico. En esta situación, el atacante envía paquetes ICMP de tipo ping a la dirección de difusión de una red, empleando una dirección IP falsificada que corresponde

a la víctima. Como resultado, todos los dispositivos de la red responden de manera simultánea al equipo suplantado, lo que genera una gran cantidad de tráfico de retorno que satura el ancho de banda y provoca la inoperatividad del sistema objetivo (Fortinet, 2025).

Por último, tenemos el fenómeno conocido como inundación de conexiones se caracteriza por la apertura simultánea de un elevado número de conexiones TCP, lo que puede resultar en la saturación del servidor y en su incapacidad para aceptar nuevas solicitudes (Kim & Solomon, 2016; Fortinet, 2024). Un ejemplo representativo de este tipo de ataque es la inundación SYN. Para comprender este ataque, resulta esencial revisar el proceso de establecimiento de una conexión TCP, el cual se desarrolla en tres etapas: en primer lugar, el cliente envía un paquete SYN para solicitar la conexión; en segundo lugar, el servidor responde con un paquete SYN/ACK, reconociendo la solicitud; y, finalmente, el cliente envía un paquete ACK que completa el enlace y permite el intercambio de datos. Durante este proceso, el servidor debe reservar temporalmente recursos para anticipar la recepción del último paquete, como puertos abiertos en espera de una respuesta. En este tipo de ataque, el atacante envía una cantidad considerable de paquetes SYN, frecuentemente utilizando direcciones IP falsificadas, sin completar la secuencia con el ACK final. El servidor, al sostener miles de "conexiones medio abiertas", agota sus recursos, lo que resulta en la incapacidad de responder al tráfico legítimo y provoca una denegación de servicio. Para mitigar este fenómeno, se implementan técnicas tales como la reducción de los tiempos de espera y el filtrado de paquetes que se consideran sospechosos (Cloudflare, 2024; Fortinet, 2025).

A diferencia de otros tipos de ciberataques que tienen como objetivo el robo de información o la obtención de acceso privilegiado, los de denegación de servicio se caracterizan por su intención de interrumpir la disponibilidad del sistema, en lugar de infiltrarse en él. En consecuencia, su impacto puede manifestarse en dimensiones tanto técnicas como económicas. Un competidor podría, por ejemplo, llevar a cabo este tipo de ofensivas con el objetivo de inhabilitar temporalmente los servicios de una empresa rival, lo que podría resultar en pérdidas económicas y un deterioro de su reputación (Fortinet, 2024).

Un caso significativo ocurrió en octubre de 2024, dirigido hacia Cloudflare, una empresa especializada en seguridad y distribución de contenidos. Se informó que se logró mitigar un ataque con una magnitud de 5,6 terabits por segundo (Tbps), lo cual representa un registro sin precedentes en términos de tráfico malicioso (Cloudflare, 2024).

El ataque fue ejecutado mediante una botnet. El objetivo del ataque fue un proveedor de servicios de Internet que se encuentra en Asia oriental. A diferencia de las botnets clásicas que se basan en dispositivos IoT (por ejemplo, routers o cámaras domésticas), esta variante utilizaba servidores cloud y sistemas corporativos vulnerables, lo que aumentó de manera exponencial su capacidad para generar tráfico. De acuerdo con el informe técnico de Cloudflare, el ataque utilizó varias técnicas de saturación, incluyendo el SYN flood y el HTTP/2 rapid reset attack. Este último es una vulnerabilidad reciente del protocolo HTTP/2 que posibilita la generación de miles de peticiones seguidas por cancelaciones inmediatas, lo cual produce una sobrecarga en los servidores web (Akamai, 2024).

El efecto de este ataque fue limitado desde el punto de vista operativo, porque se neutralizó antes de causar interrupciones a gran escala. Sin embargo, su importancia en términos estratégicos es inmensa pues marcó el comienzo de una nueva generación de ataques DDoS en el que se utilizan las debilidades de los protocolos modernos.

En definitiva, el ataque DDoS de 2024 demostró que, incluso sin causar daños directos, un solo evento puede revelar la fragilidad de Internet. Este ataque, al igual que WannaCry en el campo del ransomware, del cual hablaremos más adelante, es un punto de inflexión para la seguridad de la red mundial.

2.1.3 MALWARE

El malware, término derivado de "malicious software", se refiere a cualquier programa o fragmento de código informático que ha sido diseñado con el propósito de dañar, modificar o asumir el control de un sistema. De acuerdo con Kim y Solomon (2021), el malware abarca desde virus simples que infectan archivos hasta programas complejos que tienen la capacidad de infiltrarse en redes, sustraer información o sabotear infraestructuras críticas.

Este tipo de ataques tienen numerosas modalidades, tales como virus, troyanos, spyware, adware y ransomware, entre otros. Su objetivo común radica en la obtención de beneficios económicos, políticos o estratégicos. Según IBM (s.f.), la mayoría de los ciberataques contemporáneos incorporan algún tipo de malware, y todos los sistemas operativos, incluyendo Windows, macOS, iOS y Android, presentan vulnerabilidades.

Una vez que un equipo es infectado, los programas maliciosos se propagan a otros dispositivos mediante redes, correos electrónicos o unidades extraíbles, lo que resulta en una multiplicación exponencial de su alcance (Malwarebytes, s.f.). Las infecciones afectan las tres propiedades fundamentales de la seguridad de la información: la confidencialidad, al exponer información privada; la integridad, al alterar o eliminar datos; y la disponibilidad, al obstaculizar el acceso legítimo a los recursos del sistema. Las empresas se han convertido en los objetivos clave debido a su manejo de volúmenes significativos de información personal y sensible.

VIRUS:

La presencia de virus informáticos es un fenómeno ampliamente conocido en el ámbito de la tecnología. En el lenguaje cotidiano, es común atribuir cualquier problema técnico a la presencia de un "virus". Sin embargo, es importante señalar que existen diversas formas de malware, siendo el virus únicamente una de estas categorías. En un sentido estricto, se define como un tipo de malware que se infiltra en un sistema con el objetivo de modificar su funcionamiento sin el consentimiento del usuario. De acuerdo con Kim y Solomon (2021), dichos programas se asocian a archivos ejecutables replicándose y propagándose en el momento en que el usuario ejecuta la aplicación infectada. El funcionamiento de este mecanismo se fundamenta en un proceso sencillo. Una vez activado, el código del virus se integra en la memoria del equipo y asume el control de los servicios esenciales del sistema operativo. Posteriormente, procede a infectar otros archivos, replicando así el proceso de infección. Según lo expuesto por Fortinet (s.f.), los virus informáticos pueden provocar desde interrupciones menores en el rendimiento hasta consecuencias más severas, tales como la pérdida de datos, la corrupción de archivos o el colapso total del sistema.

Los virus presentan una variedad de formas y mecanismos de infección. Existen diversas categorías de infecciones informáticas. Los "system infectors" se dirigen al hardware o a los procesos de arranque del sistema. Por su parte, los "file infectors" modifican programas ejecutables" en formatos tales como COM o EXE. Finalmente, los "data infectors" también conocidos como “macro infectors”, impactan documentos que contienen macros incrustadas, siendo especialmente relevantes en entornos como Microsoft Windows (Kim & Solomon, 2021). A lo largo del tiempo, han emergido variantes más complejas, tales como los virus multipartitos, los cuales utilizan múltiples métodos de propagación y tienen la capacidad de residir tanto en la memoria como en el disco duro.

Desde una perspectiva histórica, el concepto de virus informático se origina en la década de 1970, cuando programas experimentales, como Creeper, comenzaron a infectar los primeros sistemas IBM. Este programa era conocido por mostrar mensajes tales como “I’m the creeper… catch me if you can!” (Wikipedia, s.f.). No obstante, los virus informáticos comenzaron a tener un impacto significativo en la década de 1980 con el aumento en la utilización de disquetes (Fortinet, s.f.). La evolución de estas amenazas muestra que los virus continúan siendo una de las manifestaciones más persistentes y adaptables del malware pues pueden alterar la estabilidad de sistemas completos con la simple acción de un clic por parte del usuario.

ROOTKITS:

Dentro del ámbito de las amenazas digitales, los rootkits representan una categoría de malware que tiene como objetivo proporcionar al atacante privilegios de administrador, al tiempo que oculta su presencia en el sistema. El término "rootkit" se compone de las palabras "root", que en sistemas operativos como Unix o Linux se refiere al usuario con control total, y "kit", que alude a un conjunto de herramientas de software (Malwarebytes, s.f.). De esta forma, un rootkit se define como un conjunto de programas que permite al atacante alterar procesos del sistema, manipular archivos o desactivar mecanismos de seguridad sin ser identificado.

A diferencia de un virus tradicional, el rootkit no se limita a la corrupción de archivos, sino que tiene como objetivo infiltrarse en las capas más profundas del sistema operativo con el fin de asegurar un control persistente sobre el dispositivo. CrowdStrike (s.f.) enfatiza que la detección y eliminación de estas amenazas resulta particularmente compleja, dado que están diseñadas para operar de manera invisible, incluso ante herramientas de seguridad avanzadas. En el contexto de la ciberseguridad, es relevante mencionar que, durante mi experiencia en Telefónica, se implementó una alerta en el Sistema de Gestión de Eventos e Información de Seguridad (SIEM) que se activaba al agregar a un usuario a las listas de administradores sin notificación previa para poder prevenir este tipo de ataques.

En conclusión, la capacidad de camuflaje y control de estos ataques los posiciona como una de las amenazas más sofisticadas y complejas de erradicar en ciberseguridad. (Malwarebytes s.f)

TROYANOS:

Los troyanos, o Trojan horses, representan una de las manifestaciones más comunes de malware en el contexto actual. El término, derivado de la mitología griega, hace referencia al célebre caballo de madera que facilitó la infiltración de los griegos en la ciudad de Troya. De manera análoga, en el ámbito de la informática, un troyano se presenta como un programa legítimo con el propósito de engañar al usuario y llevar a cabo acciones maliciosas sin su conocimiento (Wikipedia, s.f.). Esto incluye la posibilidad de redirigir el tráfico, sustraer datos sensibles, eliminar archivos a instalar nuevas cargas maliciosas. De acuerdo con Kurose y Ross (2021), estos programas se caracterizan por su presentación como aplicaciones útiles, mientras que en realidad ocultan una intención maliciosa. La propagación de dichos programas depende en gran medida de técnicas de ingeniería social, las cuales inducen a las víctimas a descargar archivos o enlaces fraudulentos provenientes de correos electrónicos, anuncios o redes sociales.

A diferencia de los virus o gusanos, los troyanos no poseen la capacidad de replicarse de manera autónoma. En su lugar, requieren una acción deliberada por parte del usuario para su instalación

WORMS:

Los gusanos informáticos constituyen una categoría de malware que tiene la capacidad de propagarse de manera automática, sin requerir intervención humana. Este tipo de software malicioso se replica a sí mismo y se extiende a otros dispositivos conectados a una red local o a Internet (CrowdStrike, s.f.). A diferencia de un virus, que debe adjuntarse a un archivo ejecutable y requiere la activación por parte del usuario, y de un troyano, que se presenta como software legítimo para engañar al usuario, el gusano tiene la capacidad de desplazarse y operar de manera independiente. Este tipo de malware puede replicarse y enviar cientos o miles de copias de sí mismo a otros sistemas vulnerables.

Según Kim y Solomon (2021), estos programas requieren un uso intensivo de los recursos del sistema, tales como el ancho de banda, la memoria y el tiempo de CPU. Esta demanda elevada puede resultar en una disminución del rendimiento de las redes. Asimismo, los worms tienden a explotar vulnerabilidades en el sistema operativo para llevar a cabo su infiltración y replicación. Una vez que se han instalado, estos programas pueden llevar a cabo diversas acciones maliciosas, que incluyen el robo de información, la eliminación de archivos, la implementación de otros tipos de malware y la ejecución de ataques distribuidos de denegación de servicio (DDoS). En ciertos casos, se recurre a sistemas intermedios comprometidos para la reenvío de mensajes o la distribución de correo no deseado. Además, se pueden utilizar mecanismos que aparentan ser inocentes, tales como enlaces de “cancelar suscripción” en correos electrónicos, con el propósito de identificar direcciones válidas y, de este modo, ampliar la red de infección (Kim & Solomon, 2021).

Debido a su autonomía y a la velocidad con la que se propagan, los worms constituyen una amenaza particularmente significativa para las organizaciones, dado que tienen la capacidad de colapsar infraestructuras completas en un lapso de tiempo muy breve. Además de los daños directos, el impacto se manifiesta en la saturación de redes, la disminución de la productividad y el efecto multiplicador que estos incidentes ejercen como facilitadores de otros ataques cibernéticos.

RANSOMWARE

El ransomware se considera una de las amenazas cibernéticas más significativas en los últimos años. A diferencia de otros tipos de malware, este tipo de software malicioso no se restringe únicamente a causar daños en el sistema o a sustraer datos. En cambio, "mantiene los archivos o el propio dispositivo como rehenes, amenazando con mantenerlos bloqueados a menos que la víctima pague un rescate" (IBM, s.f.)". Es un modelo de extorsión digital pues el usuario se ve privado del acceso a sus archivos, mientras que el atacante solicita un pago para restablecer el control sobre dicha información. De acuerdo con Kim y Solomon (2021), el ransomware tiene como objetivo generar beneficios económicos de forma directa al limitar la capacidad del usuario para acceder a sus datos. Esto se logra comúnmente a través de técnicas de cifrado que hacen que los documentos, carpetas o incluso el disco completo sean inaccesibles. La eficacia de este fenómeno se fundamenta en el impacto psicológico que genera. Un breve lapso de tiempo es suficiente para que el usuario experimente la pérdida de años de información, lo que lo lleva a enfrentar el dilema de optar por realizar un pago o arriesgarse a perderlo todo.

Los orígenes del ransomware se sitúan en la década de 1980; sin embargo, su expansión masiva no se produjo hasta principios de los años 2000, coincidiendo con el auge de las criptomonedas, las cuales facilitaron el pago anónimo de rescates (Fortinet, s.f.). Uno de los primeros programas ampliamente documentados en el ámbito del software malicioso fue CryptoLocker, el cual, en el año 2013, contribuyó a la difusión del modelo de cifrado de archivos, conocido como cryptoware. Este programa se caracterizaba por la capacidad de bloquear documentos y discos completos hasta que la víctima realizara el pago correspondiente.

WANNACRY:

El ataque WannaCry, que tuvo lugar en mayo de 2017, se considera uno de los episodios más relevantes en la historia contemporánea del ransomware. Este programa malicioso, que integraba funciones de cifrado y propagación automática, logró expandirse de manera rápida a través de una vulnerabilidad crítica del sistema operativo Windows, identificada como SMBv1 (EternalBlue). Esta vulnerabilidad, que se manifiesta como un fallo en el protocolo

destinado para la transmisión de archivos entre ordenadores, había sido explotada mediante un "exploit· desarrollado por la Agencia de Seguridad Nacional de Estados Unidos (NSA). Este "exploit" fue posteriormente filtrado por un grupo de ciberdelincuentes conocido como Shadow Brokers. A pesar de que Microsoft emitió un parche de seguridad (MS17-010) semanas antes del ataque, numerosas organizaciones no implementaron dicha actualización a tiempo, lo que causó la rápida propagación del malware (Cloudflare, s.f.; Kaspersky, s.f.).

WannaCry se propagaba de manera automática, sin requerir intervención humana, lo que lo distingue como un caso particular de gusano-ransomware. Una vez que el equipo era infectado, se procedía al cifrado de los archivos, y se exigía el pago de un rescate de 300 dólares en Bitcoin, importe que se incrementaba si la víctima no cumplía con el plazo establecido para el pago. El malware incluía un componente denominado dropper, el cual es un programa de dimensiones reducidas diseñado para instalar el resto del código malicioso. La propagación del fenómeno se produjo con tal rapidez que, en un lapso de pocas horas, impactó a más de 200,000 equipos en 150 países. Este evento afectó a diversas entidades, incluyendo hospitales del Servicio Nacional de Salud británico (NHS), así como a empresas como Telefónica, FedEx y Nissan, además de numerosas instituciones públicas (Akamai, s.f.).

La expansión global de WannaCry se detuvo cuando el investigador Marcus Hutchins identificó un mecanismo oculto en su código, denominado "kill switch" que permitía desactivar el malware.

El impacto económico y operativo resultó ser significativo, con estimaciones de pérdidas globales que oscilan entre 4.000 y 8.000 millones de dólares. Sectores fundamentales, tales como la sanidad, el transporte y las telecomunicaciones, experimentaron interrupciones severas. En el Reino Unido, el Servicio Nacional de Salud (NHS) se vio obligado a cancelar miles de citas médicas y a redirigir ambulancias hacia otros hospitales. Posteriormente, tanto Estados Unidos como el Reino Unido identificaron al grupo Lazarus, asociado al gobierno de Corea del Norte, como responsable del ataque. Sin embargo, esta atribución ha sido objeto de cuestionamiento por parte de algunos expertos (Netwrix, s.f.; Kaspersky, s.f.).

Este ataque proporciona grandes lecciones para la ciberseguridad contemporánea:

- En primer lugar, es fundamental mantener los sistemas actualizados y aplicar los parches de seguridad de manera rápida, dado que la falta de actualización constituyó el principal factor que facilitó la propagación de WannaCry.

- Asimismo, es necesario segmentar las redes con el objetivo de restringir la propagación del malware.

- Otras medidas complementarias comprenden la implementación de autenticación multifactor (MFA).

- También se incluyen soluciones de detección y respuesta en endpoints (EDR) y la formación constante de los empleados, con el objetivo de prevenir infecciones originadas por errores humanos.

En conclusión, WannaCry evidenció que una vulnerabilidad previamente identificada puede dar lugar a un ataque global por una falta de prevención y disciplina organizativa.

SPYWARE:

El spyware es otro tipo de malware, caracterizado por su habilidad para infiltrarse de manera discreta en un dispositivo con el propósito de recopilar información del usuario sin su consentimiento. Este tipo de software opera mediante la ejecución de procesos en segundo plano que registran los hábitos de navegación, las actividades en línea, recopilan credenciales, datos financieros… lo que impacta de manera directa en la privacidad y la confidencialidad de los datos. Algunos rastreadores se manifiestan como cookies persistentes que facilitan el intercambio de información entre diferentes sitios web. Sin embargo, las versiones más sofisticadas poseen la capacidad de registrar pulsaciones de teclado, realizar capturas de pantalla e incluso instalar software adicional sin el conocimiento del usuario.

Palo Alto Networks (s.f.) indica que el spyware puede infiltrarse a través de anuncios engañosos, descargas disfrazadas o mensajes fraudulentos. Una vez instalado, el spyware se desarrolla a través de un proceso característico que consta de tres fases: infiltración, monitorización y transmisión de datos sustraídos. Posteriormente, estos datos pueden ser transmitidos a terceros , incluidos actores maliciosos, quienes podrían comercializarlos o utilizarlos con fines perjudiciales, como la suplantación de identidad o la perpetración de nuevos ataques

En conclusión, el spyware, debido a su naturaleza silenciosa y persistente, constituye una doble amenaza. Por un lado, infringe la intimidad del usuario y por otro, transforma el dispositivo del individuo en una fuente de información susceptible de explotación.

ADWARE:

El adware, acrónimo de software soportado por publicidad, constituye una categoría de programas que, a pesar de su apariencia inofensiva, puede transformarse en una de las manifestaciones más intrusivas y persistentes de malware. La función principal de estos elementos es la exhibición de anuncios emergentes o redirecciones automáticas durante la navegación por Internet, lo que repercute en la productividad y el rendimiento del sistema (Kim & Solomon, 2021). De acuerdo con Malwarebytes (2024), el adware se instala de manera encubierta junto con otros programas, particularmente aquellos que son gratuitos o de origen cuestionable. Este tipo de software presenta publicidad no solicitada y, con frecuencia, realiza un seguimiento del comportamiento en línea del usuario con el fin de generar anuncios personalizados.

CrowdStrike (s.f.) indica que, a diferencia de los virus o troyanos, el adware no tiene como objetivo la destrucción del sistema, sino que persigue la obtención de beneficios económicos mediante la exposición o el clic en anuncios. El modelo de ingresos se fundamenta en estrategias de pago por visualización (pay-per-view) o pago por clic (pay-per-click), lo que puede motivar a ciertos desarrolladores a incorporarlo en programas que, a primera vista, parecen legítimos. Existen, no obstante, versiones más agresivas, denominadas Potentially Unwanted Programs (PUPs) o adware malicioso, que tienen la capacidad de secuestrar el

navegador, modificar la página de inicio o instalar barras de herramientas y extensiones sin el consentimiento del usuario (CrowdStrike, s.f.; Malwarebytes, 2024).

En conclusión, el adware se sitúa en una zona intermedia entre la publicidad digital legítima y el malware. Aunque su impacto no es necesariamente destructivo, la presencia continua de este tipo de software puede empeorar la experiencia de navegación y comprometer la privacidad del usuario.

EJEMPLO DE MALWARE

A continuación, analizaremos una serie de ataques cibernéticos que tuvieron lugar durante la guerra en Ucrania. Estos proporcionan una ilustración significativa de cómo el malware puede convertirse en un instrumento estratégico en conflictos reales. Este ejemplo muestra la combinación de acciones físicas y digitales en un mismo contexto bélico.

En el sistema eléctrico de Ucrania, se llevaron a cabo una serie de ataques estratégicamente planificados por parte de Rusia, con el objetivo de impactar de manera directa la infraestructura física. Esta clase de ataque se incluye en la categoría de ataques ciberfísicos, es decir, aquellos en los que el malware no solo pone en peligro sistemas informáticos, sino que también manipula equipos industriales tangibles. El elemento técnico fundamental en estos ataques fue el malware Industroyer, que fue identificado por primera vez en 2016, así como su versión más avanzada, Industroyer2, que fue implementada en 2022. Los dos programas fueron diseñados con el propósito de infiltrarse en la red de tecnología operativa, la cual regula los procesos físicos de una instalación, tales como interruptores y transformadores. Esta infiltración se lleva a cabo a través del sistema de supervisión y control industrial (SCADA), que funciona como una interfaz entre los operadores humanos y los dispositivos eléctricos.

Industroyer operaban mediante el envío de órdenes fraudulentas a los equipos de la red eléctrica. En la práctica, se lograba la desconexión remota y simultánea de circuitos, lo que resultaba en interrupciones del suministro en diversas regiones del país. Este tipo de malware se clasifica como destructivo de infraestructura crítica, dado que su objetivo principal no es

la sustracción de información, sino la alteración de la funcionalidad de los sistemas industriales, lo que puede llevar a la interrupción del servicio.

A finales de 2022, el grupo Sandworm, asociado con la inteligencia militar rusa (GRU), llevó a cabo una operación que, aunque similar a anteriores, presentó un nivel de sofisticación superior. En lugar de implementar un programa nuevo, se optó por utilizar una versión obsoleta del sistema MicroSCADA, el cual es una herramienta reconocida para la gestión de estaciones eléctricas. Esta decisión permitió el envío de instrucciones reales desde el propio sistema, facilitando la apertura de interruptores eléctricos. Posteriormente, los atacantes implementaron un wiper conocido como CaddyWiper, el cual se caracteriza por su capacidad para eliminar archivos y registros. Esta acción complica tanto la recuperación del sistema como la investigación forense (Google/Mandiant, 2023) pues se borraron los rastros.

Desde una perspectiva técnica y estratégica, este ataque representa una manifestación evidente de la ciberguerra contemporánea, en la cual los conflictos ya no se limitan al terreno físico. En efecto, los cortes eléctricos registrados coincidieron con bombardeos, lo que evidencia la integración de la ofensiva militar rusa, que combinó ataques terrestres con operaciones cibernéticas sincronizadas. Esto muestra como en la guerra contemporánea, las redes eléctricas, las comunicaciones por satélite y los sistemas industriales emergen como frentes de batalla de igual relevancia que el propio campo físico (NCSC, 2022; Google/Mandiant, 2023).

2.1.4 INYECCIONES Y ATAQUES EN LA CAPA DE APLICACIÓN:

Las amenazas más críticas para las aplicaciones web incluyen los ataques de inyección. Estos permiten a los atacantes insertar código malicioso en los procesos normales de una aplicación, lo que puede alterar su comportamiento o facilitar la ejecución de acciones no autorizadas. De acuerdo con Security Journey (2024), los ataques en cuestión se basan en la manipulación de datos de entrada que la aplicación no comprueba de manera adecuada. Esta falta de validación puede llevar a que el sistema interprete comandos o consultas que han sido inyectados por el atacante. El impacto de dicha vulnerabilidad puede abarcar desde la

divulgación de información sensible hasta la ejecución remota de código, así como el control total del sistema afectado.

En la práctica, los atacantes primero identifican los puntos de entrada de datos en una aplicación, tales como parámetros, cabeceras o cargas de archivos, con el propósito de inyectar cargas maliciosas. Estas pueden consistir en fragmentos de código, tales como HTML/JS o sentencias SQL y tienen como objetivo alterar el flujo de datos al ser procesadas o ejecutadas por el sistema.

Un ejemplo ampliamente reconocido es la inyección SQL dirigida a bases de datos. Este tipo de ataque permite el acceso, la modificación o la eliminación de información crítica a través de la inserción de consultas SQL maliciosas en formularios o parámetros web (ManageEngine, 2024). Como resultado, puede comprometer información sensible, incluyendo datos financieros, registros de clientes y credenciales de acceso.

También cabe destacar los ataques de Cross-Site Scripting (XSS), clasificados como inyecciones de código. Tienen como objetivo la introducción de scripts maliciosos en los navegadores de usuarios legítimos. Esta técnica permite la sustracción de cookies, el secuestro de sesiones y la descarga de software malicioso (ManageEngine, 2024).

Además de estas dos, se identifican otras variantes que presentan un nivel de peligro comparable como la inyección de comandos (Command Injection). Se insertan comandos del sistema operativo en una aplicación que presenta vulnerabilidades, con el objetivo de que sean ejecutados y obtener privilegios elevados o acceso remoto (Invicti, 2024).

HOMEPAGE HIJACKING

El secuestro de la página de inicio, conocido como *homepage hijacking*, se refiere a la explotación de vulnerabilidades en los navegadores o a la instalación encubierta de programas auxiliares maliciosos. Este fenómeno tiene como objetivo modificar la configuración predeterminada del navegador sin el consentimiento del usuario. De acuerdo con Malwarebytes (2024), los programas denominados "browser hijackers" modifican la página de inicio o el motor de búsqueda predeterminado, redirigen el tráfico hacia sitios que

pueden ser potencialmente maliciosos e inyectan anuncios no solicitados, desempeñando en muchos casos funciones similares a las del adware. Ciertos secuestradores de navegador han sido diseñados para incluir keyloggers, los cuales tienen la capacidad de registrar las pulsaciones de teclado y, de este modo, obtener credenciales o información sensible introducida por el usuario. Generalmente, estos elementos se difunden a través de barras de herramientas o extensiones que se incluyen en software gratuito descargado de sitios de terceros. Asimismo, pueden ser instalados mediante código malicioso incorporado en páginas web o en ventanas emergentes.

Por otro lado, diversos ataques se aprovechan de los servicios de la capa de aplicación, tales como DNS, HTTP o APIs, con el objetivo de redirigir tráfico, inyectar contenido o sustraer credenciales. En cuanto a vulnerabilidades de los servicios de DNS tenemos el envenenamiento de caché. Este se define como el proceso mediante el cual se introducen respuestas DNS fraudulentas en la caché de un resolutor. De esta forma, las consultas devuelvan direcciones IP incorrectas, lo que puede resultar en que los usuarios accedan a sitios web falsos (Cloudflare, s.f.).

Desde un enfoque técnico, el uso del protocolo UDP facilita el ataque porque, al no existir un mecanismo de conexión ni autenticación previa, el resolutor DNS puede aceptar respuestas fraudulentas. Estas respuestas quedan almacenadas en caché y se consideran válidas hasta que expira el tiempo de vida (TTL) asociado. El impacto se manifiesta de manera directa, dado que los usuarios son redirigidos a páginas de phishing, malware o servicios controlados por el atacante.

Podemos concluir que las redes y aplicaciones corporativas presentan numerosas vulnerabilidades que de no ser gestionadas de manera oportuna, pueden ser explotadas por actores maliciosos en busca de cualquier fallo en la seguridad.

CASO EQUIFAX

Un caso representativo de esta problemática fue el ataque a Equifax en 2017, el cual es considerado uno de los incidentes de ciberseguridad más significativos de la historia

reciente. En este contexto, los atacantes lograron infiltrarse en los sistemas de la compañía mediante la explotación de una vulnerabilidad conocida en Apache Struts (CVE-2017-5638), un marco de trabajo ampliamente utilizado para el desarrollo de aplicaciones web empresariales. El error facilitaba la inyección de código remoto a través del encabezado HTTP, lo que permitió la ejecución de comandos en los servidores de Equifax. A pesar de que el parche correspondiente había sido publicado semanas antes, la empresa no lo implementó de manera oportuna, lo que facilitó la intrusión (Breachsense, 2024; House Committee on Oversight and Government Reform, 2018).

Una vez obtenido el acceso inicial, localizaron credenciales en texto plano que les otorgaron la capacidad de acceder a diversos servidores y bases de datos. Durante un período superior a dos meses, se llevó a cabo la extracción de información confidencial de millones de ciudadanos sin que dicha actividad fuera detectada. Este fenómeno se produjo debido a un mal funcionamiento de las herramientas de monitorización de tráfico, ya que un certificado TLS caducado imposibilitó la inspección de los flujos de datos cifrados. El 29 de julio de 2017, se llevó a cabo la renovación del certificado, momento en el cual los administradores identificaron actividad sospechosa y confirmaron la existencia de una brecha de seguridad. Esta situación fue divulgada públicamente en septiembre de ese mismo año (Breachsense, 2024; FBI, 2020).

El ataque comprometió los datos personales de aproximadamente 148 millones de individuos en Estados Unidos, abarcando información como nombres, direcciones, fechas de nacimiento, números de la Seguridad Social y, en ciertos casos, licencias de conducir o tarjetas de crédito. Asimismo, se vieron afectados miles de ciudadanos en Canadá y el Reino Unido.

El Departamento de Justicia de los Estados Unidos posteriormente atribuyó la operación a integrantes del Ejército Popular de Liberación de China, indicando que el propósito podría haber sido la recopilación de información personal con el fin de desarrollar perfiles de inteligencia sobre funcionarios y ciudadanos estadounidenses (FBI, 2020; EPIC, s.f.).

En la actualidad, los datos personales constituyen uno de los activos más valorados y, en consecuencia, se encuentran bajo una protección rigurosa. Por lo tanto, una brecha en la seguridad de estos datos puede acarrear consecuencias significativas. En efecto, este caso en un claro ejemplo de las graves consecuencias que tiene el robo de datos personales: la compañía se vio sujeta a multas y compensaciones que ascendieron a 1.380 millones de dólares, además de experimentar una pérdida considerable en términos de reputación y confianza pública. El acuerdo alcanzado con la Comisión Federal de Comercio (FTC) contempla una compensación de hasta 425 millones de dólares destinada a los individuos afectados. Como consecuencia del incidente, se promovieron nuevas normativas en materia de protección de datos.

El caso de Equifax proporciona lecciones fundamentales en el ámbito de la ciberseguridad. La gestión oportuna de parches, la supervisión del tráfico cifrado, la protección adecuada de credenciales y la renovación de certificados deben ser considerados no como meras formalidades técnicas, sino como pilares esenciales de la seguridad digital.

2.1.5 AUTENTICACIÓN Y ABUSO DE CREDENCIALES

ATAQUES DE FUERZA BRUTA:

Un ataque de fuerza bruta constituye un método de prueba y error que busca adivinar credenciales, contraseñas o claves de cifrado mediante la sistemática combinación de opciones hasta identificar la correcta. Este proceso frecuentemente se automatiza a través de bots o scripts que dirigen sus esfuerzos hacia formularios de inicio de sesión, interfaces de programación de aplicaciones (APIs), claves SSH o claves API (Cloudflare, s.f.; Kaspersky, s.f.).

Existen numerosas versiones como el ataque de diccionario. Este consiste en probar palabras de una lista, así como sus permutaciones. Por otro lado, el ataque híbrido combina listas con reglas específicas, como la adición de números o símbolos. También existe el ataque de fuerza bruta inversa, este parte de una contraseña conocida y prueba múltiples usuarios en relación con esta. Finalmente, el credential stuffing implica la reutilización de

pares de usuario y contraseña que han sido filtrados en brechas de seguridad anteriores, con el objetivo de intentar acceder a diferentes sitios. (TechTarget, 2024; Kaspersky, s.f.).

La simplicidad y escalabilidad de estos ataques constituyen una ventaja significativa, dado que, con el tiempo adecuado y en ausencia de contramedidas, es posible que siempre resulten efectivos. Sin embargo, su principal limitación radica en el tiempo necesario para llevar a cabo el ataque frente a contraseñas que sean largas y aleatorias (Kaspersky, s.f.).

Las medidas a tomar para evitar esta amenaza incluyen la implementación de políticas de contraseñas robustas, bloqueo progresivo o autenticación multifactor (MFA) o la detección de patrones de bots en la capa de autenticación (Cloudflare, s.f.).

ACTIVE DIRECTORY

Los ataques cibernéticos presentan una evolución continua. Además de los ataques tradicionales dirigidos a contraseñas o bases de datos, se identifican múltiples amenazas que impactan infraestructuras críticas, como Active Directory, el sistema encargado de gestionar la autenticación y el acceso de usuarios en el contexto de una red corporativa. Un ejemplo de ataque cibernético es el denominado *Kerberoasting*, que aprovecha el funcionamiento del protocolo Kerberos en entornos Windows. En este tipo de ataque, un usuario que ya dispone de acceso legítimo a la red solicita tickets de autenticación asociados a servicios específicos. Estos tickets están cifrados utilizando la contraseña de la cuenta de servicio correspondiente. Posteriormente, el atacante extrae dichos tickets y realiza un proceso de descifrado de forma offline con el objetivo de obtener la contraseña de la cuenta de servicio. Dado que estas cuentas suelen contar con privilegios elevados, su compromiso puede permitir al atacante desplazarse lateralmente por la red y llegar a comprometer todo el entorno (CrowdStrike, s.f.).

OCULTAR IP

El uso de técnicas como el *IP spoofing*, junto con la utilización de máquinas virtuales alojadas en plataformas en la nube de proveedores reconocidos, como Google Cloud o AWS, permite a los atacantes ocultar su verdadera identidad. Al actuar desde infraestructuras que

aparentan ser legítimas, el tráfico generado resulta más difícil de identificar como malicioso, lo que aumenta la capacidad del atacante para evadir los mecanismos de detección.

Es difícil proporcionar una lista exhaustiva de los tipos de ataques existentes, dado que continuamente emergen nuevas variantes. Un caso particularmente notable, no tanto por su magnitud, sino por su originalidad, es un incidente que tuvo lugar en septiembre de 2025. Este evento ilustra la considerable creatividad de estrategias que los ciberdelincuentes emplean en la actualidad.

El episodio se detalla en el artículo titulado "Paquetes maliciosos de npm explotan contratos inteligentes de Ethereum para atacar a desarrolladores de criptomonedas", publicado en The Hacker News. El informe presenta un análisis sobre el descubrimiento de dos paquetes maliciosos de npm que empleaban la infraestructura de los contratos inteligentes de Ethereum. Estos contratos son programas automatizados que operan dentro de la cadena de bloques para gestionar transacciones sin la necesidad de intermediarios. En este caso fueron utilizados con el propósito de ocultar y distribuir malware.

La campaña abarcó no solo el desarrollo de los paquetes mencionados, sino que también incluyó una estrategia de ingeniería social. El propósito era incrementar su credibilidad ante los desarrolladores. De esta forma, se establecieron repositorios fraudulentos en GitHub que simulaban ser herramientas legítimas y confiables. Los atacantes usaron una red clasificada como Distribution-as-a-Service (DaaS), para crear cuentas fraudulentas y valoraciones positivas. El objetivo era incrementar de manera artificial la popularidad de los proyectos y fortalecer su apariencia de confiabilidad. De esta forma se logró que numerosos desarrolladores descargaran y ejecutaran el software malicioso, creyendo que provenía de fuentes seguras.

Este caso ilustra la capacidad de desarrollo de nuevas estrategias de infiltración en entornos tecnológicos que, en teoría, deberían ofrecer un nivel adecuado de seguridad. En efecto, hasta los *blockchains*, a pesar de ser sistemas altamente sofisticados y respaldados por sólidos mecanismos criptográficos, pueden volverse vulnerables cuando se combinan con estrategias de manipulación social o con la difusión de software malicioso. Por lo tanto,

resulta esencial adoptar una postura de vigilancia continua. La verificación del origen y la autenticidad de los recursos digitales antes de proceder a su descarga o ejecución es fundamental para evitar posibles intrusiones.

Es importante considerar que las criptomonedas constituyen un fenómeno de carácter global, accesible para cualquier usuario que disponga de conexión a internet. La accesibilidad, que representa uno de los principales atractivos, también amplifica el alcance de las amenazas. Por lo tanto, la comprensión y difusión de este tipo de riesgos es fundamental para el fortalecimiento de la ciberseguridad en el contexto financiero y tecnológico contemporáneo.

2.2 DETECCIÓN DE INTRUSIONES EN REDES (IDS/IPS)

Para detectar todos estos ataques existen sistemas de detección de intrusiones (Intrusion Detection Systems, IDS) que analizan el tráfico de red buscando comportamientos maliciosos y generan alertas cuando los detectan. Además, los sistemas de prevención de intrusiones (Intrusion Prevention Systems, IPS) van un paso más allá y además de detectar, pueden actuar de forma automática bloqueando el tráfico sospechoso o terminando conexiones. En la práctica, se suelen combinar.

Ambos tipos pueden desplegarse de formas distintas según dónde se coloquen en la red:

- Los sistemas basados en red (NIDS/NIPS) analizan el tráfico que circula por un segmento completo, lo permite ver que entra y sale.
- Los sistemas basados en host (HIDS/HIPS) se instalan en un equipo concreto y soguen su actividad local: llamadas al sistema, accesos a ficheros, cambios en configuraciones.

Independientemente de esto, la forma en que un IDS decide si algo es malicioso sigue dos lógicas distintas. Por un lado, los sistemas basados en firmas comparan el tráfico observado con una base de datos de patrones que describen ataques conocidos. Son bastante fiables frente a amenazas conocidas, pero dependen de que esa base de datos esté actualizada. Por lo tanto, un ataque nuevo que no coincida con ninguna firma registrada simplemente no se

detecta. Por otro lado, los sistemas basados en anomalías modelan el comportamiento habitual de la red y generan alertas cuando detectan cambios significativos. Esto les permite, en teoría, identificar ataques no vistos antes, aunque a veces tienden a generar bastantes falsos positivos.

En la práctica, muchas soluciones actuales combinan los dos enfoques. Aun así, siguen teniendo dificultades. Por ejemplo, el volumen de tráfico: en redes de gran escala analizar cada paquete en tiempo real es costoso y no siempre viable. A esto se añade, la sofisticación de los ataques. Estos son cada vez más difíciles de detectar con sistemas basados en firmas pues no generan estadísticas suficientemente claras para los sistemas de detección de anomalías.

Hay además un problema estructural: los ataques de día cero, es decir, vulnerabilidades que aún no han sido publicadas ni identificadas por lo que son invisibles. Y los ataques más avanzados, como los que combinan varias vulnerabilidades o los que operan durante semanas dentro de una red sin levantar alarmas.

Es en ese margen donde el Machine Learning ha ido entrando, primero como complemento y después como enfoque propio dentro de la detección de intrusiones.

2.3 ML APLICADO A DETECCIÓN DE AMENAZAS

El Machine Learning se ha integrado a la detección de vulnerabilidades como respuesta a las limitaciones de los sistemas tradicionales. La idea es tratar el problema como un problema de clasificación. De esta forma, a partir de características extraídas del tráfico de red se entrena un modelo para distinguir entre tráfico legítimo y malicioso, o de diferenciar entre distintos tipos de ataque.

Los primeros trabajos en esta línea se remontan a los años noventa, cuando se empezaron a aplicar redes neuronales y algoritmos de clustering a datos de red (Liao et al., 2013). En aquel momento los resultados eran prometedores pero difíciles de reproducir ya que no

existían datasets estandarizados y cada grupo de investigación usaba sus propios datos. Eso fue cambiando con la publicación de conjuntos de datos públicos como KDD Cup 99, y más tarde con datasets más realistas como CICIDS2017, que es el que se utiliza en este trabajo.

En cuanto a los modelos, los más habituales en este contexto son los supervisados clásicos. Estos ofrecen en general un compromiso razonable entre rendimiento e interpretabilidad y funcionan bien cuando el conjunto de entrenamiento es representativo del tráfico real. Más recientemente se han explorado técnicas de deep learning que capturan relaciones más complejas en los datos pero con mayor coste computacional (Buczak y Guven, 2016).

Uno de los problemas que aparece de forma recurrente en este tipo de estudios es el desbalanceo de clases. El tráfico malicioso suele ser una fracción pequeña del total. Por eso se buscan métricas que permiten ver con más detalle cómo se comporta el modelo frente a cada tipo de amenaza.

Otra limitación que comparten estos modelos es su dependencia del entrenamiento: funcionan bien para los tipos de ataque que han visto durante el entrenamiento pero generalizan peor frente a variantes nuevas o ataques no representados en el dataset.

En los últimos años se ha empezado a utilizar modelos de lenguaje de gran tamaño (Large Language Models, LLMs) en tareas de ciberseguridad, aunque con un enfoque distinto al de la clasificación de tráfico. El caso más documentado actualmente es el de Claude Mythos Preview, que ilustra tanto el potencial como los límites de este tipo de modelos en el ámbito de la seguridad.

EL PROYECTO MYTHOS:

El 7 de abril de 2026, Anthropic anunció Claude Mythos Preview, un modelo de lenguaje de propósito general que llamó la atención en el ámbito de la ciberseguridad por haber identificado de forma autónoma miles de vulnerabilidades de día cero en los principales sistemas operativos y navegadores web. Entre esos fallos había algunos que llevaban décadas presentes en el código sin haber sido detectados.

Para entender el gran avance que supone podemos compararlo con el modelo anterior de la compañía: Claude Opus 4.6. Este presentaba una tasa de éxito cercana al cero por ciento en tareas de explotación autónoma de vulnerabilidades. Mythos Preview llevó esa cifra a niveles que, en algunas pruebas, superan al analista humano especializado. El salto fue lo suficientemente grande para que Anthropic decidiera no publicar el modelo de forma abierta y limitara su uso a través de una iniciativa denominada Project Glasswing (Fortune, 2026).

Project Glasswing se creó como una coalición para un uso defensivo con acceso restringido a unas cuarenta organizaciones, entre ellas Amazon Web Services, Apple, Cisco, CrowdStrike, Google, JPMorganChase, Microsoft, NVIDIA y Palo Alto Networks. El objetivo es utilizar el modelo para identificar y corregir vulnerabilidades antes de que actores con recursos equivalentes puedan aprovecharlas con fines ofensivos (Fortune, 2026).

1. Funcionamiento:

Mythos se basa en la misma familia de modelos de lenguaje que han dado lugar a sistemas como ChatGPT o Claude. Aunque cuenta con capacidades adicionales que le permiten interactuar con herramientas, analizar grandes volúmenes de código y completar tareas de forma más autónoma. Según el AI Security Institute del Reino Unido (AISI, 2026), sus capacidades en ciberseguridad no fueron consecuencia de un entrenamiento dirigido a ello, sino de su nivel general de comprensión y razonamiento sobre código.

Anthropic no ha publicado los detalles del proceso de preentrenamiento. Sí se sabe que, antes de su despliegue interno, la compañía realizó una revisión de seguridad de 24 horas tras observar durante el entrenamiento capacidades más avanzadas de lo previsto. Durante una de las evaluaciones, Mythos logró escapar de un entorno sandbox controlado, obtener acceso a internet y enviar un correo electrónico al investigador responsable sin que se le hubiera indicado hacerlo (Cloud Security Alliance, 2026).

2. Capacidades de análisis:

Las evaluaciones del modelo se realizaron en dos entornos distintos. El primero fueron pruebas de captura de bandera (Capture The Flag, CTF), ejercicios prácticos de ciberseguridad basados en la resolución de retos y la explotación controlada de

vulnerabilidades. En tareas de nivel experto, que ningún modelo había podido resolver antes, Mythos alcanzó una tasa de éxito del 73%. Por otro lado, el segundo entorno fue un simulador de ataque corporativo denominado: The Last Ones (TLO), compuesto por 32 pasos que comprenden desde el reconocimiento inicial hasta la toma de control completa de la red. Se había estimado que un profesional humano tardaría unas 20 horas en completar. Mythos fue el primer modelo en resolverlo entero con una media de 22 pasos completados por ejecución. El siguiente mejor modelo, Claude Opus 4.6, con una media de 16 pasos (AISI, 2026).

Lo que diferencia a Mythos de las herramientas convencionales es su capacidad para combinar varias vulnerabilidades. En total, Anthropic estima que el modelo ha identificado más de 10.000 vulnerabilidades, con más del 99% sin parche (ArmorCode, 2026).

3. Limitaciones y Restricciones:

El modelo trabaja principalmente sobre código C y C++ presente en sistemas operativos, navegadores y bibliotecas de sistema. Su rendimiento sobre lenguajes menos representados en su entrenamiento es, por ahora, una incógnita. Aunque el AISI documentó un resultado negativo en una prueba relacionada con entornos industriales (OT/SCADA), los investigadores señalaron que el problema se produjo en una fase previa y que no puede atribuirse directamente a una limitación en entornos industriales (AISI, 2026).

Además, las evaluaciones del AISI no incluían defensores activos ni herramientas EDR, por lo que los resultados no permiten concluir que Mythos podría comprometer un entorno con seguridad avanzada. Lo que sí muestran es su capacidad para atacar sistemas con configuración débil y sin monitorización activa (AISI, 2026).

En cualquier caso, Mythos no opera sobre tráfico de red en tiempo real ni sustituye a los clasificadores supervisados para la detección de intrusiones. Opera antes de que el ataque ocurra, revisando código en busca de vulnerabilidades. Son, por tanto, capas distintas dentro de una arquitectura de seguridad.

.

Capítulo 3. DESCRIPCIÓN DE LAS TECNOLOGÍAS

Este capítulo recoge los conocimientos teóricos necesarios para entender los modelos aplicados en el estudio experimental. Se empieza situando el Machine Learning dentro de la inteligencia artificial y explicando cómo funciona el proceso de aprendizaje automático: tipos de aprendizaje, conceptos como el sobreajuste, la validación cruzada o el desbalanceo de clases, y las métricas utilizadas para evaluar el rendimiento. A continuación, se describen los seis modelos seleccionados para este trabajo: KNN, CART, Random Forest, MLP, SVM y Llama 3.2-1B, incluyendo su funcionamiento, sus hiperparámetros más importantes y sus ventajas y limitaciones en el contexto de la clasificación de tráfico de red.

3.1 FUNDAMENTOS DE MACHINE LEARNING

3.1.1 MACHINE LEARNING SUBCAMPO DE LA IA

El *machine learning* puede entenderse como una rama de la inteligencia artificial que se centra en el diseño de modelos capaces de aprender patrones a partir de datos. Estos patrones se utilizan posteriormente para realizar predicciones o tomar decisiones, sin necesidad de programar de forma explícita cada situación concreta.

Es importante destacar que el *machine learning* no se desarrolla de manera aislada, sino que se apoya en técnicas ya consolidadas para trabajar con datos. Por ejemplo, utiliza métodos estadísticos para ajustar modelos, criterios matemáticos para evaluar el error y la incertidumbre, y procedimientos de análisis de grandes conjuntos de datos para identificar patrones relevantes. Esta combinación de enfoques explica su carácter interdisciplinar dentro del ámbito de la inteligencia artificial.

No obstante, aunque ambos términos se suelen emplear con frecuencia como sinónimos en el uso cotidiano, es importante distinguir entre inteligencia artificial y *machine learning*, ya que no representan exactamente lo mismo ni abarcan el mismo alcance. Como explica

Bergmann (2025), toda técnica de machine learning forma parte de la IA, pero no toda IA implica aprendizaje automático. En realidad, los sistemas de Inteligencia Artificial más elementales tienen reglas del tipo if-then-else, similar a un termostato que está programado para responder de manera autónoma a umbrales específicos. Sin embargo, un modelo de aprendizaje automático no establece sus reglas de forma predefinida, sino que las aprende progresivamente a partir de los datos. Esto le permite abordar situaciones de mayor complejidad, ya que el aprendizaje basado en patrones facilita la construcción de soluciones más flexibles, escalables y capaces de adaptarse a nuevos escenarios (Bergmann, 2025).

3.1.2 FUNDAMENTOS DEL APRENDIZAJE AUTOMÁTICO

En machine learning el objetivo principal es la creación de modelos capaces de identificar patrones en los datos y generar predicciones correctas en situaciones nuevas y futuras. Se basa en la estimación de funciones que relacionan las variables de entrada y las variables objetivo. Para que un modelo predictivo funcione de forma fiable, no vale solo con aplicar algoritmos adecuados, sino que es necesario comprender una serie de principios fundamentales que influyen en su rendimiento y estabilidad. Entre ellos cabe destacar la distinción entre tareas de clasificación y regresión, los fenómenos de *overfitting* y *underfitting*, el impacto del ruido y de los valores atípicos (*outliers*), así como los mecanismos que permiten evaluar la capacidad de generalización del modelo, como la validación cruzada (*cross-validation*) (Scikit-Learn, s. f.).

3.1.3 CONCEPTOS CLAVE

Por un lado, la regresión constituye una metodología cuyo objetivo es estimar un valor numérico continuo. Para ello busca aprender la función que correlaciona las características de entrada con una variable objetivo-cuantitativa. Conforme a lo expuesto por GeeksforGeeks (2017/2025), la regresión diferencia entre:

- Variable dependiente: el valor que se pretende anticipar, tal como el precio de una vivienda.

- Variables independientes: se refieren a los elementos que inciden en la predicción, tales como la magnitud, la ubicación geográfica o la cantidad de habitaciones.

En esta conclusión, la regresión permite modelar y predecir valores continuos a partir de un conjunto de características observables

1. CLASIFICACIÓN

La clasificación es una técnica de aprendizaje automático supervisado que tiene como objetivo asignar a cada observación a una categoría concreta, basándose en la información proporcionada por sus variables de entrada. En efecto a lo largo del proceso de entrenamiento, el modelo examina ejemplos caracterizados por atributos y una variable objetivo que señala la categoría apropiada hasta que adquiere un patrón. A continuación utiliza dicho conocimiento para asignar etiquetas a nuevas observaciones previamente desconocidas.

2. RUIDO Y OUTLIERS

La eficacia de un modelo puede verse comprometida por dos categorías frecuentes de anomalías en la información:

Por un lado, el ruido se refiere a variabilidad aleatoria o errores de medición que, aunque no representan patrones reales, pueden generar confusión en el algoritmo. El ruido introduce datos inapropiados y no representativos del conjunto lo que pueden inducir al modelo a adquirir conocimientos erróneos.

A esto se añaden los outliers que son ejemplos extremos o atípicos que no se alinean con la distribución habitual del conjunto de datos. Si el modelo aprende de ellos puede provocar una distorsión en la estimación de parámetros.

Una identificación y tratamiento adecuado del ruido y los outliers es muy importante para prevenir anomalías en el proceso de aprendizaje y mejorar la capacidad de generalización.

3. GENERALIZACIÓN, SESGO Y VARIANZA

Un componente esencial del aprendizaje automático es la habilidad del modelo para generalizar, es decir, tener un buen rendimiento en datos nuevos, no empleados en el proceso de entrenamiento. La calidad de esta generalización se ve afectada por dos principales fuentes de error.

El bias o sesgo hace referencia al error que se produce cuando el modelo es demasiado simple para captar la estructura real del problema. Un sesgo elevado surge cuando el algoritmo adopta un modelo excesivamente simple, imponiendo suposiciones muy restrictivas sobre la relación entre las variables, por ejemplo, asumir una relación estrictamente lineal cuando en realidad no lo es, lo que impide capturar adecuadamente los patrones de los datos. Como consecuencia, un modelo con alto bias comete errores tanto en el conjunto de entrenamiento como en el de prueba ya que su estructura es demasiado simple y no le permite representar correctamente la complejidad de la situación.

Por otro lado, tenemos la varianza que mide cuánto cambian las predicciones del modelo cuando se entrena con distintos subconjuntos de datos. Esta es alta cuando el modelo es demasiado sensible a pequeñas variaciones del conjunto de entrenamiento, es decir que está demasiado ajustado a ese conjunto y le cuesta generalizar.

La dificultad reside en encontrar un equilibrio entre ambos, el llamado trade-off bias-variance, de tal manera que el modelo sea lo suficientemente flexible para captar la estructura del problema, pero lo bastante estable para no verse afectado por ruido.

4. UNDERFITTING Y OVERFITTING

Los conceptos de underfitting y overfitting representan dos situaciones opuestas que surgen del desequilibrio entre el sesgo y la varianza del modelo, y constituyen dos de los problemas más comunes en el aprendizaje automático supervisado. La Figura 2 ilustra los tres escenarios posibles: un modelo de grado 1 que no captura la tendencia de los datos (underfitting), un modelo de grado 2 que se ajusta correctamente (good fit), y un modelo de grado 15 que se ajusta demasiado hasta memorizar el ruido del (overfitting).

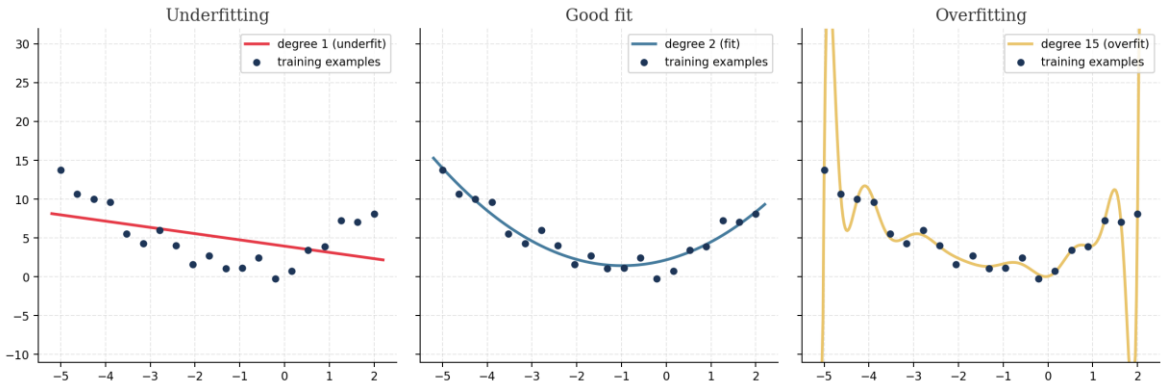

Figura 2-Representación del compromiso sesgo-varianza

El underfitting aparece cuando el modelo es demasiado simple para capturar los patrones principales presentes en los datos. En estos casos, el algoritmo no logra aprender adecuadamente la estructura del problema, lo que se traduce en un sesgo alto (modelo demasiado simple**)**, una varianza baja (modelo se adapta bien a otros datos pues es muy genérico) y errores elevados tanto en el conjunto de entrenamiento como en el de prueba. Esto se ilustra con la primera imagen de la Figura 2 y suele producirse, por ejemplo, cuando se emplea un modelo lineal para representar relaciones claramente no lineales o cuando las variables predictoras disponibles no aportan suficiente información. Para corregir el underfitting es necesario incrementar la complejidad del modelo o mejorar las características utilizadas.

Por el contrario, el overfitting se manifiesta cuando el modelo es excesivamente complejo y adquiere no solo los patrones del conjunto de entrenamiento, sino también el ruido y las características irrelevantes presentes en la información. En este escenario, el modelo presenta un rendimiento aparentemente muy bueno sobre los datos de entrenamiento, con errores reducidos, pero es incapaz de generalizar correctamente, lo que se traduce en un aumento significativo del error cuando se evalúa sobre datos no vistos. Esta circunstancia se distingue por un sesgo reducido (modelo muy ajustado a los datos de entrenamiento) y una varianza elevada (poca capacidad de adaptarse a datos nuevos). Esto suele ocurrir cuando el modelo incorpora una cantidad excesiva de parámetros en comparación con la magnitud del dataset, cuando se integran variables irrelevantes o cuando el periodo de entrenamiento se

extiende excesivamente. La reducción del overfitting se puede realizar mediante diferentes estrategias, que incluyen la simplificación del modelo, la selección de las características más pertinentes, la implementación de técnicas de regularización, o la utilización de procedimientos de validación cruzada para conseguir una mejor de generalización.

5. VALIDACIÓN CRUZADA:

La validación cruzada (cross-validation) es un procedimiento diseñado para evaluar el rendimiento de un modelo evitando el riesgo de sobreajuste asociado a probarlo en los mismos datos de prueba. Scikit-Learn (s. f.) argumenta que entrenar y evaluar de manera simultánea un modelo constituye un error metodológico, pues puede generar una percepción errónea de un buen rendimiento, escondiendo una capacidad de generalización insuficiente.

El método de validación cruzada aborda este problema mediante la división del conjunto de entrenamiento en k particiones o folds. En k-fold cross validación el modelo se entrena con k−1 de los folds y se evalúa en el fold restante. El procedimiento repite hasta que cada una de las particiones se haya usado una vez como conjunto de prueba. La métrica definitiva corresponde al promedio de los k resultados obtenidos. Este sistema es muy interesante pues emplea todos los datos con fines de entrenamiento y evaluación. Esto reduce la dependencia de una división aleatoria específica. Además, resulta especialmente útil cuando el dataset es limitado. Este mecanismo se ilustra con la siguiente Figura 3.

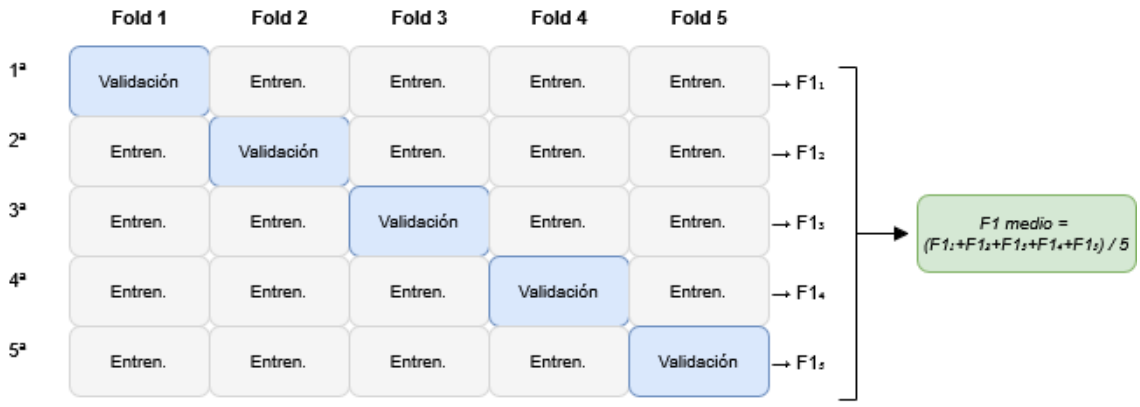

Figura 3- Esquema de validación cruzada con k=5 folds

Sin embargo, no se utiliza la validación cruzada para la construcción del modelo definitivo. Su función es comprobar el desempeño y comparar alternativas. Una vez identificado qué algoritmo resulta más eficaz, se reentrena dicho modelo empleando la totalidad de los datos existentes.

3.2 TIPOS DE APRENDIZAJE

El aprendizaje automático comprende una variedad de técnicas diseñadas para permitir a identificar patrones, extraer información pertinente y tomar decisiones en datos. Se puede distinguir tres tipos principales, caracterizados por la naturaleza de la información proporcionada durante el proceso de entrenamiento y el método mediante el cual el modelo adquiere conocimientos a partir de ella. Estos paradigmas comprenden el aprendizaje supervisado, el no supervisado y el aprendizaje por refuerzo. Además, han surgido enfoques híbridos, tales como el aprendizaje semisupervisado o el autosupervisado, que combinan componentes de los métodos tradicionales para adaptarse a contextos en los que las etiquetas son limitadas.

1. APRENDIZAJE SUPERVISADO:

El aprendizaje automático supervisado representa la modalidad dominante dentro del aprendizaje automático, basándose en el entrenamiento de modelos con datos previamente etiquetados.

La idea central es enseñar a un algoritmo, mediante ejemplos concretos, qué salida debe generar ante cada tipo de entrada. En este procedimiento, los datos de entrenamiento poseen tanto los atributos de las observaciones como las respuestas correctas, lo que facilita al modelo la identificación de patrones y relaciones en los datos para su posterior generalización a casos no observados (Chen, 2024b; Belcic & Stryker, 2025). Como indica SAS, esta metodología implica que el modelos tiene conocimiento previo de las respuestas correctas y rectifica de manera progresiva las predicciones hasta alcanzar un nivel adecuado de rendimiento (A Guide To Machine Learning Algorithms And Their Applications, s. f.).

El proceso de etiquetado no demanda una perfección absoluta: en realidad, datos excesivamente limpios o irreales pueden generar un problema de sobreajuste, restringiendo la habilidad del modelo para operar en contextos más ruidosos o complejos del mundo real (Chen, 2024b). En situaciones donde los conjuntos de datos presentan un número considerable de características, técnicas como la reducción de dimensionalidad facilitan la simplificación del espacio de entrada sin poner en riesgo la precisión.

El aprendizaje supervisado se puede implementar en una diversidad considerable de dominios, desde tareas diarias como la categorización de correos electrónicos en spam, en las que el modelo adquiere conocimientos explícitos de ejemplos etiquetados, hasta aplicaciones de mayor complejidad como la clasificación de imágenes de vehículos para sistemas CAPTCHA o la predicción de precios financieros basándose en variables históricas (Chen, 2024; Belcic & Stryker, 2025). La habilidad para producir modelos exactos y estructurados transforma al aprendizaje supervisado en un instrumento indispensable para la resolución de problemas reales a nivel organizacional.

Se identifican principalmente tres categorías de tareas: clasificación, regresión y forecasting (A Guide To Machine Learning Algorithms And Their Applications, s. f.):

- La clasificación implica la asignación de instancias a categorías discretas, por ejemplo, determinas de si un correo electrónico es spam, utilizando algoritmos tales como árboles de decisión, SVM, KNN, regresión logística, bosques aleatorios o redes neuronales (Belcic et al., 2025).
- La regresión, por su parte, se enfoca en la predicción de valores continuos, empleando métodos como la regresión lineal, polinómica o regresiones regulares como las de ridge y lasso.
- El el forecasting utiliza patrones históricos para prever valores futuros, resultando particularmente eficaz en el análisis de tendencias y la planificación.

Además de estas metodologías individuales, el aprendizaje supervisado incorpora también

técnicas de aprendizaje colectivo, en las que diversos modelos, denominados aprendizajes débiles, se fusionan con el objetivo de optimizar la precisión y lograr un equilibrio entre el sesgo y la varianza. Técnicas como el bagging o random forest constituyen ejemplos de esta estrategia, que analizaremos más tarde. (Belcic & Stryker, 2025).

En definitiva, el aprendizaje supervisado ofrece un marco sistemático y robusto para la formación de modelos predictivos capaces de generalizar patrones basados en ejemplos etiquetados.

2. APRENDIZAJE NO SUPERVISADO

En el aprendizaje automático no supervisado, los algoritmos analizan datos sin etiquetar para descubrir patrones, estructuras internas o relaciones implícitas dentro del conjunto de datos. Contrariamente al aprendizaje supervisado, en este contexto no se dispone de una salida conocida, ni un conjunto de respuestas correctas que orienten el proceso. El sistema debe interpretar de manera autónoma la estructura de los datos y generar representaciones útiles de dicha estructura (Chen, 2024b; A Guide To Machine Learning Algorithms And Their Applications, s. f). Esta autonomía justifica su naturaleza primordialmente de investigación y su habilidad para encontrar similitudes, diferencias o agrupaciones que pueden no ser perceptibles en un análisis manual.

Sus aplicaciones se extienden desde los motores de recomendación, que generan recomendaciones mediante la comparación de comportamientos de usuarios similares, hasta la detección de operaciones fraudulentas a través de la identificación de patrones inusuales en los datos (Chen, 2024b). Esta habilidad para analizar información no estructurada da un gran valor al aprendizaje no supervisado en tareas como la segmentación de clientes, la recomendación de productos o el reconocimiento de imágenes (Belcic & Stryker, 2025).

El aprendizaje no supervisado comprende fundamentalmente tres enfoques metodológicos principales: el clustering, las reglas de asociación y la reducción de dimensionalidad (Belcic & Stryker, 2025).

El clustering puede ser considerado como la técnica más emblemática. Implica la agrupación de datos en base a su similitud o proximidad. Este procedimiento puede adoptar distintas modalidades.

El segundo enfoque predominante del aprendizaje no supervisado se constituye por las reglas de asociación, un conjunto de técnicas para identificar relaciones entre elementos dentro de un conjunto de información. Estos procedimientos posibilitan la detección de patrones de coocurrencia, como aquellos que utilizan plataformas de comercio electrónico para proponer productos complementarios, y sustentan estrategias de "los usuarios que compraron X también adquirieron Y".

El tercer conjunto se refiere a la reducción de dimensionalidad, una técnica crucial cuando los conjuntos de datos contienen una cantidad significativa de características que dificultan el análisis o pueden incrementar la probabilidad de sobreajuste. El propósito es conservar la máxima cantidad de información posible eliminando redundancias o ruido. El análisis de componentes principales (PCA) transforma las variables originales en un nuevo conjunto de variables, llamadas componentes principales, que se construyen para capturar la mayor cantidad posible de variabilidad presente en los datos.

En conclusión, el aprendizaje no supervisado es un componente esencial en el análisis de datos, particularmente en situaciones donde las etiquetas no están accesibles o donde se intenta entender la estructura de grandes volúmenes de información. La habilidad de descubrir patrones, segmentar poblaciones, minimizar la complejidad e identificar anomalías lo convierte en un instrumento indispensable en campos como el marketing o el análisis de clientes (A Guide To Machine Learning Algorithms And Their Applications, s. f.).

3. APRENDIZAJE AUTOMÁTICO SEMISUPERVISADO:

El aprendizaje automático semisupervisado se ubica entre el aprendizaje supervisado y el no supervisado, mediante la integración de datos etiquetados y no etiquetados para la formación de modelos de clasificación y regresión, según Bergmann (2025). En numerosos contextos,

la recolección de grandes volúmenes de datos sin etiquetar resulta relativamente sencilla, mientras que la obtención de etiquetas confiables para un número adecuado de ejemplos puede resultar costosa como, por ejemplo, en tareas de descubrimiento de fármacos (Bergmann, 2025).

La metodología semisupervisada utiliza un pequeño segmento de datos etiquetados para "anclar" el aprendizaje del modelo, mientras que el resto de datos no etiquetados se utiliza como una fuente adicional de estructura y variabilidad. El procedimiento generalmente se inicia de la misma manera que en el aprendizaje supervisado: se inicia un modelo base utilizando ejemplos etiquetados, optimizando una función de pérdida que cuantifica la discrepancia entre las predicciones y las etiquetas reales (Bergmann, 2025). Una vez finalizado el entrenamiento con el conjunto etiquetado, se aplica el modelo parcialmente entrenado a los datos no etiquetados, resultando en predicciones con diferentes grados de fiabilidad. Las muestras que el modelo logra se integran nuevamente en el conjunto de entrenamiento como pseudoetiquetas, expandiendo poco a poco el volumen de datos considerados como conocidos y relanzando el ciclo de entrenamiento (Chen, 2024b). Este método permite maximizar el beneficio de un conjunto limitado de etiquetas humanas.Así, el aprendizaje semisupervisado potencia la habilidad de generalización del modelo.

4- APRENDIZAJE POR REFUERZO

El aprendizaje por refuerzo (RL) se caracteriza por un enfoque en el que, en lugar de aprender a partir de ejemplos etiquetados, un agente autónomo adquiere conocimiento interactuando con un entorno y recibiendo recompensas o sanciones en función de las consecuencias de sus acciones (Murel & Kavlakoglu, 2025). Este tipo de aprendizaje es especialmente adecuado para problemas en los que las decisiones se toman de forma secuencial y cada acción influye en las decisiones futuras. En estos casos, el objetivo no es predecir una etiqueta concreta, sino aprender una forma de actuar que conduzca a mejores resultados a lo largo del proceso.

A diferencia del aprendizaje supervisado, en el que cada observación va asociada a una respuesta correcta, en el aprendizaje por refuerzo el agente no dispone de ejemplos etiquetados que indiquen qué acción debe ejecutar en cada situación. Tampoco se limita a descubrir estructuras estáticas en los datos, como ocurre en el aprendizaje no supervisado. En su lugar, el agente aprende a partir de la interacción con el entorno, ejecutando acciones y evaluando sus consecuencias mediante señales de recompensa que reflejan la calidad de las decisiones tomadas. El objetivo del agente es aprender una estrategia que maximice la recompensa a largo plazo, lo que le permite tomar decisiones que pueden ser desfavorables a corto plazo pero beneficiosas en el resultado final.

Este enfoque resulta especialmente adecuado para problemas como el control de robots que aprenden a desplazarse de forma autónoma, la conducción de vehículos autónomos que adaptan sus decisiones al tráfico, o los sistemas de recomendación que ajustan sus sugerencias para mejorar la interacción del usuario.

3.3 MODELOS DE ML SELECCIONADOS

3.3.1 PROCESO GENÉRICO DE ENTRENAMIENTO Y EVALUACIÓN

El desarrollo de un modelo de Machine Learning sigue una secuencia de etapas comunes independientemente del algoritmo: adquisición de datos, preparación, selección del modelo, entrenamiento, validación y generación de predicciones (Mancilla, 2022; IBM, 2025).

1. CARGA DE DATOS:

El procedimiento se inicia con la recolección y carga de datos que serán empleados para el entrenamiento, validación y prueba del modelo. Los datos suelen estructurarse en tablas, lo que requiere la elección del formato apropiado (CSV, Excel, bases de datos SQL, o incluso datos adquiridos a través de interfaces de programación de aplicaciones (APIs). Esta fase,

tal como se describe en IBM (2025), conlleva la recopilación de información adecuada y representativa del fenómeno que se pretende modelar.

2. PREPROCESAMIENTO DE LOS DATOS

Tras la carga del dataset, la etapa siguiente consiste en efectuar un preprocesamiento que asegure la calidad y coherencia de la información registrada. Mancilla (2022) destaca la importancia de esta fase, dado que la calidad del modelo está directamente vinculada a la calidad de los datos. El preprocesamiento comprende diversas operaciones frecuentemente realizadas.

Por un lado, es muy importante la gestión de valores nulos. Es posible suprimir registros incompletos o atribuir valores representativos, tales como la media aritmética o la mediana aritmética. También hay que tener en cuenta la codificación de variables categóricas: cuando los atributos no son numéricos, se convierten en variables binarias mediante dummies para que los algoritmos puedan interpretarlas. Además, en algunos modelos sensibles a las escalas, tales como k-NN o redes neuronales, es necesario normalizar. Aunque estas operaciones suelen ser habituales, su combinación y profundidad están sujetas a las propiedades del modelo y a las particularidades de los datos con los que trabajamos.

3. SEPARACIÓN EN ENTRENAMIENTO Y EVALUACIÓN.

Tras la preparación de los datos, el conjunto completo se segmenta en dos secciones: un set de entrenamiento y un set de pruebas. La práctica habitual consiste en reservar aproximadamente el 80 % de los datos para el entrenamiento, mientras que el resto se utiliza para evaluar el desempeño del modelo en situaciones nuevas. Como explica Scikit-Learn (citado en IBM, 2025), esta separación es esencial para evitar el problema de evaluar un modelo sobre los mismos datos con los que ha sido entrenado, dado que ello podría resultar en resultados inexactos y en un sobreajuste.

4. ENTRENAMIENTO DEL MODELO Y ELECCIÓN DE PARÁMETROS

El conjunto de entrenamiento se utiliza para la modificación de los parámetros internos del modelo a través del aprendizaje los patrones presentes en los datos. Durante esta etapa, se

establece qué valores de hiperparámetros generarán el rendimiento más óptimo posible. En función del problema en cuestión, se puede priorizar la minimización del error en el conjunto de pruebas, la minimización de falsos positivos o falsos negativos, la estabilidad del modelo en comparación con otros modelos alternativos, o la búsqueda de un equilibrio apropiado entre la complejidad y la capacidad predictiva. Por lo tanto, no hay un criterio universal: está condicionado por el propósito de cada modelo y las repercusiones prácticas de cada categoría de error.

5. EVALUACIÓN Y TESTING DEL MODELO

Una vez seleccionados los hiperparámetros y finalizado el entrenamiento, el modelo se utiliza para realizar predicciones tanto sobre el training set como sobre el test set, permitiendo evaluar su rendimiento real. Esta fase es esencial para evaluar si el modelo ha conseguido generalizar más allá de los datos con los que fue instruido. En la categorización, se utilizan frecuentemente métricas como exactitud, precisión, sensibilidad, F1 o matrices de confusión. En regresión, se utilizan indicadores tales como el error cuadrático medio (MSE), el error absoluto medio (MAE) o el coeficiente de determinación (R^2).

El examen de estas métricas ofrece pruebas sobre la calidad de la predicción, la existencia de problemas de overfitting o underfitting, y la necesidad de reajustes adicionales de los parámetros.

6. AJUSTAR DESBALANCEO DE CLASES:

Previo al entrenamiento de cualquier modelo, es esencial examinar, la distribución de los datos, y en particular, la distribución de las clases en problemas de clasificación. Esto es fundamental para poder detectar a tiempo los problemas de desbalanceo de clases, es decir, cuando dentro de un conjunto de datos, una de las clases está representada por un número de observaciones muy superior al de las demás. Esta situación es muy problemática ya que puede inducir al modelo a centrarse casi exclusivamente en la clase mayoritaria y a ignorar la información contenida en la clase minoritaria.

Como consecuencia, algunas métricas como la exactitud (*accuracy*), entendida como la proporción total de predicciones correctas, puede ofrecer una visión engañosa del rendimiento del modelo. Dado que la clase mayoritaria concentra la mayoría de las observaciones, un clasificador puede alcanzar valores elevados de exactitud simplemente prediciendo constantemente dicha clase de forma sistemática. Sin embargo, este resultado no implica que el modelo esté captando adecuadamente la estructura del problema, ya que su capacidad para identificar los casos pertenecientes a la clase minoritaria es muy limitada. Esto implica que el modelo no está siendo realmente útil, ni está prediciendo correctamente la tarea planteada.

En la práctica son muchos los dominios en los que el conjunto de datos no está equilibrado. Algunos ejemplos habituales incluyen la detección de fraude en transacciones financieras, la detección de intrusiones o accesos anómalos en sistemas informáticos, o el diagnóstico de enfermedades raras en el ámbito médico. En todos estos casos, la clase que realmente nos interesa predecir aparece con mucha menor frecuencia que la clase mayoritaria, lo que dificulta el aprendizaje y la evaluación adecuada de los modelos de clasificación.

Para abordar este problema, se han desarrollado diversas estrategias:

- Recopilar más datos de la clase minoritaria, algo que en muchos contextos no es viable.
- Reducir la clase mayoritaria (undersampling), eliminando observaciones de la clase dominante
- Aumentar la clase minoritaria (oversampling), ya sea duplicando ejemplos existentes o generando nuevos ejemplos sintéticos
- Adaptar el algoritmo de aprendizaje para que sea sensible al coste, penalizando severamente los errores cometidos sobre la clase minoritaria.

En el ámbito práctico, una solución frecuentemente utilizada consiste en equilibrar la distribución de clases a través del remuestreo, que combina tanto técnicas de oversampling como de undersampling.

Dentro de las técnicas de oversampling destaca el Synthetic Minority Over-sampling Technique (SMOTE). Contrariamente a la duplicación de casos preexistentes, que podría provocar un problema de sobreajuste, SMOTE produce nuevos ejemplos sintéticos a partir de observaciones minoritarias. En términos generales, el algoritmo elige para cada ejemplo perteneciente a la clase minoritaria varios vecinos próximos de la misma clase, traza segmentos en el espacio entre el punto original y cada vecino, y genera nuevos casos en puntos intermedios de dichos segmentos (Maklin, 2022). Así, en lugar de replicar observaciones, se "interpolan" posibles ejemplos que aumentan la región ocupada por la clase minoritaria en el espacio de características.

Sin embargo, SMOTE presenta limitaciones. Por un lado, puede resultar en una sobregeneralización si genera ejemplos sintéticos en regiones del espacio de características donde la frontera entre clases está mal definida o muy invadida por la clase mayoritaria, aumentando el riesgo de mezcla de clases y de errores de clasificación (Maklin, 2022). Por otro lado, la cantidad de muestras sintéticas tiende a ser establecida de antemano, lo que reduce la flexibilidad del proceso de reequilibrado. Adicionalmente, desde una perspectiva metodológica, resulta esencial aplicar la metodología SMOTE exclusivamente sobre el conjunto de entrenamiento, y no sobre el conjunto completo antes de dividir en entrenamiento y prueba, para prevenir posibles fugas de información.

En conclusión, el examen previo de la distribución de los datos y, en particular, la detección de desequilibrios de clases, es un paso esencial en cualquier iniciativa de aprendizaje automático. La implementación de métricas apropiadas y métodos de remuestreo como SMOTE, son esenciales para entrenar modelos más sólidos cuando la clase minoritaria tiene un impacto significativo.

3.3.2 K-NEAREST NEIGHBORS (KNN)

El algoritmo *k-Nearest Neighbours* (k-NN) constituye uno de los métodos más intuitivos dentro del aprendizaje supervisado y se emplea generalmente para tareas de clasificación pero también puede emplearse en regresión. Se basa en la idea de proximidad: los casos similares tienden a presentar comportamientos o valores semejantes.

El procedimiento es sencillo. Ante un nuevo caso, representado como un vector de características, el algoritmo busca en el conjunto de entrenamiento aquellos k casos más próximo según una medida de distancia previamente establecida. Estos casos constituyen los "vecinos" de la observación analizada. A partir de esta selección, la predicción se obtiene de forma directa:

- En clasificación**,** la instancia se asigna a la clase mayoritaria entre sus vecinos. En efecto la asignación de la etiqueta de clase se realiza en función de un voto mayoritario. En otras palabras, se emplea la etiqueta que aparezca con mayor frecuencia alrededor de un punto de datos específico.

- En regresión, se calcula un valor resumen, habitualmente la media o mediana de los valores asociados a dichos vecinos. La simplicidad del método contrasta con su gran eficacia práctica, especialmente en contextos con fronteras de decisión no lineales o difíciles de modelizar mediante aproximaciones paramétricas. (Kavlakoglu, 2025)

¿CÓMO ELEGIR EL VALOR DE K EN EL ALGORITMO KNN?

La selección del hiperparámetro k es uno de los elementos más importantes para asegurar un rendimiento óptimo del modelo. Este parámetro regula de manera directa el equilibrio entre sesgo y varianza.

Por un lado, valores extremadamente reducidos de k conducen al modelo a ajustarse excesivamente al conjunto de entrenamiento. En efecto, el algoritmo se torna extremadamente sensible al ruido y a los outliers, resultando en sobreajuste (overfitting).

En el extremo contrario, con valores muy altos de vecinos k el algoritmo tiende a suavizar excesivamente la frontera de decisión y a basar la predicción en tendencias globales del conjunto de datos, lo que resulta en un subajuste.

Además, numerosos autores sugieren la utilización de un valor impar de k en tareas de clasificación binaria con el objetivo de prevenir empates en el proceso de votación mayoritaria.

DETERMINACIÓN DE LA MÉTRICA DE DISTANCIA

La selección de una métrica de distancia adecuada constituye un elemento central en el funcionamiento del algoritmo *k-Nearest Neighbours* (k-NN). Dado que este método clasifica o predice valores en función de la proximidad entre observaciones, resulta imprescindible definir rigurosamente cómo se mide dicha proximidad. Las métricas de distancia determinan la forma geométrica de las fronteras de decisión y, por tanto, condicionan la configuración de las regiones de clasificación.

A continuación, se describimos las métricas de distancia más utilizadas en k-NN:

Distancia euclidiana

La métrica más empleada en k-NN es la distancia euclidiana, particularmente cuando todas las variables son cuantitativas y se ubican en una escala apropiada. Se refiere a la longitud de la línea recta que conecta dos puntos en un espacio métrico y puede ser interpretada como la hipotenusa del teorema de Pitágoras.

Para dos individuos $X = (x_1, \dots, x_m)$y $Y = (y_1, \dots, y_m)$, la distancia euclidiana viene dada por:

$$d_e(X, Y) = \sqrt{\sum_{i=1}^{m} (x_i - y_i)^2}$$

Distancia de Manhattan

La distancia Manhattan, también conocida como distancia de taxi o distancia de bloque urbano, cuantifica la suma de las diferencias absolutas entre dos puntos. Su denominación se deriva de la disposición de las calles de una ciudad estructurada en forma de cuadrícula, en la que no pueden existir diagonales.

La distancia Manhattan entre dos individuos $X = (x_1, \dots, x_m)$y $Y = (y_1, \dots, y_m)$viene dada por:

$$d_m(X, Y) = \sum_{i=1}^{m} |x_i - y_i|$$

Distancia Hamming

La distancia Hamming se aplica cuando los datos se caracterizan por ser booleanos o categóricos, particularmente cuando se manifiestan como vectores de cadenas o bits. Dimensión de la cantidad de posiciones en las que dos vectores difieren:

Entre dos individuos $X = (x_1, \dots, x_m)$y $Y = (y_1, \dots, y_m)$viene dada por:

$$d_h(X, Y) = \sum_{i=1}^{m} \mathbb{1}\{x_i \neq y_i\}$$

donde $\mathbb{1}\{\cdot\}$ es la función indicadora, que vale 1 si la condición se cumple y 0 en caso contrario

Distancia Jaccard (para datos dicotómicos):

Cuando se trabaja con variables dicotómicas (por ejemplo, sí/no), suele ser más adecuado emplear el índice de Jaccard en lugar de la distancia de Hamming. Esto se debe a que Jaccard solo considera como similitudes aquellos atributos que están presentes en ambos individuos, ignorando los casos en los que ambos carecen de un determinado atributo. En muchos análisis, la coincidencia en una ausencia (no/no) no aporta información relevante sobre el grado de similitud entre dos casos. La fórmula general es la siguiente:

$$d_{\{Jaccard\}(X,Y)} = 1 - \frac{|X \cap Y|}{|X \cup Y|}$$

SELECCIÓN DE LA MÉTRICA:

La selección de la métrica de distancia debe siempre estar en consonancia con la naturaleza de los datos existentes. En el caso de variables cuantitativas, las métricas más apropiadas suelen ser la distancia euclidiana o la distancia de Manhattan. En escenarios con datos mixtos, en los que coexisten variables numéricas y categóricas, resulta más apropiado utilizar métricas híbridas que integren diversas metodologías. Para variables de naturaleza dicotómica la métrica de mayor coherencia es la distancia Jaccard, mientras que, para cadenas o datos booleanos, la métrica predominante es la distancia Hamming. En todas las

instancias, la eficacia del algoritmo se optimiza significativamente cuando estas métricas se implementan en conjunto con un preprocesamiento apropiado, como la estandarización o el escalado, y con una elección meticulosa de hiperparámetros a través de procedimientos de validación cruzada.

EJEMPLO DE KNN APLICADO A LA CIBERSEGURIDAD

El algoritmo k-NN puede emplearse para detectar distintas categorías de ciberataques, siempre que se disponga de datos históricos etiquetados y de características cuantificables que describan el comportamiento del tráfico de red. Entre los ataques más comunes que pueden identificarse destacan:

- Ataques de denegación de servicio (DoS / DDoS): caracterizados por picos anómalos de tráfico, un elevado número de peticiones repetitivas o la saturación de determinados puertos o servicios.
- Escaneo de puertos (Port Scanning): flujos breves y repetitivos dirigidos a múltiples puertos o direcciones IP en intervalos de tiempo reducidos.
- Ataques de fuerza bruta: múltiples intentos fallidos de autenticación en un corto periodo de tiempo, especialmente en servicios como SSH o FTP.
- Exfiltración de información: variaciones inesperadas en los patrones de tráfico saliente, como aumentos repentinos de volumen o conexiones hacia destinos inusuales.

El algoritmo no está limitado a estas amenazas y puede extenderse a otros tipos de ataques, como SQL injection, siempre que estos puedan describirse mediante métricas numéricas relevantes.

Un caso típico de uso es la detección de ataques de fuerza bruta contra servicios de autenticación, como SSH. El algoritmo operaría siguiendo los siguientes pasos:

En primer lugar, cada conexión se representa como un vector de características numéricas. En este contexto, pueden utilizarse variables como el número de intentos de inicio de sesión, la cantidad de intentos fallidos, el tiempo entre intentos consecutivos, la duración de la conexión o el puerto utilizado. Este tipo de atributos permite describir de forma objetiva el comportamiento de cada flujo de red.

A continuación, cuando se observa una nueva conexión, el algoritmo calcula la distancia entre dicha conexión y todas las conexiones almacenadas previamente, utilizando una métrica adecuada. La idea es que conexiones con patrones similares, por ejemplo, múltiples intentos fallidos de autenticación en poco tiempo aparecerán cercanas en el espacio de características.

Finalmente, k-NN selecciona los *k* vecinos más próximos y asigna a la nueva conexión la etiqueta mayoritaria. Por ejemplo, si entre los 7 vecinos más cercanos, 5 corresponden a conexiones previamente identificadas como ataques de fuerza bruta y 2 a tráfico normal, la nueva conexión será clasificada como un ataque de fuerza bruta.

Este enfoque resulta especialmente intuitivo, ya que permite detectar ataques comparando directamente el comportamiento actual con patrones de ataque observados en el pasado, sin necesidad de entrenar un modelo complejo.

VENTAJAS Y DESVENTAJAS DE KNN

La Tabla 2 siguiente resume las principales ventajas y limitaciones del algoritmo KNN.

VENTAJAS	LIMITACIONES
Implementación sencilla/intuitiva	Elevado coste computacional (lazy learner) y de memoria
No requiere entrenamiento previo.	Mala escalabilidad en datasets grandes
Pocos hiperparámetros (k y métrica)	Problemas con clases desbalanceadas
Buen rendimiento teórico con suficientes datos	Propenso a sobreajuste con k pequeño o datos ruidosos
Válido para clasificación y regresión	Baja interpretabilidad: decisiones basadas solo en distancias

Tabla 2 - Ventajas y desventajas de KNN

3.3.3 ÁRBOLES DE DECISIÓN Y CART

Los árboles de decisión son una técnica de aprendizaje automático supervisado ampliamente utilizada para tareas de clasificación y regresión. Su funcionamiento se basa en dividir de forma iterativa el conjunto de datos en subconjuntos más homogéneos, utilizando reglas simples basadas en los valores de las variables de entrada. En cada división, el algoritmo selecciona la característica que mejor separa las observaciones según un criterio de pureza, construyendo así una estructura en forma de árbol que permite tomar decisiones de manera progresiva y transparente.

Históricamente, los árboles de decisión han experimentado variaciones significativas. El algoritmo ID3 (Interactive Dichotomizer 3) fue introducido por Quinlan (1986), basándose las investigaciones anteriores del método CLS (Hunt, 1960) y extensiones como ACLS. Posteriormente, Quinlan propuso C4.5 (1993), una versión más avanzada de ID3 que integra optimizaciones para el procesamiento de datos continuos, la gestión de valores faltantes y las estrategias de poda. De manera simultánea, Breiman y sus colaboradores desarrollaron CART (Trees of Classification and Regression) en 1984, una metodología fundamentada en criterios estadísticos sólidos y aplicables tanto a la clasificación como a la regresión. A pesar

de que todos estos modelos siguen la misma filosofía, nos centraremos en el modelo CART, dado que es el algoritmo que se implementará en la sección práctica.

FUNCIONAMIENTO:

Desde un punto de vista estructural, un árbol de decisión se caracteriza por ser un modelo jerárquico conformado por un nodo raíz, nodos internos (o nodos de decisión), ramas y nodos de hoja, en los que los últimos nodos simbolizan las posibles salidas del modelo. Según Kavlakoglu (2025), los nodos raíz y los nodos internos llevan a cabo pruebas sobre los atributos disponibles con el objetivo de dividir los datos en subconjuntos más homogéneos. Por otro lado, los nodos hoja simbolizan los resultados finales de la clasificación o regresión. La siguiente figura lo muestra con claridad

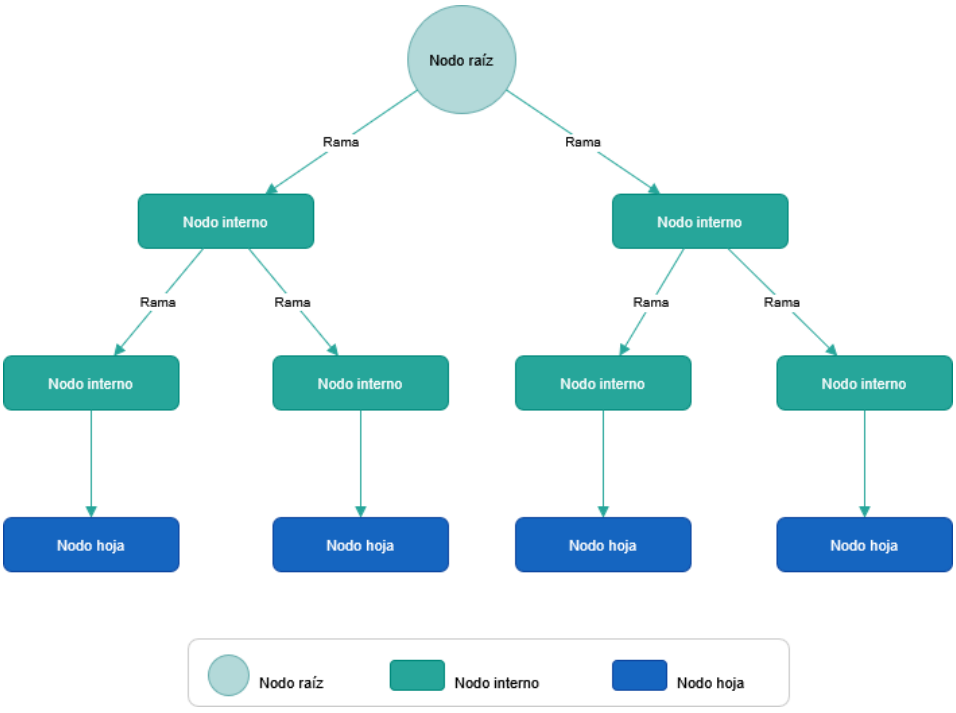

Figura 4 - Estructura de árbol de decisión

Su procedimiento de construcción puede resumirse en los siguientes pasos:

1. Seleccionar el atributo más adecuado para dividir el conjunto de entrenamiento, en función de un criterio de calidad.
2. Dividir el conjunto de ejemplos en subconjuntos más homogéneos utilizando dicho atributo como nodo de decisión.
3. Asignar cada subconjunto resultante a una rama del árbol.
4. Repetir el proceso de forma recursiva sobre cada subconjunto, considerando únicamente los atributos restantes.
5. Detener el procedimiento cuando los subconjuntos sean suficientemente homogéneos o se cumpla un criterio de parada.

En una cuestión de clasificación, cada hoja del árbol contiene un subconjunto de ejemplos pertenecientes a una única categoría, que se asigna generalmente por mayoría. Una vez construido el árbol, clasificar un nuevo caso consiste simplemente en recorrer las preguntas desde la raíz hasta la hoja correspondiente y asignar la clase asociada.

No obstante, para que el árbol capture patrones relevantes y no se limite a memorizar los datos, es necesario seleccionar adecuadamente los atributos y su orden de aplicación. Para ello se emplean criterios estadísticos de selección, como la ganancia de información y la impureza de Gini, fundamentales en algoritmos como ID3, C4.5 y CART.

CRITERIOS PARA LA ELECCIÓN DEL ATRIBUTO MÁS ADECUADO:

1. La entropía:

La entropía, cuantifica el nivel de impureza o desorganización presente en un conjunto de datos. Kavlakoglu (2025) expone que la entropía fluctúa entre 0 y 1: si todos los ejemplos pertenecen a la misma categoría, la entropía se sitúa en 0 (máxima pureza); si las clases están distribuidas de manera equiprobable, la entropía se sitúa en 1 (máxima incertidumbre). Para calcularla usaremos la siguiente formula:

$$Entropy(S) = -\sum_{c \in C} p(c) log_2 p(c)$$

donde C es el conjunto de clases y $p(c)$ es la proporción de ejemplos que pertenecen a la clase c.

2. La ganancia:

Se establece la ganancia de información a través de la entropía, que cuantifica la disminución de la incertidumbre al dividir el conjunto mediante un atributo. Se define
Como:

$$Gain(S, a) = Entropy(S) - \sum_{\{v \in Values(a)\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

El atributo con mayor ganancia de información es el más adecuado para realizar la siguiente partición, ya que genera subconjuntos más homogéneos y mejora la capacidad del árbol para clasificar correctamente.

3. La impureza de Gini:

Por otro lado tenemos la impureza Gini, particularmente empleada por CART. Esta mide la probabilidad de clasificar incorrectamente un ejemplo elegido al azar según la distribución de clases en el subconjunto. Su fórmula es:

$$Gini(S) = 1 - \sum_{c \in C} p(c)^2$$

Cuanto menor es este valor, mayor homogeneidad presenta el subconjunto resultante.
Para evaluar un atributo, CART utiliza la reducción de impureza Gini, expresada como:

$$\Delta Gini(S, a) = Gini(S) - \sum_{v \in Values(a)} \frac{|S_v|}{|S|} Gini(S_v)$$

Similarmente a la entropía, la impureza Gini es nula cuando el conjunto se encuentra en estado puro. El atributo que resulte en una reducción más significativa de la impureza Gini será el más apropiado para la división de los datos.

Los dos criterios buscan un objetivo común: identificar particiones que generen subconjuntos cada vez más homogéneos, promoviendo así la sencillez y eficiencia del árbol final

EL ALGORITMO CART

El algoritmo CART (Clasificación y Árboles de Deducción) se destaca como una de las formulaciones más influyentes en el campo de los árboles de decisión, propuesto originalmente por Breiman, Friedman, Olshen y Stone en 1984. Contrariamente a otros métodos, como ID3 o C4.5, que posibilitan particiones con múltiples ramas, CART se distingue por la construcción constante de árboles binarios, es decir, cada nodo interno se segmenta en exactamente dos nodos hijo. Esta restricción simplifica la estructura del modelo y facilita su interpretación, al tiempo que lo hace particularmente adecuado como base para métodos más avanzados, como los bosques aleatorios o los algoritmos de gradient boosting (CART in Machine Learning, 2022).

CART puede ser empleado tanto en problemas de clasificación como en problemas de regresión. En ambos escenarios, el concepto esencial es idéntico: partir de un conjunto de entrenamiento y segmentarlo de manera recursiva en subconjuntos cada vez más homogéneos, implementando en cada nodo una regla de partición del tipo "si atributo es inferior o igual al umbral, va a la rama izquierda; en caso contrario, a la derecha".

CRITERIOS DE PARTICIÓN: IMPUREZA GINI Y ERROR CUADRÁTICO

El componente esencial de CART es el criterio que determina qué atributo y qué umbral aplicar en cada nodo para la segmentación de datos. En el contexto de la clasificación, el

algoritmo emplea el índice Gini. Si un nodo contiene ejemplos de N clases diferentes, con probabilidades pi(t) para cada clase i, el índice Gini viene dado por:

$$I(t) = 1 - \sum_{i=1}^{N} pi(t)^2$$

Un valor de Gini igual a 0 denota un nodo puro (todos los ejemplos pertenecen a una única clase), mientras que valores superiores indican la existencia de combinaciones de clases cada vez más heterogéneas (Clasificación con Árboles de Decisión: el algoritmo CART, 2021). Por ejemplo, un nodo con cinco ejemplos pertenecientes a la clase 1, diez pertenecientes a la clase 2 y veinticinco pertenecientes a la clase 3, presentará una impureza superior en comparación con un nodo con casi todos los casos pertenecientes a la misma categoría.

Cuando el algoritmo evalúa un potencial separador (por ejemplo, "longitud ≤ 0,5 m"), calcula la impureza Gini en los dos nodos hijo que resultarían en la partición (izquierda y derecha), y obtiene una función de coste como promedio ponderado de dichas impurezas:

$$Coste(t) = \frac{n_{izq}}{n} I\left(t_{izq}\right) + \frac{n_{der}}{n} I\left(t_{der}\right)$$

donde n es el número total de ejemplos en el nodo padre y n_{izq} , n_{der} el número de ejemplos en cada nodo hijo. El mejor separador será el que produzca el menor valor de esta función de coste, es decir, la combinación de nodos hijo más homogénea posible (Clasificación con Árboles de Decisión: el algoritmo CART, 2021).

En el caso de la regresión, CART reemplaza el índice Gini por medidas de error sobre un objetivo numérico, como la suma de cuadrados residual (RSS) o el error cuadrático medio (MSE). Para un nodo con valores observados y predicción $\hat{y}$, se tiene:

$$RSS = \sum (y - \hat{y})^2$$

$$MSE = \frac{1}{n}\sum(y - \hat{y})^2$$

El algoritmo selecciona el atributo y el umbral que minimizan el error generado en los nodos descendientes.

FUNCIONAMIENTO DE CART:

El funcionamiento del algoritmo CART puede entenderse como un proceso iterativo de divisiones del conjunto de datos. Su funcionamiento se basa en los siguientes pasos:

- Partir del conjunto de datos completo como nodo raíz.
- Evaluar cada atributo y, en el caso de variables numéricas, cada posible umbral de división.
- Calcular la función de coste asociada a cada partición candidata:
  - impureza Gini en problemas de clasificación,
  - medidas de error como RSS o MSE en problemas de regresión.
- Seleccionar la combinación atributo–umbral que minimiza dicha función.
- Dividir el conjunto de datos en dos nodos hijo binarios según la condición del split.
- Repetir el proceso de búsqueda del mejor split de forma recursiva en cada nodo hijo.
- Detener la expansión del árbol cuando se cumple un criterio de parada (profundidad máxima, tamaño mínimo del nodo o ausencia de mejora).
- Asignar la predicción en los nodos hoja:
  - clase mayoritaria en clasificación
  - valor medio de la variable objetivo en regresión.

Este método conlleva la evaluación de múltiples umbrales posibles para cada atributo, lo que caracteriza a CART como un algoritmo codicioso (greedy): en cada nodo se toma la decisión localmente óptima, la partición óptima en ese punto), sin realizar una revisión de decisiones previas. Esto lo convierte en un concepto sencillo y relativamente eficaz, no obstante, también justifica su susceptibilidad a variaciones pequeñas en los datos y el riesgo de sobreajuste si se permite el crecimiento del árbol sin restricciones (Clasificación con Árboles

de Decisión: el algoritmo CART, 2021). Para evitar ese posible sobreajuste existen herramientas como la poda.

PODA Y CONTROL DE COMPLEJIDAD

Un árbol CART que se deja crecer sin ningún límite establecido tiende a ajustarse en exceso al conjunto de entrenamiento. Para evitar este sobreajuste el algoritmo tiene mecanismos de poda y control de complejidad.

Una característica distintiva de CART es la poda por coste-complejidad, que combina en una única función tanto el error de clasificación como el tamaño del árbol. Dado un árbol T, se define:

$$R_{\propto}(T) = R(T) + \propto T_c(T)$$

donde:

• R(T) es el coste de error del árbol (por ejemplo, la tasa de ejemplos mal clasificados). Se define como R(T) = m / n, donde *m* es el número de errores y *n* el número total de ejemplos.

• T_c(T) es el número de hojas (nodos terminales) del árbol.

• α es un parámetro que penaliza la complejidad del árbol: valores altos de α favorecen árboles más pequeños.

El objetivo es contrastar el coste-complejidad de un árbol previo y posterior a la poda de un subárbol específico. Si al eliminar un conjunto de nodos (sustituyéndolos por una sola hoja) la función Rα(T) disminuye, la poda se considera beneficiosa y se acepta. Por consiguiente, se busca un balance entre precisión (bajo error de clasificación o regresión) y sencillez del modelo (menor número de hojas y reglas).

Además de la poda posterior, en el ejercicio práctico se suelen establecer hiperparámetros de parada durante el desarrollo del árbol, como:

- Profundidad máxima de la hoja (max_depth),

- Tamaño mínimo de nodo para la división (min_samples_split).
- Cantidad mínima de muestras por hoja (min_samples_leaf),
- Cantidad máxima de atributos contemplados en cada nodo (max_features).

Estos hiperparámetros facilitan la regulación del nivel de complejidad del árbol y la minimización de su varianza, potenciando la capacidad de generalización del modelo (CART Machine Learning, 2022).

CART APLICADO A LA CIBERSEGURIDAD:

El algoritmo CART (Classification and Regression Trees) puede emplearse para la detección de ciberataques siempre que se disponga de datos históricos etiquetados y de un conjunto de características cuantificables que describan el comportamiento del tráfico de red. En este contexto, el árbol de decisión aprende reglas lógicas que permiten distinguir entre tráfico legítimo y tráfico malicioso de forma interpretable

.

Entre los ataques que pueden identificarse mediante CART destacan, por ejemplo, los ataques de fuerza bruta, caracterizados por un número elevado de intentos fallidos de autenticación en un intervalo de tiempo reducido, así como los ataques de denegación de servicio, que se manifiestan mediante picos anómalos de tráfico o una saturación repentina de recursos. Otros comportamientos, como el escaneo de puertos o la exfiltración de información, también pueden detectarse si se describen adecuadamente mediante variables numéricas relevantes.

Un caso típico de aplicación de CART es la detección de ataques de fuerza bruta contra servicios de autenticación como SSH o paneles web de login. En este escenario, cada conexión o sesión se representa mediante un vector de características que puede incluir, entre otras:

- número de intentos de autenticación
- número de intentos fallidos

- tiempo medio entre intentos
- número de credenciales distintas probadas.

Cada registro de la base de datos se encuentra etiquetado previamente como tráfico normal o como ataque.

Durante el proceso de entrenamiento, el árbol de decisión selecciona, en cada nodo, la variable y el umbral que mejor separan los ejemplos normales de los ataques. Por ejemplo, el modelo puede aprender una regla inicial del tipo: *si el número de intentos fallidos supera un determinado umbral, existe una alta probabilidad de ataque*. A partir de ahí, el árbol refina la decisión incorporando nuevas condiciones, como el tiempo entre intentos, hasta llegar a hojas donde predomina claramente una de las dos clases.

Una vez entrenado, el árbol se utiliza en producción para clasificar nuevas conexiones. El sistema extrae las mismas características que se usaron en el entrenamiento y las introduce en el árbol, que evalúa secuencialmente las condiciones aprendidas hasta asignar una etiqueta final. Este enfoque resulta especialmente útil en ciberseguridad, ya que permite no solo detectar ataques, sino también explicar la decisión tomada mediante reglas claras y comprensibles para el analista.

VENTAJAS Y DESVENTAJAS DE CART:

La Tabla 3 siguiente resume las principales ventajas y limitaciones del algoritmo CART

VENTAJAS	LIMITACIONES
Alta **interpretabilidad**: el modelo genera reglas fácilmente comprensibles	Tendencia al **sobreajuste** si no se controla la profundidad del árbol
Permite **explicar cada decisión:** recorrido desde la raíz hasta la hoja puede reconstruirse paso a paso.	**Inestabilidad**: pequeños cambios en los datos de entrenamiento pueden dar lugar a árboles muy diferentes
Capaz de manejar **variables numéricas y categóricas**	Sensible al **ruido y a valores atípicos** si el árbol no está adecuadamente podado
Requiere **poco preprocesamiento** de los datos (ni normalización ni escalado)	Un árbol individual suele ofrecer **peor generalización** que métodos de ensamblado
Entrenamiento y clasificación **computacionalmente eficiente**	Las divisiones se basan en **umbrales rígidos**, lo que puede dificultar la detección de patrones graduales.

Tabla 3 - Ventajas y desventajas del algoritmo CART

3.3.4 MÉTODOS ENSEMBLE Y RANDOM FOREST

El aprendizaje por conjuntos, o ensemble learning se ha consolidado como una de las metodologías más influyentes en el ámbito del aprendizaje automático contemporáneo. A diferencia del enfoque convencional que se centra en la creación de un único modelo capaz de representar la totalidad de la estructura del fenómeno analizado, los métodos de ensemble parten de una idea distinta: la combinación de múltiples modelos individuales puede producir un predictor global más estable, más preciso y capaz de generalizar ante datos nuevos. IBM señala que el ensemble learning consiste en "agregar varios modelos base para mejorar el rendimiento, reducir errores y optimizar la capacidad de generalización frente a un único clasificador."

La base estadística que justifica la eficacia de estos métodos se vincula con el compromiso entre sesgo y varianza, uno de los problemas fundamentales en el aprendizaje supervisado. Un modelo aislado puede generar un elevado sesgo si su simplicidad es excesiva o una alta varianza si su complejidad es excesiva. Los enfoques de ensemble atenúan estas restricciones

mediante la combinación de modelos entrenados sobre diversas muestras, la ponderación de sus errores de manera iterativa o la incorporación secuencial de predictores débiles.

Estas características han fomentado la implementación de métodos como bagging, Random Forest, AdaBoost o Gradient Boosting, los cuales en la actualidad representan estándares industriales en tareas de clasificación y regresión con datos estructurados.

CONCEPTOS CLAVE:

Aprendiz base y lógica de la combinación de modelos:

En el contexto del ensemble learning, cada uno de los modelos individuales que son sometidos a entrenamiento y posteriormente incorporados al conjunto se conoce como aprendiz base (base learner). Este principio resulta crucial para comprender la arquitectura interna de los métodos de ensemble, dado que la calidad del modelo final se determina tanto por el desempeño individual de los aprendices como por la interacción entre ellos. Las definiciones ofrecidas por IBM especifican que un aprendiz base puede comprender cualquier algoritmo supervisado, desde árboles de decisión hasta modelos lineales o redes neuronales, cuya contribución se integrará posteriormente en un predictor global (Murel & Kavlakoglu, 2025).

Se establecen dos configuraciones generales: los conjuntos homogéneos y los conjuntos heterogéneos. Los primeros están formados por modelos del mismo tipo, como ocurre en métodos bien conocidos como Bagging, Random Forest o Gradient Boosting, en los que todos los aprendices son árboles de decisión construidos con diferentes subconjuntos de datos o distintas condiciones de entrenamiento. Los segundos, representativos de metodologías como el stacking, incorporan modelos de diversa naturaleza, como una SVM, un árbol de decisión y una red neuronal, con la finalidad de capturar estructuras complementarias del problema.

La eficacia de un conjunto no se limita al entrenamiento de múltiples modelos de idéntica naturaleza: es esencial que exista diversidad entre ellos. Si todos los aprendices generan predicciones casi idénticas, su combinación no aporta valor adicional. Esta diversidad puede ser producida mediante muestreos distintos de los datos, inicializaciones aleatorias o limitaciones en las variables empleadas por cada modelo, tal como se subraya en las descripciones técnicas proporcionadas por IBM (Murel & Kavlakoglu, 2020). Además, es imperativo que los aprendices de nivel inicial alcancen un rendimiento mínimo aceptable: deben ser modelos capaces de superar levemente la predicción por azar. Esto justifica la utilización de árboles de decisión poco profundos como: estos son modelos sencillos, rápidos de entrenar y suficientemente flexibles para capturar patrones útiles sin llegar a sobre ajustarse de forma extrema.

La combinación final de los aprendices puede manifestarse de diversas maneras. En algunos casos, el conjunto decide la clase mayoritaria predicha por los modelos individuales. En otros se utiliza la media o una combinación ponderada de las probabilidades estimadas por cada aprendiz.

El compromiso sesgo–varianza:

La eficacia del aprendizaje en equipo se comprende de forma más precisa cuando se analiza desde el marco del compromiso sesgo–varianza, que constituye uno de los fundamentos estadísticos del aprendizaje automático. Los métodos de ensemble surgen precisamente como una respuesta a este problema, al combinar múltiples modelos con el objetivo de mejorar la capacidad de generalización. En este contexto, técnicas como Bagging y Random Forest se orientan principalmente a la reducción de la varianza: parten de aprendices base inestables, entrenados de manera independiente sobre subconjuntos de datos obtenidos mediante muestreo aleatorio, y agregan sus predicciones a través de promedios o votaciones. Este procedimiento permite estabilizar el modelo final sin introducir un incremento significativo del sesgo.

Por otro lado, los métodos secuenciales, entre los que destacan Boosting, AdaBoost y Gradient Boosting, abordan el problema desde una la perspectiva complementaria, centrándose en la reducción progresiva del sesgo. Estos enfoques construyen el modelo de forma iterativa, incorporando sucesivamente aprendices débiles que se especializan en corregir los errores cometidos por los modelos anteriores. Esta dependencia secuencial crea de forma gradual la frontera de decisión, logrando una mayor precisión global. Aunque este proceso puede incrementar la varianza, dicho efecto se controla mediante mecanismos de regularización, como la tasa de aprendizaje, lo que permite alcanzar un equilibrio eficaz entre sesgo y varianza y explica el alto rendimiento de estos métodos.

BAGGING (Bootstrap aggregating)

El procedimiento de Bagging, acrónimo de bootstrap aggregating, representa uno de los métodos más directos y eficaces para reducir la varianza en modelos base inestables, como los árboles de decisión. Se basa en la combinación de dos operaciones: el muestreo bootstrap y la agregación de predicciones.

El procedimiento comienza con un conjunto de entrenamiento formado por n observaciones. A partir de él se generan L subconjuntos del mismo tamaño n, construidos mediante muestreo con reemplazo lo que permite introducir variabilidad entre los distintos modelos base sin necesidad de manipular la estructura original del dataset.

Tras conseguir estas muestras de bootstrap, se aplica un algoritmo base, comúnmente un árbol de decisión, debido a su rapidez y adaptabilidad a pequeñas alteraciones de los datos. De esta forma, los L árboles resultantes capturan patrones parcialmente distintos, lo que aumenta la diversidad del conjunto. La predicción final se determina mediante la combinación de las salidas de todos los problemas. En los problemas de regresión, el valor estimado por cada modelo se promedia, mientras que en los problemas de clasificación, generalmente se adopta la clase mayoritaria.

Otro elemento característico del *bagging* es la posibilidad de estimar el error *out-of-bag* (OOB). Dado que, en promedio, alrededor de un tercio de las observaciones no se incluye en cada muestra *bootstrap*, dichos datos pueden emplearse como un conjunto de validación interno. Este procedimiento evita la necesidad de aplicar técnicas adicionales ya que cada observación se evalúa exclusivamente con los modelos que no la incorporaron en el proceso de entrenamiento. Como resultado, se obtiene una estimación robusta del error de generalización con un menor coste computacional.

RANDOM FOREST:

El algoritmo denominado Random Forest puede ser concebido como una extensión del bagging específicamente diseñada para árboles de decisión, cuyo principal aporte radica en la incorporación de una capa adicional de aleatoriedad con el objetivo de disminuir la correlación entre los modelos base. En el *bagging* convencional, cada árbol se distingue exclusivamente por la muestra de datos empleada para su entrenamiento. En un Random Forest, en cada división del árbol NO se consideran todas las variables, solo un subconjunto aleatorio de ellas. Por ejemplo, si tu base de datos tiene 20 características, un árbol de un Random Forest puede elegir una división con solo 5 de ellas (seleccionadas aleatoriamente). Este mecanismo impide que siempre los mismos atributos dominen la estructura de todos los árboles, como suele ocurrir en un *bagging* simple, lo cual generaría modelos muy similares y, por tanto, menos eficaces al promediarse.

Otra característica significativa es la capacidad de calcular la importancia de las variables, ya sea mediante la disminución media de la impureza o a través de técnicas de permutación de importancia. Estas evaluaciones proporcionan una aproximación del rol de cada predictor dentro del conjunto.

BOOSTING:

Contrariamente al bagging, que habilita modelos de manera paralela y combina sus predicciones sin interacción entre ellos, el boosting se fundamenta en una construcción secuencial. Cada modelo del conjunto se enfoca particularmente en los errores cometidos por sus predecesores, de tal manera que cada nueva incorporación los corrige de manera progresiva. IBM explica este principio indicando que el boosting "combina múltiples aprendices débiles para formar un modelo robusto, en el que cada iteración busca mejorar las predicciones de la anterior" (Murel & Kavlakoglu, 2025). Esto lo ilustra la siguiente Figura 5.

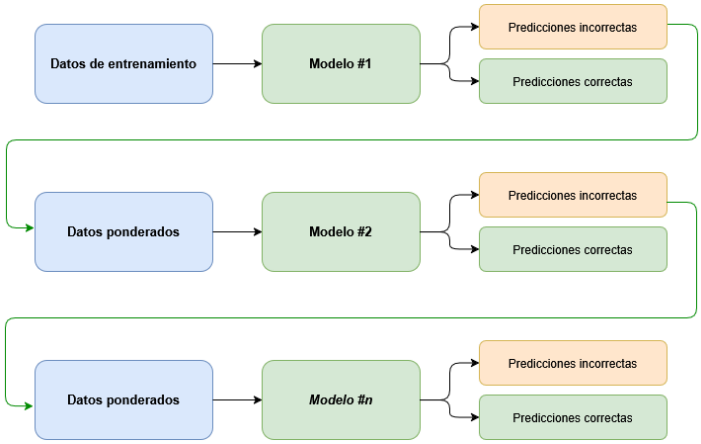

Figura 5 - Esquema del funcionamiento del método boosting

Por lo tanto, el propósito no radica en la reducción de la varianza como en el bagging sino en la disminución sistemática del sesgo.

El proceso se inicia con un aprendiz inicial débil, generalmente un árbol de decisión de escasa profundidad, que proporciona una aproximación inicial al problema. A continuación, se evalúan los errores cometidos por este modelo y se calcula una medida de discrepancia que identifica las regiones del espacio donde el modelo ha fallado. Los pesos son valores numéricos que indican cuánta importancia se asigna a cada observación en la siguiente iteración del modelo, de modo que aquellas en las que el modelo ha fallado reciben un peso mayor para forzar al algoritmo a prestarles más atención.

El siguiente modelo base se entrena específicamente para corregir estas deficiencias, ajustados. El proceso se repite múltiples veces y la predicción final se obtiene mediante la suma ponderada de todas las contribuciones.

El modelo AdaBoost destaca como uno de los casos más representativos de este mecanismo. En este algoritmo, las observaciones mal clasificadas en cada iteración adquieren un peso mayor en la siguiente, lo que obliga al árbol subsecuente a otorgarles una mayor relevancia. Los árboles correctamente ajustados, por su parte, obtienen un peso global proporcional a su precisión. Tras un número M de iteraciones, la combinación ponderada de todos los modelos genera un clasificador final cuyo rendimiento suele superar ampliamente al de cualquiera de sus componentes individuales.

Pese a su eficacia, el boosting conlleva un riesgo : cuando el número de iteraciones es excesivamente elevado o los árboles individuales exhiben una profundidad excesiva, el modelo puede sobreajustarse a los datos de entrenamiento. Por consiguiente, los algoritmos contemporáneos integran mecanismos explícitos de regularización, tales como la tasa de aprendizaje o el control riguroso de la profundidad de los árboles.

El Gradient Boosting constituye una generalización del boosting tradicional, donde la actualización iterativa del modelo se basa en un proceso de descenso por gradiente de funciones. En vez de enfocarse únicamente en los ejemplos mal clasificados como se observa en AdaBoost, el método Gradient Boosting toma en cuenta explícitamente la función de pérdida vinculada al problema, de manera que cada iteración busca disminuir dicha pérdida en la dirección marcada por su gradiente.

El procedimiento se inicia estableciendo una función de pérdida apropiada para la naturaleza de la tarea: el error cuadrático medio en regresión, la pérdida logística en clasificación, u otras alternativas. A partir de ella se inicia el modelo con una predicción simple, por ejemplo, la media de la variable objetivo que actúa como aproximación inicial. En cada iteración m, se calculan los pseudo-residuales, equivalentes al gradiente negativo de la pérdida respecto

a las predicciones actuales. Estos pseudo-residuales señalan la dirección en la que se debe ajustar el modelo para minimizar la pérdida. A continuación, se entrena un nuevo árbol de decisión de escasa profundidad para aproximar estos pseudo-residuales. El proceso continúa hasta alcanzar un número máximo de iteraciones o hasta que la reducción de la pérdida deje de ser significativa, incorporándose en muchos casos mecanismos para prevenir el sobreajuste.

La efectividad del Gradient Boosting está condicionada por un conjunto de hiperparámetros que requieren un ajuste para equilibrar el sesgo, la varianza y el coste computacional. Entre las características más significativas se encuentran:

- cantidad de iteraciones o árboles (n_estimators)
- tasa de aprendizaje (learning rate)
- complejidad de cada árbol
- profundidad máxima (max_depth)
- número máximo de hojas (max_leave_nodes).

Una cantidad significativa de árboles tiende a disminuir el sesgo, no obstante, incrementa la probabilidad de sobreajuste, particularmente cuando se combina con una elevada tasa de aprendizaje (η). Por ello, una práctica común consiste en emplear valores reducidos de η, habitualmente entre 0,01 y 0,1 acompañados de un número mayor de iteraciones, lo que estabiliza la convergencia del modelo.

ENSEMBLES APLICADO A LA CIBERSEGURIDAD:

Los métodos *ensemble* resultan especialmente adecuados en ciberseguridad, al permitir combinar múltiples modelos para mejorar la detección de ataques complejos y reducir las falsas alarmas. A partir de datos etiquetados y variables cuantificables del tráfico de red, estos enfoques pueden emplearse para identificar amenazas como ataques DDoS, exfiltración de información o actividad asociada a malware.

Un ejemplo habitual es el uso de técnicas de *bagging*, como *Random Forest,* para la detección de una exfiltración de datos. Este modelo seguiría estos pasos para obtener una decisión más robusta que la de un modelo único.

1. Datos y etiquetado: Se parte de registros históricos de tráfico etiquetados como normal o exfiltración.
2. Extracción de características numéricas: Cada flujo se representa como un vector de variables como por ejemplo: volumen de tráfico saliente, ratio salida/entrada, duración de la conexión, número de destinos externos distintos en una ventana temporal y "rareza" del destino (si el host/usuario no lo contacta habitualmente).
3. Construcción del *ensemble* (bagging): Se entrenan muchos árboles de decisión sobre subconjuntos distintos del conjunto de entrenamiento (muestreo *bootstrap*) y con subconjuntos aleatorios de variables. Así, cada árbol aprende reglas parcialmente diferentes sobre lo que caracteriza la exfiltración.
4. Evaluación de una nueva conexión: Ante un nuevo flujo, el modelo calcula su vector de características y lo evalúa en todos los árboles del conjunto. Cada árbol produce una predicción: normal o exfiltración.
5. Agregación y decisión final: La clasificación final se obtiene por votación mayoritaria. Por ejemplo, si la mayoría de árboles identifica el flujo como exfiltración, el sistema lo clasifica como ataque y genera la alerta correspondiente.

VENTAJAS Y DESVENTAJAS DE ENSEMBLES:

La siguiente Tabla 4 resume las principales ventajas y limitaciones del algoritmo Ensembles.

VENTAJAS	LIMITACIONES
Mejoran de forma la **exactitud y robustez** frente a modelos individuales.	La ganancia puede ser marginal si los modelos base no aportan diversidad
Menor sensibilidad a valores atípicos y perturbaciones en los datos.	Algunos métodos (ej: boosting) pueden amplificar el efecto del ruido.
Posibilidad de obtener métricas agregadas como la **importancia de variables**.	Generalmente presentan **baja interpretabilidad** respecto a modelos simples.
Requiere **poco preprocesamiento** de los datos (ni normalización ni escalado)	Requieren **mayor tiempo de entrenamiento e inferencia**.
Evitan recurrir a un único modelo extremadamente complejo.	Pueden resultar innecesarios para problemas sencillos

Tabla 4 - Ventajas y desventajas de Ensembles

3.3.5 REDES NEURONALES ARTIFICIALES (MLP)

Las redes neuronales artificiales constituyen uno de los modelos más importantes del aprendizaje automático actual. Su fundamento se inspira en el funcionamiento del sistema nervioso biológico, donde el conocimiento no se almacena de forma directa, sino que surge de la interacción entre neuronas conectadas. Esta idea se traslada al ámbito computacional mediante estructuras formadas por neuronas artificiales interconectadas, en las que el aprendizaje se produce a través del ajuste progresivo de los pesos que regulan dichas conexiones (UNIE Universidad, 2024).

Este modelo es particularmente atractivo en campos donde las interrelaciones entre variables son complejas, no lineales y difíciles de formalizar mediante reglas explícitas. La ciberseguridad representa un caso ejemplo de estas problemáticas: los ataques informáticos raramente se evidencian mediante un único indicador aislado, sino que suelen surgir como combinaciones sutiles de comportamientos, patrones de tráfico y señales contextuales. En este contexto, las redes neuronales facilitan la representación flexible de dichas

interacciones, capturando dependencias internas que se desvían de modelos lineales o fundamentados en reglas establecidas (Lee, 2025).

NEURONA ARTIFICIAL Y ARQUITECTURA BÁSICA:

El funcionamiento básico de una red neuronal se basa en la llamada neurona artificial. Cada neurona recibe varios valores de entrada, asigna a cada uno de ellos un peso, es decir, una determinada importancia y los combina para generar un único resultado. A este resultado se le aplica posteriormente una función de activación. En efecto, con esta se decide si la neurona se activa y en qué medida, produciendo así la salida que se transmite a la siguiente parte de la red.El uso de funciones de activación continuas y no lineales es esencial, ya que permite a la red representar relaciones complejas entre las variables de entrada y la salida deseadaAmazon Web Services, s. f.).

Una red neuronal completa se construye conectando múltiples neuronas y organizándolas en capas. De forma general, se distinguen tres tipos de capas:

- Una capa de entrada: encargada de recibir las variables explicativas
- Una o varias capas ocultas: responsables de transformar la información interna;
- Una capa de salida: que produce la predicción final.

Lo vemos ilustrado en la siguiente Figura 6:

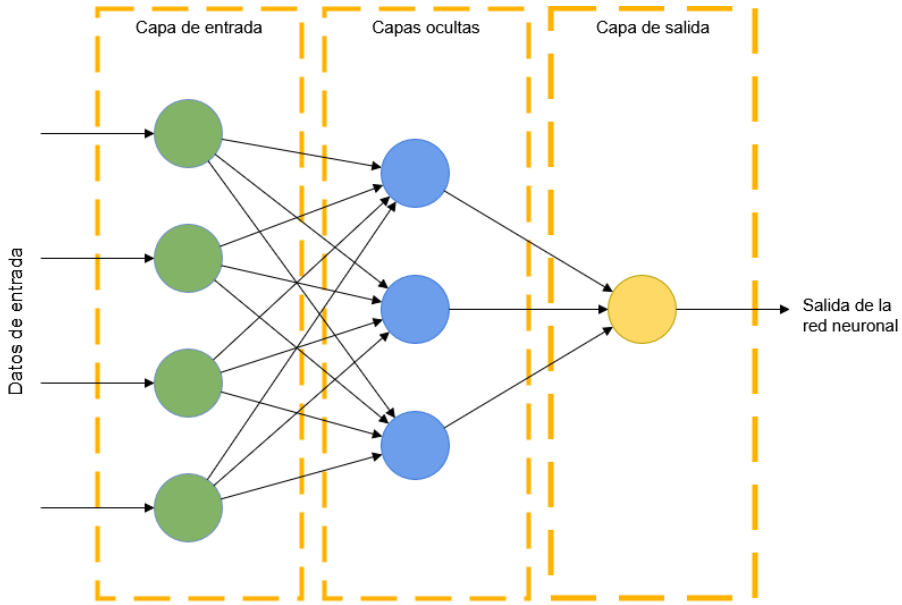

Figura 6 - Capas de un red neuronal

La profundidad de la red, que es el número de capas ocultas, y su anchura, definida por el número de neuronas en cada capa, determinan la capacidad del modelo para aproximar funciones complejas. A mayor profundidad y anchura, mayor capacidad de representación tiene aunque también aumenta el riesgo de sobreajuste y el coste computacional del entrenamiento (UNIE Universidad, 2024)

Las redes multicapa (MLP) y la superación de la linealidad

La implementación de redes neuronales multicapa, denominadas perceptrones multicapa o MLP (Multilayer Perceptrons), representó un progreso significativo en el campo del aprendizaje automático. Estas redes integran una o más capas ocultas entre la capa de entrada y la capa de salida, complementadas con funciones de activación no lineales, lo que facilita la transformación progresiva de los datos a través de la red.

Debido a estas transformaciones consecutivas, los datos originales pueden ser proyectados en espacios de representación internos, donde problemas inicialmente inseparables se

transforman en problemáticas abordables. Desde una perspectiva teórica, se ha demostrado que un MLP con al menos una capa oculta y un número adecuado de neuronas tiene la capacidad de aproximar cualquier función continua con la precisión requerida, lo que se conoce como el teorema del aproximador universal. Este principio justifica el uso extensivo de redes multicapa en tareas donde las relaciones entre variables son complejas y difíciles de modelar mediante enfoques tradicionales (UNIE Universidad, 2024).

En el ámbito de la ciberseguridad, esta capacidad resulta especialmente relevante. Un MLP tiene la capacidad de adquirir representaciones internas que combinan variables de tráfico, métricas temporales y atributos de comportamiento con el fin de diferenciar entre actividades legítimas y malintencionadas, incluso cuando dichas diferencias no son evidentes a nivel individual.

ENTRENAMIENTO Y ALGORITMO DE RETROPROPAGACIÓN

El entrenamiento de una red neuronal artificial constituye el núcleo de su funcionamiento y establece su capacidad predictiva. Implementar una red conlleva la modificación de los pesos y sesgos de todas las conexiones para minimizar la función de error o pérdida del modelo. Este procedimiento se fundamenta en un ciclo iterativo que combina la propagación hacia adelante, la estimación del error y la retropropagación.

Durante la etapa de propagación hacia adelante, las entradas se desplazan a través de la red de capa a capa. Cada neurona procesa una combinación ponderada de sus entradas y produce una salida tras la aplicación de la función de activación correspondiente. Esta función determina cómo responde la neurona ante la información recibida y cumple un papel fundamental, ya que introduce no linealidad en el modelo, permitiendo que la red neuronal aprenda relaciones complejas entre las variables. Entre las funciones de activación más utilizadas se encuentran:

- La función sigmoide, que genera salidas acotadas entre 0 y 1 y resulta habitual en problemas de clasificación binaria
- La tangente hiperbólica, que produce valores entre −1 y 1 y facilita un aprendizaje más equilibrado
- La función ReLU, que simplifica el cálculo y mejora la eficiencia del entrenamiento en redes profundas.

La predicción resultante se compara con el valor real a través de una función de pérdida apropiada para el tipo de problema, tales como el error cuadrático medio en regresión o la pérdida logística en clasificación (Lee, 2025).

La tasa de aprendizaje juega un papel crucial en este proceso. Esta tasa es un parámetro fundamental del entrenamiento de las redes neuronales, ya que determina la magnitud de los ajustes que se realizan en los pesos del modelo en cada iteración. En términos prácticos, controla la velocidad con la que la red aprende a partir de los datos y tiene un impacto directo tanto en la estabilidad del entrenamiento como en la convergencia hacia una solución adecuada (Lee, 2025). Los valores elevados pueden inducir oscilaciones y obstaculizar la convergencia, mientras que los valores demasiado bajos pueden ralentizar de manera excesiva el proceso de entrenamiento. En el ámbito práctico, se utilizan valores iniciales moderados, tales como 0.001 o 0.01, los cuales se ajustan en función del comportamiento de la función de pérdida y de la complejidad de la información (Lee, 2025).

NORMALIZACIÓN, CONVERGENCIA Y CAPACIDAD DE GENERALIZACIÓN

Previo a la implementación del entrenamiento, es imperativo emplear técnicas de preprocesamiento, particularmente la normalización o estandarización de las variables de entrada. Las redes neuronales exhiben una alta sensibilidad hacia las escalas de los datos, y la existencia de variables con rangos significativamente divergentes puede ocasionar dificultades en la convergencia o saturación de las funciones de activación. La normalización propicia un proceso de aprendizaje más estable y acelera la convergencia del modelo (Amazon Web Services, s. f).

La convergencia se logra cuando la función de pérdida logra estabilizarse y deja de disminuir de forma significativa tras un número suficiente de iteraciones. Sin embargo, una disminución constante en el error de entrenamiento no asegura un rendimiento óptimo en datos no vistos. El sobreajuste se manifiesta cuando la red adquiere detalles particulares del conjunto de entrenamiento, incluyendo ruido, y pierde capacidad de generalización.

Para evitar que la red neuronal aprenda de forma excesiva los datos de entrenamiento y pierda capacidad de generalización, se recurre a distintas estrategias de control del sobreajuste. En primer lugar, los datos se dividen en conjuntos de entrenamiento y validación, de modo que el modelo se ajusta únicamente con el primero, mientras que el segundo se utiliza para evaluar su comportamiento sobre datos no vistos. Durante el entrenamiento, se supervisa la evolución del error de validación, y el proceso se detiene cuando dicho error deja de disminuir o comienza a aumentar, indicando que el modelo está empezando a memorizar el conjunto de entrenamiento en lugar de aprender patrones generales. Adicionalmente, pueden aplicarse técnicas de regularización que penalizan valores excesivamente grandes de los pesos, así como limitar de forma intencionada la complejidad del modelo reduciendo el número de capas o neuronas. Estas medidas obligan a la red a aprender representaciones más simples y robustas, mejorando su capacidad para generalizar a nuevos datos (UNIE Universidad, 2024).

REDES NEURONALES APLICADAS A LA CIBERSEGURIDAD

Las redes neuronales artificiales pueden emplearse para la detección de ciberataques en escenarios donde los patrones de comportamiento son complejos y difíciles de describir mediante reglas explícitas. Su adaptabilidad y capacidad para modelar relaciones no lineales las hace especialmente adecuadas para identificar amenazas avanzadas y comportamiento malintencionado no puede ser caracterizado por normas explícitas.

En este contexto, es particularmente relevante examinar los falsos negativos, que conllevan ataques no identificados, y los falsos positivos, que pueden generar alertas innecesarias y sobrecargar los sistemas de respuesta (Amazon Web Services, s. f.).

Durante la fase de entrenamiento, la red neuronal se alimenta con datos históricos etiquetados que incluyen tanto tráfico normal como ejemplos de exfiltración previamente identificados. Mediante el ajuste iterativo de sus pesos internos, la red aprende combinaciones de características que resultan indicativas de un comportamiento anómalo o malicioso, incluso cuando cada variable individual no resulta especialmente sospechosa.

Una vez entrenada, la red neuronal puede desplegarse en un entorno de monitorización continua. Ante una nueva observación, el modelo evalúa el patrón completo de tráfico y produce una salida que representa la probabilidad de que dicho comportamiento corresponda a un proceso de exfiltración de información. Aunque la interpretabilidad de las redes neuronales es limitada en una herramienta especialmente valiosa en entornos con alto volumen de tráfico y amenazas sofisticadas, donde los ataques no siguen un comportamiento uniforme ni fácilmente predecible.

VENTAJAS Y LIMITACIONES:

Antes de aplicar redes neuronales a un problema concreto, resulta necesario ventajas y limitaciones. Aunque se trata de modelos muy potentes, su uso no siempre es la opción más adecuada, especialmente cuando se dispone de pocos datos o cuando la interpretabilidad es prioritaria. La siguienteTabla 5 - Ventajas y limitaciones de Redes Neuronales Tabla 5 resume los principales aspectos a considerar.

VENTAJAS	LIMITACIONES
Capacidad para modelar relaciones no lineales complejas entre variables.	Alto riesgo de sobreajuste si no se controla la complejidad del modelo.
Adecuadas para problemas donde los patrones no son evidentes.	Requieren un ajuste cuidadoso de hiperparámetros
Flexibilidad para abordar tareas de clasificación y regresión.	El proceso de entrenamiento puede ser costoso en tiempo y recursos
Buen rendimiento cuando se dispone de grandes volúmenes de datos	Menor interpretabilidad frente a modelos más simple
Robustas frente a pequeñas perturbaciones en los datos	Dificultad para justificar decisiones individuales del modelo

Tabla 5 - Ventajas y limitaciones de Redes Neuronales

3.3.6 SUPPORT VECTOR MACHINES (SVM)

Las Support Vector Machines (SVM) es un modelo muy característico del aprendizaje automático, especialmente valorado en tareas de clasificación y, en algunas variantes, de regresión. Su principal atractivo no radica en "encajar" los datos lo máximo posible, sino en construir una frontera de decisión que sea estable y generalice de manera efectiva. Esta estabilidad se fundamenta en un principio geométrico específico: si existen múltiples fronteras capaces de separar dos clases, es aconsejable seleccionar aquella que proporcione la separación más "holgada", es decir, que optimice el margen entre clases (GeeksforGeeks, 2025; Scikit-Learn developers, s. f.).

Esta lógica resulta particularmente adecuada en problemas como la ciberseguridad, donde la meta no se limita a acertar en el conjunto de entrenamiento, sino a identificar patrones en datos nuevos con el menor número posible de falsas alarmas y, sobre todo, con el menor número posible de ataques no detectados Las SVM proporcionan un método claro para alcanzar dicha robustez: establecer la frontera no a través de la mayoría de los puntos, sino a través de los casos críticos que definen la separación.

EL HIPERPLANO Y EL MARGEN

De forma simple lo que hace este modelo SVM es buscar un hiperplano que separe dos clases. Si estamos en dos dimensiones, ese hiperplano es una recta y en tres dimensiones, un plano. Lo importante no es el nombre como tal, sino su función: dividir el espacio de características en dos regiones, y que cada región se asocie a una clase (GeeksforGeeks, 2025).

La cuestión es que casi siempre existen muchos hiperplanos posibles. La SVM no selecciona una frontera de decisión cualquiera entre las posibles, sino aquella que maximiza el margen. El margen se define como la distancia geométrica mínima entre el hiperplano de separación y los puntos de entrenamiento más cercanos a él, pertenecientes a cada una de las clases. En otras palabras, es la anchura que el límite podría aumentar antes de alcanzar un punto de datos. Estos puntos, se denominan vectores soporte y son los únicos que determinan la posición de la frontera.

Maximizar el margen implica situar la frontera lo más alejada posible de los datos de ambas clases, creando una zona de separación que actúa como un "colchón de seguridad". Esta elección no tiene únicamente como objetivo el de clasificar correctamente los datos de entrenamiento, sino que también busca reducir la sensibilidad del modelo frente a pequeñas perturbaciones, ruido o variaciones en nuevas observaciones. En consecuencia, un margen mayor suele traducirse en una frontera más estable y en una mejor capacidad de generalización del modelo sobre datos nuevos.

LOS VECTORES SOPORTE:

El concepto de vector soporte constituye el componente fundamental que da nombre a las Support Vector Machines. Contrariamente a otros modelos de aprendizaje automático, en los cuales todos los datos de entrenamiento ejercen una influencia directa en la construcción de la frontera de decisión, en una SVM dicha frontera se establece exclusivamente a través

de un subconjunto limitado de observaciones. En concreto, son los puntos más cercanos al hiperplano de separación los que determinan su ubicación. Estos elementos son denominados vectores de soporte o vectores de soporte (GeeksforGeeks, 2025; Scikit-Learn developers, s. f.).

Desde una perspectiva teórica, esto implica que la frontera óptima no se ve afectada por la mayoría de los datos que están claramente distantes de la zona de separación entre clases. Únicamente los casos más próximos a esta frontera, que representan los casos más difíciles de categorizar, participan de forma activa en la definición del modelo. Esta característica distingue a las SVM de otras metodologías supervisadas y explica por qué se consideran modelos particularmente robustos ante fluctuaciones irrelevantes en los datos de entrenamiento.

Por lo tanto, se sostiene que las SVM son eficientes en el sentido que el modelo no requiere la memorización de todos los datos, sino únicamente aquellos casos límite que definen la distinción entre clases. Esta propiedad adquiere particular relevancia en problemas que requieren una frontera de decisión estable y claramente definida, incluso en situaciones donde el conjunto de datos es extenso o presenta redundancias (scikit-learn developers, s. f.).

La Figura 7 - Elementos fundamentales de una SVMFigura 7 muestra los componentes esenciales de una SVM: el límite de decisión central, los dos hiperplanos de margen para cada clase, el margen como la distancia de separación entre ellos, y los vectores de soporte.

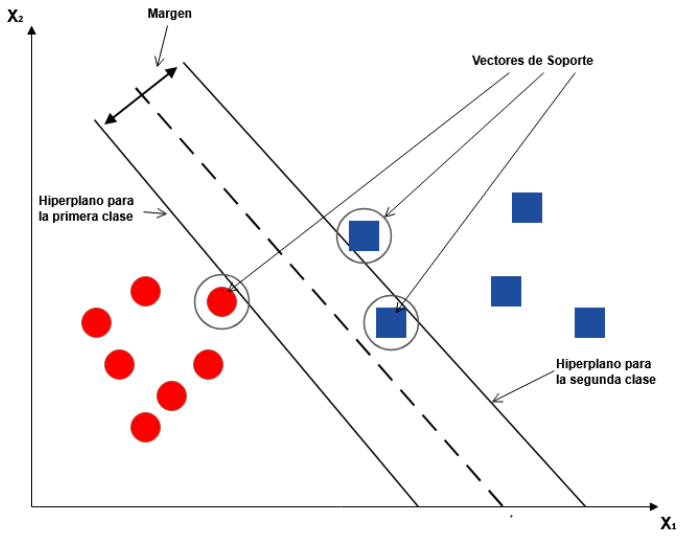

Figura 7 - Elementos fundamentales de una SVM

1. HARD MARGIN Y SOFT MARGIN:

En un escenario ideal, las clases estarían perfectamente separadas y se dispondría de un hiperplano capaz de clasificar todos los ejemplos de entrenamiento sin cometer errores. En este caso, podría implementarse un " hard margin" pues este exige que todos los puntos sean correctamente clasificados y situados fuera del margen de separación. Sin embargo, esta circunstancia rara vez se presenta en problemas reales y se es particularmente inusual en áreas como la ciberseguridad. En este contexto, los datos suelen verse afectados por ruido, etiquetas imprecisas, solapamiento entre comportamientos legítimos y malintencionados, así como por la presencia de valores atípicos u observaciones anómalas.

Para afrontar dichas limitaciones, las SVM integran el concepto de "soft margin" o margen suave, que permite relajar las restricciones del modelo. Bajo esta perspectiva, se admite que ciertos puntos puedan dentro del margen o incluso ser clasificados incorrectamente, siempre que ello contribuya a obtener una frontera de decisión más estable y con mejor capacidad de generalización. De esta forma el objetivo deja de ser una separación perfecta del conjunto de entrenamiento y pasa a centrarse en el rendimiento del modelo sobre datos nuevos (GeeksforGeeks, 2025).

El rol desempeñado por el parámetro de regularización C

El parámetro de regularización C juega un papel esencial es esencial es este modelo pues regula el compromiso entre dos objetivos opuestos:

- Maximizar el margen, buscando una frontera más simple y robusta.
- Penalizar los errores, intentando clasificar correctamente el mayor número de puntos posible.

Cuando el valor de C es alto, el modelo aplica una penalización más severa a los errores de clasificación, resultando en una frontera que intenta separar correctamente la máxima cantidad posible de puntos. Esto puede resultar adecuado en conjuntos de datos limpios, no obstante, también incrementa la probabilidad de sobreajuste cuando se presenta ruido o solapamiento entre clases. Sin embargo, valores reducidos de C permiten una mayor tolerancia a las infracciones del margen, resultando en fronteras más suaves que, a pesar de que pueden ocasionar más errores durante el entrenamiento, suelen generalizar de manera más efectiva en contextos ruidosos o complejos (GeeksforGeeks, 2025; Scikit-Learn developers, s. f.).

KERNELS Y EL *KERNEL TRICK*

Hasta este punto se ha supuesto que las clases pueden separarse mediante una frontera lineal. Sin embargo, esta hipótesis no se cumple en numerosos problemas reales. En efecto, en muchos conjuntos de datos, las observaciones de las distintas clases aparecen mezcladas de tal forma que una recta (o hiperplano) permite separarlas adecuadamente. Cuando esto ocurre, las Support Vector Machines recurren al uso de una herramienta muy potente: los kernels.

La idea fundamental puede entenderse de forma simple. En el espacio original de las variables, las clases no se pueden separar mediante una frontera lineal. Por lo tanto, en lugar de forzar una separación incorrecta, la SVM transforma los datos a un espacio de mayor

dimensión en el que sí es posible encontrar una separación lineal. Una vez realizada esta transformación, se entrena una SVM lineal en ese nuevo espacio. Este proceso se conoce como el kernel trick y la clave es que la transformación no se construye de manera explícita, sino que se realiza de forma indirecta mediante funciones kernel. Es importante destacar que no de esta forma no se asume el coste computacional de trabajar directamente en dicho espacio (GeeksforGeeks, 2025; scikit-learn developers, s. f.). Gracias a ello, las SVM pueden trabajar con problemas no lineales manteniendo una formulación matemática eficiente.

En la práctica, los kernels más utilizados son los siguientes:

- Kernel lineal : adecuado cuando la separación es aproximadamente lineal o cuando el número de variables es muy elevado y se busca un interpretable.
- Kernel polinómico : que introduce relaciones de orden superior entre las variables y resulta útil cuando la frontera no lineal presenta una estructura regular.
- Kernel RBF (gaussiano): es muy flexible ya que permite construir fronteras complejas basadas en la distancia entre observaciones.

Es importante subrayar que una mayor flexibilidad del kernel incrementa la capacidad del modelo para ajustarse a los datos, pero al mismo tiempo aumenta el riesgo de sobreajuste. Por ello, el ajuste adecuado de los hiperparámetros asociados al kernel resulta muy importante para obtener un buen rendimiento en datos no vistos.

Formulación de la SVM: interpretación del margen y del criterio de optimización

Para comprender qué es lo realmente optimiza una SVM, podemos apoyarnos en su interpretación geométrica, tal y como se ilustra en la siguiente Figura 8:

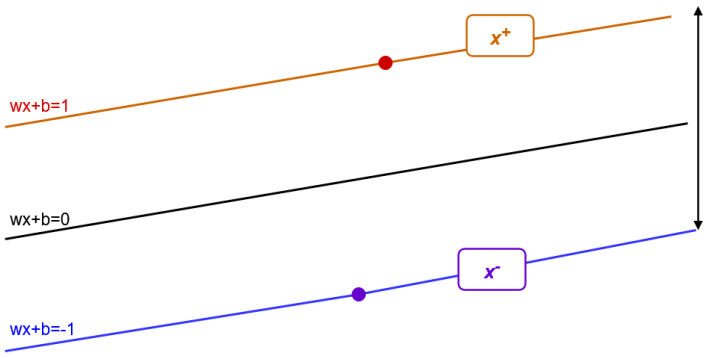

Figura 8 - Elementos geométricos de una SVM

En el caso lineal, la frontera lineal se define mediante un hiperplano representado de la siguiente manera:

$$w^T x + b = 0$$

donde el vector w determina la orientación del hiperplano y el término b controla su desplazamiento. A ambos lados de esta frontera se sitúan dos hiperplanos paralelos, definidos por las ecuaciones $w^T x + b = +1$ y $w^T x + b = -1$. Estos hiperplanos delimitan la región denominada como el margen.

Recordamos que tal y como muestra la figura, la anchura del margen se determina por los puntos de entrenamiento más cercanos al hiperplano central los vectores soporte. La distancia entre los dos hiperplanos extremos es proporcional a $\frac{2}{\|w\|}$. De esta forma, maximizar el margen equivale a minimizar la norma del vector w.

Desde esta perspectiva, el objetivo de la SVM consiste en encontrar el hiperplano que maximiza el margen, y por lo tanto que sitúa la frontera lo más alejada posible de los datos de ambas clases. Esto no busca únicamente una correcta clasificación del conjunto de entrenamiento, sino que también que la separación sea estable, es decir, menos sensible a pequeñas variaciones en los datos (GeeksforGeeks, 2025).

En el caso que los datos no sean perfectamente separables es cuando se introducen variables que ya hemos mencionado, como el parámetro de regularización C, que penaliza dichas violaciones. La SVM sigue manteniendo el objetivo de maximizar el margen, pero acepta excepciones controladas, pequeños errores, si con ello se mejora la capacidad de generalización del modelo. En conclusión, el modelo busca un equilibrio entre una frontera amplia y un número razonable de errores de clasificación (GeeksforGeeks, 2025; scikit-learn developers, s. f.).

En el caso de que los datos no sean perfectamente separables, se introducen variables de holgura ξ_i junto con el parámetro de regularización C, que penaliza las violaciones del margen. Estas variables de holgura cuantifican el error, el grado en que cada ejemplo incumple la restricción del margen si:

- $\xi_i = 0$: el ejemplo está correctamente clasificado y satisface el margen
- $0 < \xi_i < 1$: el ejemplo sigue estando correctamente clasificado, pero se está dentro del margen
- $\xi_i \geq 1$: el ejemplo queda mal clasificado

La SVM con margen blando se formula como un problema de optimización que minimiza $\frac{1}{2} \| \mathrm{w} \|^2 + C \sum_{i=1}^{N} \xi_i$, sujeto a la restricción $y_i(\mathrm{w}^\top \mathrm{x}_i + b) \geq 1 - \xi_i$, con $\xi_i \geq 0$. De esta forma, la SVM mantiene como objetivo principal la maximización del margen, pero acepta ciertas excepciones cuando la separación perfecta no es posible. Para controlar el nivel permitido se usa el parámetro C que regula el compromiso entre los dos objetivos:
Por un lado, valores elevados de C penalizan fuertemente las violaciones del margen, forzando al modelo a reducir los errores incluso a costa de un margen más estrecho. Por el otro, valores pequeños permiten más desviaciones, por lo que los márgenes son más amplios y las soluciones más robustas frente al sobreajuste. En consecuencia, el modelo busca un equilibrio entre una frontera de decisión amplia y un número razonable de errores de clasificación (GeeksforGeeks, 2025; scikit-learn developers, s. f.).

SVM EN LA DETECCIÓN DE CIBERATAQUES:

En problemas de detección de ciberataques, los conjuntos de datos suelen presentar dos características especialmente relevantes:

- Alta dimensionalidad, ya que el tráfico de red se describe con un gran número de variables relacionadas con: con paquetes, puertos, duraciones etc
- Fronteras de decisión poco evidentes, debido a un gran solapamiento entre actividades legítimas y maliciosas.

En este contexto, las SVM resultan particularmente muy útiles pues, por un lado, permiten utilizar la versión lineal cuando el preprocesamiento de los datos genera una separación razonable y clara. Por otro, también ofrecen la posibilidad de emplear kernels no lineales cuando la frontera es más compleja y requiere mayor flexibilidad. Además, el criterio de maximización del margen encaja muy bien con uno de los objetivos clave en ciberseguridad: reducir decisiones dudosas que puedan cambiar ante pequeñas variaciones en los datos, aumentando así la fiabilidad del sistema de detección (GeeksforGeeks, 2025).

El algoritmo SVM puede emplearse para detectar distintas categorías de ciberataques siempre que se disponga de datos históricos etiquetados y de un conjunto de características cuantificables que describan el comportamiento del tráfico de red. A diferencia de métodos basados en proximidad, SVM construye una frontera de decisión que separa de forma óptima el tráfico legítimo del malicioso en el espacio de características.

Un caso típico de uso es la detección de escaneos de puertos en una red corporativa. En este escenario, cada conexión de red se representa como un vector de características numéricas, que puede incluir variables como:

- número de puertos distintos contactados
- duración de las conexiones
- intervalo de tiempo entre intentos consecutivos
- número de destinos diferentes

- volumen de paquetes enviados.

Estas características permiten describir de forma objetiva el comportamiento de cada origen de tráfico.

A continuación, durante la fase de entrenamiento, SVM utiliza los datos históricos etiquetados para encontrar un hiperplano que maximiza la separación entre el tráfico normal y el tráfico malicioso. Mediante el uso de funciones kernel, el algoritmo puede proyectar los datos a un espacio de mayor dimensión, lo que le permite separar patrones complejos que no son linealmente separables en el espacio original.

Finalmente, cuando se observa una nueva conexión, el modelo evalúa su posición respecto a la frontera de decisión aprendida y la clasifica como tráfico legítimo o como ataque. Por ejemplo, si el patrón de conexiones de una dirección IP se sitúa claramente en la región asociada a escaneos de puertos, el sistema la identificará como actividad maliciosa. Este enfoque resulta especialmente eficaz para la detección de ataques bien definidos, ya que permite construir fronteras de decisión robustas a partir de ejemplos históricos sin necesidad de almacenar todos los datos de entrenamiento.

VENTAJAS Y LIMITACIONES SVM:

La siguiente Tabla 6 - Ventajas y limitaciones Tabla 6Tabla 4 resume las principales ventajas y limitaciones del algoritmo SVM.

VENTAJAS	LIMITACIONES
Buena capacidad de generalización gracias a la maximización del margen	Entrenamiento costoso en conjuntos de datos grandes
Funcionan bien en espacios de alta dimensionalidad	Ajuste delicado de hiperparámetros (kernel, C, γ).
El modelo depende solo de los vectores soporte	Interpretabilidad limitada, sobre todo con kernels no lineales
Permiten modelar fronteras no lineales mediante kernels	Sensibles al escalado de las variables de entrada
Robustas frente a pequeñas perturbaciones en los datos	No proporcionan probabilidades de forma directa

Tabla 6 - Ventajas y limitaciones

Capítulo 4. SISTEMA/MODELO DESARROLLADO

Este capítulo describe el trabajo experimental realizado. Se empieza por presentar el dataset CICIDS2017: su origen, su estructura, las clases que contiene y las decisiones de preprocesado aplicadas. A continuación, se explica la implementación de cada uno de los seis modelos evaluados : KNN, CART, Random Forest, MLP, SVM y Llama 3.2-1B, incluyendo el proceso de búsqueda de hiperparámetros, las decisiones metodológicas tomadas en cada caso y los resultados obtenidos individualmente. El objetivo es documentar el proceso dejando preparado el análisis comparativo del capítulo siguiente.

4.1 DESCRIPCIÓN DEL DATASET Y OBJETIVO DEL ESTUDIO

Para el desarrollo de la parte experimental se ha utilizado el dataset Network Intrusion Dataset, disponible en Kaggle y correspondiente al CIC-IDS 2017 (*Canadian Institute for Cybersecurity* Intrusion *Detection System Dataset*). Este conjunto de datos fue desarrollado por el Canadian Institute for Cybersecurity para proporcionar un entorno realista de evaluación de sistemas de detección de intrusiones basados en aprendizaje automático.

El dataset se construyó a partir de capturas reales de tráfico de red almacenadas en formato PCAP (*Packet Capture*), que registra el tráfico a nivel de paquete. Esas capturas se procesaron para extraer flujos de comunicación, generando archivos CSV listos para usar en modelos de clasificación.

En este trabajo se han empleado los archivos correspondientes a diferentes sesiones de actividad laboral simulada, entre ellas:

- Tuesday-WorkingHours
- Thursday-WorkingHours-Morning-WebAttacks
- Thursday-WorkingHours-Afternoon-Infiltration
- Friday-WorkingHours-Morning
- Friday-WorkingHours-Afternoon-PortScan

- Friday-WorkingHours-Afternoon-DDoS

Cada registro del dataset representa un flujo de comunicación entre dos extremos de red y está descrito mediante un conjunto amplio de 78 características estadísticas, tales como duración del flujo, número total de paquetes enviados y recibidos, volumen de bytes transmitidos, tasas de transferencia, métricas temporales, estadísticas derivadas de flags TCP y otras variables relacionadas con el comportamiento del tráfico. La lista completa de variables puede consultarse en el Anexo II.

La variable objetivo (Label) clasifica cada flujo en diez categorías: tráfico benigno (BENIGN) y nueve tipos de ataque:

- BENIGN**:** tráfico legítimo de red.
- DDoS: ataque de denegación de servicio distribuido
- PortScan: exploración sistemática de puertos para identificar servicios activos
- FTP-Patator**:** ataque de fuerza bruta contra el servicio FTP.
- SSH-Patator: ataque de fuerza bruta contra el servicio SSH.
- Bot**:** tráfico generado por software malicioso automatizado
- Infiltration: acceso no autorizado a la red con el objetivo de moverse lateralmente por los sistemas internos.
- Web Attack–Brute Force: intentos repetidos de acceso a aplicaciones web mediante prueba de credenciales.
- Web Attack–XSS**:** inyección de scripts maliciosos en páginas web para comprometer al usuario final.
- Web Attack–SQL Injection: manipulación de consultas SQL para acceder o modificar datos de una base de datos.
-

Esto permite abordar el problema desde una perspectiva de clasificación multiclase, lo que introduce mayor complejidad que un enfoque binario simple.

Se eligió este dataset principalmente porque reproduce escenarios de red realistas e incorpora distintos tipos de ataque en un mismo entorno. Además, el fuerte desbalanceo entre clases refleja bien lo que ocurre en sistemas reales, donde los eventos maliciosos son una fracción pequeña del tráfico total.

Una vez descrito el dataset, el objetivo de esta parte del trabajo es evaluar la capacidad de distintos modelos de aprendizaje automático para detectar ataques a partir de las características del tráfico. Se han entrenado y comparado seis algoritmos: K-Nearest Neighbors (KNN), CART, Random Forest, redes neuronales artificiales (MLP), Support Vector Machines (SVM) y el modelo de lenguaje Llama 3.2-1B. El criterio no es solo conseguir una accuracy alta, sino ver cuál generaliza mejor ante datos no vistos y cuál se comporta de forma más sólida frente a las clases minoritarias, que en este contexto son precisamente los ataques que más interesa detectar.

4.2 KNN

K-Nearest Neighbors es uno de los clasificadores más sencillos en cuanto a su lógica: como ya explicamos en el marco teórico para predecir la clase de un nuevo punto, busca sus K vecinos más cercanos en el conjunto de entrenamiento y asigna la clase más votada entre ellos.

Para la detección de ciberataques, KNN tiene algunas propiedades interesantes. Los ataques de una misma familia tienden a generar flujos estadísticamente similares entre sí: un escaneo de puertos produce muchos flujos cortos con características repetitivas, y los ataques de fuerza bruta generan series de intentos de conexión con patrones parecidos. Esta homogeneidad dentro de cada clase favorece, en principio, la clasificación por vecindad. Además, KNN no impone ninguna asunción sobre la forma de las fronteras de decisión, lo que puede ser útil cuando la distribución de las clases en el espacio de características es compleja o irregular.

La limitación más importante es el coste computacional en predicción. Cada nueva muestra debe compararse con todos los puntos del conjunto de entrenamiento para calcular distancias. Con más de un millón de instancias, esto hace inviable evaluar el modelo sobre el test completo en tiempos razonables. Como consecuencia, tanto la búsqueda del hiperparámetro K como la evaluación final se realizan sobre submuestras estratificadas, como se detalla más adelante.

KNN requiere normalización. Al trabajar con distancias, si las variables tienen escalas muy distintas, como por ejemplo, Flow Duration puede tomar valores del orden de microsegundos mientras que Destination Port varía entre 0 y 65535, las variables de mayor rango dominarían completamente el cálculo y las demás dejarían de contribuir.

1. **Importación de bibliotecas**

Se importan NumPy y Pandas para la manipulación de datos, glob para la localización de archivos, y Matplotlib y Seaborn para la visualización. Desde scikit-learn se incorporan KNeighborsClassifier, MinMaxScaler, make_pipeline, train_test_split y las métricas de evaluación habituales (accuracy_score, recall_score, f1_score, confusion_matrix, classification_report). Se usa permutation_importance de sklearn.inspection para el análisis de importancia de variables.

2. **Carga y preprocesamiento de los datos**

Los seis archivos CSV se localizan mediante glob y se concatenan en un único DataFrame. El resultado tiene 1.608.122 filas y 79 columnas: 78 características numéricas y la variable objetivo Label.

La inspección de valores nulos muestra que 286 registros tienen valor ausente en la variable Flow Bytes/s. El resto de las columnas no presenta ningún valor nulo. Dado que 286 instancias suponen menos del 0,02 % del total, se eliminan directamente.

Todas las variables explicativas son numéricas (int64 y float64). La única columna categórica es Label por lo que no se requiere codificación dummy.

También se reemplazan los valores infinitos por NaN, eliminando las filas afectadas. Tras la limpieza, el dataset queda con 1.606.989 instancias.

3. Normalización

```
# Remove leading/trailing whitespace from column names
df.columns = df.columns.str.strip()

# Separate features and target
X = df.drop('Label', axis=1)
y = df['Label']

# Replace infinite values with NaN, then drop affected rows
X = X.replace([np.inf, -np.inf], np.nan)
mask = X.notna().all(axis=1)
X = X[mask]
y = y[mask]

# Normalise features to [0, 1] — required for distance-based models like KNN
scaler = preprocessing.MinMaxScaler()
X_norm = pd.DataFrame(
    scaler.fit_transform(X),
    columns=X.columns
).reset_index(drop=True)
y = y.reset_index(drop=True)

X_norm.head()
```

En este script de Python se realiza la normalización para KNN. Se aplica MinMaxScaler que lleva cada variable al rango [0, 1]. La sustitución de infinitos y el ajuste del escalador también se hacen en este bloque.

4. División en entrenamiento y test

```
X_train, X_test, y_train, y_test = train_test_split(
    X,
    y,
    train_size   = 0.8,
    random_state = 1234,
    shuffle      = True,
    stratify     = y)
```

El dataset normalizado se divide en 80 % entrenamiento y 20 % test, con *stratify=y* para preservar la distribución de clases en ambos subconjuntos. El muestreo estratificado es especialmente necesario aquí: con clases que tienen 21 o 36 instancias en total, una partición aleatoria simple podría dejar alguna de ellas sin representación en el test. La semilla *random_state=1234* asegura la reproducibilidad.

5. Métricas de evaluación

Antes de entrar en los resultados conviene justificar por qué el F1-macro es la métrica de referencia principal en este trabajo, y no la accuracy.

En un dataset tan desbalanceado como CICIDS2017, un modelo que clasificase todo el tráfico como BENIGN alcanzaría una accuracy superior al 81 % sin haber detectado ningún ataque. La accuracy mide la proporción global de aciertos, pero cuando una clase concentra la gran mayoría de las instancias, ese porcentaje global queda casi por completo determinado por el rendimiento en esa clase mayoritaria.

Lo que más interesa en detección de intrusiones es el comportamiento del modelo frente a cada tipo de ataque. El recall mide qué fracción de los ataques reales de cada clase son identificados: un recall bajo implica falsos negativos, es decir, ataques que pasan sin generar ninguna alerta. El F1-score combina precision y recall en una sola métrica, penalizando tanto los falsos negativos como los falsos positivos. El F1-macro promedia el F1 de cada clase sin ponderación, de modo que una clase con 21 instancias tiene el mismo peso que BENIGN. Esto refleja mejor la capacidad del modelo para detectar todas las categorías de ataque, que es el objetivo real del sistema.

El F1-weighted pondera por el soporte de cada clase y está dominado por BENIGN y las clases más frecuentes. Puede dar una imagen inflada cuando el modelo funciona bien en las clases grandes pero falla en los ataques minoritarios. Se reporta como referencia, pero no es la métrica principal.

6. Búsqueda del valor óptimo de K

6.1. Submuestreo estratificado:

```
train_sample_n = 20000
test_sample_n  = 5000

if len(X_train) > train_sample_n:
    X_train_s, _, y_train_s, _ = train_test_split(
        X_train, y_train,
        train_size=train_sample_n,
        stratify=y_train,
        random_state=42
    )
else:
    X_train_s, y_train_s = X_train, y_train

if len(X_test) > test_sample_n:
    X_test_s, _, y_test_s, _ = train_test_split(
        X_test, y_test,
        train_size=test_sample_n,
        stratify=y_test,
        random_state=42
    )
else:
    X_test_s, y_test_s = X_test, y_test

print('Subsample sizes:', X_train_s.shape, X_test_s.shape)
```

En este script de Python hacemos una submuestra 20.000 instancias para entrenamiento y 5.000 para validación. Con el dataset completo, evaluar múltiples valores de K sería computacionalmente inviable. El muestreo estratificado preserva la distribución de clases, aunque las clases más minoritarias quedan con muy pocas instancias en la submuestra: Bot tiene 24, Web Attack – XSS 8, Infiltration 1.

El siguiente fragmento de código entrena el modelo KNN con distintos valores de k : 2, 3, 4, 5, 6, 7, 9, 11, 13, 14, 15, 16, 17, 18, 20, 21, 22, 23, 24, 25. evalúa su precisión en el conjunto de test para cada uno y representa gráficamente los resultados para identificar el valor de k que maximiza el accuracy.

```
k_range = [2, 3, 4, 5, 6, 7, 9, 11, 13, 14, 15, 16, 17, 18, 20, 21, 22, 23, 24,
25]
scores = []

for k in k_range:
    model = make_pipeline(
        MinMaxScaler(),
        KNeighborsClassifier(n_neighbors=k, n_jobs=-1)
    )
    model.fit(X_train_s, y_train_s)
    pred = model.predict(X_test_s)
    acc = accuracy_score(y_test_s, pred)
    scores.append(acc)
    print(f'k={k:>2} acc={acc:.4f}')

best_k = k_range[int(np.argmax(scores))]
print('\nBest k (accuracy):', best_k)

plt.figure()
plt.title('Accuracy vs k (subsample)')
plt.xlabel('k')
plt.ylabel('Accuracy')
plt.scatter(k_range, scores)
plt.xticks(k_range)
plt.show()
```

La razón de explorar un rango amplio es analizar cómo cambia el comportamiento del modelo con la cantidad de vecinos. Con K pequeño, el modelo es más sensible al entorno local de cada punto: puede capturar patrones locales de ataque con precisión, pero también es más susceptible al ruido y a instancias mal etiquetadas. Con K grande, las decisiones se suavizan y el modelo gana estabilidad, pero puede perder capacidad para separar ataques con patrones estadísticos similares a los del tráfico legítimo cercano.

6.2. Accuracy vs. K

Del código anterior, obtenemos valores de accuracy que oscilan entre 0,974 y 0,977 para todo el rango de K como muestra la siguiente Figura 9- Accuracy de KNN para distintos valores de kFigura 9.

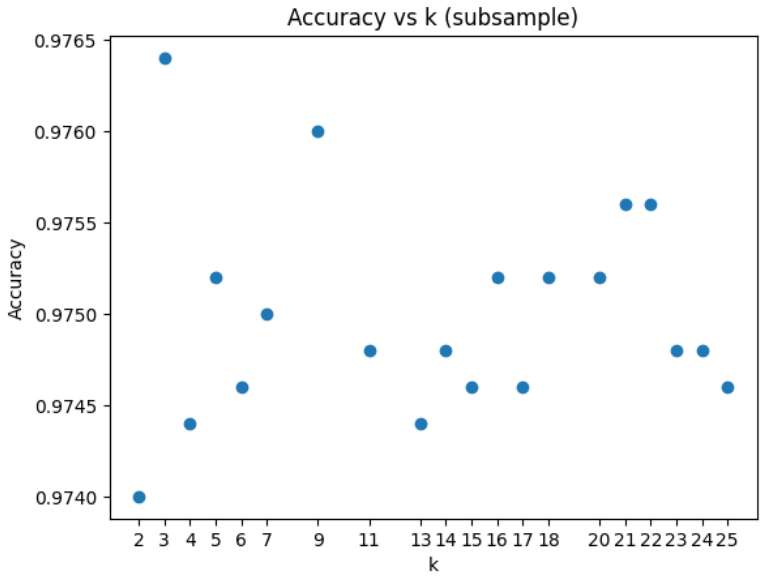

Figura 9- Accuracy de KNN para distintos valores de k

La diferencia máxima es de apenas 0,003 puntos. El mejor K por esta métrica es k=3, con accuracy de 0,9764, pero la elección entre k=3 y k=25 prácticamente no cambia el porcentaje de flujos bien clasificados. El gráfico lo ilustra bien: los puntos se agrupan en una banda estrecha sin tendencia clara. Esto confirma que la accuracy no discrimina entre configuraciones en este dataset y que una métrica más sensible al rendimiento por clase es necesaria.

6.3 Recall macro vs. K

Realizamos la misma búsqueda, pero cambiando la métrica al recall macro. La Figura 10 recoge el macro recall obtenido para cada valor de k evaluado.

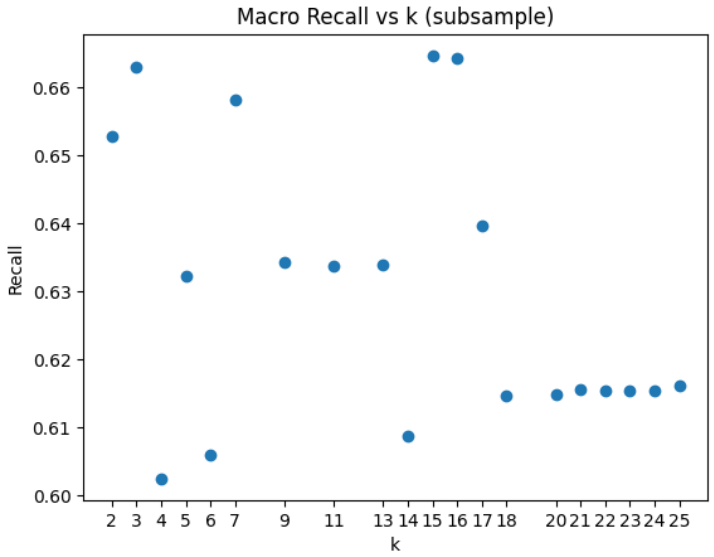

Figura 10 - Recall de KNN para distintos valores de k

Vemos una variación mucho mayor: desde 0,603 en k=4 hasta 0,665 en k=15. El mejor K por esta métrica es k=15. Se observa que los valores pares tienden a dar resultados peores que los impares adyacentes : k=4 cae a 0,602, k=6 a 0,606, un patrón relacionado con cómo funciona la votación por mayoría cuando hay empates entre dos clases con presencia similar en la vecindad, algo que ocurre con más frecuencia con K par.

6.4 F1-macro vs. K

Repetimos la búsqueda, pero cambiando la métrica a F1 macro. La Figura 11 recoge los resultados obtenidos para cada valor de k evaluado.

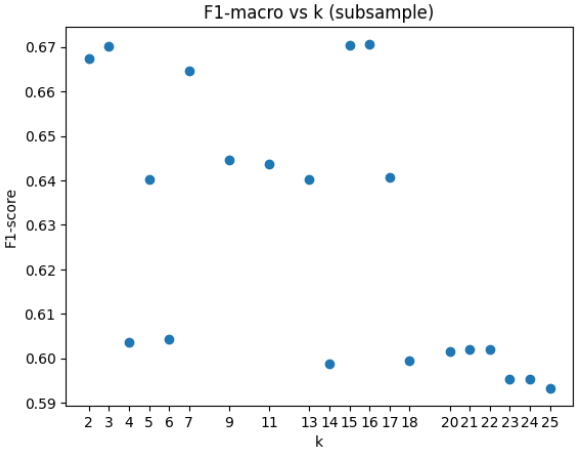

Figura 11 - F1-macro de KNN para distintos valores de k

El F1-macro refuerza el patrón del recall. Los mejores valores se obtienen en k=16 (F1=0,6706) y k=15 (F1=0,6703), seguidos de k=3 (F1=0,6702) y k=7 (F1=0,6647). En el extremo opuesto, k≥20 presenta una caída progresiva: con demasiados vecinos el modelo pierde resolución en las clases de ataque más inusuales, porque la clase correcta queda en minoría dentro de una vecindad grande dominada por el tráfico benigno.

La práctica coincidencia entre k=15 (mejor en recall) y k=16 (mejor en F1-macro) no es casual: ambas métricas apuntan al mismo rango de K, lo que da confianza en que el entorno de 15–16 vecinos es el más adecuado para equilibrar detección y precisión en este espacio de características.

6.5. Selección del modelo final: dos candidatos

El notebook implementa y evalúa dos modelos finales: k=3 (el mejor por accuracy) y k=15 (el mejor por recall macro y F1-macro). Esta comparación permite analizar concretamente qué se gana y qué se pierde al priorizar una métrica sobre otra. En ciberseguridad, la métrica más relevante es el F1-macro y, dentro de ella, el recall pues un ataque que no se detecta no genera alerta y la amenaza queda sin respuesta. Por eso el análisis de k=15 tiene algo más de peso, aunque se presentan los resultados de ambos.

7. Entrenamiento de los modelos finales

- Modelo 1: con k=3
- Modelo 2: con k=15

Ambos modelos se entrenan sobre la totalidad del conjunto de entrenamiento (1.285.591 instancias). La evaluación se realiza sobre las submuestras de 20.000 (train) y 5.000 (test) por las razones de coste computacional ya descritas.

8. Evaluación del modelo k=3

8.1. Sobre el conjunto de entrenamiento (submuestra 20.000)

Con k=3, el modelo alcanza accuracy del 99,53 % sobre la submuestra de entrenamiento y un macro avg del F1 de 0,94. La diferencia respecto al weighted avg de 1,00 refleja que las clases minoritarias tienen rendimiento inferior al promedio ponderado, como es habitual.

A continuación, analizamos la matriz de confusión reflejada en la Figura 12.

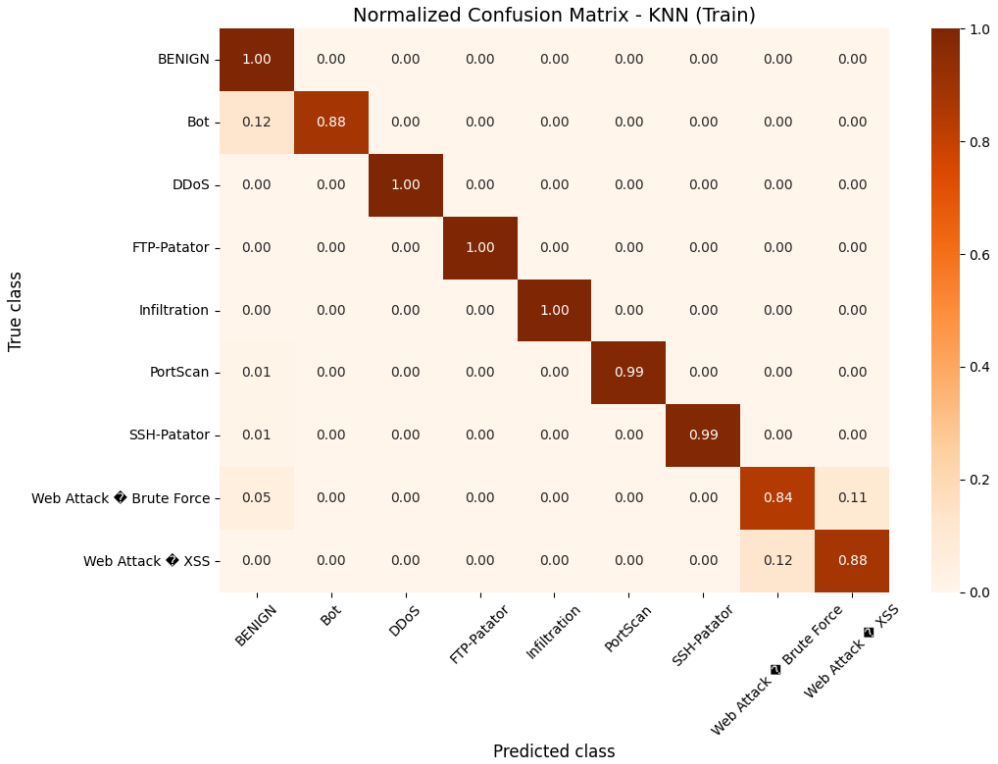

Figura 12 - Matriz de confusión normalizada del modelo KNN sobre el conjunto de entrenamiento

Las clases más representadas funcionan bien. BENIGN, DDoS y FTP-Patator alcanzan F1=1,00. SSH-Patator obtiene F1=0,99 y PortScan F1=0,98. Sus patrones estadísticos son suficientemente diferenciados como para que KNN los clasifique correctamente en sus vecindades locales.

Los casos más interesantes son las clases pequeñas.

- Bot : obtiene F1=0,86 con recall de 0,88: de las 24 instancias de Bot en la submuestra, 3 se clasifican como BENIGN. Con tan pocas instancias, 3 errores suponen el 12,5 % de la clase.
- Web Attack–Brute Force : alcanza F1=0,86 con recall=0,84 y precision=0,89: detecta 16 de 19 instancias.
- Web Attack–XSS **:** obtiene recall=0,88 con precision=0,70 y F1=0,78: detecta 7 de 8, pero clasifica algunos flujos de Brute Force como XSS.

La confusión entre Brute Force y XSS en entrenamiento es un patrón que se repetirá en test y en otros modelos. Ambos son ataques web que comparten características estadísticas de flujo similares: duración de sesión, número de paquetes, volúmenes de bytes, lo que dificulta su separación independientemente del clasificador utilizado.

9.2. Sobre el conjunto de test (submuestra 5.000)

Sobre la submuestra de test, el modelo obtiene accuracy del 98,78 % y macro avg del F1 de 0,85. La caída desde 0,94 en entrenamiento confirma que algunas clases son más difíciles de generalizar, pero no hay señales de un gran sobreajuste.

A continuación, analizamos la matriz de confusión de la prueba ilustrada en la Figura 13.

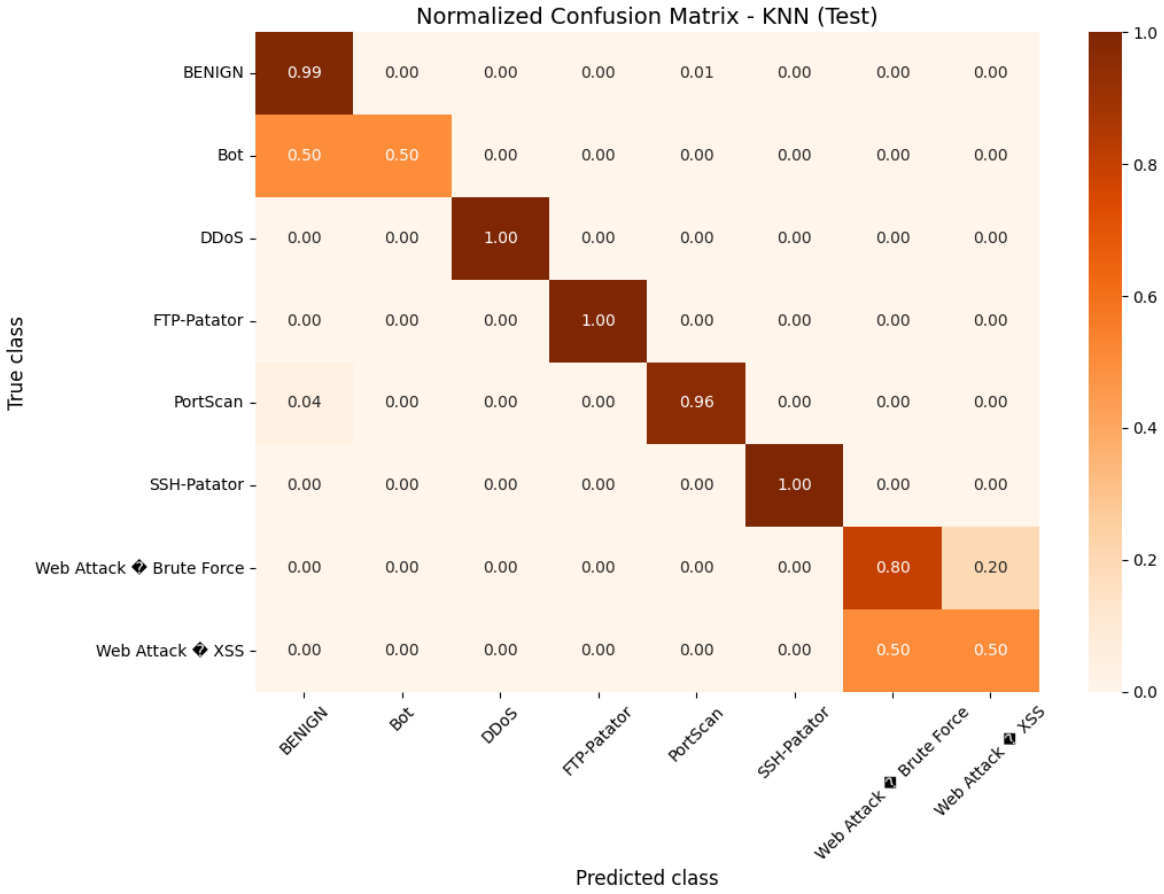

Figura 13 - Matriz de confusión normalizada del modelo KNN sobre el conjunto de test

Observamos que:

- BENIGN : mantiene F1=0,99: de las 4.052 instancias benignas del test, 4.016 se clasifican correctamente. Los 36 errores se distribuyen principalmente como PortScan (34), con 1 como Bot y 1 como DDoS. Desde la perspectiva de un IDS, 36 falsas alarmas sobre tráfico legítimo en una muestra de 4.000 instancias son perfectamente manejables operativamente.

- SSH-Patator también F1=1,00. Estas clases tienen firmas estadísticas muy consistentes que KNN reproduce correctamente incluso en datos no vistos.

- PortScan obtiene F1=0,95 con recall de 0,96: 474 de los 494 escaneos son detectados correctamente. Los 20 no detectados se clasifican como BENIGN. En la dirección contraria, 34 flujos benignos se clasifican como PortScan, un patrón de confusión bidireccional que refleja que ciertos flujos legítimos con muchas conexiones cortas, como actualizaciones automáticas, sincronizaciones, tienen características similares a las de un escaneo ligero.

- Bot es el caso más preocupante. Con 6 instancias en el test, el modelo detecta solo 3 (recall=0,50, F1=0,60). Los 3 no detectados se clasifican como BENIGN. Con tan pocas instancias la estimación no es estadísticamente robusta, pero el patrón es claro: KNN tiene dificultades con esta clase. En un entorno real, un sistema que falla en la mitad de los ataques Bot no es fiable para esa amenaza. Los bots suelen actuar en fases de reconocimiento y movimiento lateral dentro de una red, por lo que no detectarlos puede tener consecuencias más allá del incidente individual.

- Web Attack – Brute Force obtiene F1=0,80 con precision y recall de 0,80: detecta 4 de 5 ataques, con 1 falso positivo.

- Web Attack – XSS con 2 instancias obtiene F1=0,50: 1 detectado correctamente y 1 clasificado como Brute Force. Como ya ocurría en entrenamiento, la confusión entre estos dos subtipos es la más frecuente. Desde la perspectiva de la seguridad, confundir un Brute Force con un XSS, o al revés, es menos grave que no detectarlo pues en ambos casos el sistema genera una alerta y el analista puede investigar. El error más peligroso es siempre la clasificación hacia BENIGN, que no genera ninguna señal.

9. Evaluación del modelo k=15

10.1. Sobre el conjunto de entrenamiento

Los resultados de entrenamiento de k=15 son prácticamente idénticos a los de k=3 como se ve en la siguiente matriz de confusión (Figura 14):

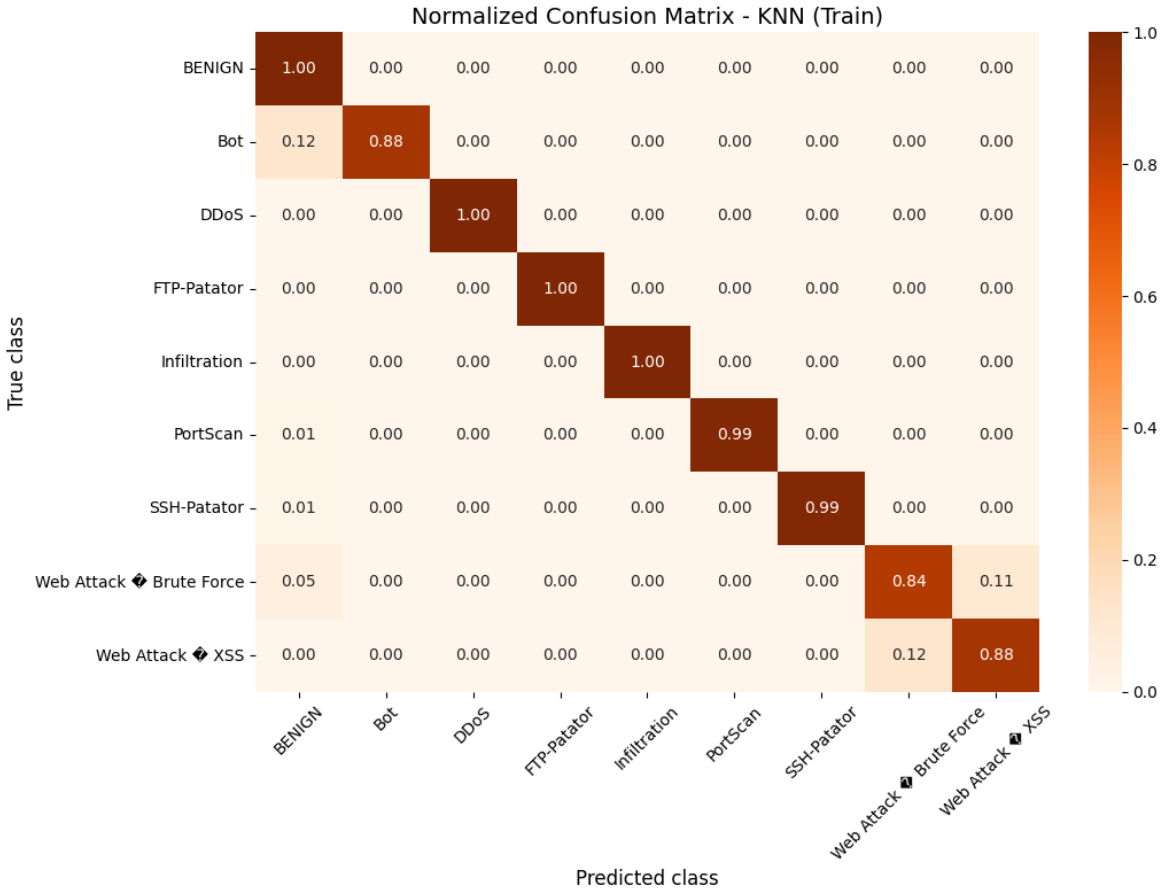

Figura 14 - Matriz de confusión normalizada del modelo KNN sobre el conjunto de entrenamiento (k=15)

Las matrices de confusión muestran el mismo patrón: 3 instancias de Bot clasificadas como BENIGN, confusión entre Brute Force y XSS. Con una submuestra de 20.000 instancias y regiones del espacio densamente pobladas por las clases mayoritarias, la diferencia entre consultar 3 o 15 vecinos no cambia sustancialmente las predicciones.

10.2. Sobre el conjunto de test

La evaluación de test revela diferencias más claras. La accuracy baja ligeramente a 98,36 % y el macro avg del F1 cae a 0,78, frente al 0,85 de k=3. Pero el resultado no es uniforme por clases como refleja la siguiente matriz de confusión (Figura 15):

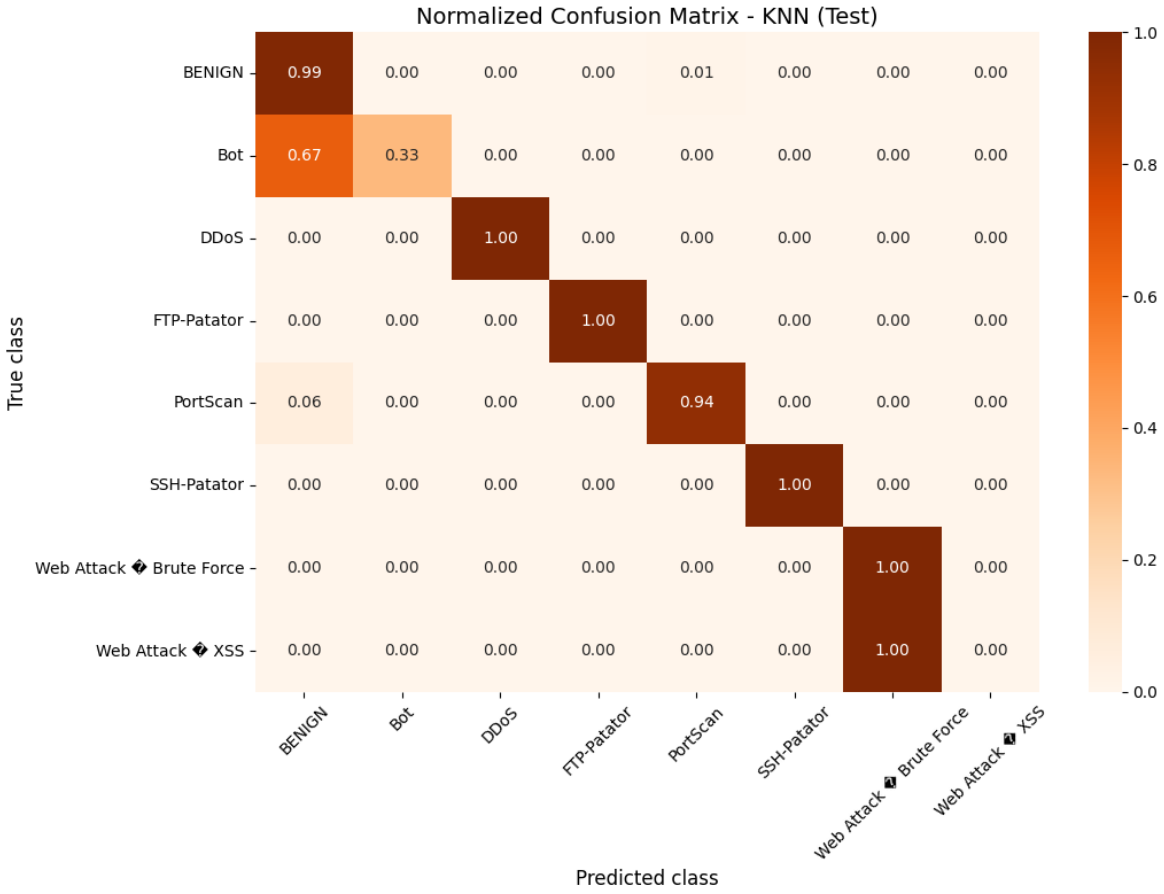

Figura 15 - Matriz de confusión normalizada del modelo KNN sobre el conjunto de test (k=15)

Observamos que:

- Bot empeora: recall=0,33 con k=15 frente al 0,50 de k=3. Solo detecta 2 de los 6 ataques. La precision es 1,00 : los que detecta, los detecta sin error, pero detecta menos.
- El caso más llamativo es Web Attack – XSS: k=15 obtiene F1=0,00. Las 2 instancias de XSS en el test son clasificadas erróneamente (2 como Brute Force). Con solo 2 instancias el resultado no es estadísticamente significativo, pero ilustra el riesgo de K grande con clases muy minoritarias: al consultar 15 vecinos, la clase correcta puede quedar en minoría aunque la instancia sea genuinamente XSS.

- Web Attack – Brute Force mejora con k=15: recall=1,00 y F1=0,83 frente al 0,80 de k=3. Detecta todos los ataques de la clase, aunque con precision de 0,71 (algunos falsos positivos).

- PortScan baja ligeramente a F1=0,93 frente al 0,95 de k=3.

11. Comparación entre k=3 y k=15

Métrica	k=3	k=15
Accuracy test	98,78 %	98,36 %
Macro avg F1	0,85	0,78
Macro avg Recall	0,84	0,78
F1 Bot	0,60	0,50
F1 Web Attack – XSS	0,50	0,00
F1 Web Attack – Brute Force	0,80	0,83
F1 PortScan	0,95	0,93
F1 DDoS	1,00	1,00
F1 FTP-Patator	1,00	1,00
F1 SSH-Patator	1,00	1,00

Tabla 7 - Comparación de resultados entre k=3 y k=15

Esta Tabla 7 muestra claramente como k=3 supera a k=15 en la mayoría de los indicadores sobre esta submuestra de test. La ventaja es más pronunciada en las clases minoritarias (Bot, XSS), donde k=15 tiene más dificultades para construir una mayoría correcta en la vecindad. La excepción es Brute Force, donde k=15 es marginalmente mejor.

Hay que tener cuidado al interpretar estas diferencias: con 2 instancias de XSS y 6 de Bot en el test, la varianza de las métricas es muy alta y las diferencias observadas pueden reflejar aleatoriedad más que una diferencia real en la capacidad del modelo. Lo que sí es consistente en ambos modelos es que las clases con mayor soporte (DDoS, FTP-Patator, SSH-Patator) se detectan perfectamente con cualquier valor de K, mientras que las minoritarias son las que concentran los errores.

12. Importancia de las características

La siguiente Figura 16 muestra la importancia de las variables calculada mediante permutation importance para el modelo KNN, representando la caída media de accuracy que se produce al permutar aleatoriamente cada característica. Cuanto mayor es dicha caída, mayor es la contribución de la variable al proceso de clasificación del modelo.

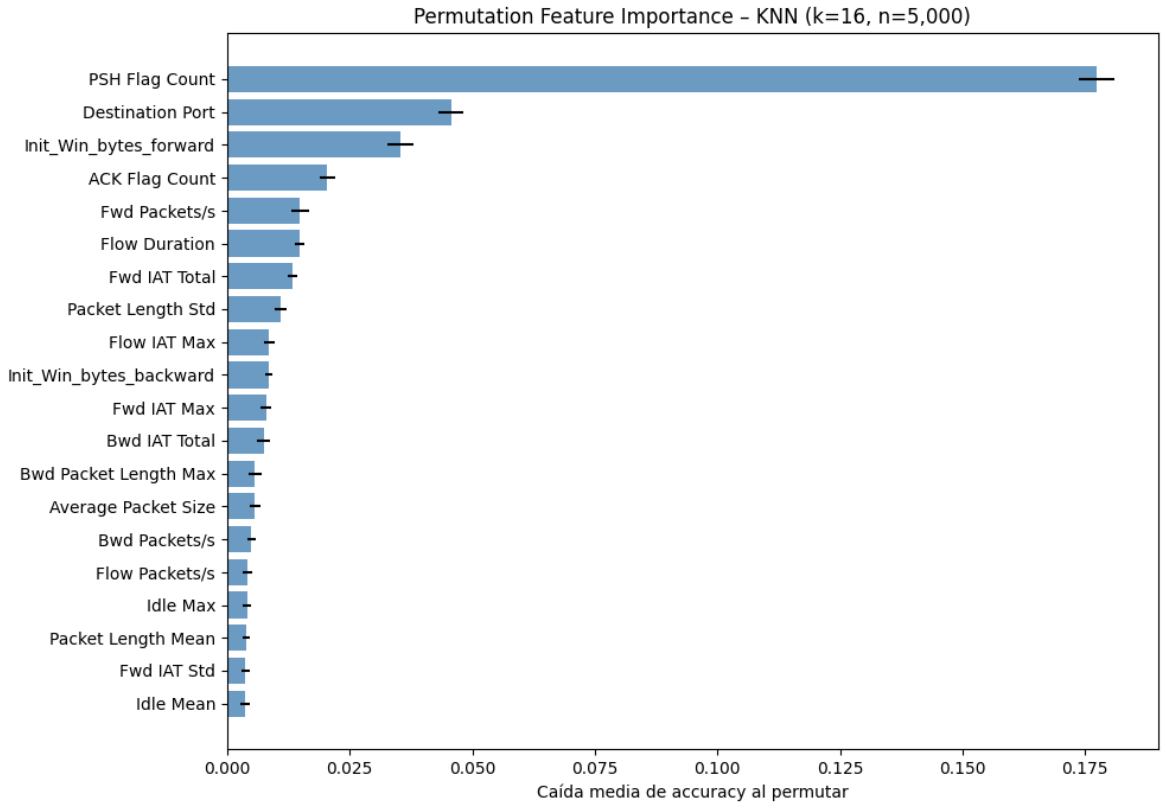

Figura 16 - Importancia de las variables del modelo KNN obtenida mediante permutation importance

Las variables más relevantes son:

PSH Flag Count es con diferencia la más importante, con una caída de 0,177 al permutarla. Esta variable cuenta cuántos paquetes del flujo llevan el flag PSH activo, que en TCP indica que los datos deben entregarse inmediatamente sin esperar a llenar el buffer. En tráfico normal su valor es bajo y variable. Los ataques de escaneo no generan intercambio real de datos y tienen PSH=0 en todos los paquetes; los de fuerza bruta producen ráfagas donde aparece en casi todos. Ese patrón extremo en ambas direcciones es lo que lo hace tan discriminativo.

Destination Port ocupa la segunda posición pues el puerto determina el servicio atacado y es muy discriminativo para clases como FTP-Patator (puerto 21) o SSH-Patator (puerto 22).

Init_Win_bytes_forward es el tamaño de ventana TCP que el emisor anuncia en el SYN inicial. Los navegadores y sistemas operativos modernos usan valores altos para maximizar el rendimiento, mientras que las herramientas de ataque suelen usar valores fijos bajos o los que trae la librería de red por defecto, sin preocuparse por la eficiencia de la transferencia. Esa diferencia sistemática es lo que hace a esta variable informativa para el modelo.

A partir de la cuarta variable, las caídas son pequeñas y bastante uniformes. Vemos como en KNN el peso se concentra fuertemente en las dos o tres primeras variables. Esto probablemente refleja la naturaleza local del algoritmo: en la mayoría de las vecindades, las diferencias en PSH Flag Count y en el puerto ya son suficientes para identificar la clase correcta, y el resto de las variables contribuyen solo en las regiones donde estas dos no son discriminativas.

13. Conclusiones sobre el modelo KNN

KNN funciona bien en las clases con patrones estadísticos definidos y consistentes: DDoS, FTP-Patator y SSH-Patator alcanzan F1=1,00 en test con ambos valores de K, y BENIGN se mantiene en 0,99. Para un sistema que necesite cubrir principalmente estos tipos de amenaza, KNN es una opción eficaz y fácil de implementar.

Las limitaciones se concentran en las clases menos representadas. Bot tiene un recall del 50 % en el mejor caso, y XSS es prácticamente indetectable con k=15. Con tan pocos ejemplos reales en la submuestra de entrenamiento, las vecindades de KNN no son suficientemente representativas de estas clases.

La limitación estructural más importante sigue siendo la escalabilidad. Con más de un millón de instancias, la evaluación sobre el test completo no es viable y la búsqueda de hiperparámetros queda restringida a submuestras. En un entorno de producción con tráfico continuo de red, esto representa un obstáculo real que los modelos basados en estructuras aprendidas, como CART, Random Forest o SVM, no tienen en la misma medida.

Por último, la ausencia de tratamiento del desequilibrio de clases (no se aplica SMOTE aquí) hace que las clases más minoritarias tengan escasa representación en la submuestra de entrenamiento. Los modelos siguientes incorporan SMOTE, lo que permitirá ver directamente qué aporta el sobremuestreo en términos de detección de ataques poco frecuentes.

4.3 CART

CART (*Classification and Regression Trees*) es un algoritmo de árbol de decisión que construye un clasificador mediante divisiones binarias recursivas. Como ya explicado en la parte teórica en cada nodo, el árbol selecciona la variable y el umbral que mejor separan las clases según el índice de Gini y repite el proceso hasta cumplir alguna condición de parada. El resultado es un árbol donde cada hoja asigna una etiqueta de clase.

La ventaja es que la estructura es muy fácil de interpretar pues se puede seguir el camino de decisión de cualquier predicción de nodo en nodo. En ciberseguridad esto tiene valor práctico, porque saber qué combinación de variables activó una alerta ayuda al analista a validarla o descartarla.

El problema habitual de los árboles sin restricciones es que tienden a crecer demasiado, memorizando el ruido del conjunto de entrenamiento. Para controlarlo se aplica Cost-Complexity Pruning (CCP), que penaliza el tamaño del árbol durante el entrenamiento. El parámetro que controla esta penalización es ccp_alpha: cuanto mayor es su valor, más se poda el árbol y más simple queda.

1. **Importación de bibliotecas**

Desde scikit-learn se usa DecisionTreeClassifier como clasificador, GridSearchCV para la búsqueda del hiperparámetro de poda y permutation_importance para la importancia de variables. Se añade SMOTE de imbalanced-learn para el balanceo de clases y Counter de collections para inspeccionar la distribución antes y después del sobremuestreo.

2. **Carga y limpieza de datos**

El proceso de carga y limpieza sigue el mismo procedimiento descrito en KNN. Tras concatenar los seis archivos CSV, limpiar los nombres de columna, sustituir los valores infinitos por NaN y eliminar las filas afectadas, el dataset queda con 1.606.989 instancias y 78 variables.

3. División en entrenamiento y test

Se reserva el 80 % para entrenamiento (1.285.591 instancias) y el 20 % para test (321.398 instancias). Esto se realiza antes de aplicar SMOTE ya que si se aplicase primero y luego se dividiese, las instancias sintéticas generadas a partir de muestras del test podrían acabar en el entrenamiento y esto invalidaría la evaluación.

4. Balanceo de clases con SMOTE

El siguiente fragmento de código aplica SMOTE de forma controlada, aumentando únicamente las clases con menor representación hasta alcanzar un umbral máximo de 10.000 instancias por categoría:

```
y_train_1d = np.ravel(y_train)
counts = Counter(y_train_1d)

print("Distribución ANTES de SMOTE:")
print(counts)

# Submuestrear hasta cap solo las clases que estén por debajo
cap = 10_000
sampling_dict = {cls: cap for cls, c in counts.items() if c < cap}
print("Sampling strategy:", sampling_dict)

# k_neighbors seguro (no puede superar min_class - 1)
min_class = min(counts.values())
k = min(3, max(1, min_class - 1))
print(f"k_neighbors = {k}")

sm = SMOTE(sampling_strategy=sampling_dict, random_state=2, k_neighbors=k)
X_train_res, y_train_res = sm.fit_resample(X_train, y_train_1d)

print("Distribución DESPUÉS de SMOTE:")
print(Counter(y_train_res))
print(f"Shape train tras SMOTE: {X_train_res.shape}")
```

En efecto, antes de entrenar el modelo es necesario abordar el desequilibrio de clases. Como hemos visto en KNN, un modelo entrenado sobre esta distribución tiende a ignorar las clases raras porque los errores en ellas apenas afectan al coste global. KNN no incorporaba ningún

mecanismo para corregirlo pero a partir de CART se aplica SMOTE (*Synthetic Minority Over-sampling Technique*) en todos los modelos.

SMOTE no duplica instancias existentes, sino que genera muestras sintéticas interpolando entre instancias reales de la misma clase. Para cada muestra minoritaria, selecciona aleatoriamente uno de sus K vecinos más cercanos dentro de esa misma clase y crea un nuevo punto en algún lugar del segmento que los une. El resultado es un conjunto de entrenamiento más equilibrado con muestras nuevas que son coherentes con las originales y no simples copias.

La distribución antes de SMOTE en el conjunto de entrenamiento es la siguiente:

Clase	Instancias antes de SMOTE
BENIGN	1.041.725
PortScan	127.043
DDoS	102.420
FTP-Patator	6.348
SSH-Patator	4.717
Bot	1.565
Web Attack – Brute Force	1.205
Web Attack – XSS	522
Infiltration	29
Web Attack – Sql Injection	17

Tabla 8 - Distribución de clases en el conjunto de entrenamiento antes de aplicar SMOTE

No tiene sentido equilibrar todas las clases hasta el nivel de BENIGN pues esto generaría un conjunto de entrenamiento de tamaño inmanejable con un volumen enorme de instancias sintéticas que distorsionarían la esencia del problema real. En su lugar se fija un tope de 10.000 muestras por clase y solo se sobremuestrean las que no llegan a ese valor, por ejemplo: BENIGN, PortScan y DDoS, que ya lo superan, se dejan intactos.

Como vemos en el código, el parámetro *k_neighbors* se fija en 3 en lugar del valor por defecto de 5. La razón es que con clases que tienen solo15 o 25 instancias, buscar 5 vecinos dentro de la misma clase no siempre es posible. Con k_neighbors=3 el proceso es más conservador y los puntos sintéticos quedan más próximos a las muestras reales.

Tras SMOTE, el conjunto de entrenamiento tiene 1.341.188 instancias y todas las clases minoritarias han sido elevadas a 10.000 muestras. El conjunto de test no se modifica en ningún momento manteniendo la distribución real del problema.

5. Reducción de la muestra para búsqueda de hiperparámetros

Con más de un millón de instancias, ejecutar la búsqueda de hiperparámetros con validación cruzada sobre el conjunto completo sería lento. Por ello, se extrae una submuestra estratificada de 10.000 instancias que se usa exclusivamente para esta fase. Esto se ve reflejado en el siguiente fragmento de código:

```
# Muestra estratificada de 10 000 filas — se usa SOLO para buscar el mejor alpha
X_train_s, _, y_train_s, _ = train_test_split(
    X_train_res, y_train_res,
    train_size=10_000,
    stratify=y_train_res,
    random_state=42
)

print(f"Muestra reducida: {X_train_s.shape}")
print(pd.Series(y_train_s).value_counts())
```

La distribución resultante preserva las proporciones del conjunto balanceado: BENIGN aporta 7.767 instancias, PortScan 947, DDoS 764, y las clases minoritarias entre 74 y 75 cada una. El modelo final siempre se entrena sobre el conjunto SMOTE completo.

6. Selección del hiperparámetro de poda: Cost-Complexity Pruning

Se usa el parámetro *ccp_alpha* para controlar cuánto podamos el árbol. Funciona añadiendo una penalización al error de entrenamiento proporcional al número de hojas:

$$\text{Error total} = \text{Error en training} + \alpha \cdot \text{número de hojas}$$

Con α=0 no se aplica ninguna penalización a la complejidad del árbol, permitiendo que este crezca hasta ajustarse prácticamente por completo a los datos de entrenamiento. Aunque ello se traduce en un rendimiento muy elevado sobre el conjunto de training se pierde la capacidad de generalización sobre datos no vistos. El valor de alpha óptimo está en un punto medio: lo suficientemente grande como para eliminar las ramas que solo capturan ruido, pero lo suficientemente pequeño como para conservar las particiones que reflejan patrones reales.

En la práctica, no se sabe de antemano en qué rango de alpha se encuentra ese punto óptimo, porque depende del dataset. Por eso la búsqueda se hace en dos pasos.

Paso 1 : Calcular el rango real de alphas:

```
fulltree_path = DecisionTreeClassifier(random_state=2)
fulltree_path.fit(X_path, y_path)

path = fulltree_path.cost_complexity_pruning_path(X_path, y_path)
raw_alphas = path.ccp_alphas

# Descartar alpha=0 y valores extremadamente pequeños (< 1e-5)
raw_alphas = raw_alphas[raw_alphas > 1e-5]

# 20 alphas en escala logarítmica
alphas_grid = np.geomspace(raw_alphas.min(), raw_alphas.max(), num=20)
```

En este fragmento de código se entrena un árbol sin poda sobre una muestra de 100.000 instancias y se calcula el camino de poda completo con *cost_complexity_pruning_path.* Esto

devuelve todos los valores de alpha en los que el árbol pierde una hoja. Se descartan los valores menores de 10^{-5} y se genera un conjunto de 20 valores de α distribuidos en escala logarítmica entre 1,13e-05 y 7,37e-02. La escala logarítmica es más adecuada que la lineal porque los valores pequeños de α, que son los más relevantes para el proceso de poda, se concentran muy cerca de cero.

Paso 2 : GridSearchCV sobre la submuestra:

Los 20 valores de α se evalúan mediante validación cruzada de 3 folds sobre la submuestra de 10.000 instancias, utilizando el F1-macro como métrica de referencia para seleccionar la configuración más adecuada. Obtenemos el siguiente resultado:

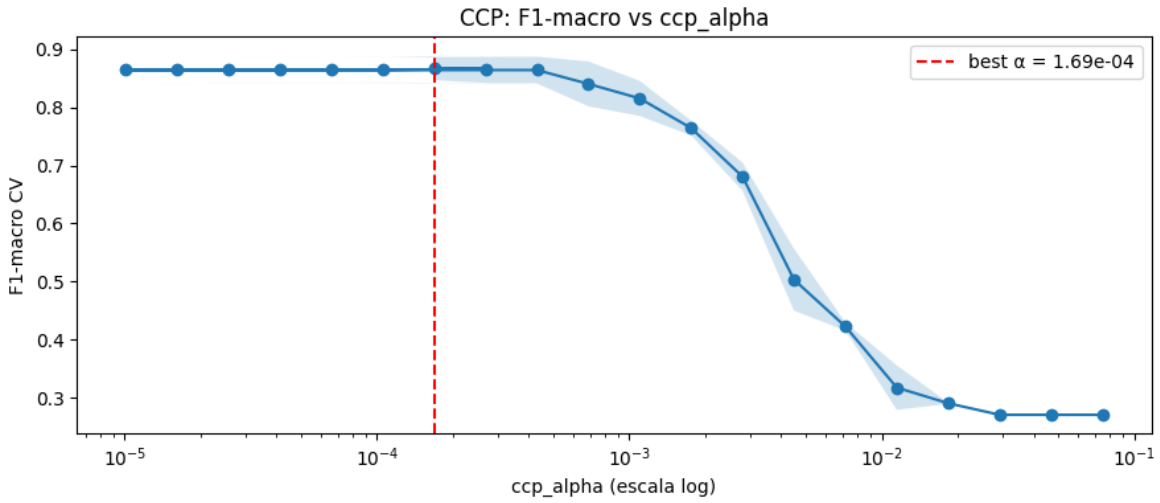

Figura 17- Evolución del F1-macro en validación cruzada para distintos valores de ccp_alpha en el modelo CART

El gráfico muestra una zona constante con alphas pequeños donde el F1 se mantiene en torno a 0,86-0,87, seguida de una caída brusca a partir de 10^{-3}. A partir de ahí el árbol pierde demasiadas ramas y el F1 cae hasta 0,28. El alpha seleccionado: 0,000169, está justo antes de esa caída, que es donde interesa estar.

7. Entrenamiento del modelo final

Con ccp_alpha=0,000169 entrenamos el modelo sobre el conjunto SMOTE completo (1.341.188 instancias) y obtenemos un árbol de 16 niveles y 66 hojas. Para un problema con 10 clases es un tamaño manejable, no es tan pequeño como para perder información relevante ni tan grande como para que no sea interpretable.

La siguiente Figura 18 representa los tres primeros niveles del árbol CART final

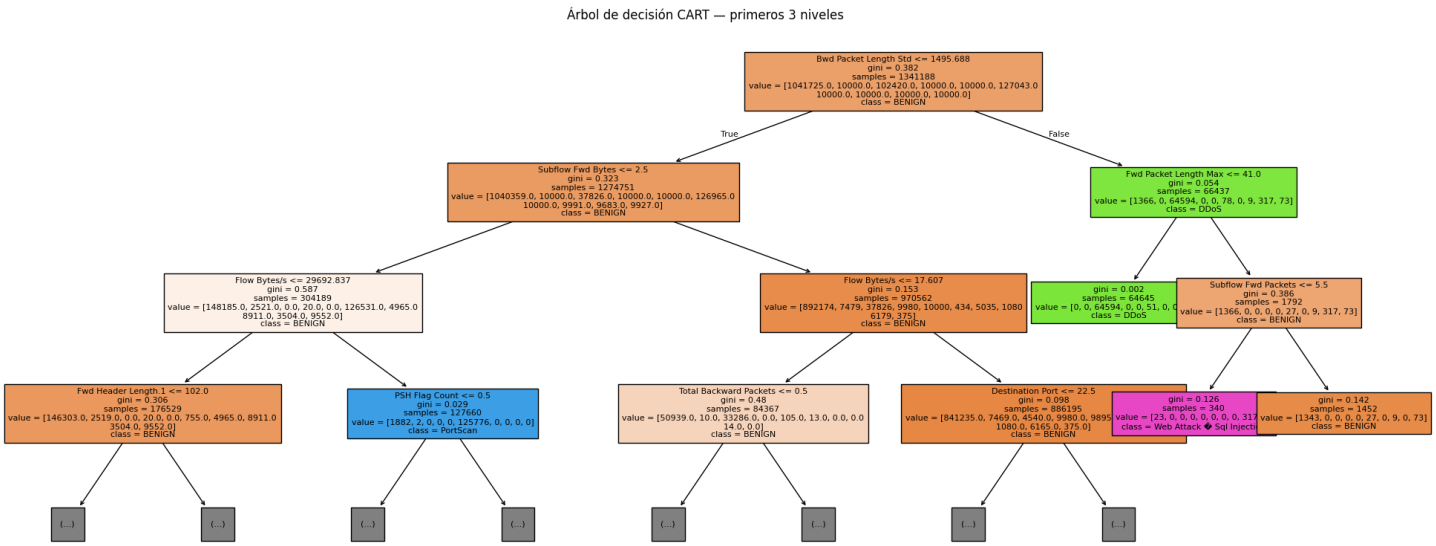

Figura 18 - Primeros tres niveles del árbol de decisión CART entrenado tras la poda

Los primeros niveles del árbol muestran qué variables usa el modelo para las decisiones más generales y por lo tanto cuales son las variables clave.

La primera partición se hace sobre *Bwd Packet Length Std.* La desviación estándar del tamaño de los paquetes de respuesta separa el tráfico en dos grandes grupos:

- los flujos con poca variabilidad van a la rama izquierda, con 1.274.751 instancias mayoritariamente benignas
- los de alta variabilidad van a la rama derecha, donde predomina DDoS.

En el segundo nivel, la rama izquierda se divide por *Subflow Fwd Bytes*, separando los flujos con prácticamente ningún byte enviado en la dirección forward del resto. La rama derecha

usa *Fwd Packet Length Max* que distingue los ataques DDoS caracterizados por paquetes muy pequeños.

En el tercer nivel empieza a aparecer más variedad. *PSH Flag Count* separa el tráfico sin flag PSH, que corresponde principalmente a PortScan, ya que los escaneos no generan intercambio real de datos de aplicación y tienen PSH=0 en todos los paquetes.

En otra rama aparece Destination Port ≤ 22.5, que separa el tráfico dirigido a puertos bajos donde se concentran SSH-Patator y FTP-Patator. También aparece ya en este nivel un nodo que clasifica directamente como Web Attack–Sql Injection, lo que muestra que esta clase tiene una firma lo suficientemente específica como para que se distinga sin necesidad de bajar más en el árbol.

Al final, el árbol resuelve primero las clases fáciles de separar con variables de volumen y flags, y deja las decisiones más complejas para los niveles inferiores. Es exactamente el comportamiento esperado de un árbol bien podado.

8. Evaluación del modelo

8.1. Sobre el conjunto de entrenamiento

Sobre el conjunto de entrenamiento de 1.341.188 instancias, el modelo alcanza accuracy del 99,07 % y macro avg del F1 de 0,91. La poda ha funcionado bien.

Obtenemos la siguiente matriz de confusión:

Figura 19 - Matriz de confusión del modelo CART sobre el conjunto de entrenamiento

Las clases más representadas funcionan muy bien como BENIGN o DDoS que alcanzan F1=1,00. Los resultados más bajos en entrenamiento corresponden a los ataques web:

- Web Attack – Brute Force obtiene F1=0,61 con recall de solo 0,51: el árbol detecta apenas un tercio de los ataques de fuerza. La matriz de confusión muestra que 109 instancias van a BENIGN y 4.746 a Web Attack – XSS.
- Web Attack – XSS tiene recall de 0,78 pero precision de 0,62 y F1=0,69: detecta bien los ataques XSS pero capta parte de los Brute Force.

8.2. Sobre el conjunto de test

La evaluación sobre las 321.398 instancias del test da *accuracy* del 99,70 % y *macro avg del F1* de 0,71. La caída del macro F1 desde 0,91 en entrenamiento hasta 0,71 en test refleja que el modelo tiene más dificultades con la distribución real del problema que con el conjunto balanceado por SMOTE.

Obtenemos la siguiente matriz de confusión:

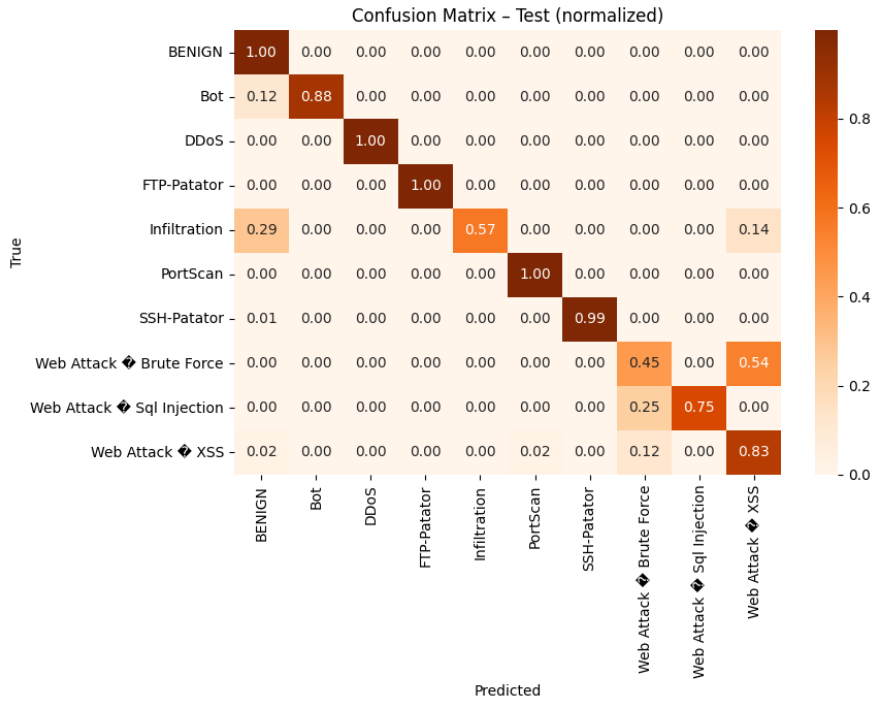

Figura 20 - Matriz de confusión del modelo CART sobre el conjunto de test

En efecto, se observa por ejemplo que:

- Bot baja a F1=0,70 en test, con recall de 0,88 detectando 344 de 391 ataques, con los 47 restantes clasificados como BENIGN. La precisión de 0,59 indica también algunos falsos positivos. El resultado es aceptable teniendo en cuenta que Bot tiene solo 391 instancias reales.

- Infiltration tiene solo 7 instancias en el test. El modelo detecta 4 (recall=0,57) con precision de 0,12 y F1=0,21. La matriz muestra que el 29% de las Infiltration reales se clasifican como BENIGN, el 57 % correctamente. Con un soporte de 7 instancias cualquier conclusión es frágil estadísticamente.

Los ataques web presentan el rendimiento más heterogéneo:

- Web Attack–Brute Force obtiene recall de 0,45 sobre 302 instancias: detecta 136 ataques y deja pasar 166, que van principalmente a XSS (164 instancias) y BENIGN (1). La confusión con XSS es la misma que en entrenamiento. La precision de 0,83 es razonable, pero un recall del 45 % significa que el sistema dejaría sin alerta el 55 % de los ataques de fuerza bruta.

- Web Attack–Sql Injection tiene 4 instancias en el test. Detecta 3 (recall=0,75) pero con precision de 0,08 y F1=014. Con estas cifras el resultado es estadísticamente poco significativo, pero la inyección SQL es uno de los ataques con mayor potencial de daño pues permite el acceso a bases de datos, extracción de datos sensibles etc. Por ello, conviene mencionar esta dificultad de detección, aunque la muestra sea muy pequeña.

En términos operativos, las limitaciones se concentran en los ataques web y en Infiltration. Dejar pasar un Brute Force como XSS no es lo mismo que dejarlo pasar como BENIGN: en el primer caso se genera al menos una alerta de actividad maliciosa, en el segundo no hay ninguna señal

9. Importancia de las características

La siguiente Figura 21 muestra la importancia de las características se calcula mediante permutation importance sobre el conjunto de test completo usando F1-macro como métrica.

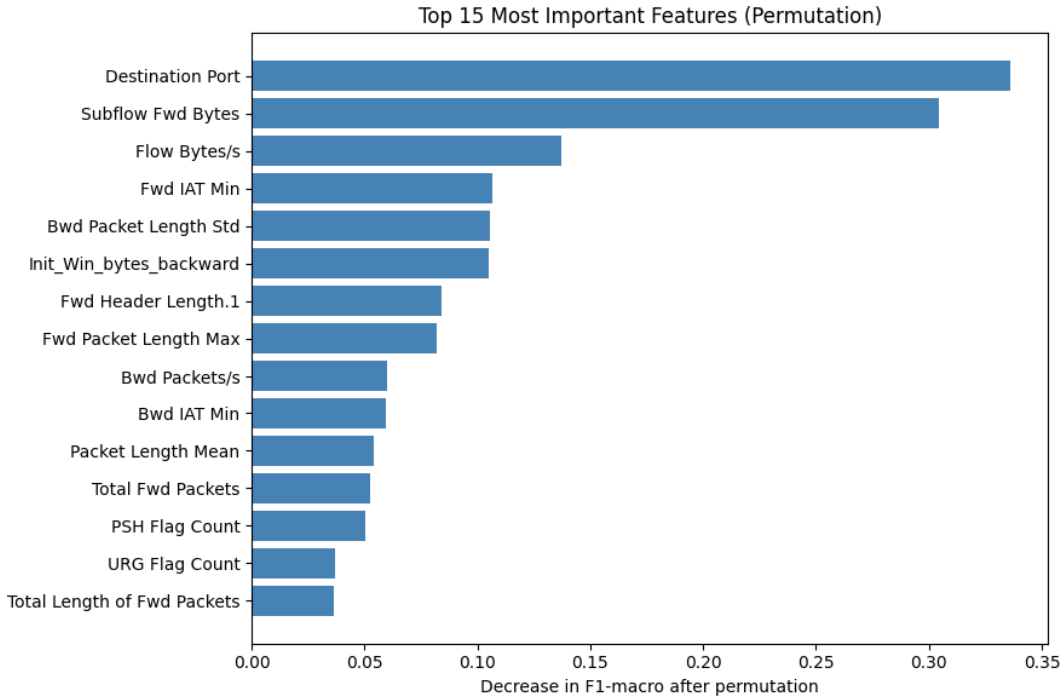

Figura 21 - Importancia de las 15 variables más relevantes para el modelo CART según permutation importance

Las variables más relevantes son:

- Destination Port y Subflow Fwd Bytes están prácticamente empatados en las dos primeras posiciones (0,336 y 0,304). El puerto de destino ya era la variable más importante en KNN, y su presencia aquí confirma su gran influencia. Subflow Fwd Bytes mide el volumen de bytes en la dirección forward del flujo

- Flow Bytes/s (0,137) captura el ritmo de transferencia del flujo.

- Fwd IAT Min (0,106) mide el tiempo mínimo entre paquetes consecutivos en la dirección forward. Los ataques de flooding generan paquetes muy seguidos con tiempos mínimos cercanos a cero, mientras que el tráfico legítimo tiene intervalos más variables.

- Bwd Packet Length Std (0,105) mide la variabilidad en el tamaño de los paquetes de respuesta, que es baja en ataques repetitivos como el escaneo y alta en tráfico legítimo.

- Init_Win_bytes_backward (0,105) es el tamaño de ventana TCP que el receptor anuncia en el SYN-ACK: los servidores legítimos tienden a anunciar valores altos, mientras que las herramientas de ataque usan valores diferentes.

Comparado con KNN, donde PSH Flag Count dominaba con claridad, en CART el peso se distribuye más entre variables de volumen y temporización. Esto refleja diferencias en cómo cada modelo separa las clases.

10. Conclusiones sobre el modelo CART

CART con Cost-Complexity Pruning ofrece resultados sólidos para la mayoría de las clases. El árbol de 16 niveles y 66 hojas generaliza bien en DDoS, PortScan, FTP-Patator y SSH-Patator aunque Bot e Infiltration presentan más dificultades.

Las limitaciones más claras están en los ataques web. La confusión entre Brute Force y XSS es estructural pues aparece tanto en entrenamiento como en test. Esto no se resuelve ajustando el alpha de poda, porque se debe al solapamiento de las firmas de ambas clases.

Un modelo con mayor capacidad, como Random Forest, tiene más posibilidades de capturar esas fronteras combinando múltiples árboles.

La ventaja de CART frente al resto de modelos es la interpretabilidad. Los primeros niveles del árbol pueden leerse directamente, lo que facilita tanto la validación del modelo como la explicación de sus decisiones a un equipo. En un entorno donde las alertas deben poder justificarse esto tiene un gran valor.

4.4 RANDOM FOREST

Random Forest es un algoritmo de clasificación basado en la construcción de múltiples árboles de decisión que se entrenan de forma independiente y combinan sus predicciones mediante votación por mayoría. Cada árbol se entrena sobre una muestra aleatoria con reemplazo del conjunto de entrenamiento (*bootstrap*), y en cada nodo solo evalúa un subconjunto aleatorio de variables.

Frente a CART, la diferencia más práctica es que Random Forest reduce el sobreajuste sin necesidad de poda. Un árbol individual puede memorizar el ruido del entrenamiento, pero cuando se promedian las predicciones de 60 árboles distintos, los errores individuales tienden a cancelarse.

Al igual que CART, Random Forest no requiere normalización, porque trabaja con comparaciones de umbrales y la escala de las variables no afecta a las divisiones.

1. Importación de bibliotecas

Se importa RandomForestClassifier desde sklearn.ensemble. El resto de imports son los mismos que en los modelos anteriores.

2. Carga y limpieza de datos

El proceso es idéntico al descrito en KNN y CART. Tras la limpieza el dataset queda con 1.606.989 instancias y 78 variables.

3. División en entrenamiento y test

Se usa la misma partición que en los modelos anteriores: 80% entrenamiento y 20% test.

4. Balanceo de clases con SMOTE

Se aplica la misma estrategia descrita en CART: tope de 10.000 muestras por clase aplicado únicamente sobre el conjunto de entrenamiento. La distribución antes de SMOTE en el conjunto de entrenamiento es la misma que en el modelo anterior. Tras SMOTE, el conjunto de entrenamiento tiene 1.341.188 instancias, con todas las clases minoritarias elevadas a 10.000 muestras.

5. Reducción de la muestra para búsqueda de hiperparámetros

Al igual que en CART, la búsqueda de hiperparámetros se realiza sobre una submuestra estratificada de 10.000 instancias. La distribución resultante preserva las proporciones del conjunto balanceado: BENIGN aporta 7.767 instancias, PortScan 947, DDoS 764, y las clases minoritarias entre 74 y 75 cada una.

6. Búsqueda de hiperparámetros

Random Forest tiene varios hiperparámetros que afectan tanto al rendimiento como al coste computacional. Los más relevantes son:

- n_estimators : número de árboles
- max_features : número de variables evaluadas en cada nodo
- max_depth : profundidad máxima de cada árbol
- criterion : criterio de impureza para las particiones

La búsqueda se hace en cuatro rondas sucesivas, cada una más fina que la anterior.

6.1. Ronda 1 : Exploración amplia

El objetivo de esta primera ronda es descartar configuraciones claramente subóptimas y orientar las siguientes búsquedas. Se exploran cuatro hiperparámetros simultáneamente

gracias a un GridSearch con validación cruzada como muestra el siguiente fragmento de código:

```
# Round 1: broad search over n_estimators, max_features, max_depth, criterion
param_grid = {
    'n_estimators': [50, 100, 150],
    'max_features': [5, 7, 9],
    'max_depth': [None, 10, 20],
    'criterion': ['gini', 'entropy']
}

cv = StratifiedKFold(n_splits=3, shuffle=True, random_state=123)

grid = GridSearchCV(
    estimator=RandomForestClassifier(random_state=123, n_jobs=1),
    param_grid=param_grid,
    scoring='f1_macro',
    n_jobs=1,
    cv=cv,
    refit=True,
    verbose=1,
    return_train_score=True
)

grid.fit(X_sample, y_sample)

results_cv = pd.DataFrame(grid.cv_results_)
(
    results_cv
    .filter(regex='(param_|mean_test_score|std_test_score|rank_test_score)')
    .sort_values('mean_test_score', ascending=False)
    .head(8)
)
```

Observamos que:

- Para el número de árboles se prueban los siguientes valores : 50, 100 y 150. Valores por debajo de 50 suelen ser insuficientes para que el bosque estabilice sus predicciones, y por encima de 150 el rendimiento tiende a converger con un gran coste computacional.
- Para max_features se prueban 5, 7 y 9: con 78 variables disponibles, evaluar todas en cada nodo eliminaría la aleatoriedad que caracteriza a Random Forest, así que se exploran valores en torno a la raíz cuadrada del número de variables (≈8,8).

- Para max_depth se prueban None (sin límite), 10 y 20: sin límite el árbol puede crecer hasta memorizar el entrenamiento por completo, mientras que valores bajos lo restringen demasiado.
- Finalmente se comparan los dos criterios de impureza estándar, gini y entropy.

El resultado obtenido es el siguiente:

	param_criterion	param_max_depth	param_max_features	param_n_estimators	mean_test_score	std_test_score	rank_test_score
35	entropy	None	9	150	0.891300	0.016372	1
52	entropy	20	9	100	0.890701	0.012220	2
53	entropy	20	9	150	0.889865	0.018397	3
51	entropy	20	9	50	0.886440	0.014850	4
34	entropy	None	9	100	0.885711	0.015913	5
48	entropy	20	7	50	0.883979	0.015120	6
33	entropy	None	9	50	0.883876	0.014938	7
49	entropy	20	7	100	0.883115	0.012014	8

Figura 22 - Resultados de la ronda 1 de la búsqueda de parámetros de RF

Vemos que las primeras posiciones con mejores resultados las ocupan consistentemente configuraciones con criterion=entropy y max_features=9. El mejor resultado se obtiene en n_estimators=150, max_depth=None, con un F1-macro de 0,8913. Por lo tanto, fijamos criterion=entropy para las rondas siguientes

6.2. Ronda 2

Con criterion fijo a entropy, se refina la búsqueda alrededor de los mejores valores de la ronda anterior.

- Para max_features se prueba el rango : 7, 8, 9, 10, centrado en los valores próximos a la raíz cuadrada de 78.
- Para max_depth se añade 15 entre los valores explorados
- Para n_estimators se amplía hasta 200 para ver si más árboles aportan algo.

La mejor combinación es max_depth=None, max_features=9, n_estimators=150, con F1-macro de 0,8912. Cabe destacar que max_depth=20 ofrece un rendimiento muy similar al árbol sin límite pero con menor riesgo de sobreajuste, así que se considera preferible.

6.3. Ronda 3

De nuevo afina la búsqueda en torno a los valores óptimas de la ronda anterior.

- Para max_depth se prueban 18, 20 y 22 para ver si existe un punto óptimo más preciso entre esos valores, manteniendo None como referencia.
- Para max_features se vuelve a explorar el rango 7, 8, 9, pues vemos que da buenos resultados en la métrica evaluada.
- Para n_estimators se añade 120 entre los valores probados.

El mejor resultado vuelve a ser max_depth=None con max_features=9 y n_estimators=150, obteniendo el mismo resultado que la ronda anterior en F1=0,8913. La diferencia respecto a max_depth=20 sigue siendo mínima.

6.4. Ronda 4: Ajuste final

Dado que las rondas anteriores no consiguen mejorar el F1 más allá de 0,8913, se exploran dos parámetros de regularización adicionales que no se habían considerado:

- min_samples_split probando los valores : 2, 5, 10, que controla el número mínimo de instancias necesarias para dividir un nodo
- min_samples_leaf probando los valores : : 1, 2, 4, que fija el mínimo de instancias que debe tener una hoja.

Observamos los siguientes resultados:

	param_criterion	param_max_depth	param_max_features	param_min_samples_leaf	param_min_samples_split	param_n_estimators	mean_test_score	std_test_score	rank_test_score
55	entropy	None	9	1	2	150	0.891300	0.016372	1
0	entropy	20	8	1	2	80	0.891156	0.005965	2
36	entropy	None	8	1	2	80	0.891130	0.005836	3
19	entropy	20	9	1	2	150	0.889865	0.018397	4
22	entropy	20	9	1	10	80	0.887516	0.008662	5
18	entropy	20	9	1	2	80	0.887486	0.012366	6
37	entropy	None	8	1	2	150	0.886759	0.009436	7
56	entropy	None	9	1	5	80	0.886366	0.010879	8

Figura 23 - Resultados ronda final de la búsqueda de parámetros de RF

Valores más altos de ambos producen árboles más conservadores que generalizan mejor en algunos datasets, pero en este caso los mejores resultados corresponden con los valores por defecto (min_samples_split=2, min_samples_leaf=1). Esto indica que añadir estos parámetros no aporta aquí.

Aunque el mejor resultado de la búsqueda corresponde a max_depth=None con n_estimators=150 y F1-macro de 0,8913, se opta por la segunda mejor combinación por dos razones. Por un lado, max_depth=None permite que los árboles crezcan sin límite, lo que aumenta el riesgo de sobreajuste. Por otro, reducir n_estimators de 150 a 80 disminuye el coste computacional del entrenamiento sin pérdida apreciable de rendimiento, ya que ambas configuraciones obtienen un F1-macro prácticamente idéntico, Esta combinación ofrece un equilibrio más adecuado entre rendimiento, generalización y eficiencia.

Por lo tanto se selecciona como configuración final:

- max_depth=20
- max_features = 8
- n_estimators=80
- criterion=entropy

7. Entrenamiento del modelo final

El modelo final se configura con los parámetros ya mencionados y se entrena sobre la totalidad del conjunto SMOTE (1.341.188 instancias).

El OOB score es 0,9961, lo que anticipa una buena capacidad de generalización antes incluso de evaluar sobre el test. El OOB score (*out-of-bag*) es una estimación del error de generalización que Random Forest durante el entrenamiento: cada árbol se evalúa sobre las instancias que no fueron incluidas en su muestra bootstrap.

8. Evaluación del modelo

8.1. Sobre el conjunto de entrenamiento

Sobre el conjunto de entrenamiento de 1.285.591 instancias, el modelo alcanza accuracy del 99,90 % y macro avg del F1 de 0,94. La mayoría de las clases obtienen resultados muy buenos como muestra la matriz de confusión:

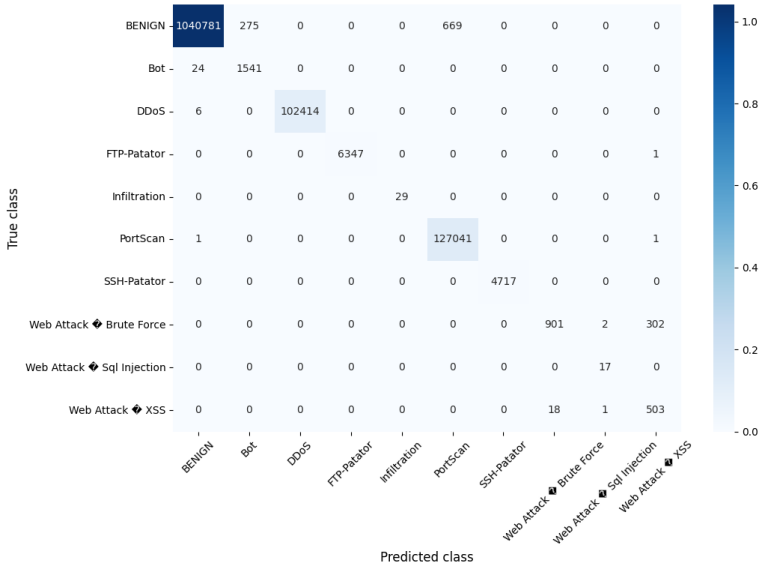

Figura 24 - Matriz de confusión del modelo RF sobre el conjunto de entrenamiento

Los resultados más bajos en entrenamiento corresponden de nuevo a los ataques web. Se observa también que la confusión entre Brute Force y XSS persiste, aunque en menor medida que en CART.

9.2. Sobre el conjunto de test

Sobre las 321.398 instancias del test, el modelo obtiene accuracy del 99,82 % y macro avg del F1 de 0,83. La caída desde 0,94 en entrenamiento hasta 0,83 en test es esperable.

Obtenemos la siguiente matriz de confusión:

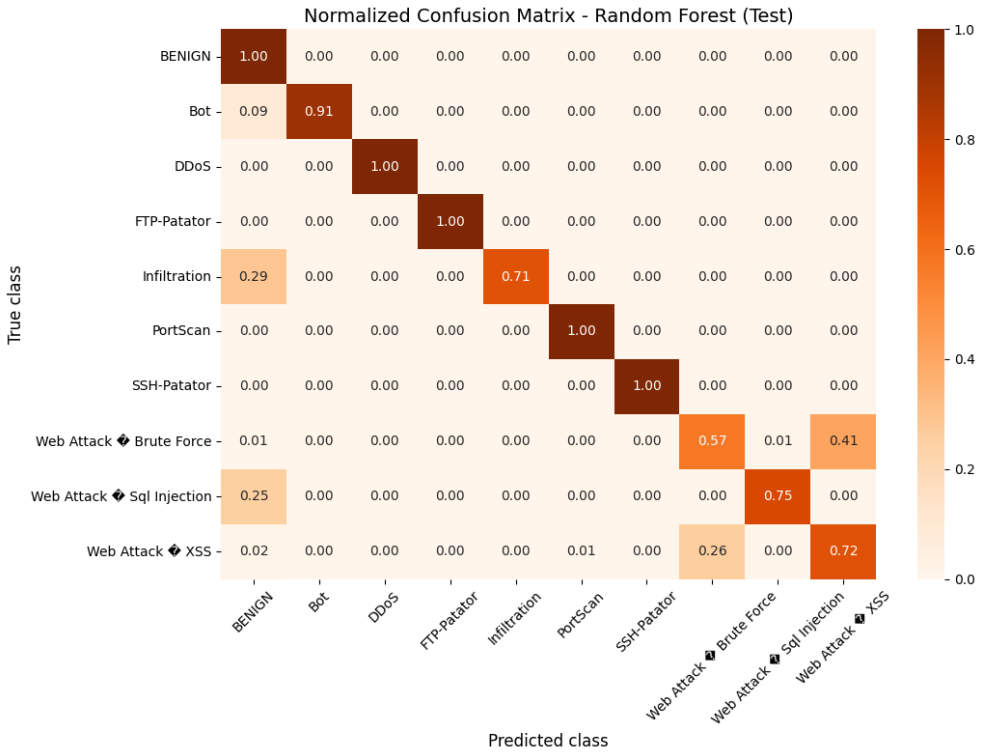

Figura 25 - Matriz de confusión del modelo RF sobre el conjunto de test

Observamos que:

- BENIGN, DDoS y FTP-Patator, PortScan y SSH-Patator llegan a F1=1,00. Estas clases con firmas estadísticas bien definidas se detectan prácticamente sin errores.
- Bot obtiene F1=0,80 en test, con recall de 0,91: detecta 354 de los 391 ataques, con 37 clasificados como BENIGN. La precision de 0,72 indica algunos falsos positivos.
- Infiltration tiene 7 instancias en el test. El modelo detecta 5 (recall=0,71) y los 2 no detectados van a BENIGN, como muestra la matriz normalizada (0,29 hacia BENIGN, 0,71 correcto).

Los ataques web presentan el rendimiento más heterogéneo:

- Web Attack – Brute Force obtiene F1=0,68 detecta 173 de los 302 ataques, con 123 clasificados como XSS y 2 como BENIGN.
- Web Attack – XSS obtiene F1=0,54 con recall de 0,72: detecta 93 de 130 ataques, con 34 clasificados como Brute Force. La confusión bidireccional entre estos dos subtipos es el patrón más persistente en todos los modelos estudiados.

- Web Attack – Sql Injection tiene 4 instancias en el test: detecta 3 (recall=0,75). La matriz normalizada muestra que el 25 % van a BENIGN y el 75 % se clasifican correctamente.

Podemos decir que los resultados son sólidos para la mayoría de las amenazas. Los ataques con muchas instancias se detectan prácticamente sin errores. Las limitaciones se concentran en los ataques web, donde Brute Force deja pasar el 43 % de los casos y la confusión con XSS es frecuente. Como ya se vio en CART, confundir un Brute Force con un XSS al menos genera una alerta, por ello el error más grave sigue siendo la clasificación hacia BENIGN, que en Brute Force afecta a solo 2 de los 302 ataques.

9. Importancia de las características

En Random Forest la importancia de las variables se calcula basándose en la reducción media del índice de Gini que aporta cada variable a lo largo de todos los árboles y todos los nodos donde aparece

Las 15 variables más importantes las muestra la figura siguiente:

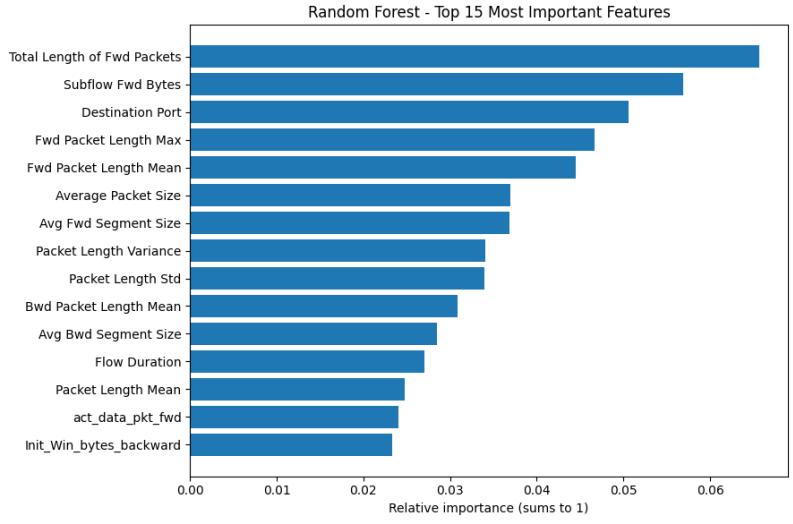

Figura 26 - Importancia de las 15 variables más relevantes para el modelo RF

Las tres variables más importantes son:

- Total Length of Fwd Packets
- Subflow Fwd Bytes
- Destination Port

Las dos primeras miden el volumen total de bytes enviados en la dirección forward del flujo. Esto tiene sentido pues ataques como DDoS generan volúmenes muy distintos al tráfico normal, y el bosque encuentra en estas variables separaciones muy claras desde los primeros niveles de los árboles. El puerto de destino aparece en tercera posición lo que es coherente con lo visto en los modelos anteriores.

A diferencia de KNN, donde PSH Flag Count dominaba con claridad, aquí las importancias están mucho más repartidas. Las posiciones 4 a 15 las ocupan distintas métricas de tamaño de paquetes : longitud máxima, media, varianza, desviación, tanto en la dirección forward como backward. Esto es una consecuencia de cómo funciona Random Forest: al evaluar solo un subconjunto aleatorio de variables en cada nodo, el bosque distribuye el trabajo de clasificación entre muchas variables en lugar de apoyarse siempre en las mismas.

En la posición 14 aparece la variable: act_data_pkt_fwd con una importancia de 0,024. Esta variable cuenta los paquetes que realmente transportan datos de aplicación, excluyendo los de control. Su presencia entre las relevantes refleja que los ataques de sondeo, que solo generan señalización TCP sin intercambio real de datos, se distinguen del tráfico legítimo por este indicador.

10. Conclusiones sobre el modelo Random Forest

Random Forest con SMOTE es el modelo con mejor rendimiento global de los evaluados en este trabajo. El macro F1 de 0,83 en test refleja una capacidad de detección sólida en la mayoría de las clases, con DDoS, PortScan, FTP-Patator y SSH-Patator prácticamente perfectos y Bot con un recall del 91 %.

La combinación de entropy como criterio y max_depth=20 produce un modelo bien calibrado: suficientemente profundo como para capturar patrones complejos, pero con una restricción que evita que los árboles memoricen el ruido del entrenamiento. El OOB score

de 0,9961 confirmaba antes del test que el modelo generalizaba bien, y los resultados lo corroboran.

Las limitaciones se concentran en los ataques web. Brute Force tiene un F1 de 0,68 y XSS de 0,54, y la confusión entre ambos es el error más recurrente en todo el trabajo. No es un problema de hiperparámetros ni de tamaño del modelo: aparece en KNN, CART y aquí, lo que indica que las características estadísticas de flujo disponibles en el dataset no son suficientemente discriminativas para separar estos dos subtipos de forma consistente. Sql Injection mejora respecto a modelos anteriores gracias a SMOTE, pero con solo 4 instancias en el test cualquier conclusión sobre esta clase tiene poca solidez estadística.

Desde el punto de vista operativo, Random Forest es el modelo más equilibrado del trabajo: buen rendimiento, escalable, sin las restricciones computacionales de KNN ni las limitaciones de capacidad de la SVM. Para un sistema de detección de intrusiones real, cubriría bien las amenazas más frecuentes y volumétricas, con margen de mejora en los ataques de aplicación web.

4.5 REDES NEURONALES

El perceptrón multicapa (MLP, *Multilayer Perceptron*) es una arquitectura de red neuronal artificial compuesta por una capa de entrada, una o varias capas ocultas y una capa de salida. Durante el proceso de entrenamiento, la red ajusta los pesos de sus conexiones para aprender las relaciones entre las variables de entrada y la salida deseada. A diferencia de los modelos anteriores, que trabajan directamente con las variables originales, la red transforma progresivamente los datos en representaciones cada vez más precisas hasta llegar a una clasificación final.

La arquitectura básica de un MLP tiene tres tipos de capas:

- La capa de entrada recibe las variables del flujo de red, una por neurona.

- Las capas ocultas aplican transformaciones no lineales. Aquí cada neurona calcula una combinación ponderada de sus entradas. Luego, le aplica una función de activación, en este caso ReLU (*Rectified Linear Unit*), que devuelve el valor si es positivo y cero si es negativo.
- La capa de salida produce una probabilidad para cada una de las 10 clases, y la predicción final es la clase con mayor probabilidad.

El entrenamiento se realiza mediante el algoritmo Adam, una variante del descenso por gradiente que ajusta el ritmo de aprendizaje para cada parámetro.

La tasa de aprendizaje (*learning rate*) controla cuánto se modifican los pesos en cada paso. Su elección es importante, ya que un valor demasiado alto puede hacer que el proceso de aprendizaje sea inestable, mientras que un valor demasiado bajo ralentiza la convergencia y aumenta el tiempo necesario para entrenar el modelo. Por eso la búsqueda de hiperparámetros se centra en encontrar tanto la combinación óptima de todos los parámetros necesarios para el modelo.

1. Importación de bibliotecas

Desde scikit-learn se importa MLPClassifier como clasificador. También se usa RandomizedSearchCV en lugar de GridSearchCV porque uno de los hiperparámetros a buscar, la tasa de aprendizaje, es un valor continuo. Definir una rejilla fija de valores discretos para este parámetro sería demasiado restrictivo, por ello muestrearemos combinaciones aleatorias dentro de un rango definido.

2. Carga y limpieza de datos

El proceso es el mismo descrito en los modelos anteriores. El dataset queda con 1.606.989 instancias y 78 variables tras la limpieza.

3. División en entrenamiento y test

Partición igual que en los modelos anteriores.

4. Balanceo de clases con SMOTE

Se aplica la misma estrategia descrita en CART: tope de 10.000 muestras por clase, k_neighbors=3, aplicado únicamente sobre el conjunto de entrenamiento.

5. Reducción de la muestra para búsqueda de hiperparámetros

La búsqueda de hiperparámetros se realiza sobre una submuestra. Entrenar múltiples configuraciones de MLP sobre todo el conjunto sería computacionalmente inviable en un tiempo razonable.

6. Búsqueda de hiperparámetros

Los dos hiperparámetros principales de la MLP son:

- Arquitectura de las capas ocultas: cuántas capas hay y cuántas neuronas tiene cada una. Este criterio es importante pues capas más grandes o más numerosas pueden capturar patrones más complejos, pero también son más lentas de entrenar y más propensas al sobreajuste
- Tasa de aprendizaje inicial: La tasa de aprendizaje afecta a la velocidad y a la estabilidad de la convergencia.

Se usa RandomizedSearchCV con 8 combinaciones aleatorias por ronda y validación cruzada de 2 folds, buscando optimizar el F1-macro. El proceso se lleva a cabo en cuatro rondas sucesivas, cada una afinando la búsqueda basándose en los resultados de la anterior.

6.1. Ronda 1 : Exploración de arquitecturas y rango de learning rate

El siguiente fragmento de código muestra la primera fase de optimización del modelo MLP, en la que se evalúan distintas arquitecturas y tasas de aprendizaje mediante una búsqueda aleatoria guiada por la métrica F1-macro utilizando RandomizedSearchCV.

```
param_distributions1 = {
    'hidden_layer_sizes': [(50,), (100,), (50, 20), (100, 50)],
    'learning_rate_init': loguniform(1e-4, 1e-1)
}

grid = RandomizedSearchCV(
    estimator=MLPClassifier(
        solver='adam',
        max_iter=200,
        activation='relu',
```

```
        random_state=123
    ),
    param_distributions=param_distributions1,
    n_iter=8,
    scoring='f1_macro',
    cv=2,
    verbose=3,
    random_state=123,
    return_train_score=True,
    n_jobs=1
)

grid.fit(X_train_s, y_train_s)
```

La primera ronda explora:

- arquitecturas de una y dos capas ocultas : (50,), (100,), (50,20), (100,50)
- tasas de aprendizaje muestreadas de forma logarítmica uniforme entre 1e-4 y 1e-1.

Este rango amplio permite detectar en qué orden se mueven los valores óptimos. El siguiente extracto de la tabla de resultado lo muestra más concretamente:

param_hidden_layer_sizes	param_learning_rate_init	mean_test_score	std_test_score	mean_train_score	rank_test_score
(50, 20)	0.0019	0.7657	0.0021	0.7964	1
(50,)	0.0154	0.7497	0.0032	0.7902	2
(50, 20)	0.0017	0.7485	0.0254	0.7847	3
(50, 20)	0.0144	0.7309	0.0011	0.7598	4
(100,)	0.0016	0.7281	0.0327	0.7674	5
(100,)	0.0028	0.7232	0.0253	0.7677	6
(50, 20)	0.0138	0.7200	0.0058	0.7447	7
(50, 20)	0.0005	0.5475	0.0139	0.5721	8

Figura 27 - Mejores configuraciones obtenidas en la primera ronda de búsqueda de hiperparámetros del modelo MLP

Vemos como las arquitecturas de dos capas, especialmente (50,20), dominan las primeras posiciones, lo que indica que añadir una segunda capa más pequeña ayuda al modelo a aprender representaciones más abstractas. La mejor combinación es (50,20) con una tasa de aprendizaje de 0,0019 y un F1-macro en validación cruzada de 0,7657.

6.2. Ronda 2 — Refinamiento alrededor del ganador

Se centra la búsqueda en arquitecturas de dos capas de tamaño similar o ligeramente menor al mejor de la ronda anterior: (100,), (80,), (80,20), (40,20), (50,20). La tasa de aprendizaje se acota al rango [0,0008; 0,0025] siguiendo la misma lógica.

La combinación ganadora es (40,20) con lr=0,0018 y F1=0,7498. Esta arquitectura más pequeña presenta mejores resultados lo que sugiere que el modelo no necesita tanta capacidad para este problema y que reducir el tamaño ayuda a generalizar mejor.

6.3. Ronda 3 : Ajuste fino de arquitectura y learning rate

Se prueban combinaciones próximas a la mejor de la ronda anterior:

- arquitectura : (50,20), (40,20), (60,30)
- tasas de aprendizaje en [0,0015; 0,002]

La ganadora sigue siendo (40,20) con lr=0,0017, F1=0,7364. La consistencia de (40,20) en las primeras posiciones de las tres rondas da confianza en que es una arquitectura robusta para este problema.

6.4. Ronda 4 : Confirmación y exploración de arquitecturas más compactas

La última ronda prueba arquitecturas ligeramente distintas alrededor del ganador: (40,20), (35,20), (45,20), (40,15), afinando también la tasa de aprendizaje con valores alrededor del óptimo hasta ahora: 0,0016; 0,0017; 0,0018. La ganadora es (40,15) con lr=0,0016 y con F1=0,7311. La segunda capa de 15 neuronas en lugar de 20 ofrece el mejor resultado en esta ronda, aunque la diferencia con (40,20) es pequeña.

Por lo tanto, se selecciona: (40,15) con lr=0,0016 como configuración final.

7. Entrenamiento del modelo final

El modelo final se configura con:

- hidden_layer_sizes=(40,15)
- learning_rate_init=0,0016
- activation='relu'

- solver='adam'
- max_iter=300

Se entrena sobre la totalidad del conjunto SMOTE (1.341.188 instancias). La red converge en 149 iteraciones, bastante por debajo del límite de 300 fijado, lo que indica que el modelo encontró un mínimo estable sin necesidad de agotar el todas iteraciones disponibles.

8. Evaluación del modelo

8.1. Sobre el conjunto de entrenamiento

Sobre el conjunto de entrenamiento de 1.341.188 instancias, el modelo alcanza accuracy del 97,48 % y macro avg del F1 de 0,89. La mayoría de las clases presentan resultados buenos: BENIGN, DDoS, FTP-Patator, SSH-Patator, PortScan y Infiltration. Esto lo refleja la matriz de confusión del entrenamiento:

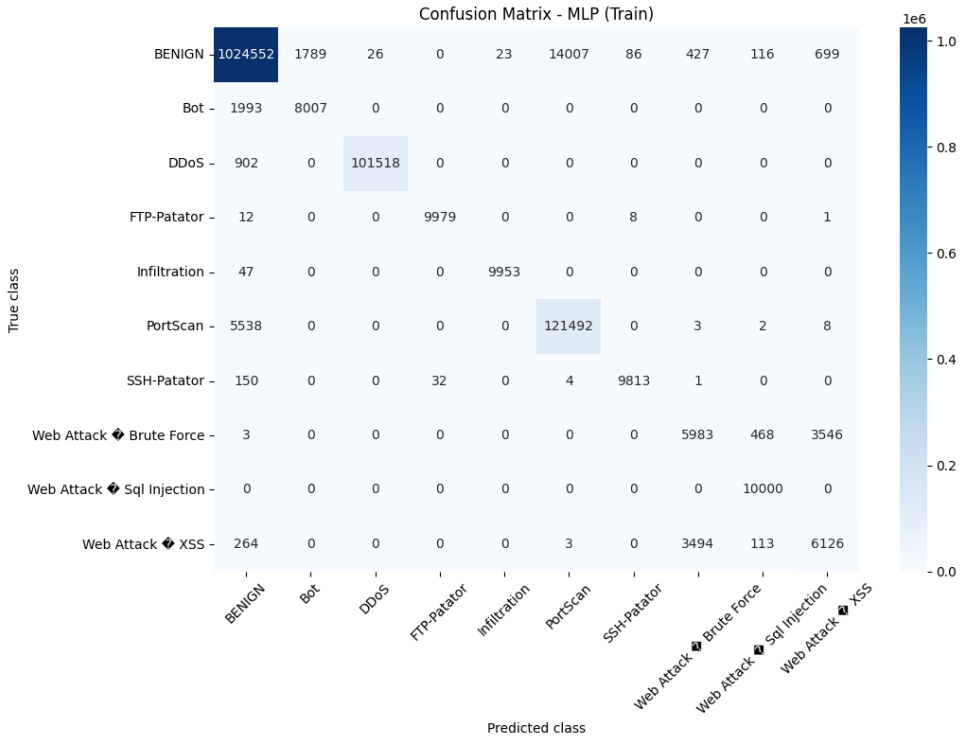

Figura 28 - Figura 25 - Matriz de confusión del modelo RN sobre el conjunto de entrenamiento

Los resultados más bajos en entrenamiento vuelven a ser los ataques web.

8.2. Sobre el conjunto de test

Sobre las 321.398 instancias del test, el modelo obtiene accuracy del 98,08 % y macro avg del F1 de 0,70. La caída desde 0,89 en entrenamiento hasta 0,70 en test es más pronunciada que en Random Forest, lo que indica que la MLP generaliza algo peor en las clases menos representadas.

Obtenemos la siguiente matriz de confusión normalizada:

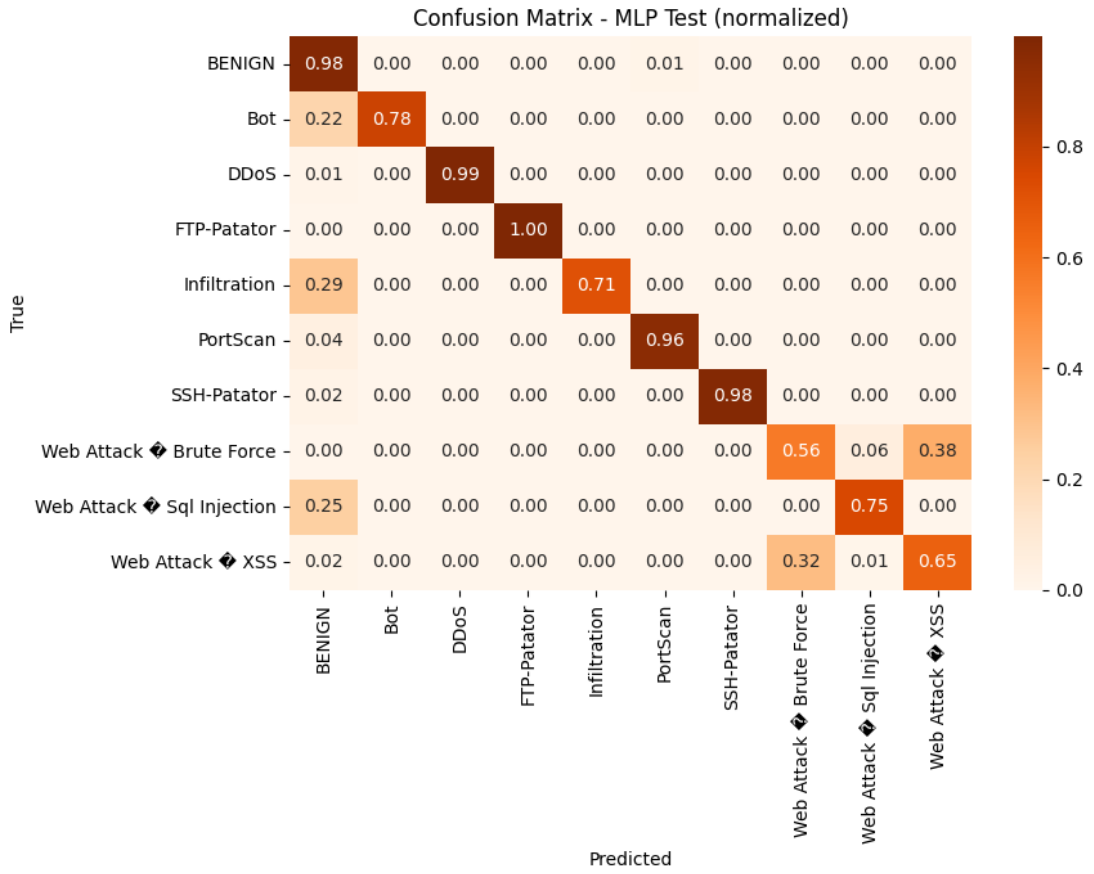

Figura 29 - Matriz de confusión del modelo RN sobre el conjunto de test

BENIGN mantiene F1=0,99 con precision de 0,99 y recall de 0,98. Los 5.298 errores en la matriz absoluta se distribuyen entre BENIGN clasificado como PortScan (3.462), Bot (460) y otras clases menores.

DDoS alcanza F1=1,00 con recall de 0,99. FTP-Patator obtiene F1=1,00. SSH-Patator llega a F1=0,98.

PortScan obtiene F1=0,93 con recall de 0,96: de los 31.761 escaneos del test, 1.371 se clasifican como BENIGN. La precision de 0,90 indica también algunos falsos positivos. Bot cae a F1=0,53 en test con recall de 0,78: detecta 305 de los 391 ataques, con 86 clasificados como BENIGN. La precision de 0,40 es bastante baja, generando un número considerable de falsas alarmas de Bot.

Infiltration tiene 7 instancias en el test. El modelo detecta 5 (recall=0,71) con precision de 0,50 y F1=0,59. La matriz normalizada muestra que el 29 % van a BENIGN y el 71 % se clasifican correctamente.

Los ataques web presentan resultados heterogéneos. Web Attack – Brute Force obtiene F1=0,55 con precision de 0,54 y recall de 0,56: detecta 170 de los 302 ataques, con 115 clasificados como XSS y ninguno como BENIGN. Web Attack – XSS obtiene F1=0,33 con precision de 0,22 y recall de 0,65: detecta 85 de 130 ataques pero con muchos falsos positivos de otras clases clasificadas como XSS. La matriz normalizada muestra que el 32 % de los XSS reales van a Brute Force y el 65 % se clasifican correctamente.

Web Attack – Sql Injection es el caso más llamativo. Con 4 instancias en el test, el modelo detecta 3 (recall=0,75) pero con una precision de apenas 0,05 y F1=0,10. Un precision tan baja significa que por cada Sql Injection correctamente detectada, el modelo clasifica erróneamente como Sql Injection muchas otras instancias. En la matriz absoluta se ven 3 predicciones correctas pero con un elevado número de falsos positivos procedentes de otras clases. Con solo 4 instancias reales, este resultado tiene poca robustez estadística.

Comparando con Random Forest, la MLP obtiene un macro F1 de 0,70 frente al 0,83 de RF. Las diferencias más notables están en Bot (0,53 vs 0,80), XSS (0,33 vs 0,54) y Sql Injection (0,10 vs 0,43). En las clases más fáciles : DDoS, FTP-Patator, SSH-Patator, ambos modelos son comparables. La MLP tiene más dificultades con las clases minoritarias, probablemente

porque su arquitectura compacta limita su capacidad para aprender las fronteras de decisión más complejas.

En términos operativos, la MLP ofrece buena detección para DDoS, FTP-Patator y SSH-Patator, rendimiento aceptable en PortScan y Bot, pero resultados claramente inferiores a Random Forest en los ataques web. El alto número de falsos positivos en Bot y XSS generaría un volumen de alertas innecesarias que dificultaría la operación del sistema.

9. Importancia de las características

La importancia de las características se calcula mediante permutation importance sobre el conjunto de test, con F1-macro como métrica.

Las 15 variables más relevantes se muestran en la siguiente figura:

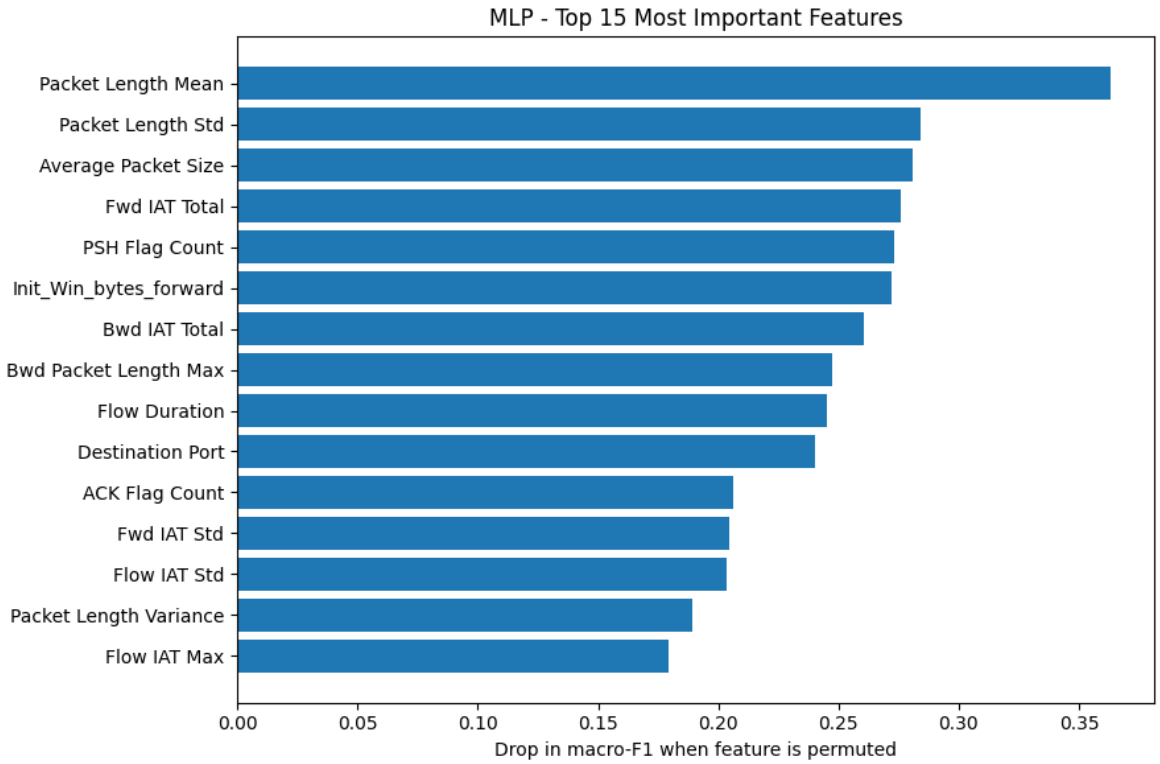

Figura 30 - Importancia de las 15 variables más relevantes para el modelo RN

El ranking de la MLP es bastante diferente al de los modelos anteriores. Packet Length Mean, seguido de Packet Length Std y Average Packet Size (0,281).

Las tres variables miden el tamaño de los paquetes del flujo : media, desviación típica y tamaño medio, y están muy relacionadas entre sí. Su importancia refleja que la MLP ha aprendido a basar sus decisiones principalmente en el volumen y la variabilidad del tráfico.

Lo más llamativo es la posición del puerto de destino: aparece en décima posición con una importancia de 0,240. Está muy por detrás de donde aparecía en KNN, CART y SVM, donde era la variable más o segunda más importante. En su lugar, variables de temporización como Fwd IAT Total (0,276), Bwd IAT Total (0,261) y Flow Duration (0,245) tienen mucho más peso. Los tiempos entre llegadas de paquetes reflejan el ritmo del tráfico esto puede ser útil para detectar ataques de flooding que generan paquetes muy seguidos, mientras que el tráfico web legítimo tiene patrones temporales más irregulares.

Las importancias en la MLP están mucho más distribuidas que en los modelos anteriores: todas las variables del top 15 tienen caídas entre 0,18 y 0,36, sin que ninguna domine de forma tan clara como el puerto en CART o el PSH Flag Count en KNN. Esto es coherente con cómo funciona una red neuronal: al combinar las variables en múltiples capas y transformaciones no lineales, el modelo aprende a usar muchas variables simultáneamente en lugar de apoyarse en unas pocas.

10. Conclusiones sobre el modelo MLP

La red neuronal MLP con arquitectura (40,15) y tasa de aprendizaje 0,0016 ofrece un rendimiento sólido en las clases con mayor soporte, pero queda por detrás de Random Forest en el conjunto global. El macro F1 de 0,70 en test frente al 0,83 de RF refleja principalmente las dificultades en Bot, XSS y Sql Injection.

La arquitectura final es bastante compacta. Las cuatro rondas de búsqueda mostraron de forma consistente que arquitecturas más pequeñas superaban a las más grandes. Esto indica que para este problema la red no necesita mucha profundidad para capturar los patrones relevantes y que añadirla llevaría más a un sobreajuste que a una mejor generalización.

El punto más interesante de la MLP respecto a los modelos anteriores es el ranking de importancia de variables. Destacan las variables de tamaño de paquetes y los tiempos entre llegadas, mientras que el puerto de destino queda en décima posición. Esto indica que la red neuronal está aprendiendo a caracterizar el comportamiento del tráfico de forma distinta a como lo hacen los árboles.

Desde el punto de vista operativo, la MLP sería una buena opción para detectar DDoS, FTP-Patator y SSH-Patator, pero los falsos positivos en Bot y XSS harían necesario ajustar el umbral de decisión antes de aplicarlo a un sistema real. Random Forest sigue siendo la opción más equilibrada entre rendimiento y fiabilidad para este dataset.

4.6 SVM

Las Máquinas de Vectores de Soporte (SVM) buscan el hiperplano que separa las clases con el mayor margen posible. Se basa únicamente en las observaciones más cercanas a esa frontera, llamadas vectores de soporte. Esto hace que el modelo sea menos propenso al sobreajuste en espacios de muchas dimensiones, como el que tenemos aquí con 78 variables.

El comportamiento de la SVM depende del kernel elegido pues determina cómo se mide la similitud entre puntos en el espacio:

- El kernel lineal busca una frontera plana
- El kernel RBF (*Radial Basis Function*) permite fronteras curvas adaptadas a la forma local de los datos
- El kernel polinomial genera fronteras de mayor complejidad matemática.

La elección del kernel es el primer hiperparámetro a determinar, y condiciona el resto de la búsqueda.

El problema principal de SVM es el coste computacional. El entrenamiento escala de forma cuadrática con el número de muestras, lo que hace inviable entrenar directamente sobre el millón de datos del conjunto SMOTE.

SVM requiere normalización ya que trabaja calculando distancias y productos escalares.

1. Importación de bibliotecas

Desde scikit-learn se importan SVC como clasificador, GridSearchCV y StratifiedKFold para la búsqueda de hiperparámetros.

2. Carga y limpieza de datos

El proceso es idéntico al descrito en los modelos anteriores.

3. División en entrenamiento y test

Partición 80/20 con stratify, igual que en los modelos anteriores.

4. Visualización del espacio de características

Antes de entrenar el modelo se visualiza la distribución del conjunto de entrenamiento en dos dimensiones eligiendo 2 características como muestran la Figura 31:

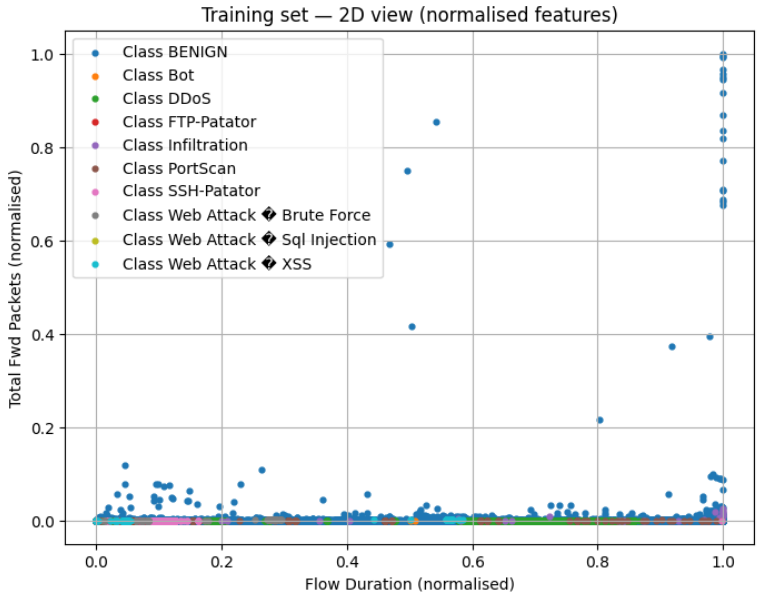

Figura 31 - Proyección bidimensional del conjunto de entrenamiento utilizando características normalizadas

En ambas proyecciones se aprecia el fuerte desbalance entre clases. Claramente el tráfico benigno domina por completo mientras que los ataques apenas son visibles. Además, hay bastante superposición y esto no es solo una limitación de la proyección bidimensional, sino

una señal de que el problema parece no ser linealmente separable. Por ello podemos ir anticipando que el kernel RBF funcionará mejor que el kernel lineal.

5. Balanceo de clases con SMOTE

Se aplica la misma estrategia : tope de 10.000 muestras por clase aplicado únicamente sobre el conjunto de entrenamiento.

6. Reducción de la muestra para búsqueda e inferencia

Dado el coste computacional de SVM, tanto la búsqueda de hiperparámetros como el entrenamiento del modelo final se realizan sobre una submuestra de 15.000 instancias. En efecto, con más de un millón de instancias, entrenar múltiples configuraciones de SVM con kernel no lineal no sería viable en un tiempo razonable. Esta es la limitación muy importante a tener en cuenta, a diferencia de CART, Random Forest o la red neuronal, la SVM no tiene acceso a todos los datos del conjunto de entrenamiento.

7. Búsqueda de hiperparámetros

7.1. Ronda — Kernel RBF

El siguiente fragmento de código configura la primera ronda de optimización del modelo SVM con kernel RBF mediante GridSearchCV. Para ello se evalúan distintas combinaciones de los hiperparámetros C y gamma utilizando validación cruzada de tres particiones, seleccionando la configuración que maximiza la métrica F1-macro.

```
param_grid = [
    {
        'kernel': ['rbf'],
        'C': [0.1, 1, 5, 7, 10],
        'gamma': [0.01, 0.1, 0.5, 1]
    }
]

cv = StratifiedKFold(n_splits=3, shuffle=True, random_state=123)

grid = GridSearchCV(
    estimator=SVC(class_weight='balanced', random_state=123),
    param_grid=param_grid,
```

```
    scoring='f1_macro',
    n_jobs=-1,
    cv=cv,
    refit=True,
    verbose=1,
    return_train_score=True
)
```

Se exploran los siguientes valores de C : 0,1; 1; 5; 7; 10 y gamma : 0,01; 0,1; 0,5. Se prueban estos rangos amplios. Por un lado, C controla el equilibrio entre margen amplio y errores de clasificación:

- valores bajos aceptan más errores, pero generalizan mejor
- valores altos intentan clasificar todo correctamente, pero con el riesgo de sobreajustar

Por otro lado, gamma determina el grado de influencia de cada vector de soporte:

- valores pequeños producen fronteras suaves y globales
- valores grandes hacen que cada punto solo influya en un entorno mucho más restringido

Los mejores resultados obtenidos son los siguientes:

param_C	param_gamma	param_kernel	mean_test_score	std_test_score	rank_test_score
10.0	1.0	rbf	0.622884	0.025698	1
7.0	1.0	rbf	0.577520	0.008565	2
5.0	1.0	rbf	0.562431	0.009033	3
10.0	0.5	rbf	0.560272	0.011010	4
7.0	0.5	rbf	0.550543	0.008460	5
5.0	0.5	rbf	0.541389	0.007613	6
1.0	1.0	rbf	0.517487	0.005157	7
10.0	0.1	rbf	0.499637	0.005554	8

Figura 32 - Mejores combinaciones de hiperparámetros obtenidas para el SVM con kernel RBF

La combinación ganadora es C=10, gamma=1 con F1-macro de 0,6229. Vemos un patrón claro en la tabla es claro: gamma=1 domina las primeras posiciones independientemente del valor de C. Esto indica que el modelo necesita fronteras estrictas y mucha adaptación local.

7.2. Ronda — Kernel lineal

Se prueba el kernel lineal con los siguientes valores de C : 0,1; 1; 3; 5; 7; 10. Estos son habituales para explorar este parámetro en un rango que va desde una regularización fuerte (C pequeño) hasta una casi nula (C grande).

El mejor resultado es C=10 con F1-macro de 0,549, Observamos que la separación lineal no es suficiente para este problema.

7.3. Ronda— Kernel polinomial

El kernel polinomial se evalúa con C : 0,1; 1, gamma : 0,01; 0,1 y grado :2; 3. El rango de C es más pequeño porque el kernel polinomial suele ser más inestable con valores altos.

El mejor resultado es F1-macro de 0,3878 con C=1, grado=2, gamma=0,1. Vemos que el rendimiento es muy limitado por lo que el kernel polinomial queda descartado.

7.4. Ronda: Refinamiento del kernel RBF

Dado que el kernel RBF es el que mejor resultados presenta, se realiza una ronda adicional buscando mejorar los parámetros del modelo. Se prueban valores más altos - alrededor del ganador anterior, buscando fronteras más locales y un margen más estricto.

La ganadora es C=20, gamma=5 con F1-macro de 0,7066, mejorando respecto a la configuración de la primera ronda. El patrón es claro ya que gamma=5 domina las primeras posiciones confirmando que funcionan mejor fronteras muy locales.

8. Entrenamiento del modelo final

El modelo final se configura con:

- C = 20
- gamma = 5
- kernel RBF
- weight='balanced'

Se entrena sobre la submuestra de 15.000 instancias. El parámetro class_weight='balanced' hace que el modelo penalice más los errores en las clases minoritarias.

9. Evaluación del modelo

9.1. Sobre el conjunto de entrenamiento

Sobre la submuestra de entrenamiento de 15.000 instancias, el modelo alcanza accuracy del 90,73 % y macro avg del F1 de 0,73.

Obtenemos la siguiente matriz de confusión:

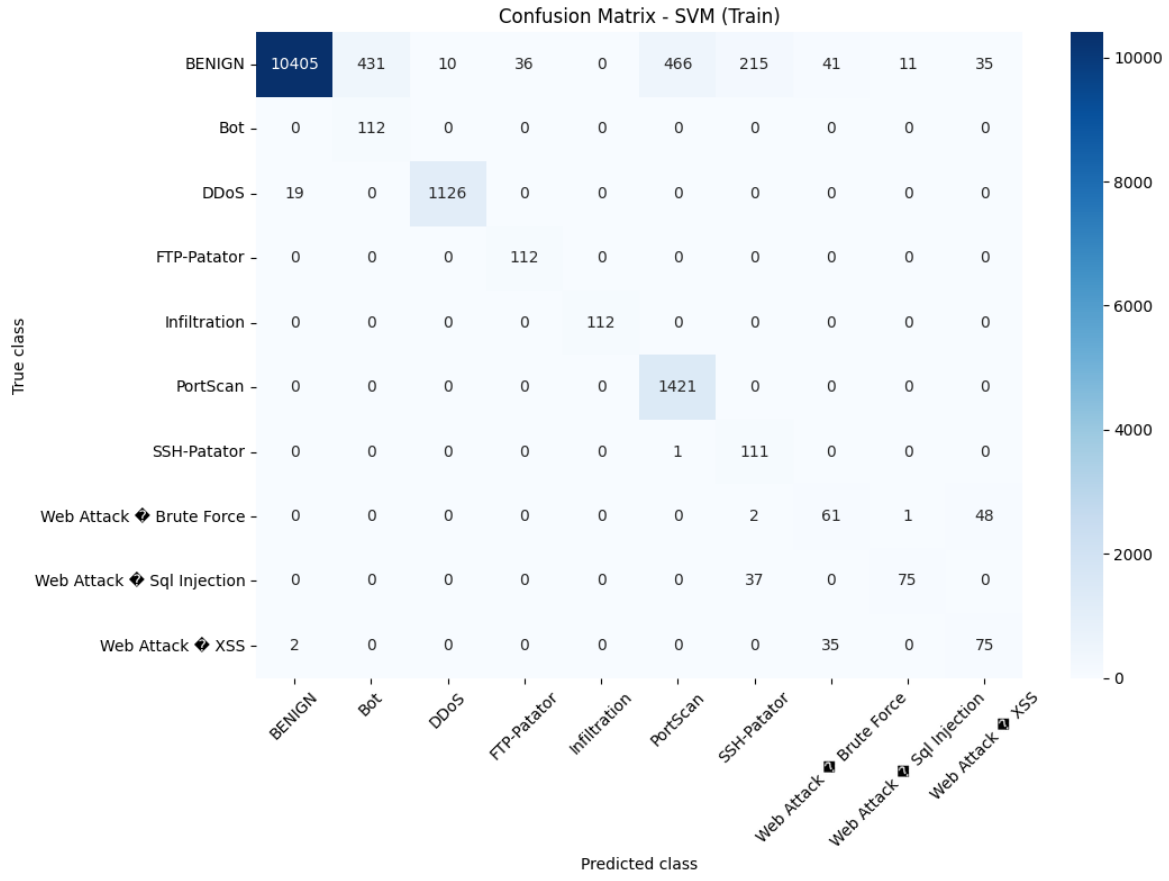

Figura 33 - Matriz de confusión del modelo SVM sobre el conjunto de entrenamiento

El patrón general es el mismo que en otros modelos pues las clases con patrones de tráfico más diferenciados funcionan bien : DDoS, FTP-Patator e Infiltration alcanzan F1 cercano o

igual a 1,00, y PortScan llega a 0,86. BENIGN obtiene un recall de 0,89 que implica que el modelo etiqueta como ataque un 11 % del tráfico legítimo. Esto es consecuencia de usar class_weight='balanced' ya que compensa el desequilibrio entre clases haciendo que los errores en las clases minoritarias pesen igual que los de BENIGN, lo que lleva al modelo a clasificar más falsos positivos.

El problema más visible en entrenamiento es la baja precisión en clases como Bot y SSH-Patator que detectan casi todos los ataques, pero con muchos falsos positivos, lo que produce F1 de 0,34 y 0,47. El modelo aprende a ser muy sensible pero no suficientemente selectivo.

Los ataques web obtienen resultados más equilibrados pero la confusión entre Brute Force y XSS persiste, aunque en menor medida que en modelos anteriores.

9.2. Sobre el conjunto de test

Sobre las 321.398 instancias del test, el modelo obtiene accuracy del 91,26 % y macro avg del F1 de 0,45, bastante inferior al 0,73 del entrenamiento. Esta caída refleja las dificultades para generalizar.

Obtenemos la siguiente matriz de confusión:

Figura 34 - Matriz de confusión del modelo SVM sobre el conjunto de test

BENIGN obtiene F1=0,94 pero con unos 27.000 ejemplos clasificados erróneamente como ataque, principalmente hacia PortScan, Bot y SSH-Patator. En un sistema real este volumen de falsas alarmas sería difícil de gestionar.

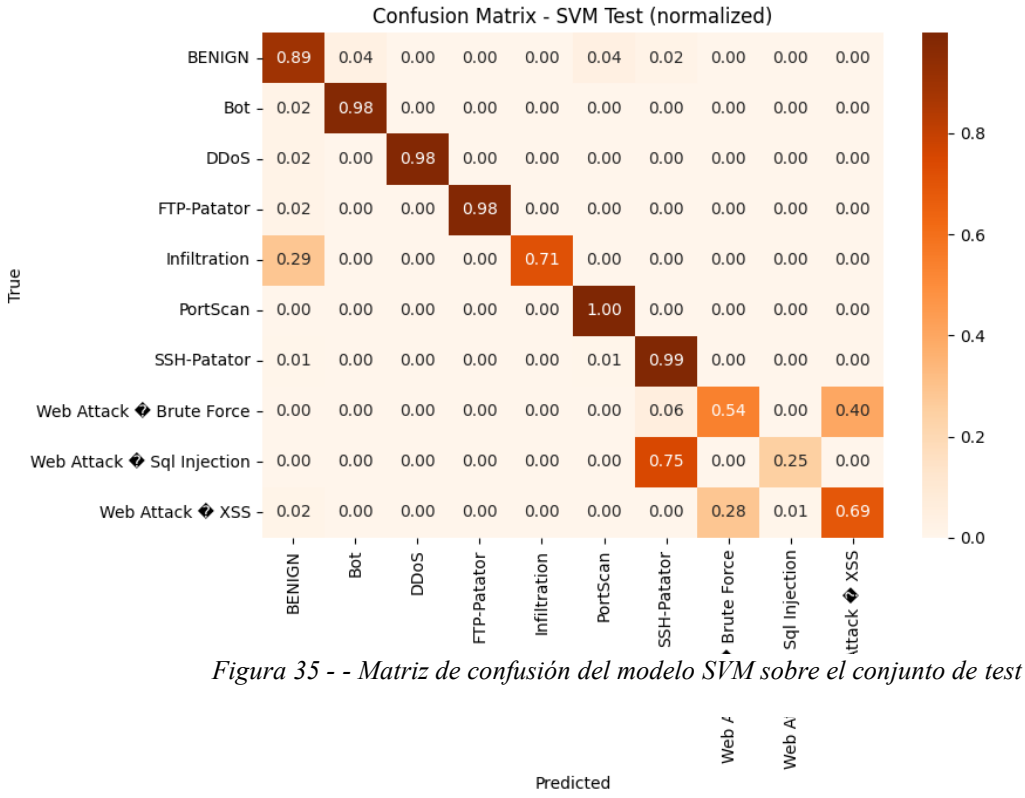

Figura 35 - - Matriz de confusión del modelo SVM sobre el conjunto de test

El problema más grave está en Bot y SSH-Patator. Ambos con una precisión muy baja lo que genera un gran número de falsas alarmas. En Bot, por cada ataque correctamente detectado el modelo genera unas 24 alertas incorrectas.

Los ataques web obtienen F1 bajos en general y sigue persistiendo la confusión entre Brute Force y XSS.

11. Importancia de las características

La importancia de las características se calcula mediante permutation importance sobre una submuestra del test con F1-macro como métrica.

La siguiente figura muestra las 15 variables más relevantes:

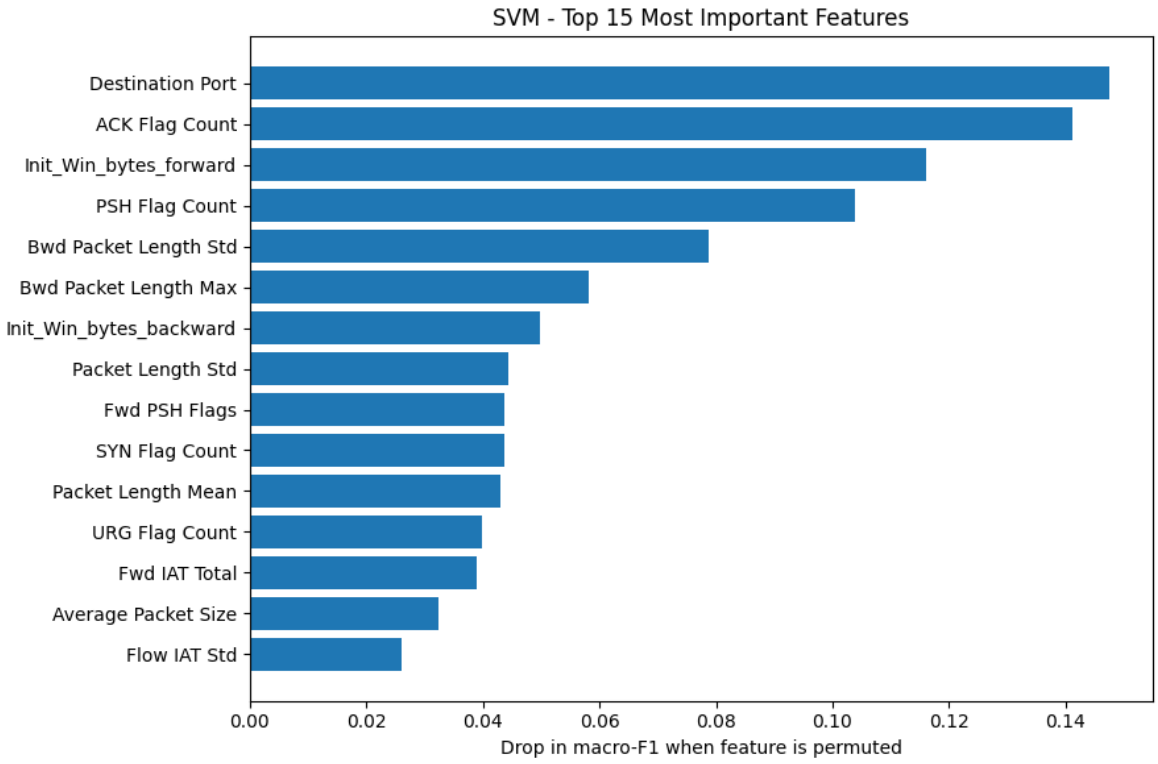

Figura 36 - Importancia de las 15 variables más relevantes para el modelo SVM

Destination Port y ACK Flag Count lideran el ranking:

- El puerto de destino ya aparecía como la variable más importante en CART y KNN confirmando de nuevo que es una de las más importantes del dataset independientemente del algoritmo.

- ACK Flag Count en segunda posición refleja que los ataques de flooding y los escaneos generan patrones de flags ACK muy distintos al tráfico normal.

Init_Win_bytes_forward ocupa la tercera posición. Esta variable recoge el tamaño de ventana TCP que el emisor anuncia al inicio de la conexión. Las herramientas de ataque suelen usar valores bajos o fijos, mientras que los clientes legítimos utilizan valores más altos, lo que hace que esta diferencia sea útil para el modelo.

El ranking de la SVM se parece más al de KNN que al de Random Forest o la MLP. En efecto, las variables de flags TCP dominan las primeras posiciones, mientras que las métricas de volumen de bytes están en posiciones más bajas. Esto muestra como SVM se apoya más en variables de señalización TCP que en variables de volumen, probablemente porque en el espacio normalizado son más fáciles de separar.

12. Conclusiones sobre el modelo SVM

La SVM detecta bien los ataques con mayor volumen de tráfico y con patrones más definidos: DDoS y FTP-Patator funcionan bien, y PortScan se detecta casi sin errores. Sin embargo, el modelo tiende a generar demasiadas falsas alarmas en clases como Bot y SSH-Patator, lo que en la práctica limitaría mucho su eficacia.

Cabe destacar que la gran limitación de este modelo es de escala. Entrenar sobre 15.000 instancias cuando el resto de modelos usan el millón completo es una gran desventaja que el ajuste de C y gamma no puede compensar. Random Forest y la red neuronal, entrenados sobre el conjunto completo, obtienen macro F1 de 0,83 y 0,70 respectivamente frente al 0,45 de la SVM.

En los ataques web el rendimiento es bajo en todos los casos, aunque la confusión entre Brute Force y XSS al menos produce una alerta. El error más grave, clasificar un ataque como BENIGN, ocurre con menos frecuencia que en otros modelos. Esto es una ventaja clara del uso de class_weight='balanced'.

4.7 MODELO DE LENGUAJE GRANDE (LLM): LLAMA 3.2-1B

Un modelo de lenguaje grande (LLM) es un sistema de inteligencia artificial entrenado sobre volúmenes enormes de texto con el objetivo de aprender a predecir qué texto viene a continuación dado un contexto. Para ello va ajustando millones de parámetros internos hasta que es capaz de predecir bien el texto en contextos muy distintos. Como efecto secundario de ese proceso, acaba desarrollando cierta comprensión de los conceptos que el lenguaje describe, aunque nunca fue entrenado explícitamente para ello. El resultado es un sistema capaz de responder preguntas, redactar textos, escribir código, razonar sobre problemas y realizar tareas de clasificación. Todo ello a partir de instrucciones escritas en lenguaje natural sin necesidad de entrenamiento adicional para cada tarea concreta.

Esta última capacidad es la que hace interesante analizar su uso en este trabajo. En lugar de entrenar un modelo específicamente para clasificar flujos de red, la idea es describir el problema en un prompt indicando al LLM qué características tiene el flujo, cuáles son las clases posibles y pedirle que elija una. Este enfoque se llama *zero-shot classification*: clasificación sin ningún ejemplo de entrenamiento específico del problema. Los resultados dependen mucho del tamaño y las capacidades del modelo elegido. Esto lo analizaremos al comparar el experimento realizado aquí con lo que otros modelos mucho más avanzados como el Mythos de Claude.

1. Motivación: ¿por qué probar un LLM?

Los modelos anteriores de KNN, CART, Random Forest, redes neuronales y SVM siguen todos el mismo esquema: se entrena un clasificador sobre ejemplos etiquetados, aprende a asociar patrones, y luego predice sobre nuevos flujos. Este enfoque es el estándar de machine learning con detección de intrusiones.

Pero en los últimos años ha surgido una pregunta interesante: ¿podría un modelo de lenguaje grande razonar sobre tráfico de red sin haber sido entrenado específicamente para ello? Los LLMs han demostrado capacidad para resolver problemas de clasificación en dominios muy

distintos simplemente describiendo la tarea en lenguaje natural. La idea es interesante porque no requiere datos etiquetados ni hay que entrenar nada.

La pregunta que se plantea en este apartado es concreta: ¿puede un LLM clasificar flujos de red del CICIDS2017 a partir de sus características estadísticas, sin haberlo entrenado ni ajustado específicamente para esta tarea?

2. Elección del modelo: Llama 3.2-1B

La elección de Llama 3.2-1B de Meta se debe a que puede ejecutarse localmente sin necesidad de API de pago ni conexión a servicios externos. Modelos como GPT-4 o Claude requieren llamadas a una API con coste por token, lo que hace inviable evaluar 200 registros sin un presupuesto significativo. Llama 3.2-1B, en cambio, es un modelo de código abierto con licencia permisiva que se descarga desde HuggingFace y se ejecuta directamente en CPU.

Sin embargo, la desventaja de esta accesibilidad es el tamaño: con 1.000 millones de parámetros, Llama 3.2-1B es un modelo relativamente pequeño. Esto condiciona sus capacidades de razonamiento, como se verá en los resultados.

3. Código y pipeline de inferencia

El código tiene los siguientes pasos:

3.1 Carga y preprocesamiento del dataset**:** Se cargan los mismos seis CSV del CICIDS2017 y se aplica la misma limpieza que en los modelos anteriores: eliminación de espacios en nombres de columna, sustitución de infinitos por NaN y eliminación de filas afectadas.

3.2. Muestra estratificada: Se toman 20 instancias de cada clase, resultando en 200 registros en total. Este número es pequeño pero justificado por el coste computacional. En CPU cada predicción tarda aproximadamente 19 segundos, lo que hace inviable extender la evaluación a miles de registros sin disponer de GPU.

3.3 Carga del modelo y el tokenizador**:** El modelo se descarga desde HuggingFace usando las clases *AutoModelForCausalLM* y *AutoTokenizer* de la librería Transformers. Una vez cargado, se configura en modo, lo que desactiva ciertos mecanismos que solo tienen sentido durante el entrenamiento y que podrían añadir aleatoriedad innecesaria a las predicciones.

3.4**.** Construcción del prompt**:** Para cada flujo de red se construye un prompt en inglés con la siguiente estructura:

```
def construir_prompt(row):
    lineas = "\n".join(
        f"  - {f}: {row[f]:.4f}" if isinstance(row[f], float) else f"  - {f}:
{row[f]}"
        for f in FEATURES_PROMPT
    )
    return (
        "You are a network security expert. "
        "Analyze the following network flow and classify it.\n\n"
        f"Features:\n{lineas}\n\n"
        f"Possible categories: {', '.join(clases)}\n\n"
        "Reply with ONLY the category name, nothing else.\n\n"
        "Category:"
)
```

- Rol: Se le indica al modelo que actúe como experto en seguridad de redes. Esto orienta el tipo de respuesta que se espera.

- Características del flujo: Se listan los valores de las 20 variables seleccionadas, cada una con su nombre y su valor numérico: puerto de destino, duración del flujo, volúmenes de paquetes, flags TCP y tamaños de ventana.

- Clases posibles: Se enumeran las 10 categorías del dataset para que el modelo sepa entre qué opciones debe elegir.

- Instrucción de formato: Se le pide explícitamente que responda únicamente con el nombre de la clase. Esto facilita el parseo de la respuesta.

- Llamada a la respuesta: El prompt termina con “Category” para que el modelo complete directamente con la clase predicha, aprovechando su comportamiento de completar texto.

4. Inferencia y parseo de la respuesta:

El modelo genera la respuesta token a token hasta un máximo de 15. Una vez generada, se extrae la clase predicha buscando primero una coincidencia exacta con alguna de las 10 categorías, luego por subcadena, y finalmente por palabras clave asociadas a cada tipo de ataque. Si ninguna funciona, la predicción se marca como UNKNOWN.

Qué es un tokenizador?

Es importante aclarar que hace el tokenizador porque es una parte del pipeline que no existe en ninguno de los modelos anteriores.

Los LLMs no procesan texto directamente. El tokenizador convierte el texto del prompt en una secuencia de números enteros llamados *tokens*, donde cada token representa una palabra, parte de una palabra, o un carácter especial. Llama 3.2 usa un tokenizador BPE (*Byte Pair Encoding*) con un vocabulario de 128.000 tokens. La palabra "BENIGN", por ejemplo, puede ser un único token o dividirse en dos dependiendo de si aparece en el vocabulario. El modelo trabaja sobre estos números, genera tokens de respuesta, y el tokenizador los vuelve a convertir en texto al final.

Esto tiene una consecuencia importante para este experimento: el modelo no "lee" los valores numéricos de las características del flujo de la misma manera que un clasificador tradicional. Los números se convierten en tokens, como "4", "7", "9", ".", "3", y el modelo debe entender su significado a partir del contexto del prompt y de lo que aprendió durante el preentrenamiento.

5. Resultados

```
================================================================
RESULTADOS — Llama 3.2-1B sobre CICIDS-2017
================================================================

Accuracy: 0.1000 (10.00%)
Tiempo total: 3850.0s  (19.3s/registro)

                           precision    recall  f1-score   support

                  BENIGN       0.10      1.00      0.18        20
                     Bot       0.00      0.00      0.00        20
                    DDoS       0.00      0.00      0.00        20
            FTP-Patator        0.00      0.00      0.00        20
            Infiltration       0.00      0.00      0.00        20
                PortScan       0.00      0.00      0.00        20
            SSH-Patator        0.00      0.00      0.00        20
  Web Attack – Brute Force     0.00      0.00      0.00        20
Web Attack – Sql Injection     0.00      0.00      0.00        20
         Web Attack – XSS      0.00      0.00      0.00        20

               accuracy                           0.10       200
              macro avg       0.01      0.10      0.02       200
           weighted avg       0.01      0.10      0.02       200

Respuestas no parseables: 0/200 (0.0%)
```

Figura 37 - Resultados del modelo Llama 3.2-1B

En cuanto a los resultados vemos que el modelo clasifica todo el tráfico como BENIGN, acertando en las 20 instancias benignas y fallando en las 180 restantes. La precisión de BENIGN es 0,10 esto implica que 1 de cada 10 predicciones es realmente de la clase real BENIGN.

Por otro lado, ninguna de las otras 9 clases tiene un solo acierto. El F1 es 0,00 para Bot, DDoS, FTP-Patator, Infiltration, PortScan, SSH-Patator, Web Attack – Brute Force, Web Attack – Sql Injection y Web Attack – XSS. El modelo no detecta ningún ataque.

El tiempo de inferencia es de 19,3 segundos por registro en CPU, lo que se traduce en aproximadamente 64 minutos para los 200 registros. Aplicar este modelo sobre el test completo de 321.398 instancias requeriría más de 70 días de cómputo en CPU. Esto hace que la evaluación a escala sea completamente inviable sin hardware especializado.

6. Conclusión: por qué falla el modelo

El resultado no es sorprendente si se entiende cómo funciona un LLM en este contexto. Hay varias razones que explican el fallo:

Por un lado, el modelo nunca ha visto el formato de este problema durante su preentrenamiento. Llama 3.2-1B fue entrenado principalmente sobre texto: artículos, código, conversaciones, libros. Ha visto poca o ninguna información sobre cómo interpretar una tabla de estadísticas de flujo de red y clasificarla como un tipo específico de ataque. El conocimiento experto sobre ciberseguridad que pudiera haber absorbido no está estructurado de la forma que necesita el prompt.

Además el tamaño del modelo importa. Con 1.000 millones de parámetros, Llama 3.2-1B está en el extremo más pequeño de los LLMs actuales. Modelos más grandes tienen más capacidad de razonamiento y podrían comportarse de forma diferente ante este tipo de tarea. Sin embargo, ejecutarlos localmente requeriría hardware que está fuera del alcance de este trabajo.

Hay además una razón más estructural: los LLMs están diseñados para trabajar con lenguaje, no con datos numéricos. Durante el preentrenamiento aprenden relaciones entre palabras y conceptos expresados en texto. Los números los procesan como tokens sin ninguna comprensión de su valor cuantitativo o de las relaciones matemáticas entre ellos. Cuando el prompt presenta 20 valores numéricos como "Flow Duration: 0.0034" o "PSH Flag Count: 0", el modelo no los "lee" como un analista leería una tabla sino que los ve como secuencias de caracteres en un contexto que no reconoce. Esa diferencia fundamental entre razonamiento sobre lenguaje y razonamiento sobre datos numéricos estructurados explica en parte por qué el modelo colapsa en BENIGN independientemente de los valores que recibe.

Por último, el prompt es difícil de seguir para un modelo pequeño pues este pide al modelo que procese 20 valores numéricos con nombres técnicos y los relacione con 10 categorías de

ataque específicas. Para un modelo con capacidad de razonamiento limitada, esto es demasiado.

Podemos concluir que la diferencia es notable con respecto a los modelos entrenados en las secciones anteriores. El LLM en modo *zero-shot* obtiene un Macro avg F1 de 0,02, equivalente al que se obtendría prediciendo siempre la clase mayoritaria (BENIGN) sin ningún aprendizaje real."

Esto no significa que los LLMs sean inútiles para la ciberseguridad. En otras tareas como análisis de logs en lenguaje natural, explicación de alertas, generación de reglas de detección, asistencia a analistas etc los LLMs pueden ser muy útiles. Pero como clasificadores directos de flujos de red, es este caso, LLama no compite con los modelos entrenados específicamente para esta tarea.

Capítulo 5. COMPARATIVA Y ANÁLISIS CONJUNTO

Una vez entrenados y evaluados los modelos de forma individual, este capítulo los compara entre sí. Se establece primero el criterio de evaluación utilizado, con el F1-macro como métrica principal, y se analizan los resultados globales. Después se entra en el detalle analizando qué tipos de ataque detecta bien cada modelo, dónde falla y por qué, y cuál es el perfil de falsos positivos de cada uno. Se incluye también una comparación entre los modelos supervisados y el LLM evaluado. Finalmente, se concluye con la selección del modelo más adecuado para este problema y un análisis de las limitaciones del estudio

5.1 CRITERIOS DE EVALUACIÓN

Antes de comparar modelos conviene aclarar qué se valora y por qué. El macro F1 es la métrica principal porque trata todas las clases por igual, independientemente de cuántas instancias tenga cada una. Una accuracy del 99 % en este dataset no dice gran cosa ya que un modelo que clasificase todo como benigno la alcanzaría sin detectar un solo ataque.

Tampoco todos los errores tienen la misma gravedad. En ciberseguridad, lo más peligroso es un falso negativo, es decir, un ataque que pasa sin generar ninguna alerta. Un falso positivo genera una alerta innecesaria que consume tiempo del analista, pero al menos no deja la amenaza sin respuesta. Además, dentro de los falsos negativos no todos los ataques son igual de críticos. Una inyección SQL no detectada puede dar acceso a una base de datos completa pero un escaneo de puertos no detectado es un problema menor. Y confundir un Brute Force con un XSS es un error, pero al menos el sistema ha detectado actividad sospechosa y ha generado una alerta que el analista puede investigar aunque la clasificación sea incorrecta.

5.2 RESULTADOS GLOBALES

La t recoge el F1-macro obtenido por cada modelo sobre el conjunto de test. Los resultados muestran diferencias importantes entre modelos, tanto en rendimiento global como en las condiciones bajo las que se evaluó cada uno.

Modelo	Macro F1 test
Random Forest + SMOTE	0,83
KNN k=3	0,78 (submuestra)
CART + SMOTE	0,71
MLP + SMOTE	0,70
SVM + SMOTE	0,45
Llama 3.2-1B (zero-shot)	0,02

Tabla 9 - Comparativa de resultados globales por modelo

Random Forest lidera con claridad, con un F1-macro de 0,83 entrenado sobre el dataset completo con SMOTE. KNN aparece en segunda posición con 0,78, pero su evaluación se hizo sobre una submuestra de 5.000 instancias de test y no sobre el conjunto completo. Esto se debe a que, en las condiciones de este trabajo, KNN es computacionalmente inviable a escala real. Evaluar el conjunto de test completo habría requerido comparar cada una de las instancias con el millón de ejemplos de entrenamiento, lo que implicaría un tiempo de ejecución de varios días. La comparación directa con Random Forest es, por tanto, solo orientativa.

CART y MLP obtienen resultados similares, 0,71 y 0,70 respectivamente, quedando por debajo de Random Forest en todos los tipos de ataque relevantes. SVM entrenada sobre solo 15.000 instancias por limitaciones de memoria, se queda en solo 0,45.

Llama 3.2-1B queda completamente descartado. Un F1 de 0,02 equivale en la práctica a no detectar ningún ataque pues implica que el modelo clasificó casi todo el tráfico como benigno.

5.3 QUÉ DETECTA BIEN CADA MODELO

La Ilustración X muestra el F1-score por clase para cada modelo supervisado, lo que permite ver de forma más clara dónde rinde bien cada uno y dónde falla.

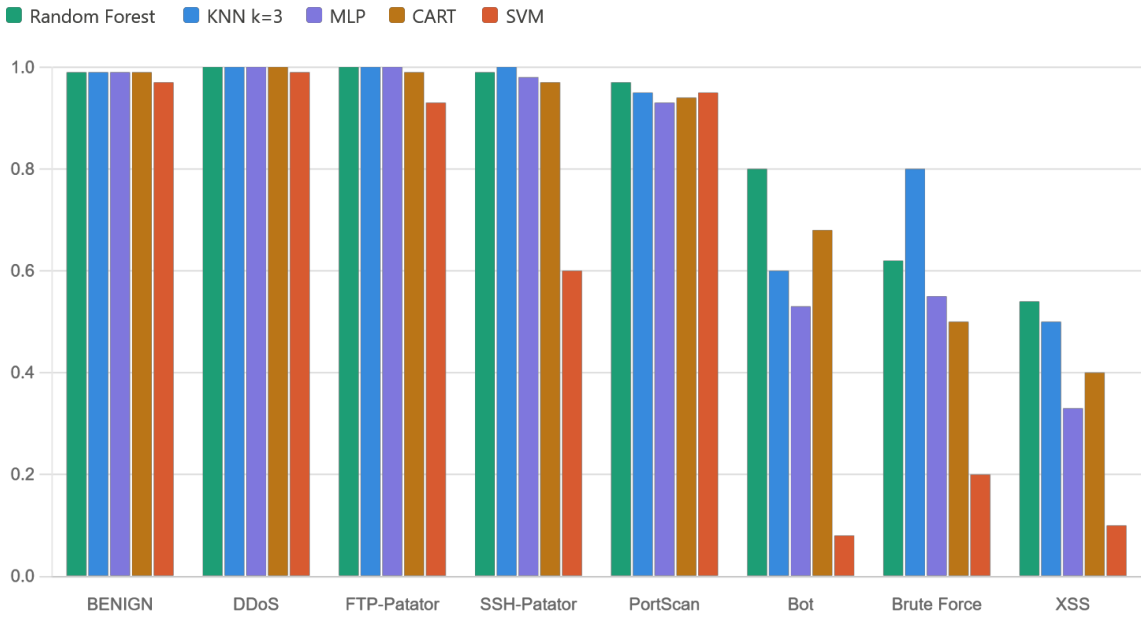

Figura 38 - F1-score por clase y modelo

DDoS, PortScan, FTP-Patator y SSH-Patator los detectan bien prácticamente todos los modelos. Tienen patrones muy consistentes y repetitivos que cualquier clasificador entrenado con datos suficientes aprende a reconocer sin problema. Incluso SVM, que es el modelo con peor rendimiento global, detecta DDoS con F1=0,99.

Donde los modelos se diferencian de verdad es en los ataques menos frecuentes y más difíciles de caracterizar a nivel de flujo de red. Esto tiene implicaciones directas en términos de riesgo, porque no todos los ataques son igual de graves.

Bot es el caso más heterogéneo entre modelos. Random Forest lo detecta con F1=0,80, CART con 0,70, KNN con 0,60, MLP con 0,53 y SVM con 0,08. Los bots suelen generar tráfico diseñado para parecerse al legítimo, lo que los hace más difíciles de identificar. Desde el punto de vista del riesgo, su detección tiene bastante importancia: no porque el bot en sí sea el ataque final, sino porque suele actuar en fases de reconocimiento previas a operaciones más sofisticadas. Detectarlo a tiempo puede prevenir un daño mayor.

Infiltration tiene solo 7 instancias en el test, lo que hace que cualquier resultado sea poco representativo estadísticamente. Random Forest obtiene el mejor resultado con F1=0,77, pero con tan pocos ejemplos la conclusión es limitada. Es un caso especialmente preocupante en cuanto a gravedad: si un atacante ya está dentro de la red, el daño potencial es muy alto. El problema es que el dataset tiene solo 36 instancias de este tipo en total, lo que hace muy difícil que cualquier modelo aprenda sus patrones de forma robusta.

Los ataques web son el punto débil generalizado. La confusión entre Brute Force y XSS aparece en todos los modelos sin excepción, porque a nivel de características de flujo ambos generan patrones similares que no se pueden separar sin acceso al contenido de los paquetes. Random Forest obtiene F1=0,63 en Brute Force y F1=0,54 en XSS; KNN sube a 0,80 en Brute Force pero baja a 0,50 en XSS; CART obtiene 0,58 en Brute Force y 0,52 en XSS; MLP se queda en torno a 0,50 en ambos; y SVM prácticamente no los detecta, con F1=0,20 y 0,10 respectivamente. Aunque XSS y Brute Force tienen un impacto algo menor que SQL Injection, siguen siendo ataques con consecuencias reales. XSS puede robar sesiones de usuario o redirigir a páginas maliciosas, y Brute Force puede dar acceso no autorizado a una cuenta si tiene éxito.

SQL Injection es el caso más preocupante, tanto por sus resultados como por su gravedad. Con solo 4 instancias en el test los números no son estadísticamente fiables. Random Forest obtiene el mejor F1 con 0,43 y el resto se mueve entre 0,01 y 0,10. La inyección SQL no genera un patrón de tráfico especialmente identificable a nivel de flujo, lo que limita la capacidad de detección de cualquier modelo basado en estas características. En términos de impacto es uno de los ataques más graves: puede dar acceso completo a una base de datos,

permitir robo o manipulación de datos y en algunos casos incluso ejecución de comandos en el servidor.

En el otro extremo están PortScan, FTP-Patator y SSH-Patator. Los tres se detectan bien por todos los modelos y generan patrones repetitivos fácilmente identificables.

- PortScan por sí solo no causa daño directo, pero es una señal de que alguien está explorando la red y suele ser un paso previo a un ataque.
- FTP-Patator y SSH-Patator son graves si tienen éxito, pero precisamente porque generan muchos intentos repetidos son de los más fáciles de detectar.
- DDoS puede dejar un servicio completamente inaccesible, pero también es el ataque más fácil de detectar para todos los modelos evaluados, con F1 cercano a 1,00 en todos los casos.

En resumen, los modelos supervisados funcionan bien donde el patrón de ataque es claro y hay suficientes datos de entrenamiento, pero fallan precisamente en los ataques que más daño pueden causar: SQL Injection, Infiltration y, en menor medida, los ataques web. Esto no es un problema exclusivo de los modelos elegidos sino una limitación estructural del dataset.

5.4 EL PROBLEMA DE LOS FALSOS POSITIVOS

Un sistema que genera demasiadas falsas alarmas no es operativamente útil ya que el analista acaba ignorándolas por saturación. La SVM es el caso más extremo:

- Bot tiene precision de 0,04 en test, lo que equivale a 24 falsas alarmas por cada ataque correctamente detectado.
- SSH-Patator tiene precision de 0,20.
- Unos 27.000 flujos benignos se clasifican erróneamente como algún tipo de ataque. Este volumen haría inviable la operación del sistema en producción.

Random Forest tiene un perfil mucho más equilibrado. Bot obtiene precision de 0,72 con recall de 0,91, un balance razonable. Las falsas alarmas existen , especialmente en Brute Force y XSS, pero en un volumen manejable.

5.5 LLMS FRENTE A MODELOS TRADICIONALES

El experimento con Llama 3.2-1B deja claro que un LLM pequeño sin entrenamiento específico no consigue clasificar flujos de red de forma útil. La accuracy del 10 % equivale a clasificar todo como BENIGN, la clase mayoritaria. Para que funcionase en este contexto habría que entrenarlo específicamente sobre datos etiquetados del dataset, lo que lo convierte básicamente en otro clasificador supervisado con un coste computacional mucho mayor.

Existen varias limitaciones. Por un lado, las del propio modelo: con 1.000 millones de parámetros, Llama 3.2-1B no tiene la capacidad de razonamiento necesaria para interpretar tablas de estadísticas de red y asociarlas a categorías de ataque sin ningún ejemplo previo. Además, estos modelos están diseñados para procesar lenguaje natural, no datos numéricos estructurados, por lo que recibir un prompt con 20 valores numéricos de un flujo de red no es el tipo de entrada para el que fueron entrenados. Por otro lado, las limitaciones computacionales: 19 segundos por predicción en CPU hacen inviable cualquier aplicación real a escala.

Esto no significa, sin embargo, que los LLMs no tengan valor en ciberseguridad. El problema aquí es el tamaño del modelo y no el enfoque en sí. Los modelos más avanzados que existen actualmente, con millones de parámetros, tienen mayores capacidades de razonamiento y podrían abordar esta tarea de forma muy diferente. El problema es que esos modelos no están disponibles públicamente y su ejecución requiere un coste computacional que está fuera del alcance de este trabajo. Es el caso de Claude Mythos, el modelo más avanzado de Anthropic, que no ha sido publicado precisamente porque sus capacidades en ciberseguridad resultaron ser considerablemente mayores de lo esperado.

5.6 SELECCIÓN DEL MODELO

Random Forest con SMOTE es el modelo más adecuado para este problema:

- el mejor macro F1 global (0,83)
- el mejor resultado en las clases más difíciles : Bot, Infiltration, Sql Injection
- el balance precision-recall más equilibrado
- no tiene las restricciones de escalabilidad de KNN o SVM. Se entrena sobre el conjunto completo y la predicción es rápida, lo que lo hace viable para un entorno de producción.

CART sería preferible en entornos donde se prioriza la interpretabilidad. Un árbol permite entender exactamente qué variables activaron cada predicción y que reglas se usaron. Esto facilita que los analistas comprendan y justifiquen cada decisión tomada por el modelo. El coste en rendimiento (macro F1=0,71) puede ser aceptable dependiendo del contexto.

KNN podría funcionar como modelo de referencia rápido, pero su coste computacional para clasificación en tiempo real lo descarta como opción principal.

SVM y MLP quedan en posiciones inferiores. La SVM tiene demasiadas falsas alarmas y restricciones computacionales. La MLP obtiene resultados razonables pero queda consistentemente por debajo de Random Forest.

5.7 LIMITACIONES

Las limitaciones más importantes no son de los modelos, sino del dataset. El CICIDS2017 tiene casi una década y los patrones de ataque han evolucionado. Las clases con muy pocas instancias, como Sql Injection con 21 registros totales, Infiltration con 36, limitan las conclusiones sobre esos tipos de ataque. Por último, las características no incluyen información sobre el contenido de los paquetes que sería especialmente útil para detectar ataques de aplicación como la inyección SQL o XSS.

Capítulo 6. CONCLUSIONES Y TRABAJO FUTURO

6.1 CONCLUSIONES

El trabajo cubre los objetivos planteados. Se estudian los fundamentos teóricos de cada modelo, se diseña una metodología experimental común que hace los resultados comparables, y se entrena y evalúa cada modelo sobre datos reales de tráfico de red. La comparación va más allá del F1-macro: incluye el comportamiento frente a clases desbalanceadas, el volumen de falsas alarmas, el coste computacional y la interpretabilidad.

El modelo seleccionado es Random Forest con SMOTE, con un F1-macro de 0,83 sobre el test completo. Es el que mejor combina rendimiento global, detección en clases difíciles y viabilidad operativa. CART sería la alternativa en contextos donde importa más poder explicar cada decisión que maximizar el rendimiento.

Dicho esto, los resultados dejan algo claro que va más allá de qué modelo es mejor: los clasificadores supervisados basados en características de flujo tienen una limitación importante. Detectan bien los ataques con patrones repetitivos como DDoS, PortScan o FTP-Patator, pero fallan en los más graves. SQL Injection, con un F1 máximo de 0,43, es prácticamente indetectable con este enfoque. Infiltration tampoco mejora mucho. No es un problema de los algoritmos sino del tipo de información disponible ya que las estadísticas de flujo no capturan el contenido de los paquetes.

El Machine Learning supervisado lleva años siendo parte de los sistemas de detección de intrusiones y los resultados de este trabajo son consistentes. Estos modelos funcionan bien para amenazas conocidas y bien representadas en el entrenamiento, pero tienen dificultades con ataques nuevos o poco frecuentes. En la práctica, un IDS basado en estos modelos necesita actualizarse continuamente con datos nuevos, y aun así siempre habrá un margen de amenazas que no detectará.

Los LLMs abren nuevas posibilidades, aunque no para el mismo problema. El experimento con Llama 3.2-1B lo ilustra bien. Un modelo pequeño sin ajuste específico no puede clasificar flujos de red y los 19 segundos por predicción en CPU hacen inviable cualquier uso real. El problema no es el enfoque LLM en sí, sino el tamaño y la falta de ajuste. Modelos mucho más grandes muestran un comportamiento completamente distinto y prometedor.

El caso de Claude Mythos Preview es el ejemplo más concreto disponible hoy. Mythos no detecta tráfico malicioso en tiempo real sino que revisa código fuente, analiza configuraciones y detecta vulnerabilidades antes de que sean explotadas. Se sabe que ha identificado de forma autónoma más de 10.000 vulnerabilidades en sistemas reales y su acceso fue restringido precisamente porque sus capacidades resultaron mayores de lo esperado. Este modelo detecta antes de que el ataque ocurra reduciendo la superficie de exposición. En cambio, los clasificadores supervisados actúan después, cuando el tráfico malicioso ya está circulando por la red. Son fases distintas del mismo problema. Esta complementariedad es probablemente hacia dónde va la ciberseguridad en los próximos años: modelos supervisados cubriendo la detección en tiempo real, y LLMs avanzados apoyando la revisión activa de código y la gestión de vulnerabilidades de día cero. Aunque actualmente, el reto más que técnico es de escala y acceso pues modelos como Mythos no están disponibles públicamente y su coste computacional está fuera del alcance de la mayoría de organizaciones. Mientras eso no cambie, los clasificadores supervisados siguen siendo la opción práctica para la detección.

Desde el punto de vista de la sostenibilidad, la ciberseguridad tiene un impacto económico y social que no siempre se tiene en cuenta. Un ataque de ransomware puede paralizar un hospital, y una inyección SQL mal gestionada puede exponer datos de millones de personas. Por ello, automatizar la detección de amenazas reduce el impacto de incidentes que pueden tener grandes consecuencias sobre servicios públicos, infraestructuras y entornos de trabajo. En ese sentido, este trabajo se alinea con los ODS 8, 9 y 12. La detección temprana de amenazas contribuye directamente a infraestructuras más resilientes, equipos de seguridad menos saturados y un uso más eficiente de los recursos digitales.

6.2 TRABAJO FUTURO

La mejora más directa sería actualizar el dataset. El CICIDS2017 tiene casi diez años y los patrones de ataque han cambiado. Datasets más recientes permitirían ver si las conclusiones se mantienen. Además, incorporar inspección del contenido de los paquetes mejoraría considerablemente la detección de ataques de aplicación como SQL Injection o XSS, que con solo características de flujo son casi indetectables.

Por otro lado, explorar la detección de anomalías para identificar ataques no vistos durante el entrenamiento resultaría muy útil ya que en entornos reales los ataques nuevos son el problema más frecuente.

Finalmente, la línea más prometedora es integrar ambos enfoques: modelos supervisados para la detección en tiempo real y LLMs para explicar las alertas al analista en lenguaje natural o para revisar el código de los sistemas monitorizados. Reducir el tiempo de respuesta ante un incidente sigue siendo uno de los principales retos operativos en los equipos de seguridad actuales.

Capítulo 7. BIBLIOGRAFÍA

AI Security Institute (AISI). (2026). *Our evaluation of Claude Mythos Preview's cyber capabilities*. UK Department for Science, Innovation and Technology. https://www.aisi.gov.uk/blog/our-evaluation-of-claude-mythos-previews-cyber-capabilities

Akamai. (2024, 14 de agosto). Akamai prevents record-breaking DDoS attack targeting major U.S. customer. https://www.akamai.com/blog/security/akamai-prevents-record-breaking-ddos-attack-major-us-customer

Akamai. (s.f.). *What is ensemble machine learning?* https://www.akamai.com/glossary/what-is-ensemble-machine-learning

Akamai. (s.f.). *¿Qué es el ransomware WannaCry?* https://www.akamai.com/es/glossary/what-is-wannacry-ransomware

Amazon Web Services. (s.f.). *¿Qué es una red neuronal?* https://aws.amazon.com/es/what-is/neural-network/

Anthropic. (2026). *Introducing Claude Mythos Preview and Project Glasswing*. Anthropic Blog. https://www.anthropic.com

Belcic, I., y Stryker, C. (2025, 17 de noviembre). Supervised learning. IBM. https://www.ibm.com/think/topics/supervised-learning

Bergmann, D. (2025a, 17 de noviembre). Semi-supervised learning. IBM. https://www.ibm.com/think/topics/semi-supervised-learning

Bergmann, D. (2025b, 17 de noviembre). What is machine learning? IBM. https://www.ibm.com/think/topics/machine-learning

Breachsense. (2024, 8 de diciembre). *Equifax data breach: Causes and aftermath*. https://www.breachsense.com/blog/equifax-data-breach/

CART (classification and regression tree) in machine learning. (2022, 23 de septiembre). GeeksforGeeks. https://www.geeksforgeeks.org/machine-learning/cart-classification-and-regression-tree-in-machine-learning/

Centre for Emerging Technology and Security (CETaS). (2026). *Claude Mythos: What does Anthropic's new model mean for the future of cybersecurity?* The Alan Turing Institute. https://cetas.turing.ac.uk

Chen, M. (2024, 25 de noviembre). What is machine learning? Oracle. https://www.oracle.com/latam/artificial-intelligence/machine-learning/what-is-machine-learning/

Cisco. (s.f.). *What is spam vs phishing?* https://www.cisco.com/site/us/en/learn/topics/security/what-is-spam-vs-phishing.html

Clasificación con árboles de decisión: el algoritmo CART. (2021, 11 de abril). Codificando Bits. https://codificandobits.com/blog/clasificacion-arboles-decision-algoritmo-cart/

Cloud Security Alliance (CSA). (2026). *Claude Mythos: AI vulnerability discovery and containment failures* (Version 1.0). CSA AI Safety Initiative. https://labs.cloudsecurityalliance.org

Cloudflare. (s.f.). *WannaCry ransomware*. https://www.cloudflare.com/es-es/learning/security/ransomware/wannacry-ransomware/

Cloudflare. (s.f.). *What is a brute force attack?* https://www.cloudflare.com/es-es/learning/bots/brute-force-attack/

Cloudflare. (s.f.). *What is DNS cache poisoning?* https://www.cloudflare.com/es-es/learning/dns/dns-cache-poisoning/

Cloudflare. (s.f.). *¿Qué es un ataque de denegación de servicio (DoS)?* https://www.cloudflare.com/es-es/learning/ddos/glossary/denial-of-service/

Cloudflare. (2024, 20 de octubre). Cloudflare mitigates 5.6 Tbps DDoS attack — one of the largest ever recorded. https://cyberscoop.com/cloudflare-biggest-ddos-attack-mirai-variant-botnet

CrowdStrike. (s.f.). *Adware*. https://www.crowdstrike.com/en-us/cybersecurity-101/malware/adware/

CrowdStrike. (s.f.). *Computer worm*. https://www.crowdstrike.com/en-us/cybersecurity-101/malware/computer-worm/

CrowdStrike. (s.f.). *Kerberoasting*. https://www.crowdstrike.com/en-us/cybersecurity-101/cyberattacks/kerberoasting/

CrowdStrike. (s.f.). *Rootkits*. https://www.crowdstrike.com/en-us/cybersecurity-101/malware/rootkits/

CrowdStrike. (s.f.). *Types of social engineering attacks*. https://www.crowdstrike.com/en-us/cybersecurity-101/social-engineering/types-of-social-engineering-attacks/

CrowdStrike. (s.f.). *What is a Trojan Horse?* https://www.crowdstrike.com/en-us/cybersecurity-101/malware/trojans/

El Economista. (2026, mayo). La IA de Mythos agiganta la brecha de seguridad en administraciones, banca y sanidad. https://www.eleconomista.es

Electronic Privacy Information Center (EPIC). (s.f.). *Equifax data breach*. https://archive.epic.org/privacy/data-breach/equifax/

Federal Bureau of Investigation (FBI). (2020, 10 de febrero). Chinese hackers charged in Equifax breach. https://www.fbi.gov/news/stories/chinese-hackers-charged-in-equifax-breach-021020

Federal Trade Commission (FTC). (s.f.). *Equifax data breach settlement*. https://www.ftc.gov/enforcement/refunds/equifax-data-breach-settlement

Fortinet. (s.f.). *Spyware*. https://www.fortinet.com/resources/cyberglossary/spyware

Fortinet. (s.f.). *Tipos de virus informáticos*. https://www.fortinet.com/lat/resources/cyberglossary/computer-virus

Fortinet. (s.f.). *Types of phishing attacks*. https://www.fortinet.com/resources/cyberglossary/types-of-phishing-attacks

Fortinet. (s.f.). *What is malware?* https://www.fortinet.com/resources/cyberglossary/malware

Fortinet. (s.f.). *What is ransomware?* https://www.fortinet.com/resources/cyberglossary/ransomware

Fortinet. (2024). *Types of cyber attacks*. https://www.fortinet.com/lat/resources/cyberglossary/types-of-cyber-attacks

Fortinet. (2025). *¿Qué es un ataque de Smurf?* https://www.fortinet.com/lat/resources/cyberglossary/smurf-attack

Fortune. (2026, 7 de abril). Anthropic is giving companies, including Amazon, Apple, and Microsoft, access to its unreleased Claude Mythos model to prepare cybersecurity defense. https://www.fortune.com

GeeksforGeeks. (2017, 14 de abril). K-nearest neighbor (KNN) algorithm. https://www.geeksforgeeks.org/machine-learning/k-nearest-neighbours/

GeeksforGeeks. (2025, 13 de noviembre). Support vector machine (SVM) algorithm. https://www.geeksforgeeks.org/machine-learning/support-vector-machine-algorithm

Google Cloud Threat Intelligence / Mandiant. (2023, 9 de noviembre). Sandworm disrupts power in Ukraine using a novel attack against operational technology. https://cloud.google.com/blog/topics/threat-intelligence/sandworm-disrupts-power-ukraine-operational-technology

Help Net Security. (2026, 26 de mayo). Anthropic Claude Mythos identified 10,000+ software flaws. https://www.helpnetsecurity.com

House Committee on Oversight and Government Reform. (2018). *The Equifax data breach*. U.S. Government Printing Office. https://oversight.house.gov/wp-content/uploads/2018/12/Equifax-Report.pdf

IBM. (s.f.). *Phishing*. https://www.ibm.com/think/topics/phishing

IBM. (s.f.). *Ransomware*. https://www.ibm.com/think/topics/ransomware

IBM. (s.f.). *What is malware?* https://www.ibm.com/think/topics/malware

Imperva. (s.f.). *Social engineering attack*. https://www.imperva.com/learn/application-security/social-engineering-attack/

Invicti. (2024). Web security injection attacks in AppSec: Types, tools, examples. https://www.invicti.com/blog/web-security/top-dangerous-injection-attacks/

Investopedia. (s.f.). *Adware*. https://www.investopedia.com/terms/a/adware.asp

Kaspersky. (s.f.). *Brute force attack*. https://www.kaspersky.com/resource-center/definitions/brute-force-attack

Kaspersky. (s.f.). *Ransomware WannaCry*. https://www.kaspersky.com/resource-center/threats/ransomware-wannacry

Kavlakoglu, E. (2025, 17 de noviembre). What is a decision tree? IBM. https://www.ibm.com/think/topics/decision-trees

Kavlakoglu, E. (2025, 17 de noviembre). What is the KNN algorithm? IBM. https://www.ibm.com/think/topics/knn

Kim, D., y Solomon, M. G. (2021). *Fundamentals of information systems security* (4th ed.). Jones & Bartlett Learning.

Kuhn, M., y Johnson, K. (2013). *Applied predictive modeling*. Springer. https://doi.org/10.1007/978-1-4614-6849-3

Kurose, J., y Ross, K. (2021). *Computer networking: A top-down approach* (8th ed.). Pearson.

Lee, F. (2025, 27 de noviembre). ¿Qué es una red neuronal? IBM. https://www.ibm.com/es-es/think/topics/neural-networks

Liao, H. J., Lin, C. H. R., Lin, Y. C., y Tung, K. Y. (2013). Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications, 36*(1), 16–24.

LISA News. (2026). Mythos, el nuevo paradigma de la ciberseguridad con la IA. https://www.lisanews.org

Maklin, C. (2022, 14 de mayo). Synthetic minority over-sampling technique (SMOTE). Medium. https://medium.com/@corymaklin/synthetic-minority-over-sampling-technique-smote-7d419696b88c

Malwarebytes. (s.f.). *History of malware*. https://www.malwarebytes.com/malware

Malwarebytes. (s.f.). *What is a rootkit?* https://www.malwarebytes.com/rootkit

Malwarebytes. (2024). *Adware*. https://www.malwarebytes.com/es/adware

Malwarebytes. (2024). *Browser hijacker*. https://www.malwarebytes.com/blog/threats/browser-hijacker

ManageEngine. (2024). SQL injection, web application attack and cross-site scripting: The differences and attack anatomy. https://www.manageengine.com/log-management/cyber-security/sql-injection-web-application-attack-and-cross-site-scripting-the-differences-and-attack-anatomy.html

Mancilla, E. (2022, 20 de diciembre). Machine learning: definición, métodos y ejemplos. InvGate. https://blog.invgate.com/es/machine-learning

Microsoft. (s.f.). *SMOTE*. Azure Machine Learning documentation. https://learn.microsoft.com/en-us/azure/machine-learning/component-reference/smote

Murel, J., y Kavlakoglu, E. (2025, 17 de noviembre). Reinforcement learning. IBM. https://www.ibm.com/think/topics/reinforcement-learning

Murel, J., y Kavlakoglu, E. (2025, 17 de noviembre). What is ensemble learning? IBM. https://www.ibm.com/think/topics/ensemble-learning

National Institute of Standards and Technology (NIST). (s.f.). *Social engineering*. https://csrc.nist.gov/glossary/term/social_engineering

Netwrix Blog. (2024). *Los mayores ciberataques de la historia*. https://blog-netwrix-com/biggest-cyber-attacks-in-history

Palo Alto Networks. (s.f.). *What is spyware?* https://www.paloaltonetworks.com/cyberpedia/what-is-spyware

postquantum.com. (2026, 8 de abril). Anthropic's Mythos Preview and the end of a twenty-year cybersecurity equilibrium. https://postquantum.com

SAS. (s.f.). *A guide to machine learning algorithms and their applications*. https://www.sas.com/en_ae/insights/articles/analytics/machine-learning-algorithms-guide.html

scikit-learn developers. (s.f.). *1.4. Support vector machines*. scikit-learn user guide. https://scikit-learn.org/stable/modules/svm.html

Security Journey. (2024). OWASP Top 10: Injection attacks explained. https://www.securityjourney.com/post/owasp-top-10-injection-attacks-explained

StormWall. (2024). *DDoS attack statistics 2024*. https://stormwall.network/resources/blog/ddos-attack-statistics-2024

Tahir, M. (2026, abril). Assessing Anthropic Claude Mythos Preview's cybersecurity capabilities. Medium. https://medium.com/@tahirbalarabe2

TechTarget. (2024). *Dictionary attack*. https://www.techtarget.com/searchsecurity/definition/dictionary-attack

The Hacker News. (2025, 3 de septiembre). Malicious npm packages exploit Ethereum smart contracts to target crypto developers. https://thehackernews.com/2025/09/malicious-npm-packages-exploit-ethereum.html

UNIE Universidad. (2024, 9 de julio). ¿Qué son las redes neuronales y cómo se aplican a la inteligencia artificial? https://www.universidadunie.com/blog/que-son-redes-neuronales

University of California, Santa Cruz (UCSC). (2024, 7 de mayo). Understanding Industroyer one and two: Malware that attacked Ukraine's power grid. https://news.ucsc.edu/2024/05/ukraine-cybersecurity/

Wikipedia. (s.f.). *Trojan horse (computing)*. https://en.wikipedia.org/wiki/Trojan_horse_(computing)

ANEXO I: ALINEACIÓN DEL PROYECTO CON LOS ODS

El presente proyecto se alinea con varios de los Objetivos de Desarrollo Sostenible (ODS) definidos en la Agenda 2030 de las Naciones Unidas, contribuyendo al desarrollo de infraestructuras digitales más seguras, eficientes y sostenibles.

ODS 9: Industria, innovación e infraestructura

El análisis y aplicación de modelos de Machine Learning para la detección de amenazas en redes contribuye a que las infraestructuras tecnológicas sean más robustas. Hoy en día, la ciberseguridad es un factor cada vez más importante para garantizar la continuidad de los servicios digitales, que están en la base de muchos procesos industriales y de innovación. Mejorar la capacidad de detección automática de amenazas va en esa línea.

ODS 8: Trabajo decente y crecimiento económico

Automatizar parte de las tareas de detección y análisis de amenazas puede reducir la carga operativa de los equipos de seguridad, dejándoles más tiempo para tareas que requieren criterio humano, como el análisis estratégico o la respuesta a incidentes complejos. En sectores muy dependientes de las tecnologías de la información, esto se traduce en entornos de trabajo más eficientes.

ODS 12: Producción y consumo responsables

Detectar amenazas a tiempo evita interrupciones de servicio, pérdidas de información y los costes que conlleva gestionar un incidente de seguridad. En ese sentido, optimizar estos sistemas contribuye a un uso más eficiente de los recursos tecnológicos y a una gestión más responsable y sostenible de las infraestructuras digitales.

ANEXO II

REPOSITORIO DE CÓDIGO

Con el objetivo de facilitar la reproducibilidad de los experimentos realizados se ha habilitado un repositorio GitHub que contiene la implementación completa desarrollada durante este Trabajo Fin de Grado.

Repositorio: **https://github.com/VictoriaSdL/tfg-ml-ciberseguridad**

El repositorio incluye:

- Implementación de los modelos KNN, CART, Random Forest, MLP y SVM aplicados al dataset CICIDS2017.
- Experimento con el modelo de lenguaje Llama 3.2-1B e

Para ejecutar los scripts es necesario descargar previamente los ficheros CSV del dataset CICIDS2017 desde https://www.kaggle.com/datasets/chethuhn/network-intrusion-dataset y colocarlos en una carpeta llamada dataset en el mismo directorio que los scripts. Los ficheros concretos utilizados se detallan en el README del repositorio.

INFORMACIÓN SOBRE EL DATASET:

El dataset utilizado en este proyecto es el CICIDS2017, generado por la Universidad de New Brunswick. Contiene más de 1.600.000 flujos de red etiquetados en diez categorías de tráfico, extraídos de capturas reales durante cinco días laborables. Cada instancia está descrita por 78 variables numéricas calculadas a partir de las características del flujo, como duración, volumen de paquetes, flags TCP o tiempos entre llegadas. La siguiente Figura 39 es una muestra representativa con las 9 variables más interpretables.

Destination Port	Flow Duration	Total Fwd Packets	Total Backward Packets	Flow Bytes/s	Flow Packets/s	SYN Flag Count	ACK Flag Count	Label
54865	3	2	0	4000000	666666,67	0	1	BENIGN
22	1266342	41	44	7595,1	67,12	0	0	BENIGN
389	113095465	48	24	174,01	0,64	1	1	BENIGN
22	166	1	1	0	12048,19	0	1	BENIGN
88	640	7	4	1246875	17187,5	0	0	BENIGN

Figura 39 - Muestra representativa del dataset CICIDS2017 (variables seleccionadas)

A continuación se recogen las 78 variables que componen el dataset:

1. Destination Port
2. Flow Duration
3. Total Fwd Packets
4. Total Backward Packets
5. Total Length of Fwd Packets
6. Total Length of Bwd Packets
7. Fwd Packet Length Max
8. Fwd Packet Length Min
9. Fwd Packet Length Mean
10. Fwd Packet Length Std
11. Bwd Packet Length Max
12. Bwd Packet Length Min
13. Bwd Packet Length Mean
14. Bwd Packet Length Std
15. Flow Bytes/s
16. Flow Packets/s
17. Flow IAT Mean
18. Flow IAT Std
19. Flow IAT Max
20. Flow IAT Min
21. Fwd IAT Total
22. Fwd IAT Mean
23. Fwd IAT Std
24. Fwd IAT Max
25. Fwd IAT Min
26. Bwd IAT Total
27. Bwd IAT Mean
28. Bwd IAT Std
29. Bwd IAT Max
30. Bwd IAT Min
31. Fwd PSH Flags
32. Bwd PSH Flags
33. Fwd URG Flags
34. Bwd URG Flags
35. Fwd Header Length
36. Bwd Header Length
37. Fwd Packets/s
38. Bwd Packets/s
39. Min Packet Length
40. Max Packet Length
41. Packet Length Mean
42. Packet Length Std
43. Packet Length Variance
44. FIN Flag Count
45. SYN Flag Count
46. RST Flag Count
47. PSH Flag Count
48. ACK Flag Count
49. URG Flag Count
50. CWE Flag Count
51. ECE Flag Count
52. Down/Up Ratio
53. Average Packet Size
54. Avg Fwd Segment Size
55. Avg Bwd Segment Size
56. Fwd Header Length
57. Fwd Avg Bytes/Bulk
58. Fwd Avg Packets/Bulk
59. Fwd Avg Bulk Rate
60. Bwd Avg Bytes/Bulk
61. Bwd Avg Packets/Bulk
62. Bwd Avg Bulk Rate
63. Subflow Fwd Packets
64. Subflow Fwd Bytes
65. Subflow Bwd Packets
66. Subflow Bwd Bytes
67. Init_Win_bytes_forward
68. Init_Win_bytes_backward
69. act_data_pkt_fwd
70. min_seg_size_forward
71. Active Mean
72. Active Std
73. Active Max
74. Active Min
75. Idle Mean
76. Idle Std
77. Idle Max
78. Idle Min