GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

APLICACIÓN WEB PARA LA DETECCIÓN DE INTRUSIONES EN REDES MEDIANTE MACHINE LEARNING

Autor: Paula Bonet Sánchez

Director: David Martín-Corral Calvo

Madrid

Declaración de originalidad

Declaro bajo mi responsabilidad que el Proyecto presentado con el título **Aplicación Web Para La Detección De Intrusiones En Redes Mediante Machine Learning** e la ETS de Ingeniería – ICAI de la Universidad Pontificia Comillas en el curso académico 2025-2026 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.

X 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 IA ha sido utilizada para apoyar la lluvia de ideas inicial, para la correcta estructura y organización de los capítulos, mejorar la calidad de la redacción definitiva, generar un gráfico incluido en el documento y revisar el texto en busca de errores ortográficos. En todo momento ha sido utilizada como herramienta de apoyo, nunca como sustituta del desarrollo intelectual del proyecto.

[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…)

Firmado (alumno): Paula Bonet Sánchez
Fecha: 28/06/2026

Autorización para la entrega del Proyecto

El Director del Proyecto
MARTIN CORRAL CALVO DAVID - 03909768K Digitally signed by MARTIN CORRAL CALVO DAVID - 03909768K Date: 2026.06.30 10:42:51 +02'00'
Fdo:
Fecha:

GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

APLICACIÓN WEB PARA LA DETECCIÓN DE INTRUSIONES EN REDES MEDIANTE MACHINE LEARNING

Autor: Paula Bonet Sánchez

Director: David Martín-Corral Calvo

Madrid

Agradecimientos

APLICACIÓN WEB PARA LA DETECCIÓN DE INTRUSIONES EN REDES MEDIANTE MACHINE LEARNING

Autor: Bonet Sánchez, Paula.
Director: Martín-Corral Calvo, David.
Entidad Colaboradora: ICAI – Universidad Pontificia Comillas

RESUMEN DEL PROYECTO

Este trabajo presenta NetRadar, una aplicación web para la detección y clasificación de intrusiones en red mediante aprendizaje automático. Sobre el conjunto de datos NSL-KDD se entrenan y comparan cuatro modelos que asignan cada conexión a una de cinco familias (Normal, DoS, Probe, R2L y U2R). La evaluación sigue el protocolo independiente KDDTrain+/KDDTest+, que mide la capacidad de generalizar ante ataques no vistos. XGBoost obtuvo el mejor resultado, con un 76,7% de precisión global y un 59,7% de F1-macro: detecta con fiabilidad los ataques de mayor volumen, mientras que las intrusiones más sigilosas siguen siendo el reto principal.

Palabras clave: detección de intrusiones, NSL-KDD, clasificación multiclase, aprendizaje automático, XGBoost, FastAPI

Introducción

Durante años, la defensa de las redes corporativas se ha apoyado en un modelo perimetral basado en firmas: cortafuegos y sistemas de detección que comparan el tráfico con patrones de ataques ya conocidos. Este enfoque resulta eficaz frente a amenazas catalogadas, pero comparte una debilidad de fondo: no reconoce lo que no ha visto antes. Ante un volumen de ataques en crecimiento sostenido y un coste económico cada vez mayor de las brechas de seguridad, el sector ha desplazado parte de su atención hacia técnicas de aprendizaje automático, capaces de inferir el carácter malicioso de una conexión a partir de sus características y no solo de una firma exacta. En este contexto se enmarca NetRadar, que aplica esa idea a un problema concreto: no limitarse a separar tráfico legítimo de malicioso, sino identificar el tipo de ataque.

Definición del proyecto

El objetivo del proyecto es diseñar y desarrollar una aplicación web que permita a un analista cargar capturas de tráfico de red y obtener, para cada conexión, una clasificación dentro de cinco categorías: tráfico normal y cuatro familias de ataque (denegación de servicio, sondeo,

acceso remoto no autorizado y escalada de privilegios). El sistema se concibe como una herramienta de análisis forense de carácter diferido, opera sobre capturas exportadas, no sobre tráfico en vivo, y como sistema de detección, no de prevención. El alcance abarca el ciclo completo: el tratamiento de los datos, el entrenamiento y la comparación de varios modelos de clasificación, y el desarrollo de una plataforma funcional que integre esos modelos y registre el historial de los análisis realizados.

Descripción del sistema

NetRadar se estructura en tres capas. La capa de cliente es una aplicación web de página única, ejecutada en el navegador, desde la que el analista se autentica, carga ficheros y consulta los resultados. La capa de servidor, construida con FastAPI, expone una API REST que gestiona la autenticación, ejecuta los análisis y atiende las consultas del historial; las credenciales se almacenan cifradas mediante hash bcrypt. La capa de persistencia emplea SQLite para registrar usuarios y análisis. La Figura 1 resume esta arquitectura.

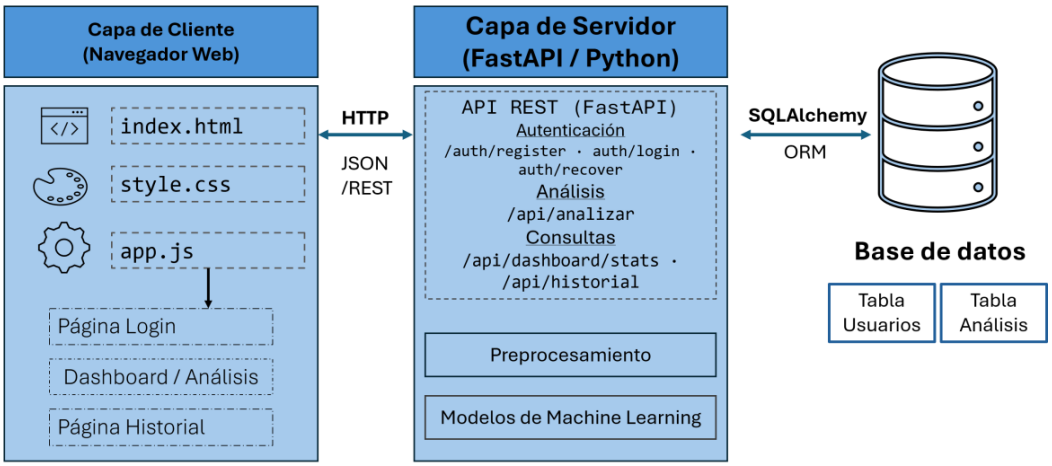

Figura 1 Arquitectura de 3 capas de NetRadar

El núcleo analítico se apoya en un tratamiento de datos cuidadoso. En una única fase de preprocesamiento se limpia el conjunto NSL-KDD, se agrupan los ataques en sus cinco familias y se ajustan los codificadores de variables categóricas y el escalador, que quedan guardados para aplicarse de forma idéntica durante el entrenamiento y en cada nuevo análisis. Sobre estos datos se entrenan cuatro modelos: Random Forest, máquinas de vectores de soporte (SVM), Gradient Boosting y XGBoost; todos ellos con estrategias de compensación del fuerte desequilibrio entre clases, ya que las categorías más peligrosas son también las menos frecuentes. El servidor carga los modelos entrenados al arrancar y los aplica bajo demanda, devolviendo para cada modelo la distribución de amenazas detectadas y una muestra detallada de las conexiones analizadas.

Resultados

La evaluación se realizó con el protocolo estándar de NSL-KDD, entrenando con KDDTrain+ y midiendo sobre el conjunto independiente KDDTest+, que incluye ataques ausentes del entrenamiento. Bajo este criterio exigente, los cuatro modelos se situaron en una banda de precisión del 74-77%, coherente con la literatura para esta tarea, y XGBoost resultó el más equilibrado (76,7% de precisión y 59,7% de F1-macro), además del más rápido en inferencia. El análisis por familias, recogido en la Figura 2, muestra un patrón claro: los ataques de gran volumen (DoS y Probe) se detectan con solvencia, mientras que las intrusiones minoritarias y sigilosas presentan más dificultad. La estrategia de balanceo permitió recuperar buena parte de la categoría de escalada de privilegios (U2R), pero los accesos remotos no autorizados (R2L), muy parecidos al tráfico legítimo, siguen siendo el caso más complejo. El error predominante de todos los modelos consiste en clasificar un ataque como tráfico normal, es decir, falsos negativos, el tipo de fallo más delicado en un sistema de detección.

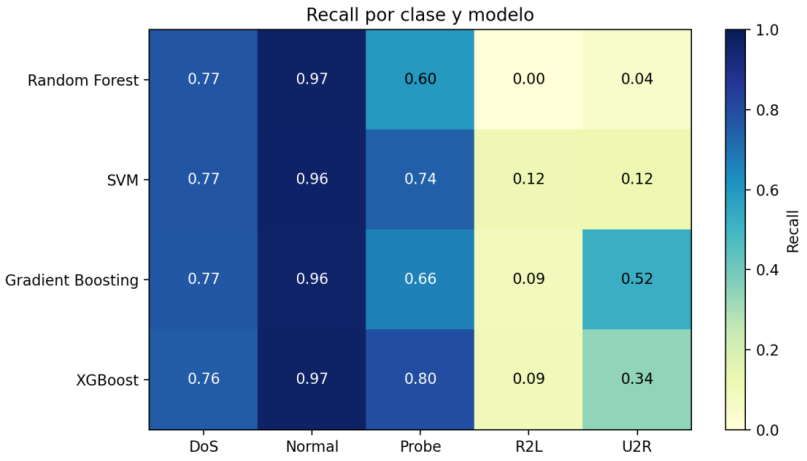

Figura 2 Recall por clase y modelo (mapa de calor sobre KDDTest+).

Conclusiones

El proyecto cumple su objetivo: NetRadar integra, en una herramienta reproducible y utilizable, todo el recorrido desde el dato en bruto hasta la clasificación multiclase de intrusiones, con resultados consistentes con el estado del arte. Más allá de las cifras, el trabajo deja una lectura crítica de valor: una precisión global elevada puede ocultar un sistema que deja pasar precisamente los ataques más peligrosos, por lo que la evaluación honesta de un detector exige mirar el comportamiento por clase y no un único porcentaje. XGBoost se perfila como la opción más adecuada para su despliegue, y las principales líneas

de mejora, validación con conjuntos de datos más recientes, calibración del umbral de decisión y detección de amenazas desconocidas, quedan planteadas como trabajo futuro.

WEB APPLICATION FOR NETWORK INTRUSION DETECTION USING MACHINE LEARNING

Author: Bonet Sánchez, Paula.
Supervisor: Martín-Corral Calvo, David.
Collaborating Entity: ICAI – Universidad Pontificia Comillas

ABSTRACT

This work presents NetRadar, a web application for detecting and classifying network intrusions through machine learning. Using the NSL-KDD dataset, four models are trained and compared to assign each connection to one of five families (Normal, DoS, Probe, R2L and U2R). Evaluation follows the independent KDDTrain+/KDDTest+ protocol, which measures the ability to generalise to unseen attacks. XGBoost achieved the best result, with 76.7% overall accuracy and 59.7% macro F1-score: it detects high-volume attacks, while stealthier intrusions remain the main challenge.

Keywords: intrusion detection, NSL-KDD, multiclass classification, machine learning, XGBoost, FastAPI

Introduction

For years, the defence of corporate networks has relied on a signature-based perimeter model: firewalls and detection systems that match traffic against patterns of already known attacks. This approach is effective against catalogued threats, but it shares one underlying weakness: it does not recognise what it has not seen before. Faced with a steadily growing volume of attacks and an increasing economic cost of security breaches, the field has shifted part of its attention towards machine learning techniques, which can infer the malicious nature of a connection from its characteristics rather than from an exact signature. NetRadar applies this idea to a specific problem: going beyond separating legitimate from malicious traffic to identify the type of attack.

Project Definition

The goal of the project is to design and develop a web application that allows an analyst to upload network traffic captures and obtain, for each connection, a classification into five categories: normal traffic and four attack families (denial of service, probing, unauthorised remote access and privilege escalation). The system is conceived as an offline forensic analysis tool —it operates on exported captures rather than live traffic— and as a detection rather than a prevention system. Its scope covers the full cycle: data processing, the training

and comparison of several classification models, and the development of a functional platform that integrates those models and records the history of the analyses performed.

System Description

NetRadar is organised in three layers. The client layer is a single-page web application, executed in the browser, from which the analyst authenticates, uploads files and reviews results. The server layer, built with FastAPI, exposes a REST API that handles authentication, runs the analyses and serves history queries; credentials are stored using bcrypt hashing. The persistence layer uses SQLite to store users and analyses. Figure 1 summarises this architecture.

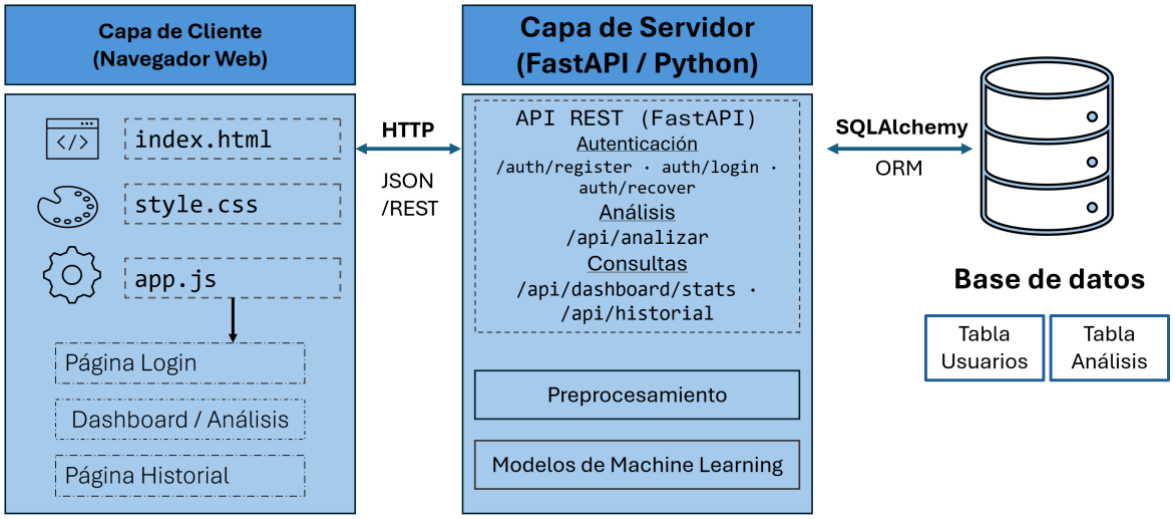

Figure 1 Three-layer architecture of NetRadar

The analytical core relies on careful data handling. In a single preprocessing stage, the NSL-KDD dataset is cleaned, attacks are grouped into their five families, and the categorical encoders and the scaler are fitted and saved, so that they are applied identically during training and for every new analysis. Four models —Random Forest, support vector machines (SVM), Gradient Boosting and XGBoost— are then trained, all of them with strategies to compensate for the strong class imbalance, since the most dangerous categories are also the least frequent. The server loads the trained models at start-up and applies them on demand, returning, for each model, the distribution of detected threats and a detailed sample of the analysed connections.

Results

Evaluation followed the standard NSL-KDD protocol, training on KDDTrain+ and measuring on the independent KDDTest+ set, which contains attacks absent from training. Under this demanding criterion, the four models fell within an accuracy band of 74–77 %, consistent with the literature for this task, and XGBoost proved the most balanced (76.7 % accuracy and 59.7 % macro F1-score) as well as the fastest at inference. The per-family analysis, shown in Figure 2, reveals a clear pattern: high-volume attacks (DoS and Probe) are detected reliably, whereas minority and stealthy intrusions are more difficult. The balancing strategy recovered much of the privilege-escalation category (U2R), but unauthorised remote access attacks (R2L), which closely resemble legitimate traffic, remain the hardest case. The predominant error of all models is classifying an attack as normal traffic, false negatives, the most critical type of failure in a detection system.

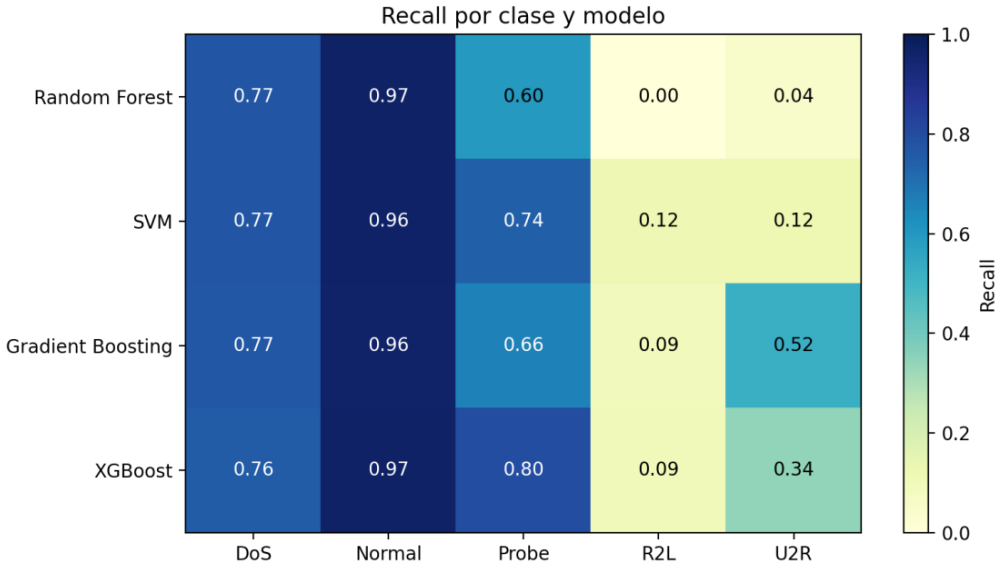

Figure 2 Recall per class and model (heat map on KDDTest+).

Conclusions

The project meets its objective: NetRadar integrates, in a reproducible and usable tool, the entire path from raw data to multiclass intrusion classification, with results consistent with the state of the art. Beyond the figures, the work offers a valuable critical insight: a high overall accuracy can hide a system that misses precisely the most dangerous attacks, so an honest evaluation of a detector requires looking at per-class behaviour rather than a single percentage. XGBoost emerges as the most suitable option for deployment, and the main lines of improvement, validation with more recent datasets, calibration of the decision threshold, and detection of unknown threats, are proposed as future work.

Índice de la memoria

Índice de figura

Índice de tablas

Capítulo 1. INTRODUCCIÓN

La defensa de las redes corporativas ha cambiado de naturaleza en los últimos años. Ni el volumen de tráfico ni la variedad de los ataques se parecen a los de hace una década, y las herramientas pensadas para un escenario más simple empiezan a quedarse cortas. Este capítulo presenta la motivación de NetRadar y delimita el problema concreto que aborda, dejando para los capítulos siguientes el desarrollo del estado del arte y la justificación cuantitativa.

1.1 MOTIVACIÓN DEL PROYECTO

La idea de NetRadar surge de una observación sencilla: la mayoría de las herramientas de detección de intrusiones al alcance de un equipo pequeño o de un estudiante se limitan a decidir si una conexión es normal o anómala, sin precisar qué tipo de ataque se está produciendo. Para un analista esa distinción importa, porque la respuesta ante un sondeo de puertos no es la misma que ante un intento de escalada de privilegios. Quien recibe una alerta binaria sigue teniendo que investigar a mano la naturaleza del incidente, justo el trabajo que una herramienta de apoyo debería ahorrarle.

A esa carencia se añade otra. Muchas soluciones potentes operan desde la línea de comandos o exigen una infraestructura considerable, lo que las aleja de quien se inicia en el análisis forense de red. El interés por este proyecto nace, por tanto, de reunir dos cosas que rara vez van juntas: la capacidad de un modelo de aprendizaje automático para clasificar el tipo concreto de ataque y una interfaz web que un analista pueda usar sin montar un servidor.

NetRadar responde a ese doble objetivo. Aplica varios modelos de clasificación multiclase sobre el tráfico de red para asignarlo a una de cinco familias (tráfico normal y cuatro tipos de ataque) y los integra en una aplicación web que guarda el historial de cada análisis. El contexto de amenazas que justifica numéricamente esta necesidad se desarrolla en el apartado 3.1; aquí basta con señalar que el crecimiento sostenido del volumen y la

sofisticación de los ataques ha vuelto inviable el análisis manual y ha desplazado el foco hacia técnicas capaces de generalizar más allá de las firmas conocidas.

Más allá de su dimensión técnica, el proyecto se enmarca en los Objetivos de Desarrollo Sostenible de la Agenda 2030 de Naciones Unidas. Proteger las comunicaciones es una condición para que funcionen las infraestructuras digitales (ODS 9), los servicios urbanos interconectados de las *Smart City* (ODS 11) y las propias instituciones (ODS 16). Poner capacidades de análisis avanzado al alcance de equipos con pocos recursos contribuye, de forma modesta, a esa resiliencia. La alineación detallada con cada objetivo se recoge en el Anexo I.

1.2 EL PROBLEMA ACTUAL DE LA CIBERSEGURIDAD

El problema que aborda este trabajo puede resumirse en dos tensiones que la seguridad de red arrastra desde hace años. La primera es de volumen: la cantidad de tráfico que circula por una red actual supera con holgura lo que un equipo humano puede inspeccionar registro a registro, de modo que el análisis manual ha dejado de ser una opción realista. La segunda es de método: la defensa clásica, construida sobre firmas y reglas estáticas, reconoce bien lo que ya ha visto, pero queda ciega ante ataques nuevos o ligeramente modificados, que no encajan en ningún patrón previo.

Estas dos limitaciones explican el giro de la industria y la academia hacia el aprendizaje automático, capaz de inferir el carácter de una conexión a partir de su comportamiento y no solo de una firma exacta. La evolución concreta de las herramientas de detección, desde los primeros cortafuegos hasta las plataformas actuales, se examina en el Capítulo 2, y los datos que cuantifican la magnitud del problema (frecuencia de ataques y coste de las brechas) se presentan en el apartado 3.1. NetRadar se sitúa en ese punto de partida: asume las carencias descritas y propone una respuesta práctica, centrada en clasificar el tipo de ataque y en hacerlo desde una interfaz accesible.

Capítulo 2. ESTADO DE LA CUESTIÓN

Para entender por qué una plataforma como NetRadar resulta necesaria, conviene analizar el contexto actual de la ciberseguridad. Durante las últimas décadas, el crecimiento acelerado de internet ha significado un aumento proporcional en el volumen del tráfico de red, de modo que analizar estos datos a mano se ha vuelto casi imposible para los equipos de ciberseguridad. Al mismo tiempo, los ciberataques son cada vez más sofisticados, sigilosos y automatizados.

En este capítulo, se recorre la evolución de las herramientas de monitorización de red, desde los enfoques tradicionales basados en reglas estáticas hasta la irrupción de la Inteligencia Artificial, evaluando qué soluciones existen en el mercado y qué vacíos pretende cubrir este proyecto.

2.1 EVOLUCIÓN DE LOS SISTEMAS DE DETECCIÓN DE INTRUSIONES

Históricamente, la defensa de las redes corporativas se basaba en un modelo perimetral. Sistemas como los *Firewalls* y los primeros Sistemas de Detección de Intrusiones funcionaban de manera aislada, analizando el tráfico que entraba y salía de la red en busca de patrones conocidos de *malware*. Aunque estos sistemas tradicionales son altamente efectivos contra amenazas conocidas, presentan una limitación estructural: son incapaces de detectar ataques nuevos o desconocidos para los que todavía no existe una firma registrada.

Para cubrir esta carencia, la industria desarrolló los sistemas SIEM *(Security Information and Event Management)*. Soluciones como Splunk, IBM QRadar o Microsoft Sentinel fueron diseñadas para centralizar y correlacionar los registros *(logs)* de todos los dispositivos de una red. Sin embargo, a pesar de su capacidad analítica, los SIEM tradicionales dependen de reglas estáticas preconfiguradas y requieren equipos de analistas especializados para interpretar las alertas. El resultado habitual es que los equipos quedan enterrados bajo miles de notificaciones diarias, lo que dificulta identificar los incidentes realmente críticos.

2.2 SOLUCIONES COMERCIALES Y REFERENTES

Con la migración de las empresas hacia entornos Cloud y el trabajo en remoto, el perímetro tradicional de la red ha desaparecido. Los empleados ya no trabajan desde una oficina física protegida por un *Firewall,* sino accediendo a aplicaciones desde cualquier lugar del mundo. Es en este contexto donde surgen arquitecturas modernas como SASE *(Secure Access Service Edge)* y plataformas CASB *(Cloud Access Security Broker)*.

Plataformas como Netskope [15] han redefinido la forma de visualizar y proteger el tráfico de red. Netskope no solo monitoriza, si no que ofrece paneles de control intuitivos y categorización de riesgos en tiempo real, permitiendo a los administradores tomar decisiones rápidas sin necesidad de descifrar *logs* en texto plano. Su filosofía de diseño, articulada en torno a tarjetas de información y navegación simplificada, ha sido una referencia directa para el desarrollo de la interfaz y la usabilidad de NetRadar.

2.3 APLICACIÓN DE MACHINE LEARNING EN CIBERSEGURIDAD

Si los sistemas tradicionales fallan ante amenazas desconocidas y el volumen de tráfico actual supera la capacidad humana de análisis, la respuesta natural de la industria y la academia ha sido recurrir a la Inteligencia Artificial. La aplicación de técnicas de Machine Learning en la ciberseguridad representa un cambio de paradigma: pasar de detección por firmas a la detección por anomalías de comportamiento [4],[14].

En lugar de decirle al sistema como es un ataque específico, se entrena a un modelo matemático con millones de registros de tráfico histórico para que aprenda cómo es el comportamiento normal y cómo varían matemáticamente los distintos tipos de intrusiones (ataques de Denegación de Servicio, escaneos de puertos, etc.). Modelos como Random Forest, *Support Vector Machines* (SVM) o *Gradient Boosting* son capaces de encontrar relaciones no lineales entre decenas de características de una conexión en milisegundos.

Este es el núcleo de NetRadar: poner estos algoritmos al alcance de cualquier equipo mediante una aplicación web accesible que, más allá de distinguir tráfico legítimo de

malicioso, clasifica cada conexión en su familia de ataque, con independencia de si esa variante concreta había sido catalogada antes por un antivirus comercial.

2.4 BASES DE DATOS ESTÁNDAR DE TRÁFICO DE RED

Para que la investigación en sistemas IDS basados en Machine Learning sea válida, reproducible y comparable a nivel académico, es necesario usar conjuntos de datos estandarizados. Durante años, la base de datos KDD Cup 99 [1] fue la referencia del campo. Sin embargo, hoy debido a sus carencias estructurales y a la gran cantidad de registros redundantes que sesgaban los modelos, los investigadores propusieron una versión depurada: NSL-KDD, que es el conjunto de datos estándar en investigación de detección de intrusiones. Contiene decenas de miles de registros de conexiones de red caracterizadas por 41 variables predictoras (duración, protocolo, servicios, bytes, intentos de login fallidos, etc.) y etiquetadas en categorías específicas de ataque. El uso de esa base de datos en NetRadar permite que el rendimiento de los modelos desarrollados pueda contrastarse con el estado del arte de la disciplina.

Capítulo 3. DEFINICIÓN DEL TRABAJO

3.1 JUSTIFICACIÓN

En la actualidad, la transformación digital y la hiperconectividad han derivado en un crecimiento notable en el volumen y la complejidad del tráfico de red. Esta evolución, aunque impulsa nuevos servicios y arquitecturas empresariales, ha ampliado considerablemente la superficie de ataque y ha favorecido la aparición de amenazas cibernéticas más frecuentes y difíciles de contener.

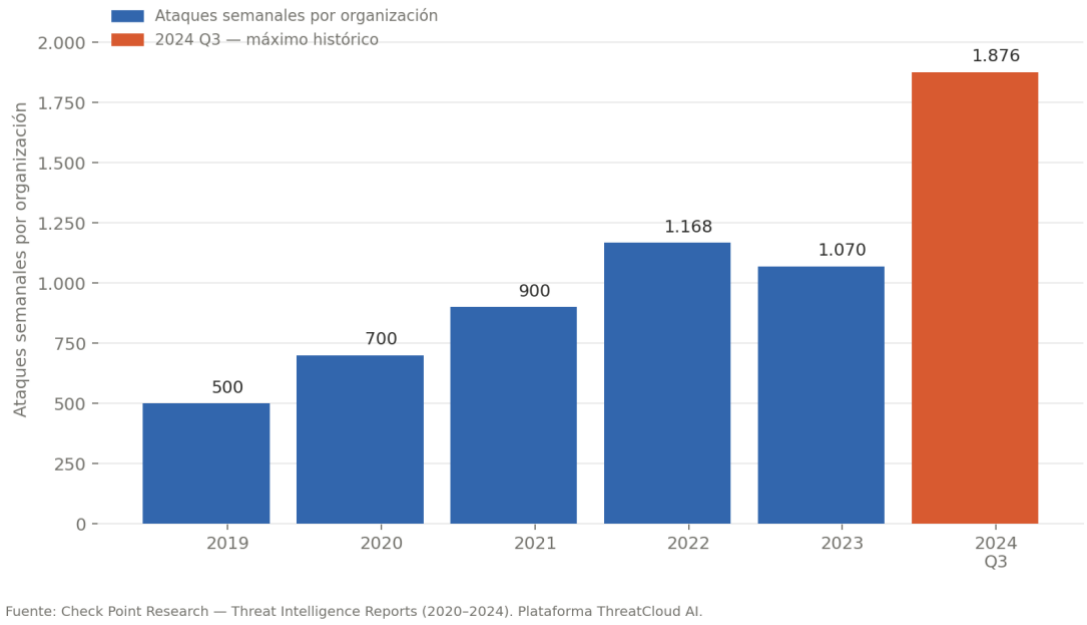

Figura 3.1 Evolución de ciberataques semanales

Esta tendencia se refleja en la evolución reciente del panorama de amenazas globales. Según los datos de Check Point Research, ilustrados en la Figura 3.1, la frecuencia de los ciberataques ha crecido de forma sostenida en los últimos años. En 2022, los ataques semanales por organización aumentaron un 38% respecto al año anterior, situándose en una media de 1.168 incidentes. El tercer trimestre de 2024 superó ese registro con una media de

1.876 ataques semanales por empresa, un 75% más que en el mismo periodo del año anterior [5].

Las consecuencias de este volumen de ataques no son solo técnicas. El informe Cost of a Data Breach 2024 de IBM Security cifra el coste medio global de una brecha de datos en 4,88 millones de dólares, un 10% más que en 2023 [7]. El Fondo Monetario Internacional, por su parte, advierte en su Global Financial Stability Report que la frecuencia de estos incidentes se ha duplicado desde la pandemia de COVID-19 [8].

En este contexto, la seguridad perimetral tradicional, construida en torno a *firewalls* clásicos y sistemas de detección de intrusos con reglas estáticas, tiene una gran carencia: su eficacia depende de amenazas previamente catalogadas. Frente a ataques de día cero o variantes de malware conocido, estos mecanismos son insuficientes, dado que no disponen de la flexibilidad necesaria para identificar comportamientos que no correspondan a un patrón predefinido. A esta restricción se añade el problema del volumen: la cantidad de paquetes que circulan por las redes actuales hace que la inspección y el análisis forense manual sean tareas difícilmente abordables para los equipos de ciberseguridad.

Para superar estas carencias, los modelos basados en Inteligencia Artificial y *Machine Learning* ofrecen una alternativa. Permiten procesar grandes volúmenes de datos, caracterizar el comportamiento habitual del tráfico y detectar cosas que pasarían inadvertidas en un sistema basado en reglas fijas. Aun así, muchas soluciones actuales carecen de interfaces accesibles o se limitan a una clasificación binaria entre tráfico normal y anómalo, lo que limita la utilidad para el analista, que necesita conocer la naturaleza concreta del ataque para actuar con criterio.

En este escenario se enmarca el desarrollo de NetRadar. El proyecto da respuesta a la necesidad de una solución práctica para el análisis de tráfico de red, mediante modelos de clasificación multiclase capaces no solo de detectar intrusiones, sino de categorizarlas en familias específicas (DoS, Probe, R2L o U2R). La combinación de algoritmos de IA con una arquitectura de software robusta dota a los analistas de una herramienta web que agiliza el

análisis forense, acorta los tiempos de respuesta y se ajusta a la realidad de un panorama de amenazas en crecimiento continuo.

3.2 OBJETIVOS

Una vez descrito el ámbito actual de la ciberseguridad y la necesidad de herramientas de análisis avanzadas, este apartado recoge los propósitos que guían el desarrollo del proyecto. Para estructurar el alcance del trabajo con claridad, se distingue entre un objetivo general, que refleja el fin principal del sistema, y un conjunto de objetivos específicos que concretan los hitos técnicos necesarios para alcanzarlo.

3.2.1 OBJETIVO GENERAL

El objetivo principal de este Trabajo de Fin de Grado es diseñar y desarrollar NetRadar, una aplicación web orientada al análisis forense de red y la detección de intrusiones. La plataforma debe ser capaz de procesar datos de tráfico de red y aplicar modelos de *Machine Learning* para una clasificación multiclase, es decir, no solo separar tráfico normal del malicioso, sino identificar la categoría concreta del ataque. El sistema se desarrolla sobre una arquitectura cliente-servidor que garantice un análisis rápido, el registro persistente del historial forense y una interfaz accesible para los analistas de ciberseguridad que trabajen con ella.

3.2.2 OBJETIVOS ESPECÍFICOS

Para cumplir el objetivo general, el proyecto se descompone en los siguientes objetivos de carácter técnico, metodológico y de desarrollo:

- **Preprocesamiento y adecuación de datos:** Analizar, limpiar y transformar la base de datos estándar NSL-KDD, aplicando codificación de variables categóricas mediante Label Encoding y normalización matemática de las variables numéricas. El resultado debe ser un espacio de características coherente y adecuado para las fases de entrenamiento y prueba.

- **Gestión del desbalanceo de clases:** Incorporar estrategias de ponderación o técnicas algorítmicas durante el entrenamiento para corregir el desequilibrio natural presente en las bases de datos de tráfico de red, con especial atención a la detección de ataques minoritarios como R2L o U2R, que tienden a quedar infrarrepresentados.

- **Entrenamiento y evaluación de algoritmos de Machine Learning:** Entrenar, ajustar y comparar múltiples modelos de clasificación: Random Forest, Máquinas de Vectores de Soporte (SVM), Gradient Boosting y XGBoost. El proceso incluirá tanto la optimización de hiperparámetros como la comparativa sistemática de resultados entre modelos.

- **Clasificación multiclase de amenazas:** Implementar la lógica predictiva necesaria para etiquetar cada conexión de red analizada dentro de una de las cinco categorías definidas: tráfico Normal, Ataques de Denegación de Servicio (DoS), Pruebas y escaneos (Probe), Accesos remotos no autorizados (R2L) y Escalada de privilegios locales (U2R).

- **Desarrollo del backend y exposición de API REST:** Construir la lógica de servidor utilizando FastAPI (Python). Este componente es responsable de cargar los modelos predictivos entrenados y exponer los endpoints que permiten la comunicación asíncrona con la interfaz de usuario.

- **Implementación de la capa de persistencia:** Diseñar e integrar una base de datos relacional con SQLite para registrar de forma persistente el historial de análisis realizados, facilitando la trazabilidad y la consulta de auditorías forenses pasadas.

- **Construcción del frontend:** Desarrollar una aplicación web de página única (SPA) que permita al usuario cargar conjuntos de datos, seleccionar el modelo de IA, visualizar métricas de rendimiento y consultar los resultados del análisis con el nivel de detalle que cada caso requiera.

- **Evaluación del rendimiento del sistema:** Medir la eficacia de la solución utilizando métricas estándar de ciencia de datos: Precisión (Accuracy), F1-Score, Recall y análisis de las matrices de confusión para cada uno de los modelos entrenados.

3.3 ALCANCE Y LIMITACIONES

Este apartado delimita qué cubre NetRadar y qué queda deliberadamente fuera, tanto por decisiones de diseño como por las restricciones asumidas durante el desarrollo.

3.3.1 ALCANCE DEL PROYECTO

NetRadar es un primer prototipo funcional cuyo propósito es comprobar, en la práctica, que el aprendizaje automático sirve no solo para separar tráfico legítimo de malicioso, sino para clasificar el tipo concreto de ataque. El alcance de la plataforma comprende las siguientes capacidades:

Ingesta y procesamiento de datos bajo demanda: la aplicación permite cargar archivos de tráfico en formato CSV con la estructura de características de NSL-KDD. Sobre ellos aplica de forma automática el preprocesamiento que los modelos necesitan en su entrada: la codificación de las variables categóricas y el escalado.

Análisis y clasificación multiclase: el núcleo del sistema ejecuta los modelos previamente entrenados que el analista seleccione, entre Random Forest, máquinas de vectores de soporte, Gradient Boosting y XGBoost, y clasifica cada registro en una de cinco categorías: tráfico Normal, DoS, Probe, R2L y U2R.

Persistencia y auditoría: el sistema almacena en una base de datos relacional los metadatos de cada análisis (fecha, modelos empleados y archivo analizado) junto con sus resultados, de modo que el analista puede consultar el histórico de análisis anteriores. Esta función es la que sostiene su uso como herramienta de soporte forense.

Interfaz de usuario interactiva: incluye una interfaz web accesible desde el navegador, pensada para que el analista no tenga que manejar la complejidad de los modelos. Los resultados se presentan mediante tablas y métricas legibles.

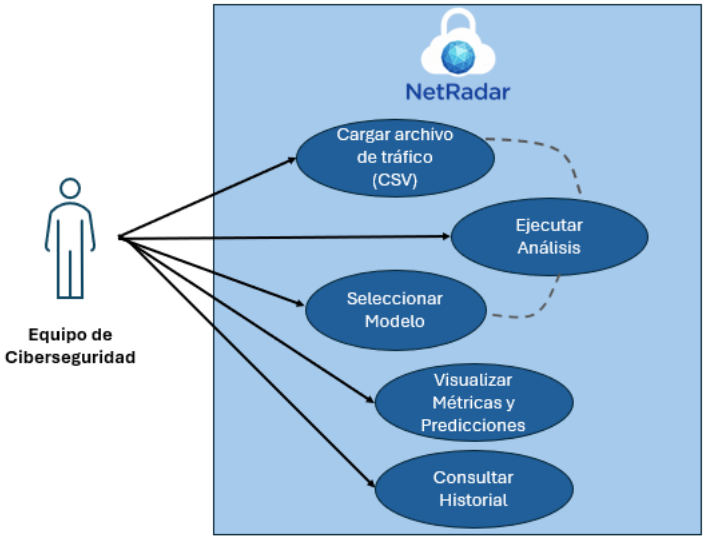

Figura 3.2 Diagrama de Casos de Uso

3.3.2 LIMITACIONES

Delimitar lo que el sistema no hace es tan importante como describir lo que sí hace. Las limitaciones asumidas son las siguientes:

Análisis diferido, no en tiempo real: NetRadar trabaja sobre capturas ya exportadas a formato tabular. No captura paquetes en vivo desde una interfaz de red física, con herramientas tipo pcap o sniffers, sino que analiza ficheros generados con anterioridad.

Detección, no prevención: el sistema funciona como IDS (detección), no como IPS (prevención). No actúa sobre la red: no reconfigura *firewalls*, no bloquea direcciones IP ni interrumpe conexiones maliciosas en curso. Se limita a identificar el ataque y registrarlo.

Dependencia del conjunto de datos: la capacidad de generalización de los modelos depende directamente de los datos con los que se entrenan, en este caso NSL-KDD. Aunque es un conjunto de referencia en la investigación, no recoge los ataques de día cero aparecidos

tras su recopilación, de modo que ese tipo de amenazas queda fuera de lo que el sistema puede reconocer.

Datos de origen público, no corporativo: los modelos se han entrenado y evaluado sobre datos públicos. No se ha dispuesto de tráfico real de una organización, ya que ese tipo de información no suele estar disponible de forma abierta por motivos de privacidad y confidencialidad. Todos los datos utilizados proceden de conjuntos publicados en internet, principalmente NSL-KDD. Esto condiciona la validez externa de los resultados: describen el comportamiento del sistema sobre un escenario de referencia, no necesariamente sobre el tráfico concreto de un entorno de producción real.

Capa de persistencia: esta versión emplea SQLite por su ligereza, su integración con Python y la facilidad de despliegue. Es suficiente para un prototipo, pero un entorno con varios analistas concurrentes y un volumen de registros mucho mayor exigiría migrar a un motor más robusto, como PostgreSQL o MySQL. Esa migración queda fuera de esta primera iteración.

3.4 METODOLOGÍA

El desarrollo de NetRadar siguió una metodología iterativa e incremental, organizada en torno al ciclo de vida habitual de un proyecto de ciencia de datos: comprensión del problema, obtención y preparación de los datos, modelado, evaluación e integración en una aplicación. No fue un proceso estrictamente lineal. El modelado, en particular, avanzó por iteraciones sucesivas en las que cada evaluación de resultados realimentaba decisiones anteriores, desde el tratamiento del desbalanceo hasta la elección de hiperparámetros. La construcción del software se abordó de forma modular, separando el núcleo analítico de la capa web para poder probar cada parte por separado. Un criterio guió todo el trabajo: mantener la coherencia entre lo que afirma esta memoria y lo que hace realmente el código, verificando cada decisión técnica sobre la implementación en ejecución. Los apartados siguientes describen cómo se abordó cada actividad.

3.4.1 INVESTIGACIÓN Y SELECCIÓN DEL CONJUNTO DE DATOS

El trabajo requirió de una gran revisión bibliográfica de artículos científicos y literatura especializada en ciberseguridad e Inteligencia Artificial. El objetivo de esta inmersión teórica fue comprender el estado del arte y, de forma crítica, localizar una fuente de datos que fuera lo suficientemente completa, robusta y equilibrada para permitir no solo la detección de anomalías, sino la clasificación detallada de diferentes tipos de ataques. Tras evaluar distintas opciones, se seleccionó la base de datos estándar NSL-KDD como el cimiento analítico del proyecto.

3.4.2 PREPROCESAMIENTO DE DATOS

Todo proceso de análisis de datos requiere de una limpieza y transformación de los registros en bruto. Dado que los algoritmos matemáticos no procesan texto, fue necesario aplicar técnicas de codificación (Label Encoding) a las variables categóricas, así como normalizar las características numéricas para que compartieran una escala común. En esta fase se invirtió un gran esfuerzo en el tratamiento y adecuación de los datos para garantizar que fueran plenamente utilizables y eficientes antes de alimentar a la Inteligencia Artificial.

3.4.3 ENTRENAMIENTO DE LOS MODELOS

El trabajo exigía una fase de experimentación y generación de los modelos de Machine Learning. Se implementaron y probaron diversos algoritmos, y se seleccionaron los cuatro más relevantes. El proceso implicó un entrenamiento iterativo de estos modelos, ajustando parámetros y aplicando técnicas de balanceo de clases, evaluando con un protocolo exigente que da cifras realistas.

3.4.4 DESARROLLO DE LA INTERFAZ E INTEGRACIÓN DEL SISTEMA

Otro paso importante era el desarrollo del software de NetRadar. Se diseñó y programó el Frontend de la aplicación, priorizando la creación de una interfaz de usuario visual e intuitiva. Asimismo, se llevó a cabo la fase más crítica de ensamblaje: se encapsularon los

modelos ya entrenados y se conectaron, mediante una API y un servidor Backend, tanto con la base de datos para el historial como con la interfaz gráfica web.

3.5 PLANIFICACIÓN

Para que se desarrollen correctamente los objetivos de un proyecto de ingeniería de software e inteligencia artificial, se requiere una planificación estructurada que garantice la optimización del tiempo y de los recursos disponibles. Dado el carácter secuencial e iterativo de NetRadar, el desarrollo se ha dividido en una serie de fases lógicas que abarcan desde el inicio de la investigación hasta la entrega del producto final y su documentación.

3.5.1 FASES DEL PROYECTO

El ciclo de vida del proyecto se ha estructurado en cinco fases principales, como se ilustra en la Figura 3.3.

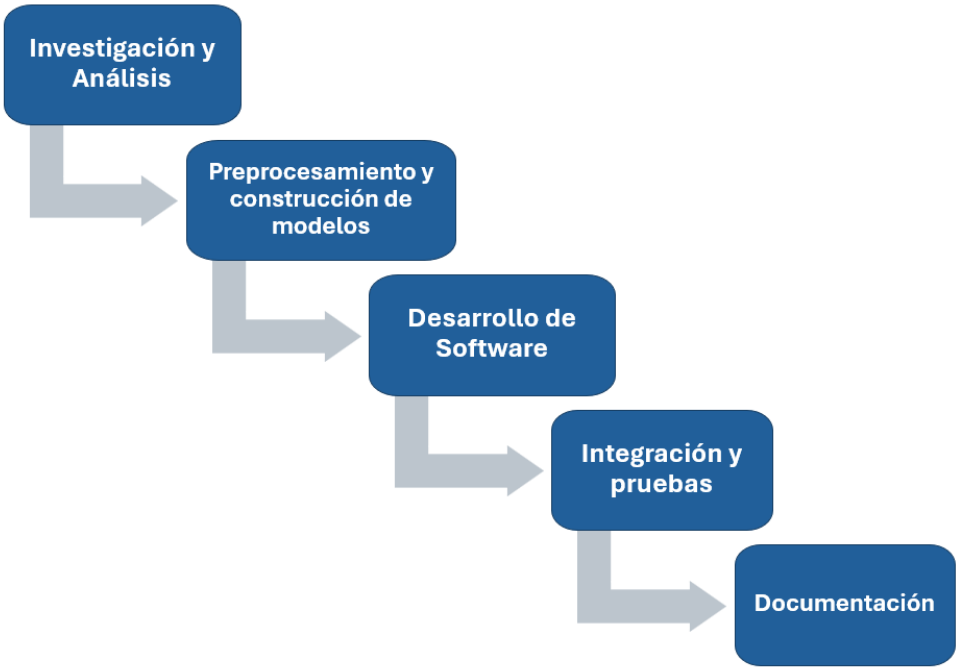

Figura 3.3 Etapas de desarrollo

El reparto exacto de cada fase en el tiempo, junto con los tramos que se ejecutaron en paralelo, se representa en el diagrama de Gantt del apartado siguiente.

3.5.2 CALENDARIO DE EJECUCIÓN

En esta gráfica se puede apreciar cómo ciertas tareas analíticas requirieron una ejecución estrictamente secuencial, mientras que otras, como el desarrollo del Frontend y del Backend, o la propia redacción de la memoria, se solaparon en el tiempo para optimizar los plazos de entrega.

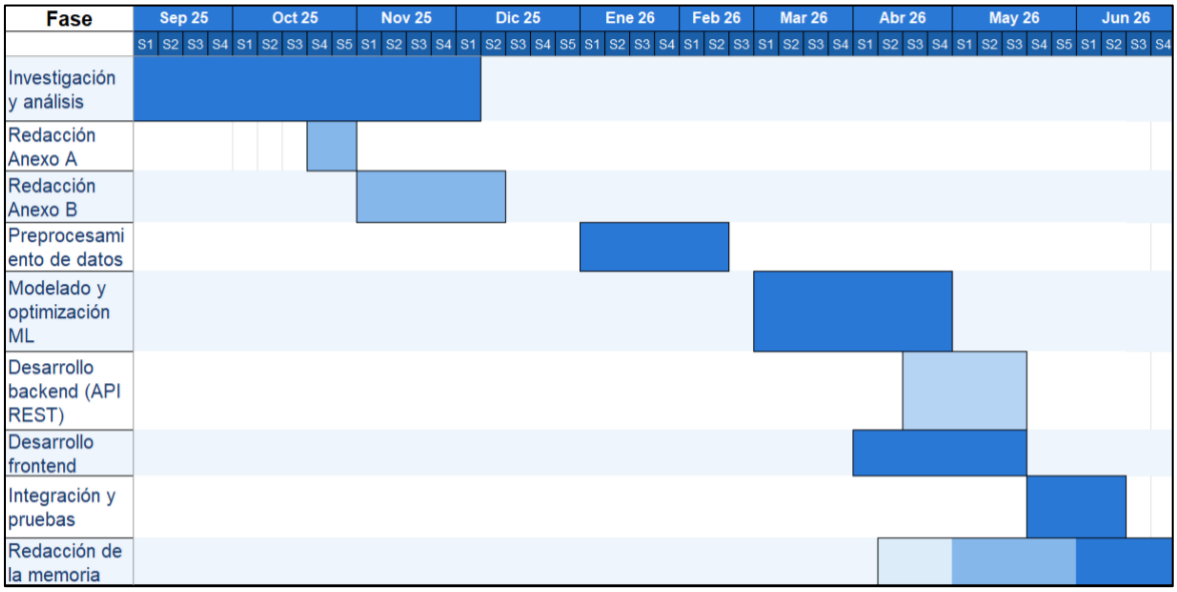

Figura 3.4 Diagrama de Gantt (Sep. 2025 - Jun 2026)

3.6 ESTIMACIÓN ECONÓMICA

En este apartado se presupuesta el coste equivalente que habría supuesto el desarrollo comercial de NetRadar en un entorno empresarial real. Para ello, se han contabilizado los costes derivados de las horas de trabajo invertidas, el uso de infraestructura física y el coste de las licencias de software.

3.6.1 COSTES DE PERSONAL

El principal coste del proyecto radica en el desarrollo humano, tanto en el ámbito de la ciencia de datos como en la ingeniería de software. Para esta estimación, se asume el rol de un Ingeniero de Software Junior o de un *Data Scientist* encargado del desarrollo completo. Considerando una inversión aproximada de 300 horas de trabajo desde la fase de investigación hasta la entrega, y aplicando un coste estándar de mercado de 25 €/hora para este perfil en España, el coste de personal se detalla en la Tabla 3-1.

Tabla 3-1 Coste de personal

Perfil Profesional	Horas Invertidas	Coste por hora (€/h)	Coste total (€)
Ingeniero Junior de Software o Datos	300h	25€	7500€
TOTAL			**7500€**

3.6.2 COSTES DE HARDWARE Y SOFTWARE

En cuanto a los recursos materiales, NetRadar se ha desarrollado utilizando un ordenador portátil estándar, sin requerir aceleración por hardware externa (como granjas de GPUs) para el entrenamiento de los modelos. Asumiendo un coste de adquisición del equipo de 1.200 € y una vida útil contable de 3 años (36 meses), se ha calculado la amortización correspondiente a los 5 meses de duración del proyecto, resultando en un coste imputable de 166,67 €.

Por otro lado, el apartado de software constituye una de las mayores ventajas competitivas de la plataforma. Por decisión de diseño y viabilidad técnica, todo el stack tecnológico utilizado en NetRadar está fundamentado en lenguajes y librerías de código abierto (Open Source). Python, FastAPI, Scikit-Learn, SQLite, Chart.js y el editor Visual Studio Code son

de uso libre y gratuito, lo que implica que el coste asociado a licencias de software es de 0,00 €.

	Coste de adquisición	Amortización (5 meses)	Coste Imputable (€)
Ordenador Portátil	1200€	(1.200 € / 36) × 5	166,67 €
Software	0,00 €	N/A	0,00 €
TOTAL			**166,67 €**

3.6.3 COSTE TOTAL

A partir de los cálculos anteriores, se elabora el presupuesto final consolidado. A la suma de los costes directos de personal y de infraestructura, se le ha añadido un margen de del 10% para cubrir imprevistos durante el desarrollo. Finalmente, se aplica el Impuesto sobre el Valor Añadido (IVA) del 21% vigente en España.

De este modo, se concluye que el desarrollo de NetRadar requiere un presupuesto total estimado de 10.204,34 €, lo cual demuestra que es un proyecto altamente viable económicamente gracias a la nula dependencia de licencias de software de pago.

Capítulo 4. DESCRIPCIÓN DE LAS TECNOLOGÍAS

El objetivo de este capítulo es detallar y justificar las tecnologías seleccionadas para el desarrollo y despliegue de la plataforma NetRadar. La elección de los lenguajes de programación, *frameworks* y librerías que conforman este entorno no ha sido arbitraria, sino que resuelve los requisitos de rendimiento, eficacia y rigor analítico necesarios para el proyecto. A lo largo de las siguientes secciones se explican exhaustivamente los cuatro pilares principales: la interfaz de usuario (*frontend*), la lógica del servidor (*backend),* el motor de inteligencia artificial *(Machine Learning)* y la capa de persistencia de datos.

4.1 LENGUAJES Y ENTORNO WEB

Para el desarrollo de la interfaz de usuario final, la plataforma es minimalista y se centra en el rendimiento, la accesibilidad y la fluidez. En esta sección se detalla los lenguajes que componen el entorno web, los cuales han sido seleccionados e implementados siguiendo el paradigma de *Single Page Application* (SPA). Para proporcionar a los equipos de ciberseguridad una herramienta ágil e interactiva, se ha tomado la decisión de diseño de prescindir de *frameworks* de alto nivel o librerías externas pesadas.

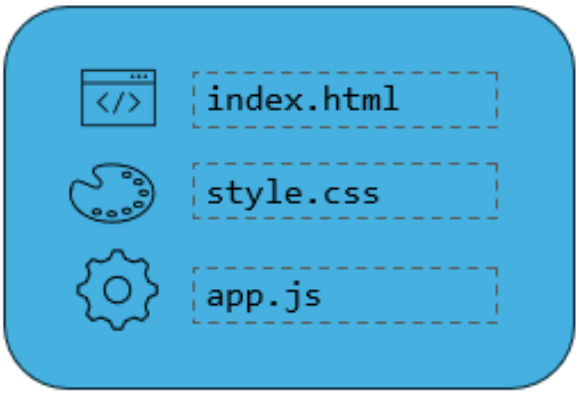

Figura 4.1 Estructura del Frontend

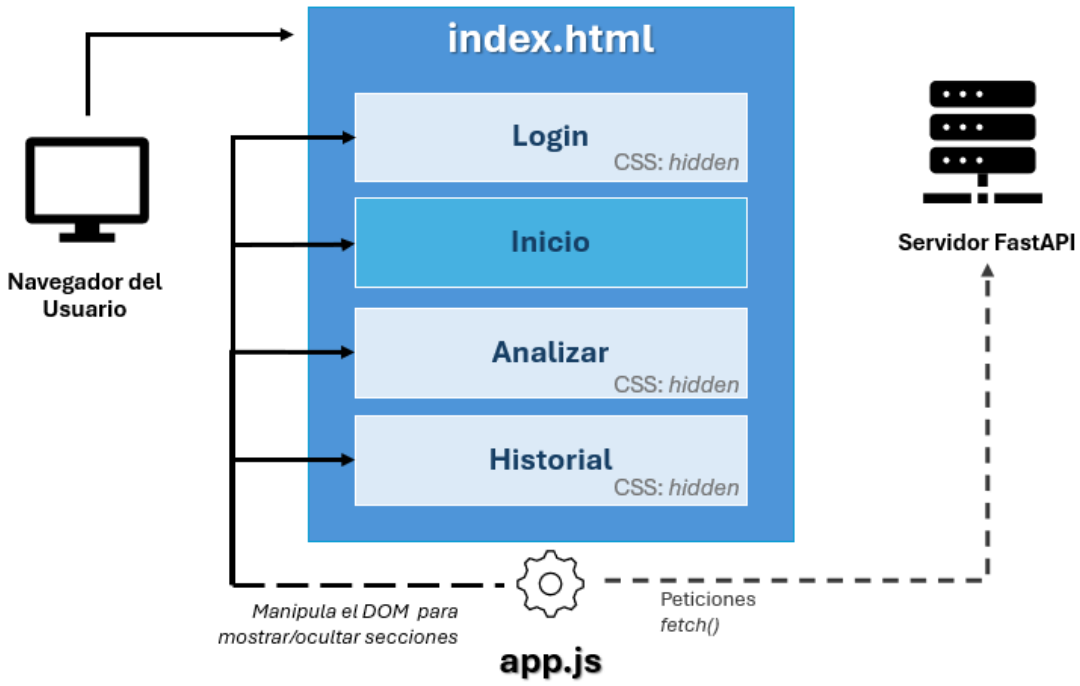

Figura 4.2 Flujo Single Page Application

4.1.1 HTML

Para el desarrollo de la plataforma, se ha utilizado la quinta versión de HTML. Este define la estructura del frontend de NetRadar. Toda la interfaz está en un único archivo contenedor, lo que simplifica el árbol de dependencias y localiza cualquier modificación estructural en un único punto. Como se ha mencionado previamente, no se ha recurrido a librerías externas de componentes: la estructura es HTML estándar.

4.1.2 CSS

Para la gestión del estilo visual de la plataforma, se ha utilizado CSS en su tercera versión. Lejos de ser este su único trabajo, también cumple una función arquitectónica. Mediante diseño responsivo, la interfaz se adapta a distintos tamaños de pantalla sin versiones alternativas del código. Dentro de la lógica de navegación de NetRadar, CSS3 controla la visibilidad de los contenedores: las secciones inactivas se ocultan y solo la vista solicitada por el cliente permanece visible, sin necesidad de recargar la página, lo cual se puede interpretar en la figura 4.2.

4.1.3 JAVASCRIPT

JavaScript es el lenguaje de programación de alto nivel, interpretado y orientado a objetos. Gracias a este, se gestiona el comportamiento, la interactividad y el dinamismo de la interfaz NetRadar. En concreto se ha utilizado Vanilla JavaScript, es decir, el lenguaje en estado puro, sin *frameworks*. Al interactuar directamente con el DOM *(Document Object Model)* sin intermediarios, la aplicación no carga ninguna librería en el arranque, lo que acorta el tiempo hasta la primera interacción y elimina dependencias de terceros que podrían introducir incompatibilidades o vulnerabilidades. El script principal decide qué vista renderizar, gestiona los eventos del usuario y establece la comunicación asíncrona con el servidor.

4.2 BACKEND Y SERVIDOR

Una vez definidas las tecnologías de la capa de presentación interactiva, hay que establecer las tecnologías que actúan como el núcleo de Netradar. El backend de la plataforma es el responsable de orquestar la lógica de negocio, procesar la información de manera segura y fundamentalmente, servir como puente de comunicación eficiente entre las peticiones del cliente web y los modelos predictivos. Dado que el sistema debe analizar archivos que contienen miles de registros de tráfico de red, la selección de estas herramientas se ha regido por criterios estrictos de alto rendimiento, capacidad de ejecución y compatibilidad con la ciencia de datos.

4.2.1 PYTHON

Python es el lenguaje principal del backend. Es el estándar en inteligencia artificial y ciencia de datos, con un ecosistema de librerías que cubre desde la manipulación de datos tabulares hasta el entrenamiento y serialización de modelos. Usar Python en el servidor permite que la misma capa que gestiona las peticiones HTTP ejecute directamente los algoritmos de clasificación, sin puentes entre lenguajes ni procesos externos.

4.2.2 FASTAPI

FastAPI es el entorno de trabajo utilizado para construir la Interfaz de Programación de Aplicaciones (API) del servidor. En términos prácticos, actúa como el puente de comunicación entre la interfaz de usuario que maneja el analista y el motor de Inteligencia Artificial desarrollado en Python. Dado que el frontend no puede ejecutar los modelos predictivos solo, necesita enviar los datos a un servidor que realice el trabajo pesado y devuelva los resultados. FastAPI es la tecnología encargada de recibir, gestionar y responder estas peticiones.

Este entorno ha sido seleccionado en lugar de otros como Flask o Django, por su rendimiento y ejecución asíncrona. El análisis de redes implica manejar archivos que pueden contener miles de registros. FastAPI funciona de forma asíncrona, esto significa que, mientras el sistema procesa un escaneo pesado de la red, el servidor no se bloquea. Puede seguir atendiendo otras solicitudes sin colapsar, por tanto, los tiempos de respuesta son extremadamente rápidos.

Además, FastAPI valida de forma automática los cuerpos de las peticiones que se declaran como modelos de datos (mediante la biblioteca Pydantic) y comprueba los tipos de los parámetros de ruta y de consulta. Esta validación cubre los datos estructurados de la API, como las credenciales de autenticación. El contenido de los ficheros CSV que se suben para analizar, en cambio, no lo valida el *framework*: se procesa de forma específica dentro de la propia aplicación, como se detalla en el apartado 5.4.

4.2.3 UVICORN

Uvicorn es un servidor web del tipo ASGI (*Asynchronous Server Gateway Interface*) sobre el cual se ejecuta FastAPI. Para entender su función dentro de la arquitectura de la plataforma, es importante distinguir entre la capa lógica y la capa de red. Mientras que FastAPI indica cómo se deben procesar los datos, Uvicorn es el encargado de escuchar activamente la red, gestionar las conexiones HTTP asíncronas y recibir el tráfico entrante. Dado que los *frameworks* de Python no se comunican directamente con los protocolos de internet de bajo nivel, necesitan un intermediario.

4.3 *LIBRERÍAS DE MACHINE LEARNING*

La detección de anomalías en redes exige un sistema capaz de identificar patrones maliciosos o comportamientos irregulares dentro de grandes volúmenes de tráfico. Esto supone una tarea imposible mediante sistemas estáticos basados en reglas tradicionales. Por ello, Netradar, se basa en un sistema robusto y especializado en Machine Learning. En esta sección se detallarán las librerías implementadas para gestionar el ciclo completo desde la subida de datos hasta el despliegue de los modelos predictivos.

4.3.1 PANDAS

Pandas gestiona la lectura, limpieza y transformación de los archivos CSV que el analista sube a la plataforma. Su estructura principal, el DataFrame, permite aplicar operaciones vectorizadas sobre miles de filas sin bucles explícitos, lo que reduce el tiempo de preprocesamiento. En el pipeline de NetRadar, Pandas convierte los registros de tráfico en matrices numéricas que los modelos pueden procesar directamente.

4.3.2 SCIKIT-LEARN

Scikit-Learn implementa los algoritmos Random Forest y SVM, y proporciona las herramientas de preprocesamiento:

- StandardScaler para el escalado de variables numéricas,
- LabelEncoder para la codificación de etiquetas.

Ambas transformaciones son necesarias antes del entrenamiento, ya que los algoritmos asumen variables en rangos comparables y clases representadas como valores numéricos. [3]

4.3.3 XGBOOST

Se hace uso de XGBoost para el desarrollo de uno de los modelos. Esta herramienta implementa Gradient Boosting de forma optimizada. A diferencia del Random Forest, que construye sus árboles en paralelo e independientemente, XGBoost los construye de forma secuencial: cada árbol nuevo se entrena para corregir los errores del conjunto anterior. Es una herramienta muy competitiva en datos tabulares.

4.4 ALMACENAMIENTO DE DATOS

El último pilar tecnológico que hay que mencionar pertenece a la capa de persistencia. Para que la herramienta resulte verdaderamente útil, no es suficiente procesar la información en tiempo real. Se necesita un sistema capaz de guardar, organizar y recuperar de forma estructurada tanto los perfiles de los usuarios como el historial de los escaneos realizados. En este apartado se define y justifica el sistema de gestión de bases de datos relacional adoptado para asegurar la integridad de la información generada. Para la elección del almacenamiento se ha priorizado la ligereza y la falta de dependencias de servidor, frente a infraestructuras pesadas tradicionales.

4.4.1 SQLITE

Por todo ello, se ha seleccionado SQLite para la persistencia. No es necesario un proceso servidor separado: toda la base de datos habita en un único archivo binario que la aplicación abre directamente. NetRadar es así completamente autocontenido y puede desplegarse sin instalar ni configurar ningún motor de base de datos externo, lo que elimina una barrera de entrada habitual en entornos de análisis forense donde los equipos no siempre tienen infraestructura de servidor disponible.

Capítulo 5. SISTEMA/MODELO DESARROLLADO

Este capítulo constituye la base técnica y práctica del presente proyecto, cuyo objetivo primordial es la materialización de los conceptos teóricos comentados previamente en una solución de software funcional, escalable y eficiente: NetRadar. A lo largo de las siguientes secciones se detallará de manera exhaustiva el ciclo de vida completo del desarrollo del sistema de detección de anomalías en red. Este proceso abarca desde la concepción de la arquitectura global y el tratamiento de los datos de tráfico, hasta el entrenamiento de los algoritmos de Machine Learning y la implementación de la interfaz web para el usuario final.

La idea de NetRadar nace de la necesidad de unificar las capacidades analíticas de los modelos predictivos con una herramienta que sea accesible e intuitiva. Por ello, el desarrollo no se limita únicamente a la programación de un script de Inteligencia Artificial, sino que engloba la construcción de un producto de software integral. Esto requiere aplicar principios de ingeniería del software, diseño de bases de datos relacionales y despliegue de servicios web, garantizando así un rendimiento óptimo en la clasificación de paquetes de red en tiempo real o en diferido.

5.1 ARQUITECTURA GLOBAL DEL SISTEMA

Para garantizar que NetRadar responda a los requisitos de robustez, mantenibilidad y escalabilidad exigidos en los sistemas modernos de monitorización de redes, su arquitectura se ha diseñado basándose en un modelo de tres capas. Este modelo arquitectónico permite separar las partes lógica y física de las responsabilidades del sistema, permitiendo una separación entre la interfaz de usuario, la lógica de procesamiento y la gestión de la información.

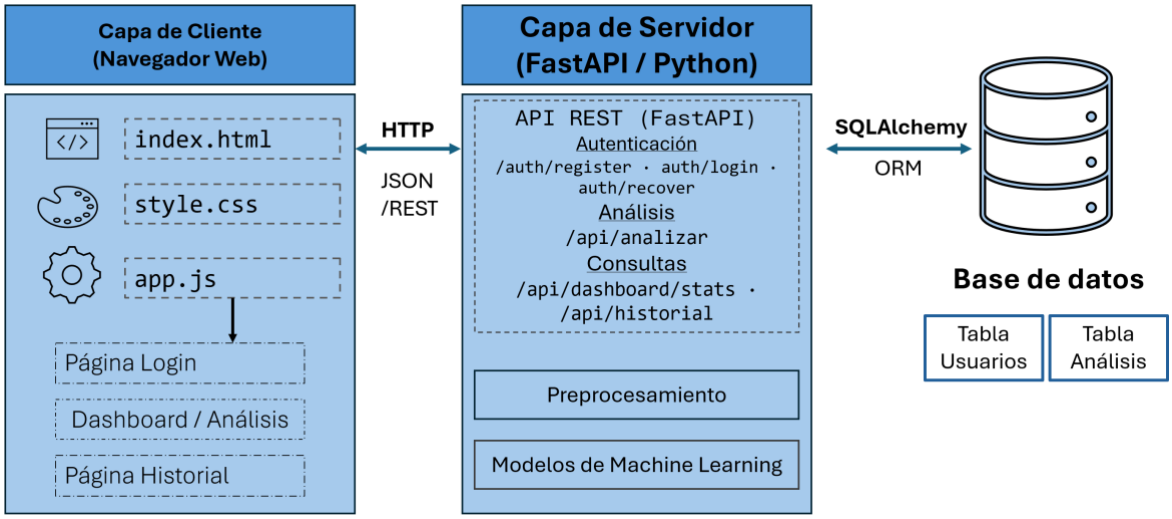

Figura 5.1 Esquema de funcionamiento de la plataforma NetRadar

5.1.1 CAPA DE PRESENTACIÓN

La primera capa es el punto de interacción con el usuario y se ha desarrollado como una SPA, como ya se había comentado previamente. Su principal cometido es ofrecer una experiencia de usuario fluida y reactiva sin necesidad de recargar la página completa durante la navegación. Esta capa se encarga de recopilar los ficheros de tráfico de red proporcionados por el cliente, enviar las peticiones al servidor a través de la red y, posteriormente, renderizar los resultados del análisis. Para la visualización de estas métricas, la capa de presentación utiliza librerías dinámicas que transforman los datos crudos devueltos por la predicción en gráficos interactivos fácilmente interpretables.

5.1.2 CAPA DE LÓGICA DE NEGOCIO Y PROCESAMIENTO

La arquitectura se centra alrededor de la capa de servidor, diseñada para soportar la carga computacional intensiva del proyecto. Expuesta a través de una API RESTful, esta capa actúa como el puente de comunicación orquestando todas las operaciones.

Cuando el servidor recibe un archivo de red desde el cliente, ejecuta una secuencia estricta: primero, realiza el parseo y preprocesamiento de los datos (normalización y codificación); segundo, inyecta este conjunto de datos en los modelos de predictivos previamente

entrenados; y tercero, consolida las predicciones multiclase (Normal, DoS, Probe, R2L, U2R). Al centralizar los algoritmos de Machine Learning en el backend, se garantiza que los modelos operen en un entorno controlado y seguro, independientemente de las capacidades del hardware del cliente.

5.1.3 CAPA DE PERSISTENCIA

Finalmente, la tercera capa asegura la trazabilidad, la persistencia a largo plazo y la seguridad de la información. Mediante un sistema de gestión de bases de datos relacionales basado en SQL, esta capa almacena de forma estructurada dos bloques fundamentales de información: el control de acceso, que es de tipo *stateless* básico: el identificador de usuario se transmite en cada petición, no se implementa gestión de sesiones con tokens y, el registro histórico de análisis forenses (endpoint /api/historial). Esta separación garantiza que las consultas de historiales previos se realicen con latencias mínimas y sin interferir con los recursos de procesamiento destinados a los análisis en curso.

La comunicación entre estas tres capas se realiza mediante protocolos estándar de la industria. El intercambio de información entre el Frontend y el Backend utiliza el protocolo HTTP estructurando las cargas útiles (payloads) en formato JSON, lo cual facilita la serialización y deserialización de las predicciones de red. Por su parte, la comunicación entre el servidor y la base de datos se efectúa mediante consultas SQL transaccionales que aseguran la integridad referencial de los datos del usuario.

5.2 SELECCIÓN Y PREPROCESAMIENTO DE DATOS

El rendimiento de un modelo de Machine Learning depende de la calidad de los datos con los que se entrena. En detección de intrusiones esa dependencia es especialmente marcada, porque capturar tráfico de red real plantea problemas de privacidad, rara vez viene etiquetado y genera volúmenes difíciles de manejar. Por ese motivo, NetRadar se entrena y evalúa sobre un conjunto de datos estandarizado y etiquetado, que permite medir la eficacia del sistema sobre una referencia reconocida y reproducible.

5.2.1 DESCRIPCIÓN DE LA BASE DE DATOS (NSL-KDD)

El conjunto elegido para entrenar y evaluar los modelos de NetRadar es NSL-KDD, una versión depurada del KDD Cup 1999, que fue durante años la referencia habitual en la investigación sobre sistemas de detección de intrusiones (IDS). La elección de NSL-KDD, del año 2007, frente al original responde a un defecto concreto de este último: la presencia de registros duplicados en los conjuntos de entrenamiento y prueba. Esa redundancia sesgaba el aprendizaje hacia las clases más frecuentes, los ataques masivos, y dificultaba la detección de las intrusiones poco representadas, que son justamente las que más interesa identificar. Al eliminar los duplicados, NSL-KDD reparte mejor el peso entre clases y ofrece una evaluación más fiable del modelo.

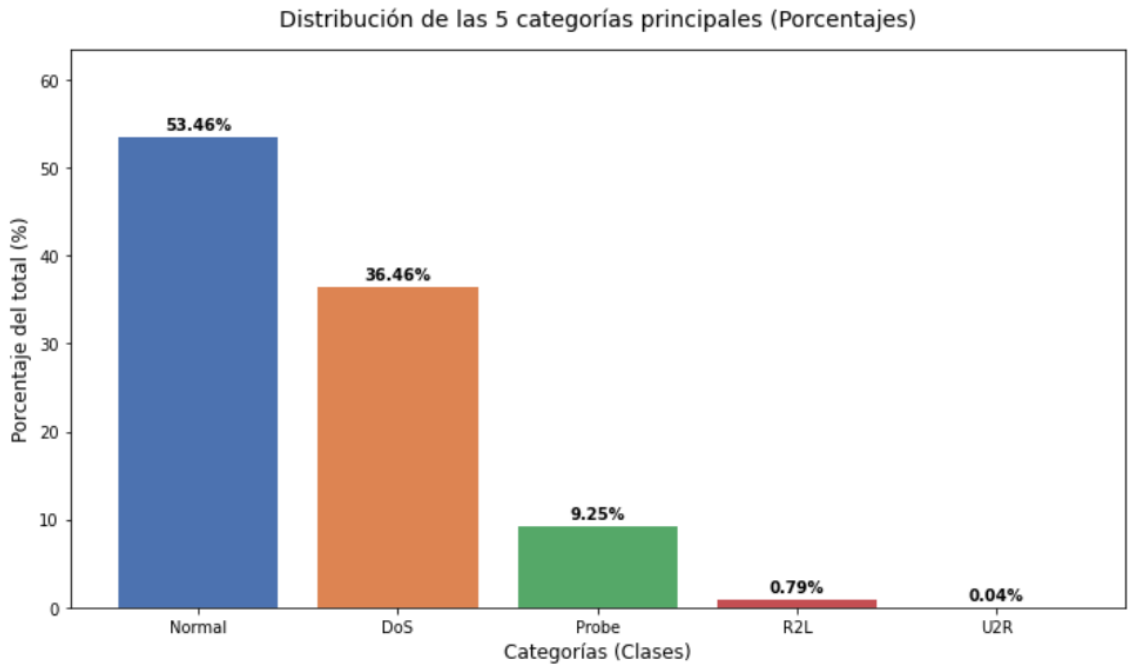

Figura 5.2 Diagrama de barras de la distribución del dataset de entrenamiento

El tráfico de red capturado en NSL-KDD no solo clasifica el estado del sistema como "anómalo" o "normal", sino que categoriza detalladamente las conexiones. A nivel conceptual, el sistema desarrollado en este proyecto aborda un problema de clasificación multiclase, mapeando cada conexión en una de las siguientes cinco familias principales:

- **Normal:** Representa el tráfico legítimo y seguro del sistema. Son conexiones rutinarias que no parecen tener ningún comportamiento malicioso, como la navegación web estándar o el tráfico de red interno esperado.
- **DoS *(Denial of Service):*** Ataques de denegación de servicio. Su objetivo es agotar la memoria de la máquina víctima, impidiendo que atienda peticiones legítimas. Ejemplos clásicos incluyen ataques Smurf o Neptune.
- **Probe:** Ataques de sondeo, vigilancia y escaneo. El atacante recopila información sobre la red objetivo, buscando puertos abiertos, mapeando topologías o identificando vulnerabilidades antes de lanzar un ataque más perjudicial.
- **R2L (*Remote to Local*):** Ataques de acceso remoto no autorizado. Ocurren cuando un atacante envía paquetes a una máquina a través de la red careciendo de una cuenta válida, buscando explotar alguna vulnerabilidad para ganar acceso local (por ejemplo, adivinar contraseñas mediante fuerza bruta).
- **U2R (*User to Root*):** Ataques de escalada de privilegios. En este escenario, el atacante ya ha conseguido acceso a una cuenta de usuario normal en el sistema y busca explotar vulnerabilidades para adquirir privilegios de administrador o administrador. Son los ataques menos frecuentes en volumen, pero los más críticos.

Para modelar matemáticamente estas conexiones, el dataset NSL-KDD extrae un total de 41 características por cada vector de red, más una columna adicional correspondiente a la etiqueta de clase, el objetivo a predecir. Estas variables abarcan desde la información básica del encabezado del paquete TCP/IP hasta características de contenido derivadas del conocimiento del dominio. A continuación, se desglosan en la Tabla 5.1, divididas en sus cuatro categorías funcionales:

Tabla 5-1 Descripción de las 41 características del dataset NSL-KDD

Nombre de la característica	Tipo	Descripción Breve
duration	Continua	Longitud (en segundos) de la conexión.
protocol_type	Categórica	Tipo de protocolo (tcp, udp, icmp).

service	Categórica	Servicio de red destino (http, telnet, ftp...).
flag	Categórica	Estado de la conexión según flags (SF, S0, REJ...).
src_bytes	Continua	Bytes enviados desde el origen al destino.
dst_bytes	Continua	Bytes enviados desde el destino al origen.
land	Discreta (0/1)	Es 1 si origen y destino comparten IP/puerto.
wrong_fragment	Continua	Número de fragmentos erróneos detectados.
urgent	Continua	Número de paquetes urgentes.
hot	Continua	Número de indicadores de compromiso (ej. directorios de sistema).
num_failed_logins	Continua	Intentos fallidos de inicio de sesión.
logged_in	Discreta (0/1)	Es 1 si el login ha sido exitoso.
num_compromised	Continua	Número de condiciones comprometidas detectadas.
root_shell	Discreta (0/1)	Es 1 si se obtiene un terminal de root.
su_attempted	Discreta (0/1)	Es 1 si se intenta ejecutar el comando su root.
num_root	Continua	Número de accesos root en la sesión.

num_file_creations	Continua	Número de archivos creados.
num_shells	Continua	Número de terminales de comandos iniciados.
num_access_files	Continua	Intentos de acceso a archivos de control de acceso.
num_outbound_cmds	Continua	Número de comandos salientes en una sesión ftp.
is_host_login	Discreta (0/1)	Es 1 si el inicio de sesión pertenece a la lista de hosts.
is_guest_login	Discreta (0/1)	Es 1 si la sesión corresponde a un usuario invitado.
count	Continua	Conexiones hacia el mismo host destino.
srv_count	Continua	Conexiones hacia el mismo servicio destino.
serror_rate	Continua	% de conexiones con error SYN.
srv_serror_rate	Continua	% de conexiones con error SYN (mismo servicio).
rerror_rate	Continua	% de conexiones con error REJ.
srv_rerror_rate	Continua	% de conexiones con error REJ (mismo servicio).
same_srv_rate	Continua	% de conexiones al mismo servicio.
diff_srv_rate	Continua	% de conexiones a servicios diferentes.

srv_diff_host_rate	Continua	% de conexiones a diferentes hosts (mismo servicio).
dst_host_count	Continua	Recuento de conexiones al mismo host destino.
dst_host_srv_count	Continua	Recuento de conexiones al mismo host y servicio.
dst_host_same_srv_rate	Continua	% de conexiones al mismo servicio.
dst_host_diff_srv_rate	Continua	% de conexiones a servicios diferentes.
dst_host_same_src_port_rate	Continua	% de conexiones desde el mismo puerto origen.
dst_host_srv_diff_host_rate	Continua	% de conexiones a hosts diferentes (mismo servicio).
dst_host_serror_rate	Continua	% de conexiones con error SYN hacia el host.
dst_host_srv_serror_rate	Continua	% de conexiones con error SYN (mismo host y servicio).
dst_host_rerror_rate	Continua	% de conexiones con error REJ hacia el host.
dst_host_srv_rerror_rate	Continua	% de conexiones con error REJ (mismo host y servicio).

La riqueza y heterogeneidad de estas variables evidencian la necesidad de aplicar una sólida etapa de ingeniería de datos previa al entrenamiento de los modelos. Los algoritmos predictivos no pueden asimilar texto crudo ni variables con escalas distintas como, por

ejemplo, comparar el estado binario de *logged_in* con el volumen de bytes continuo en *src_bytes*. Por este motivo, el sistema requiere una fase rigurosa de transformación de datos, que se detallará en el siguiente subapartado.

5.2.2 LIMPIEZA Y TRANSFORMACIÓN DE DATOS

Como se ha demostrado en la descripción de las características, el dataset NSL-KDD contiene una mezcla de variables continuas, discretas y categóricas. Los algoritmos de *Machine Learning*, en su gran mayoría, requieren que las entradas sean vectores numéricos estandarizados para poder realizar cálculos de distancias y optimizaciones de gradiente de manera eficiente. Por tanto, se diseñó un pipeline de preprocesamiento estructurado en tres fases fundamentales:

1. **Codificación de variables categóricas (Encoding):** Las variables *protocol_type*, *service* y *flag* contienen información en formato de texto ("tcp", "http", "SF"). Para que los modelos matemáticos puedan procesar esta información, se aplicaron técnicas de codificación. Dependiendo del modelo y de la variable, se transformaron estos atributos en representaciones numéricas, garantizando que el algoritmo no asuma relaciones de orden inexistentes entre categorías independientes.
2. **Escalado y Normalización:** Atributos como *src_bytes*, que puede tener valores en los millones y *wrong_fragment* que suele ser un número bajo, presentan escalas muy diferentes. Si no se normalizan, las variables con magnitudes mayores dominarían el cálculo del modelo, ocultando la relevancia de características más sutiles. Se aplicaron técnicas de estandarización como *StandardScaler* para centrar las variables numéricas continuas en una media de cero y una desviación estándar de uno.
3. **Balanceo de Clases:** Uno de los mayores retos del tráfico de red en el mundo real, reflejado en el NSL-KDD, es el desbalanceo de clases. Mientras que las conexiones "Normal" o los ataques "DoS" son extremadamente abundantes, intrusiones críticas como "U2R" o "R2L" aparecen con mucha menor frecuencia. Para evitar que los modelos desarrollen un sesgo predictivo hacia las clases mayoritarias, se integró el parámetro `class_weight='balanced'` durante la fase de entrenamiento. Esta

técnica asigna dinámicamente un peso inversamente proporcional a la frecuencia de la clase, penalizando de forma más severa los errores de clasificación en los ataques minoritarios.

Todos los transformadores estadísticos y codificadores generados en esta fase se exportaron en formato `.pkl` para ser integrados posteriormente en el servidor. De esta manera, el tráfico nuevo introducido en la plataforma sufre exactamente la misma transformación matemática antes de ser evaluado por los modelos.

5.3 MODELOS MULTICLASE

Una vez finalizada la fase de preprocesamiento, el siguiente paso en el desarrollo de NetRadar es la construcción del motor predictivo. A diferencia de los Sistemas de Detección de Intrusos (IDS) tradicionales basados en firmas, que solo reconocen ataques previamente registrados en una lista negra, NetRadar emplea modelos de *Machine Learning* capaces de identificar patrones anómalos y generalizar frente a amenazas desconocidas.

El problema abordado en este proyecto no se limita a una clasificación binaria, es decir, va más allá de la distinción entre tráfico normal y ataque. Se ha planteado como un desafío de clasificación multiclase. El objetivo algorítmico es predecir a qué familia específica pertenece una conexión de red, asignándola a una de las cinco categorías definidas en la base de datos NSL-KDD: Normal, DoS, Probe, R2L o U2R.

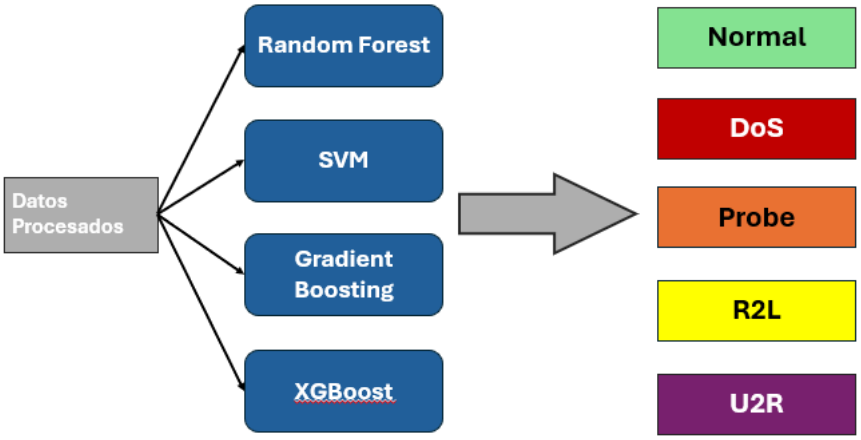

Figura 5.3 Proceso de clasificación de datos

Para garantizar un análisis forense riguroso y comparativo, se optó por implementar cuatro algoritmos distintos de aprendizaje supervisado. El desarrollo de estos modelos se ha llevado a cabo principalmente utilizando la librería matemática Scikit-Learn de Python, estandarizando el flujo de trabajo mediante la creación de *pipelines* que culminan en la exportación de los modelos entrenados en archivos binarios (`.pkl`). A continuación, se detalla la fundamentación matemática, el funcionamiento interno y la configuración de hiperparámetros de cada uno de los modelos desarrollados.

5.3.1 RANDOM FOREST

5.3.1.1 Fundamento teórico y funcionamiento

Random Forest es un algoritmo de aprendizaje supervisado basado en el concepto de aprendizaje conjunto (*Ensemble Learning*), específicamente utilizando la técnica de *Bagging* (Bootstrap Aggregating). Su funcionamiento interno se basa en la construcción de múltiples árboles de decisión independientes durante la fase de entrenamiento.

Cuando un paquete de red entra en el modelo Random Forest de NetRadar, no es evaluado por un único bloque lógico, sino por todo un "bosque" de árboles de decisión. Cada árbol evalúa las características del paquete: puerto, bytes, flags de error, etc. y emite un "voto" prediciendo la clase de la conexión, por ejemplo, DoS. La predicción final del modelo es la

moda estadística, es decir, la clase que ha recibido la mayoría de los votos de los árboles individuales. [7]

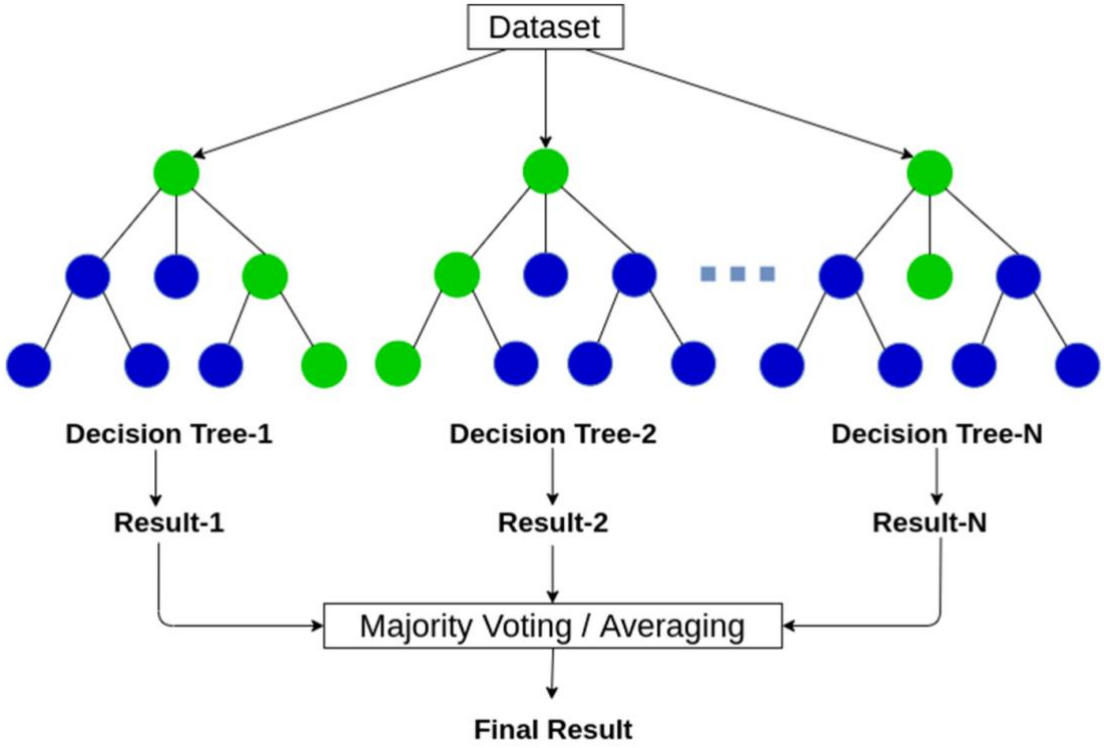

Figura 5.4 Árbol de decisión

De esta manera Random Forest obtiene una gran robustez. Dado que cada árbol se entrena con un subconjunto aleatorio de datos y características, el algoritmo reduce drásticamente la varianza y es altamente resistente al *overfitting*, un problema común cuando se analizan bases de datos tan densas como el NSL-KDD.

5.3.1.2 Implementación y configuración de hiperparámetros

A nivel de código, el modelo se instanció utilizando la clase `RandomForestClassifier` de la librería `sklearn.ensemble`. Para optimizar su rendimiento en la detección de intrusiones, se configuraron los siguientes hiperparámetros clave:

- ✓ **`n_estimators=100`**: Define el número de árboles de decisión que componen el bosque. Se estableció en 100 empíricamente, logrando un equilibrio óptimo entre la

precisión de la detección y el coste computacional, es decir, el tiempo de entrenamiento y la latencia en la predicción.

- ✓ **`class_weight='balanced'`:** Este es el parámetro más importante de la implementación. Dado que ataques como U2R y R2L son minoritarios frente al tráfico Normal y DoS, este hiperparámetro ajusta automáticamente los pesos de las clases de forma inversamente proporcional a su frecuencia en los datos de entrada. Así, el modelo penaliza severamente a los árboles que se equivocan al clasificar los ataques críticos y minoritarios.

- ✓ **`random_state=42`:** Se establece una semilla fija para el generador de números pseudoaleatorios, garantizando que el entrenamiento sea determinista y los resultados sean completamente reproducibles en futuras auditorías del código.

- ✓ **`n_jobs=-1`:** Parámetro de optimización de hardware. Al asignarle el valor -1, se instruye a la librería para que utilice todos los núcleos lógicos disponibles en el procesador del servidor, paralelizando la construcción de los 100 árboles y reduciendo drásticamente el tiempo de compilación.

- ✓ **`criterion='gini'`:** se conservó el criterio de división por defecto de scikit-learn, el índice de impureza de Gini. Este índice mide la probabilidad de clasificar mal una conexión si se le asignara una clase al azar siguiendo la distribución de clases presente en un nodo: vale cero cuando todas las muestras del nodo pertenecen a la misma familia (nodo puro) y crece a medida que se mezclan. En cada división, el árbol elige la variable y el umbral que más reducen esa impureza, separando el tráfico en grupos cada vez más homogéneos. Frente a la entropía, la otra alternativa habitual, el índice de Gini evita el cálculo del logaritmo y resulta algo más rápido de computar, una ventaja que se nota al construir cien árboles sobre las más de 125.000 conexiones del conjunto de entrenamiento. En la práctica ambos criterios producen resultados muy similares, así que se optó por el más eficiente sin penalizar la calidad de la clasificación.

5.3.2 SUPPORT VECTOR MACHINES

5.3.2.1 Fundamento teórico y funcionamiento

Las Máquinas de Vector de Soporte constituyen un algoritmo de aprendizaje supervisado fundamentado en principios de la teoría de aprendizaje estadístico. El objetivo principal de una SVM es encontrar un hiperplano óptimo de separación en un espacio N-dimensional, donde N es el número de variables de la base datos, que clasifique correctamente los puntos de datos minimizando el riesgo empírico general.

A nivel conceptual, los registros de tráfico de red se proyectan como vectores de entrada en este espacio geométrico. La distancia entre el hiperplano decisorio y los puntos más cercanos de cada clase se denomina margen. El algoritmo busca maximizar este margen de separación, asegurando que la "barrera" que divide las distintas categorías se encuentre lo más alejada posible de los datos de entrenamiento. Aquellos vectores críticos que se sitúan exactamente en la frontera del margen y que definen la orientación y posición del hiperplano se conocen como Vectores de Soporte *(Support Vectors)*. Si estos puntos se eliminaran, la posición del hiperplano cambiaría por completo, de ahí su relevancia matemática.

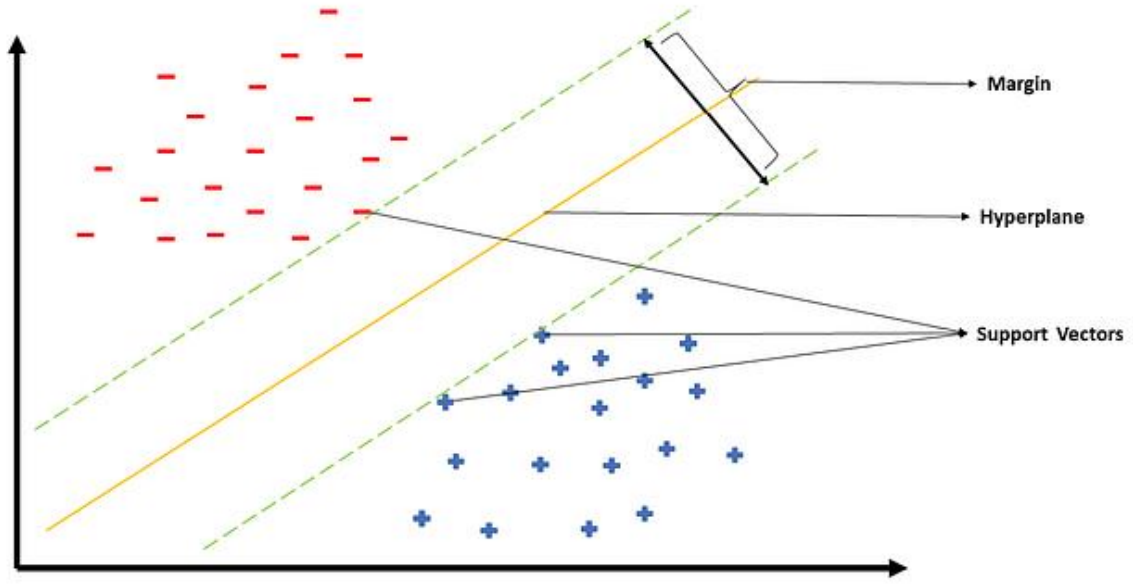

Figura 5.5 Funcionamiento de SVM

Sin embargo, el tráfico de red real capturado en el dataset NSL-KDD es intrínsecamente no lineal; características como el volumen de bytes o las tasas de error no se pueden separar mediante una línea recta o un plano plano. Para resolver esta limitación, las SVM implementan el denominado Truco del Kernel. Esta técnica matemática aplica una función de mapeo no lineal que proyecta los datos originales de baja dimensión a un espacio de características de una dimensionalidad significativamente mayor, lo que se llama espacio de Hilbert. En este nuevo espacio hiperdimensional, un problema de clasificación que originalmente era imposible de resolver linealmente se vuelve perfectamente separable mediante un hiperplano lineal.

5.3.2.2 Adaptación al escenario multiclase

Dado que las SVM están diseñadas matemáticamente de forma nativa para clasificaciones binarias, la librería *Scikit-Learn* resuelve el problema multiclase de NetRadar, que contiene 5 familias de tráfico, implementando la estrategia *One-vs-One (OvO)*. Bajo este enfoque, el sistema no entrena un único modelo global, sino un conjunto de clasificadores binarios individuales para cada combinación posible de parejas de clases:

$$Nº\ de\ clasificadores = \frac{K \cdot (K-1)}{2} = \frac{5 \cdot (5-1)}{2} = 10\ clasificadores\ binarios$$

Cuando el backend procesa un archivo forense, el paquete de red pasa por los 10 subclasificadores (Normal vs DoS, Normal vs Probe, DoS vs Probe, etc.)

Cada clasificador emite un voto y el sistema construye el veredicto mediante un mecanismo de votación por mayoría, asignando finalmente la etiqueta de la clase que más votos ha obtenido.

5.3.2.3 Implementación y configuración de hiperparámetros

El modelo se programó en el entorno Python mediante la clase SVC (*Support Vector Classification*) del módulo `sklearn.svm`. Debido a la dimensionalidad del NSL-KDD, la optimización y calibración de sus hiperparámetros requirió un análisis riguroso:

- ✓ **`kernel='rbf'`:** Se seleccionó el kernel de Función de Base Radial, también conocido como kernel gaussiano. Es el estándar en ciberseguridad debido a su flexibilidad geométrica, permitiendo al modelo trazar fronteras de decisión curvas y concéntricas muy complejas en el espacio de Hilbert para aislar con precisión anomalías sofisticadas como Probe o R2L.
- ✓ **`C=1.0`:** Es el parámetro de regularización o penalización por error. Actúa como un factor de compensación entre la maximización del margen y la tolerancia a clasificaciones erróneas en el conjunto de entrenamiento. Un valor de C=1.0 establece una regularización suave, permitiendo un cierto grado de flexibilidad (margen blando o *soft-margin*) para evitar que el modelo se sobreajuste a firmas de ruido de red específicas, favoreciendo la generalización ante tráfico desconocido.
- ✓ **`class_weight='balanced'`:** Como en el apartado anterior, esto es indispensable para contrarrestar el submuestreo de los ataques U2R y R2L. El algoritmo modifica internamente la matriz de penalización, multiplicando el parámetro C de cada clase de forma inversamente proporcional a su frecuencia de aparición. Gracias a esto, el hiperplano se desplaza intencionadamente para ensanchar el

margen de las clases minoritarias, evitando que pasen desapercibidas geométricamente por la gran cantidad de los registros normales o de denegación de servicio que hay.

- ✓ **`probability=True`:** Por defecto, las SVM solo devuelven la etiqueta de la clase predicha de forma determinista. Al activar este parámetro, se obliga al modelo a emplear internamente el método de Calibración de Platt, un ajuste de regresión logística sobre las distancias geométricas al hiperplano. Esto permite al backend extraer un vector con las probabilidades de cada clase (por ejemplo: 92% DoS, 5% Normal, 3% Probe). Este JSON con porcentajes es crítico para que la interfaz web pueda renderizar las gráficas dinámicas de confianza para el analista.
- ✓ **`gamma='scale'`:** el parámetro gamma controla el alcance de la influencia de cada vector de soporte en el kernel RBF, es decir, hasta qué distancia llega el efecto de un punto de entrenamiento sobre la frontera de decisión. Con un gamma muy alto esa influencia queda muy localizada y el modelo traza fronteras muy ceñidas a cada muestra, lo que tiende al sobreajuste; con un gamma demasiado bajo la influencia se extiende en exceso y la frontera se vuelve casi plana, perdiendo capacidad para separar las clases. El valor 'scale', que es el de por defecto en scikit-learn, fija gamma en 1 dividido por el producto del número de características y la varianza de los datos, de modo que se adapta de forma automática a la escala del conjunto. Como las variables de NetRadar se estandarizan previamente con StandardScaler (media cero y varianza uno), 'scale' equivale aproximadamente a 1 entre el número de características y ofrece un punto de partida bien calibrado, sin necesidad de ajustar el parámetro a mano.

5.3.3 GRADIENT BOOSTING

5.3.3.1 Fundamento teórico y funcionamiento

A diferencia de Random Forest, que basa su arquitectura en el paradigma de *Bagging*, Gradient Boosting pertenece a la familia de algoritmos de Boosting. Esta metodología de aprendizaje supervisado no busca crear un modelo robusto combinando las opiniones de

muchos árboles complejos, sino que se basa en el principio de la mejora continua, combinando secuencialmente múltiples "aprendices débiles" (*weak learners*), que en este caso son árboles de decisión muy poco profundos.

El funcionamiento interno del algoritmo es estrictamente secuencial y se fundamenta en la optimización del error mediante el descenso de gradiente (*Gradient Descent*). El proceso comienza entrenando un primer árbol de decisión básico sobre el conjunto de datos de red NSL-KDD. Como es un aprendiz débil, este árbol cometerá errores de clasificación. El algoritmo evalúa estos fallos calculando los residuos, la diferencia matemática entre la predicción real y la etiqueta verdadera.

A continuación, en lugar de entrenar un segundo árbol con los datos originales, el segundo árbol se entrena exclusivamente para predecir y corregir los residuos (errores) del primer árbol. El tercer árbol corregirá los errores residuales del segundo, y así sucesivamente.

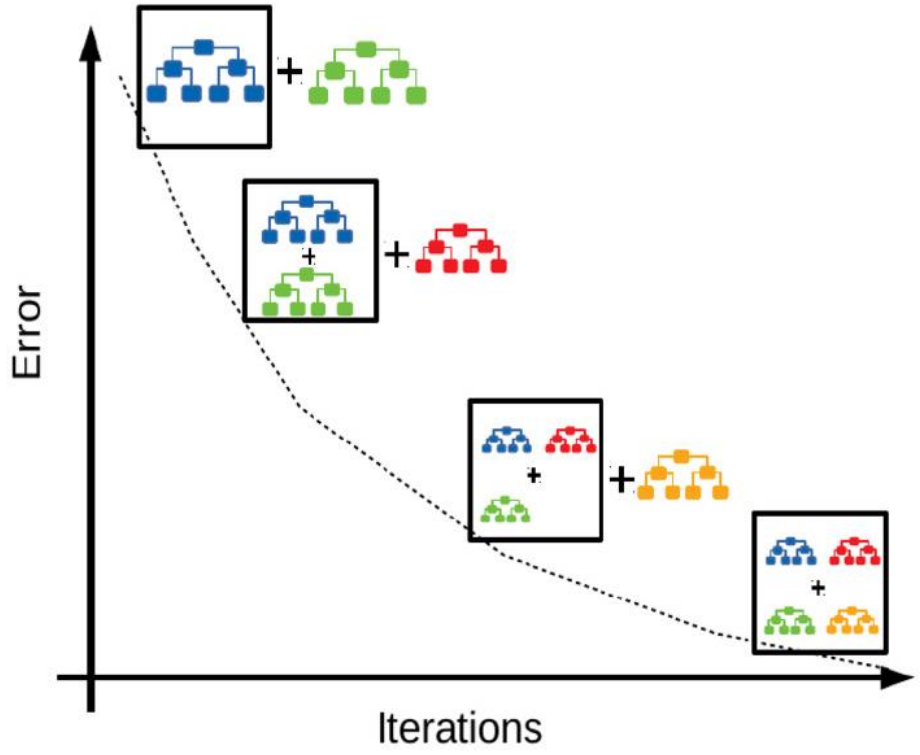

Figura 5.6 Funcionamiento Gradient Boosting

Matemáticamente, el modelo final $F_M(x)$ después de M iteraciones se construye sumando las predicciones de todos los árboles anteriores, ponderadas por una tasa de aprendizaje υ:

$$F_M(x) = F_{0(x)} + \sum_{m=1}^{M} \upsilon \cdot h_m(x)$$

Donde $h_m(x)$ representa el árbol de decisión entrenado en la iteración m, y υ es el *learning rate* que controla la contribución de cada nuevo árbol.

5.3.3.2 Adaptación al escenario multiclase

Para abordar la clasificación en las cinco categorías finales, Gradient Boosting no utiliza la estrategia Uno-contra-Uno de las SVM. En su lugar, el algoritmo construye internamente un conjunto de árboles para cada clase en cada iteración. Si se configuran 50 iteraciones para 5 clases, el modelo construirá y evaluará un total de 250 árboles direccionales. Para emitir el veredicto final, los valores matemáticos de las hojas de los árboles se transforman en probabilidades porcentuales utilizando la función de activación *Softmax*, asignando la conexión a la clase con mayor nivel de confianza.

5.3.3.3 Implementación y configuración de hiperparámetros

El modelo se implementó utilizando la clase `GradientBoostingClassifier` de Scikit-Learn. Dado que este algoritmo es propenso al sobreajuste si se le permite crecer sin límites, los parámetros se escogieron con mucho cuidado:

- ✓ **`n_estimators=50`**: Define el número de etapas de boosting, el número de iteraciones. Se fijó en 50 para permitir que el modelo alcance la convergencia en la minimización del error, compensando el coste computacional que conlleva un algoritmo que no puede ser paralelizado, ya que cada árbol depende estructuralmente del anterior.

- ✓ **`learning_rate=0.1`**: Controla la magnitud de las correcciones que cada nuevo árbol aporta al modelo general. Un valor de 0,1 es un estándar en la industria de la ciberseguridad, ya que obliga al modelo a aprender "lentamente". Esta restricción

mejora drásticamente su capacidad de generalización frente a ataques de red nunca antes vistos, evitando que el algoritmo memorice el ruido de los datos.

- ✓ `random_state=42:` Se establece una semilla de aleatoriedad fija. Aunque el entrenamiento principal es secuencial, las submuestras internas que utiliza el algoritmo se rigen por esta semilla, garantizando que el modelo y sus resultados sean completamente reproducibles en ejecuciones futuras.

- ✓ **Manejo del desbalanceo (`sample_weight`):** a diferencia de Random Forest, la clase GradientBoostingClassifier de Scikit-Learn no dispone del parámetro `class_weight`. Para que el modelo no ignorase los ataques minoritarios (U2R y R2L), el balanceo se aplica de forma explícita durante el entrenamiento mediante el argumento `sample_weight` del método fit. Antes de entrenar se calcula un peso por muestra inversamente proporcional a la frecuencia de su clase, con la función `compute_sample_weight("balanced", y_train)`, de manera que un error sobre una conexión U2R penaliza mucho más que un error sobre tráfico Normal. Este mecanismo es el que recupera buena parte de la categoría U2R: como se detalla en el Capítulo 6, Gradient Boosting alcanza un recall de 0,52 en esa clase, el mejor de todos los modelos. El efecto del balanceo es, por tanto, un resultado medible del entrenamiento, no una propiedad implícita del algoritmo.

5.3.4 eXTREME GRADIENT BOOSTING (XGBOOST)

5.3.4.1 Fundamento teórico y funcionamiento

El algoritmo eXtreme Gradient Boosting, conocido como XGBoost, es una implementación avanzada, optimizada y escalable del modelo de *Gradient Boosting* detallado en el apartado anterior. Desarrollado originalmente por Tianqi Chen, no cambia la filosofía matemática subyacente, el aprendizaje secuencial mediante la corrección de residuos, sino que introduce una reingeniería profunda en la forma en que los árboles de decisión se construyen y procesan a nivel de *hardware* y *software*.

Mientras que el *Gradient Boosting* clásico construye los árboles de forma estrictamente secuencial, lo que supone un cuello de botella computacional, XGBoost sortea esta limitación mediante una técnica de paralelización a nivel de nodo. Aunque no puede paralelizar la creación de árboles sucesivos porque el árbol *M* siempre dependerá del error del árbol *M-1*, sí puede paralelizar el proceso de búsqueda de las divisiones óptimas dentro de un mismo árbol. Utilizando algoritmos de ordenación de datos aproximados y almacenamiento en caché eficiente, XGBoost evalúa múltiples variables de red como src_bytes, duration, flag simultáneamente en todos los núcleos disponibles del servidor.

Además de su velocidad, la principal aportación matemática de XGBoost frente a su predecesor es la inclusión nativa de términos de regularización en su función objetivo. La función de pérdida que el algoritmo intenta minimizar no solo penaliza la diferencia entre la predicción y el valor real del ataque, sino que añade un término Ω(f) que penaliza la complejidad excesiva del árbol. Por ejemplo, tener demasiadas hojas o pesos muy grandes. Esta regularización, análoga a las técnicas L1 y L2 (Lasso y Ridge), convierte a XGBoost en un modelo extremadamente resistente al sobreajuste, una característica necesaria para evitar falsos positivos ante el ruido impredecible del tráfico web normal.

5.3.4.2 Adaptación al escenario multiclase

Al igual que el Gradient Boosting tradicional, XGBoost aborda la clasificación multiclase entrenando internamente múltiples secuencias de árboles, una por cada clase objetivo, y aplicando finalmente la función *Softmax* para calcular el vector de probabilidades y extraer el veredicto más probable.

5.3.4.3 Implementación y configuración de hiperparámetros

El algoritmo XGBoost necesita una librería que, en algunos servidores o sistemas operativos, podría no funcionar. Para asegurar que la plataforma de NetRadar no colapse, a diferencia de los tres modelos anteriores, se ha programado un mecanismo de tolerancia a fallos mediante un bloque condicional `try-execept`. El código intenta importar la librería externa `xgboost`. Si el entorno no cuenta con ella, el sistema aplica un *fallback*, es decir un plan de

respaldo, automático, que importa en su lugar `HistGradientBoostingClassifier` de Scikit-Learn. Este algoritmo alternativo está diseñado para procesar grandes volúmenes de datos a velocidades similares a XGBoost. En caso de usar este algoritmo, se renombra internamente para que el resto del sistema pueda seguir entrenando y prediciendo ataques de red sin interrupciones de servicio.

Para la instanciación del modelo, se configuraron los siguientes parámetros:

- ✓ **`n_estimators=50`** (o **`max_iter=50`** en el modelo de respaldo): Establece el límite de árboles a construir. Al reducir este número a 50 (al igual que en el Gradient Boosting clásico), se permite una comparativa de rendimiento justa entre ambos enfoques secuenciales y se garantiza una latencia de predicción en el servidor casi instantánea.
- ✓ **`learning_rate=0.1`:** La tasa de aprendizaje se mantiene en este estándar de la industria para asegurar una curva de convergencia suave y garantizar que la corrección de errores residuales se realice de forma conservadora, favoreciendo la generalización del modelo forense.
- ✓ **`random_state=42`:** Al fijar esta semilla, se garantiza que la construcción paralela de los nodos y las submuestras generen exactamente el mismo bosque de árboles en cada ejecución del servidor, facilitando la auditoría de los resultados.
- ✓ **Manejo del desbalanceo (`sample_weight`):** igual que Gradient Boosting, XGBoost no emplea `class_weight`, por lo que recibe los mismos pesos por muestra calculados con `compute_sample_weight("balanced", y_train)` a través del argumento `sample_weight` de fit. Gracias a este ajuste detecta una parte de los ataques U2R, llegando a obtener un recall de 0,34, una clase que sin balanceo quedaría prácticamente sin cubrir.

5.4 DESARROLLO DEL BACKEND Y API REST

La capa de lógica, o Backend, es el motor central de NetRadar: gestiona la comunicación entre la interfaz, los modelos de Machine Learning y la base de datos. Está construida con

FastAPI, cuya elección y características se justificaron en el apartado 4.2.2; aquí se describe cómo se ha empleado en el proyecto. El desarrollo del servidor se dividió en dos bloques: el diseño de las rutas de comunicación (la API REST) y el diseño del modelo de datos (la persistencia).

5.4.1 ENDPOINTS

La interacción entre el navegador del cliente y el servidor se realiza exclusivamente a través de peticiones HTTP estandarizadas bajo una arquitectura RESTful. Para cubrir los requisitos funcionales de la plataforma, se han implementado tres *endpoints* principales, cuyo flujo de datos e integración con los componentes internos se ilustra en la Figura 5.7, el resto de *endpoints* no están representados en la figura, pero funcionan de forma similar.

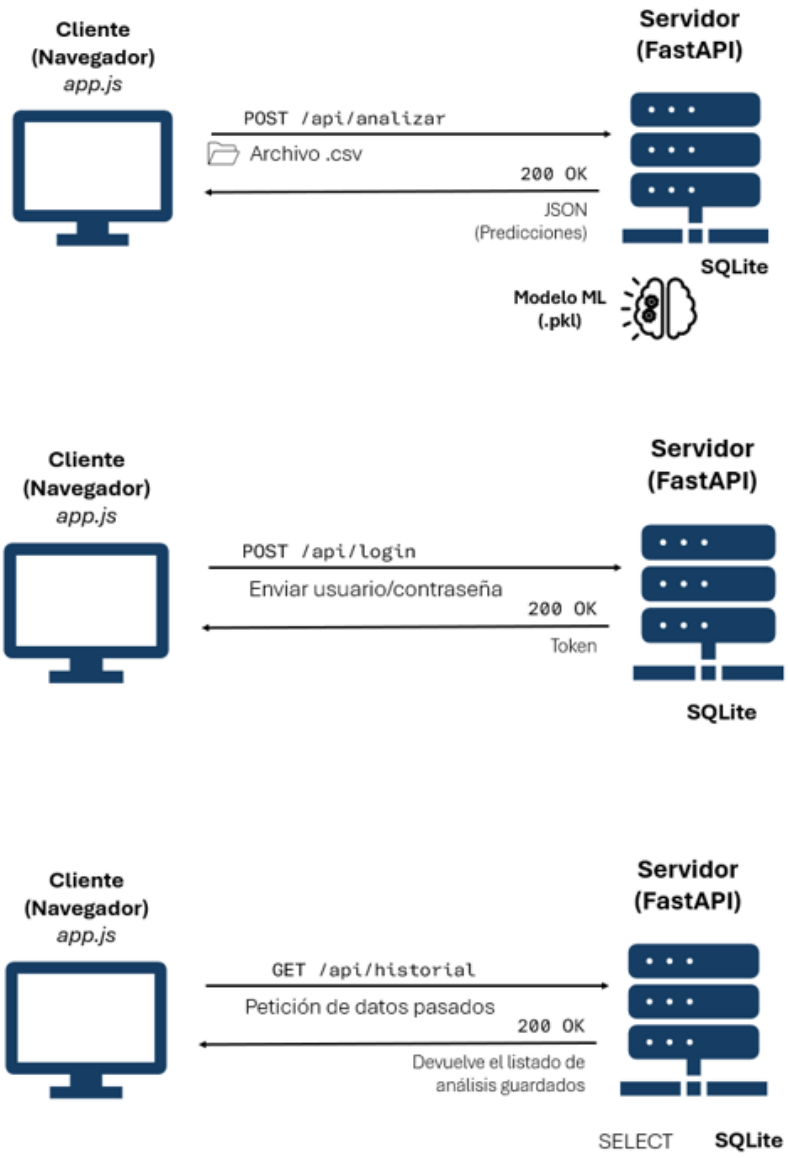

Figura 5.7 Diagrama de flujo de las peticiones HTTP y respuestas en la API REST de NetRadar

Como se observa en el diagrama, la API expone las siguientes rutas:

- **`POST /api/auth/login`:** Encargado de la autenticación y control de acceso. El cliente envía las credenciales del usuario: `username` y `password`. Seguidamente, el servidor las verifica contra la base de datos y, en caso de que coincidan, devuelve un código de estado 200 OK con el identificador único del usuario (`usuario_id`) y el nombre del usuario para personalizar la web a cada persona.

- **`POST /api/analizar:`** es el núcleo computacional del sistema. Recibe el archivo .csv con las capturas de tráfico y la lista de modelos que el analista ha seleccionado, que viaja en el parámetro de consulta modelos. Como criterio de optimización, los archivos binarios (.pkl) del preprocesador y de los cuatro modelos se cargan una sola vez al arrancar el servidor, lo que evita lecturas de disco repetidas en cada petición. El endpoint preprocesa los datos y ejecuta únicamente los modelos solicitados, uno o varios, de modo que permite comparar algoritmos cuando interesa o lanzar un solo modelo cuando se busca rapidez. La interfaz preselecciona un modelo por defecto y el analista puede activar los demás antes de analizar. Por último, guarda el registro del análisis en la base de datos y devuelve un objeto JSON con el desglose de las predicciones.

- **`GET /api/historial`**: Recibe el `usuario_id` del cliente y ejecuta una consulta selectiva ordenando los registros de forma descendente. Devuelve un listado JSON con los análisis pasados realizados por ese perfil específico, incluyendo el nombre del archivo, la fecha exacta y el volumen total de tráfico inspeccionado.

  El servidor consta de rutas no mencionadas en el diagrama:

- **`POST /api/auth/register`**: Endpoint destinado al registro de nuevos usuarios en la plataforma. Recibe el nombre, nombre de usuario, contraseña y el par de pregunta/respuesta de seguridad, almacenándolos en la persistencia del sistema.
- **`POST /api/auth/recover`**: Endpoint dedicado a la recuperación de credenciales. Permite al usuario restablecer su contraseña en caso de olvido, validando su identidad mediante la pregunta de seguridad y la respuesta, que se comprueba contra su hash bcrypt almacenado.
- **`GET /api/dashboard/stats`**: Endpoint de analíticas globales. Recibe el `usuario_id` y calcula métricas agregadas en tiempo real (total de archivos subidos por el usuario, suma de todos los registros de red analizados históricamente y los detalles del último archivo procesado) para renderizar el panel de bienvenida de la aplicación.

Conviene distinguir dos niveles de validación en la API. Los endpoints de autenticación reciben datos estructurados (usuario, contraseña, pregunta y respuesta de seguridad) declarados como modelos de Pydantic; FastAPI rechaza de forma automática cualquier petición cuyo cuerpo no se ajuste a esos modelos o cuyos tipos no coincidan, y lo mismo ocurre con parámetros como `usuario_id`, declarado como entero. El *endpoint* de análisis es distinto: recibe un fichero CSV cuyo contenido el *framework* no puede validar por sí mismo. Esa comprobación se realiza dentro de la función `preparar_datos()`, que ordena las columnas según las que espera el modelo, codifica las variables categóricas con los codificadores guardados (asignando un 0 a los valores no vistos en el entrenamiento), convierte a numéricas las demás columnas y aplica el escalador. La validación de los datos que alimentan a los modelos es, por tanto, una responsabilidad explícita de la aplicación y no una garantía automática del *framework*.

5.4.2 MODELO DE DATOS Y PERSISTENCIA

Como ya se explicó en el apartado 4.4.1, el sistema usa SQLite como base de datos embebida.

El diseño del esquema relacional se ha optimizado para cumplir con los principios de normalización, asegurando la integridad referencial de los datos. La estructura se compone de dos entidades principales, ilustradas en la siguiente figura:

Figura 5.8 Diagrama Entidad-Relación (ER) de la base de datos de NetRadar.

El acceso a datos va por SQLAlchemy, un ORM que evita escribir SQL a mano y cierra la puerta a inyecciones de código. El esquema tiene dos tablas con relación 1:N:

- **Tabla usuarios:** credenciales y perfil de cada analista. Guarda id, username, password, nombre real y los campos de recuperación de cuenta (pregunta_seguridad, respuesta_seguridad).

- **Tabla analisis:** cada fila es un análisis forense completo: el archivo cargado (nombre_archivo), cuándo se hizo (fecha_analisis), cuántas conexiones se procesaron (`total_registros`) y quién lo ejecutó (usuario_id). Los resultados de los cuatro modelos de IA se almacenan serializados como JSON en detalles_modelos_json, lo que evita añadir más tablas al esquema.

Separar autenticación de historial de análisis tiene una ventaja práctica: las consultas de cada usuario no interfieren entre sí y la latencia se mantiene baja.

5.5 *DESARROLLO DEL FRONTEND E INTERFAZ DE USUARIO*

El frontend de NetRadar parte de un problema concreto: las herramientas de ciberseguridad habituales operan mediante consolas de comandos que requieren un perfil técnico específico. La interfaz desarrollada busca reducir esa barrera, utilizando HTML5, CSS3 y JavaScript sin dependencias de *frameworks* de componentes, solo `Chart.js` para la visualización.

Este apartado se estructura en tres bloques: la arquitectura del código, la identidad visual y el flujo de las vistas.

5.5.1 ARQUITECTURA FRONTEND (SINGLE PAGE APPLICATION)

A diferencia del modelo web clásico, en el que cada acción del usuario provoca una nueva petición al servidor y la carga de un documento HTML distinto, NetRadar se ha implementado como una Single Page Application (SPA). Esto implica que el navegador realiza una única carga inicial del archivo `index.html`, que ya contiene todos los contenedores de la aplicación: panel de autenticación, dashboard analítico, módulo de

inferencia e historial de análisis. La visibilidad de cada bloque se controla mediante la clase `CSS .hidden`, de modo que en el arranque solo se muestra la pantalla de inicio de sesión.

El control de la navegación recae íntegramente en `app.js`, que actúa como controlador central de la aplicación. Su funcionamiento se apoya en dos mecanismos:

1. **Manipulación del DOM:** El script intercepta los eventos generados por el usuario (clics en el menú lateral, envíos de formularios, carga de archivos) y responde modificando el árbol DOM, añadiendo o eliminando `.hidden` en los contenedores afectados. El cambio de vista es inmediato y no requiere ninguna recarga del navegador.
2. **Comunicación asíncrona con el backend:** El intercambio de datos con la API FastAPI se realiza mediante la API fetch de JavaScript, con el patrón *async/await*. El cliente envía las peticiones correspondientes (credenciales, archivos CSV o consultas de historial) y permanece a la espera sin bloquear la interfaz. Una vez recibida la respuesta JSON, el controlador actualiza las tablas y gráficos con los resultados del modelo.

Esta separación de responsabilidades entre presentación y procesamiento es lo que permite que la aplicación responda con rapidez independientemente del volumen de datos analizado.

5.5.2 DISEÑO DE LA INTERFAZ

El punto de partida del diseño fue el logotipo de NetRadar, que combina una nube, un candado y una bola de radar para representar, respectivamente, la infraestructura de red, la seguridad y la capacidad de análisis predictivo. A partir de ese elemento, se definieron la paleta cromática y la jerarquía visual del resto de la interfaz.

Figura 5.9 Logotipo NetRadar

Integración del logotipo y coherencia visual

El menú lateral tiene fondo blanco puro, lo que permite que el logotipo se integre sin bordes ni discontinuidades. Esta decisión, implementada en `style.css`, sigue la estética habitual de plataformas profesionales de ciberseguridad, donde la marca ocupa el *sidebar* sin elementos decorativos que compitan con ella.

Sistema de contraste y legibilidad

El área de trabajo central utiliza un gris claro como fondo. Los paneles de análisis, formularios y resultados se muestran en blanco, lo que genera un efecto de elevación que separa visualmente el contenido del fondo y reduce la fatiga visual en sesiones de trabajo

prolongadas.

Figura 5.10 Página de analizador de datos con sidebar abierta

Paleta de estados para la clasificación de amenazas

El color base de la plataforma es el azul corporativo. Las etiquetas de clasificación de tráfico utilizan tonos pastel diferenciados por tipo de ataque, permitiendo identificar la categoría de una conexión sin leer el texto de la etiqueta:

- Normal: verde
- DoS: rojo
- Probe: amarillo
- R2L: azul
- U2R: morado oscuro

Toda la paleta está definida mediante variables CSS, lo que desacopla los valores de color de los componentes y simplifica cualquier ajuste posterior de la identidad visual.

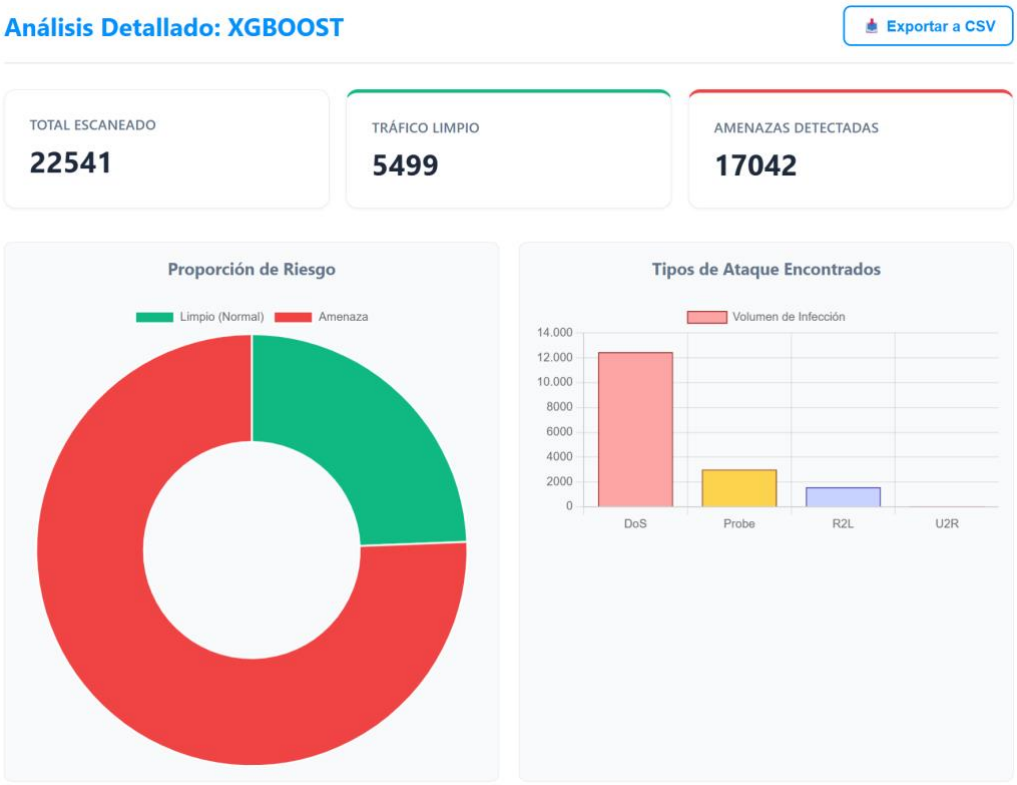

Figura 5.11 Análisis detallado del modelo XGBoost en la plataforma de NetRadar

Capítulo 6. ANÁLISIS DE RESULTADOS

Una vez entrenados los cuatro modelos descritos en el capítulo anterior, este capítulo recoge su evaluación sobre datos no vistos y la comparación entre ellos. El objetivo no es solo medir cuántas conexiones se clasifican correctamente, sino entender dónde acierta cada modelo, dónde falla y qué implica todo ello para el uso real de NetRadar como herramienta de detección.

6.1 METODOLOGÍA DE EVALUACIÓN

La evaluación se ha realizado siguiendo el protocolo estándar del conjunto NSL-KDD: los modelos se entrenan con KDDTrain+ y se evalúan con KDDTest+, dos particiones independientes y sin solapamiento. Esta decisión es deliberada y tiene consecuencias en las cifras obtenidas. El conjunto de prueba incluye familias y variantes de ataque que no aparecen en el entrenamiento [1], de modo que la evaluación no mide la capacidad del modelo para reconocer lo que ya ha visto, sino su capacidad de generalizar ante amenazas nuevas. Una alternativa habitual como mezclar ambos ficheros y hacer una partición aleatoria, produce precisiones cercanas al 99 %, pero introduce fuga de información, ya que registros casi idénticos acaban en entrenamiento y prueba a la vez [2]. Se ha descartado por no reflejar el comportamiento real del sistema frente a tráfico desconocido.

El conjunto de prueba contiene 22 541 conexiones distribuidas de forma muy desigual entre las cinco clases:

Tabla 6-1 Distribución de los datos en el dataset

Normal	DoS	Probe	R2L	U2R
9711	7460	2418	2885	67

Esta distribución, junto con la del entrenamiento, se muestra en la Figura 6.1. El desequilibrio es importante porque condiciona qué métricas resultan informativas.

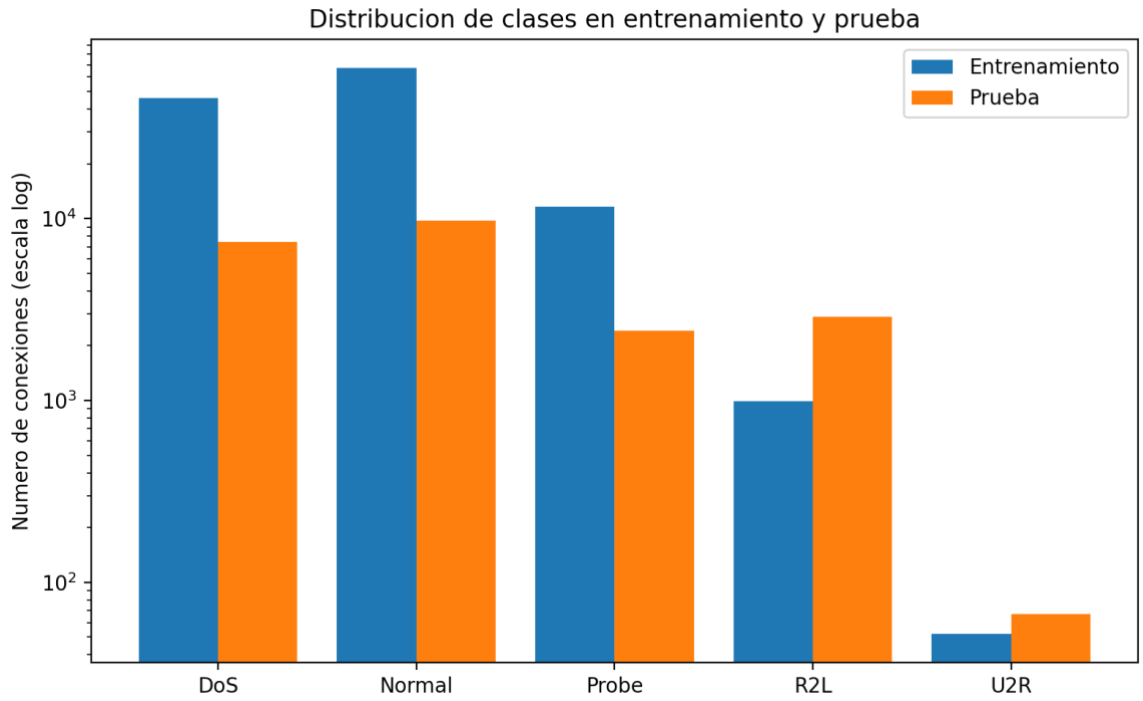

Figura 6.1 Distribución de clases en los conjuntos de entrenamiento vs de prueba

La precisión global *(accuracy)* mide el porcentaje de aciertos sobre el total, pero en un problema tan desbalanceado puede ser engañosa: un modelo que clasificara todo como Normal o DoS obtendría una *accuracy* aparentemente alta ignorando por completo los ataques minoritarios. Por ello, la métrica de referencia en este capítulo es el *F1-score macro*, que promedia el *F1* de cada clase sin ponderar por su frecuencia y penaliza, por tanto, a los modelos que descuidan las clases raras. Se complementa con la precisión y el recall por clase y con la matriz de confusión de cada modelo, que permiten ver el tipo concreto de error cometido.

6.2 RENDIMIENTO GLOBAL Y COMPARATIVA DE MODELOS

La Tabla 6.1 resume el rendimiento de los cuatro modelos sobre el conjunto de prueba.

Tabla 6.1. Precisión global y F1-macro de los cuatro modelos sobre KDDTest+.

Modelo	**Accuracy**	**F1-macro**
Random Forest	73,98 %	48,39 %
SVM	76,55 %	55,26 %
Gradient Boosting	75,44 %	58,53 %
XGBoost	**76,68 %**	**59,67 %**

XGBoost es el modelo más equilibrado: lidera tanto en *accuracy* como en *F1-macro*. Las cuatro precisiones se mueven en una banda estrecha (74-77 %), coherente con los resultados publicados para algoritmos clásicos sobre KDDTest+, donde los mejores clasificadores se sitúan en torno al 78-80 % [3]. Que NetRadar no alcance los valores del 99 % que aparecen en otros trabajos no es una deficiencia, sino la consecuencia directa de evaluar sobre ataques no vistos.

El dato más revelador de la tabla es la divergencia entre las dos columnas. Random Forest y XGBoost tienen una *accuracy* parecida (73,98 % frente a 76,68 %), pero su F1-macro difiere en más de once puntos: 48,39 % frente a 59,67 %. Esto confirma la advertencia de la sección anterior: la *accuracy* oculta el comportamiento sobre las clases minoritarias. Random Forest acierta el grueso del tráfico mayoritario, pero apenas detecta R2L y U2R, lo que hunde su *F1-macro*. La Figura 6.2 representa visualmente esta comparativa.

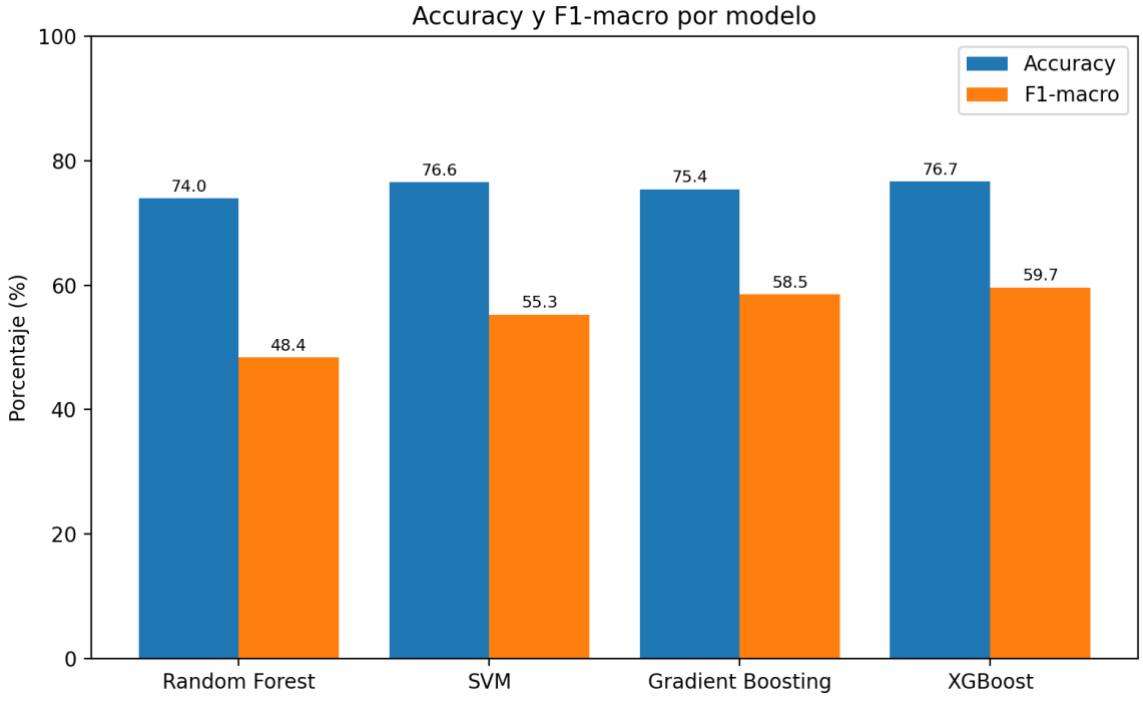

Figura 6.2 Comparativa de métricas por modelo

6.3 ANÁLISIS DE LA DETECCIÓN POR FAMILIA DE ATAQUE

El comportamiento global se entiende mejor descomponiéndolo por clase. La Tabla 6.2 recoge el *recall* (proporción de ataques de cada tipo correctamente detectados) de cada modelo, y la Figura 6.3 lo representa en forma de mapa de calor.

Tabla 6.2. Recall por clase y modelo (KDDTest+).

Modelo	**Normal**	**DoS**	**Probe**	**R2L**	**U2R**
Random Forest	0,97	0,77	0,60	0,00	0,04
SVM	0,96	0,77	0,74	0,12	0,12
Gradient Boosting	0,96	0,77	0,66	0,09	0,52

XGBoost	0,97	0,76	0,80	0,09	0,34

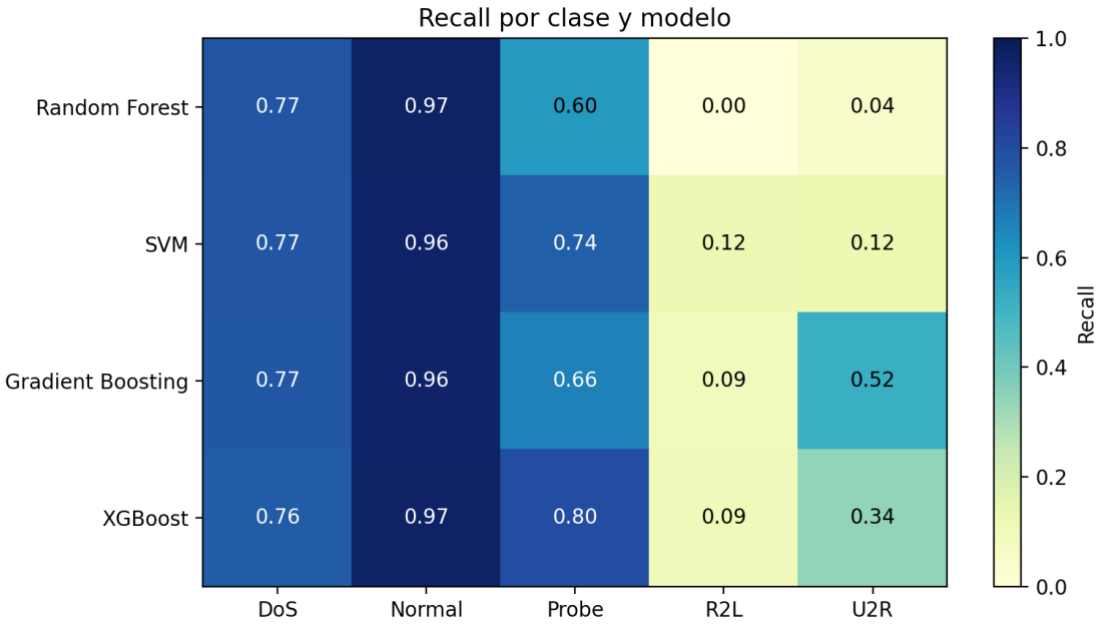

Figura 6.3 Mapa de calor del recall por clase y por modelo

Tráfico Normal y DoS: Las dos clases mayoritarias se gestionan con solvencia. El tráfico Normal se reconoce con un *recall* del 96 al 97% en todos los modelos, y los ataques de denegación de servicio se detectan con una precisión muy alta, 96%, y un *recall* en torno al 77%. Son patrones con una firma estadística marcada, volúmenes de tráfico y tasas de error características, lo que los hace relativamente fáciles de aislar.

Probe: Los ataques de sondeo se detectan de forma aceptable, con diferencias notables entre modelos: XGBoost alcanza un *recall* del 80%, frente al 60% de Random Forest. La estructura secuencial de XGBoost, que concentra el aprendizaje en los ejemplos peor clasificados, parece capturar mejor estos patrones que el promediado de Random Forest.

U2R: Esta clase muestra la importancia del balanceo. Pese a contar únicamente con 52 ejemplos de entrenamiento, Gradient Boosting detecta más de la mitad de los ataques U2R del conjunto de prueba (*recall* 0,52) y XGBoost un tercio (0,34), mientras que Random

Forest, sin pesos por muestra, se queda en 0,04. La escalada de privilegios deja, por tanto, un rastro aprendible siempre que se obligue al modelo a prestarle atención.

R2L: Es la clase donde todos los modelos fracasan. El mejor *recall* lo logra la SVM, y aun así apenas alcanza el 12%. La matriz de confusión de XGBoost (Figura 6.4) lo muestra: de las 2 885 conexiones R2L del conjunto de prueba, 2 313 se clasifican como tráfico Normal. La explicación es doble. Por un lado, R2L pasa de 995 ejemplos en entrenamiento a 2 885 en prueba, un salto de proporción que ningún reajuste de pesos compensa del todo. Por otro, y más importante, los ataques R2L como la obtención de credenciales por fuerza bruta, se comportan, a nivel de paquete, de forma casi indistinguible del tráfico legítimo, y el conjunto de prueba incorpora variantes ausentes del entrenamiento.

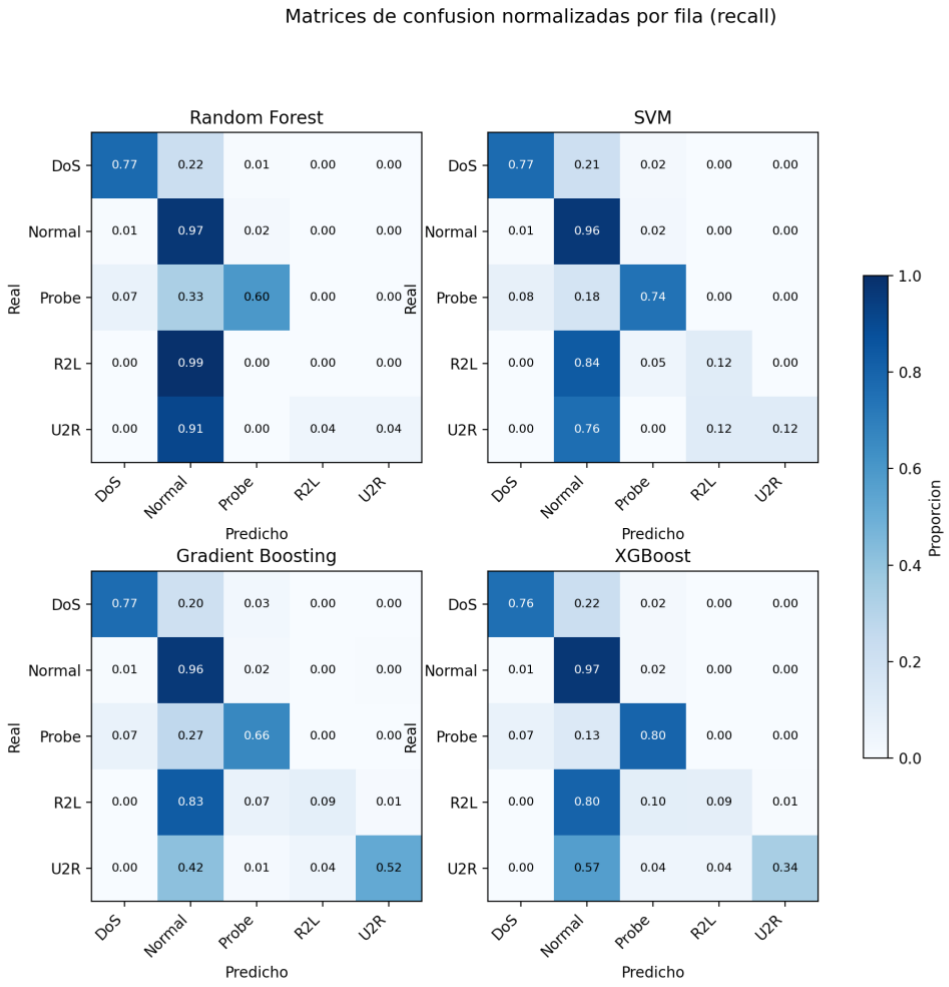

Figura 6.4 Matrices de confusión de cada modelo

Las matrices de confusión de la Figura 6.4 permiten extraer la conclusión más relevante del capítulo en términos de seguridad. El error dominante en los cuatro modelos no es confundir un tipo de ataque con otro, sino clasificar un ataque como tráfico Normal: las celdas más cargadas fuera de la diagonal son siempre las de la columna Normal. Es decir, los fallos del sistema son mayoritariamente falsos negativos, que en un IDS equivalen a intrusiones que pasan desapercibidas, precisamente el tipo de error que más conviene evitar. Esto explica también por qué la precisión de la clase Normal es comparativamente baja: una parte de lo que el sistema etiqueta como tráfico legítimo son en realidad ataques R2L no detectados.

6.4 IMPORTANCIA DE LAS CARACTERÍSTICAS

La importancia de cada variable en XGBoost se calcula a partir de la reducción media de impureza que aporta en las divisiones del árbol. La Figura 6.5 recoge las quince más importantes.

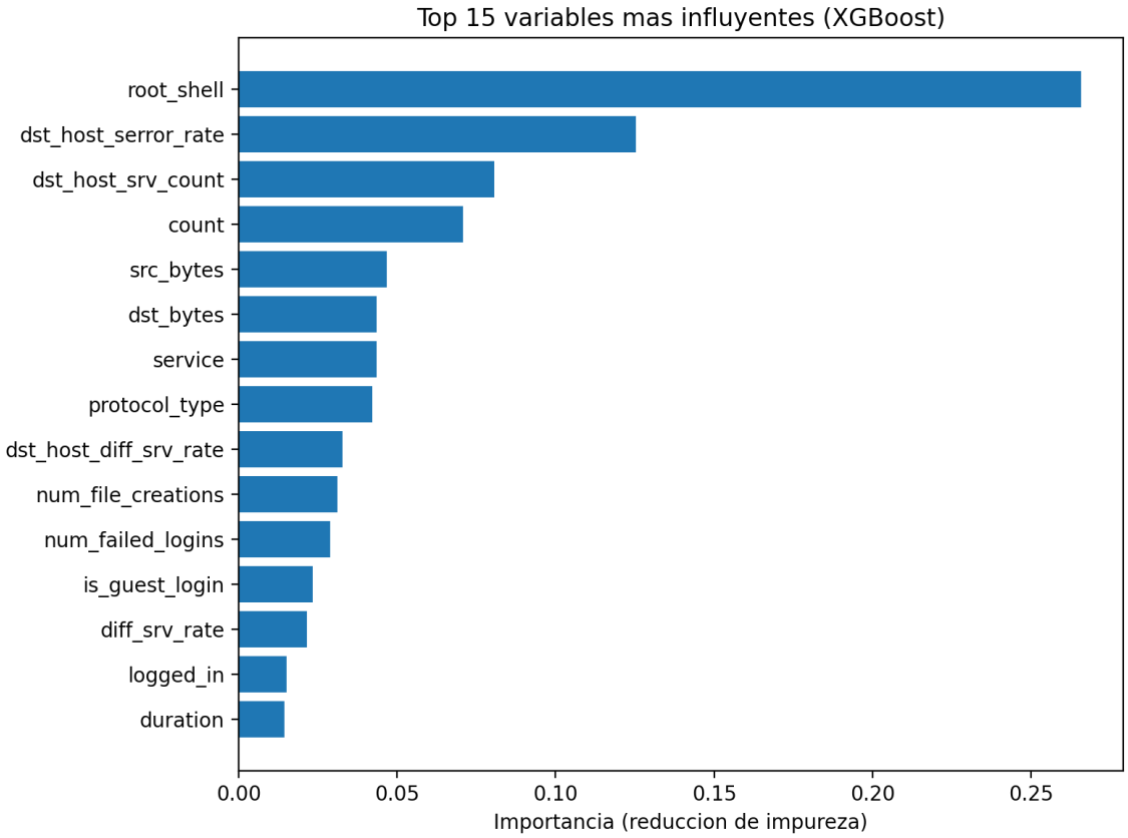

Figura 6.5 Importancia de las variables

La primera, `root_shell`, destaca con claridad sobre las demás: su importancia ronda 0,27, casi el doble que la siguiente. Es un atributo binario que vale 1 cuando la conexión consigue un terminal de `root`, con lo que funciona como un indicador casi directo de escalada de privilegios. Su peso en el modelo es consistente con la mejora observada en U2R tras el balanceo por clases: cuando esa señal está presente, el clasificador tiene un criterio muy fiable y lo usa.

A continuación, aparecen estadísticos de conexión: `dst_host_serror_rate`, `dst_host_srv_count` y `count` recogen tasas de error SYN y volúmenes de tráfico hacia un mismo host o servicio, que son precisamente los patrones que caracterizan los ataques DoS y los sondeos. Más abajo se sitúan descriptores de volumen y tipo de tráfico (`src_bytes`, `dst_bytes`, `service`, `protocol_type`) junto a atributos de contenido como `num_file_creations`, `num_failed_logins` e `is_guest_login`, todos ellos ligados a los comportamientos típicos de R2L y U2R.

Hay que leer este ranking con cuidado. La importancia por reducción de impureza sobrevalora las variables muy discriminantes en pocos casos, así que el peso de `root_shell` significa que el modelo se apoya en ella cuando aparece, no que esa única variable explique la mayor parte de las clasificaciones. También resulta llamativo que `flag`, un atributo habitual en trabajos sobre NSL-KDD, no aparezca entre los quince primeros: lo más probable es que su información esté capturada por otras variables con las que correlaciona.

6.5 CAPACIDAD DE DISCRIMINACIÓN Y UMBRAL DE DECISIÓN

Todas las métricas discutidas hasta ahora se calculan con el umbral de decisión por defecto: el modelo asigna cada conexión a la clase cuya probabilidad estimada es más alta. Las curvas precisión-recall permiten examinar qué sucede si ese umbral se mueve. La Figura 6.6 recoge las de XGBoost para Probe, R2L y U2R, las tres clases con mayor dificultad de detección.

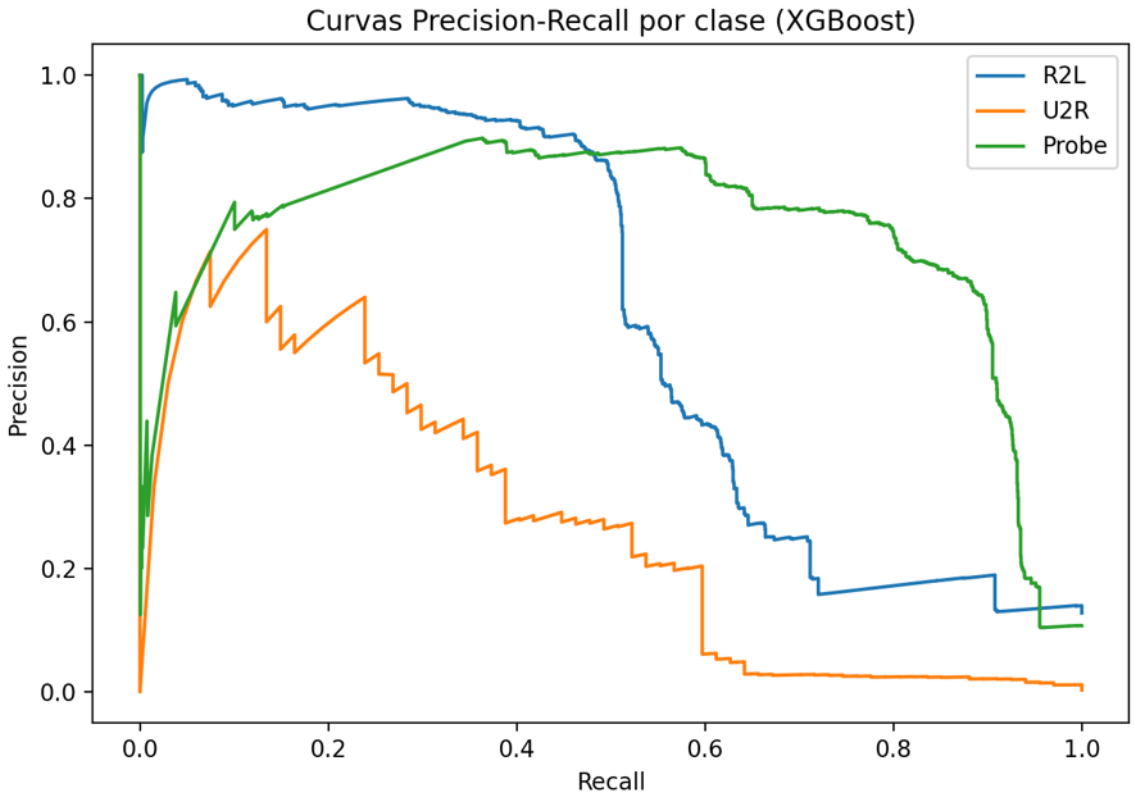

Figura 6.6 Curvas de precisión -Recall de XGBoost

Para Probe, el resultado es el esperado: la curva cae de forma suave y la precisión se mantiene por encima de 0,80 hasta un *recall* de ~0,6, lo que confirma que el modelo distingue bien este tipo de ataque en un amplio rango de configuraciones. El caso de R2L es más instructivo. Su *recall* en el punto de operación por defecto es de apenas 0,09, lo que podría hacer pensar que el modelo tiene serias dificultades con esta clase. La curva, sin embargo, cuenta otra historia: la precisión se mantiene cercana a 0,95 hasta un *recall* de alrededor de 0,45, y la degradación solo se produce a partir de ahí. El problema no es que el clasificador sea incapaz de detectar estas conexiones, sino que el umbral por defecto es demasiado conservador y descarta como tráfico normal muchas conexiones que el modelo considera, de hecho, sospechosas. Ajustando ese umbral se podría recuperar un porcentaje considerable de ataques R2L sin sacrificar apenas precisión. La curva de U2R, en cambio, es errática desde

el principio: con solo 67 muestras de prueba, la estimación es demasiado ruidosa para extraer conclusiones fiables sobre el comportamiento del modelo a distintos umbrales.

Este análisis apunta a una mejora concreta y abordable: la calibración del umbral de decisión de forma independiente por clase. Su implementación se plantea como trabajo futuro.

6.6 *COSTE COMPUTACIONAL*

Más allá de la calidad de la clasificación, una herramienta pensada para asistir al analista debe responder en tiempos razonables. La Figura 6.7 compara la latencia de inferencia de los cuatro modelos sobre el conjunto de prueba, medida en milisegundos por cada 1 000 conexiones clasificadas.

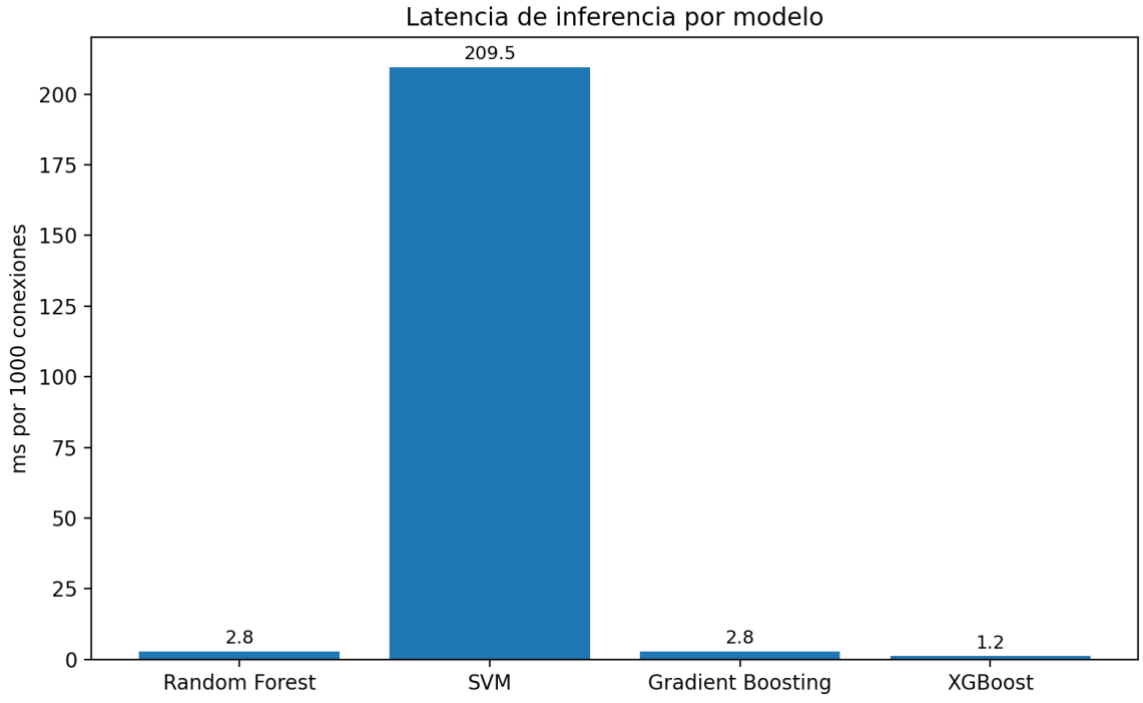

Figura 6.7 Latencia de inferencia por modelo

Las diferencias son muy marcadas: XGBoost es el más rápido, con 1,2 ms por cada 1 000 conexiones; Random Forest y Gradient Boosting se sitúan en 2,8 ms; y la SVM, con 209,5 ms, resulta unas 175 veces más lenta que XGBoost.

Este resultado refuerza la elección de XGBoost: no solo ofrece la mejor calidad de clasificación, sino también la inferencia más rápida, lo que lo hace ideal para analizar ficheros con muchos registros en la plataforma. La SVM, en cambio, queda penalizada por partida doble: es el modelo más costoso en inferencia y, además, el único que no pudo entrenarse con la totalidad de los datos. El coste de entrenamiento de una SVM con kernel RBF crece de forma cuadrática con el número de muestras, lo que obligó a entrenarla sobre una submuestra aleatoria estratificada de 20 000 registros. Sus resultados deben leerse teniendo en cuenta esta restricción.

6.7 DISCUSIÓN, LIMITACIONES Y VALIDEZ DE LOS RESULTADOS

De los cuatro modelos evaluados, XGBoost es el que mejor se adapta a los requisitos de NetRadar: obtiene la mayor *accuracy*, *el F1-macro* más alto y, además, es el más rápido en inferencia. Gradient Boosting merece mencionarse como alternativa cuando la prioridad es no dejar escapar ataques minoritarios; su *recall* de 0,52 en U2R es el mejor resultado obtenido en esa clase entre todos los modelos. Random Forest, en cambio, funciona bien en DoS y Normal pero no consigue recuperar R2L ni U2R de forma significativa.

Antes de dar por definitivos estos números conviene señalar tres aspectos del experimento que condicionan su interpretación.

El primero tiene que ver con la escasez de datos en las clases minoritarias. El balanceo por pesos mejora la detección de U2R, pero en R2L el problema es distinto: no es solo que haya pocos ejemplos en el entrenamiento, sino que el conjunto no recoge suficiente variedad de ese tipo de ataque como para que el modelo aprenda a generalizarlo.

El segundo es la composición de KDDTest+. Este conjunto de prueba incluye ataques que el modelo no ha visto durante el entrenamiento, lo que baja las métricas, pero da una idea más realista de cómo se comportaría el sistema ante tráfico desconocido. Que R2L sea tan difícil de detectar no sorprende en este contexto: es un problema habitual en detección de intrusiones con aprendizaje supervisado, donde los ataques sin firma conocida tienden a pasar inadvertidos.

El tercero afecta exclusivamente a la SVM. Este modelo se entrenó con una submuestra de 20 000 registros para evitar el coste cuadrático de su kernel, así que sus métricas no son directamente comparables con las del resto en igualdad de condiciones.

Teniendo en cuenta todo lo anterior, los resultados obtenidos se sitúan dentro de lo que cabe esperar para este conjunto de datos y permiten concluir que NetRadar detecta con solidez los ataques de mayor volumen, DoS y Probe, mientras que R2L sigue siendo el principal punto débil. Cómo abordarlo, junto con otras líneas de mejora del sistema, se discute en el Capítulo 7.

Capítulo 7. CONCLUSIONES Y TRABAJOS FUTUROS

7.1 CONCLUSIONES

Este Trabajo de Fin de Grado tenía un objetivo bastante concreto: construir NetRadar, una aplicación web que analizara tráfico de red y clasificara las intrusiones por familia de ataque, no con una simple etiqueta de normal o anómalo, sobre una arquitectura cliente-servidor pensada para un analista de seguridad real. Lo que sigue es una valoración de hasta dónde se ha llegado y de lo que no ha salido exactamente como estaba previsto.

El resultado en el plano del desarrollo es un prototipo funcional de extremo a extremo. Se ha construido todo el recorrido: desde el preprocesamiento del conjunto NSL-KDD, que deja los codificadores y el escalador ajustados en una única fase, hasta la interfaz web desde la que el analista carga ficheros y consulta resultados, pasando por el módulo de entrenamiento, el servidor FastAPI con su API REST y la capa de persistencia en SQLite. No es una arquitectura ideal sobre el papel; es la que realmente se ejecuta.

En el plano analítico, la comparativa de modelos ha sido más instructiva que un ranking de porcentajes. Lo primero que quedó claro es que la *accuracy* global es una métrica engañosa cuando las clases están tan desbalanceadas: dos modelos con cifras casi idénticas pueden comportarse de forma muy distinta ante los ataques minoritarios, y esa diferencia solo se ve con el *F1-macro* o el *recall* por clase. Lo segundo es que el balanceo por pesos tuvo un efecto real y medible: sin él, los modelos de boosting ignoraban por completo U2R; con él, empezaron a detectar una parte de esa categoría. Lo tercero, y lo más relevante desde el punto de vista de la seguridad, es que el sistema detecta bien los ataques de gran volumen como DoS y Probe, pero falla sistemáticamente con R2L, que confunde con tráfico normal. No es un problema particular de este proyecto: es una limitación conocida del aprendizaje supervisado en detección de intrusiones, y el análisis de las curvas *precisión-recall* ayudó a precisar que parte del problema está en el umbral de decisión, no en el modelo en sí.

Los objetivos planteados al inicio se han cumplido en su mayor parte. Quedan fuera la optimización sistemática de hiperparámetros, que se abordó con configuraciones razonadas, pero sin búsqueda exhaustiva, y algunas mejoras de la interfaz que quedaron pendientes por prioridad. El resto, preprocesamiento, entrenamiento comparativo, clasificación multiclase, backend, persistencia, frontend y evaluación con métricas estándar, está implementado y documentado.

Lo que este proyecto aporta no es una cifra de precisión especialmente alta; de hecho, se ha argumentado a lo largo de la memoria que las cifras moderadas son las honestas para este escenario. Lo que aporta es tener todo el ciclo de un sistema de detección integrado en una herramienta reproducible, evaluado con un criterio que evita la trampa habitual de partir el *dataset* de forma aleatoria. El valor está tanto en lo que se construye como en la lectura crítica de sus limitaciones.

A nivel personal, lo que más me ha costado y más me ha enseñado no ha sido implementar los modelos, sino mantener la coherencia entre lo que dice la memoria y lo que hace el código. Es fácil presentar una *accuracy* alta; es bastante más difícil entender qué ataques se están dejando pasar, por qué, y ser honesto al escribirlo.

7.2 TRABAJOS FUTUROS

NetRadar es un primer prototipo funcional, y como tal tiene límites claros. Las líneas de continuación que se describen en este apartado no son ideas genéricas: son las prolongaciones directas de lo que no ha funcionado del todo bien o de lo que se ha dejado fuera por alcance.

- **Validación sobre datos más recientes**: NSL-KDD tiene casi veinte años. Los modelos entrenados sobre él pueden no comportarse igual ante el tráfico y los ataques actuales. Evaluarlos sobre conjuntos como CIC-IDS2017 o UNSW-NB15, que recogen patrones modernos, permitiría saber si las conclusiones de este trabajo se sostienen en un escenario más realista o si son específicas del entorno de NSL-KDD.

- **Calibración del umbral y mejora del desbalanceo:** El análisis de las curvas *precisión-recall* dejó claro que R2L no se detecta bien principalmente porque el umbral por defecto es demasiado conservador, no porque el modelo sea incapaz de distinguirla. Ajustar ese umbral por clase sobre un conjunto de validación separado es una mejora concreta y de bajo coste. Aplicar SMOTE para generar ejemplos sintéticos de R2L y U2R complementaría esa calibración enriqueciendo un entrenamiento que ahora mismo es muy escaso en esas categorías.
- **Detección de lo desconocido:** El sistema falla ante ataques que no ha visto. Eso no es un problema fácil de resolver con más datos etiquetados, porque los ataques de día cero no aparecen en ningún conjunto de entrenamiento. Los *autoencoders* abordan el problema desde otro ángulo: aprenden qué es tráfico normal y señalan lo que no encaja. Integrar un módulo de este tipo añadiría una capa capaz de alertar sobre amenazas genuinamente nuevas, algo que la clasificación supervisada no puede ofrecer por diseño.
- **Análisis en tiempo real**: Ahora mismo el sistema trabaja sobre capturas exportadas en formato tabular. Leer directamente de la interfaz de red y clasificar en un flujo continuo de datos convertiría NetRadar en algo más cercano a un IDS operativo y menos a una herramienta forense.
- **Respuesta activa:** Detectar un ataque y limitarse a registrarlo tiene un valor operativo limitado. Integrar el sistema con un cortafuegos para bloquear automáticamente las conexiones maliciosas identificadas sería el paso hacia un IPS. Es un paso que exige mucha más fiabilidad: un falso positivo ya no sería una alerta errónea sino un corte de servicio legítimo.
- **Escalabilidad:** SQLite funciona bien para un prototipo con un solo usuario. Para un entorno con varios analistas trabajando en paralelo, la migración a PostgreSQL, la inferencia en procesos de trabajo separados y el empaquetado en contenedores son los cambios mínimos necesarios para que el sistema aguante en producción.
- **Seguridad de la plataforma:** El sistema almacena credenciales con *bcrypt*, pero el control de acceso sigue siendo básico. Antes de desplegarlo fuera de un entorno controlado habría que implementar sesiones con JWT, forzar HTTPS, restringir

CORS y añadir control de permisos por roles. Sin esto, la herramienta no debería exponerse a una red real.

- **Integración con el ecosistema:** Un IDS que no se comunica con nada más tiene un alcance limitado. Conectar NetRadar con una plataforma SIEM para que sus detecciones entren en un sistema centralizado de correlación de alertas aumentaría su utilidad operativa de forma considerable. Añadir explicabilidad mediante valores SHAP, que muestren al analista por qué una conexión concreta ha sido marcada, haría el sistema más transparente y fácil de operar en la práctica.

Capítulo 8. BIBLIOGRAFÍA

[1] Tavallaee, M., Bagheri, E., Lu, W., & Ghorbani, A. A. (2009). A detailed analysis of the KDD CUP 99 data set. 2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA), 1–6. https://doi.org/10.1109/CISDA.2009.5356528

[2] Revathi, S., & Malathi, A. (2013). A detailed analysis on NSL-KDD dataset using various machine learning techniques for intrusion detection. International Journal of Engineering Research & Technology (IJERT), 2(12), 1848–1853.

[3] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., et al. (2011). Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12, 2825–2830.

[4] Sommer, R., & Paxson, V. (2010). Outside the closed world: On using machine learning for network intrusion detection. 2010 IEEE Symposium on Security and Privacy, 305–316. https://doi.org/10.1109/SP.2010.25

[5] Check Point Research. (2023). 2023 Security Report. Check Point Software Technologies.

[6] Check Point Research. (2024). Q3 2024 cyber attack trends [informe trimestral]. Check Point Software Technologies. [Verificar el título exacto del informe del que se tomó la cifra de 1.876 ataques semanales].

[7] IBM Security. (2024). Cost of a Data Breach Report 2024. IBM Corporation.

[8] International Monetary Fund. (2024). Cyber risk: A growing concern for macrofinancial stability. En Global Financial Stability Report, April 2024 (cap. 3). International Monetary Fund.

[9] Breiman, L. (2001). Random forests. Machine Learning, 45(1), 5–32. https://doi.org/10.1023/A:1010933404324

[10] Cortes, C., & Vapnik, V. (1995). Support-vector networks. Machine Learning, 20(3), 273–297. https://doi.org/10.1007/BF00994018

[11] Platt, J. C. (1999). Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. En A. J. Smola, P. Bartlett, B. Schölkopf & D. Schuurmans (Eds.), Advances in Large Margin Classifiers (pp. 61–74). MIT Press.

[12] Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. The Annals of Statistics, 29(5), 1189–1232. https://doi.org/10.1214/aos/1013203451

[13] Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 785–794. https://doi.org/10.1145/2939672.2939785

[14] Buczak, A. L., & 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. https://doi.org/10.1109/COMST.2015.2494502

[15] Netskope Threat Labs. (2024). Cloud and Threat Report. Netskope.

ANEXO I: ALINEACIÓN DEL PROYECTO CON LOS ODS

NetRadar no responde únicamente a un objetivo tecnológico. Su desarrollo parte de la premisa de que garantizar la seguridad de las comunicaciones es una condición necesaria para el funcionamiento de la sociedad digitalizada. Por eso, este TFG toma como referencia los Objetivos de Desarrollo Sostenible (ODS) de la Agenda 2030 de Naciones Unidas y considera su alineación con ellos una parte explícita del trabajo.

ODS 9 - Industria, Innovación e Infraestructura: La integración de modelos predictivos de *Machine Learning* para la detección temprana de intrusiones responde al objetivo de desarrollar infraestructuras tecnológicas resilientes. Aplicar inteligencia artificial a la ciberseguridad no es solo una cuestión de rendimiento: amplía el acceso a capacidades de análisis avanzado en redes industriales e institucionales que difícilmente podrían sostener esas capacidades de otro modo.

ODS 11 - Ciudades y comunidades sostenibles: La sostenibilidad urbana depende cada vez más de que las infraestructuras digitales sean seguras. Los servicios interconectados y los entornos de *Smart City* multiplican la superficie de exposición a amenazas. Al desarrollar un sistema orientado a mitigar vulnerabilidades y proteger el tráfico de red, este trabajo contribuye a que las comunidades puedan operar en el ciberespacio sin comprometer la privacidad ni la integridad de sus usuarios.

ODS 16 - Paz, justicia e instituciones sólidas: La capacidad de las instituciones para cumplir sus funciones depende, en parte, de su seguridad digital. NetRadar ofrece una plataforma accesible de análisis forense y protección frente a ataques como la denegación de servicio o el *probing*, lo que reduce la exposición al cibercrimen y contribuye a mantener la confianza de la ciudadanía en entornos institucionales.

En conjunto, este proyecto parte de la convicción de que la ingeniería de telecomunicaciones tiene un papel concreto en la construcción de entornos más seguros. Esa contribución no se

agota en los resultados técnicos del sistema: se extiende a las decisiones de diseño que lo orientan desde el principio.

ANEXO II

entrenamiento.py:

```
import os
import joblib
import pandas as pd
from sklearn.ensemble import RandomForestClassifier,
GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.utils.class_weight import compute_sample_weight
from sklearn.metrics import (
    accuracy_score, f1_score, classification_report, confusion_matrix,
)

try:
    from xgboost import XGBClassifier
    HAY_XGBOOST = True
except ImportError:
    from sklearn.ensemble import HistGradientBoostingClassifier
    HAY_XGBOOST = False

# Tope de muestras para la SVM: con kernel RBF su coste crece de forma
cuadratica
# con el numero de filas, asi que se entrena sobre una submuestra
ALEATORIA y
# ESTRATIFICADA (mantiene la proporcion de cada clase). Es una limitacion
conocida.
MAX_SVM = 20000


def cargar_datos(ruta_train, ruta_test, ruta_encoders):
    artefactos = joblib.load(ruta_encoders)
    scaler = artefactos["scaler"]
    feature_cols = artefactos["feature_cols"]
    le_clases = artefactos["le_clases"]

    df_train = pd.read_csv(ruta_train)
    df_test = pd.read_csv(ruta_test)
```

```
    # Las caracteristicas ya vienen codificadas; solo se aplica el
escalador guardado.
    X_train = scaler.transform(df_train[feature_cols].values)
    X_test = scaler.transform(df_test[feature_cols].values)
    y_train = le_clases.transform(df_train["class"])
    y_test = le_clases.transform(df_test["class"])
    return X_train, y_train, X_test, y_test, le_clases


def evaluar(nombre, modelo, X_test, y_test, clases):
    pred = modelo.predict(X_test)
    acc = accuracy_score(y_test, pred) * 100
    f1 = f1_score(y_test, pred, average="macro") * 100
    print(f"\n=== {nombre} ===")
    print(f"Accuracy: {acc:.2f}%   F1-macro: {f1:.2f}%")
    print(classification_report(y_test, pred, target_names=clases,
zero_division=0))
    print("Matriz de confusion (filas=real, columnas=predicho):")
    print(pd.DataFrame(confusion_matrix(y_test, pred), index=clases,
columns=clases).to_string())
    return acc, f1


def entrenar_sistema():
    base = os.path.dirname(__file__)
    datos = os.path.join(base, "..", "datos")
    modelos = os.path.join(base, "..", "modelos")
    ruta_encoders = os.path.join(modelos, "encoders.pkl")

    if not os.path.exists(ruta_encoders):
        raise FileNotFoundError("Falta encoders.pkl. Ejecuta
preprocesamiento.py primero.")

    X_train, y_train, X_test, y_test, le_clases = cargar_datos(
        os.path.join(datos, "kdd_train_limpio.csv"),
        os.path.join(datos, "kdd_test_limpio.csv"),
        ruta_encoders,
    )
    clases = list(le_clases.classes_)
    # Pesos por muestra para los modelos sin parametro class_weight (GB y
XGBoost).
    pesos = compute_sample_weight("balanced", y_train)
    resumen = {}
```

```
    # 1. Random Forest
    print("\nEntrenando Random Forest...")
    rf = RandomForestClassifier(
        n_estimators=100, class_weight="balanced", random_state=42,
n_jobs=-1
    )
    rf.fit(X_train, y_train)
    joblib.dump(rf, os.path.join(modelos, "randomforest.pkl"))
    resumen["Random Forest"] = evaluar("Random Forest", rf, X_test,
y_test, clases)

    # 2. SVM (submuestra aleatoria estratificada)
    print("\nEntrenando SVM...")
    if len(X_train) > MAX_SVM:
        X_svm, _, y_svm, _ = train_test_split(
            X_train, y_train, train_size=MAX_SVM, stratify=y_train,
random_state=42
        )
        print(f"  SVM entrenada con {MAX_SVM:,} de {len(X_train):,}
muestras (estratificadas).")
    else:
        X_svm, y_svm = X_train, y_train
    svm = SVC(kernel="rbf", C=1.0, probability=True,
class_weight="balanced", random_state=42)
    svm.fit(X_svm, y_svm)
    joblib.dump(svm, os.path.join(modelos, "svm.pkl"))
    resumen["SVM"] = evaluar("SVM", svm, X_test, y_test, clases)

    # 3. Gradient Boosting (balanceo mediante sample_weight)
    print("\nEntrenando Gradient Boosting...")
    gb = GradientBoostingClassifier(n_estimators=50, learning_rate=0.1,
random_state=42)
    gb.fit(X_train, y_train, sample_weight=pesos)
    joblib.dump(gb, os.path.join(modelos, "gradientboost.pkl"))
    resumen["Gradient Boosting"] = evaluar("Gradient Boosting", gb,
X_test, y_test, clases)

    # 4. XGBoost (balanceo mediante sample_weight; respaldo si la libreria
no esta)
    print("\nEntrenando XGBoost...")
    if HAY_XGBOOST:
```

```
        xgb = XGBClassifier(n_estimators=50, learning_rate=0.1,
random_state=42)
    else:
        print("  xgboost no disponible: se usa
HistGradientBoostingClassifier como respaldo.")
        xgb = HistGradientBoostingClassifier(max_iter=50,
learning_rate=0.1, random_state=42)
    xgb.fit(X_train, y_train, sample_weight=pesos)
    joblib.dump(xgb, os.path.join(modelos, "xgboost.pkl"))
    resumen["XGBoost"] = evaluar("XGBoost", xgb, X_test, y_test, clases)

    print("\n\nRESUMEN (test)")
    print(f"{'Modelo':<20}{'Accuracy':>12}{'F1-macro':>12}")
    for nombre, (acc, f1) in resumen.items():
        print(f"{nombre:<20}{acc:>11.2f}%{f1:>11.2f}%")
    print("\nEntrenamiento finalizado. Ya puedes arrancar el servidor.")


if __name__ == "__main__":
    entrenar_sistema()
```

Preprocesamiento.py

```
import os
import joblib
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder, StandardScaler

# Columnas estandar del formato NSL-KDD (41 caracteristicas + etiqueta +
dificultad).
COLUMNAS = [
    "duration", "protocol_type", "service", "flag",
    "src_bytes", "dst_bytes", "land", "wrong_fragment", "urgent",
    "hot", "num_failed_logins", "logged_in", "num_compromised",
    "root_shell", "su_attempted", "num_root", "num_file_creations",
    "num_shells", "num_access_files", "num_outbound_cmds",
    "is_host_login", "is_guest_login", "count", "srv_count",
    "serror_rate", "srv_serror_rate", "rerror_rate", "srv_rerror_rate",
    "same_srv_rate", "diff_srv_rate", "srv_diff_host_rate",
    "dst_host_count", "dst_host_srv_count",
    "dst_host_same_srv_rate", "dst_host_diff_srv_rate",
```

```
    "dst_host_same_src_port_rate", "dst_host_srv_diff_host_rate",
    "dst_host_serror_rate", "dst_host_srv_serror_rate",
    "dst_host_rerror_rate", "dst_host_srv_rerror_rate",
    "attack_type", "difficulty_level",
]

# Cada ataque concreto del dataset se agrupa en una de las 5 familias.
MAPEO_FAMILIAS = {
    "normal": "Normal",
    # DoS
    "back": "DoS", "land": "DoS", "neptune": "DoS", "pod": "DoS",
    "smurf": "DoS", "teardrop": "DoS", "apache2": "DoS", "udpstorm":
"DoS",
    "processtable": "DoS", "worm": "DoS", "mailbomb": "DoS",
    # Probe
    "ipsweep": "Probe", "nmap": "Probe", "portsweep": "Probe",
    "satan": "Probe", "mscan": "Probe", "saint": "Probe",
    # R2L
    "ftp_write": "R2L", "guess_passwd": "R2L", "imap": "R2L", "multihop":
"R2L",
    "phf": "R2L", "spy": "R2L", "warezclient": "R2L", "warezmaster":
"R2L",
    "sendmail": "R2L", "named": "R2L", "snmpgetattack": "R2L",
"snmpguess": "R2L",
    "xlock": "R2L", "xsnoop": "R2L", "httptunnel": "R2L",
    # U2R
    "buffer_overflow": "U2R", "loadmodule": "U2R", "perl": "U2R",
    "rootkit": "U2R", "sqlattack": "U2R", "xterm": "U2R", "ps": "U2R",
}

COLS_CATEGORICAS = ["protocol_type", "service", "flag"]


def cargar_txt(ruta):
    """Lee un .txt de NSL-KDD (separado por comas y sin cabecera)."""
    print(f"  Leyendo {ruta}")
    df = pd.read_csv(ruta, header=None, names=COLUMNAS, na_values=["?"])
    print(f"    -> {len(df):,} filas")
    return df


def mapear_familias(df, nombre):
    """Normaliza la etiqueta de ataque y la convierte en su familia
(columna 'class')."""
```

```
    df["class"] = (
        df["attack_type"].astype(str).str.strip().str.strip("'\"").str.low
er()
        .str.rstrip(".").map(MAPEO_FAMILIAS)
    )
    antes = len(df)
    df = df[df["class"].notna()].copy()
    if len(df) < antes:
        print(f"  [{nombre}] {antes - len(df)} filas eliminadas (ataques
no reconocidos)")
    return df.drop(columns=["attack_type", "difficulty_level"])


def limpiar(df, nombre):
    """Convierte tipos, elimina duplicados e imputa nulos."""
    num_cols = [c for c in df.columns if c not in COLS_CATEGORICAS +
["class"]]
    df[num_cols] = df[num_cols].apply(pd.to_numeric, errors="coerce")

    antes = len(df)
    df = df.drop_duplicates()
    if len(df) < antes:
        print(f"  [{nombre}] {antes - len(df):,} duplicados eliminados")

    if df.isnull().values.any():
        df[num_cols] = df[num_cols].fillna(df[num_cols].median())
        for col in COLS_CATEGORICAS:
            df[col] = df[col].fillna(df[col].mode()[0])
        print(f"  [{nombre}] nulos imputados")
    return df


def preprocesar(ruta_train, ruta_test, salida_train, salida_test,
ruta_encoders):
    print("Preprocesamiento NSL-KDD")

    # 1. Cargar y preparar ambos conjuntos.
    df_train = limpiar(mapear_familias(cargar_txt(ruta_train), "TRAIN"),
"TRAIN")
    df_test = limpiar(mapear_familias(cargar_txt(ruta_test), "TEST"),
"TEST")

    # 2. Codificadores categoricos: se ajustan con train+test para conocer
todos
```

```
    #    los valores (p. ej. todos los servicios) y evitar categorias no
vistas.
    encoders_categoricos = {}
    for col in COLS_CATEGORICAS:
        le = LabelEncoder()
        le.fit(pd.concat([df_train[col], df_test[col]]).astype(str))
        df_train[col] = le.transform(df_train[col].astype(str))
        df_test[col] = le.transform(df_test[col].astype(str))
        encoders_categoricos[col] = le
        print(f"  '{col}' -> {len(le.classes_)} categorias")

    # 3. Codificador de la clase objetivo (Normal, DoS, Probe, R2L, U2R).
    le_clases = LabelEncoder()
    le_clases.fit(pd.concat([df_train["class"], df_test["class"]]))

    # 4. Reordenar: caracteristicas primero, 'class' al final.
    feature_cols = [c for c in df_train.columns if c != "class"]
    df_train = df_train[feature_cols + ["class"]]
    df_test = df_test[feature_cols + ["class"]]

    # 5. Escalador: se ajusta SOLO con el entrenamiento para no filtrar el
test.
    scaler = StandardScaler()
    scaler.fit(df_train[feature_cols].values)

    # 6. Guardar transformadores y datasets limpios.
    os.makedirs(os.path.dirname(ruta_encoders), exist_ok=True)
    joblib.dump({
        "encoders_categoricos": encoders_categoricos,
        "le_clases": le_clases,
        "scaler": scaler,
        "feature_cols": feature_cols,
    }, ruta_encoders)

    df_train.to_csv(salida_train, index=False)
    df_test.to_csv(salida_test, index=False)

    print(f"\nGuardado: {salida_train} ({df_train.shape[0]:,} x
{df_train.shape[1]})")
    print(f"Guardado: {salida_test} ({df_test.shape[0]:,} x
{df_test.shape[1]})")
    print(f"Guardado: {ruta_encoders}")
    print("\nDistribucion de clases (TRAIN):")
```

```
    print(df_train["class"].value_counts().to_string())


if __name__ == "__main__":
    base = os.path.dirname(__file__)
    datos = os.path.join(base, "..", "datos")
    modelos = os.path.join(base, "..", "modelos")
    preprocesar(
        ruta_train=os.path.join(datos, "KDDTrain.txt"),
        ruta_test=os.path.join(datos, "KDDTest.txt"),
        salida_train=os.path.join(datos, "kdd_train_limpio.csv"),
        salida_test=os.path.join(datos, "kdd_test_limpio.csv"),
        ruta_encoders=os.path.join(modelos, "encoders.pkl"),
    )
```

servidor.py

```
import os
import io
import json
from datetime import datetime

import bcrypt
import joblib
import numpy as np
import pandas as pd
from fastapi import FastAPI, File, UploadFile, HTTPException, Query
from fastapi.middleware.cors import CORSMiddleware
from fastapi.responses import JSONResponse
from pydantic import BaseModel
from sqlalchemy import create_engine, Column, Integer, String, DateTime,
Text, ForeignKey
from sqlalchemy.orm import DeclarativeBase, Session, relationship

base_dir = os.path.dirname(__file__)
ruta_db = os.path.join(base_dir, "..", "historial.db")
folder_modelos = os.path.join(base_dir, "..", "modelos")

# --- Base de datos (SQLite + SQLAlchemy) ---
engine = create_engine(f"sqlite:///{ruta_db}", echo=False)
```

```
class Base(DeclarativeBase):
    pass


class Usuario(Base):
    __tablename__ = "usuarios"
    id = Column(Integer, primary_key=True, autoincrement=True)
    username = Column(String(100), unique=True, nullable=False)
    password = Column(String(255), nullable=False)              # hash
bcrypt
    nombre = Column(String(100), nullable=False)
    pregunta_seguridad = Column(String(255), nullable=False)
    respuesta_seguridad = Column(String(255), nullable=False)   # hash
bcrypt
    analisis = relationship("Analisis", back_populates="usuario")


class Analisis(Base):
    __tablename__ = "analisis"
    id = Column(Integer, primary_key=True, autoincrement=True)
    usuario_id = Column(Integer, ForeignKey("usuarios.id"),
nullable=False)
    nombre_archivo = Column(String(255))
    fecha_analisis = Column(DateTime, default=datetime.utcnow)
    total_registros = Column(Integer)
    detalles_modelos_json = Column(Text)
    usuario = relationship("Usuario", back_populates="analisis")


Base.metadata.create_all(engine)

# --- Transformadores y modelos (se cargan una vez al arrancar) ---
ruta_encoders = os.path.join(folder_modelos, "encoders.pkl")
if not os.path.exists(ruta_encoders):
    raise RuntimeError("Falta encoders.pkl. Ejecuta preprocesamiento.py y
entrenamiento.py.")

artefactos = joblib.load(ruta_encoders)
encoders_categoricos = artefactos["encoders_categoricos"]
le_clases = artefactos["le_clases"]
scaler = artefactos["scaler"]
feature_cols = artefactos["feature_cols"]
```

```
columnas_texto = list(encoders_categoricos.keys())  # protocol_type,
service, flag

nombres_modelos = {
    "randomforest": "randomforest.pkl",
    "svm": "svm.pkl",
    "gradientboost": "gradientboost.pkl",
    "xgboost": "xgboost.pkl",
}
modelos_disponibles = {
    clave: joblib.load(os.path.join(folder_modelos, fichero))
    for clave, fichero in nombres_modelos.items()
    if os.path.exists(os.path.join(folder_modelos, fichero))
}

# --- App ---
app = FastAPI(title="NetRadar API")
app.add_middleware(CORSMiddleware, allow_origins=["*"],
allow_methods=["*"], allow_headers=["*"])


class UserRegister(BaseModel):
    username: str
    password: str
    nombre: str
    pregunta_seguridad: str
    respuesta_seguridad: str


class UserLogin(BaseModel):
    username: str
    password: str


class UserRecover(BaseModel):
    username: str
    pregunta_seguridad: str
    respuesta_seguridad: str
    nueva_password: str


# --- Utilidades ---
def hashear(texto):
```

```
    return bcrypt.hashpw(texto.encode("utf-8"),
bcrypt.gensalt()).decode("utf-8")


def verificar(texto, hash_guardado):
    return bcrypt.checkpw(texto.encode("utf-8"),
hash_guardado.encode("utf-8"))


def normalizar_respuesta(texto):
    return texto.strip().lower()


def preparar_datos(df):
    """Aplica a un CSV crudo la misma transformacion que el entrenamiento:
    codifica las categoricas con los encoders guardados, ordena las
columnas y
    aplica el escalador. Los valores categoricos no vistos se mapean a
0."""
    df = df.copy()
    for col in feature_cols:
        if col not in df.columns:
            df[col] = 0
    for col in feature_cols:
        if col in columnas_texto:
            le = encoders_categoricos[col]
            conocidos = set(le.classes_)
            df[col] = df[col].astype(str).str.strip().apply(
                lambda v: int(le.transform([v])[0]) if v in conocidos else
0
            )
        else:
            df[col] = pd.to_numeric(df[col], errors="coerce").fillna(0)
    return scaler.transform(df[feature_cols].values)


# --- Autenticacion ---
@app.post("/api/auth/register")
def registrar_usuario(datos: UserRegister):
    with Session(engine) as session:
        if session.query(Usuario).filter(Usuario.username ==
datos.username).first():
            raise HTTPException(status_code=400, detail="Usuario ya
existe")
```

```
        nuevo = Usuario(
            username=datos.username,
            password=hashear(datos.password),
            nombre=datos.nombre,
            pregunta_seguridad=datos.pregunta_seguridad,
            respuesta_seguridad=hashear(normalizar_respuesta(datos.respues
ta_seguridad)),
        )
        session.add(nuevo)
        session.commit()
        return {"status": "success"}


@app.post("/api/auth/login")
def login_usuario(datos: UserLogin):
    with Session(engine) as session:
        user = session.query(Usuario).filter(Usuario.username ==
datos.username).first()
        if not user or not verificar(datos.password, user.password):
            raise HTTPException(status_code=401, detail="Credenciales
incorrectas")
        return {"status": "success", "user_id": user.id, "nombre":
user.nombre}


@app.post("/api/auth/recover")
def recuperar_password(datos: UserRecover):
    with Session(engine) as session:
        user = session.query(Usuario).filter(Usuario.username ==
datos.username).first()
        if not user:
            raise HTTPException(status_code=404, detail="Usuario no
encontrado")
        respuesta_ok =
verificar(normalizar_respuesta(datos.respuesta_seguridad),
user.respuesta_seguridad)
        if user.pregunta_seguridad != datos.pregunta_seguridad or not
respuesta_ok:
            raise HTTPException(status_code=401, detail="Pregunta o
respuesta incorrecta")
        user.password = hashear(datos.nueva_password)
        session.commit()
        return {"status": "success"}
```

```
# --- Analisis ---
@app.post("/api/analizar")
async def analizar(usuario_id: int, modelos: str = Query(...), archivo:
UploadFile = File(...)):
    try:
        contenido = await archivo.read()
        df = pd.read_csv(io.BytesIO(contenido), sep=None, engine="python")
        df.dropna(how="all", inplace=True)
    except Exception as e:
        raise HTTPException(status_code=400, detail="Error leyendo CSV: "
+ str(e))

    df_modelo = df.drop(columns=["class", "label"], errors="ignore")
    X = preparar_datos(df_modelo)
    solicitados = [m.strip() for m in modelos.split(",") if m.strip()]

    # Muestra de hasta 100 filas para la tabla forense (las metricas usan
todo el fichero).
    muestra = [{
        "id": i + 1,
        "protocolo": str(df_modelo.iloc[i].get("protocol_type", "-")),
        "servicio": str(df_modelo.iloc[i].get("service", "-")),
        "src_bytes": int(df_modelo.iloc[i].get("src_bytes", 0)),
    } for i in range(min(100, len(df_modelo)))]

    resultado = {}
    for nombre in solicitados:
        if nombre not in modelos_disponibles:
            continue
        clf = modelos_disponibles[nombre]
        pred_texto = le_clases.inverse_transform(clf.predict(X))
        confianzas = np.max(clf.predict_proba(X), axis=1) * 100

        desglose = {c: 0 for c in ["Normal", "DoS", "Probe", "R2L",
"U2R"]}
        for p in pred_texto:
            desglose[p] += 1

        total = len(pred_texto)
        ataques = total - desglose["Normal"]
        riesgo = round(ataques / total * 100, 2) if total else 0
```

```
        tabla = []
        for i, fila in enumerate(muestra):
            f = dict(fila)
            f["prediccion"] = pred_texto[i]
            f["confianza"] = round(float(confianzas[i]), 1)
            tabla.append(f)

        resultado[nombre] = {
            "kpis": {
                "total": total, "normales": desglose["Normal"], "ataques":
ataques,
                "riesgo_pct": riesgo, "desglose": desglose,
            },
            "tabla": tabla,
        }

    with Session(engine) as session:
        session.add(Analisis(
            usuario_id=usuario_id,
            nombre_archivo=archivo.filename,
            total_registros=len(df),
            detalles_modelos_json=json.dumps({k: v["kpis"] for k, v in
resultado.items()}),
        ))
        session.commit()

    return JSONResponse(content={
        "nombre_archivo": archivo.filename,
        "fecha": datetime.utcnow().isoformat(),
        "total_registros": len(df),
        "datos_modelos": resultado,
    })


# --- Panel e historial ---
@app.get("/api/dashboard/stats")
def stats_dashboard(usuario_id: int):
    with Session(engine) as session:
        hist = session.query(Analisis).filter(Analisis.usuario_id ==
usuario_id).all()
        if not hist:
            return {"tiene_datos": False}
        ultimo = hist[-1]
        return {
```

```
            "tiene_datos": True,
            "total_archivos": len(hist),
            "total_registros": sum(a.total_registros for a in hist),
            "ultimo_archivo": ultimo.nombre_archivo,
            "ultimo_detalles": json.loads(ultimo.detalles_modelos_json) if
ultimo.detalles_modelos_json else {},
        }


@app.get("/api/historial")
def consultar_historial(usuario_id: int, limite: int = 15):
    with Session(engine) as session:
        entradas = (
            session.query(Analisis)
            .filter(Analisis.usuario_id == usuario_id)
            .order_by(Analisis.id.desc())
            .limit(limite)
            .all()
        )
        return JSONResponse(content=[{
            "id": e.id,
            "nombre_archivo": e.nombre_archivo,
            "fecha_analisis": e.fecha_analisis.isoformat() if
e.fecha_analisis else None,
            "total_registros": e.total_registros,
            "comparativa": json.loads(e.detalles_modelos_json) if
e.detalles_modelos_json else {},
        } for e in entradas])
```

App.js

```
const API_BASE = "http://localhost:8000";
let CONFIG_APP = { currentUser: null, currentName: "", archivoActual: null
};
let RESPUESTA_GLOBAL = {};
let chartsInstances = {};

const DOM = {
    authWrapper: document.getElementById("auth-wrapper"),
    appWrapper: document.getElementById("app-wrapper"),
    viewLogin: document.getElementById("auth-login-view"),
```

```
    viewRegister: document.getElementById("auth-register-view"),
    viewRecover: document.getElementById("auth-recover-view"),

    inLoginUser: document.getElementById("login-username"),
    inLoginPass: document.getElementById("login-password"),
    btnDoLogin: document.getElementById("btn-do-login"),

    inRegNombre: document.getElementById("reg-nombre"),
    inRegUser: document.getElementById("reg-username"),
    inRegPass: document.getElementById("reg-password"),
    inRegPregunta: document.getElementById("reg-pregunta"),
    inRegRespuesta: document.getElementById("reg-respuesta"),
    btnDoRegister: document.getElementById("btn-do-register"),

    inRecUser: document.getElementById("rec-username"),
    inRecPregunta: document.getElementById("rec-pregunta"),
    inRecRespuesta: document.getElementById("rec-respuesta"),
    inRecNewPass: document.getElementById("rec-newpassword"),
    btnDoRecover: document.getElementById("btn-do-recover"),

    lnkGoRegister: document.getElementById("link-go-register"),
    lnkGoRecover: document.getElementById("link-go-recover"),
    linksBackLogin: document.querySelectorAll(".link-back-login"),

    sidebar: document.getElementById("sidebar"),
    mainContent: document.getElementById("main-content"),
    btnToggleSidebar: document.getElementById("btn-toggle-sidebar"),
    btnLogout: document.getElementById("btn-logout"),
    userDisplayName: document.getElementById("user-display-name"),

    welcomeMsg: document.getElementById("welcome-message"),
    dashNoData: document.getElementById("dashboard-no-data"),
    dashDataView: document.getElementById("dashboard-data-view"),
    dashTotalFiles: document.getElementById("dash-total-files"),
    dashTotalRecords: document.getElementById("dash-total-records"),
    dashLastFile: document.getElementById("dash-last-file"),
    dashCompCards: document.getElementById("dash-comparative-cards"),

    dropZone: document.getElementById("drop-zone"),
    fileInput: document.getElementById("file-input"),
    fileNameDisplay: document.getElementById("file-name-display"),
    btnAnalizar: document.getElementById("btn-analizar"),
    loader: document.getElementById("loader"),
```

```
    resultados: document.getElementById("resultados"),
    compModelosDiv: document.getElementById("contenedor-comparativo-
modelos"),
    panelDetalle: document.getElementById("panel-detalle-modelo"),
    detalleTitulo: document.getElementById("detalle-titulo"),
    tablaBody: document.getElementById("tabla-body"),
    btnExportar: document.getElementById("btn-exportar-csv"),

    historialContent: document.getElementById("historial-content")
};

/* ---------- HELPERS DE ERRORES INLINE ---------- */
function showError(elId, msg) {
    const el = document.getElementById(elId);
    if (!el) return;
    el.textContent = msg;
    el.classList.remove("hidden");
}
function clearError(elId) {
    const el = document.getElementById(elId);
    if (el) el.classList.add("hidden");
}

/* ---------- NAVEGACIÓN Y HAMBURGUESA ---------- */
DOM.btnToggleSidebar.onclick = () => {
    DOM.sidebar.classList.toggle("collapsed");
    DOM.mainContent.classList.toggle("expanded");
};

document.querySelectorAll(".nav-btn").forEach(btn => {
    btn.onclick = () => {
        document.querySelectorAll(".nav-btn").forEach(b =>
b.classList.remove("active"));
        document.querySelectorAll(".tab-panel").forEach(p =>
p.classList.add("hidden"));
        btn.classList.add("active");
        document.getElementById(`tab-
${btn.dataset.tab}`).classList.remove("hidden");

        if (btn.dataset.tab === "dashboard") cargarDatosDashboard();
        if (btn.dataset.tab === "historial") cargarHistorial();
    };
```

```
});

/* ---------- VISTAS AUTH ---------- */
DOM.lnkGoRegister.onclick = (e) => {
    e.preventDefault();
    ocultarAuth();
    DOM.viewRegister.classList.remove("hidden");
    DOM.inRegNombre.focus();
};
DOM.lnkGoRecover.onclick = (e) => {
    e.preventDefault();
    ocultarAuth();
    DOM.viewRecover.classList.remove("hidden");
    DOM.inRecUser.focus();
};
DOM.linksBackLogin.forEach(btn => btn.onclick = (e) => {
    e.preventDefault();
    ocultarAuth();
    DOM.viewLogin.classList.remove("hidden");
    DOM.inLoginUser.focus();
});

function ocultarAuth() {
    DOM.viewLogin.classList.add("hidden");
    DOM.viewRegister.classList.add("hidden");
    DOM.viewRecover.classList.add("hidden");
    clearError("login-error");
    clearError("register-error");
    clearError("recover-error");
}

/* ---------- ENTER EN FORMULARIOS ---------- */
// Login: Enter en usuario o contraseña dispara el login
[DOM.inLoginUser, DOM.inLoginPass].forEach(el => {
    el.addEventListener("keydown", (e) => { if (e.key === "Enter")
DOM.btnDoLogin.click(); });
});

// Registro: Enter en el último campo dispara el registro
DOM.inRegRespuesta.addEventListener("keydown", (e) => { if (e.key ===
"Enter") DOM.btnDoRegister.click(); });

// Recuperación: Enter en el último campo dispara la recuperación
```

```
DOM.inRecNewPass.addEventListener("keydown", (e) => { if (e.key ===
"Enter") DOM.btnDoRecover.click(); });

/* ---------- LÓGICA AUTH ---------- */
DOM.btnDoRegister.onclick = async () => {
    clearError("register-error");
    const payload = {
        username: DOM.inRegUser.value.trim(),
        password: DOM.inRegPass.value,
        nombre: DOM.inRegNombre.value.trim(),
        pregunta_seguridad: DOM.inRegPregunta.value,
        respuesta_seguridad: DOM.inRegRespuesta.value
    };
    if (!payload.username || !payload.password || !payload.nombre) {
        showError("register-error", "Por favor, rellena todos los campos
obligatorios.");
        return;
    }
    try {
        DOM.btnDoRegister.disabled = true;
        DOM.btnDoRegister.textContent = "Registrando…";
        const res = await fetch(`${API_BASE}/api/auth/register`, {
            method: "POST", headers: { "Content-Type": "application/json"
}, body: JSON.stringify(payload)
        });
        const data = await res.json();
        if (!res.ok) throw new Error(data.detail || "Error al registrar");
        ocultarAuth();
        DOM.viewLogin.classList.remove("hidden");
        DOM.inLoginUser.value = payload.username;
        DOM.inLoginPass.focus();
        showError("login-error", ""); // limpio por si había algo
        // Mensaje positivo reutilizando la caja de error con color verde
        const loginErr = document.getElementById("login-error");
        if (loginErr) {
            loginErr.textContent = "✓ Cuenta creada correctamente. Ya
puedes iniciar sesión.";
            loginErr.style.background = "#d1fae5";
            loginErr.style.color = "#065f46";
            loginErr.style.borderColor = "#6ee7b7";
            loginErr.classList.remove("hidden");
            setTimeout(() => { loginErr.classList.add("hidden");
loginErr.removeAttribute("style"); }, 4000);
```

```
        }
    } catch (err) {
        showError("register-error", err.message);
    } finally {
        DOM.btnDoRegister.disabled = false;
        DOM.btnDoRegister.textContent = "Registrar";
    }
};

DOM.btnDoLogin.onclick = async () => {
    clearError("login-error");
    const payload = { username: DOM.inLoginUser.value.trim(), password:
DOM.inLoginPass.value };
    if (!payload.username || !payload.password) {
        showError("login-error", "Introduce tu usuario y contraseña para
continuar.");
        return;
    }
    try {
        DOM.btnDoLogin.disabled = true;
        DOM.btnDoLogin.textContent = "Accediendo…";
        const res = await fetch(`${API_BASE}/api/auth/login`, {
            method: "POST", headers: { "Content-Type": "application/json"
}, body: JSON.stringify(payload)
        });
        const data = await res.json();
        if (!res.ok) throw new Error(data.detail || "Credenciales
incorrectas");

        CONFIG_APP.currentUser = data.user_id;
        CONFIG_APP.currentName = data.nombre;
        DOM.userDisplayName.innerText = `Operador: ${data.nombre}`;

        DOM.authWrapper.classList.add("hidden");
        DOM.appWrapper.classList.remove("hidden");
        cargarDatosDashboard();
    } catch (err) {
        showError("login-error", err.message);
        DOM.inLoginPass.value = "";
        DOM.inLoginPass.focus();
    } finally {
        DOM.btnDoLogin.disabled = false;
        DOM.btnDoLogin.textContent = "Acceder";
```

```
    }
};

DOM.btnDoRecover.onclick = async () => {
    clearError("recover-error");
    const payload = {
        username: DOM.inRecUser.value.trim(),
        pregunta_seguridad: DOM.inRecPregunta.value,
        respuesta_seguridad: DOM.inRecRespuesta.value,
        nueva_password: DOM.inRecNewPass.value
    };
    if (!payload.username || !payload.nueva_password ||
!payload.respuesta_seguridad) {
        showError("recover-error", "Por favor, rellena todos los
campos.");
        return;
    }
    try {
        DOM.btnDoRecover.disabled = true;
        DOM.btnDoRecover.textContent = "Verificando…";
        const res = await fetch(`${API_BASE}/api/auth/recover`, {
            method: "POST", headers: { "Content-Type": "application/json"
}, body: JSON.stringify(payload)
        });
        const data = await res.json();
        if (!res.ok) throw new Error(data.detail || "Error de
validación");
        ocultarAuth();
        DOM.viewLogin.classList.remove("hidden");
        DOM.inLoginUser.value = payload.username;
        DOM.inLoginPass.focus();
    } catch (err) {
        showError("recover-error", err.message);
    } finally {
        DOM.btnDoRecover.disabled = false;
        DOM.btnDoRecover.textContent = "Recuperar Contraseña";
    }
};

DOM.btnLogout.onclick = () => {
    CONFIG_APP.currentUser = null;
    CONFIG_APP.currentName = "";
    DOM.appWrapper.classList.add("hidden");
```

```
    DOM.authWrapper.classList.remove("hidden");
    DOM.inLoginUser.value = "";
    DOM.inLoginPass.value = "";
    clearError("login-error");
    DOM.inLoginUser.focus();
};

/* ---------- SUBIDA DE ARCHIVOS (DROPZONE) ---------- */
DOM.fileInput.onchange = () => procesarArchivo(DOM.fileInput.files[0]);

DOM.dropZone.ondragover = (e) => {
    e.preventDefault();
    DOM.dropZone.classList.add("dragover");
};
DOM.dropZone.ondragleave = () => {
    DOM.dropZone.classList.remove("dragover");
};
DOM.dropZone.ondrop = (e) => {
    e.preventDefault();
    DOM.dropZone.classList.remove("dragover");
    const file = e.dataTransfer.files[0];
    if (file) procesarArchivo(file);
};

function procesarArchivo(file) {
    if (!file || !file.name.endsWith(".csv")) {
        DOM.fileNameDisplay.textContent = "⚠ Solo se aceptan ficheros
.csv";
        DOM.fileNameDisplay.style.color = "var(--danger)";
        DOM.btnAnalizar.classList.add("hidden");
        return;
    }
    CONFIG_APP.archivoActual = file;
    DOM.fileNameDisplay.innerHTML = `📄 <strong>${file.name}</strong>
(${(file.size / 1024).toFixed(1)} KB)`;
    DOM.fileNameDisplay.style.color = "var(--success)";
    DOM.btnAnalizar.classList.remove("hidden");
}

/* ---------- ANALIZAR ---------- */
DOM.btnAnalizar.onclick = async () => {
    if (!CONFIG_APP.archivoActual) return;
```

```
    const modelos = [];
    if (document.getElementById("chk-
rf").checked)  modelos.push("randomforest");
    if (document.getElementById("chk-svm").checked) modelos.push("svm");
    if (document.getElementById("chk-
gb").checked)  modelos.push("gradientboost");
    if (document.getElementById("chk-xgb").checked)
modelos.push("xgboost");

    if (modelos.length === 0) {
        alert("Selecciona al menos un motor de IA antes de analizar.");
        return;
    }

    const formData = new FormData();
    formData.append("archivo", CONFIG_APP.archivoActual);

    DOM.loader.classList.remove("hidden");
    DOM.btnAnalizar.classList.add("hidden");
    DOM.resultados.classList.add("hidden");
    DOM.panelDetalle.classList.add("hidden");

    try {
        const res = await fetch(
            `${API_BASE}/api/analizar?usuario_id=${CONFIG_APP.currentUser}
&modelos=${modelos.join(",")}`,
            { method: "POST", body: formData }
        );
        const data = await res.json();
        if (!res.ok) throw new Error(data.detail || "Error en el servidor
de Python.");

        RESPUESTA_GLOBAL = data.datos_modelos;
        DOM.resultados.classList.remove("hidden");

        DOM.compModelosDiv.innerHTML =
Object.keys(RESPUESTA_GLOBAL).map(key => {
            const m = RESPUESTA_GLOBAL[key].kpis;
            const esRiesgo = m.riesgo_pct > 20;
            return `
                <div class="comp-card"
onclick="cargarVistaDetallada('${key}')" id="card-${key}">
```

```
                    <div style="display:flex; justify-content:space-
between; align-items:flex-start; margin-bottom:10px;">
                        <h4 style="color:var(--primary); font-
size:0.95rem; margin:0; text-transform:uppercase; letter-
spacing:0.05em;">${key}</h4>
                        <span style="font-size:0.7rem; color:var(--text-
muted); font-weight:600;">VER DETALLE →</span>
                    </div>
                    <div style="font-size:1.8rem; font-weight:800;
color:${esRiesgo ? 'var(--danger)' : 'var(--success)'}; margin:8px 0;">
                        ${m.riesgo_pct}%
                    </div>
                    <div style="font-size:0.8rem; color:var(--text-muted);
font-weight:600;">RIESGO DETECTADO</div>
                    <div style="margin-top:10px; font-size:0.8rem;
color:var(--text-muted);">${m.ataques} amenazas · ${m.normales}
limpias</div>
                </div>
            `;
        }).join("");

    } catch (err) {
        alert("Error al analizar: " + err.message);
    } finally {
        DOM.loader.classList.add("hidden");
        DOM.btnAnalizar.classList.remove("hidden");
    }
};

/* ---------- VISTA DETALLADA ---------- */
window.cargarVistaDetallada = function (modeloKey) {
    // Restablecer tarjetas
    document.querySelectorAll(".comp-card").forEach(c =>
c.classList.remove("selected"));
    document.getElementById(`card-
${modeloKey}`).classList.add("selected");

    const datos = RESPUESTA_GLOBAL[modeloKey];
    const kpi = datos.kpis;

    DOM.detalleTitulo.innerText = `Análisis detallado —
${modeloKey.toUpperCase()}`;
    document.getElementById("det-total").innerText = kpi.total;
```

```
    document.getElementById("det-normal").innerText = kpi.normales;
    document.getElementById("det-ataques").innerText = kpi.ataques;

    // Tabla con barra de confianza
    DOM.tablaBody.innerHTML = datos.tabla.map(r => {
        const cat = r.prediccion.replace(/\s+/g, '');
        const conf = parseFloat(r.confianza);
        const barColor = conf >= 80 ? 'var(--success)' : conf >= 60 ?
'var(--warn)' : 'var(--danger)';
        return `<tr>
            <td style="color:var(--text-muted);">#${r.id}</td>
            <td><span class="badge badge-cat-
${cat}">${r.prediccion}</span></td>
            <td>
                <div class="confidence-bar-wrap">
                    <div class="confidence-bar-bg"><div class="confidence-
bar-fill" style="width:${conf}%; background:${barColor};"></div></div>
                    <span class="confidence-val">${r.confianza}%</span>
                </div>
            </td>
            <td>${r.protocolo}</td>
            <td><code style="background:#f1f5f9; padding:0.15rem 0.4rem;
border-radius:4px; font-size:0.8rem;">${r.servicio}</code></td>
            <td>${r.src_bytes} B</td>
        </tr>`;
    }).join("");

    // Gráficos
    if (chartsInstances.doughnut) chartsInstances.doughnut.destroy();
    if (chartsInstances.bar) chartsInstances.bar.destroy();

    const ctxDoughnut =
document.getElementById('chartDoughnut').getContext('2d');
    chartsInstances.doughnut = new Chart(ctxDoughnut, {
        type: 'doughnut',
        data: {
            labels: ['Limpio (Normal)', 'Amenaza'],
            datasets: [{ data: [kpi.normales, kpi.ataques],
backgroundColor: ['#10b981', '#ef4444'], borderWidth: 0 }]
        },
        options: { plugins: { legend: { position: 'bottom' } }, cutout:
'65%' }
    });
```

```
    const ctxBar = document.getElementById('chartBar').getContext('2d');
    const categoriasAtaque = ['DoS', 'Probe', 'R2L', 'U2R'];
    const valoresAtaque = categoriasAtaque.map(cat => kpi.desglose[cat] ||
0);

    chartsInstances.bar = new Chart(ctxBar, {
        type: 'bar',
        data: {
            labels: categoriasAtaque,
            datasets: [{
                label: 'Registros detectados',
                data: valoresAtaque,
                backgroundColor: ['#fca5a5', '#fcd34d', '#c7d2fe',
'#fbcfe8'],
                borderColor: ['#991b1b', '#92400e', '#3730a3', '#9d174d'],
                borderWidth: 2,
                borderRadius: 6
            }]
        },
        options: {
            scales: { y: { beginAtZero: true, grid: { color: '#f1f5f9' }
}, x: { grid: { display: false } } },
            plugins: { legend: { display: false } }
        }
    });

    DOM.panelDetalle.classList.remove("hidden");
    // Scroll suave hacia el panel
    setTimeout(() => DOM.panelDetalle.scrollIntoView({ behavior: 'smooth',
block: 'start' }), 100);
};

/* ---------- EXPORTAR CSV ---------- */
DOM.btnExportar.onclick = () => {
    const modeloKey = DOM.detalleTitulo.innerText.replace("Análisis
detallado — ", "").toLowerCase();
    const filas = RESPUESTA_GLOBAL[modeloKey].tabla;
    let csvContent = "data:text/csv;charset=utf-
8,ID,Prediccion,Confianza,Protocolo,Servicio,Bytes\n";
    filas.forEach(r => { csvContent +=
`${r.id},${r.prediccion},${r.confianza},${r.protocolo},${r.servicio},${r.s
rc_bytes}\n`; });
```

```
    const encodedUri = encodeURI(csvContent);
    const link = document.createElement("a");
    link.setAttribute("href", encodedUri);
    link.setAttribute("download", `Reporte_Forense_${modeloKey}.csv`);
    document.body.appendChild(link);
    link.click();
    document.body.removeChild(link);
};

/* ---------- DASHBOARD ---------- */
async function cargarDatosDashboard() {
    if (!CONFIG_APP.currentUser) return;
    DOM.welcomeMsg.innerText = `Bienvenido, ${CONFIG_APP.currentName}`;
    try {
        const res = await
fetch(`${API_BASE}/api/dashboard/stats?usuario_id=${CONFIG_APP.currentUser
}`);
        const data = await res.json();

        if (!data.tiene_datos) {
            DOM.dashNoData.classList.remove("hidden");
            DOM.dashDataView.classList.add("hidden");
        } else {
            DOM.dashNoData.classList.add("hidden");
            DOM.dashDataView.classList.remove("hidden");
            DOM.dashTotalFiles.innerText = data.total_archivos;
            DOM.dashTotalRecords.innerText =
data.total_registros.toLocaleString("es-ES");
            DOM.dashLastFile.innerText = data.ultimo_archivo;

            DOM.dashCompCards.innerHTML =
Object.keys(data.ultimo_detalles).map(k => {
                const info = data.ultimo_detalles[k];
                const esRiesgo = info.riesgo_pct > 20;
                return `<div class="comp-card" style="cursor:default;">
                    <span style="font-size:0.75rem; font-weight:700;
color:var(--text-muted); text-transform:uppercase; letter-
spacing:0.05em;">${k}</span>
                    <div style="font-size:1.5rem; font-weight:800;
color:${esRiesgo ? 'var(--danger)' : 'var(--success)'}; margin:6px
0;">${info.riesgo_pct}% Riesgo</div>
```

```
                    <div style="font-size:0.8rem; color:var(--text-
muted);">${info.ataques} anomalías · ${info.normales} limpias</div>
                </div>`;
            }).join("");
        }
    } catch (err) {
        console.error("Error Dashboard:", err);
    }
}

/* ---------- HISTORIAL ---------- */
async function cargarHistorial() {
    DOM.historialContent.innerHTML = `<div style="color:var(--text-muted);
padding:1rem 0;">Cargando historial…</div>`;
    try {
        const res = await
fetch(`${API_BASE}/api/historial?usuario_id=${CONFIG_APP.currentUser}`);
        const datos = await res.json();

        if (datos.length === 0) {
            DOM.historialContent.innerHTML = `
                <div class="empty-state">
                    <div class="empty-icon">📋</div>
                    <h3>Sin auditorías registradas</h3>
                    <p>Los análisis que realices en
<strong>Analizar</strong> aparecerán aquí automáticamente.</p>
                </div>`;
            return;
        }

        DOM.historialContent.innerHTML = datos.map((h, i) => {
            const stringModelos = Object.keys(h.comparativa)
                .map(k => `<span style="background:var(--primary-light);
color:var(--primary); border-radius:4px; padding:0.15rem 0.5rem; font-
size:0.75rem; font-weight:700;">${k}:
${h.comparativa[k].riesgo_pct}%</span>`)
                .join(" ");
            return `<div class="hist-card">
                <div style="display:flex; justify-content:space-between;
align-items:flex-start; flex-wrap:wrap; gap:0.5rem;">
                    <strong style="color:var(--sidebar-dark); font-
size:1rem;">🗀 ${h.nombre_archivo}</strong>
```

```
                    <span style="font-size:0.78rem; color:var(--text-
muted);">Auditoría #${datos.length - i}</span>
                </div>
                <div style="font-size:0.875rem; color:var(--text-main);
margin-top:8px;">${h.total_registros.toLocaleString("es-ES")} registros
analizados</div>
                <div style="display:flex; flex-wrap:wrap; gap:0.4rem;
margin-top:10px;">${stringModelos}</div>
            </div>`;
        }).join("");
    } catch (err) {
        DOM.historialContent.innerHTML = `<div class="empty-state"><div
class="empty-icon">⚠</div><h3>Error de conexión</h3><p>No se pudo cargar
el historial. Comprueba que el servidor está en marcha.</p></div>`;
    }
}
```

Index.html

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>NetRadar - Plataforma Analítica</title>
    <link rel="stylesheet" href="style.css">
    <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
</head>
<body>
    <div id="auth-wrapper" class="auth-container">
        <div class="auth-card">
            <div class="auth-brand">
                <img src="netradar_logo.png" alt="Logo NetRadar"
class="netradar-logo-auth">
                <h2>Iniciar Sesión</h2>
            </div>

            <div id="auth-login-view">
                <input type="text" id="login-username"
placeholder="Usuario corporativo">
                <input type="password" id="login-password"
placeholder="Contraseña">
                <button class="btn-primary full-width" id="btn-do-
login">Acceder</button>
                <div class="auth-links">
```

```
                    <a href="#" id="link-go-register">Crear cuenta</a>
                    <a href="#" id="link-go-recover">¿Olvidó su
contraseña?</a>
                </div>
            </div>

            <div id="auth-register-view" class="hidden">
                <input type="text" id="reg-nombre" placeholder="Nombre
completo">
                <input type="text" id="reg-username" placeholder="Nombre
de usuario">
                <input type="password" id="reg-password"
placeholder="Establecer contraseña">
                <select id="reg-pregunta" style="width: 100%; padding:
0.75rem; margin-bottom: 1rem; border-radius: 8px; border: 1px solid var(--
border-color);">
                    <option value="mascota">¿Cuál es el nombre de tu
primera mascota?</option>
                    <option value="ciudad">¿En qué ciudad nacieron tus
padres?</option>
                </select>
                <input type="text" id="reg-respuesta"
placeholder="Respuesta secreta">
                <button class="btn-primary full-width" id="btn-do-
register">Registrar</button>
                <a href="#" class="link-back-login" style="display:block;
text-align:center; margin-top:10px;">Volver al login</a>
            </div>

            <div id="auth-recover-view" class="hidden">
                <input type="text" id="rec-username" placeholder="Usuario
corporativo">
                <select id="rec-pregunta" style="width: 100%; padding:
0.75rem; margin-bottom: 1rem; border-radius: 8px; border: 1px solid var(--
border-color);">
                    <option value="mascota">¿Cuál es el nombre de tu
primera mascota?</option>
                    <option value="ciudad">¿En qué ciudad nacieron tus
padres?</option>
                </select>
                <input type="text" id="rec-respuesta"
placeholder="Respuesta secreta">
```

```
                <input type="password" id="rec-newpassword"
placeholder="Nueva contraseña">
                <button class="btn-primary full-width" id="btn-do-
recover">Recuperar Contraseña</button>
                <a href="#" class="link-back-login" style="display:block;
text-align:center; margin-top:10px;">Volver al login</a>
            </div>
        </div>
    </div>

    <div id="app-wrapper" class="app-layout hidden">

        <aside class="sidebar" id="sidebar">
            <div class="sidebar-header">
                <div class="brand-logo-container">
                    <img src="netradar_logo.png" alt="NetRadar Core"
class="netradar-logo-sidebar">
                </div>
                <button id="btn-toggle-sidebar" class="hamburger-
btn">☰</button>
            </div>

            <nav class="sidebar-menu">
                <button class="nav-btn active" data-tab="dashboard">
                    <span class="nav-text">Inicio</span>
                </button>
                <button class="nav-btn" data-tab="analizar">
                    <span class="nav-text">Analizar</span>
                </button>
                <button class="nav-btn" data-tab="historial">
                    <span class="nav-text">Historial</span>
                </button>
                <button class="nav-btn" data-tab="documentacion">
                    <span class="nav-text">Información</span>
                </button>
            </nav>

            <div class="sidebar-footer">
                <div id="user-display-name" class="nav-text"
style="margin-bottom:10px; color:#1e293b; font-size:0.9rem; font-
weight:bold;">Operador: -</div>
                <button id="btn-logout" class="btn-logout">
                    <span class="nav-text">Cerrar Sesión</span>
```

```
                </button>
            </div>
        </aside>

        <main class="main-content" id="main-content">
            <section id="tab-dashboard" class="tab-panel active">
                <div class="welcome-box"><h1 id="welcome-
message">Cargando...</h1></div>
                <div id="dashboard-no-data" class="hidden" style="text-
align:center; padding:3rem; background:white; border-radius:12px; border:
1px solid var(--border-color);">
                    <h3>No hay datos recientes.</h3><p>Analiza un archivo
csv para tener una captura de tráfico.</p>
                </div>
                <div id="dashboard-data-view" class="hidden">
                    <div class="stats-grid">
                        <div class="stat-card stat-
ok"><span>Archivos</span><b id="dash-total-files">0</b></div>
                        <div class="stat-card stat-warn"><span>Tramas
Analizadas</span><b id="dash-total-records">0</b></div>
                        <div class="stat-card stat-danger"><span>Último
Fichero</span><b id="dash-last-file" style="font-size: 1.2rem; margin-
top:10px; word-break: break-all;">-</b></div>
                    </div>
                    <h3 style="margin-top: 2rem;">Métricas de Riesgo
(Última Evaluación)</h3>
                    <div id="dash-comparative-cards" class="comparison-
grid"></div>
                </div>
            </section>

            <section id="tab-analizar" class="tab-panel hidden">
                <div class="upload-card" id="drop-zone">
                    <h2>Analizador de Datos Offline</h2>
                    <input type="file" id="file-input" accept=".csv"
hidden>
                    <label class="btn-primary" for="file-input"
style="margin-top: 10px;">Seleccionar Fichero CSV</label>
                    <p id="file-name-display" style="margin-top:10px;
color:#64748b;">O arrástralo hasta aquí</p>

                    <div class="model-selection-box" style="margin-
top:25px; text-align:left;">
```

```
                        <h3>Motores de Inteligencia Artificial
Activos</h3>
                        <div class="checkbox-group" style="display:flex;
gap:15px; margin-top:10px; flex-wrap:wrap;">
                            <label><input type="checkbox" id="chk-rf"
checked value="randomforest"> Random Forest</label>
                            <label><input type="checkbox" id="chk-svm"
value="svm"> SVM</label>
                            <label><input type="checkbox" id="chk-gb"
value="gradientboost"> Gradient Boosting</label>
                            <label><input type="checkbox" id="chk-xgb"
value="xgboost"> XGBoost</label>
                        </div>
                    </div>
                    <button id="btn-analizar" class="btn-primary hidden"
style="margin-top:20px; font-size: 1.1rem;">Analizar Tráfico
Ahora</button>
                </div>

                <div id="loader" class="loader hidden"><div
class="spinner"></div><p>Ingiriendo e infiriendo datos
secuencialmente...</p></div>

                <div id="resultados" class="hidden" style="margin-top:
30px;">
                    <h3>1. Resumen Ejecutivo (Haz clic en una tarjeta para
ver sus detalles)</h3>
                    <div id="contenedor-comparativo-modelos"
class="comparison-grid"></div>

                    <div id="panel-detalle-modelo" class="hidden detail-
panel">
                        <div class="detail-header">
                            <h2 id="detalle-titulo">Detalles del
Modelo</h2>
                            <button class="btn-secondary" id="btn-
exportar-csv">📥 Exportar a CSV</button>
                        </div>

                        <div class="detail-kpis" style="display: grid;
grid-template-columns: repeat(auto-fit, minmax(200px, 1fr)); gap: 1rem;
margin-bottom: 2rem;">
```

```
                            <div class="stat-card"><span>Total
Escaneado</span><b id="det-total">0</b></div>
                            <div class="stat-card stat-ok"><span>Tráfico
Limpio</span><b id="det-normal">0</b></div>
                            <div class="stat-card stat-
danger"><span>Amenazas Detectadas</span><b id="det-ataques">0</b></div>
                        </div>

                        <div class="detail-charts" style="display:flex;
flex-wrap:wrap; gap:20px; margin-bottom:20px;">
                            <div class="chart-box" style="flex:1; min-
width:300px; background:#f8fafc; padding:15px; border-radius:8px;
border:1px solid var(--border-color); text-align:center;">
                                <h4 style="color:var(--text-muted);
margin-top:0;">Proporción de Riesgo</h4>
                                <canvas id="chartDoughnut"></canvas>
                            </div>
                            <div class="chart-box" style="flex:1; min-
width:300px; background:#f8fafc; padding:15px; border-radius:8px;
border:1px solid var(--border-color); text-align:center;">
                                <h4 style="color:var(--text-muted);
margin-top:0;">Tipos de Ataque Encontrados</h4>
                                <canvas id="chartBar"></canvas>
                            </div>
                        </div>

                        <h3 style="margin-top: 2rem;">2. Extracción
Forense Lineal</h3>
                        <div class="table-wrap">
                            <table id="tabla-resultados" style="width:
100%; text-align: left; border-collapse: collapse;">
                                <thead>
                                    <tr>
                                        <th style="padding: 1rem; border-
bottom: 1px solid var(--border-color);">ID</th>
                                        <th style="padding: 1rem; border-
bottom: 1px solid var(--border-color);">Categoría IA</th>
                                        <th style="padding: 1rem; border-
bottom: 1px solid var(--border-color);">Confianza</th>
                                        <th style="padding: 1rem; border-
bottom: 1px solid var(--border-color);">Protocolo</th>
                                        <th style="padding: 1rem; border-
bottom: 1px solid var(--border-color);">Servicio</th>
```

```
                                        <th style="padding: 1rem; border-
bottom: 1px solid var(--border-color);">Bytes</th>
                                    </tr>
                                </thead>
                                <tbody id="tabla-body"></tbody>
                            </table>
                        </div>
                    </div>
                </div>
            </section>

            <section id="tab-historial" class="tab-panel hidden">
                <h2>Historial</h2>
                <div id="historial-content" style="margin-top:
20px;"></div>
            </section>

             <!-- INFORMACIÓN -->
            <section id="tab-documentacion" class="tab-panel hidden">
                <div class="welcome-box">
                    <h1 style="font-size:1.75rem;">Documentación
técnica</h1>
                    <p class="welcome-subtitle">Información sobre los
modelos, el conjunto de datos y cómo interpretar los resultados</p>
                </div>

                <!-- Bloque: Conjunto de datos -->
                <div class="info-card">
                    <div class="info-card-header">
                        <span class="info-card-icon">🗄</span>
                        <div>
                            <h2 class="info-card-title">Conjunto de datos:
NSL-KDD</h2>
                            <p class="info-card-subtitle">Base del
entrenamiento y la evaluación de todos los modelos</p>
                        </div>
                    </div>
                    <p>NetRadar utiliza el conjunto de datos <strong>NSL-
KDD</strong>, un estándar de referencia en detección de intrusiones en
red. Fue preprocesado eliminando duplicados y codificando variables
categóricas, resultando en <strong>125.964 muestras de
entrenamiento</strong> y <strong>22.541 muestras de test</strong>, cada
una descrita por <strong>42 características</strong> de red.</p>
```

```
                    <div class="dataset-stats">
                        <div class="ds-stat"><span class="ds-
num">125.964</span><span class="ds-label">Muestras de
entrenamiento</span></div>
                        <div class="ds-stat"><span class="ds-
num">22.541</span><span class="ds-label">Muestras de test</span></div>
                        <div class="ds-stat"><span class="ds-
num">42</span><span class="ds-label">Características por
muestra</span></div>
                        <div class="ds-stat"><span class="ds-
num">5</span><span class="ds-label">Clases objetivo</span></div>
                    </div>
                    <p style="font-size:0.85rem; color:var(--text-muted);
margin-bottom:0;"><strong>Nota sobre el desequilibrio de clases:</strong>
Las clases R2L (995 muestras) y U2R (52 muestras) están fuertemente
subrepresentadas frente a Normal (67.343) y DoS (45.927). Esto explica la
dificultad de todos los modelos para detectar estas categorías con alta
sensibilidad.</p>
                </div>

                <!-- Bloque: Resultados reales de los modelos -->
                <div class="info-card">
                    <div class="info-card-header">
                        <span class="info-card-icon">📊</span>
                        <div>
                            <h2 class="info-card-title">Rendimiento real
de los modelos</h2>
                            <p class="info-card-subtitle">Métricas
obtenidas sobre el conjunto de test NSL-KDD (22.541 muestras)</p>
                        </div>
                    </div>
                    <div class="metrics-table-wrap">
                        <table class="metrics-table">
                            <thead>
                                <tr>
                                    <th>Modelo</th>
                                    <th>Accuracy</th>
                                    <th>F1-macro</th>
                                    <th>Fortaleza principal</th>
                                </tr>
                            </thead>
                            <tbody>
                                <tr>
```

```
                                    <td><strong>Random
Forest</strong></td>
                                    <td><span class="metric-badge metric-
ok">73,98 %</span></td>
                                    <td><span class="metric-badge metric-
warn">48,39 %</span></td>
                                    <td>Alta precisión en DoS y Probe. Muy
robusto y rápido.</td>
                                </tr>
                                <tr>
                                    <td><strong>SVM</strong></td>
                                    <td><span class="metric-badge metric-
ok">76,55 %</span></td>
                                    <td><span class="metric-badge metric-
warn">55,26 %</span></td>
                                    <td>Mejor sensibilidad a R2L entre
todos los modelos.</td>
                                </tr>
                                <tr>
                                    <td><strong>Gradient
Boosting</strong></td>
                                    <td><span class="metric-badge metric-
ok">75,44 %</span></td>
                                    <td><span class="metric-badge metric-
ok">58,53 %</span></td>
                                    <td>Mayor recall en U2R (52 %).
Equilibrado entre clases.</td>
                                </tr>
                                <tr class="metrics-best-row">
                                    <td><strong>XGBoost ☆</strong></td>
                                    <td><span class="metric-badge metric-
ok">76,68 %</span></td>
                                    <td><span class="metric-badge metric-
best">59,67 %</span></td>
                                    <td>Mejor F1-macro global. Recomendado
para análisis completos.</td>
                                </tr>
                            </tbody>
                        </table>
                    </div>
                    <p style="font-size:0.85rem; color:var(--text-muted);
margin-bottom:0;">El <strong>F1-macro</strong> es la métrica más relevante
aquí porque pondera por igual todas las clases, incluyendo las
```

```
minoritarias. Un Accuracy alto no implica que el modelo detecte bien los
ataques raros.</p>
                </div>

                <!-- Bloque: Tipos de ataques -->
                <div class="info-card">
                    <div class="info-card-header">
                        <span class="info-card-icon">🛡</span>
                        <div>
                            <h2 class="info-card-title">Categorías de
amenazas detectadas</h2>
                            <p class="info-card-subtitle">Taxonomía NSL-
KDD de ataques de red</p>
                        </div>
                    </div>
                    <div class="attack-grid">
                        <div class="attack-card attack-dos">
                            <h3>DoS <span class="attack-label">Denial of
Service</span></h3>
                            <p>Saturan memoria o ancho de banda
inutilizando servicios legítimos. Son los más frecuentes en el dataset
(45.927 muestras de entrenamiento). <em>Ejemplos: neptune, smurf,
pod.</em></p>
                        </div>
                        <div class="attack-card attack-probe">
                            <h3>Probe <span class="attack-
label">Sondeo</span></h3>
                            <p>Técnicas de escaneo para recolectar
información de puertos y vulnerabilidades del objetivo. <em>Ejemplos:
portsweep, nmap, satan.</em></p>
                        </div>
                        <div class="attack-card attack-r2l">
                            <h3>R2L <span class="attack-label">Remote to
Local</span></h3>
                            <p>Acceso no autorizado explotando fallos del
sistema desde una estación remota. Clase muy minoritaria y difícil de
detectar. <em>Ejemplos: ftp_write, imap.</em></p>
                        </div>
                        <div class="attack-card attack-u2r">
                            <h3>U2R <span class="attack-label">User to
Root</span></h3>
```

```
                        <p>Elevación de privilegios desde una cuenta
de usuario normal hasta acceso root. La clase más escasa del dataset (52
muestras). <em>Ejemplos: buffer_overflow, loadmodule.</em></p>
                    </div>
                </div>
            </div>

            <!-- Bloque: Algoritmos -->
            <div class="info-card">
                <div class="info-card-header">
                    <span class="info-card-icon">🤘</span>
                    <div>
                        <h2 class="info-card-title">Algoritmos de
aprendizaje automático</h2>
                        <p class="info-card-subtitle">Principio de
funcionamiento de cada motor de IA</p>
                    </div>
                </div>
                <div class="algo-list">
                    <div class="algo-item">
                        <div class="algo-name">Random Forest</div>
                        <div class="algo-desc">Construye múltiples
árboles de decisión independientes y combina sus votos. Alta resiliencia
ante datos ruidosos y muy interpretable. Línea base recomendada.</div>
                    </div>
                    <div class="algo-item">
                        <div class="algo-name">SVM</div>
                        <div class="algo-desc">Proyecta los datos a un
espacio de alta dimensión buscando el hiperplano que maximiza la
separación entre clases. Entrenado con 20.000 muestras estratificadas por
limitaciones de escala.</div>
                    </div>
                    <div class="algo-item">
                        <div class="algo-name">Gradient Boosting</div>
                        <div class="algo-desc">Árbol secuencial que en
cada iteración corrige los errores del árbol anterior. Robusto ante clases
minoritarias como U2R, donde obtuvo el mejor recall (52 %).</div>
                    </div>
                    <div class="algo-item">
                        <div class="algo-name">XGBoost</div>
                        <div class="algo-desc">Evolución optimizada de
Gradient Boosting con paralelización extrema. Obtuvo el mejor F1-macro del
```

```
conjunto (59,67 %), siendo el modelo más recomendado para análisis
completos.</div>
                        </div>
                    </div>
                </div>

                <!-- Bloque: Cómo interpretar resultados -->
                <div class="info-card">
                    <div class="info-card-header">
                        <span class="info-card-icon">💡</span>
                        <div>
                            <h2 class="info-card-title">Cómo interpretar
los resultados</h2>
                            <p class="info-card-subtitle">Guía para leer
correctamente el análisis forense</p>
                        </div>
                    </div>
                    <div class="tips-list">
                        <div class="tip-item">
                            <span class="tip-icon">🎯</span>
                            <div><strong>Porcentaje de riesgo:</strong>
Indica qué fracción del tráfico analizado fue clasificada como amenaza. Un
valor elevado no implica necesariamente un ataque real; puede deberse a
tráfico inusual o características del fichero de entrada.</div>
                        </div>
                        <div class="tip-item">
                            <span class="tip-icon">🔢</span>
                            <div><strong>Confianza:</strong> Probabilidad
estimada por el modelo de que una trama pertenezca a la categoría
asignada. Valores por debajo del 60 % deben revisarse manualmente.</div>
                        </div>
                        <div class="tip-item">
                            <span class="tip-icon">⚖️</span>
                            <div><strong>Comparar modelos:</strong> Se
recomienda ejecutar al menos dos modelos y comparar sus resultados. Si
XGBoost y Random Forest coinciden en una categoría, la predicción es más
fiable.</div>
                        </div>
                        <div class="tip-item">
                            <span class="tip-icon">⚠️</span>
                            <div><strong>Limitaciones:</strong> Los
modelos fueron entrenados con datos 2007 (NSL-KDD). Ataques modernos no
```

```
contemplados en ese dataset pueden ser clasificados incorrectamente como
tráfico normal.</div>
                        </div>
                    </div>
                </div>
            </section>
        </main>
    </div>
    <script src="app.js"></script>
</body>
</html>
```

Style.css

```
:root {
    --bg-main: #f1f5f9;
    --bg-white: #ffffff;
    --sidebar-bg: #ffffff;
    --sidebar-text: #1e293b;
    --sidebar-dark: #1e293b;
    --primary: #0091ff;
    --primary-light: #e6f4ff;
    --text-main: #334155;
    --text-muted: #64748b;
    --border-color: #e2e8f0;
    --danger: #ef4444;
    --success: #10b981;
    --warn: #f59e0b;
    --radius: 12px;
    --shadow-sm: 0 1px 3px rgba(0,0,0,0.06);
    --shadow-md: 0 4px 12px rgba(0,0,0,0.08);
    --transition: 0.2s ease;
}

/* --- RESET Y BASE --- */
*, *::before, *::after { box-sizing: border-box; }
body {
    font-family: -apple-system, BlinkMacSystemFont, "Segoe UI", Roboto,
sans-serif;
    background-color: var(--bg-main);
    color: var(--text-main);
    margin: 0;
    padding: 0;
    font-size: 15px;
```

```
    line-height: 1.6;
}
.hidden { display: none !important; }

/* --- CAMPOS DE FORMULARIO --- */
.field-group { margin-bottom: 1rem; }
.field-label {
    display: block;
    font-size: 0.8rem;
    font-weight: 600;
    color: var(--text-muted);
    text-transform: uppercase;
    letter-spacing: 0.04em;
    margin-bottom: 0.35rem;
}
.field-error {
    background: #fef2f2;
    color: #b91c1c;
    border: 1px solid #fecaca;
    border-radius: 8px;
    padding: 0.6rem 0.9rem;
    font-size: 0.85rem;
    margin-bottom: 1rem;
    animation: fadeIn 0.2s ease;
}
@keyframes fadeIn { from { opacity: 0; transform: translateY(-4px); } to {
opacity: 1; transform: none; } }

/* --- AUTH --- */
.auth-container {
    display: flex;
    justify-content: center;
    align-items: center;
    min-height: 100vh;
    background: radial-gradient(circle at 60% 40%, #e0f0ff 0%, #cbd5e1
100%);
}
.auth-card {
    background: var(--bg-white);
    padding: 2.5rem;
    border-radius: 18px;
    box-shadow: 0 12px 32px rgba(0,0,0,0.08);
    width: 100%;
```

```
    max-width: 400px;
    border: 1px solid var(--border-color);
}
.auth-brand { text-align: center; margin-bottom: 2rem; }
.auth-brand h2 { margin: 0.5rem 0 0; color: var(--sidebar-dark); font-
size: 1.25rem; font-weight: 600; }
.netradar-logo-auth { max-width: 160px; height: auto; margin-bottom:
0.25rem; }

.auth-card input,
.auth-card select {
    width: 100%;
    padding: 0.7rem 0.9rem;
    border: 1px solid var(--border-color);
    border-radius: 8px;
    background-color: #f8fafc;
    font-size: 0.95rem;
    color: var(--text-main);
    transition: border-color var(--transition), box-shadow var(--
transition);
    outline: none;
}
.auth-card input:focus,
.auth-card select:focus {
    border-color: var(--primary);
    box-shadow: 0 0 0 3px rgba(0,145,255,0.12);
    background: white;
}
.auth-links { display: flex; justify-content: space-between; margin-top:
1.25rem; font-size: 0.85rem; }
.auth-links a { color: var(--primary); text-decoration: none; }
.auth-links a:hover { text-decoration: underline; }
.link-back-login { color: var(--text-muted) !important; font-size:
0.875rem; }
.link-back-login:hover { color: var(--primary) !important; }

/* --- LAYOUT APP --- */
.app-layout { display: flex; min-height: 100vh; }

/* --- SIDEBAR --- */
.sidebar {
    width: 260px;
    background-color: var(--sidebar-bg);
```

```
    color: var(--sidebar-text);
    display: flex;
    flex-direction: column;
    position: fixed;
    height: 100vh;
    left: 0; top: 0;
    z-index: 100;
    transition: width 0.3s ease;
    overflow: hidden;
    border-right: 1px solid var(--border-color);
}
.sidebar-header {
    padding: 1.25rem 1.25rem;
    display: flex;
    align-items: center;
    justify-content: space-between;
    width: 260px;
    box-sizing: border-box;
    border-bottom: 1px solid var(--border-color);
}
.brand-logo-container { display: flex; align-items: center; }
.netradar-logo-sidebar { max-width: 130px; height: auto; }
.hamburger-btn {
    background: transparent;
    border: none;
    color: var(--sidebar-text);
    font-size: 1.4rem;
    cursor: pointer;
    padding: 0.25rem 0.4rem;
    border-radius: 6px;
    transition: background var(--transition);
}
.hamburger-btn:hover { background: var(--bg-main); }

.sidebar-menu {
    flex: 1;
    padding: 1rem 0.75rem;
    display: flex;
    flex-direction: column;
    gap: 0.25rem;
    width: 260px;
    box-sizing: border-box;
}
```

```
.sidebar-menu .nav-btn {
    display: flex;
    align-items: center;
    gap: 0.65rem;
    padding: 0.7rem 1rem;
    background: none;
    border: none;
    color: var(--sidebar-text);
    text-align: left;
    cursor: pointer;
    border-radius: 8px;
    font-weight: 500;
    font-size: 0.92rem;
    transition: background var(--transition), color var(--transition);
    width: 100%;
}
.nav-icon { font-size: 1rem; flex-shrink: 0; }
.sidebar-menu .nav-btn:hover {
    background-color: var(--primary-light);
    color: var(--primary);
}
.sidebar-menu .nav-btn.active {
    background-color: var(--primary);
    color: white;
    font-weight: 600;
}

.sidebar-footer {
    padding: 1rem;
    border-top: 1px solid var(--border-color);
    background-color: var(--bg-main);
    width: 260px;
    box-sizing: border-box;
}
.user-display {
    color: var(--sidebar-dark);
    font-size: 0.85rem;
    font-weight: 600;
    white-space: nowrap;
    overflow: hidden;
    text-overflow: ellipsis;
}
.btn-logout {
```

```
    width: 100%;
    background: #e2e8f0;
    color: #0f172a;
    border: none;
    padding: 0.5rem;
    border-radius: 6px;
    cursor: pointer;
    font-weight: 600;
    font-size: 0.875rem;
    transition: background var(--transition);
}
.btn-logout:hover { background: #cbd5e1; }

/* Sidebar colapsada */
.sidebar.collapsed { width: 60px; }
.sidebar.collapsed .brand-logo-container,
.sidebar.collapsed .sidebar-menu,
.sidebar.collapsed .sidebar-footer { display: none; }
.sidebar.collapsed .sidebar-header {
    width: 60px;
    padding: 1.25rem 0;
    justify-content: center;
}

/* --- CONTENIDO PRINCIPAL --- */
.main-content {
    flex: 1;
    margin-left: 260px;
    padding: 2rem 2.5rem;
    box-sizing: border-box;
    transition: margin-left 0.3s ease;
    background-color: var(--bg-main);
}
.main-content.expanded { margin-left: 60px; }
.tab-panel { max-width: 1100px; margin: 0 auto; }

/* --- WELCOME BOX --- */
.welcome-box {
    margin-bottom: 2rem;
}
.welcome-box h1 {
    font-size: 2rem;
    font-weight: 700;
```

```
    color: var(--sidebar-dark);
    margin: 0 0 0.25rem 0;
}
.welcome-subtitle {
    color: var(--text-muted);
    margin: 0;
    font-size: 0.9rem;
}

/* --- SECTION TITLE --- */
.section-title {
    font-size: 1rem;
    font-weight: 700;
    color: var(--sidebar-dark);
    text-transform: uppercase;
    letter-spacing: 0.04em;
    margin: 1.75rem 0 1rem 0;
}

/* --- EMPTY STATE --- */
.empty-state {
    text-align: center;
    padding: 3.5rem 2rem;
    background: white;
    border-radius: var(--radius);
    border: 1px solid var(--border-color);
    box-shadow: var(--shadow-sm);
}
.empty-icon { font-size: 2.5rem; margin-bottom: 1rem; }
.empty-state h3 { margin: 0 0 0.5rem; color: var(--sidebar-dark); }
.empty-state p { color: var(--text-muted); margin: 0; font-size: 0.9rem; }

/* --- TARJETAS ESTADÍSTICAS --- */
.stats-grid { display: grid; grid-template-columns: repeat(auto-fit,
minmax(240px, 1fr)); gap: 1.25rem; }
.stat-card {
    background: var(--bg-white);
    padding: 1.5rem;
    border-radius: var(--radius);
    border: 1px solid var(--border-color);
    display: flex;
    flex-direction: column;
    box-shadow: var(--shadow-sm);
```

```
    transition: box-shadow var(--transition), transform var(--transition);
}
.stat-card:hover { box-shadow: var(--shadow-md); transform: translateY(-
1px); }
.stat-card span { font-size: 0.78rem; color: var(--text-muted); font-
weight: 700; text-transform: uppercase; letter-spacing: 0.05em; }
.stat-card b { font-size: 1.75rem; margin-top: 0.5rem; color: var(--
sidebar-dark); }
.stat-ok  { border-top: 4px solid var(--success); }
.stat-warn { border-top: 4px solid var(--warn); }
.stat-danger { border-top: 4px solid var(--danger); }

/* --- BOTONES --- */
.btn-primary {
    background-color: var(--primary);
    color: white;
    border: none;
    padding: 0.65rem 1.25rem;
    border-radius: 8px;
    cursor: pointer;
    font-weight: 600;
    font-size: 0.95rem;
    transition: background var(--transition), transform var(--transition),
box-shadow var(--transition);
    display: inline-block;
    text-decoration: none;
}
.btn-primary:hover {
    background-color: #0077d6;
    transform: translateY(-1px);
    box-shadow: 0 4px 10px rgba(0,145,255,0.25);
}
.btn-primary:active { transform: translateY(0); }
.full-width { width: 100%; text-align: center; }

.btn-secondary {
    background-color: transparent;
    color: var(--primary);
    border: 2px solid var(--primary);
    padding: 0.5rem 1rem;
    border-radius: 8px;
    font-weight: 600;
    font-size: 0.875rem;
```

```
    cursor: pointer;
    transition: all var(--transition);
    display: inline-flex;
    align-items: center;
    gap: 0.4rem;
}
.btn-secondary:hover {
    background-color: var(--primary-light);
    transform: translateY(-1px);
    box-shadow: 0 4px 8px rgba(0,145,255,0.15);
}

/* --- UPLOAD / DROP ZONE --- */
.upload-card {
    background: var(--bg-white);
    border: 2px dashed #cbd5e1;
    border-radius: 16px;
    padding: 2.5rem 2rem;
    text-align: center;
    box-shadow: var(--shadow-sm);
    transition: border-color var(--transition), box-shadow var(--
transition);
}
.upload-card.dragover {
    border-color: var(--primary);
    box-shadow: 0 0 0 4px rgba(0,145,255,0.1);
}
.upload-icon { font-size: 2.5rem; margin-bottom: 0.75rem; }

/* --- SELECCIÓN DE MODELOS --- */
.model-selection-box {
    margin-top: 2rem;
    padding: 1.25rem 1.5rem;
    background: #f8fafc;
    border-radius: 10px;
    border: 1px solid var(--border-color);
    text-align: left;
}
.checkbox-group { display: flex; gap: 0.75rem; flex-wrap: wrap; margin-
top: 10px; }
.model-checkbox {
    display: flex;
    align-items: center;
```

```
    gap: 0.4rem;
    background: white;
    border: 1px solid var(--border-color);
    border-radius: 8px;
    padding: 0.45rem 0.85rem;
    cursor: pointer;
    font-size: 0.875rem;
    font-weight: 500;
    transition: border-color var(--transition), background var(--
transition);
    user-select: none;
}
.model-checkbox:hover { border-color: var(--primary); background: var(--
primary-light); }
.model-checkbox input[type="checkbox"] { accent-color: var(--primary); }
.model-tag {
    font-size: 0.7rem;
    font-weight: 700;
    background: #d1fae5;
    color: #065f46;
    border-radius: 4px;
    padding: 0.1rem 0.4rem;
    text-transform: uppercase;
    letter-spacing: 0.03em;
}
.model-tag-best {
    background: #fef3c7;
    color: #92400e;
}

/* --- LOADER --- */
.loader { text-align: center; padding: 2.5rem; }
.spinner {
    width: 40px;
    height: 40px;
    border: 4px solid var(--border-color);
    border-top: 4px solid var(--primary);
    border-radius: 50%;
    margin: 0 auto 1rem auto;
    animation: spin 0.8s linear infinite;
}
@keyframes spin { 0% { transform: rotate(0deg); } 100% { transform:
rotate(360deg); } }
```

```
/* --- TARJETAS DE COMPARATIVA --- */
.comparison-grid { display: grid; grid-template-columns: repeat(auto-fit,
minmax(220px, 1fr)); gap: 1.25rem; }
.comp-card {
    cursor: pointer;
    background: white;
    padding: 1.25rem;
    border-radius: var(--radius);
    border: 2px solid var(--border-color);
    box-shadow: var(--shadow-sm);
    transition: border-color var(--transition), box-shadow var(--
transition), transform var(--transition), background var(--transition);
}
.comp-card:hover {
    border-color: var(--primary);
    box-shadow: var(--shadow-md);
    transform: translateY(-2px);
}
.comp-card.selected {
    border-color: var(--primary);
    background: var(--primary-light);
}

/* --- PANEL DETALLE --- */
.detail-panel {
    background: white;
    border-radius: var(--radius);
    padding: 2rem;
    margin-top: 2rem;
    border: 1px solid var(--border-color);
    box-shadow: var(--shadow-md);
    animation: fadeIn 0.25s ease;
}
.detail-header {
    display: flex;
    justify-content: space-between;
    align-items: center;
    border-bottom: 1px solid var(--border-color);
    padding-bottom: 1rem;
    margin-bottom: 1.5rem;
    flex-wrap: wrap;
    gap: 0.75rem;
```

```
}
.detail-header h2 { margin: 0; color: var(--primary); font-size: 1.25rem;
}

.detail-kpis {
    display: grid;
    grid-template-columns: repeat(auto-fit, minmax(180px, 1fr));
    gap: 1rem;
    margin-bottom: 1.75rem;
}

.detail-charts {
    display: flex;
    flex-wrap: wrap;
    gap: 1.25rem;
    margin-bottom: 1.5rem;
}
.chart-box {
    flex: 1;
    min-width: 280px;
    background: #f8fafc;
    padding: 1.25rem;
    border-radius: 8px;
    border: 1px solid var(--border-color);
    text-align: center;
}
.chart-title {
    color: var(--text-muted);
    margin: 0 0 1rem 0;
    font-size: 0.85rem;
    font-weight: 700;
    text-transform: uppercase;
    letter-spacing: 0.04em;
}

/* --- TABLA RESULTADOS --- */
.table-wrap {
    background: var(--bg-white);
    border-radius: var(--radius);
    border: 1px solid var(--border-color);
    overflow: hidden;
    margin-top: 1rem;
}
```

```
#tabla-resultados {
    width: 100%;
    text-align: left;
    border-collapse: collapse;
    font-size: 0.875rem;
}
#tabla-resultados thead th {
    padding: 0.85rem 1rem;
    border-bottom: 2px solid var(--border-color);
    background: #f8fafc;
    font-size: 0.75rem;
    font-weight: 700;
    text-transform: uppercase;
    letter-spacing: 0.05em;
    color: var(--text-muted);
}
#tabla-resultados tbody tr { transition: background var(--transition); }
#tabla-resultados tbody tr:hover { background: #f8fafc; }
#tabla-resultados tbody td { padding: 0.85rem 1rem; border-bottom: 1px
solid #edf2f7; }

/* Barra de confianza en tabla */
.confidence-bar-wrap { display: flex; align-items: center; gap: 0.5rem; }
.confidence-bar-bg { flex: 1; background: #e2e8f0; border-radius: 4px;
height: 6px; overflow: hidden; }
.confidence-bar-fill { height: 100%; border-radius: 4px; background: var(-
-primary); transition: width 0.4s ease; }
.confidence-val { font-size: 0.8rem; font-weight: 600; color: var(--text-
muted); white-space: nowrap; }

/* Badges */
.badge { padding: 0.25rem 0.6rem; border-radius: 6px; font-size: 0.75rem;
font-weight: 700; }
.badge-cat-Normal         { background: #d1fae5; color: #065f46; }
.badge-cat-DoS            { background: #fee2e2; color: #991b1b; }
.badge-cat-Probe          { background: #fef3c7; color: #92400e; }
.badge-cat-R2L            { background: #e0e7ff; color: #3730a3; }
.badge-cat-U2R            { background: #fce7f3; color: #9d174d; }
.badge-cat-AtaqueDesconocido { background: #fee2e2; color: #991b1b; }

/* --- HISTORIAL --- */
.hist-card {
    background: white;
```

```
    padding: 1.25rem 1.5rem;
    margin-bottom: 0.75rem;
    border-left: 4px solid var(--primary);
    border-radius: 8px;
    box-shadow: var(--shadow-sm);
    transition: box-shadow var(--transition), transform var(--transition);
}
.hist-card:hover { box-shadow: var(--shadow-md); transform:
translateX(2px); }

/* --- SECCIÓN INFORMACIÓN --- */
.info-card {
    background: white;
    border-radius: var(--radius);
    border: 1px solid var(--border-color);
    padding: 2rem;
    margin-bottom: 1.5rem;
    box-shadow: var(--shadow-sm);
}
.info-card-header {
    display: flex;
    align-items: flex-start;
    gap: 1rem;
    margin-bottom: 1.25rem;
    padding-bottom: 1rem;
    border-bottom: 1px solid var(--border-color);
}
.info-card-icon { font-size: 1.75rem; flex-shrink: 0; line-height: 1; }
.info-card-title {
    font-size: 1.15rem;
    font-weight: 700;
    color: var(--sidebar-dark);
    margin: 0 0 0.2rem 0;
}
.info-card-subtitle {
    font-size: 0.85rem;
    color: var(--text-muted);
    margin: 0;
}

/* Dataset stats */
.dataset-stats {
    display: grid;
```

```
    grid-template-columns: repeat(auto-fit, minmax(130px, 1fr));
    gap: 1rem;
    margin: 1.25rem 0;
}
.ds-stat {
    background: #f8fafc;
    border: 1px solid var(--border-color);
    border-radius: 8px;
    padding: 1rem;
    text-align: center;
}
.ds-num { display: block; font-size: 1.5rem; font-weight: 800; color:
var(--primary); line-height: 1; }
.ds-label { display: block; font-size: 0.78rem; color: var(--text-muted);
margin-top: 0.3rem; }

/* Tabla de métricas */
.metrics-table-wrap { overflow-x: auto; margin: 1rem 0; }
.metrics-table {
    width: 100%;
    border-collapse: collapse;
    font-size: 0.875rem;
}
.metrics-table thead th {
    background: #f8fafc;
    padding: 0.75rem 1rem;
    text-align: left;
    font-size: 0.75rem;
    font-weight: 700;
    text-transform: uppercase;
    letter-spacing: 0.05em;
    color: var(--text-muted);
    border-bottom: 2px solid var(--border-color);
}
.metrics-table tbody td {
    padding: 0.85rem 1rem;
    border-bottom: 1px solid #edf2f7;
    vertical-align: middle;
}
.metrics-table tbody tr:hover { background: #f8fafc; }
.metrics-best-row { background: #fefce8; }
.metric-badge {
    display: inline-block;
```

```
    padding: 0.2rem 0.6rem;
    border-radius: 6px;
    font-weight: 700;
    font-size: 0.82rem;
}
.metric-ok   { background: #d1fae5; color: #065f46; }
.metric-warn { background: #fef3c7; color: #92400e; }
.metric-best { background: #fbbf24; color: #451a03; }

/* Tipos de ataques */
.attack-grid {
    display: grid;
    grid-template-columns: repeat(auto-fit, minmax(220px, 1fr));
    gap: 1rem;
    margin-top: 0.5rem;
}
.attack-card {
    background: #f8fafc;
    padding: 1.1rem;
    border-radius: 8px;
    border: 1px solid var(--border-color);
    font-size: 0.875rem;
}
.attack-card h3 { font-size: 1rem; margin: 0 0 0.5rem 0; display: flex;
align-items: baseline; gap: 0.5rem; }
.attack-label { font-size: 0.72rem; font-weight: 600; color: var(--text-
muted); text-transform: uppercase; letter-spacing: 0.04em; }
.attack-card p { margin: 0; color: var(--text-muted); line-height: 1.5; }
.attack-dos  h3 { color: #991b1b; }
.attack-probe h3 { color: #92400e; }
.attack-r2l  h3 { color: #3730a3; }
.attack-u2r  h3 { color: #9d174d; }

/* Algoritmos */
.algo-list { display: flex; flex-direction: column; gap: 0; }
.algo-item {
    display: flex;
    align-items: baseline;
    gap: 1rem;
    padding: 0.9rem 0;
    border-bottom: 1px solid #f1f5f9;
    font-size: 0.9rem;
}
```

```
.algo-item:last-child { border-bottom: none; }
.algo-name { font-weight: 700; color: var(--primary); white-space: nowrap;
min-width: 130px; }
.algo-desc { color: var(--text-muted); }

/* Tips / Interpretación */
.tips-list { display: flex; flex-direction: column; gap: 0.85rem; }
.tip-item {
    display: flex;
    align-items: flex-start;
    gap: 0.75rem;
    background: #f8fafc;
    border-radius: 8px;
    padding: 0.85rem 1rem;
    font-size: 0.875rem;
    color: var(--text-main);
    line-height: 1.55;
}
.tip-icon { font-size: 1.1rem; flex-shrink: 0; margin-top: 0.05rem; }

/* --- RESPONSIVE --- */
@media (max-width: 768px) {
    .main-content { margin-left: 0; padding: 1.25rem; }
    .sidebar { transform: translateX(-100%); }
    .welcome-box h1 { font-size: 1.5rem; }
    .detail-charts { flex-direction: column; }
    .algo-item { flex-direction: column; gap: 0.25rem; }
    .algo-name { min-width: unset; }
}
```