Universidad de Valladolid

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática
Mención en Tecnologías de la Información

WatchdogAI: Detección de Ataques de Intrusión mediante Inteligencia Artificial

Autor: D. Adrián Vara Lamúa

Universidad de Valladolid

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática
Mención en Tecnologías de la Información

WatchdogAI: Detección de Ataques de Intrusión mediante Inteligencia Artificial

Autor: D. Adrián Vara Lamúa

Tutor: Dr. D. Jesus Maria Vegas Hernandez

A mi familia, por estar siempre ahí, en cada paso, en cada caída, y en cada logro.

Agradecimientos

Han sido muchas las personas que han contribuido directa o indirectamente a que este proyecto viera la luz.

En primer lugar, quiero agradecer a mi tutor, Dr. D. Jesus María Vegas Hernandez, por su orientación, paciencia y constante implicación en el desarrollo de este Trabajo de Fin de Grado. Sus revisiones y comentarios han sido clave para encaminar tanto la aplicación como la memoria por el camino adecuado.

También agradezco a mis compañeros de grado, por las conversaciones técnicas, los retos compartidos y el entorno inspirador que me han ofrecido durante este tiempo.

Y, por supuesto, gracias a mis amigos y familia por su apoyo incondicional, por animarme en los días difíciles y por acompañarme a lo largo de todo este viaje.

Resumen

En un contexto de creciente digitalización y sofisticación de los ciberataques, los sistemas tradicionales de detección de intrusiones, basados en firmas estáticas, resultan insuficientes para hacer frente a amenazas nuevas o desconocidas. Este trabajo surge con la motivación de explorar alternativas más adaptativas mediante el uso de inteligencia artificial.

El objetivo principal ha sido evaluar comparativamente distintos algoritmos de aprendizaje automático aplicados a la detección de intrusiones en redes, con el fin de identificar cuál ofrece el mejor equilibrio entre precisión, eficiencia computacional y robustez frente a clases desbalanceadas. Para ello, se ha llevado a cabo un estudio del estado del arte, la selección y preprocesamiento del dataset CIC-IDS2017, y la implementación de un sistema modular que permite entrenar y evaluar modelos como Random Forest, XGBoost, SVM y MLP en distintos escenarios de clasificación.

Los resultados obtenidos muestran que XGBoost destaca como el modelo más eficaz, manteniendo un alto rendimiento incluso al reducir el número de características a las 30 más relevantes. En conclusión, este trabajo demuestra la viabilidad del uso de técnicas de aprendizaje automático en sistemas de detección de intrusiones, sentando una base sólida para el desarrollo de soluciones más inteligentes, escalables y adaptadas a entornos reales.

Abstract

In a context of increasing digitalization and increasingly sophisticated cyberattacks, traditional intrusion detection systems based on static signatures are no longer sufficient to address new or unknown threats. This project aims to explore more adaptive alternatives through the use of artificial intelligence.

The main objective is to comparatively evaluate various machine learning algorithms applied to network intrusion detection, identifying the one that offers the best balance between accuracy, computational efficiency, and robustness against imbalanced classes. To achieve this, a comprehensive study was conducted, including a review of the state of the art, preprocessing of the CIC-IDS2017 dataset, and the implementation of a modular system to train and evaluate models such as Random Forest, XGBoost, SVM, and MLP under different classification scenarios.

The results show that XGBoost stands out as the most effective model, maintaining high performance even when the number of features is reduced to the 30 most relevant ones. In conclusion, this work demonstrates the feasibility of using machine learning techniques in intrusion detection systems and lays a solid foundation for the development of smarter, more scalable, and real-world-ready solutions.

Índice general

Índice de tablas

Índice de figuras

Capítulo 1

Introducción

1.1 Introducción

La sociedad actual está inmersa en un proceso de digitalización sin precedentes. Vivimos rodeados de dispositivos conectados, servicios online y entornos tecnológicos que, cada día más, dependen de infraestructuras digitales. Este fenómeno ha traído consigo mejoras notables en eficiencia, accesibilidad y automatización, tanto en el ámbito personal como en el profesional. Sin embargo, esta hiperconectividad también conlleva nuevos riesgos. Cuanto más dependemos de los sistemas conectados, más expuestos estamos a amenazas cibernéticas cada vez más sofisticadas y persistentes. Según el informe más reciente de ENISA [12], el número y la complejidad de los ciberataques continúa aumentando, afectando a sectores críticos como sanidad, transporte o administración pública.

La ciberseguridad se ha convertido en una necesidad crítica. Ya no se trata solo de proteger datos, sino de garantizar el correcto funcionamiento de servicios esenciales, preservar la privacidad y evitar daños económicos y reputacionales. Entre los mecanismos más relevantes para lograrlo se encuentran los sistemas de detección de intrusos (IDS, por sus siglas en inglés). Estos sistemas supervisan el tráfico de red en busca de comportamientos anómalos que puedan indicar la presencia de un atacante o de una actividad maliciosa.

Sin embargo, muchos de los IDS actuales se basan en reglas o firmas previamente conocidas, lo que los limita cuando se enfrentan a amenazas nuevas o desconocidas, como los ataques zero-day [1]. Esta rigidez provoca falsos negativos y deja huecos críticos en la defensa de las redes. En este contexto, tecnologías como el aprendizaje automático (machine learning) y la inteligencia artificial (IA) ofrecen un enfoque más flexible, capaz de adaptarse a nuevas amenazas sin intervención humana constante.

El presente Trabajo de Fin de Grado nace con el propósito de explorar el uso de algoritmos de IA en la detección de intrusiones en redes informáticas. El enfoque adoptado no busca desarrollar un único sistema, sino comparar distintas técnicas de aprendizaje automático y evaluar su rendimiento ante diferentes tipos de tráfico malicioso y legítimo. A través de este estudio, se pretende determinar qué algoritmo se comporta mejor en términos de precisión, recall, F1-score y otras métricas relevantes, proporcionando una base sólida sobre la que construir futuras soluciones de seguridad más inteligentes, eficaces y proactivas.

1.2 Motivación

La motivación principal de este proyecto parte de una observación clara: los métodos tradicionales para detectar intrusiones en redes ya no son suficientes. Los ciberataques modernos evolucionan de forma constante, y cada vez es más común encontrar amenazas que no pueden ser identificadas mediante mecanismos estáticos basados en firmas. Esto deja a muchas organizaciones vulnerables, especialmente frente a ataques desconocidos o variantes de malware diseñadas para evadir las detecciones convencionales.

Además, el volumen de datos que circula por las redes hoy en día es inmenso. Con el auge del Internet de las Cosas (IoT), el trabajo remoto y la digitalización de procesos empresariales, resulta inviable analizar todo el tráfico manualmente o mediante reglas fijas. Esta necesidad de adaptarse a entornos complejos y altamente heterogéneos, como ocurre en las redes IoT, refuerza el valor del aprendizaje automático como técnica de detección flexible y escalable [29]. En este contexto, los sistemas basados en aprendizaje automático presentan una ventaja competitiva clave: su capacidad de adaptarse, aprender de los datos y tomar decisiones basadas en patrones dinámicos.

Este Trabajo de Fin de Grado responde a esa necesidad: evaluar distintas técnicas de IA y determinar cuál resulta más eficaz en la detección de intrusiones. No se trata únicamente de demostrar que la inteligencia artificial puede aplicarse a este campo, sino de comparar de forma rigurosa sus diferentes enfoques y ofrecer conclusiones basadas en resultados medibles.

En definitiva, este proyecto busca aportar valor tanto desde el punto de vista técnico como académico, sentando las bases para un futuro en el que los sistemas de detección de intrusos no solo reaccionen, sino que predigan, aprendan y evolucionen junto al panorama de amenazas.

Capítulo 2

Objetivos y Alcance

2.1 Objetivos

Este Trabajo de Fin de Grado se enmarca en el contexto de la ciberseguridad, donde los sistemas de detección de intrusiones (*IDS*) juegan un papel clave para proteger las redes frente a accesos no autorizados o comportamientos maliciosos. En concreto, se aborda la aplicación de técnicas de aprendizaje automático (*machine learning*) como herramienta para detectar estos ataques a partir del análisis del tráfico de red.

El proyecto se ha desarrollado con un enfoque exploratorio y comparativo. No se persigue la construcción de una solución lista para producción, sino un análisis riguroso de diferentes algoritmos de clasificación aplicados a la detección de intrusiones, con el fin de evaluar su rendimiento en distintos escenarios y con diferentes configuraciones de datos.

El objetivo principal de este trabajo es identificar, mediante una evaluación técnica y sistemática, cuál de los algoritmos de aprendizaje automático estudiados ofrece el mejor comportamiento global en tareas de detección de intrusiones, considerando no solo métricas de rendimiento , sino también su coste computacional, robustez frente a clases desbalanceadas y escalabilidad.

Este análisis pretende servir de referencia para desarrolladores e investigadores que busquen aplicar técnicas de inteligencia artificial en sistemas *IDS*, aportando datos empíricos sobre el comportamiento de los modelos más comunes.

De manera más específica, los objetivos concretos del proyecto son:

- Estudiar diferentes algoritmos de aprendizaje automático orientados a la detección de anomalías y ataques en redes.
- Comparar su rendimiento mediante métricas como precisión, *recall*, *F1-score* o la tasa de falsos positivos.
- Analizar su aplicabilidad en contextos reales, valorando aspectos como el coste computacional, la complejidad de entrenamiento o su escalabilidad.
- Elaborar una documentación clara y técnica que recoja el análisis realizado, los resultados obtenidos y las conclusiones derivadas de la comparativa.

Estos objetivos permiten sentar una base sólida sobre la que podrían construirse futuras soluciones más avanzadas y adaptadas a entornos productivos.

2.1.1 Tareas a realizar

Para alcanzar estos objetivos, el desarrollo del proyecto se divide en una serie de tareas estructuradas, que marcan el ritmo y la dirección del trabajo:

- Definición y planificación del proyecto: Establecer el alcance, las fases de desarrollo, los hitos y los entregables clave.
- Estudio del problema y análisis del contexto: Investigar las características de los ataques más comunes y la naturaleza del tráfico de red, así como el papel de la IA en este tipo de detección.
- Revisión del estado del arte: Analizar soluciones existentes y trabajos previos relacionados con el uso de inteligencia artificial en sistemas IDS.
- Selección de algoritmos a evaluar: Identificar un conjunto representativo de modelos de aprendizaje automático adecuados para el análisis (por ejemplo: árboles de decisión, redes neuronales, k-NN, etc.).
- Preparación de los datos de entrada: Preprocesar el dataset para que sea adecuado para el entrenamiento y la evaluación de los modelos.
- Entrenamiento, validación y prueba de los modelos: Ejecutar cada modelo sobre los datos disponibles, registrar su comportamiento y recoger las métricas correspondientes.
- Análisis de resultados y elaboración de conclusiones: Interpretar los datos obtenidos y extraer conclusiones fundamentadas sobre la idoneidad de cada algoritmo.
- Redacción de la memoria y documentación técnica: Recoger todo el proceso en una memoria académica clara, estructurada y coherente, incluyendo los fundamentos, el desarrollo y los resultados del proyecto.

2.2 Alcance

El presente Trabajo de Fin de Grado se limita a la evaluación comparativa de diferentes algoritmos de aprendizaje automático aplicados a la detección de intrusiones en redes. Se trabajará en un entorno controlado, con datos representativos, y se asumirá una fase experimental cerrada, sin desplegar los modelos en entornos productivos reales.

El proyecto comprende:

- La selección de un conjunto limitado de algoritmos con enfoques diversos dentro del aprendizaje automático supervisado.
- La preparación y preprocesamiento de datos de red, que contengan tanto tráfico legítimo como malicioso.
- El entrenamiento, evaluación y comparación de los modelos, atendiendo a métricas cuentitativas para medir su eficacia y eficiencia.
- La documentación técnica y académica del proceso y de los resultados, orientada a facilitar su comprensión y futuras extensiones.

Dado que se trata de un trabajo académico con recursos y tiempo limitados, no se contempla:

- El desarrollo de un sistema IDS completo que opere en tiempo real.
- La integración con entornos empresariales o arquitecturas de producción.

- La creación de interfaces gráficas o mecanismos automáticos de respuesta ante alertas.
- La monitorización o reentrenamiento continuo del sistema en un entorno activo.

No obstante, todas estas limitaciones pueden considerarse líneas de mejora y ampliación en trabajos futuros, donde se aborde la implementación de un sistema completamente funcional, capaz de integrarse en infraestructuras reales, con capacidades de visualización, respuesta automática y adaptabilidad continua al entorno.

En resumen, este proyecto tiene como objetivo ofrecer una base sólida de conocimiento y evaluación técnica sobre el uso de IA para la detección de intrusiones, proporcionando resultados comparativos que puedan guiar decisiones futuras en el desarrollo de sistemas IDS inteligentes y eficaces.

Capítulo 3

Planificación

Cualquier proyecto de investigación requiere una planificación estructurada que permita alcanzar los objetivos propuestos de forma ordenada y eficaz. En el caso de este Trabajo de Fin de Grado, se ha optado por una metodología en cascada, que divide el desarrollo en fases secuenciales, permitiendo avanzar paso a paso con una clara delimitación de actividades y entregables.

Esta metodología resulta especialmente adecuada para trabajos académicos como el presente, donde el alcance está bien definido desde el inicio y no se prevén grandes cambios en los requisitos durante el desarrollo. A continuación, se describen las fases establecidas para la realización del proyecto, así como su planificación temporal.

3.1 Fases y costes

El proyecto se ha organizado en cinco fases principales, cada una con una duración estimada. Esta distribución permite estructurar el trabajo de forma coherente, facilitando la gestión del tiempo y asegurando la cobertura de todas las tareas necesarias.

Nombre de actividad	**Semanas**
Estudio preliminar y análisis del problema	1 - 2
Diseño experimental	3
Desarrollo del sistema de evaluación	4 - 8
Ejecución de pruebas y análisis de resultados	9 - 10
Documentación y redacción de la memoria del TFG	1 - 13

Tabla 3.1: Fases de desarrollo del proyecto previstas.

Para facilitar la comprensión de esta planificación, a continuación se incluye un diagrama de Gantt, donde se representan gráficamente las actividades del proyecto y su distribución temporal a lo largo de las semanas:

Este diagrama proporciona una visión global del calendario del proyecto y permite identificar solapamientos, dependencias entre tareas y puntos clave de avance.

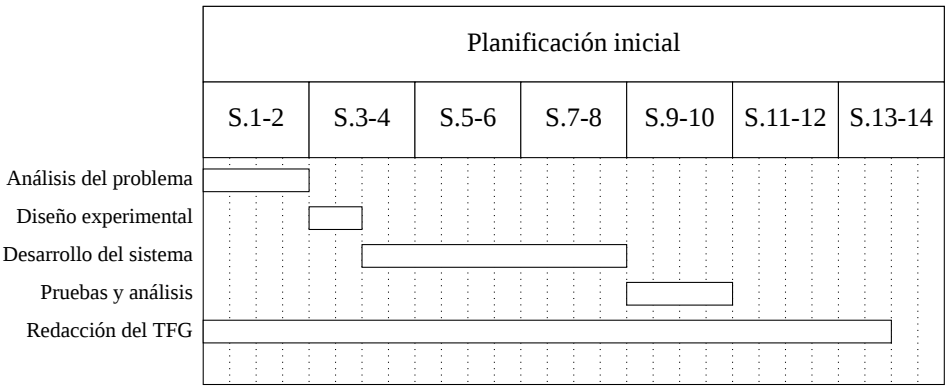


Figura 3.1: Planificación inicial

3.1.1 Descripción de las fases

1. Estudio preliminar y análisis del problema (Semana 1 - 2): En esta fase se realiza una revisión bibliográfica sobre los sistemas de detección de intrusos, las técnicas de aprendizaje automático aplicadas a la ciberseguridad y los datasets de referencia en el área. También se analizan los principales retos técnicos del problema.

2. Diseño experimental (Semana 3): Se definen los algoritmos de aprendizaje automático que serán evaluados, las métricas que se utilizarán para compararlos, los criterios de validación del experimento y el flujo general de trabajo: desde la carga del dataset hasta la obtención de los resultados.

3. Desarrollo del sistema de evaluación (Semana 4 - 8): En esta etapa se implementa el sistema encargado de entrenar, validar y comparar los modelos. Incluye tareas de preprocesamiento del dataset, extracción de características relevantes, entrenamiento de los modelos, y visualización de resultados.

4. Ejecución de pruebas y análisis de resultados (Semana 9 - 10): Se llevan a cabo las pruebas experimentales, evaluando cada algoritmo con los mismos criterios para garantizar una comparación justa. Posteriormente se analizan los resultados, se extraen conclusiones y se identifican patrones relevantes.

5. Documentación y redacción de la memoria del TFG (Semana 1 - 13): La memoria se redacta de forma progresiva a lo largo de todo el proyecto, lo que permite documentar cada fase conforme se desarrolla. Esto garantiza una mayor precisión y coherencia en la elaboración del documento final, facilitando además la incorporación de mejoras conforme avanza el trabajo.

Esta planificación permite distribuir de manera eficiente el tiempo y los recursos del proyecto, asegurando la consecución de todos los objetivos marcados dentro del marco temporal previsto.

3.1.2 Costes

Aunque este Trabajo de Fin de Grado no implica un desembolso económico directo por parte del estudiante o la institución, es posible estimar el coste real del proyecto considerando los recursos humanos y materiales utilizados durante su desarrollo.

Coste del desarrollador

El tiempo estimado dedicado al desarrollo completo del proyecto ha sido de aproximadamente 300 horas. Si se considera un coste medio de 15 €/hora como referencia para un perfil junior o de prácticas en el ámbito tecnológico, se obtiene un coste estimado de:

$$300\,\text{horas} \times 15\,€/\text{hora} = 4{,}500\,€$$

Coste del equipo utilizado

El proyecto se ha desarrollado íntegramente en un equipo personal. Las características del ordenador utilizado se detallarán más adelante en el Capitulo 8.

El coste estimado del equipo completo al momento de su adquisición fue de aproximadamente 1.200 €. Si se asume una vida útil de 4 años, se puede estimar un coste anual de:

$$\frac{1{,}200\,€}{4\,\text{años}} = 300\,€\ \text{por año}$$

Dado que el proyecto ha requerido 300 horas, y considerando un uso medio de 1500 horas anuales del equipo para actividades similares, el coste proporcional del equipo sería:

$$\left(\frac{300}{1500}\right) \times 300\,€ \approx 60\,€$$

Coste total estimado del proyecto

Sumando los dos componentes anteriores:

- **Coste del desarrollador:** 4.500 €
- **Coste del equipo (proporcional):** 60 €

Total estimado: $4{,}560\,€$

Este cálculo proporciona una visión más realista del coste asociado al desarrollo de un proyecto de investigación aplicado como este, lo cual puede resultar útil a la hora de valorar esfuerzos similares en entornos profesionales o académicos.

3.2 Gestión de riesgos y dificultades

En cualquier proyecto de investigación, especialmente aquellos relacionados con la implementación de modelos de aprendizaje automático, surge la posibilidad de enfrentarse a diversos riesgos e imprevistos que pueden influir en el éxito del proyecto, así como en su planificación temporal y presupuestaria. Por ello, se realiza en este apartado un análisis de los riesgos previstos, junto con el impacto que podrían ocasionar y las estrategias de mitigación planteadas. Además, se detallan las dificultades enfrentadas durante el desarrollo y las acciones que se implementaron para resolverlas.

3.2.1 Principales riesgos previstos

Antes de iniciar el desarrollo, se identificaron diversos riesgos relacionados con la planificación del proyecto y el manejo de las herramientas necesarias. Estos riesgos, junto con su impacto potencial y las estrategias para mitigarlos, se presentan en la tabla 3.2.

El análisis de riesgos fue un paso clave para anticipar posibles problemas en el desarrollo del trabajo y establecer medidas que pudieran minimizar sus impactos.

3.2.2 Dificultades enfrentadas y resolución

A lo largo del desarrollo del proyecto surgieron ciertos problemas que, aunque no habían sido contemplados en la planificación inicial, influyeron en el cronograma y obligaron a replantear tareas. La tabla 3.3 detalla estas dificultades junto con las soluciones aplicadas para garantizar el cumplimiento de los objetivos.

Estas dificultades fueron superadas gracias a la aplicación de estrategias flexibles y la capacidad de realizar ajustes a lo largo del proceso.

Riesgo identificado	Impacto potencial	Estrategia de mitigación
Falta de experiencia en la tecnología o herramientas requeridas para el proyecto.	Retrasos en la implementación debido a una curva de aprendizaje pronunciada.	Dedicación de tiempo inicial al aprendizaje autodidacta con recursos en línea y tutoriales, priorizando soluciones de menor complejidad.
Recursos técnicos insuficientes, como capacidad de hardware limitada para entrenar modelos.	Imposibilidad de completar experimentos de manera eficiente o resultados de baja calidad.	Utilizar servicios de cómputo en la nube o realizar pruebas con versiones reducidas de los datasets para disminuir la carga computacional.
Mal cálculo del tiempo necesario en tareas específicas.	Retrasos acumulados hacia las fases finales del proyecto.	Mantener un cronograma flexible, con márgenes específicos para las fases críticas.
Falta de claridad respecto a los objetivos del proyecto en etapas iniciales.	Cambios en la dirección del trabajo, con necesidad de rehacer tareas previas.	Coordinación regular con el tutor para revisar los avances y asegurar que las tareas cumplen con los objetivos fundacionales.
Problemas de organización personal y conciliación con otras obligaciones académicas.	Falta de dedicación suficiente al proyecto, comprometiendo la calidad o avances.	Establecimiento de horarios estrictos de trabajo dedicado al TFG, priorizando el avance progresivo frente a acumulaciones.

Tabla 3.2: Riesgos previstos en el proyecto, su impacto y estrategias de mitigación.

Dificultad	Impacto generado	Solución implementada
Dificultades técnicas en la integración de herramientas (e.g., librerías no compatibles).	Retrasos en la implementación inicial del sistema de evaluación.	Cambiar a herramientas con mayor soporte técnico y comprobar compatibilidad antes de su adopción.
Volumen elevado del dataset, causando problemas de rendimiento en el equipo.	Imposibilidad de entrenar modelos más complejos debido a limitaciones del hardware.	Utilizar muestras reducidas del dataset.
Resultados iniciales inconsistentes o de baja calidad en la evaluación de los modelos.	Necesidad de realizar ajustes repetitivos en la parametrización de los algoritmos, incrementando la carga de trabajo.	Ajustar los criterios de validación y refinar las métricas, priorizando una interpretación más clara de los resultados.
Organización del tiempo comprometida por otras asignaturas.	Retraso en ciertas entregas intermedias respecto al cronograma original.	Reajustar la planificación para dedicar sesiones semanales específicas al TFG, con objetivos parciales definidos.

Tabla 3.3: Dificultades enfrentadas durante el desarrollo y acciones de resolución.

3.2.3 Lecciones aprendidas

El desarrollo del proyecto permitió identificar una serie de lecciones clave derivadas tanto de los riesgos previstos como de las dificultades reales enfrentadas:

- La planificación debe ser flexible, con márgenes suficiente para tareas críticas, entendiendo que los tiempos iniciales suelen subestimarse.

- Dedicar tiempo al entendimiento temprano de las herramientas y procesos técnicos seleccionados puede evitar problemas en fases posteriores.
- La coordinación regular con el tutor o supervisor del TFG es esencial para alinear la dirección del proyecto y obtener un feedback constante.
- Dividir el trabajo en objetivos parciales semanales o mensuales ayuda a mantener el ritmo y reduce la acumulación de tareas hacia el final del proyecto.

En conclusión, el análisis y gestión de riesgos, junto con las estrategias implementadas para resolver dificultades, resultaron factores esenciales para llevar a buen término el proyecto dentro del marco temporal establecido y garantizando la calidad de los resultados obtenidos.

Capítulo 4

Marco Conceptual

Todo sistema, por innovador que sea, se construye sobre conceptos y tecnologías previas que lo hacen posible. Este capítulo presenta los fundamentos teóricos y técnicos que sustentan el desarrollo de WatchdogAI, ofreciendo un marco de referencia esencial para comprender su diseño y funcionamiento. En particular, se abordan los principios de la detección de intrusos, el uso del aprendizaje automático en ciberseguridad y otros elementos técnicos clave.

4.1 Sistemas de Detección de Intrusos (IDS)

Un sistema de detección de intrusos (IDS, por sus siglas en inglés) tiene como finalidad monitorizar el tráfico de red o las actividades de un sistema, con el objetivo de identificar comportamientos anómalos o potencialmente maliciosos. Este tipo de sistemas han sido ampliamente estudiados en la literatura sobre seguridad de red [30], destacando por su capacidad para identificar comportamientos maliciosos mediante diferentes enfoques de análisis. Existen dos enfoques principales:

- **Basados en firmas:** Detectan amenazas comparando el tráfico con patrones previamente identificados. Son eficaces frente a ataques conocidos, pero ineficaces ante amenazas nuevas.
- **Basados en anomalías:** Establecen un perfil de comportamiento habitual y alertan cuando se detectan desviaciones significativas. Este enfoque permite descubrir ataques novedosos, aunque puede generar una mayor tasa de falsos positivos.

WatchdogAI se enmarca dentro del enfoque basado en anomalías, incorporando técnicas de aprendizaje automático para definir y ajustar dinámicamente ese concepto de ”normalidad”, en función del entorno y de los datos observados.

4.2 Aprendizaje automático y su apliacación en ciberseguridad

El aprendizaje automático es una rama de la inteligencia artificial que se ocupa del desarrollo de algoritmos capaces de aprender a partir de datos, identificar patrones y tomar decisiones sin necesidad de instrucciones explícitas para cada caso. Su aplicación en ciberseguridad ha demostrado ser especialmente útil en contextos dinámicos, donde las amenazas son variadas y difíciles de predefinir mediante reglas estáticas.

Uno de los principales beneficios del aprendizaje automático en ciberseguridad es su capacidad de detectar amenazas desconocidas o zero-day, que no pueden ser interceptadas mediante reglas estáticas [1].

En este ámbito, permite analizar grandes volúmenes de tráfico de red para identificar comportamientos anómalos que podrían pasar inadvertidos con enfoques tradicionales. Entre sus principales ventajas destacan:

- Capacidad para detectar amenazas desconocidas (zero-day).
- Menor dependencia de reglas definidas manualmente.
- Adaptación continua a cambios en el entorno.
- Mayor rapidez y escalabilidad en la detección.

El modelo de detección desarrollado para WatchdogAI utiliza técnicas de aprendizaje supervisado, entrenadas a partir de datos etiquetados, para distinguir entre tráfico legítimo y malicioso.

4.3 Limitaciones de los sistemas convencionales

Los sistemas tradicionales de detección, basados principalmente en reglas o firmas, siguen siendo útiles en ciertos contextos, pero presentan limitaciones importantes, en particular, la inspección basada en carga útil (Deep Packet Inspection, DPI) también presenta retos de rendimiento y privacidad en entornos de alta carga, como se analiza en [13]. Su dependencia del conocimiento previo impide detectar nuevas amenazas y su efectividad en entornos complejos o cambiantes.

Además, requieren una supervisión y configuración constantes, lo que resulta difícil de sostener en redes con alto volumen de tráfico o eventos. Frente a esas limitaciones, se hace necesaria una alternativa más autónoma y flexible, como la que propone WatchdogAI, que combina técnicas modernas de análisis con capacidades de adaptación continua.

4.4 Aprendizaje supervisado

Existen distintas estrategias de entrenamiento en aprendizaje automático. En este proyecto se ha optado por el enfoque supervisado, en el que el sistema aprende a partir de un conjunto de datos etiquetado que indica si una conexión es légitima o maliciosa [2].

Este método permite obtener modelos precisos siempre que se disponga de datos representativos y equilibrados. Una vez entrenado, el modelo puede generalizar su conocimiento y clasificar nuevas conexiones en tiempo real, facilitando así una detección eficaz de amenazas.

4.5 Tipos de aprendizaje en detección de intrusos

Aunque el modelo desarrollado en WatchdogAI se basa en aprendizaje supervisado, existen otras técnicas utilizadas en este campo. El aprendizaje no supervisado permite detectar comportamientos anómalos sin necesidad de datos etiquetados, lo que resulta útil cuando no se dispone de información clasificada.

Por otro lado, al aprendizaje semi-supervisado combina una pequeña cantidad de datos etiquetados con una gran proporción de datos no etiquetados, lo que convierte en una opción atractiva en contextos donde el etiquetado manual es costoso o poco viable.

Cada enfoque presenta ventajas e inconvenientes. En este caso, se ha elegido aprendizaje supervisado por su fiablidad, capacidad de evaluación objetiva y buenos resultados en contextos controlados, lo que se ajusta a los objetivos y limitaciones del proyecto.

Capítulo 5

Soluciones Existentes

Antes de abordar el diseño y desarrollo de una investigación como WatchdogAI, resulta fundamental conocer las tecnologías y enfoques ya existentes en el ámbito de la detección de intrusos. Este capítulo presenta una revisión general de herramientas y sistemas representativos, tanto tradicionales como basados en inteligencia artificial, con el fin de contextualizar el proyecto dentro del panorama actual.

El objetivo no es ofrecer un análisis exhaustivo, sino aportar una visión comparativa que permita identificar las principales fortalezas y limitaciones de las soluciones más relevantes. De este modo, se podrá justificar con mayor claridad el enfoque adoptado en el desarrollo del sistema propuesto.

5.1 Sistemas tradicionales de detección

5.1.1 Snort

Snort [8] es uno de los sistemas de detección de intrusos (IDS) más consolidados y ampliamente utilizados. Desarrollado inicialmente por Sourcefire y actualmente mantenido por Cisco, funciona principalmente mediante detección basada en firmas. Su mecanismo consiste en comparar patrones del tráfico de red con una base de reglas predefinidas, la cual puede actualizarse para incorporar nuevas amenazas.

Una de sus principales ventajas es la posibilidad de definir reglas altamente personalizadas, lo que lo convierte en una herramienta flexible. Sin embargo, esta flexibilidad implica una fuerte dependencia del mantenimiento continuo por parte de los administradores. Aunque puede configurarse en modo inline para prevenir intrusiones (IPS), su uso más habitual es en modo pasivo, generando alertas ante posibles incidentes. Su principal limitación es la incapacidad para detectar ataques desconocidos o variantes que no coincidan con las firmas existentes.

5.1.2 Suricata

Suricata [26], desarrollado por la Open Information Security Foundation (OISF), representa una evolución moderna del enfoque de Snort. También se basa en reglas, pero incorpora mejoras técnicas importantes: permite el análisis concurrente de múltiples hilos de tráfico, es compatible con protocolos avanzados como TLS y HTTP/2, y ofrece capacidades de inspección profunda de paquetes (Deep Packet Inspection).

Además, facilita la exportación de datos en formatos estructurados como JSON, lo que mejora su integración con plataformas externas de análisis y monitoreo. Aunque no incluye capacidades de machine learning por defecto, puede conectarse con motores externos para este fin. En términos generales, proporciona mayor rendimiento y escalabilidad que Snort, pero mantiene la limitación inherente de depender de firmas estáticas.

5.2 Herramientas con enfoque de Machine Learning

5.2.1 Zeek

Zeek [33] es una plataforma de análisis de tráfico de red con amplia presencia en entornos de investigación y uso corporativo. A diferencia de Snort o Suricata, no se basa en reglas fijas, sino en políticas de análisis de eventos que permiten observar el tráfico de forma más contextual.

Esto permite generar registros detallados sobre el comportamiento de la red, los cuales pueden analizarse posteriormente mediante herramientas externas, incluyendo modelos de machine learning. Aunque no actúa como un IDS tradicional en términos de respuesta inmediata, su arquitectura modular lo convierte en una base idónea para desarrollar soluciones más avanzadas y adaptativas.

5.2.2 Herramientas académicas y experimentales

En el ámbito académico y experimental han surgido diversas herramientas, como PyIDS, centradas en aplicar técnicas de machine learning a la detección de intrusos. Estas soluciones suelen operar sobre conjuntos de datos etiquetados y emplear algoritmos como Random Forest, máquinas de soporte vectorial (SVM) o redes neuronales, entre otros.

Aunque están orientadas principalmente a entornos de prueba o simulaciones controladas, resultan esenciales para explorar nuevas metodologías y validar su eficacia. No obstante, su aplicación práctica en entornos reales suele estar limitada por restricciones en rendimiento, escalabilidad y capacidad de análisis en tiempo real.

5.3 Limitaciones comunes

A pesar de los avances tecnológicos, muchas de las soluciones actuales presentan limitaciones que afectan directamente a su efectividad en entornos operativos:

- **Dependencia de conocimiento previo:** Tanto en sistemas basados en firmas como en modelos supervisados, la necesidad de contar con datos previamente etiquetados limita su capacidad para detectar amenazas desconocidas.
- **Problemas de escalabilidad:** El creciente volumen de tráfico de red puede saturar fácilmente sistemas que no han sido diseñados para operar en tiempo real o que carecen de mecanismos de procesamiento eficiente.
- **Complejidad técnica:** Muchas herramientas requieren conocimientos especializados para su correcta instalación, configuración y mantenimiento, lo que dificulta su adopción en algunos entornos.
- **Falta de adaptabilidad:** La mayoría de los sistemas carece de mecanismos para ajustarse de forma automática a nuevas condiciones de red o a la evolución de amenazas.

En este contexto, WatchdogAI surge con el propósito de cubrir algunas de estas carencias. Sin pretender reemplazar a las herramientas existentes, propone un enfoque ligero, modular y automatizado, centrado en la detección en tiempo real mediante el aprendizaje automático, llegando a servir como base experimental para el desarrollo de de soluciones más ágiles e inteligentes, capaces de complementar los sistemas tradicionales en un entorno cada vez más complejo y dinámico.

Capítulo 6

Estudio de los datos

Antes de diseñar e implementar cualquier solución basada en inteligencia artificial, es fundamental realizar un análisis previo riguroso tanto del problema como de los datos con los que se va a trabajar. En el caso de un sistema de detección de intrusos, la calidad, variedad y estructura del conjunto de datos influyen de manera decisiva en la efectividad de los modelos empleados. Por ello, este capítulo recoge el estudio preliminar de los datasets considerados, el análisis del dataset finalmente seleccionado y la justificación de los modelos de machine learning evaluados.

Este análisis permite no solo entender mejor el contexto del problema, sino también anticipar posibles limitaciones, necesidades de preprocesamiento y decisiones clave de diseño que marcarán el desarrollo del sistema.

6.1 Descripción y comparación de los datasets

Uno de los pasos más importantes al abordar un problema de detección de intrusos mediante aprendizaje automático es la selección de un conjunto de datos adecuado. La calidad, variedad y representatividad del dataset influyen directamente en la capacidad del modelo para generalizar y detectar amenazas reales. En este proyecto se analizaron tres datasets diferentes, cada uno con características distintas, con el objetivo de elegir aquel que ofreciera el equilibrio óptimo entre realismo, complejidad y viabilidad de uso.

A continuación, se describen brevemente los tres datasets considerados:

6.1.1 CIC-IDS2017 [5]

Este conjunto de datos ha sido desarrollado por el Canadian Institute for Cybersecurity. Se trata de uno de los datasets más completos y utilizados en la literatura académica para entrenar y evaluar sistemas de detección de intrusos. Su principal fortaleza reside en que el tráfico fue generado en un entorno de red realista, con usuarios simulando actividades cotidianas (navegación web, correo electrónico, FTP, videollamadas, etc.) mientras se ejecutaban distintos tipos de ataques planificados. El tráfico está bien etiquetado, diferenciando el tráfico benigno del malicioso e identificando el tipo concreto de ataque en cada caso.

6.1.2 IDS Packet Dataset (IEEE DataPort) [19]

Este dataset, publicado en la plataforma IEEE DataPort, ofrece capturas de tráfico de red a nivel de paquetes (PCAP). Está orientado a un análisis más granular, permitiendo acceder a detalles bajos del protocolo. Si bien su

nivel de precisión puede resultar útil para sistemas que trabajan con detección muy específica, presenta algunas limitaciones: su documentación es escasa, requiere un preprocesamiento complejo para extraer características útiles, y su estructura no está tan preparada para su uso inmediato con modelos supervisados.

6.1.3 Dataset de Red Militar [21]

Este dataset simula el tráfico de red de un entorno militar, incluyendo ataques específicos y patrones de comportamiento propios de este tipo de infraestructura. Aunque resulta interesante por ofrecer un enfoque alternativo al entorno civil habitual, presenta ciertas desventajas: al ser completamente simulado, puede no generalizar bien a otros contextos reales, su variedad de ataques es limitada y no es un dataset ampliamente validado por la comunidad investigadora.

6.1.4 Justificación de la elección

A continuación, se incluye una tabla comparativa que resume las principales características de los datasets analizados:

características	**CIC-IDS2017**	**IDS Packet Dataset**	**Red Militar (Kaggle)**
Origen / Tipo de datos	Tráfico realista simulado	Capturas PCAP a bajo nivel	Simulación militar
Tamaño aprox.	~80 GB	Variable	Medio
Variedad de ataques	Alta	Media	Baja / Media
Realismo del tráfico	Alto	Medio	Bajo
Formato / Dificultad de preprocesamiento	CSVs separados por ataque	PCAP, extracción manual compleja	CSV o formato mixto
Ventajas principales	Muy completo, bien etiquetado	Análisis a nivel de paquete	Enfoque alternativo, entorno simulado
Inconvenientes principales	Tamaño elevado, requiere limpieza y balanceo	Poca documentación, extracción costosa	Poca generalización, limitado en ataques y tráfico

Tabla 6.1: Comparativa de datasets.

Tras analizar los tres conjuntos de datos, se optó por utilizar CIC-IDS2017 como base para el desarrollo del proyecto. Esta decisión se fundamenta en los siguientes motivos:

- Es uno de los datasets más utilizados y validados en investigaciones relacionadas con sistemas IDS, lo que facilita la comparación con estudios previos.
- Presenta una gran variedad de ataques y un etiquetado claro, lo que permite trabajar tanto con problemas de clasificación binaria como multiclase.
- El tráfico fue generado en condiciones realistas, simulando usuarios y comportamientos cotidianos, lo que mejora la aplicabilidad del modelo a escenarios reales.
- Aunque su tamaño y desbalanceo presentan ciertos retos técnicos, estos pueden abordarse mediante técnicas de preprocesamiento y selección de características.

Gracias a estas cualidades, el CIC-IDS2017 proporciona una base sólida para evaluar el rendimiento de distintos algoritmos de detección y comparar sus resultados de forma fiable.

6.2 Formato y estructura del dataset elegido

Una vez seleccionado el corpus de datos CIC-IDS2017 como núcleo del estudio, es fundamental comprender su estructura y formato antes de aplicar cualquier técnica de análisis o modelado. Este conocimiento previo facilita el diseño del sistema de preprocesamiento, así como la adecuación de los modelos de aprendizaje automático a los datos disponibles.

6.2.1 Estructura general

El dataset CIC-IDS2017 se distribuye originalmente en múltiples archivos CSV, donde cada archivo representa el tráfico capturado en un día concreto, asociado a un conjunto específico de ataques. Por ejemplo, uno de los ficheros puede contener ataques DDoS, otro ataques web, otro tráfico benigno, etc. Cada archivo contiene miles de muestras, siendo cada una de ellas una conexión de red representada por una serie de características estadísticas y de comportamiento.

Para facilitar su tratamiento y análisis, se puede realizar un proceso de unificación de todos los archivos CSV en un único DataFrame usando la librería Pandas de Python. Este DataFrame permite trabajar con el dataset completo de manera más eficiente y uniforme, facilitando las tareas de limpieza, transformación y modelado.

6.2.2 Volumen de datos

El dataset unificado contiene millones de registros y aproximadamente 80 características por muestra. Sin embargo, este número puede variar tras la limpieza y selección de características, como se detallará más adelante. En su estado inicial, el tamaño total del conjunto ronda los 80 GB, lo que obliga a utilizar herramientas y técnicas optimizadas para su procesamiento.

6.2.3 Tipos de datos

Las características del dataset son en su mayoría variables numéricas que describen propiedades estadísticas de las conexiones de red. Estas incluyen, entre otras:

- Duración de la conexión.
- Tamaño total de los paquetes enviados o recibidos.
- Velocidad media de transmisión.
- Conteo de paquetes o bytes hacia uno u otro sentido.
- Indicadores booleanos como flags TCP (PSH, URG, FIN...).
- Tiempos de espera o delays entre paquetes.

También hay algunas columnas con valores categóricos, como la etiqueta de clase (por ejemplo: BENIGN, Bot, DDoS, etc.), que identifica el tipo de tráfico asociado a cada muestra. Esta etiqueta es la que se utilizará como variable objetivo (y) durante el entrenamiento y evaluación de los modelos.

6.2.4 Características destacadas

Algunas de las características más relevantes y frecuentemente utilizadas en modelos de detección de intrusos incluyen:

- **Flow Duration**: tiempo total de duración de la conexión.
- **Total Fwd Packets / Total Backward Packets**: número de paquetes enviados en cada dirección.
- **Bwd Packet Length Min / Max / Mean**: estadísticas sobre el tamaño de los paquetes recibidos.
- **PSH Flag Count, URG Flag Count**: recuento de flags específicos del protocolo TCP.
- **Flow IAT (Inter Arrival Time)**: tiempos entre paquetes dentro de un mismo flujo.

Estas variables permiten capturar tanto la estructura técnica del tráfico como patrones de comportamiento que pueden diferenciar el tráfico benigno del malicioso.

6.2.5 Etiquetas de clasificación

En su forma original, el dataset permite trabajar con un enfoque multiclase, ya que contiene más de una docena de tipos de ataques distintos. Sin embargo, para facilitar ciertas pruebas y análisis, también se puede llegar a considerar un enfoque binario, agrupando todas las clases maliciosas bajo una única etiqueta `MALIGN`, frente a la clase `BENIGN`. Esta transformación permite comparar los resultados entre ambas configuraciones y explorar diferentes estrategias de detección.

6.3 Problemas detectados en los datos

Trabajar con un dataset real y de gran tamaño como CIC-IDS2017 implica enfrentarse a una serie de desafíos relacionados con la calidad, consistencia y distribución de los datos. Antes de poder entrenar modelos fiables, es necesario realizar una exploración inicial que identifique estos problemas y aplicar un proceso de limpieza riguroso que garantice la validez de los resultados.

6.3.1 Valores nulos y columnas irrelevantes

Una revisión inicial de los archivos CSV ha puesto de manifiesto la presencia de columnas con valores nulos, algunas de ellas completamente vacías o con un único valor constante. Estas variables, al no aportar información relevante ni variabilidad, podrían introducir ruido en el entrenamiento de los modelos. La existencia de valores ausentes también plantea la necesidad de decidir si se imputarán, eliminarán o tratarán mediante otro mecanismo, decisión que se tomará más adelante en función de la proporción y la importancia de cada atributo.

6.3.2 Posibles registros duplicados o inconsistentes

Dado que el dataset se compone de múltiples archivos generados en distintos días y bajo diferentes simulaciones de ataque, se sospecha que puede haber registros duplicados o inconsistentes. Aunque no se ha realizado aún una validación exhaustiva, este riesgo existe y será evaluado más adelante. Además, se han detectado valores extremos (*outliers*) en ciertas características como duración de conexiones o tamaños de paquetes, cuya interpretación no es trivial: podrían ser tanto errores de captura como muestras anómalas válidas, por lo que su tratamiento requerirá un análisis más detallado.

6.3.3 Desbalanceo en la distribución de clases

Uno de los aspectos más críticos identificados es el fuerte desbalanceo en la distribución de clases. El tráfico etiquetado como `BENIGN` constituye la mayoría abrumadora del conjunto de datos, mientras que muchas clases de ataque apenas representan una fracción mínima. Este fenómeno es habitual en datasets de ciberseguridad y puede dificultar seriamente el entrenamiento de modelos efectivos, al provocar un sesgo hacia la clase mayoritaria.

Este desbalance será uno de los principales retos a abordar en fases posteriores, tanto por su impacto en las métricas como por la necesidad de mantener una representación lo más realista posible del tráfico de red. Se prevé explorar diferentes estrategias para mitigar este problema, como la reducción de la clase mayoritaria o el reagrupamiento de clases minoritarias. [17]

6.3.4 Complejidad y heterogeneidad de los datos

El conjunto presenta una elevada dimensionalidad, con decenas de características de naturaleza diversa: algunas numéricas, otras categóricas, e incluso combinaciones que reflejan propiedades a nivel de protocolo. Esta heterogeneidad obliga a un estudio cuidadoso sobre qué variables son relevantes para el aprendizaje automático,

y cuáles podrían descartarse por redundancia o irrelevancia.

Además, muchas de las características están altamente correlacionadas entre sí, o podrían estar influenciadas por el contexto específico de la simulación que generó el tráfico. Este tipo de dependencias podría afectar negativamente a la generalización de los modelos si no se controla adecuadamente.

6.4 Modelos de Machine Learning considerados

Uno de los principales objetivos del proyecto es evaluar el rendimiento de diferentes algoritmos de aprendizaje automático aplicados a la detección de intrusos en tráfico de red. Para ello, se han seleccionado cuatro modelos representativos de distintas aproximaciones, considerando tanto su rendimiento en estudios previos como su adecuación al tipo de datos analizados.

A continuación, se describe brevemente cada uno de los algoritmos, sus características más relevantes y las razones por las que han sido considerados para su evaluación en este trabajo.

6.4.1 Bosques Aleatorios (Random Forest)

Random Forest [3] es un algoritmo de tipo *ensemble* que combina múltiples árboles de decisión entrenados sobre subconjuntos aleatorios del dataset y de las características. Su fortaleza reside en su robustez frente al *overfitting* y en su capacidad para manejar datos con muchas dimensiones sin necesidad de un preprocesamiento exhaustivo.

Además, permite extraer la importancia relativa de cada característica, lo cual es especialmente útil en contextos como el presente, donde se dispone de decenas de variables y se desea optimizar el rendimiento del modelo reduciendo la dimensionalidad.

6.4.2 Máquina de Vectores de Soporte (SVM)

Las *Support Vector Machines* [9] son modelos supervisados que intentan encontrar el hiperplano que mejor separa las clases en el espacio de características. Aunque pueden ofrecer muy buen rendimiento en datasets con pocas muestras o en problemas linealmente separables, su uso presenta algunas limitaciones en este contexto:

- Requieren escalado de los datos.
- No están pensadas para manejar grandes volúmenes de datos.
- El tiempo de entrenamiento puede ser muy elevado en conjuntos amplios como `CIC-IDS2017`.

Por ello, su uso se ha reservado a versiones más reducidas y balanceadas del dataset, con el objetivo de comparar su rendimiento en escenarios controlados.

6.4.3 XGBoost

XGBoost (*Extreme Gradient Boosting*) [7] es un algoritmo basado en *boosting* de árboles de decisión que ha demostrado ser altamente eficaz en una amplia variedad de competiciones y estudios de *machine learning*. Ofrece numerosas ventajas:

- Entrenamiento eficiente y rápido.
- Regularización integrada para evitar *overfitting*.
- Tolerancia a valores nulos.

- Posibilidad de ajustar múltiples hiperparámetros.

Gracias a su capacidad para trabajar con datos tabulares complejos y su buen rendimiento incluso en contextos con desbalanceo, *XGBoost* se ha convertido en uno de los candidatos principales en este proyecto. Tal como se detallará en capítulos posteriores, ha sido uno de los modelos que mejores resultados ha obtenido.

6.4.4 Multilayer Perceptron (MLP)

El *MLP* [16] es una red neuronal de tipo *feedforward* compuesta por varias capas de neuronas. Aunque no tan sofisticado como otras arquitecturas de *deep learning*, permite modelar relaciones no lineales complejas y generalizar bien si se entrena adecuadamente.

En este caso, se ha explorado su uso sobre versiones más pequeñas y balanceadas del dataset, dado que su entrenamiento puede resultar costoso en términos de tiempo y recursos computacionales. Además, requiere normalización de los datos y un cuidado especial para evitar problemas de sobreajuste.

En resumen, la selección de modelos busca cubrir diferentes enfoques y niveles de complejidad, desde algoritmos clásicos como *Random Forest* hasta técnicas más avanzadas como *XGBoost* y redes neuronales. Esta variedad permitirá realizar una comparativa objetiva entre sus rendimientos, analizando no solo la precisión, sino también su capacidad de generalización, su tiempo de entrenamiento y su comportamiento frente a clases desbalanceadas.

Capítulo 7

Diseño

El diseño del sistema constituye una de las fases más relevantes de este proyecto, ya que marca la hoja de ruta técnica que guiará todo el proceso de desarrollo y evaluación. A partir del análisis previo, se definen los módulos funcionales y el flujo de trabajo necesarios para construir un sistema que permita comparar distintos algoritmos de aprendizaje automático en la tarea de detección de intrusiones.

Este capítulo presenta la arquitectura general del sistema, el diseño del *pipeline* de datos, las decisiones adoptadas sobre los modelos de *machine learning* y los criterios definidos para su evaluación. Todo ello con el objetivo de garantizar un diseño reproducible, flexible y orientado a obtener resultados fiables que permitan identificar el mejor enfoque para este tipo de problemas.

7.1 Arquitectura general del sistema

La arquitectura del sistema se ha planteado con una estructura modular y flexible, que facilita tanto el desarrollo progresivo como la evaluación independiente de cada componente. Cada fase del proceso ha sido diseñada para ser lo más desacoplada posible del resto, permitiendo modificar o sustituir partes concretas sin afectar al conjunto.

A grandes rasgos, el sistema se compone de los siguientes bloques funcionales:

- **Preprocesamiento de datos**: Incluye la carga del dataset, limpieza de columnas innecesarias o vacías, tratamiento de valores nulos, eliminación de duplicados y normalización si es necesaria. También se aplica el balanceo de clases en esta etapa.
- **Entrenamiento de modelos**: Se encargan de aplicar los algoritmos seleccionados (*Random Forest*, *SVM*, *XGBoost* y *MLP*) sobre distintas versiones del dataset: completo, balanceado, reducido, binario y multiclase.
- **Evaluación de modelos**: Tras el entrenamiento, se analizan las métricas obtenidas para cada modelo (precisión, *recall*, *F1-score*, matriz de confusión), tanto en escenarios algo más balanceados como reducidos.
- **Selección de características** [4]: Una vez entrenados los primeros modelos, se analiza la importancia relativa de cada característica y se repite el entrenamiento con las más relevantes, con el objetivo de mejorar el rendimiento y reducir la complejidad.

El proceso de diseño ha sido iterativo: algunas fases, como la selección de características, se retroalimentan de los resultados de la evaluación para afinar modelos posteriores. Esta arquitectura favorece la experimentación controlada y la trazabilidad de los cambios.

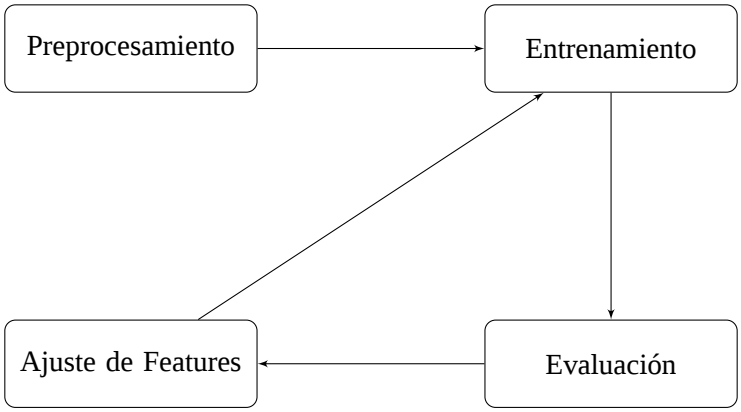


Figura 7.1: Diagrama de bloques para el flujo de trabajo.

7.2 Diseño del pipeline de datos

El *pipeline* [11] de datos constituye la columna vertebral del sistema, ya que define cómo se procesan y transforman los datos desde su estado original hasta su uso final en el entrenamiento de los modelos. Su diseño ha seguido un enfoque estructurado y progresivo, permitiendo adaptarse a diferentes enfoques experimentales sin comprometer la coherencia del análisis.

El flujo de trabajo del *pipeline* se compone de las siguientes etapas:

- **Carga y unificación del dataset:** El dataset CIC-IDS2017 se proporciona en múltiples archivos CSV, cada uno correspondiente a un tipo de ataque o tráfico. La primera fase del *pipeline* consiste en leer todos estos archivos y combinarlos en un único *DataFrame* unificado. Esta operación permite tratar el conjunto de datos como una sola entidad homogénea, simplificando las fases posteriores.
- **Limpieza de datos:** Una vez unificado, se realiza una limpieza profunda del conjunto de datos. Se eliminan columnas completamente vacías, con valores constantes, o que no aportan valor informativo (por ejemplo, *timestamps* redundantes o identificadores irrelevantes). También se eliminan duplicados y se gestionan los valores nulos para evitar errores durante el entrenamiento.
- **Análisis y transformación de etiquetas:** El dataset original incluye múltiples clases que representan tipos concretos de ataques. Se definieron varias estrategias de agrupación de etiquetas para realizar análisis comparativos:
  - **Clasificación binaria:** Se agrupan todas las clases no benignas bajo una etiqueta común `MALIGN`.
  - **Clasificación multiclase original:** Se mantienen todas las clases de ataque por separado.
  - **Multiclase con agrupación de ataques web:** Se agrupan ataques como *Brute Force*, *SQL Injection* y *XSS* en una clase común `Web Attack`, para mejorar la estabilidad del modelo y reducir la confusión entre clases similares.
- **Balanceo del dataset:** Dado el fuerte desbalanceo presente en el conjunto de datos —con la clase `BENIGN` representando la mayoría de las muestras—, se aplica un *undersampling* de esta clase para obtener un conjunto de datos más equilibrado. También se construyen versiones reducidas del dataset, limitando el

número de muestras por clase, para permitir el entrenamiento de modelos más exigentes computacionalmente (como *SVM* o *MLP*) sin necesidad de grandes recursos.

- **Gestión de versiones del dataset:** Todas las variantes generadas (original, balanceada, reducida, binaria, multiclase, con agrupaciones) se gestionan de forma controlada, permitiendo reutilizarlas según las necesidades de cada experimento. Esto facilita la comparación directa entre modelos entrenados en distintas condiciones.

Este diseño modular del *pipeline* de datos permite realizar modificaciones sobre una etapa concreta (por ejemplo, probar otra técnica de balanceo) sin tener que rehacer todo el proceso. Además, contribuye a asegurar la trazabilidad y reproducibilidad del sistema en su conjunto.

En la Figura 7.2 se representa el flujo general de procesamiento de datos definido para el sistema. Este *pipeline* abarca desde la fase inicial de carga y unificación de los archivos CSV originales del dataset, pasando por la limpieza de columnas irrelevantes o vacías, hasta la transformación de etiquetas y la aplicación de técnicas de balanceo.

A partir de este flujo base, se han generado distintas versiones del conjunto de datos —adaptadas para distintos modelos o enfoques de clasificación—, manteniendo una estructura modular que permite reproducir y escalar fácilmente el proceso. Esta organización ha sido clave para evaluar el impacto de cada etapa en el rendimiento final de los modelos.

La Tabla 7.1 muestra un resumen de las distintas versiones del dataset generadas durante el desarrollo del proyecto. Cada una ha sido creada con un propósito específico, ya sea mejorar la distribución de clases, reducir el tamaño del dataset para facilitar la experimentación con modelos más costosos computacionalmente o evaluar distintos esquemas de clasificación (binaria, multiclase o agrupada).

Esta estrategia ha permitido comparar modelos de forma más justa y estudiar cómo varía su rendimiento en función del volumen de datos y del tipo de etiquetado utilizado.

Versión del dataset	Nº muestras totales	Nº clases	Tipo de clasificación	Balanceado	Usado con modelos
Original completo	~2M+	15	Multiclase	No	RF, XGB
Balanceado (undersample)	~830k	2/15	Binaria/Multiclase	Sí	RF, XGB
Agrupado (Web Attacks)	~830k	13	Multiclase agrupada	Sí	RF, XGB
Dataset reducido	~60k	13	Multiclase agrupada	Sí	SVM, MLP
Dataset reducido 2	~78k	13	Multiclase agrupada	Sí	SVM, MLP, XGB
Dataset reducido 3	~100k	13	Multiclase agrupada	Sí	SVM, MLP

Tabla 7.1: Descripción de las versiones del dataset

7.3 Diseño de los modelos de Machine Learning

La selección y configuración de los algoritmos de *machine learning* es un elemento central en el diseño del estudio. A partir del análisis previo realizado en el capítulo anterior, se han identificado cuatro modelos que, por sus características, se consideran adecuados para abordar tanto la clasificación binaria (tráfico benigno vs malicioso) como la multiclase (diferentes tipos de ataques).

Más allá de su elección conceptual, esta sección plantea las decisiones iniciales en cuanto a su configuración y uso previsto dentro del *pipeline* de detección, dejando para el capítulo de implementación los ajustes finales y la evaluación de su rendimiento.

7.3.1 Modelos seleccionados y configuración inicial

Los modelos seleccionados y su configuración inicial prevista son los siguientes:

- **Random Forest (RF)**: Se configura con el parámetro `class_weight='balanced'` para abordar el desbalanceo de clases, aprovechando su capacidad para trabajar directamente con datos sin necesidad de normalización. Se planea ajustar posteriormente parámetros como el número de árboles (`n_estimators`) y la profundidad máxima (`max_depth`), en función de los resultados obtenidos en las primeras pruebas.
- **XGBoost**: Se configura con su esquema por defecto, incluyendo parámetros de regularización y manejo de valores nulos. Posteriormente se evaluará la conveniencia de ajustar hiperparámetros como el *learning rate*, la profundidad de los árboles o el número de *boosting rounds*. Este modelo es especialmente prometedor por su rendimiento en problemas con clases desbalanceadas y por su capacidad de análisis de importancia de características.
- **Multilayer Perceptron (MLP)**: Para este modelo, se considera una arquitectura simple con una o dos capas ocultas, activación *ReLU* y salida *softmax* para la clasificación multiclase. Se utilizará *early stopping* y técnicas de regularización para evitar el sobreajuste. Dada su sensibilidad a la escala de los datos, se normalizarán las características de entrada.
- **Support Vector Machine (SVM)**: Se prevé el uso de un *kernel* lineal, por su menor coste computacional y buen comportamiento en datasets preprocesados. El entrenamiento se limitará inicialmente a conjuntos de datos reducidos y balanceados, dado que *SVM* no escala bien con grandes volúmenes de muestras. Será necesaria una normalización previa de los datos.

7.3.2 Estrategia de entrenamiento y validación prevista

En la fase de diseño se plantea adoptar una división clásica del conjunto de datos en entrenamiento y prueba, utilizando un *train-test split* del 80 %-20 %. Esta aproximación permitirá evaluar inicialmente la capacidad de generalización de los modelos con bajo coste computacional. Además, se considera la posibilidad de aplicar validación cruzada (*k-fold cross-validation*) [22] en los modelos que muestren mejores resultados preliminares, para garantizar una mayor estabilidad en la evaluación.

Asimismo, se mantendrá un enfoque coherente en todas las pruebas para asegurar que los modelos comparados trabajen sobre los mismos datos de entrada y bajo métricas homogéneas, evitando cualquier tipo de fuga de datos del conjunto de prueba hacia el entrenamiento.

La búsqueda de hiperparámetros óptimos se realizará de forma manual y acotada, ya que el objetivo principal del proyecto no es alcanzar el mejor rendimiento absoluto de cada modelo, sino identificar cuál de ellos resulta ser el más adecuado dentro de un entorno realista de detección de intrusiones.

7.4 Diseño de la evaluación

La evaluación de los modelos es una parte crítica del sistema, ya que no solo permite medir el rendimiento de las distintas técnicas de detección, sino que también proporciona una base objetiva para comparar enfoques y seleccionar la solución más adecuada al problema planteado. Esta sección establece las métricas que se emplearán, los criterios de comparación entre modelos y la estrategia general de evaluación.

7.4.1 Métricas seleccionadas

Dado que el problema de detección de intrusiones suele implicar clases desbalanceadas y la existencia de errores con costes diferentes (por ejemplo, pasar por alto un ataque puede ser más grave que clasificar erróneamente un tráfico benigno), se han seleccionado las siguientes métricas [10]:

- **Precisión (Precision)**: Proporción de predicciones positivas correctas. Es útil para evaluar cuántos de los eventos clasificados como ataques lo eran realmente.
- **Exhaustividad o Recall**: Mide cuántos de los eventos realmente maliciosos fueron detectados. Es especialmente relevante en sistemas IDS, donde no detectar un ataque puede tener consecuencias críticas.
- **F1-score**: Media armónica entre precisión y *recall*, utilizada como métrica principal en este proyecto, ya que ofrece un balance entre ambas y penaliza los extremos.
- **Matriz de confusión**: Herramienta visual que permite analizar los aciertos y errores por clase. Resulta clave para entender cómo se comporta el modelo ante clases minoritarias o similares entre sí.

Estas métricas se calcularán tanto para los modelos entrenados sobre el conjunto binario (`BENIGN` vs `MALIGN`) como para los modelos multiclase (diferentes tipos de ataque), permitiendo observar cómo se degradan o mejoran los resultados según la granularidad del problema.

7.4.2 Criterios para la comparación entre modelos

Para asegurar una comparación justa, todos los modelos serán entrenados y evaluados bajo las mismas condiciones: mismo conjunto de entrenamiento y prueba, mismos datos preprocesados y mismas métricas de evaluación. Las pruebas se realizarán con varios enfoques:

- **Dataset completo** con clase `BENIGN` predominante (enfoque realista).
- **Dataset balanceado** mediante *undersampling* (enfoque experimental).
- **Dataset reducido** para modelos más sensibles a la escala como *MLP* o *SVM*.

La decisión final sobre el mejor modelo se basará en un equilibrio entre rendimiento (especialmente *F1-score* en clases minoritarias), tiempo de entrenamiento, simplicidad del modelo y capacidad de explicación. No se priorizará únicamente la métrica global, sino también la robustez del modelo ante cambios en la distribución del tráfico.

En este sentido, se consideran especialmente valiosas las métricas por clase en escenarios multiclase, ya que permiten valorar si el modelo falla sistemáticamente en detectar ciertos tipos de ataques, como los de tipo *Web* o *Botnet*, que tienden a presentar patrones menos diferenciados.

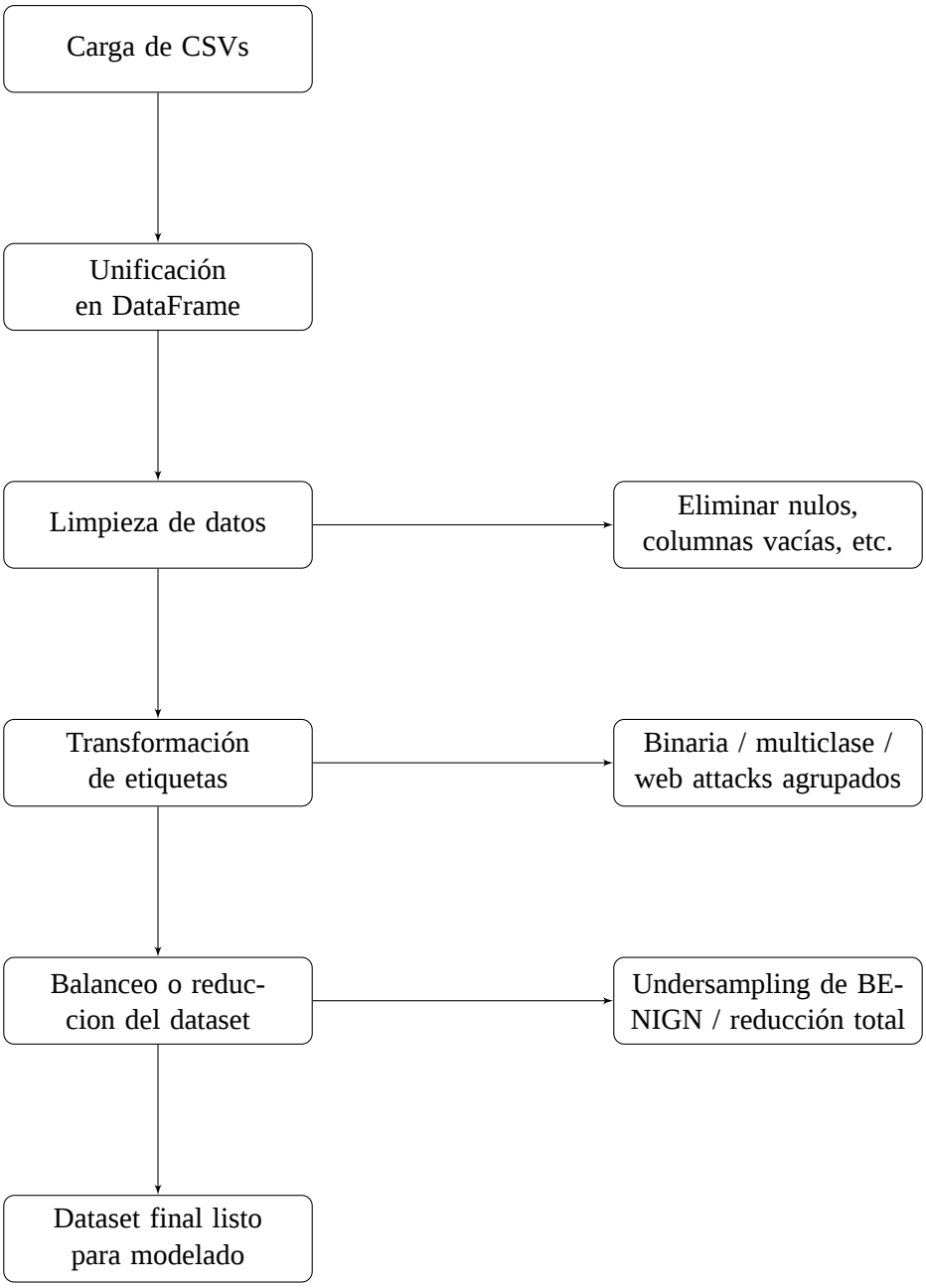


Figura 7.2: Diagrama de flujo del procesamiento del dataset

Capítulo 8

Implementación

Este capítulo describe en detalle el proceso de implementación técnica del sistema propuesto, abarcando desde la configuración del entorno de desarrollo hasta la construcción y evaluación de los modelos de aprendizaje automático. Se ha adoptado un enfoque modular y reproducible, con el objetivo de facilitar tanto la experimentación como la futura ampliación del proyecto.

A lo largo del capítulo se detallan las decisiones técnicas adoptadas, las funciones desarrolladas para automatizar tareas recurrentes, y las estrategias empleadas para abordar desafíos como el desbalanceo de clases o la alta dimensionalidad del dataset. Asimismo, se presentan las principales librerías utilizadas, justificando su elección en función de su robustez, versatilidad y adecuación al problema de detección de intrusiones mediante inteligencia artificial.

8.1 Entorno y herramientas

Para la implementación de este proyecto se ha utilizado **Python 3.11.9** [28] como lenguaje principal, debido a su popularidad en el campo de la ciencia de datos, su extensa comunidad y la gran disponibilidad de librerías especializadas en procesamiento de datos, *machine learning* y visualización.

Todo el desarrollo se ha realizado mediante *Jupyter Notebook* [20] ejecutado en local a través de *Visual Studio Code* [25], lo que ha permitido una exploración interactiva de los datos y una documentación en línea del flujo de trabajo.

El entrenamiento de los modelos se ha realizado exclusivamente en **CPU**, sin necesidad de utilizar entornos con aceleración por *GPU*, gracias a la reducción progresiva del volumen de datos y a una gestión eficiente de los recursos disponibles.

Además, para la elaboración de la memoria del proyecto se ha empleado LaTeX[23], un sistema de composición tipográfica ampliamente utilizado en entornos académicos y científicos, que permite generar documentos técnicos con gran calidad de presentación y control sobre el formato.

8.1.1 Equipo utilizado

Las pruebas y entrenamientos se han realizado en un equipo con las siguientes especificaciones:

- **Sistema operativo:** Windows 11 (64 bits)

- **Procesador:** Intel Core Ultra 5 135U @ 4.40GHz
- **Memoria RAM:** 16 GB DDR5 @ 5600Mt/s
- **Almacenamiento:** KIOXIA NVMe SSD 239 GB @ PCIe
- **GPU:** Intel Integrated Graphics (no utilizada en este proyecto)

Este entorno ha sido suficiente para ejecutar todos los experimentos de entrenamiento y evaluación sin requerir recursos de computación en la nube ni hardware especializado.

8.1.2 Principales librerías utilizadas

Entre las principales librerías utilizadas en el proyecto destacan:

- **Pandas** [31]: Para la carga, manipulación y análisis de estructuras de datos en formato tabular (*DataFrames*), así como para la unificación y limpieza del dataset.
- **NumPy** [15]: Para operaciones numéricas de bajo nivel y soporte a estructuras como *arrays* y matrices.
- **Matplotlib** [18] y **Seaborn** [32]: Utilizadas para la generación de gráficos y visualizaciones que faciliten la comprensión de los datos y los resultados obtenidos.
- **Scikit-learn** [27]: Librería central para el desarrollo de modelos de *machine learning*, incluyendo algoritmos de clasificación, herramientas de preprocesamiento, partición de datos y métricas de evaluación.
- **XGBoost**: *Framework* especializado en técnicas de *boosting*, conocido por su alto rendimiento en tareas de clasificación estructurada.
- **Imbalanced-learn** [24]: Librería utilizada para explorar técnicas de balanceo de clases, incluyendo algoritmos como *SMOTE* [6].

Estas herramientas han permitido construir un flujo de trabajo robusto, reproducible y alineado con las mejores prácticas en el campo de la detección de intrusiones basada en inteligencia artificial.

8.2 Preprocesamiento

Para facilitar la tarea del preprocesamiento y estructurar el flujo de los datos, se ha implementado un conjunto de funciones reutilizables que permiten llevar a cabo las distintas operaciones de limpieza, transformación y reducción necesarias antes del entrenamiento de los modelos.

A continuación, se detallan las principales funciones de preprocesamiento desarrolladas:

- **cargar_csvs(ruta_csvs)**: Esta función se encarga de recorrer de forma automática todos los archivos `.csv` presentes en la ruta especificada y leerlos utilizando la librería `pandas`. Cada archivo se carga en un *DataFrame* independiente, y finalmente todos se concatenan en un único *DataFrame* principal que aglutina todo el contenido del dataset. Durante la carga, se imprime por pantalla información útil como el nombre del archivo procesado y el número de filas que contiene. Esta función permite escalar fácilmente la carga de datos sin tener que especificar manualmente cada archivo.
- **limpiar_dataset(df)**: Una vez cargado el dataset, esta función limpia los datos eliminando columnas completamente vacías o constantes (es decir, sin variabilidad), así como cualquier fila que contenga valores nulos o duplicados. También se sustituyen los valores infinitos por nulos para garantizar la estabilidad de los modelos. Con esta limpieza, se mejora tanto la calidad de los datos como la eficiencia del entrenamiento posterior.

- **balancear_dataset(df)**: El conjunto de datos original está fuertemente desbalanceado, con una gran cantidad de tráfico benigno frente a un número significativamente menor de muestras maliciosas. Para contrarrestar este efecto, se implementa esta función, que realiza un *undersampling* de la clase `BENIGN`. En concreto, se iguala el número de muestras benignas al de las muestras maliciosas, y posteriormente se barajan todas las filas para evitar cualquier sesgo por orden.

- **convertir_binario(df)**: Esta función permite transformar el dataset en un problema de clasificación binaria. Para ello, todas las etiquetas distintas de `BENIGN` se convierten en la etiqueta común `MALIGN`, lo que resulta útil para enfoques iniciales donde se busca simplemente diferenciar entre tráfico normal y ataque.

- **agrupar_web_attacks(df)**: Dada la escasa representación de algunos tipos de ataques web (como *XSS* o *SQL Injection*), se opta por agruparlos en una única clase común denominada `Web Attack`. Esta función realiza dicha agrupación reemplazando las etiquetas originales por una nueva etiqueta genérica. Esta transformación mejora significativamente el rendimiento de los modelos frente a esta familia de ataques.

- **reducir_dataset(df, etiqueta_col=' Label', limites={})**: En ciertos experimentos se trabaja con versiones reducidas del dataset para facilitar las pruebas o mitigar el coste computacional. Esta función permite limitar el número de muestras por clase de forma flexible, según un diccionario de límites proporcionado como argumento. Así, se puede generar un subconjunto representativo y más equilibrado del dataset, manteniendo control sobre cada clase.

Todas estas funciones se encuentran en la Sección A.1 del anexo.

8.2.1 Funciones auxiliares para validación

Para verificar que cada uno de los datasets generados tras aplicar estas funciones es correcto y equilibrado, se han implementado dos funciones auxiliares que permiten analizar la distribución de clases de forma visual y tabular:

- **mostrar_grafica_distribucion(df, titulo_grafica)**: Genera un gráfico de barras con la cantidad de muestras por clase en el *DataFrame* proporcionado. Es útil para evaluar visualmente el grado de desbalanceo o comprobar si una transformación como el agrupamiento o la reducción ha surtido efecto.

- **mostrar_tabla_distribucion(df)**: Muestra en pantalla una tabla con la frecuencia exacta de cada clase. Esto permite complementar la gráfica anterior con datos cuantitativos precisos.

Ambas funciones también se encuentran disponibles en la Sección A.5 del anexo y han sido utilizadas repetidamente a lo largo del desarrollo para validar el resultado de cada transformación intermedia.

8.3 Entrenamiento y selección de características

Una vez preprocesados los datos, el siguiente paso consiste en preparar los conjuntos de entrenamiento y prueba, así como entrenar los distintos modelos de aprendizaje automático que se evaluarán posteriormente. Además, se ha implementado un proceso de extracción de características relevantes con el objetivo de reducir la dimensionalidad del conjunto de datos sin perder rendimiento predictivo, aunque no se usará hasta después de evaluar los modelos. Para automatizar estas tareas y permitir una comparación más justa entre modelos, se han desarrollado funciones específicas para cada modelo y etapa del flujo.

8.3.1 Entrenamiento

Con el fin de estructurar correctamente el entrenamiento de los modelos, se ha creado una serie de funciones que encapsulan tanto la preparación de los datos como el proceso de ajuste del modelo:

- **preparar_datos(df)** Esta función divide el *DataFrame* original en dos subconjuntos:
  - **X**: que contiene todas las características (todas las columnas excepto la de la etiqueta).
  - **y**: que contiene únicamente la columna de etiquetas, la cual debe ser transformada a formato numérico utilizando un `LabelEncoder`.

  Posteriormente, ambos conjuntos se dividen en entrenamiento y prueba mediante `train_test_split`, utilizando un 80 % de los datos para entrenamiento y el 20 % restante para prueba, como se especificó en la planificación de pruebas del capítulo 9.

- **entrenar_random_forest(X_train, y_train)** Esta función entrena un modelo *Random Forest* con los parámetros estándar:
  - 100 árboles (`n_estimators=100`).
  - Equilibrio automático de clases mediante `class_weight='balanced'`.
  - Reproducibilidad asegurada con `random_state=42`.
  - Ejecución en paralelo en todos los núcleos con `n_jobs=-1`.

  Esta versión se utiliza principalmente para combatir el desbalanceo entre clases.

- **entrenar_random_forest_sin_balancear(X_train, y_train)** Variante de la anterior en la que se omite el parámetro `class_weight`, permitiendo evaluar el impacto que tiene este ajuste en la clasificación de clases minoritarias.

- **entrenar_xgboost(X_train, y_train)** Entrena un modelo *XGBoost* adaptado automáticamente al tipo de clasificación:
  - Para clasificación binaria, usa `objective='binary:logistic'` y `eval_metric='logloss'`.
  - Para multiclase, usa `objective='multi:softmax'` y `eval_metric='mlogloss'`.

  También se configura `use_label_encoder=False` para evitar advertencias innecesarias, se establece la semilla con `random_state=42`, y se optimiza el rendimiento con `n_jobs=-1`.

- **entrenar_svm(X_train, y_train, C=1.0, gamma='scale', kernel='linear')** Entrena un modelo de *Support Vector Machine*, previamente escalando los datos con `StandardScaler`. Por defecto, se utiliza un *kernel* lineal, con `C=1.0` y `gamma='scale'`. Es posible ajustar estos parámetros para evaluar diferentes configuraciones. La aleatoriedad se controla con `random_state=42`.

- **entrenar_mlp(X_train, y_train, hidden_layer_sizes=(100,), max_iter=300, alpha=0.0001)** Entrena una red neuronal de tipo *Multilayer Perceptron (MLP)* con una única capa oculta de 100 neuronas como valor por defecto.
  - Se escalan previamente los datos.
  - Se emplea función de activación *ReLU* y optimizador *Adam*.
  - Se habilita la parada temprana (`early_stopping=True`) para evitar sobreajuste.
  - Se limita el entrenamiento a 300 iteraciones y se fija la semilla para reproducibilidad.

Estas funciones han sido utilizadas de manera sistemática a lo largo del proceso experimental, asegurando consistencia entre los diferentes entrenamientos y permitiendo centrarse en la comparación de resultados. Se encuentran en la Sección A.2 del Anexo.

8.3.2 Extracción de características

Para reducir el número de características del conjunto de datos sin comprometer el rendimiento del modelo, se ha realizado una fase de extracción de características basada en la importancia asignada por *XGBoost*. Este modelo permite obtener métricas internas que indican qué atributos contribuyen más a las decisiones del algoritmo.

- **obtener_importancias(modelo, tipo="gain")** Esta función accede directamente al *booster* del modelo *XGBoost* para obtener la importancia de cada característica según el criterio especificado (`gain` por defecto, aunque también permite `weight`, `cover`, etc.). A continuación, convierte la información en un *DataFrame* ordenado de mayor a menor importancia, permitiendo seleccionar las características más relevantes.

- **grafica_importancia_caracteristicas(modelo, max_features=20)** Genera una visualización de las características más importantes del modelo entrenado utilizando `xgb.plot_importance`, centrada en las `max_features` más relevantes. Esta gráfica facilita la interpretación visual de los atributos que tienen mayor peso en el rendimiento del modelo, lo cual ha sido de gran utilidad en la sección 9.2, donde se han realizado pruebas específicas con las 20 y 30 características más significativas.

El código de estas dos funciones se encuentran en la Sección A.4 del anexo

8.4 Evaluación

Para evaluar el rendimiento de los modelos entrenados se ha desarrollado un único método que automatiza tanto el cálculo de métricas como la visualización de resultados. Esta evaluación se realiza utilizando el conjunto de prueba (20 % del total de muestras), previamente separado en la fase de entrenamiento de modelos. Siendo este el siguiente:

evaluar_modelo(modelo, X_test, y_test, label_encoder, titulo="Evaluación", tamaño=(18, 12), cmap='Purples')

Este método realiza la evaluación completa del modelo entrenado a partir del conjunto de datos de prueba. En primer lugar, predice las etiquetas correspondientes a las muestras de `x_test` y compara estas predicciones con las etiquetas reales, `y_test` para generar un informe detallado utilizando la función `classification_report()` de `scikit-learn`. Este informe incluye métricas como *precision*, *recall*, *f1-score* y soporte por clase.

Además, se genera la matriz de confusión con la función `confusion_matrix()` y se representa visualmente mediante `matplotlib` con un mapa de calor (*heatmap*), lo cual permite identificar de forma rápida los errores de clasificación más frecuentes. La visualización puede personalizarse mediante los parámetros `tamaño` (para ajustar el tamaño de la figura) y `cmap` (para modificar la paleta de colores del gráfico). La decodificación de etiquetas se realiza mediante `label_encoder.classes_` para mostrar los nombres reales de las clases en lugar de índices numéricos.

Esta función ha sido utilizada de manera uniforme en todos los experimentos y pruebas presentadas en el siguiente capitulo, lo que garantiza la coherencia y comparabilidad entre modelos y configuraciones de dataset.
El código fuente de esta función puede encontrarse en el anexo, Sección A.3

8.5 Organización y gestión del código

Durante el desarrollo del sistema se ha prestado especial atención a la organización y reutilización del código, con el objetivo de facilitar tanto la implementación como las posteriores pruebas y análisis. Como se ha

mencionado antes, todo el desarrollo se ha realizado en *Jupyter Notebooks*, estructurando el flujo de trabajo en bloques diferenciados para cada etapa: carga y limpieza de datos, preprocesamiento, entrenamiento de modelos, evaluación y análisis de resultados.

Para mejorar la legibilidad y modularidad, todas las funciones desarrolladas se agrupan al inicio de los *notebooks* en secciones claramente delimitadas. Estas funciones encapsulan la lógica de tareas recurrentes como cargar datos, limpiar el dataset, entrenar modelos o evaluar métricas, lo que ha permitido reutilizarlas de forma eficiente a lo largo del proyecto.

Además, se ha hecho uso de variables de configuración para poder cambiar fácilmente parámetros como los límites de reducción, el tipo de modelo a entrenar o la ruta a los CSV originales, sin necesidad de modificar múltiples líneas de código.

Aunque no se ha dividido el código en archivos *.py* por la naturaleza exploratoria del entorno *Jupyter*, se ha seguido una lógica similar a la modularización clásica, lo que ha contribuido a mantener un flujo de trabajo ordenado y controlado.

Por último, se ha utilizado **Git** [14] como sistema de control de versiones, permitiendo registrar los avances del proyecto, recuperar versiones anteriores del código en caso necesario y gestionar distintas ramas para pruebas específicas.

Capítulo 9

Resultados

A lo largo de este capítulo se presentan los resultados obtenidos tras entrenar y evaluar los distintos modelos de aprendizaje automático seleccionados. El objetivo principal de esta fase es comprobar la eficacia del sistema propuesto a la hora de detectar intrusiones, así como analizar cómo influyen diferentes configuraciones del dataset y técnicas de preprocesamiento en el rendimiento de los clasificadores.

9.1 Planificación de las pruebas

Una vez finalizado el diseño y la implementación de los distintos modelos, se llevó a cabo una batería de pruebas con el objetivo de evaluar su rendimiento en diferentes escenarios de clasificación. Estas pruebas sirven como cierre al ciclo de desarrollo, permitiendo validar las decisiones tomadas durante el preprocesamiento, la selección de modelos y la estrategia de entrenamiento.

El conjunto de pruebas se ha construido en base al diseño previamente descrito en capitulos anteriores, utilizando distintas versiones del dataset previamente generadas:

- **Dataset balanceado** mediante *undersampling* de la clase `BENIGN` con distintos enfoques:
  - **Binario** únicamente dos clases `BENIGN` y `MALIGN` para tener un balanceo mucho mayor.
  - **Multiclase** todas las clases del dataset original para ver como se comportaba en las menos significativas.
  - **Agrupado** agrupando las clases de web attack en una sola para ver si mejora el funcionamiento.
- **Dataset reducido**, con un número limitado de muestras por clase para facilitar el entrenamiento con modelos más exigentes.
- **Características reducidas**, para buscar un equilibrio entre el número de características y el rendimiento del modelo.

9.2 Resultados obtenidos

En esta sección se recogen los resultados obtenidos tras la evaluación de los distintos modelos entrenados en cada uno de los enfoques considerados. Las métricas utilizadas han sido precisión (*precision*), exhaustividad (*recall*), puntuación F1 (*f1-score*) y número de muestras por clase (*support*), calculadas mediante la función `classification_report()` de `scikit-learn`. Asimismo, se ha generado una tabla con los aciertos

y fallos por clase a partir de la matriz de confusión correspondiente a cada modelo.

Estas matrices de confusión se encuentran en el Capitulo C del anexo, junto al resto de gráficas y visualizaciones complementarias.

Aunque el enfoque principal de esta sección es la presentación de los resultados, también se ha considerado el coste computacional de los modelos, entendido como el tiempo de entrenamiento. Este parámetro resulta especialmente relevante en los modelos más complejos como *SVM* o *MLP*, y se comentará de forma general en esta sección y en mayor profundidad en el apartado de análisis posterior.

9.2.1 Dataset binario

Para este primer enfoque se ha convertido el problema en una clasificación binaria, agrupando todas las clases de ataque en una sola clase `MALIGN`. Los modelos evaluados han sido *Random Forest* y *XGBoost* sobre el dataset balanceado.

Random Forest Binario

Informe de clasificación - Clasificación RF Binaria

	precision	recall	f1-score	support
BENIGN	1.00	1.00	1.00	85149
MALIGN	1.00	1.00	1.00	85148
accuracy			1.00	170297
macro avg	1.00	1.00	1.00	170297
weighted avg	1.00	1.00	1.00	170297

Figura 9.1: Informe Random Forest Binario.

XGBoost Binario

Informe de clasificación - Clasificación XGBoost Binaria

	precision	recall	f1-score	support
BENIGN	1.00	1.00	1.00	85149
MALIGN	1.00	1.00	1.00	85148
accuracy			1.00	170297
macro avg	1.00	1.00	1.00	170297
weighted avg	1.00	1.00	1.00	170297

Figura 9.2: Informe XGBoost Binario.

Clase	Aciertos	Fallos
BENIGN	85024	125
MALIGN	85037	111

Tabla 9.1: Aciertos/fallos Random Forest Binario.

Clase	Aciertos	Fallos
BENIGN	85028	121
MALIGN	85130	18

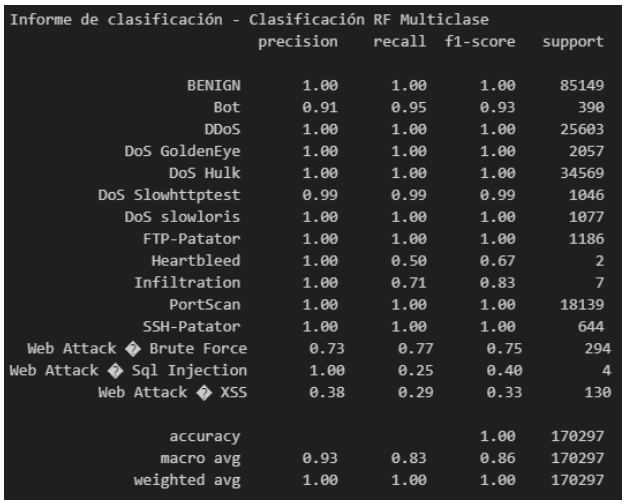
Tabla 9.2: Aciertos/fallos XGBoost Binario.

Ambos modelos se entrenaron rápidamente siendo este tiempo de entrenamiento de aproximadamente 40 segundos random forest y con una gran diferencia, XGboost con 6 segundos, ofreciendo también resultados prácticamente perfectos en este escenario simplificado.

9.2.2 Dataset multiclase

En este segundo enfoque se conserva la estructura multiclase del dataset balanceado, permitiendo evaluar la capacidad de los modelos para distinguir entre diferentes tipos de ataque. Se han vuelto a emplear *Random Forest* y *XGBoost*.

Random Forest Multiclase

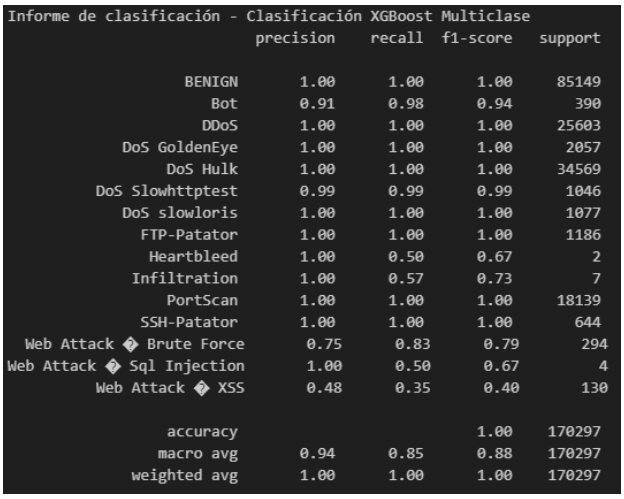


Informe de clasificación - Clasificación RF Multiclase

	precision	recall	f1-score	support
BENIGN	1.00	1.00	1.00	85149
Bot	0.91	0.95	0.93	390
DDoS	1.00	1.00	1.00	25603
DoS GoldenEye	1.00	1.00	1.00	2057
DoS Hulk	1.00	1.00	1.00	34569
DoS Slowhttptest	0.99	0.99	0.99	1046
DoS slowloris	1.00	1.00	1.00	1077
FTP-Patator	1.00	1.00	1.00	1186
Heartbleed	1.00	0.50	0.67	2
Infiltration	1.00	0.71	0.83	7
PortScan	1.00	1.00	1.00	18139
SSH-Patator	1.00	1.00	1.00	644
Web Attack Brute Force	0.73	0.77	0.75	294
Web Attack Sql Injection	1.00	0.25	0.40	4
Web Attack XSS	0.38	0.29	0.33	130
accuracy			1.00	170297
macro avg	0.93	0.83	0.86	170297
weighted avg	1.00	1.00	1.00	170297

Figura 9.3: Informe Random Forest Multiclase.

XGBoost Multiclase



Informe de clasificación - Clasificación XGBoost Multiclase

	precision	recall	f1-score	support
BENIGN	1.00	1.00	1.00	85149
Bot	0.91	0.98	0.94	390
DDoS	1.00	1.00	1.00	25603
DoS GoldenEye	1.00	1.00	1.00	2057
DoS Hulk	1.00	1.00	1.00	34569
DoS Slowhttptest	0.99	0.99	0.99	1046
DoS slowloris	1.00	1.00	1.00	1077
FTP-Patator	1.00	1.00	1.00	1186
Heartbleed	1.00	0.50	0.67	2
Infiltration	1.00	0.57	0.73	7
PortScan	1.00	1.00	1.00	18139
SSH-Patator	1.00	1.00	1.00	644
Web Attack Brute Force	0.75	0.83	0.79	294
Web Attack Sql Injection	1.00	0.50	0.67	4
Web Attack XSS	0.48	0.35	0.40	130
accuracy			1.00	170297
macro avg	0.94	0.85	0.88	170297
weighted avg	1.00	1.00	1.00	170297

Figura 9.4: Informe XGBoost Multiclase.

Clase	Aciertos	Fallos
BENIGN	85026	121
Bot	373	19
DDoS	25599	4
DoS GoldenEye	2047	10
DoS Hulk	34528	41
DoS Slowhttptest	1040	6
DoS slowloris	1073	4
FTP-Patator	1186	0
Heartbleed	1	1
Infiltration	5	2
PortScan	18102	37
SSH-Patator	642	2
Web Attack – Brute Force	227	67
Web Attack – Sql Injection	1	3
Web Attack – XSS	38	92

Tabla 9.3: Aciertos/fallos Random Forest Multiclase.

Clase	Aciertos	Fallos
BENIGN	85043	106
Bot	381	9
DDoS	25602	1
DoS GoldenEye	2051	6
DoS Hulk	34559	4
DoS Slowhttptest	1040	6
DoS slowloris	1075	2
FTP-Patator	1186	0
Heartbleed	1	1
Infiltration	4	3
PortScan	18133	6
SSH-Patator	644	0
Web Attack Brute Force	245	49
Web Attack Sql Injection	2	2
Web Attack XSS	45	85

Tabla 9.4: Aciertos/fallos XGBoost Multiclase.

El tiempo de entrenamiento sigue siendo bajo para ambos modelos, aproximadamente 46 segundos para Random Forest y un notorio aumento a 53 segundos XGBoost, incluso con la mayor cantidad de clases.

9.2.3 Dataset con ataques web agrupados

Dado que las clases de ataques web presentan pocos ejemplos individuales, se ha optado por agruparlas en una sola clase denominada `Web Attack`. Esta agrupación permite mejorar la estabilidad del modelo en dichas clases. Los modelos evaluados son nuevamente *Random Forest* y *XGBoost*.

Random Forest Agrupado

Informe de clasificación - Clasificación RF Web Agrupados

	precision	recall	f1-score	support
BENIGN	1.00	1.00	1.00	85148
Bot	0.91	0.96	0.93	390
DDoS	1.00	1.00	1.00	25603
DoS GoldenEye	1.00	0.99	1.00	2057
DoS Hulk	1.00	1.00	1.00	34569
DoS Slowhttptest	0.99	0.99	0.99	1046
DoS slowloris	1.00	1.00	1.00	1077
FTP-Patator	1.00	1.00	1.00	1186
Heartbleed	1.00	0.50	0.67	2
Infiltration	1.00	0.71	0.83	7
PortScan	1.00	1.00	1.00	18139
SSH-Patator	1.00	1.00	1.00	644
Web Attack	0.99	0.98	0.98	429
accuracy			1.00	170297
macro avg	0.99	0.93	0.95	170297
weighted avg	1.00	1.00	1.00	170297

Figura 9.5: Informe Random Forest Agrupado.

XGBoost Agrupado

Informe de clasificación - Clasificación XGBoost Web Agrupados

	precision	recall	f1-score	support
BENIGN	1.00	1.00	1.00	85148
Bot	0.91	0.97	0.94	390
DDoS	1.00	1.00	1.00	25603
DoS GoldenEye	1.00	1.00	1.00	2057
DoS Hulk	1.00	1.00	1.00	34569
DoS Slowhttptest	0.99	0.99	0.99	1046
DoS slowloris	1.00	1.00	1.00	1077
FTP-Patator	1.00	1.00	1.00	1186
Heartbleed	1.00	0.50	0.67	2
Infiltration	1.00	0.57	0.73	7
PortScan	1.00	1.00	1.00	18139
SSH-Patator	1.00	1.00	1.00	644
Web Attack	1.00	1.00	1.00	429
accuracy			1.00	170297
macro avg	0.99	0.93	0.95	170297
weighted avg	1.00	1.00	1.00	170297

Figura 9.6: Informe XGBoost Agrupado.

Clase	Aciertos	Fallos
BENIGN	85031	121
Bot	373	17
DDoS	25598	5
DoS GoldenEye	2046	10
DoS Hulk	34530	11
DoS Slowhttptest	1040	6
DoS slowloris	1074	3
FTP-Patator	1186	0
Heartbleed	1	1
Infiltration	5	2
PortScan	18100	39
SSH-Patator	643	1
Web Attack	419	10

Tabla 9.5: Aciertos/fallos Random Forest Agrupado.

Clase	Aciertos	Fallos
BENIGN	85041	107
Bot	380	10
DDoS	25602	1
DoS GoldenEye	2050	7
DoS Hulk	34562	7
DoS Slowhttptest	1040	6
DoS slowloris	1075	2
FTP-Patator	1186	0
Heartbleed	1	1
Infiltration	4	3
PortScan	18133	6
SSH-Patator	644	0
Web Attack	428	1

Tabla 9.6: Aciertos/fallos XGBoost Agrupado.

Ambos modelos presentan entrenamientos rápidos y estables en esta configuración. El tiempo de entrenamiento para Random Forest es el mismo, aproximadamente 46 segundos, pero se consigue reducir el tiempo de XGBoost llegando a unos 48 segundos.

9.2.4 Dataset reducido

En este punto se introducen modelos con mayor complejidad computacional como *MLP* y *SVM*, por lo que se ha reducido el número de muestras por clase para facilitar su entrenamiento. Se utiliza una primera versión del dataset reducido.

MLP Reducido

Informe de clasificación - Clasificación MLP Reducido

	precision	recall	f1-score	support
BENIGN	0.99	0.96	0.97	2000
Bot	0.94	0.99	0.97	390
DDoS	1.00	1.00	1.00	1000
DoS GoldenEye	0.99	1.00	1.00	2057
DoS Hulk	0.98	1.00	0.99	1000
DoS Slowhttptest	0.99	0.99	0.99	1046
DoS slowloris	0.99	0.98	0.99	1077
FTP-Patator	1.00	0.99	1.00	1186
Heartbleed	1.00	1.00	1.00	2
Infiltration	0.71	0.71	0.71	7
PortScan	0.99	1.00	0.99	1000
SSH-Patator	0.96	0.99	0.98	644
Web Attack	0.97	0.96	0.96	429
accuracy			0.99	11838
macro avg	0.96	0.97	0.96	11838
weighted avg	0.99	0.99	0.99	11838

Figura 9.7: Informe MLP reducido.

SVM Reducido

Informe de clasificación - Clasificación SVM Reducido

	precision	recall	f1-score	support
BENIGN	0.97	0.87	0.92	2000
Bot	0.82	0.99	0.90	390
DDoS	0.95	1.00	0.98	1000
DoS GoldenEye	0.98	0.99	0.99	2057
DoS Hulk	0.95	0.98	0.97	1000
DoS Slowhttptest	0.98	0.98	0.98	1046
DoS slowloris	0.98	0.98	0.98	1077
FTP-Patator	1.00	0.99	0.99	1186
Heartbleed	1.00	0.50	0.67	2
Infiltration	0.00	0.00	0.00	7
PortScan	0.96	0.98	0.97	1000
SSH-Patator	0.92	0.98	0.95	644
Web Attack	0.94	0.91	0.93	429
accuracy			0.96	11838
macro avg	0.88	0.86	0.86	11838
weighted avg	0.96	0.96	0.96	11838

Figura 9.8: Informe SVM Reducido.

Clase	Aciertos	Fallos
BENIGN	1915	85
Bot	388	2
DDoS	998	2
DoS GoldenEye	2054	3
DoS Hulk	988	2
DoS Slowhttptest	1032	14
DoS slowloris	1059	19
FTP-Patator	1179	7
Heartbleed	2	0
Infiltration	5	2
PortScan	999	1
SSH-Patator	640	4
Web Attack	410	19

Tabla 9.7: Aciertos/fallos MLP Reducido.

Clase	Aciertos	Fallos
BENIGN	1738	262
Bot	385	5
DDoS	997	3
DoS GoldenEye	2035	22
DoS Hulk	984	16
DoS Slowhttptest	1023	23
DoS slowloris	1052	25
FTP-Patator	1170	16
Heartbleed	1	1
Infiltration	0	7
PortScan	983	17
SSH-Patator	634	10
Web Attack	391	38

Tabla 9.8: Aciertos/fallos SVM Reducido.

Ambos modelos presentan tiempos de entrenamiento relativamente altos para la cantidad de muestras que se manejan, aproximadamente 25 segundos MLP y 15 segundos SVM, que además requieren de un escalado previo de los datos.

9.2.5 Dataset reducido 2

Se realiza una segunda versión del dataset reducido, aumentando el número de muestras por clase para observar cómo escalan los modelos *MLP* y *SVM* ante más datos.

MLP Reducido 2

Informe de clasificación - Clasificación MLP Reducido 2

	precision	recall	f1-score	support
BENIGN	0.99	0.98	0.98	4000
Bot	0.89	0.99	0.94	390
DDoS	1.00	1.00	1.00	1600
DoS GoldenEye	0.99	1.00	1.00	2057
DoS Hulk	0.99	0.99	0.99	1600
DoS Slowhttptest	0.99	0.99	0.99	1046
DoS slowloris	0.99	0.99	0.99	1077
FTP-Patator	0.99	0.99	0.99	1186
Heartbleed	1.00	1.00	1.00	2
Infiltration	1.00	0.57	0.73	7
PortScan	1.00	1.00	1.00	1600
SSH-Patator	0.96	0.99	0.97	644
Web Attack	0.96	0.93	0.95	429
accuracy			0.99	15638
macro avg	0.98	0.95	0.96	15638
weighted avg	0.99	0.99	0.99	15638

Figura 9.9: Informe MLP reducido 2.

SVM Reducido 2

Informe de clasificación - Clasificación SVM Reducido 2

	precision	recall	f1-score	support
BENIGN	0.93	0.93	0.93	4000
Bot	0.91	0.67	0.77	390
DDoS	0.88	1.00	0.94	1600
DoS GoldenEye	0.99	0.98	0.99	2057
DoS Hulk	0.98	0.94	0.96	1600
DoS Slowhttptest	0.98	0.98	0.98	1046
DoS slowloris	0.97	0.97	0.97	1077
FTP-Patator	0.98	0.99	0.99	1186
Heartbleed	1.00	1.00	1.00	2
Infiltration	0.00	0.00	0.00	7
PortScan	0.97	0.99	0.98	1600
SSH-Patator	0.99	0.92	0.96	644
Web Attack	0.91	0.90	0.90	429
accuracy			0.95	15638
macro avg	0.88	0.87	0.87	15638
weighted avg	0.95	0.95	0.95	15638

Figura 9.10: Informe SVM Reducido 2.

Clase	Aciertos	Fallos
BENIGN	3906	94
Bot	385	5
DDoS	1594	6
DoS GoldenEye	2056	1
DoS Hulk	1581	19
DoS Slowhttptest	1033	13
DoS slowloris	1063	14
FTP-Patator	1179	7
Heartbleed	2	0
Infiltration	4	3
PortScan	1598	2
SSH-Patator	638	6
Web Attack	400	29

Tabla 9.9: Aciertos/fallos MLP Reducido 2.

Clase	Aciertos	Fallos
BENIGN	3727	273
Bot	260	130
DDoS	1598	2
DoS GoldenEye	2022	35
DoS Hulk	1501	99
DoS Slowhttptest	1023	23
DoS slowloris	1046	31
FTP-Patator	1171	15
Heartbleed	2	0
Infiltration	0	7
PortScan	1578	22
SSH-Patator	595	49
Web Attack	385	44

Tabla 9.10: Aciertos/fallos SVM Reducido 2.

El tiempo de entrenamiento se incrementa notablemente, especialmente para *SVM*, que presenta mayor coste computacional que el resto de modelos probados. Siendo los tiempos de 31 segundos MLP y casi 1 minuto SVM

9.2.6 Dataset reducido 3

En esta tercera versión del dataset reducido se aumenta aún más el número de muestras. Se vuelven a evaluar *MLP* y *SVM* para observar su rendimiento y escalabilidad.

MLP Reducido 3

Informe de clasificación - Clasificación MLP Reducido 3

	precision	recall	f1-score	support
BENIGN	0.99	0.96	0.98	6000
Bot	0.88	0.95	0.91	390
DDoS	1.00	1.00	1.00	2400
DoS GoldenEye	1.00	1.00	1.00	2057
DoS Hulk	0.99	0.99	0.99	2400
DoS Slowhttptest	0.99	0.99	0.99	1046
DoS slowloris	0.99	0.99	0.99	1077
FTP-Patator	1.00	1.00	1.00	1186
Heartbleed	1.00	0.50	0.67	2
Infiltration	1.00	0.71	0.83	7
PortScan	0.96	1.00	0.98	2400
SSH-Patator	0.97	0.99	0.98	644
Web Attack	0.93	0.95	0.94	429
accuracy			0.98	20038
macro avg	0.98	0.93	0.94	20038
weighted avg	0.98	0.98	0.98	20038

Figura 9.11: Informe MLP reducido 3.

SVM Reducido 3

Informe de clasificación - Clasificación SVM Reducido 3

	precision	recall	f1-score	support
BENIGN	0.93	0.95	0.94	6000
Bot	0.93	0.39	0.55	390
DDoS	0.97	1.00	0.99	2400
DoS GoldenEye	0.99	0.98	0.98	2057
DoS Hulk	0.98	0.95	0.97	2400
DoS Slowhttptest	0.99	0.98	0.99	1046
DoS slowloris	0.97	0.97	0.97	1077
FTP-Patator	0.99	0.99	0.99	1186
Heartbleed	1.00	0.50	0.67	2
Infiltration	0.00	0.00	0.00	7
PortScan	0.92	0.99	0.95	2400
SSH-Patator	0.99	0.93	0.96	644
Web Attack	0.90	0.90	0.90	429
accuracy			0.95	20038
macro avg	0.89	0.81	0.83	20038
weighted avg	0.95	0.95	0.95	20038

Figura 9.12: Informe SVM Reducido 3.

Clase	Aciertos	Fallos
BENIGN	5785	215
Bot	371	19
DDoS	2400	0
DoS GoldenEye	2050	7
DoS Hulk	2388	12
DoS Slowhttptest	1034	12
DoS slowloris	1068	9
FTP-Patator	1181	5
Heartbleed	1	1
Infiltration	5	2
PortScan	2394	6
SSH-Patator	637	7
Web Attack	406	23

Tabla 9.11: Aciertos/fallos MLP Reducido 3.

Clase	Aciertos	Fallos
BENIGN	5682	318
Bot	151	239
DDoS	2399	1
DoS GoldenEye	2006	51
DoS Hulk	2278	122
DoS Slowhttptest	1029	17
DoS slowloris	1048	29
FTP-Patator	1174	12
Heartbleed	1	1
Infiltration	0	7
PortScan	2367	33
SSH-Patator	596	48
Web Attack	384	45

Tabla 9.12: Aciertos/fallos SVM Reducido 3.

Los tiempos de entrenamiento vuelven a aumentar. *SVM* especialmente comienza a ser inviable para volúmenes mayores sin optimizaciones adicionales. Los tiempos son de 34 segundos para MLP y más de 2 minutos para SVM

9.2.7 Dataset reducido 2 - XGBoost

Dado que *XGBoost* ha mostrado un rendimiento notable en configuraciones anteriores, se entrena este modelo sobre la segunda versión del dataset reducido, utilizada anteriormente con *MLP* y *SVM*.

XGBoost Reducido

Informe de clasificación - Clasificación XGBoost Reducido

	precision	recall	f1-score	support
BENIGN	1.00	1.00	1.00	4000
Bot	0.99	0.99	0.99	390
DDoS	1.00	1.00	1.00	1600
DoS GoldenEye	1.00	1.00	1.00	2057
DoS Hulk	1.00	1.00	1.00	1600
DoS Slowhttptest	1.00	0.99	0.99	1046
DoS slowloris	0.99	0.99	0.99	1077
FTP-Patator	1.00	1.00	1.00	1186
Heartbleed	1.00	1.00	1.00	2
Infiltration	1.00	0.86	0.92	7
PortScan	1.00	1.00	1.00	1600
SSH-Patator	1.00	1.00	1.00	644
Web Attack	1.00	1.00	1.00	429
accuracy			1.00	15638
macro avg	1.00	0.99	0.99	15638
weighted avg	1.00	1.00	1.00	15638

Figura 9.13: Informe XGBoost Reducido

Clase	Aciertos	Fallos
BENIGN	3991	9
Bot	387	3
DDoS	1599	1
DoS GoldenEye	2057	0
DoS Hulk	1599	1
DoS Slowhttptest	1038	8
DoS slowloris	1071	6
FTP-Patator	1186	0
Heartbleed	2	0
Infiltration	6	1
PortScan	1599	1
SSH-Patator	644	0
Web Attack	427	2

Tabla 9.13: Aciertos/fallos XGBoost Reducido.

XGBoost mantiene tiempos de entrenamiento muy bajos incluso con un mayor número de muestras, y su rendimiento sigue siendo excelente. Aproximadamente 5 segundos.

9.2.8 Dataset con características reducidas

Para evaluar el impacto de la selección de características, se han generado dos datasets derivados de la versión reducida 2, manteniendo únicamente las 20 y 30 características más relevantes según el modelo *XGBoost*. Ambos experimentos se han realizado únicamente con *XGBoost*, al ser el modelo con mejor comportamiento global. También se detalla en la Figura 9.14 las características 30 características más importantes seleccionadas junto a su importancia en el modelo.

	importancia
Bwd Packet Length Min	1397.526245
Bwd Header Length	426.188477
PSH Flag Count	310.440155
Active Mean	240.727722
Total Length of Bwd Packets	216.219391
Average Packet Size	148.923996
Bwd IAT Min	125.157654
Fwd IAT Std	115.312141
Total Length of Fwd Packets	112.217041
min_seg_size_forward	111.639282
act_data_pkt_fwd	97.365959
Packet Length Mean	82.641098
Fwd Packet Length Max	81.167145
Bwd Packet Length Std	74.872612
Bwd Packet Length Mean	74.643600
Destination Port	71.134972
Flow Duration	69.431984
Fwd Packet Length Min	67.536583
FIN Flag Count	60.595936
Total Backward Packets	54.585491
Bwd IAT Std	48.140659
Idle Mean	46.975903
Flow IAT Std	32.800896
Init_Win_bytes_backward	31.854746
Total Fwd Packets	30.767115
Flow IAT Mean	30.147799
Flow IAT Max	28.333296
Bwd IAT Mean	28.141714

Figura 9.14: Top 30 características más importantes

XGBoost Top 20 Características

Informe de clasificación - XGBoost - Top 20 características

	precision	recall	f1-score	support
BENIGN	0.99	0.99	0.99	4000
Bot	0.94	0.98	0.96	390
DDoS	1.00	1.00	1.00	1600
DoS GoldenEye	1.00	1.00	1.00	2057
DoS Hulk	0.99	1.00	1.00	1600
DoS Slowhttptest	0.99	0.99	0.99	1046
DoS slowloris	0.99	0.99	0.99	1077
FTP-Patator	1.00	1.00	1.00	1186
Heartbleed	1.00	1.00	1.00	2
Infiltration	1.00	0.86	0.92	7
PortScan	1.00	1.00	1.00	1600
SSH-Patator	1.00	1.00	1.00	644
Web Attack	0.96	0.99	0.97	429
accuracy			0.99	15638
macro avg	0.99	0.98	0.99	15638
weighted avg	0.99	0.99	0.99	15638

Figura 9.15: Informe XGBoost Top 20 Características.

XGBoost Top 30 Características

Informe de clasificación - XGBoost - Top 30 características

	precision	recall	f1-score	support
BENIGN	1.00	0.99	0.99	4000
Bot	0.97	1.00	0.98	390
DDoS	1.00	1.00	1.00	1600
DoS GoldenEye	1.00	1.00	1.00	2057
DoS Hulk	1.00	1.00	1.00	1600
DoS Slowhttptest	1.00	0.99	0.99	1046
DoS slowloris	0.99	0.99	0.99	1077
FTP-Patator	1.00	1.00	1.00	1186
Heartbleed	1.00	1.00	1.00	2
Infiltration	1.00	0.86	0.92	7
PortScan	1.00	1.00	1.00	1600
SSH-Patator	1.00	1.00	1.00	644
Web Attack	0.97	0.99	0.98	429
accuracy			1.00	15638
macro avg	0.99	0.99	0.99	15638
weighted avg	1.00	1.00	1.00	15638

Figura 9.16: Informe XGBoost Top 30 Características.

Clase	Aciertos	Fallos
BENIGN	3952	58
Bot	381	9
DDoS	1594	6
DoS GoldenEye	2053	4
DoS Hulk	1596	4
DoS Slowhttptest	1036	10
DoS slowloris	1066	11
FTP-Patator	1186	0
Heartbleed	2	0
Infiltration	6	1
PortScan	1599	1
SSH-Patator	643	1
Web Attack	423	6

Tabla 9.14: Aciertos/fallos XGBoost Top 20 Características.

Clase	Aciertos	Fallos
BENIGN	3971	29
Bot	389	1
DDoS	1595	5
DoS GoldenEye	2053	4
DoS Hulk	1597	3
DoS Slowhttptest	1038	8
DoS slowloris	1071	6
FTP-Patator	1186	0
Heartbleed	2	0
Infiltration	6	1
PortScan	1599	1
SSH-Patator	644	0
Web Attack	425	4

Tabla 9.15: Aciertos/fallos XGBoost Top 30 Características.

El tiempo de entrenamiento se ve ligeramente reducido respecto al anterior, y las métricas se mantienen prácticamente inalteradas, lo que indica que es posible reducir el número de características sin sacrificar el rendimiento del modelo. Tanto para 20 como para 30 características el tiempo de entramiento es de aproximadamente 3 segundos.

9.3 Análisis y discusión

Una vez obtenidos los resultados experimentales de los distintos modelos de clasificación entrenados con múltiples configuraciones de dataset, es fundamental realizar un análisis más profundo para interpretar su comportamiento y compararlos entre sí. Este apartado tiene como objetivo evaluar qué técnicas ofrecen un mejor rendimiento, cómo influye la estructura del dataset en los resultados y qué conclusiones se pueden extraer respecto al coste computacional, la escalabilidad y la capacidad de detección de los modelos.

A continuación, se presentan los aspectos más relevantes observados durante las pruebas y sus implicaciones prácticas.

9.3.1 Comparativa entre enfoques

Durante el proceso experimental se han utilizado diferentes configuraciones del dataset original con el fin de evaluar el comportamiento de los modelos en contextos variados: un enfoque binario, uno multiclase, un agrupamiento de los ataques web, y finalmente, varias versiones reducidas del dataset para permitir la evaluación de modelos más costosos computacionalmente.

El **enfoque binario**, que diferencia únicamente entre tráfico `BENIGN` y `MALIGN`, ha demostrado ser el más sencillo de abordar. Tanto *Random Forest* como *XGBoost* han alcanzado métricas perfectas o casi perfectas en este escenario, mostrando que el problema se vuelve trivial cuando se reduce a una clasificación binaria. Este resultado, si bien útil para validar el correcto funcionamiento general del sistema, no resulta representativo de un entorno realista donde es importante distinguir entre diferentes tipos de amenazas.

En cambio, el **enfoque multiclase** ha supuesto un reto significativamente mayor, sobre todo en lo que respecta a la detección de clases con un número muy reducido de muestras, como `Heartbleed` o `Infiltration`. Estas clases presentan una alta dificultad para los modelos, que en muchos casos no logran identificarlas correctamente, incluso cuando las métricas globales se mantienen elevadas. Esto evidencia la importancia de analizar los resultados por clase individualmente y no solo en función de los promedios.

Para mitigar en parte este problema, se propuso una estrategia intermedia que **agrupa todas las variantes de ataques web** (*Brute Force*, *XSS*, *SQL Injection*) bajo una única clase `Web Attack`. Esta decisión se tomó tras observar un rendimiento muy bajo en estas clases cuando se trataban de forma independiente. Al agruparlas, el modelo mejora notablemente su rendimiento en esta categoría, aumentando su capacidad de detección y simplificando el problema sin perder la capacidad de reconocer ataques relevantes.

Finalmente, se han creado **tres versiones reducidas del dataset**, que permiten evaluar modelos más costosos como *MLP* o *SVM* sin que el tiempo de entrenamiento resulte prohibitivo. Estas versiones también resultan útiles para valorar el impacto del tamaño del dataset en las métricas y el comportamiento general de los modelos. Como se verá más adelante, esta reducción ha permitido obtener resultados comparables a los del dataset completo en algunos casos, lo que plantea alternativas más ligeras y eficientes para ciertos contextos.

9.3.2 Comparativa entre modelos

A lo largo de los diferentes enfoques evaluados, se han probado hasta cuatro algoritmos de clasificación: *Random Forest*, *XGBoost*, *SVM* y *MLP*. Cada uno presenta ventajas y limitaciones que han sido evidentes en las distintas pruebas.

Los modelos basados en árboles, como *Random Forest* y especialmente *XGBoost*, han sido los que han mostrado mejores resultados globales en términos de precisión, *recall* y *f1-score*, tanto en el enfoque binario como en el multiclase y agrupado. Además, se caracterizan por una gran robustez ante datos desbalanceados, algo especialmente relevante en este trabajo. *XGBoost* ha demostrado ser el modelo más consistente incluso

en configuraciones más exigentes, manteniendo métricas elevadas en casi todas las clases, incluidas algunas minoritarias.

Por su parte, *Random Forest* también ofrece buenos resultados, aunque en general ligeramente inferiores a los de *XGBoost*. Sin embargo, destaca por su rapidez de entrenamiento y su menor complejidad computacional, lo que lo convierte en una alternativa válida en entornos donde se prioriza la eficiencia por encima del rendimiento máximo.

Los modelos de mayor coste computacional, como *MLP* y *SVM*, han requerido reducir el tamaño del dataset para ser evaluados de forma viable. A pesar de ello, *MLP* ha mostrado un rendimiento muy competitivo, con resultados cercanos a los de los modelos de árboles, especialmente en la versión reducida 2 del dataset. Sin embargo, su entrenamiento es notablemente más lento, y la elección de sus hiperparámetros influye considerablemente en los resultados.

Por otro lado, *SVM* ha sido el modelo con mayores limitaciones en cuanto a escalabilidad y rendimiento, especialmente cuando se incrementa el tamaño del conjunto de datos. Aunque ha conseguido buenos resultados en algunas clases, su comportamiento ha sido irregular, especialmente en aquellas con pocas muestras, como `Infiltration` o `Bot`, donde ha presentado valores de *recall* muy bajos o incluso nulos. Además, ha sido el modelo con tiempos de entrenamiento más elevados, lo que limita su viabilidad en escenarios prácticos con grandes volúmenes de datos.

En resumen, *XGBoost* ha sido el modelo más equilibrado entre rendimiento, capacidad de generalización y eficiencia, seguido de cerca por *Random Forest*. *MLP* ha demostrado ser una opción válida en contextos controlados con datasets reducidos, mientras que *SVM* ha resultado poco escalable para este tipo de problema.

9.3.3 Coste computacional

El coste computacional ha sido un factor clave a la hora de comparar y seleccionar los modelos, especialmente teniendo en cuenta la cantidad de datos y características del dataset original. Para estimar este coste se ha medido el tiempo de entrenamiento de cada modelo en sus distintas configuraciones.

Los modelos basados en árboles han demostrado una gran eficiencia en este aspecto. *Random Forest* ha sido rápido de entrenar en la mayoría de los escenarios, especialmente con datasets reducidos, aunque en el enfoque binario ha sido superado por *XGBoost*, que consiguió tiempos de entrenamiento aún menores pese a su mayor complejidad interna. Sin embargo, en el enfoque multiclase, *XGBoost* ha requerido algo más de tiempo debido al mayor número de clases y a la gestión interna del *boosting*.

Aun así, ambos modelos han mantenido tiempos de entrenamiento razonables y perfectamente asumibles, incluso con el dataset completo, lo que los hace adecuados para entornos con recursos limitados o para iteraciones frecuentes durante el desarrollo.

Por el contrario, *MLP* y *SVM* han mostrado limitaciones importantes en cuanto a coste computacional. Ambos modelos requieren escalar los datos antes del entrenamiento, lo cual añade un paso adicional al *pipeline*. Además, sus tiempos de entrenamiento aumentan considerablemente con el tamaño del dataset, siendo necesario reducir la cantidad de muestras para poder ejecutarlos en un entorno local sin agotar recursos.

En el caso de *MLP*, aunque el entrenamiento puede llevar más tiempo que los modelos de árboles, ha sido manejable en datasets de tamaño medio, y sus resultados han sido consistentes. En cambio, *SVM* ha sido el modelo más costoso computacionalmente, especialmente en los datasets más grandes, donde el tiempo de entrenamiento ha llegado a ser excesivo. En algunos casos, ha sido necesario limitar fuertemente el número de muestras para evitar tiempos de espera de más de 10-15 minutos.

Este análisis pone de manifiesto que, más allá del rendimiento en las métricas, el coste computacional es un factor decisivo. *XGBoost* destaca no solo por su precisión, sino también por su equilibrio entre rendimiento y eficiencia computacional.

9.3.4 Selección de características

Uno de los últimos experimentos realizados consistió en reducir el número de características del dataset para comprobar cómo afectaba al rendimiento del modelo. Para ello se utilizó el algoritmo *XGBoost*, aprovechando que ofrece mecanismos internos para evaluar la importancia relativa de cada *feature* según distintos criterios (en este caso, el criterio de ganancia).

Se generaron dos versiones del dataset con únicamente las 20 y 30 características más importantes, y se entrenó de nuevo el modelo *XGBoost* sobre cada una de ellas. Los resultados muestran que, si bien ambas versiones mantienen un rendimiento muy similar al obtenido con el dataset completo, existe una diferencia apreciable entre ellas: el modelo entrenado con las 30 características alcanza mejores métricas que el de 20, especialmente en clases minoritarias, donde el *recall* y el *F1-score* se ven más penalizados cuando se usa un conjunto más reducido de atributos.

Además, al comparar el coste computacional de ambos entrenamientos, se observó que los tiempos fueron prácticamente idénticos, por lo que no existe una ganancia relevante al reducir de 30 a 20 características en términos de eficiencia. Dado que la versión de 30 características conserva mayor precisión y cobertura sin perjudicar al rendimiento, se considera más adecuada y equilibrada para este problema.

Este experimento confirma que gran parte de la información necesaria para la clasificación se encuentra concentrada en un subconjunto relativamente pequeño de características, y que muchas de las variables originales no aportan valor añadido al modelo o incluso pueden introducir ruido.

Además de mejorar la interpretabilidad, esta reducción de dimensionalidad permite entrenamientos más rápidos y modelos más ligeros, lo que puede ser especialmente útil en entornos con recursos computacionales limitados o donde se requiera realizar inferencias en tiempo real.

El uso de *XGBoost* como herramienta para la selección de características se justifica por su fiabilidad y la capacidad inherente del modelo para priorizar aquellas variables que contribuyen más a la clasificación. Al tratarse de un algoritmo basado en árboles de decisión, es capaz de capturar interacciones no lineales entre variables y medir su impacto directo en la ganancia de información durante el proceso de entrenamiento, lo que lo convierte en una opción robusta y eficaz para esta tarea.

9.3.5 Elección final del modelo

Tras realizar todas las pruebas y comparar el rendimiento de los distintos modelos, se ha determinado que *XGBoost* es el algoritmo más adecuado para este problema de detección de intrusos.

- **Métricas de evaluación:** *XGBoost* ha conseguido, en prácticamente todos los escenarios, las mejores métricas globales (*accuracy*, *precision*, *recall* y *F1-score*), tanto en clasificación binaria como multiclase. Incluso en clases minoritarias, donde otros modelos como *SVM* o *MLP* presentan un rendimiento más irregular, *XGBoost* ha mostrado una mejor capacidad de detección y un menor número de errores.

- **Robustez ante clases desbalanceadas:** Mientras que otros algoritmos han mostrado dificultades para identificar correctamente clases con pocas muestras (como `Infiltration`, `Heartbleed` o `Web Attack`), *XGBoost* ha mantenido un rendimiento más consistente. Esto lo convierte en una opción más fiable en entornos reales, donde el desbalanceo es habitual.

- **Escalabilidad y coste computacional:** Aunque *XGBoost* tiene un coste computacional mayor que *Random Forest* en el caso multiclase, su rendimiento es superior y más estable a medida que crece el tamaño

del dataset. Además, en clasificación binaria, el tiempo de entrenamiento ha sido considerablemente menor que en *Random Forest*, lo que demuestra una buena capacidad de escalado en conjuntos de datos con estructuras más simples.

- **Versatilidad:** A lo largo de los experimentos se ha podido observar que *XGBoost* se adapta bien a todas las variantes de preprocesado utilizadas, manteniendo siempre un rendimiento elevado. Tanto en los datasets completos como en los reducidos o con agrupación de clases, el modelo ha sido capaz de ajustarse eficazmente sin necesidad de reconfigurar el *pipeline*.
- **Reducción de dimensionalidad:** Finalmente, se ha observado que *XGBoost* sigue manteniendo un rendimiento excelente incluso cuando se limita el número de características del dataset. En concreto, los resultados con las 30 características más importantes han sido prácticamente idénticos a los obtenidos con el conjunto completo, lo que sugiere que esta configuración es la más eficiente en términos de coste-beneficio.

Por todo lo anterior, el modelo final seleccionado es *XGBoost* entrenado con las 30 características más importantes del dataset, ya que proporciona el mejor equilibrio entre rendimiento, eficiencia y escalabilidad.

Capítulo 10

Conclusiones

Este capítulo recoge las conclusiones principales derivadas del desarrollo de esta investigación, que ha tenido como objetivo principal evaluar diferentes modelos de inteligencia artificial aplicados a la detección de ataques de intrusión en redes. A través de una serie de pruebas sistemáticas, se ha podido comprobar el rendimiento de varios algoritmos en distintos escenarios de clasificación, permitiendo identificar el modelo más eficaz en este contexto. Asimismo, se exponen también, posibles líneas de trabajo a futuro.

10.1 Modelo final seleccionado

El objetivo principal de este trabajo era encontrar el modelo de *machine learning* más adecuado para la detección de intrusiones en entornos de red, cumpliendo así con la hipótesis inicial planteada. Esta hipótesis ha quedado validada, ya que tras la evaluación comparativa de varios modelos, se ha determinado que *XGBoost* es el algoritmo que ofrece los mejores resultados en términos de precisión, robustez ante clases minoritarias, escalabilidad y tiempo de entrenamiento.

Además, se ha comprobado que el rendimiento óptimo se alcanza al utilizar las 30 características más importantes del dataset, lo que permite una mayor eficiencia sin comprometer la calidad de las predicciones.

10.2 Trabajo a futuro

Este trabajo puede ampliarse en varias direcciones:

- Explorar técnicas de aprendizaje profundo (*Deep Learning*), que podrían ofrecer mejoras en la detección de ataques más sofisticados.
- Ampliar la variedad de datasets utilizados para evaluar la capacidad de generalización del modelo final.
- Incorporar técnicas de detección en tiempo real, integrando el modelo dentro de un sistema de monitorización de red con procesamiento de paquetes en vivo.
- Aplicar métodos de selección de características más avanzados

Appendices

Apéndice A

Código desarrollado

A.1 Preprocesamiento

```
def cargar_csvs(ruta_csvs):

  archivos = glob.glob(os.path.join(ruta_csvs, "*.csv"))
  dataframes = []

  for f in archivos:
      try:
          df = pd.read_csv(f, low_memory=False)
          dataframes.append(df)
          print(f"  {os.path.basename(f)}: {len(df)} filas.")
      except Exception as e:
          print(f"  Error en {f}: {e}")

  df_total = pd.concat(dataframes, ignore_index=True)
  print(f"\nDataset combinado: {len(df_total)} filas totales, {len(df_total.columns)} columnas totales.")
  return df_total

def limpiar_dataset(df):

  df.replace([np.inf, -np.inf], np.nan, inplace=True)

  columnas_vacias = df.columns[df.isna().all()].tolist()
  df.drop(columns=columnas_vacias, inplace=True)

  columnas_constantes = [col for col in df.columns if df[col].nunique() <= 1]
  df.drop(columns=columnas_constantes, inplace=True)

  df.dropna(inplace=True)
  df.drop_duplicates(inplace=True)

  print(f"  Dataset limpio: {len(df)} filas, {df.shape[1]} columnas.")
  return df

def balancear_dataset(df):

  benignos = df[df[' Label'] == 'BENIGN']
  ataques = df[df[' Label'] != 'BENIGN']

  # Realizar undersample de la clase "BENIGN" para balancear
  benignos = benignos.sample(n=len(ataques), random_state=42)
  df_balanceado = pd.concat([benignos, ataques])

  # Barajar los datos y resetear índices
  df_balanceado = df_balanceado.sample(frac=1, random_state=42).reset_index(drop=True)
```

```
  return df_balanceado

def convertir_binario(df):
  df = df.copy()

  df[' Label'] = df[' Label'].apply(lambda x: 'BENIGN' if x == 'BENIGN' else 'MALIGN')

  return df

def agrupar_web_attacks(df):

  df = df.copy()

  df[' Label'] = df[' Label'].replace({
  'Web Attack   Brute Force': 'Web Attack',
  'Web Attack   XSS': 'Web Attack',
  'Web Attack   Sql Injection': 'Web Attack'
  })

  return df

def reducir_dataset(df, etiqueta_col=' Label', limites={}):

  clases = df[etiqueta_col].unique()
  partes = []

  for clase in clases:
      datos_clase = df[df[etiqueta_col] == clase]

      if clase in limites:
          n = limites[clase]
          datos_clase = datos_clase.sample(n=min(len(datos_clase), n), random_state=42)

      partes.append(datos_clase)

  df_reducido = pd.concat(partes).sample(frac=1, random_state=42).reset_index(drop=True)
  return df_reducido
```

A.2 Entrenamiento de modelos

```
def preparar_datos(df):
  df = df.copy()

  # Eliminar la columna de etiquetas
  X = df.drop(columns=[' Label'])

  # Convertir etiquetas
  y = df[' Label']

  # Codificar etiquetas
  le = LabelEncoder()
  y_encoded = le.fit_transform(y)

  # Dividir en train/test
  X_train, X_test, y_train, y_test = train_test_split(
      X, y_encoded, test_size=0.2, random_state=42, stratify=y_encoded
  )

  return X_train, X_test, y_train, y_test, le

def entrenar_random_forest(X_train, y_train):
  modelo = RandomForestClassifier(
      n_estimators=100,
      random_state=42,
      class_weight='balanced',
      n_jobs=-1
  )
  modelo.fit(X_train, y_train)
```

```
    return modelo

def entrenar_random_forest_sin_balancear(X_train, y_train):
    modelo = RandomForestClassifier(
        n_estimators=100,
        random_state=42,
        n_jobs=-1
    )
    modelo.fit(X_train, y_train)
    return modelo

def entrenar_xgboost(X_train, y_train):
    modelo = XGBClassifier(
        objective='multi:softmax' if len(set(y_train)) > 2 else 'binary:logistic',
        num_class=len(set(y_train)) if len(set(y_train)) > 2 else None,
        eval_metric='mlogloss' if len(set(y_train)) > 2 else 'logloss',
        use_label_encoder=False,
        random_state=42,
        n_jobs=-1
    )
    modelo.fit(X_train, y_train)
    return modelo

def entrenar_svm(X_train, y_train, C=1.0, gamma='scale', kernel='linear'):
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)

    modelo = SVC(kernel=kernel, C=C, gamma=gamma, random_state=42, verbose=False)
    modelo.fit(X_train_scaled, y_train)

    return modelo, scaler

def entrenar_mlp(X_train, y_train, hidden_layer_sizes=(100,), max_iter=300, alpha=0.0001):
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)

    modelo = MLPClassifier(
        hidden_layer_sizes=hidden_layer_sizes,
        activation='relu',
        solver='adam',
        max_iter=max_iter,
        alpha=alpha,
        random_state=42,
        early_stopping=True,
        verbose=False
    )
    modelo.fit(X_train_scaled, y_train)

    return modelo, scaler
```

A.3 Evaluación de modelos

```
def evaluar_modelo(modelo, X_test, y_test, label_encoder,titulo="Evaluación",tamaño=(18, 12),cmap='Purples'):
    y_pred = modelo.predict(X_test)

    print(f"Informe de clasificación - {titulo}")
    print(classification_report(y_test, y_pred, target_names=label_encoder.classes_))

    # Matriz de confusión con tamaño ajustado
    cm = confusion_matrix(y_test, y_pred)
    fig, ax = plt.subplots(figsize=tamaño)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=label_encoder.classes_)
    disp.plot(ax=ax, cmap=cmap, xticks_rotation=45, colorbar=True)
    plt.title(f"Matriz de Confusión - {titulo}")
    plt.grid(False)
    plt.tight_layout()
    plt.show()
```

A.4 Extracción de características

```
def grafica_importancia_caracteristicas(modelo, max_features=20):

  xgb.plot_importance(
      modelo,
      max_num_features=max_features,
      importance_type='gain',
      height=0.5
  )
  plt.title(f"Top {max_features} características más importantes")
  plt.show()

def obtener_importancias(modelo, tipo="gain"):
  importancia = modelo.get_booster().get_score(importance_type=tipo)
  imp_df = pd.DataFrame.from_dict(importancia, orient='index', columns=['importancia'])
  imp_df = imp_df.sort_values(by='importancia', ascending=False)
  return imp_df
```

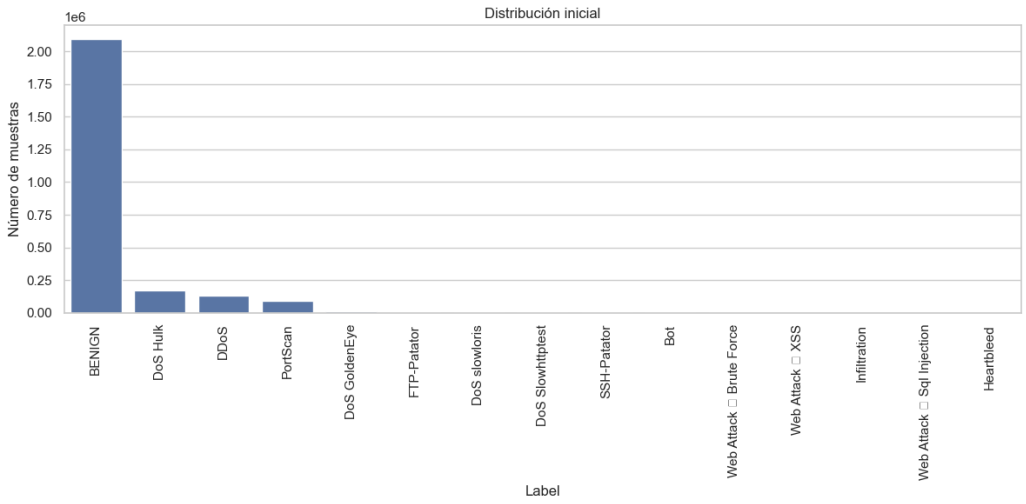
A.5 Mostar distribución

```
def mostrar_grafica_distribucion(df, titulo_grafica):

  conteo_etiquetas = df[' Label'].value_counts().sort_values(ascending=False)

  plt.figure(figsize=(12, 6))
  sns.barplot(x=conteo_etiquetas.index, y=conteo_etiquetas.values)
  plt.title(titulo_grafica)
  plt.xlabel("Label")
  plt.ylabel("Número de muestras")
  plt.xticks(rotation=90)
  plt.tight_layout()
  plt.show()

def mostrar_tabla_distribucion(df):
  conteo_etiquetas = df[' Label'].value_counts().sort_values(ascending=False)

  fig_tabla, ax_tabla = plt.subplots(figsize=(8, 6))
  tabla = pd.DataFrame({
      'Label': conteo_etiquetas.index,
      'Número de muestras': conteo_etiquetas.values
  })

  ax_tabla.axis('off')
  tabla_plot = ax_tabla.table(
      cellText=tabla.values,
      colLabels=tabla.columns,
      loc='center',
      cellLoc='center'
  )

  tabla_plot.auto_set_font_size(False)
  tabla_plot.set_fontsize(10)
  tabla_plot.scale(1.2, 1.5)

  for key, cell in tabla_plot.get_celld().items():
      if key[0] == 0:  # primera fila (encabezados)
          cell.set_text_props(weight='bold')

  plt.title("Tabla de distribución de etiquetas", pad=20)
  plt.tight_layout()
  plt.show()
```

Apéndice B

Distribuciones de los diferentes datasets

B.1 Distribución inicial



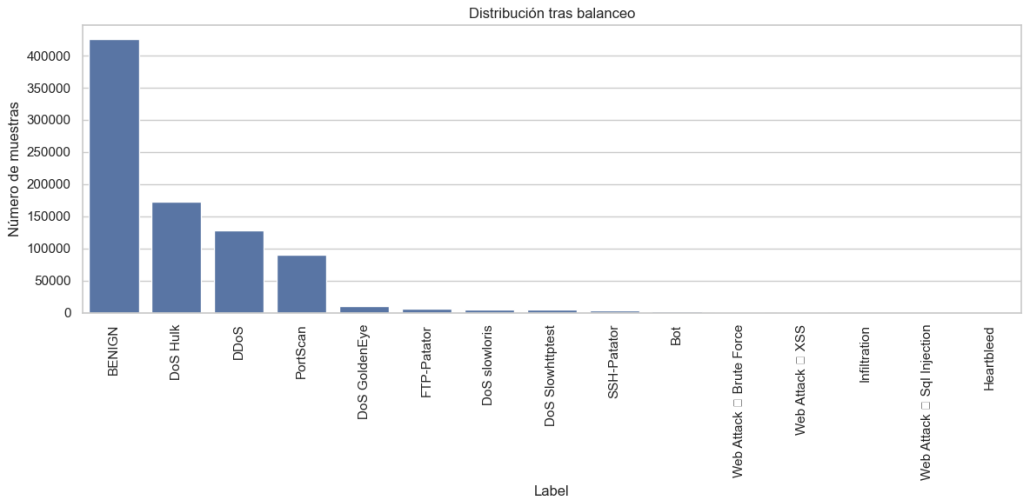
Gráfica de distribución inicial

Tabla de distribución de etiquetas

Label	Número de muestras
BENIGN	2095057
DoS Hulk	172846
DDoS	128014
PortScan	90694
DoS GoldenEye	10286
FTP-Patator	5931
DoS slowloris	5385
DoS Slowhttptest	5228
SSH-Patator	3219
Bot	1948
Web Attack □ Brute Force	1470
Web Attack □ XSS	652
Infiltration	36
Web Attack □ Sql Injection	21
Heartbleed	11

Tabla de distribución inicial

B.2 Distribución balanceada



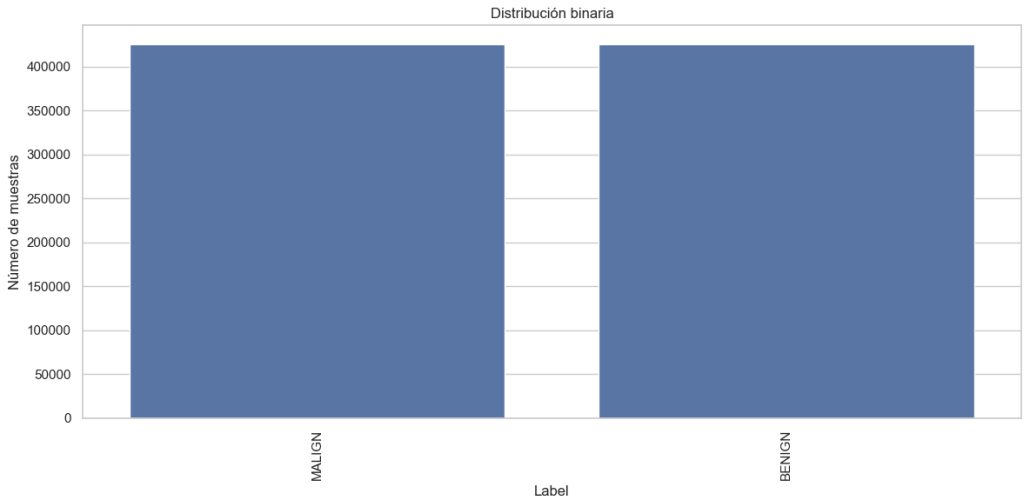
Gráfica de distribución balanceada

Tabla de distribución de etiquetas

Label	Número de muestras
BENIGN	425741
DoS Hulk	172846
DDoS	128014
PortScan	90694
DoS GoldenEye	10286
FTP-Patator	5931
DoS slowloris	5385
DoS Slowhttptest	5228
SSH-Patator	3219
Bot	1948
Web Attack □ Brute Force	1470
Web Attack □ XSS	652
Infiltration	36
Web Attack □ Sql Injection	21
Heartbleed	11

Tabla de distribución balanceada

B.3 Distribución binaria



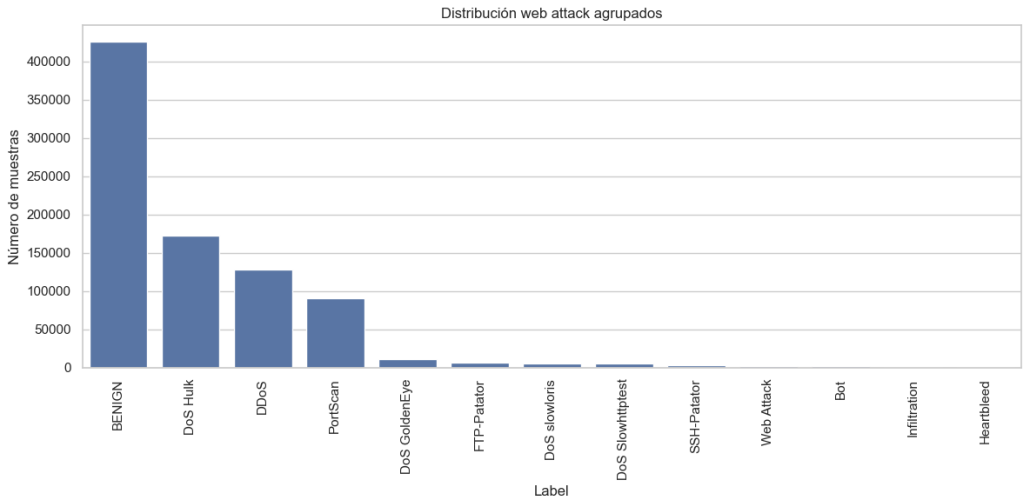
Gráfica de distribución binaria

Tabla de distribución de etiquetas

Label	Número de muestras
MALIGN	425741
BENIGN	425741

Tabla de distribución binaria

B.4 Distribución agrupada



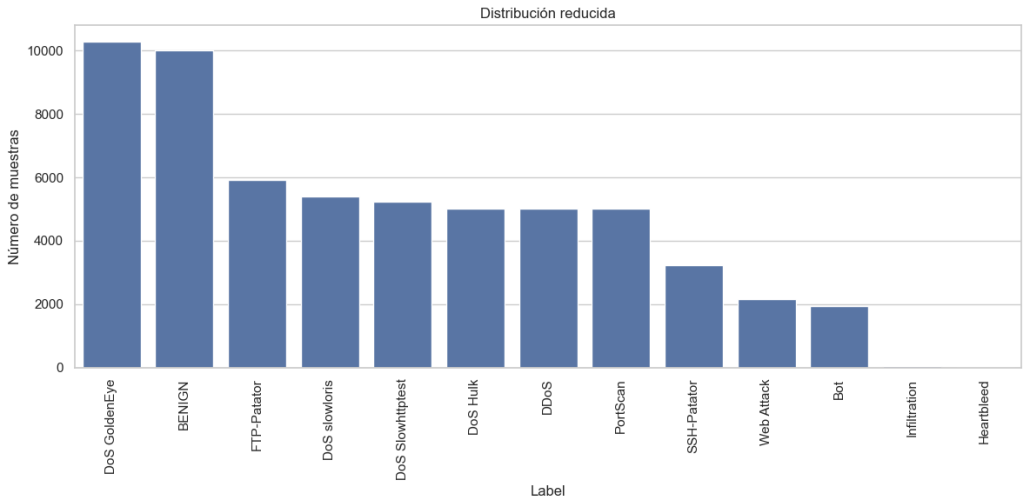
Gráfica de distribución agrupada

Tabla de distribución de etiquetas

Label	Número de muestras
BENIGN	425741
DoS Hulk	172846
DDoS	128014
PortScan	90694
DoS GoldenEye	10286
FTP-Patator	5931
DoS slowloris	5385
DoS Slowhttptest	5228
SSH-Patator	3219
Web Attack	2143
Bot	1948
Infiltration	36
Heartbleed	11

Tabla de distribución agrupada

B.5 Distribución reducida



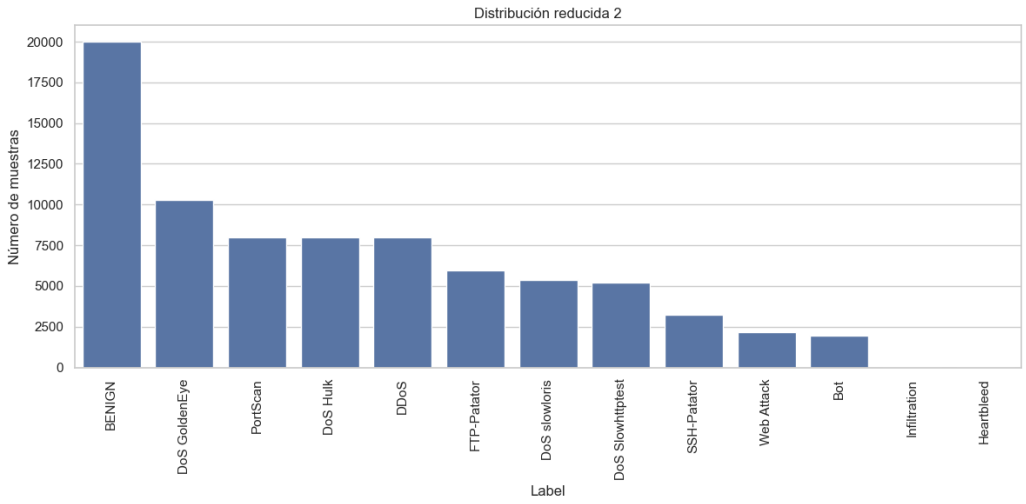
Gráfica de distribución reducida

Tabla de distribución de etiquetas

Label	Número de muestras
DoS GoldenEye	10286
BENIGN	10000
FTP-Patator	5931
DoS slowloris	5385
DoS Slowhttptest	5228
DoS Hulk	5000
DDoS	5000
PortScan	5000
SSH-Patator	3219
Web Attack	2143
Bot	1948
Infiltration	36
Heartbleed	11

Tabla de distribución reducida

B.6 Distribución reducida 2



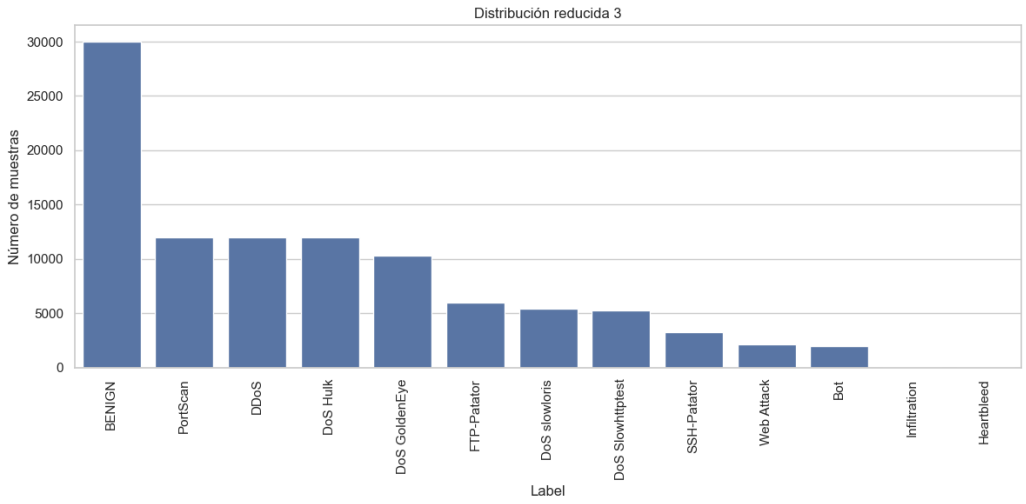
Gráfica de distribución reducida 2

Tabla de distribución de etiquetas

Label	Número de muestras
BENIGN	20000
DoS GoldenEye	10286
PortScan	8000
DoS Hulk	8000
DDoS	8000
FTP-Patator	5931
DoS slowloris	5385
DoS Slowhttptest	5228
SSH-Patator	3219
Web Attack	2143
Bot	1948
Infiltration	36
Heartbleed	11

Tabla de distribución reducida 2

B.7 Distribución reducida 3



Gráfica de distribución reducida 3

Tabla de distribución de etiquetas

Label	Número de muestras
BENIGN	30000
PortScan	12000
DDoS	12000
DoS Hulk	12000
DoS GoldenEye	10286
FTP-Patator	5931
DoS slowloris	5385
DoS Slowhttptest	5228
SSH-Patator	3219
Web Attack	2143
Bot	1948
Infiltration	36
Heartbleed	11

Tabla de distribución reducida 3

Apéndice C

Matrices de Confusión

C.1 Matrices Binarias

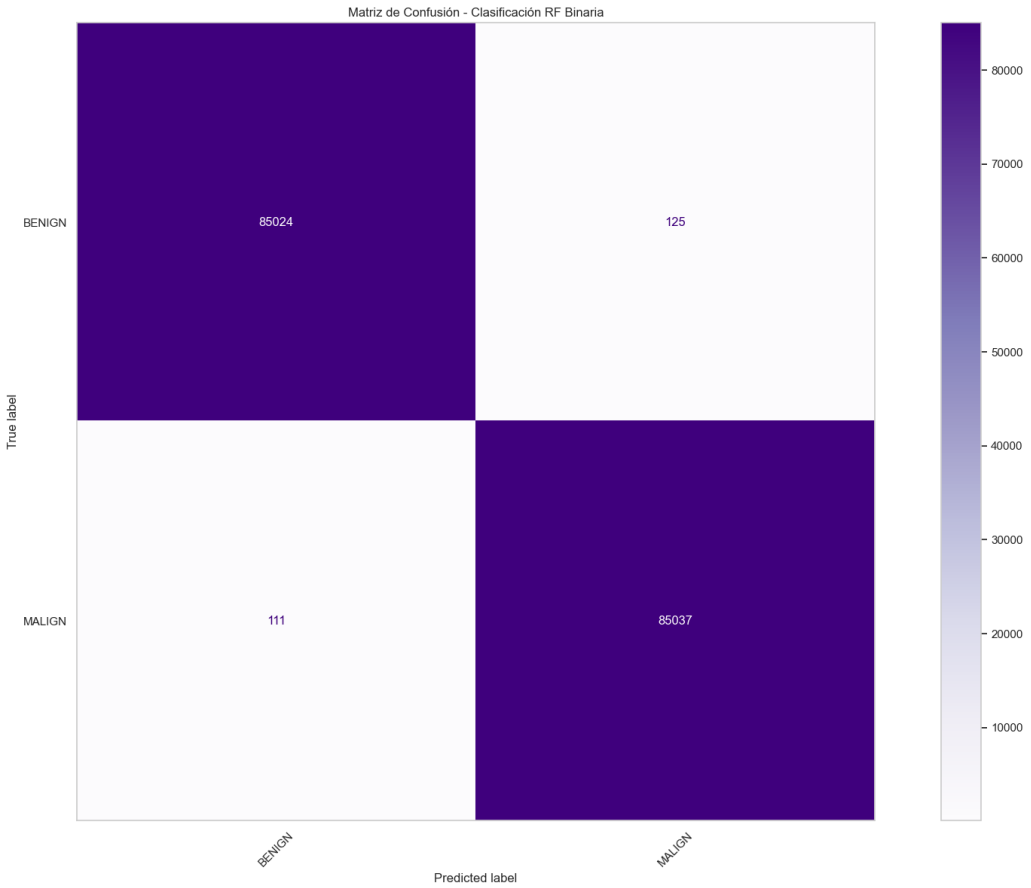


Figura C.1: Matriz de confusion Random Forest Binario

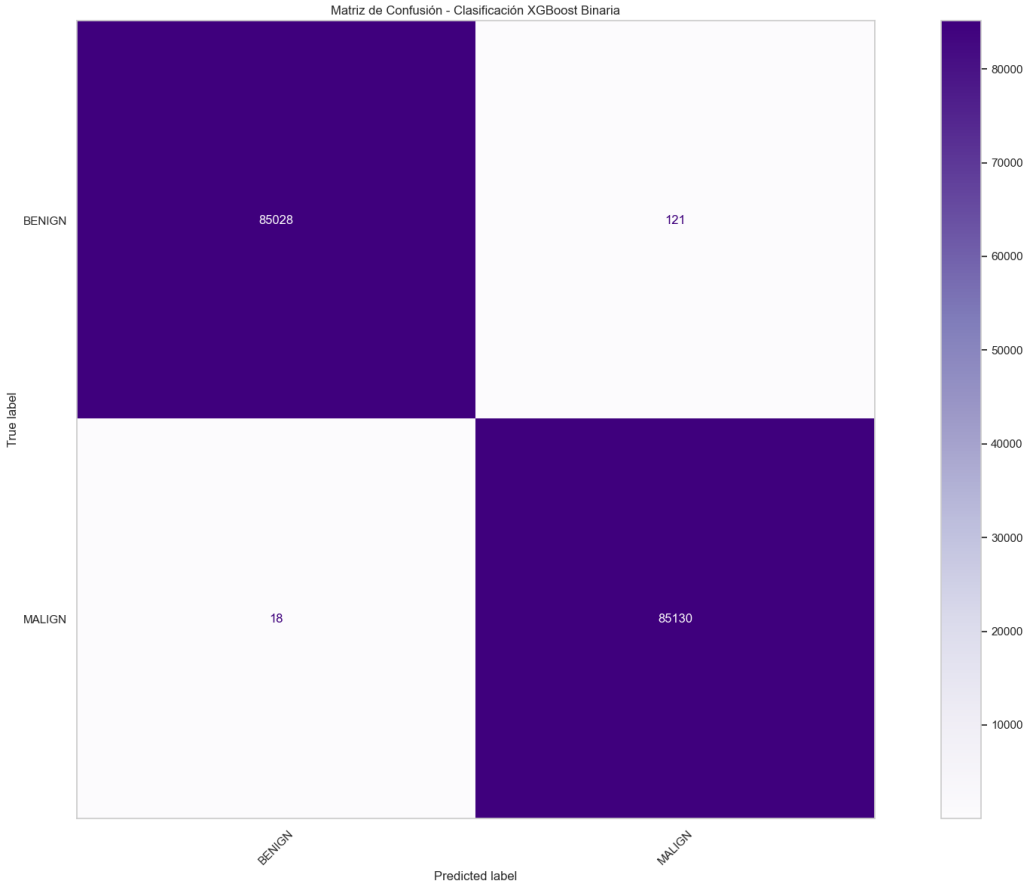


Figura C.2: Matriz de confusion XGBoost Binario

C.2 Matrices multiclase

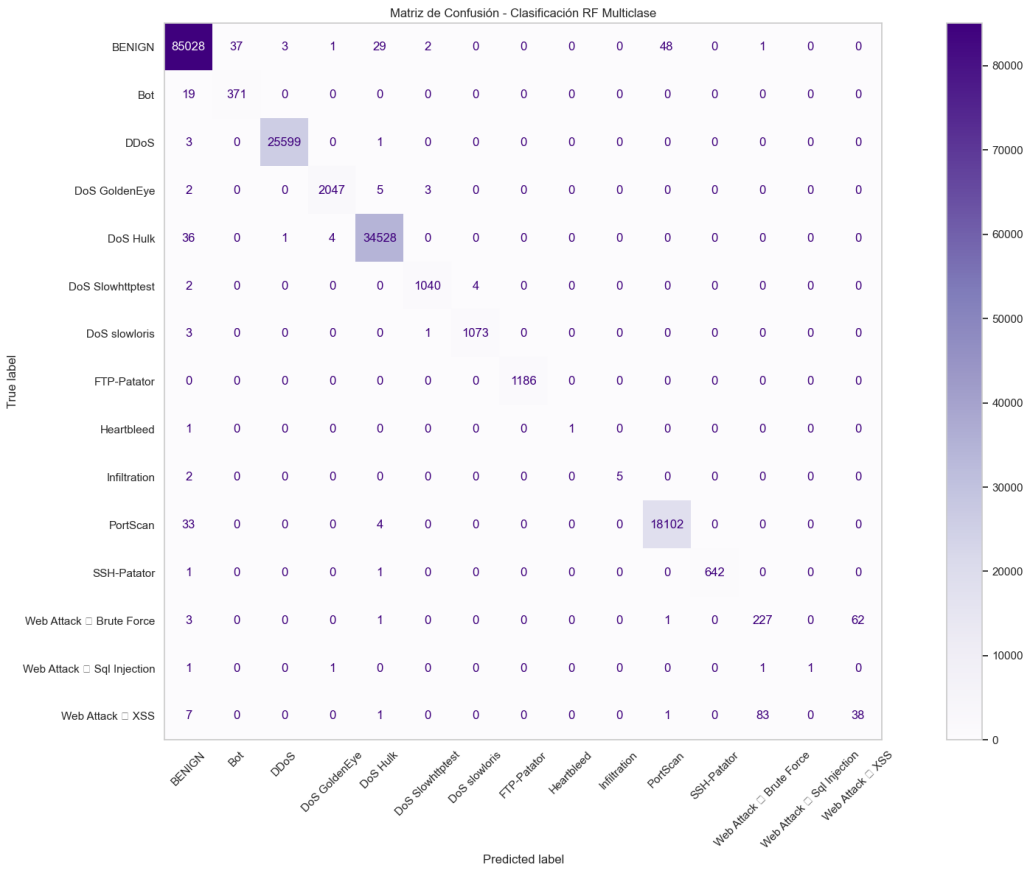


Figura C.3: Matriz de confusion Random Forest Multiclase

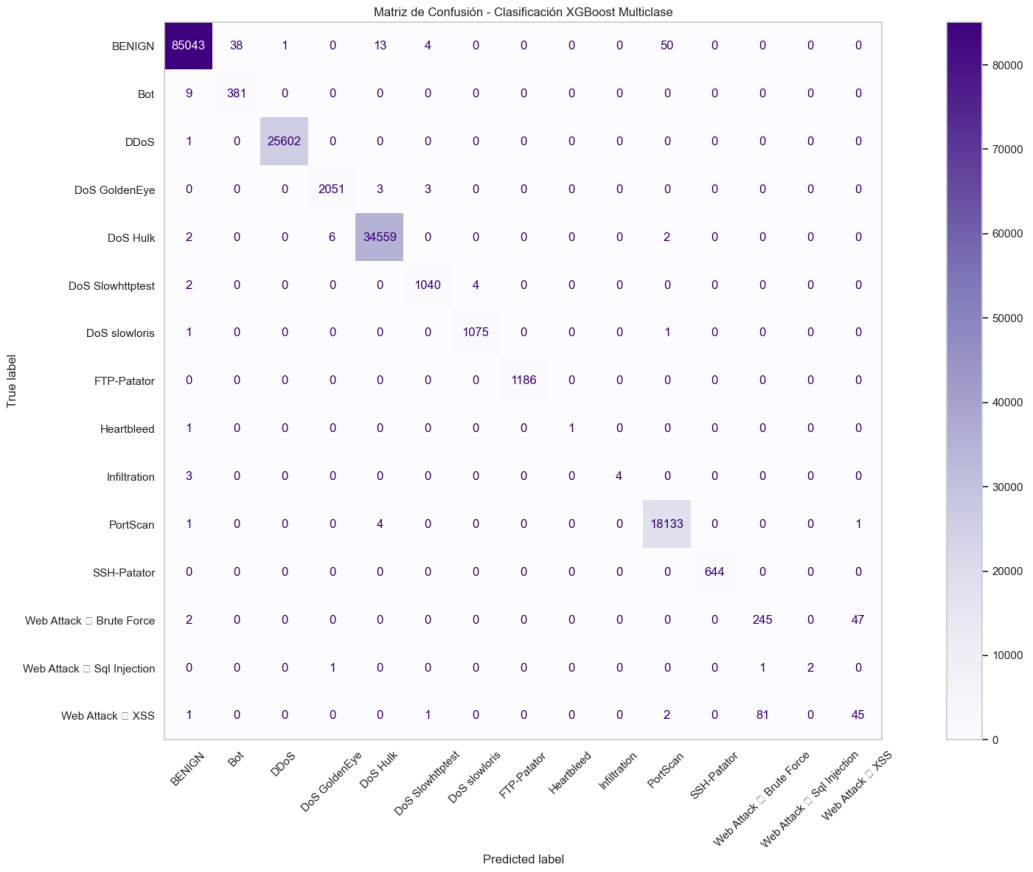


Figura C.4: Matriz de confusion XGBoost Multiclase

C.3 Matrices con ataques web agrupados

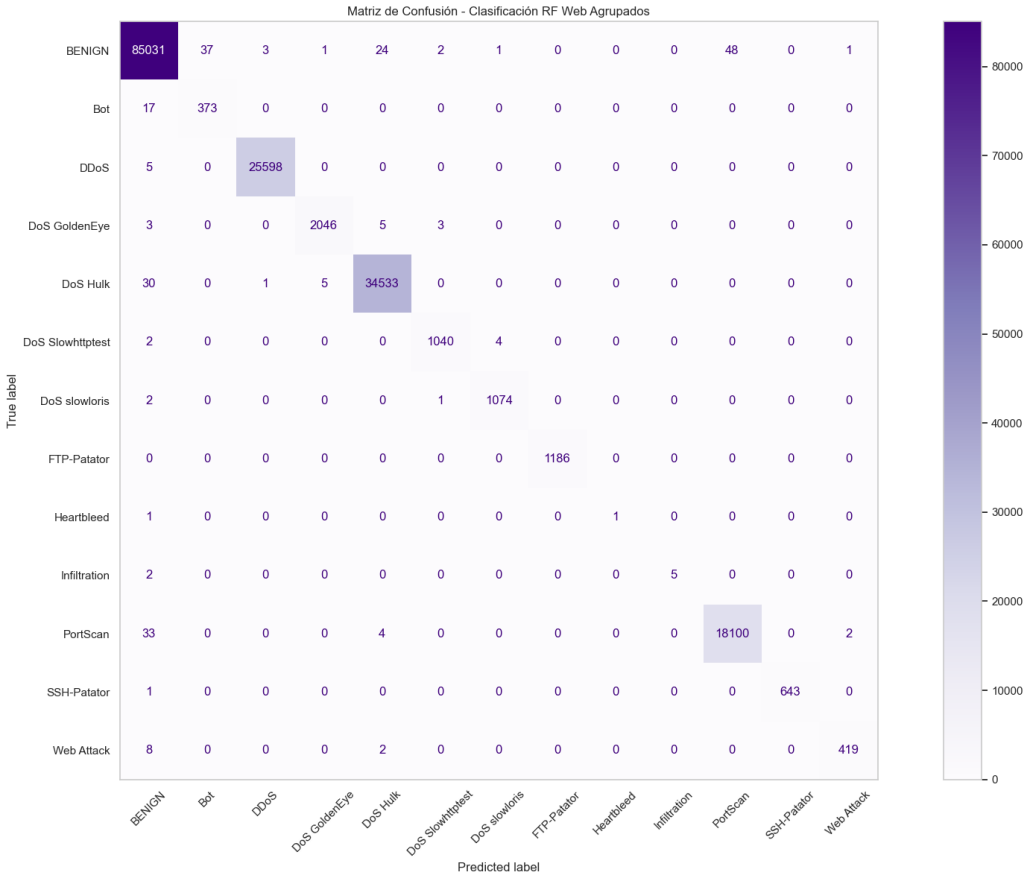


Figura C.5: Matriz de confusion Random Forest Ataques Web Agrupados

Figura C.6: Matriz de confusion XGBoost Ataques Web Agrupados

C.4 Matrices con dataset reducidos

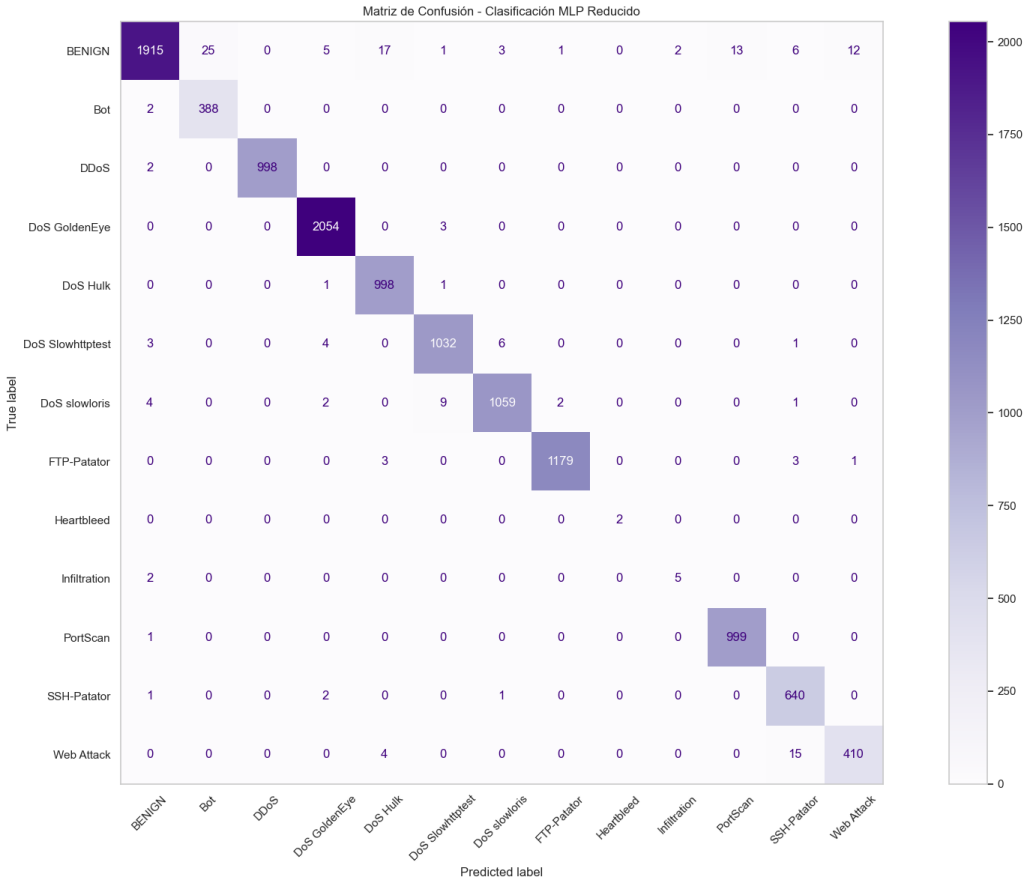


Figura C.7: Matriz de confusion MLP Reducida

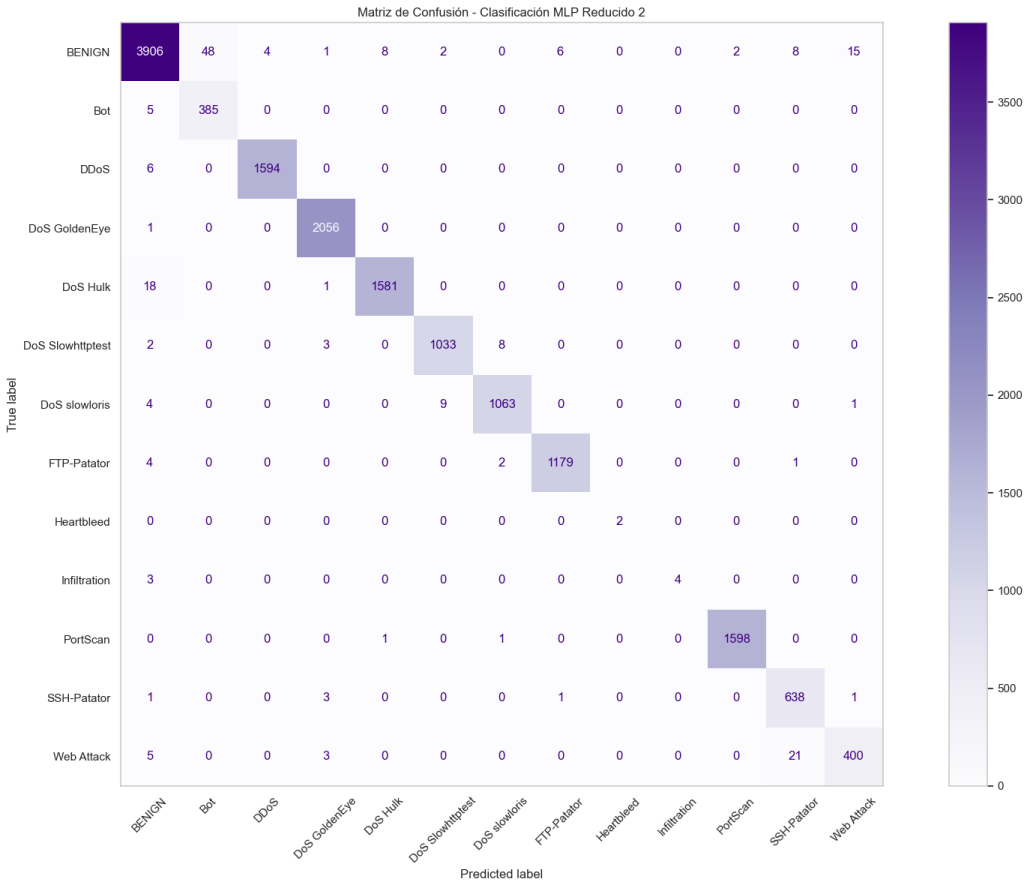


Figura C.8: Matriz de confusion MLP Reducida 2

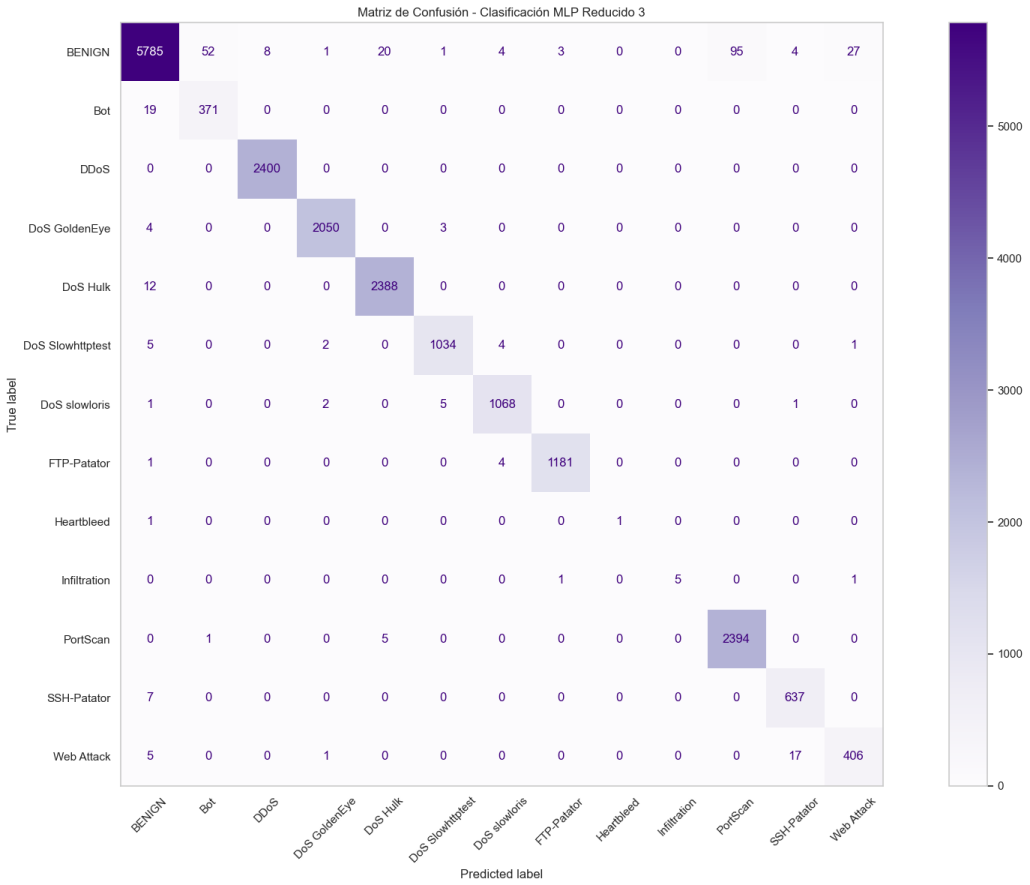


Figura C.9: Matriz de confusion MLP Reducida 3

Figura C.10: Matriz de confusion SVM Reducida

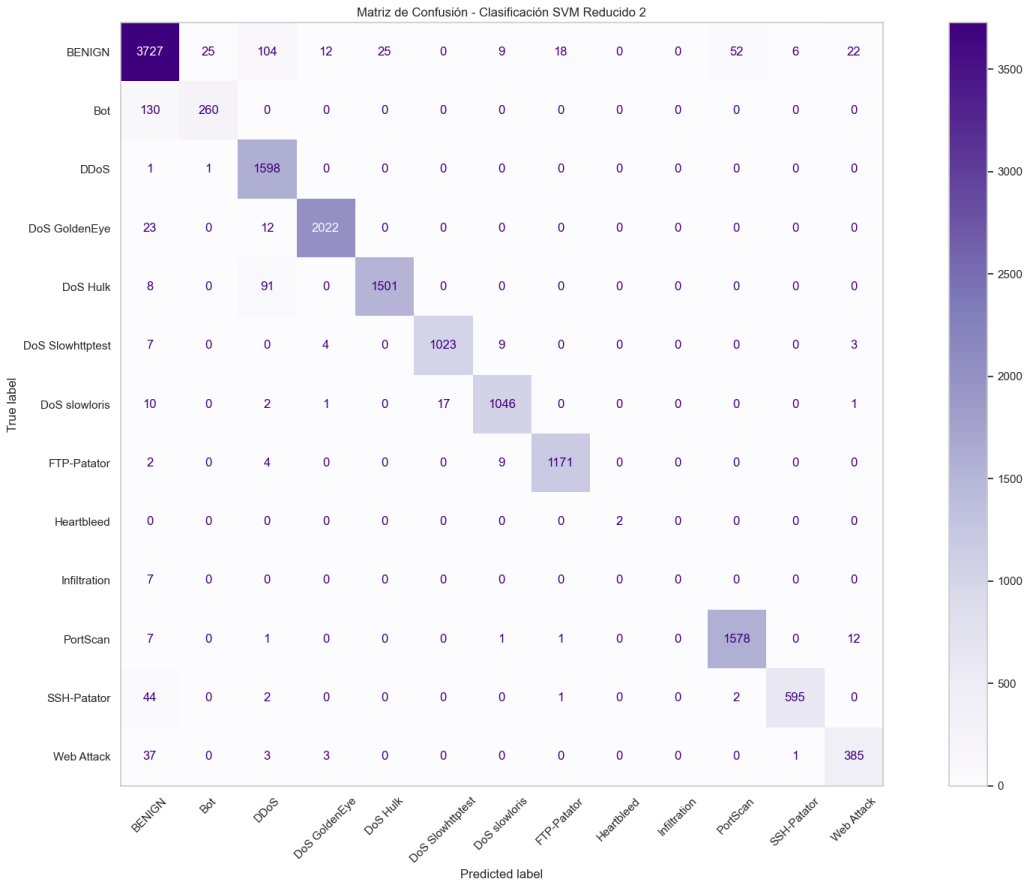


Figura C.11: Matriz de confusion SVM Reducida 2

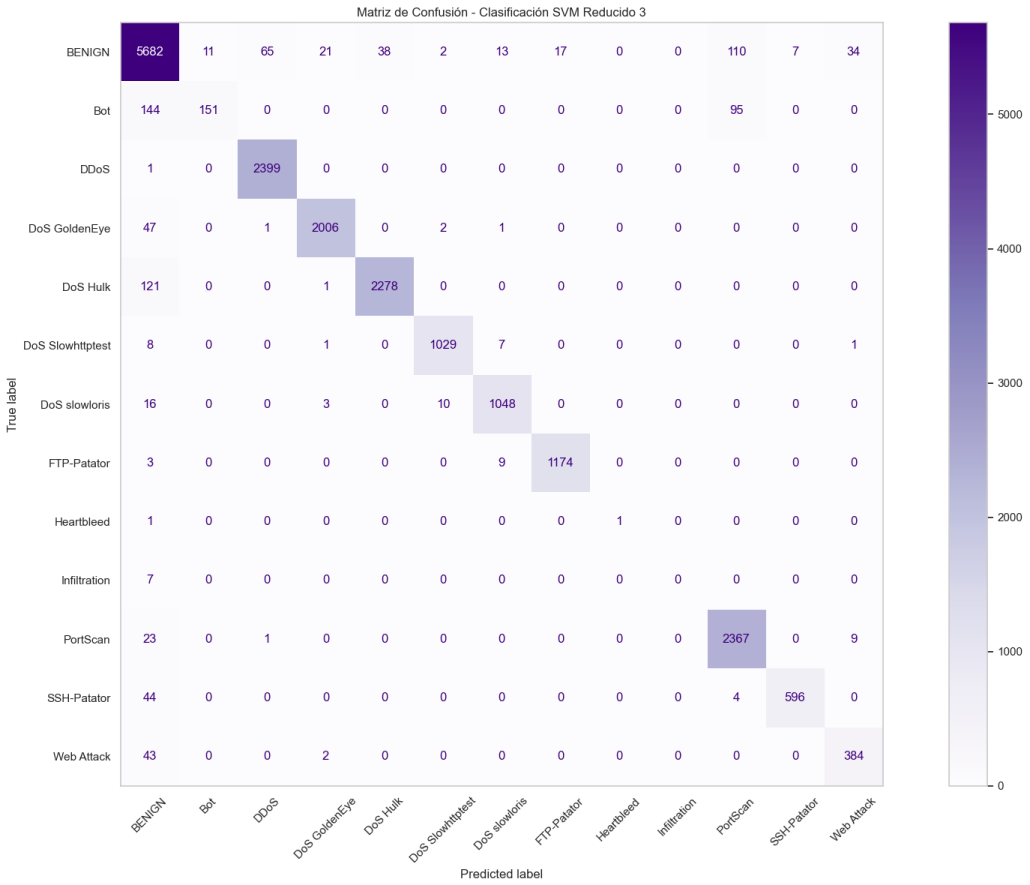


Figura C.12: Matriz de confusion SVM Reducida 3

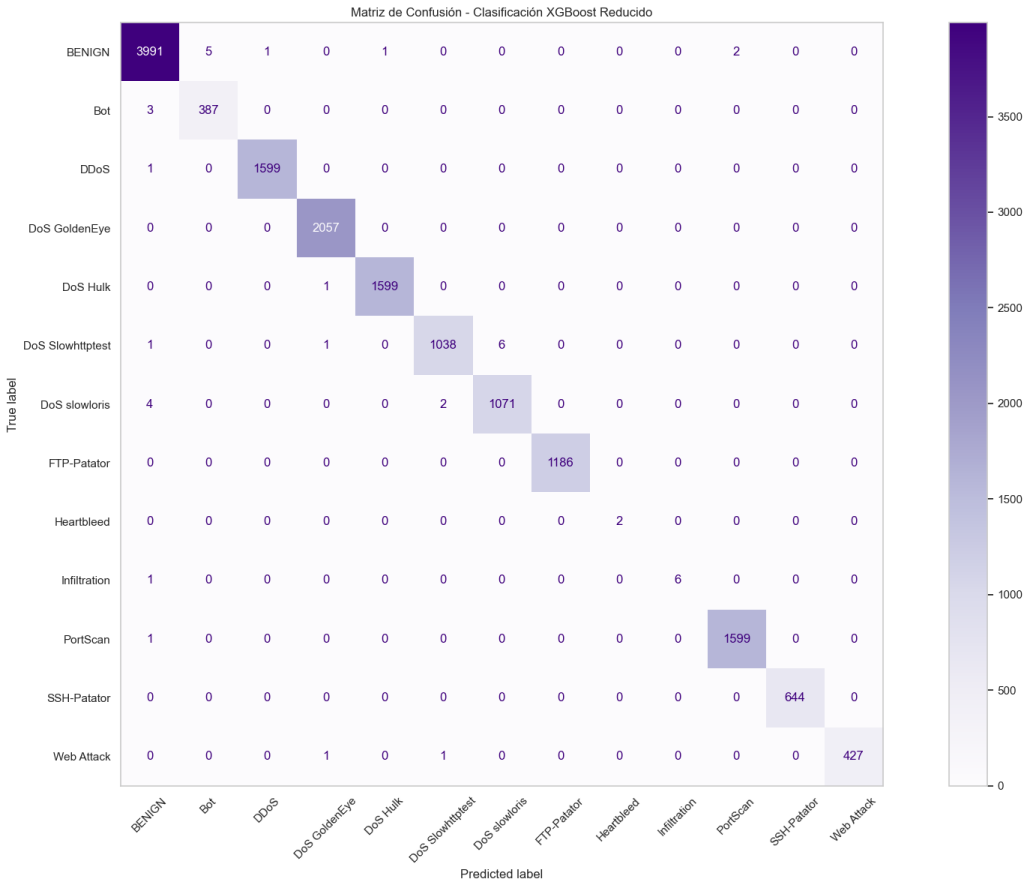


Figura C.13: Matriz de confusion XGBoost Reducida

C.5 Matrices con características reducidas

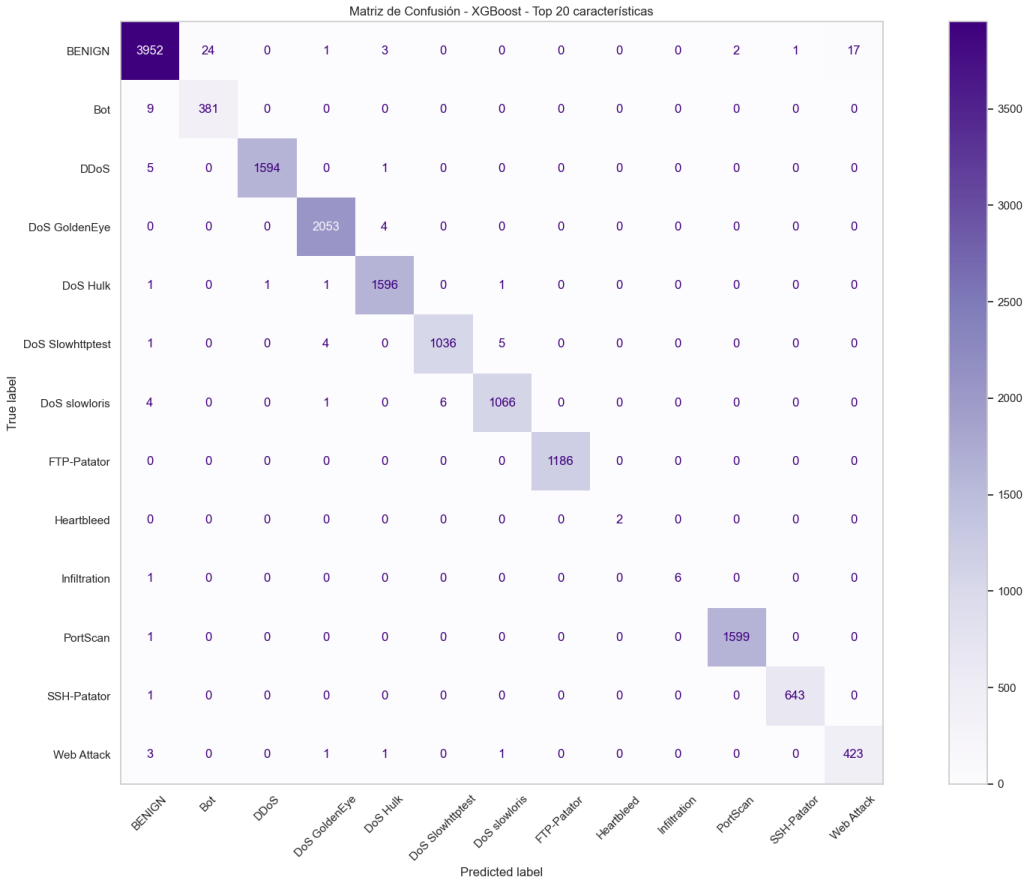


Figura C.14: Matriz de confusion XGBoost Top 20 Características

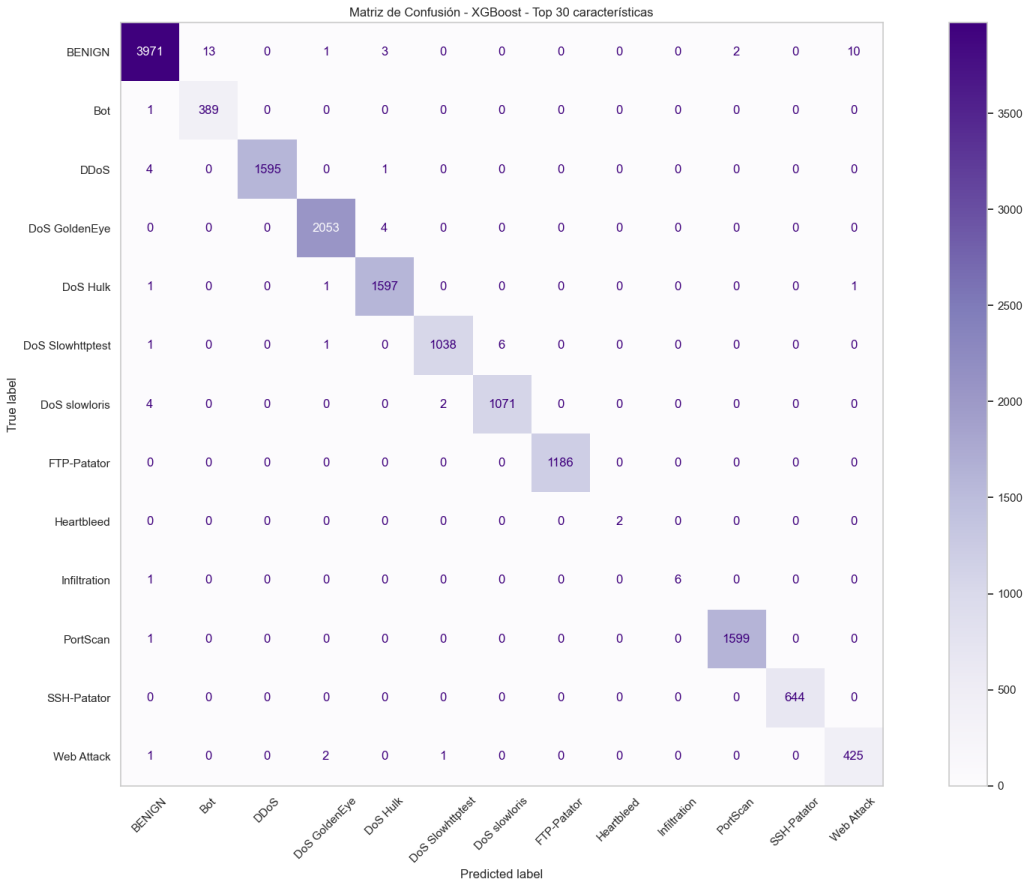


Figura C.15: Matriz de confusion XGBoost Top 30 Características

Bibliografía

[1] Leyla Bilge y Tudor Dumitras. «Before We Knew It: An Empirical Study of Zero-Day Attacks in the Real World». En: *Proceedings of the 2012 ACM Conference on Computer and Communications Security* (2012), págs. 833-844.

[2] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[3] Leo Breiman. «Random Forests». En: *Machine Learning* 45.1 (2001), págs. 5-32.

[4] Leo Breiman. «Random Forests». En: *Machine Learning* 45.1 (2001), págs. 5-32.

[5] Canadian Institute for Cybersecurity. *CIC-IDS2017 Dataset*. 2017. URL: https://www.unb.ca/cic/datasets/ids-2017.html.

[6] Nitesh V. Chawla et al. «SMOTE: Synthetic Minority Over-sampling Technique». En: *Journal of Artificial Intelligence Research*. Vol. 16. 2002, págs. 321-357.

[7] Tianqi Chen y Carlos Guestrin. «XGBoost: A Scalable Tree Boosting System». En: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM. 2016, págs. 785-794. URL: https://dl.acm.org/doi/10.1145/2939672.2939785.

[8] Cisco Systems. *Snort - Network Intrusion Detection System*. 2023. URL: https://www.snort.org/.

[9] Corinna Cortes y Vladimir Vapnik. «Support-vector networks». En: *Machine Learning* 20.3 (1995), págs. 273-297.

[10] Scikit-learn Developers. *Classification Metrics*. 2023. URL: https://scikit-learn.org/stable/modules/model_evaluation.html.

[11] Scikit-learn Developers. *Pipeline and Composite Estimators*. 2023. URL: https://scikit-learn.org/stable/modules/compose.html.

[12] European Union Agency for Cybersecurity (ENISA). *ENISA Threat Landscape 2023*. 2023. URL: https://www.enisa.europa.eu/publications/enisa-threat-landscape-2023.

[13] Matthias Finsterbusch et al. «A Survey of Payload-Based Traffic Classification Approaches». En: *IEEE Communications Surveys & Tutorials* 16.2 (2014), págs. 1135-1156.

[14] Git Contributors. *Git - Distributed Version Control System*. 2023. URL: https://git-scm.com.

[15] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt et al. «Array programming with NumPy». En: *Nature* 585 (2020), págs. 357-362. DOI: 10.1038/s41586-020-2649-2.

[16] Simon Haykin. *Neural Networks and Learning Machines*. 3.ª ed. Pearson, 2008.

[17] Haibo He y Edwardo A. Garcia. *Learning from Imbalanced Data*. 2009.

[18] Hunter, John D. *Matplotlib: Visualization with Python*. 3. 2007, págs. 90-95.

[19] IEEE DataPort Contributors. *IDS Packet Dataset - IEEE DataPort*. Dataset used for intrusion detection research. 2023. URL: https://ieee-dataport.org/keywords/ids-packet-dataset.

[20] Jupyter Team. *Project Jupyter*. 2023. URL: https://jupyter.org.

[21] Kaggle Contributors. *Simulated Military Network Traffic Dataset*. Dataset simulating military network traffic for IDS evaluation. 2023. URL: https://www.kaggle.com/datasets/sampadab17/network-intrusion-detection.

[22] Ron Kohavi. «A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection». En: (1995), págs. 1137-1143.

[23] Lamport, Leslie. *LaTeX: A Document Preparation System*. Addison-Wesley, 1994.

[24] Guillaume Lemaître, Fernando Nogueira y Christos K. Aridas. *Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning*. 2017.

[25] Microsoft. *Visual Studio Code*. 2023. URL: https://code.visualstudio.com.

[26] Open Information Security Foundation. *Suricata - Open Source IDS/IPS/NSM engine*. 2023. URL: https://suricata.io/.

[27] F. Pedregosa et al. «Scikit-learn: Machine Learning in Python». En: *Journal of Machine Learning Research* 12 (2011), págs. 2825-2830. URL: https://scikit-learn.org/.

[28] Python Software Foundation. *Python 3.11 Documentation*. 2023. URL: https://docs.python.org/3.11/.

[29] Sabrina Sicari et al. «Security, Privacy and Trust in Internet of Things: The Road Ahead». En: *Computer Networks* 76 (2015), págs. 146-164.

[30] William Stallings. *Network Security Essentials: Applications and Standards*. 4.ª ed. Pearson, 2012.

[31] The Pandas Development Team. *Pandas Documentation*. 2023. URL: https://pandas.pydata.org/docs/.

[32] Waskom, Michael L. *Seaborn: Statistical Data Visualization*. 2021. URL: https://seaborn.pydata.org.

[33] Zeek Project. *Zeek - Network Security Monitoring*. 2023. URL: https://zeek.org/.