INTEGRACIÓN DE TÉCNICAS DE INTELIGENCIA ARTIFICIAL Y SISTEMAS IDS EN LA DETECCIÓN DE ANOMALÍAS EN REDES DE DATOS

BAJO LA OPCIÓN:
TITULACIÓN INTEGRAL
TESIS

PARA OBTENER EL TÍTULO DE:
INGENIERO EN SISTEMAS COMPUTACIONALES

PRESENTA:
SAUL ISUI LUGO MARTINEZ

ASESOR:
DR. DAVID GONZALEZ MARRÓN

SINODALES:
M. en C. JORGE MARTÍNEZ MUÑOZ
M. en C. ANSELMO HERNÁNDEZ RAMÍREZ

Pachuca de Soto, Hgo. ABRIL 2024

130 páginas

Pachuca de Soto, Hidalgo., 22/marzo/2024
No. de oficio: DDSYC-149-2024
Asunto: Autorización de Titulación Integral.

C. SAUL ISUI LUGO MARTÍNEZ
PRESENTE

Por medio de la presente tengo el agrado de comunicar a usted, que habiéndose efectuado la revisión de su tema presentado a este Departamento, puede iniciar los trámites de titulación integral a través de:

Nombre del proyecto:	INTEGRACIÓN DE TÉCNICAS DE INTELIGENCIA ARTIFICIAL Y SISTEMAS IDS EN LA DETECCIÓN DE ANOMALÍAS EN REDES DE DATOS
Producto:	TESIS

Al mismo tiempo tengo a bien designarle como asesores del Proyecto de Titulación integral a los CC.

Presidente: Dr. David González Marrón
Secretario: M en C. Jorge Martínez Muñoz
Vocal: M en C. Anselmo Hernández Ramírez
Suplente: M.T.I. Luis Alejandro Santana Valadez

Sin más por el momento, me despido de usted.

ATENTAMENTE
Excelencia en Educación Tecnológica®
"El Hombre Alimenta el Ingenio en Contacto con la Ciencia" ®

EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA
TECNOLÓGICO NACIONAL DE MÉXICO
INSTITUTO TECNOLÓGICO DE PACHUCA
DEPARTAMENTO DE SISTEMAS Y COMPUTACIÓN

DR. ARTURO GONZÁLEZ CERÓN
JEFE DEL DEPARTAMENTO DE SISTEMAS Y COMPUTACION

Pachuca de Soto, Hidalgo, 26/abril/2024
No. de oficio: DDSYC-207-2024
Asunto: Liberación de proyecto
Para titulación integral.

LIC. LETICIA E. HERNÁNDEZ SAMPERIO
JEFA DEL DEPARTAMENTO DE SERVICIOS ESCOLARES
P R E S E N T E

Por este medio le informo que ha sido liberado el siguiente proyecto para la titulación integral.

Nombre del Egresado:	LUGO MARTINEZ SAUL ISUI
Carrera:	INGENIERÍA EN SISTEMAS COMPUTACIONALES
No. De Control:	19200205
Nombre del proyecto:	INTEGRACIÓN DE TÉCNICAS DE INTELIGENCIA ARTIFICIAL Y SISTEMAS IDS EN LA DETECCIÓN DE ANOMALÍAS EN REDES DE DATOS
Producto:	TESIS

Agradezco de antemano su valioso apoyo en esta importante actividad para la formación profesional de nuestros egresados.

Sin más por el momento, me despido de usted.

A T E N T A M E N T E
Excelencia en Educación Tecnológica®
"El Hombre Alimenta el Ingenio en Contacto con la Ciencia" ®

DR. ARTURO GONZÁLEZ CERÓN
JEFE DEL DEPARTAMENTO DE SISTEMAS Y COMPUTACION

EDUCACIÓN
INSTITUTO TECNOLÓGICO DE PACHUCA
DEPARTAMENTO DE SISTEMAS Y COMPUTACIÓN

ccp. Archivo
AGC/ecp

Agradecimientos

Quiero expresar mi sincero agradecimiento a todas las personas que han sido parte fundamental de este significativo proceso, contribuyendo de manera invaluable a mi trayectoria y permitiéndome alcanzar el punto en el que me encuentro hoy.

En especial, quiero dedicar un profundo agradecimiento a mi madre, Emi, por su apoyo incondicional a lo largo de este período de desarrollo, aprendizaje y desafíos. Agradezco también a mi familia, Francisco, Emmanuel, Grethel y Jonathan, por su incansable esfuerzo y ánimo diario.

No puedo pasar por alto el papel fundamental de mi asesor, el Dr. David González Marrón, cuya orientación y conocimiento han sido cruciales para la ejecución exitosa de este trabajo.

Además, deseo expresar mi más sincero agradecimiento al Dr. Alfonso Martínez Cruz, cuyo apoyo fue fundamental durante mi estancia científica.

Agradezco a mis compañeros, amigos y profesores por proporcionarme las herramientas, experiencias, y aprendizajes esenciales para abordar los desafíos de este proceso. Su contribución ha sido fundamental en mi formación académica y profesional.

Índice general

Índice de Figuras

Índice de Tablas

Resumen

La presente investigación aborda el análisis de registros provenientes de un Sistema de Detección de Intrusos (IDS), generados a partir de simulaciones de ataques cibernéticos. Estos registros conforman un conjunto de datos diseñado específicamente para la evaluación de anomalías en el tráfico de red, así como para la aplicación de clasificadores basados en algoritmos de aprendizaje automático.

En el primer capítulo, se introduce el trabajo de investigación, en donde se plantea el problema, su justificación, la motivación que impulsa la presente investigación, las preguntas de investigación, las hipótesis, los objetivos planteados, y su relación con los Objetivos de Desarrollo Sustentable (ODS).

El segundo capítulo realiza un análisis de tres trabajos previos, relacionados con el tema de investigación, exponiendo sus limitaciones, alcances y descubrimientos. En este contexto, se presenta una propuesta de valor para el presente estudio.

El tercer capítulo aborda los fundamentos teóricos mediante una exhaustiva investigación literaria, destacando la sección dedicada al análisis del campo de la inteligencia artificial y las redes neuronales.

El cuarto capitulo presenta el marco experimental de la investigación, detallando la ejecución de las pruebas y el despliegue del laboratorio virtual. Un apartado importante aborda la creación del conjunto de datos y su utilización en diversas herramientas para sus clasificación y análisis.

El quinto capítulo describe la evaluación de las pruebas, en el conjunto de datos, mediante la implementación de algoritmos de machine learning, para su clasificación. Posteriormente, se exponen los resultados y su discusión, seguidos de las conclusiones generales de la investigación, junto con una sección dedicada a los posibles trabajos futuros.

Palabras clave:

Sistema Detector de Intrusos, Ciberseguridad, Inteligencia Artificial, Aprendizaje Automático, Clasificadores, Predicciones, Ciberataques.

Abstract

This research addresses the analysis of records from an Intrusion Detection System (IDS), generated through simulations of cyber-attacks. These records constitute a dataset specifically designed for the assessment of network traffic anomalies and the application of classifiers based on machine learning algorithms.

In the first chapter, the research introduces the problem statement, its justification, the motivation driving the investigation, research questions, stated objectives, and their alignment with the Sustainable Development Goals (SDGs).

The second chapter provides a detailed analysis of three previous works related to the research topic, outlining their limitations, scopes, and findings. Within this context, a value proposition for the current study is formulated.

The third chapter delves into the theoretical foundations through an exhaustive literature review, with a particular emphasis on the section dedicated to artificial intelligence.

The fourth chapter presents the experimental framework of the research, detailing the execution of tests and the deployment of the virtual laboratory. A significant section addresses the creation of the dataset and its use in various tools for classification and analysis.

The fifth chapter describes the evaluation of the tests, on the data set, through the implementation of machine learning algorithms, for classification. Subsequently, the results and their discussion are presented, followed by the general conclusions of the research, along with a section dedicated to possible future work.

Keywords:

Intrusion Detection System, Cybersecurity, Artificial Intelligence, Machine Learning, Classifiers, Predictions, Cyber-attacks.

CAPÍTULO 1 INTRODUCCIÓN

El primer capítulo detalla la contextualización del proyecto de investigación, se presenta una introducción que establece el marco para el trabajo de investigación, se plantea y justifica el problema, destacando su relevancia y necesidad de estudio.

Posteriormente, se abordan los aspectos generales del proyecto, exponiendo las limitaciones del proyecto, así como el alcance que abarcará. Se dedica especial atención a la motivación como elemento fundamental que impulsa el desarrollo de esta investigación. A continuación, se formulan las preguntas de investigación y elaboran hipótesis que guían la presente investigación. Se plantea el objetivo general y los objetivos específicos, así como la relación con los objetivos de desarrollo sustentable de la ONU.

1.1 Introducción al trabajo de investigación

El presente trabajo de investigación aborda el análisis de técnicas y modelos de Inteligencia Artificial (IA), así como aprendizaje automático o machine learning (ML) con el objetivo de saber si mediante herramientas de software libre es posible detectar posibles ataques maliciosos a una red de datos.

Este trabajo constituye la continuación de la investigación que se inició en el Instituto Nacional de Astrofísica, Óptica y Electrónica (INAOE) del CONAHCYT, durante la edición del verano de investigación científica Delfín en 2023. La premisa del proyecto se originó con el análisis de aplicaciones móviles de tipo wearable, con el propósito de evaluar tanto la seguridad inherente a estas aplicaciones como las políticas de seguridad asociadas. Se busca determinar la fiabilidad de los datos que circulan a través del tráfico de red, considerando la sensibilidad de dicha información. Es imperativo contar con herramientas especializadas capaces de identificar posibles anomalías en el tráfico de red, así como establecer medidas de mitigación correspondientes.

En esta fase, se ha optado por proseguir con la evaluación de diversas técnicas de inteligencia artificial con el fin de detectar posibles actividades maliciosas o anómalas en el tráfico de red. Este enfoque se inscribe dentro de la búsqueda constante por mejorar

la seguridad y confiabilidad de las comunicaciones, particularmente en el contexto de información altamente sensible.

En el mes de septiembre de 2022, se registró uno de los incidentes de hackeo más destacados en la historia de México, focalizado en la Secretaría de la Defensa Nacional (SEDENA) [1]. Este episodio se tradujo en uno de los hackeos de mayor envergadura, donde aproximadamente seis terabytes de información, con categoría de confidencialidad militar, fueron objeto de una filtración. El grupo de Hacktivistas denominado “Guacamaya Leaks”, se atribuyó la responsabilidad de este acontecimiento, que requirió un extenso período para su ejecución. En contraposición, los administradores de la red, no lograron identificar ninguna anomalía respecto a la abrumadora cantidad de tráfico de red que caracterizó el desarrollo del ataqué.

Surge así la interrogante acerca de la posibilidad de prevenir este tipo de incidentes mediante la implementación de sistemas especializados en analizar el tráfico de red y detectar comportamientos anómalos o mal intencionados. La presente investigación se sustenta en la reflexión sobre la importancia de la ciberseguridad en el contexto mexicano y la necesidad de abordar estas problemáticas con enfoques científicos. ¿Puede la implementación de sistemas de análisis de tráfico de red prevenir eficazmente la perpetración de incidentes similares?, ¿Qué nivel de prioridad se otorga a la ciberseguridad en México?, ¿Existen investigaciones científicas consolidadas en el ámbito de la ciberseguridad? Estas preguntas constituyen el motor impulsor que motiva la ejecución de la presente investigación.

Con el propósito contextualizar el trabajo de investigación, se debe tomar en cuenta los conceptos de Inteligencia Artificial, Machine Learning, así como Ciberseguridad y su relación en el análisis de registros de trafico de red.

La definición de Inteligencia Artificial es compleja, porque no existe una definición estandarizada, hay distintos autores que proponen distintas definiciones, por ejemplo Russell y Norvig, proponen en [2] que “*La inteligencia artificial es la rama de la informática que se ocupa de construir programas capaces de realizar tareas que se requieren cuando son realizadas por personas*”, en [3], Nilsson propone que “*La inteligencia artificial consiste en el estudio de cómo las computadoras realicen cosas que,*

en este momento, la gente hace mejor". Si bien es muy atrevido consolidar la definición de Inteligencia Artificial, se puede inferir que es una disciplina de estudio de las ciencias de la computación y las matemáticas, que ocupa del desarrollo de sistemas capaces de realizar tareas que normalmente requieren de inteligencia humana. Estas tareas intentan imitar las actividades cognitivas, como puede ser el aprendizaje, razonamiento, la resolución de problemas, percepción, el reconocimiento de patrones, y comprensión del lenguaje natural.

Derivado de la Inteligencia Artificial, es como surge el concepto del Aprendizaje automático en ingles Machine Learning (será referido así de ahora en adelante), el cual ha sido objeto de diversas propuestas. Por ejemplo, en su obra [4], Dunning y Friedman, plantean que se trata de, "*un conjunto de técnicas y herramientas que permiten a las computadoras aprender de los datos sin ser programadas explícitamente*". En contraste, Mohri, Rostamizadeh y Talwalkar en [5], proponen que el Machine Learning se refiere a "*la creación de algoritmos de predicción precisos y eficientes*". Al evaluar la calidad de estos algoritmos de predicción, se considera la complejidad temporal y espacial de la muestra necesaria para aprender distintos conceptos, lo cual está directamente vinculado al análisis de datos y la teoría de la estadística.

En el enfoque, de intentar comprender la relación de la ciberseguridad con la inteligencia artificial, es necesario entender que es la ciberseguridad, por lo tanto se recurre a la definición proporcionada por la Agencia de Seguridad de Infraestructura y Ciberseguridad de los Estados Unidos (Cybersecurity and Infrastructure Security Agency, CISA, por sus siglas en inglés) [6], que expone que: "*La ciberseguridad es el arte de proteger redes, dispositivos y datos del acceso no autorizado o uso delictivo y la práctica de garantizar la confidencialidad, integridad y disponibilidad de la información*". A pesar de la claridad de esta definición, se plantea un cuestionamiento sobre la caracterización de la ciberseguridad como un "arte", ya que no existe una forma específica de garantizar que sea tal; sin embargo, se puede inferir que se trata de una disciplina práctica.

En una publicación reciente de la Agencia para la Ciberseguridad de la Unión europea [7], se aborda la relación entre el Machine Learning y la Ciberseguridad, está se enfoca mayormente en la detección de anomalías. Esto comprende la identificación de spam,

intrusiones, malware, ataques DoS, DDoS, múltiples peticiones, escaneo de puertos, reconocimiento de servicios y ataques de fuerza bruta entre otros. En el año 2014, Gartner [8] propuso el “Adaptive Security Architecture” un plan estratégico para la actualización de las prácticas de seguridad en las organizaciones. Este plan ha evolucionado hasta adoptar la denominación CARTA (Continuos Adaptive Risk and Trust Assessment, en español: Evaluación Continua y Adaptativa de Riesgos y Confianza), donde describe un enfoque constante y adaptable para evaluar de manera continua los riesgos y la confianza en un sistema determinado.

Como parte de CARTA, en 2016, Gartner propuso el modelo PPDR [9], el cual combina, técnicas de Machine Learning en el ámbito de la Ciberseguridad. Esté modelo, ilustrado en la Figura 1.1, muestra las funciones que debería desempeñar un sistema inteligente que integre de manera eficaz el Machine Learning con la Ciberseguridad.

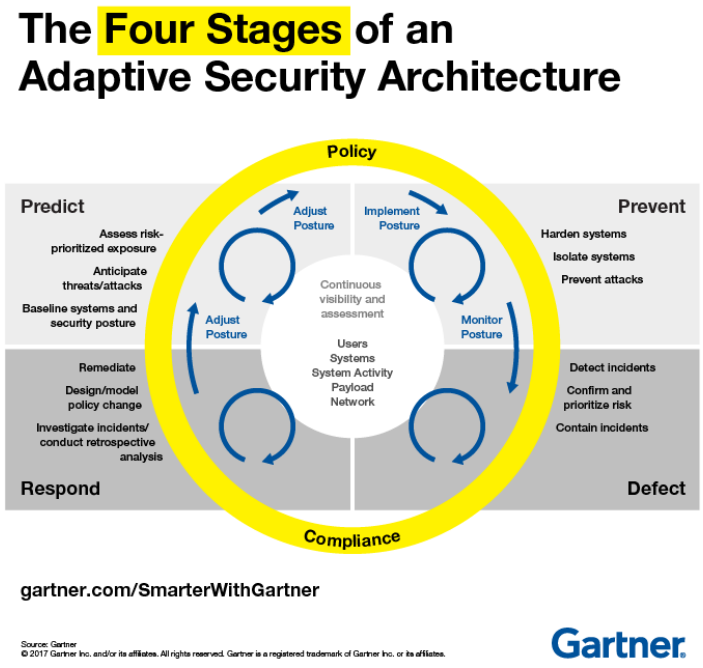


Figura 1.1 Módelo PPDR, propuesto por Gartner. Fuente [9]

1.2 Planteamiento del problema

Para el año 2022, datos de FortiGuard Labs [10], la organización de investigación de inteligencia de amenazas de Fortinet, en su Informe Global de Amenazas, indican que:

“América Latina fue el objetivo de más de 360 mil millones de intentos de ciberataques en 2022. México ocupando el primer lugar, con una cantidad mayor de 187 mil millones de intentos de ciberataques, seguido de Brasil con 103 mil millones y Colombia en tercer lugar con 20 mil millones”.

En otro informe, la Encuesta Nacional sobre Disponibilidad y Uso de Tecnologías de la Información en los Hogares (ENDUTIH) 2022 [11], realizada por el Instituto Nacional de Estadística y Geografía e Informática (INEGI), publicada por el Instituto Federal de Telecomunicaciones (IFT), que se presenta en la Figura 1.2., reveló que, en el año 2022, en México, había 93.1 millones de personas usuarias de internet, lo que representó 78.6% de la población de 6 años en adelante, conectadas a internet.

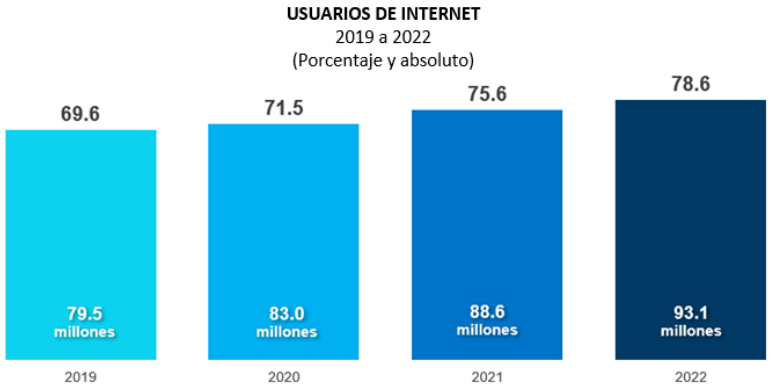


Figura 1.2 Usuarios de Internet 2019 a 2022, según ENDUTIH. Fuente [11]

Los datos presentados revelan una realidad preocupante en el panorama de ciberseguridad en México y América Latina, en donde México lidera la región con una asombrosa cifra, lo que indica una creciente amenaza cibernética que requiere una atención inmediata y estrategias de mitigación robustas.

La relación entre la cantidad significativa de intentos de ciberataques y el elevado número de usuarios conectados a internet en México, destaca la importancia de la concienciación y protección cibernética. La conectividad masiva presenta un entorno propicio para la proliferación de amenazas, y es crucial que los usuarios adopten prácticas de seguridad informática sólidas.

La alta incidencia de intentos de ciberataques en México resalta la importancia de una respuesta proactiva, inteligente y coordinada para salvaguardar la ciberseguridad en

la región, lo que será posible con la investigación y creación de sistemas inteligentes que detecten, aprendan y mitiguen, los posibles incidentes cibernéticos.

Surge la interrogante sobre la efectividad de las organizaciones para detectar el tráfico de red malicioso. Se plantea el escenario, con la aplicación de diversas técnicas de monitoreo, incluyendo el uso de técnicas de Inteligencia Artificial, en donde emerge como una estrategia fundamental. La pregunta que se plantea es si, ¿Es viable emplear herramientas de software libre para detectar tráfico malicioso de manera eficiente? algo que, en muchos casos, resulta inalcanzable con software convencional.

1.3 Justificación

El presente estudio de investigación se enfoca en áreas de sumo interés y relevancia, destacando la Inteligencia Artificial y el Machine Learning, así como su integración con la ciberseguridad. La convergencia de estos dominios estratégicos se presenta como un componente esencial para evaluar la eficacia de su aplicación conjunta.

Este análisis no solo adquiere relevancia en el ámbito académico, sino que también se propone como una guía para la implementación de proyectos y sistemas futuros, buscando armonizar las técnicas y algoritmos de Machine Learning en el contexto crítico de la ciberseguridad con la aplicación efectiva de algoritmos de aprendizaje automático es esencial para anticipar y combatir amenazas en constante evolución.

En cuanto al impacto económico, se destaca la eficiencia de esta investigación al minimizar los costos asociados. La implementación de tecnologías de software libre para el desarrollo del estudio reduce significativamente la carga financiera, optimizando recursos.

La aplicación de este proyecto de investigación se puede extender a diversas organizaciones, incluido el propio instituto donde se lleva a cabo la investigación. La información y conclusiones obtenidas tienen el potencial de contribuir significativamente al fortalecimiento de las prácticas de ciberseguridad, generando un impacto positivo, y sostenible.

1.4 Aspectos generales

Para esté trabajo de investigación se tomó en consideración; las limitaciones del proyecto y acotación del alcance.

1.4.1 Limitaciones

Las limitaciones de la presente investigación se encuentran principalmente vinculadas a restricciones de recursos en el ámbito financiero. La ausencia de apoyo económico por parte de la institución representó un significativo desafío, lo que conllevó a que el estudiante se viera obligado a llevar a cabo el proyecto de investigación con recursos propios, entre los que se destaca el equipo computo, servicio de internet, artículos científicos de paga mediante la membresía a IEEE Student, entre otros gastos. Lo que incidió directamente en la capacidad para realizar una investigación mucho más extensa y rigurosa.

En términos de infraestructura tecnológica, se dispuso de un equipo de cómputo tipo laptop con capacidades muy limitadas que se pueden ver en la Tabla 1.1.

La institución proporcionó un equipo de cómputo para realizar pruebas y simulación, pero coincidió que, durante la ejecución de la investigación, existió un evento que impidió el acceso a las instalaciones del instituto, tales que salieron del alcance del alumno e instituto hacer uso de dicho equipo. La limitación en recursos de cómputo ha repercutido en la ejecución de simulaciones, resultando en una muestra significativamente menor a la inicialmente planteada. Las pruebas se realizaron exclusivamente utilizando el poder de cómputo de la laptop, por medio de un laboratorio de pruebas en máquinas virtuales ejecutadas localmente.

Tabla 1.1 Equipo de computo. Fuente: Elaboración propia.

Especificaciones técnicas del equipo de cómputo	
Procesador	Intel i5-8250U 1.60GHz
Memoria RAM	32 GB 2400MHz
Velocidad de internet promedio	75 Mbps

Está limitación financiera ha influido en la elección de recursos, que podría afectar la representatividad de la muestra obtenida para la investigación.

Todas las pruebas fueron realizadas exclusivamente como se mencionó anteriormente en un entorno controlado, debido a las restricciones legales y éticas que limitaron la realización de experimentos en un entorno más amplio o con acceso a recursos más avanzados.

1.4.2 Alcance

El alcance del proyecto se encuentra definido por los siguientes puntos:

- Realizar el análisis de las técnicas de Inteligencia Artificial, actualmente disponibles, con énfasis en aquellas que se muestren idóneas para el análisis de tráfico de red.
- Llevar a cabo la simulación de posibles ataques de red que afectan a las redes, estableciendo una clasificación que permita la compresión detallada de las diferentes amenazas cibernéticas.
- Implementación de un laboratorio de pruebas, con herramientas de software libre.

Este conjunto de actividades delimita el ámbito y las fronteras de la investigación, enfocándose en aspectos fundamentales que abarcan la revisión de la literatura, hasta la implementación práctica.

1.5 Motivación

La motivación subyacente para la ejecución de esta investigación se origina en el impulso intrínseco común a la mayoría de las indagaciones científicas. Se fundamenta por la curiosidad científica y deseo de comprender la aplicación de modelos de inteligencia artificial en el ámbito de la ciberseguridad, esté interés no solo responde a la necesidad de comprender los fundamentos teóricos, sino que también busca explorar las aplicaciones prácticas de estas tecnologías emergentes en la protección y análisis de sistemas de redes de datos.

Desde el inicio de mi formación académica y profesional, siempre tuve claro que mi campo de especialización sería la ciberseguridad, incluso antes de ingresar al nivel superior, ya había establecido ese objetivo. Gracias a toda la formación recibida durante

la carrera, he adquirido las herramientas necesarias para abordar problemas complejos de ingeniería en el ámbito de la ciberseguridad y sus diversas aplicaciones. Durante el transcurso de la carrera, cursé tres asignaturas que fueron muy significativas para mí: “Ciberseguridad”, “Inteligencia artificial” y “Taller de investigación II”. Estas asignaturas no solo proporcionaron una base sólida, sino que también marcaron el inicio de mi apasionado interés por la investigación.

Durante el verano de 2023, partícipe activamente en el Programa Interinstitucional para el Fortalecimiento de la Investigación y el Posgrado del Pacífico, conocido como el programa Delfín. Durante esta temporada, llevé a cabo una estancia de investigación científica en el Instituto Nacional de Astrofísica, Óptica y Electrónica (INAOE) del CONAHCYT, específicamente en la línea de investigación de ciberseguridad. Esta experiencia se reveló como un capítulo profundamente enriquecedor en mi desarrollo personal, profesional y académico, sirviendo como catalizador para mi compromiso continuo con una carrera científica.

Los aprendizajes obtenidos durante esta experiencia no solo consolidaron mi perspectiva académica, sino que también fortalecieron mi motivación para continuar la presente investigación.

1.6 Preguntas de investigación

Las preguntas de investigación que se desarrollan son las siguientes:

1. ¿Qué algoritmos de clasificación, serán el más apto para la detección de posibles ciberataques?
2. ¿Es posible la creación de un conjunto de datos mediante distintas simulaciones de trafico de red de datos?
3. ¿Cómo se podrían identificar anomalías en una red de datos mediante técnicas de monitoreo combinadas con técnicas de inteligencia artificial?

1.7 Hipótesis

Las hipótesis planteadas son las siguientes:

H1: Determinar cuál es un buen algoritmo de clasificación para detectar un ataque a una red de datos.

H2: Creación de un conjunto de datos mediante la simulación de trafico de red, para su análisis, este tráfico simulara peticiones normales y posibles ciberataques.

H3: Identificar de anomalías de una red de datos, mediante técnicas de monitoreo e Inteligencia Artificial.

1.8 Objetivo general

Realizar un análisis de las diferentes técnicas de inteligencia artificial para evaluar la posibilidad de detectar tráfico de red malicioso, mediante el uso de herramientas de software libre.

1.8.1 Objetivos específicos

- Seleccionar un grupo de algoritmos de clasificación, para la posible detección de una anomalía en una red de datos.
- Interceptar tráfico de red mediante un sistema de detección de intrusos por medio de simulaciones, para la creación de un conjunto de datos.
- Evaluar el porcentaje de efectividad de algoritmos clasificadores de machine learning para la detección de tráfico de red malicioso.

1.8.2 Los objetivos y su relación con los ODS

Los Objetivos de Desarrollo Sostenible (ODS), de la ONU, vinculados con esta investigación son los siguientes:

4. **ODS 9, Industria, Innovación e Infraestructura**: La investigación aborda la innovación en el campo de la ciberseguridad y la aplicación de técnicas de inteligencia artificial en la detección de trafico de red malicioso. La mejora de infraestructura tecnológica para la seguridad es esencial para el desarrollo sostenible.
5. **ODS 16, Justicia e Instituciones Sólidas**: La detección y prevención de tráfico de red malicioso contribuye a la estabilidad de las instituciones y sistemas.

6. **ODS 17, Alianzas para lograr los Objetivos**: La investigación implica el uso de Software Libre, y se basa en investigaciones previas. La colaboración y el intercambio de ideas son esenciales para abordar desafíos y lograr los ODS.

La presente investigación no solo se centra en aspectos técnicos, sino que también tiene el potencial de contribuir al logro de varios ODS al abordar cuestiones fundamentales de seguridad, colaboración e innovación.

CAPÍTULO 2 TRABAJOS RELACIONADOS

Este capítulo proporciona una revisión de estudios de investigación, analizando sus metodologías, ventajas, limitaciones y tecnológicas aplicadas.

La síntesis de la información recopilada se enfoca en destacar los aspectos más significativos de cada estudio, con especial énfasis en la implementación de diversas técnicas de inteligencia artificial.

Finalmente se presenta la propuesta de valor de esta investigación, estableciendo su relevancia y contribución única en comparación con los trabajos previos en el dominio de la detección de tráfico de red malicioso mediante inteligencia artificial.

2.1 Comparative analysis of Machine Learning algorithms for Intrusion Detection

El estudio, de Vasudeva Pai y otros autores [12], se enfoca en la aplicación de técnicas de machine learning, para detectar tráfico de red malicioso, subrayando la importancia y los desafíos de mantener la seguridad de la red en la era moderna. Se destaca que las herramientas tradicionales como los firewalls (corta fuegos), y antivirus, no siempre son efectivas contra los crecientes ataques cibernéticos.

Se enfatiza en los Sistemas de Detección de Intrusiones (IDS) que monitorean los procesos de la red para detectar desviaciones de las operaciones normales (en anormales). Se clasifican los IDS en cinco categorías e identifican dos métodos principales de detección: basada en firmas y basada en anomalías.

La metodología del articulo describe los pasos para construir un modelo de machine learning para la detección de intrusiones, que incluyen preparación de datos, preprocesamiento, selección de características, visualización de datos, agrupamiento, entrenamiento, y validación del modelo. Se emplean algoritmos de machine learning, utilizando el conjunto de datos NSL-KDD.

El conjunto de datos NSL-KDD (National Science Foundation (NSF)'s Labelled KDD Data) es una versión mejorada del conjunto de datos KDD Cup 1999, creado para la Competición Internacional de Descubrimiento de Conocimiento en Bases de Datos (KDD

Cup) en el año 1999. El KDD Cup es un evento anual que fomenta la investigación en el campo de la minería de datos y el descubrimiento de conocimiento en grandes bases de datos.

El conjunto de datos NSL-KDD fue propuesto en el año 2009 como una mejora al conjunto original KDD Cup 1999. El objetivo principal de NSL-KDD es abordar algunas limitaciones y desafíos encontrados en el conjunto de datos KDD Cup 1999, como la presencia de redundancias y la falta de diversidad en los tipos de ataques. NSL-KDD ha sido estructurado para ser más representativo de escenarios del mundo real en términos de la variedad de ataques y la distribución de clases.

El conjunto de datos NSL-KDD tiene como propósito principal servir como un recurso de evaluación para sistemas de detección de intrusiones en redes. Los investigadores y profesionales en ciberseguridad lo utilizan para entrenar y evaluar algoritmos y modelos de aprendizaje automático diseñados para identificar patrones y comportamientos anómalos en redes. NSL-KDD incluye una variedad de tipos de ataques simulados, lo que lo hace más representativo del panorama de amenazas real. Contiene instancias de ataques DoS, U2R (User to Root), R2L (Remote to Local), entre otros tipos de ciberataques.

La sección de algoritmos detalla los utilizados en el estudio; Random Forest, Naive Bayes, Support Vector Machine, J48, Regresión logística, y árboles de decisión.

La sección de resultados, presenta el rendimiento de los clasificadores (algoritmos de machine learning) seleccionados contra diferentes tipos de ataques, como por ejemplo “Denegación de Servicio” (DoS), mostrada en la Tabla 2.1, en donde el algoritmo Random Forest, mostró una alta exactitud del 99.83% y un puntaje de precisión perfecto de 1, contra ataques DoS.

Tabla 2.1 Rendimiento de clasificadores frente a un ataque DoS. Fuente [12]

Machine Learning algorithms	Accuracy (%)	Precision	Recall	F1-Score	ROC
Random Forest	**99.83**	**1**	**0.999**	0.999	**0.999**
J48	99.76	0.99	**0.999**	0.999	0.658
Logistic Regression	99.47	0.99	0.998	0.999	0.727
Decision Tree	99.66	0.99	0.999	**1.0**	0.618
SVM	99.19	0.98	0.999	0.999	0.5
Naïve Bayes	94.19	0.97	0.903	0.948	0.844

Si bien el articulo proporciona una base fundamental sólida, adolece de la ausencia de un conjunto de datos propio, particularmente uno generado en un laboratorio que emule datos específicos y contextualmente relevantes. Mientras que el empleo de un conjunto de datos preexistentes representa una estrategia viable, tal conjunto puede tener limitaciones, como desactualizaciones. Quizá lo ideal sería la implementación de un laboratorio de pruebas dedicado que genere datos propios, lo que permitiría una evaluación más precisa y actualizada de los algoritmos.

2.2 Intrusion Detection Systems using Supervised Machine Learning Techniques: A survey

El artículo de E. Abdallah, W. Eleisah y F. Otoom, "*Intrusion Detection Systems using Supervised Machine Learning Techniques: A survey*" [13], describe una revisión de los sistemas de detección de intrusiones (IDS) basados en técnicas de machine learning supervisado. Se centra en proporcionar una taxonomía de estos sistemas y los algoritmos relacionados.

El estudio compara el rendimiento de distintos algoritmos, haciendo uso de cuatro conjuntos de datos (data sets) como KDD'99, NSL-KDD, CICIDS2017, y UNSW-NB15. Se resalta la importancia de la selección de características y el equilibrio de datos para mejorar la eficacia de estos sistemas en la identificación precisa de las amenazas de seguridad. En la Figura 2.1 se presenta la descripción tomada del artículo, de cada conjunto de datos.

Dataset	Description
KDD'99	It is generated using simulation of normal and attacks traffic in a military environment (US AirForce LAN). It contains nine weeks of simulation in rat tcpdump files. The dataset is characterized using 41 features related to intrinsic, content, and traffic. Four types of attacks are simulated: DoS, Prob, U2R, and R2L.
NSL-KDD	It is a modification to the KDD'99 dataset with solving the problems of redundancy, duplicates, the imbalance of data.
UNSW-Nb15	It was created using the IXIA PefectStorm tool to extract normal and attack network traffic based on 100 GB of raw network traffic. It is characterized using 49 features. It consists of around 175 thousand records for training and around 82 thousand records for testing. There are nine types of attacks: Fuzzers, Analysis, Backdoor, DoS, Exploit, Generic, Reconnaissance, Shellcode, Worm
CICIDS2017	It was created in an emulated environment in a 5 day period. It contains traffic in packet flow and bidirectional flow. 80 features are extracted. Attacks involve: Brute Force FTP, Brute Force SSH, DoS, Heartbleed, Web Attack, Infiltration, Botnet, and DDoS

Figura 2.1 Descripción de los distintos conjuntos de datos. Fuente [13]

El articulo concluye que los algoritmos de machine learning supervisado demuestran un rendimiento alto y prometedor en la detección de intrusiones, se resalta la importancia de la selección de datos de buena calidad. Con el propósito de mejorar la precisión y eficiencia de estos sistemas. En la Figura 2.2, se presenta la taxonomía propuesta por los autores, como resultado de la investigación.

Paper	Dataset	Is feature selection applied?	Is feature selection useful?	Supervised learning algorithms	Validation method	Best performing learning method	Best Reported results
[12]	NSL-KDD	Yes	Yes	J48, Random forest	10-fold	Random forest	Accuracy = 99.7 % FPR = 0.005 %
[24]	NSL-KDD	Yes	Yes	Naive Bayes, Bayes-Net, Logistics, Random forest, J48, Bagging, OneR, PART, ZERO	10-fold	Random forest	Accuracy = 99.9 % FPR = 0.001%
[25]	NSL-KDD	NO	NA	SVM, GMM, Random forest, logistics regression	Hold-out	Random forest	Accuracy = 99 % FPR= NA
[26]	NSL-KDD (20% sample)	YES	Results before feature selection are NA	Artificial neural networks (ANN), SVM	Hold-out	ANN	Accuracy = 94.0% FPR=NA
[27]	KDD'99	YES	Results before feature selection are NA	k-means, random forest, naïve bayes, SVM	Hold-out	Random forest	Accuracy = 99.0% FPR=NA
[28]	KDD'99	YES	NO	SVM	10-fold	SVM	Accuracy=98.7% FPR=NA
[29]	KDD'99	YES	NO	BBNN	10-fold	BBNN	Accuracy=91.0% FPR=NA
[30]	CICIDS2017	YES	YES	Adaboost	Hold-out	Adaboost	Accuracy=81.83% FPR=NA
[31]	CICIDS2017	YES	NA	LDA, QDA, BN, RF	Hold-out	QDA	Accuracy=98.8% FPR=.001%
[32]	CICIDS2017	YES	NA	ANN, RF	Hold-out	RF	Accuracy=96.4
[33]	UNSW-NB15	YES	YES	SVM, J48, RF, Zero	10-fold	J48	Accuracy=96.7% FPR=.13%
[34]	UNSW-NB15	YES	YES	DNN, RF	5-fold	DNN	Accuracy=97.0% FPR=NA

Figura 2.2 Taxonomia de los sistemas IDS. Resultados de la investigación. Fuente [13]

Este artículo se centra en revisar y sintetizar la literatura existente sobre el uso de técnicas de machine learning, en IDS. No se presenta como un experimento científico con nueva recopilación de datos o pruebas experimentales propias, sino más bien como un análisis literario de estudios previos y sus hallazgos en este campo.

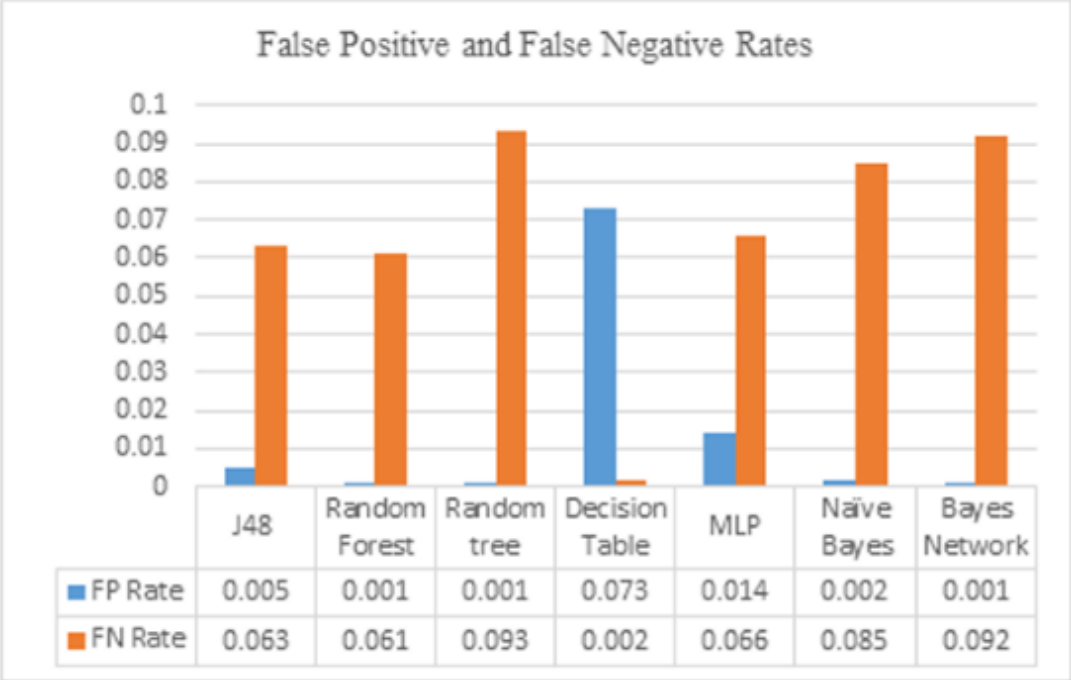
2.3 Evaluation of Machine Learning Algorithms for Intrusion Detection System

En el artículo de Mohammad Almseidin y otros, "Evaluation of Machine Learning Algorithms for Instrusion Dectection System" [14], publicado en el IEEE 15th International Symposium on Intelligent Systems and Informatics en 2017, se presenta un estudio

experimental sobre la eficacia de varios algoritmos de machine learning en la detección de intrusiones.

Se realizan pruebas en Weka con distintos clasificadores utilizando el conjunto de datos de intrusión KDD. El estudio se enfoca en métricas de rendimiento como falsos negativos y positivos para mejorar la tasa de detección de los IDS.

En la Figura 2.3, obtenida de [14], se observan las tasas de Falsos Positivos (FP) y Falso Negativos (FN) para diferentes algoritmos de machine learning utilizado en un IDS.



	J48	Random Forest	Random tree	Decision Table	MLP	Naïve Bayes	Bayes Network
FP Rate	0.005	0.001	0.001	0.073	0.014	0.002	0.001
FN Rate	0.063	0.061	0.093	0.002	0.066	0.085	0.092

Figura 2.3 Falsos positivos y Falsos negativos. Fuente [14]

Cada par de barras de la Figura 2.3, representa la tasa de FP y FN, para cada algoritmo. El objetivo es minimizar ambas tasas para mejorar la precisión y exactitud del IDS. Se menciona que lo ideal es que las tasas sean bajas en ambos casos. Indicando que puede identificar con precisión las instrucciones sin marcar actividad legitima como maliciosa y sin pasar por alto las actividades maliciosas.

Este articulo constituye un estudio experimental, no obstante, exhibe ciertas limitaciones al empleo exclusivo de un único data set. La realización de pruebas únicamente en el entorno de Weka podría restringir la generalización de los resultados,

considerando también que el número de tipos de ataques es limitado. Esta aproximación podría conllevar a que no se refleje adecuadamente la evolución reciente de las amenazas cibernéticas. Adicionalmente la interfaz de Weka para la visualización de resultados podría considerarse poco intuitiva, lo que dificulta su interpretación por usuarios no especializados que deseen acceder a los hallazgos del estudio.

2.4 Comparación y propuesta de valor

A continuación, en la Tabla 2.2, se presenta una comparación de los tres trabajos presentados en este capítulo.

Tabla 2.2 Comparación de los trabajos relacionados. Fuente: Elaboración propia

Categoría	Articulo 1	Articulo 2	Articulo 3	Propuesta
Titulo	Comparative analysis of Machine Learning algorithms for Instrusion Detection	Intrusion Detection Systems using Supervised Machine Learning Techniques: A survey	Evaluation of Machine Learning Algorithms for Intrusion Detection System	Detección de trafico de red malicioso con técnicas de inteligencia artificial
Enfoque	Comparar la efectividad de algoritmos de ML en la detección de tráfico de red malicioso	Revisión de técnicas de ML supervisado aplicadas a la detección de intrusiones	Evaluación experimental de algoritmos de ML para sistemas de detección de intrusiones	Comparar la efectividad de algoritmos de ML en la detección de tráfico de red malicioso y experimentar con un conjunto de datos propio
Metodología	Estudio experimental con pruebas de clasificadores de ML	Revisión de literatura	Estudio experimental con pruebas de clasificadores de ML	Estudio experimental con pruebas de clasificadores de ML
Conjunto de datos (Data set)	NSL-KDD	Comparación de KDD'99, NSL-KDD, CICIDS2017, UNSW-NB15	KDD	Conjunto Propio
Algoritmos	Random Forest Naive Bayes SVM J48 Logistic regression Decision Tree	Variedad de técnicas de ML supervisado	J48 Random Forest Decision Tree Naive Bayes MLP Bayes Network	J48 Random Forest Decision Tree Naive Bayes
Hallazgos clave	El Random Forest y J48 mostraron un alto rendimiento en ciertos tipos de ataques	Los algoritmos de ML supervisado son efectivos y prometedores en la detección de intrusiones	El clasificador de tabla de decisión tuvo el valor más bajo de falsos negativos	El porcentaje de efectividad de los clasificadores
Limitaciones	No se creó un conjunto de datos propio Potencial desactualización de los datos	Se remite a solo revisiones literarias, y no existe una experimentación practica	Uso de un solo conjunto de datos Pruebas limitadas a Weka Rango limitado de tipos de ataques	Limitaciones de Hardware

La propuesta de esta investigación es intentar cubrir estas limitaciones vistas en los anteriores trabajos de investigación, tales como la creación de conjunto de datos propio, en un laboratorio de pruebas especializado, y actualizado, el uso de Elasticsearch, Weka, y Tensor Flow para tener distintas perspectivas, y, por último, tipos de ataques más comunes.

CAPÍTULO 3 FUNDAMENTOS TEÓRICOS

A continuación, se abordan temas cruciales como los sistemas inteligentes, inteligencia artificial y las redes neuronales, así como su relación con la seguridad. Se explora la categorización de ataques de red malicioso, destacando las distintas formas en que estos pueden manifestarse.

3.1 Sistemas Inteligentes

Conforme a la descripción de M. Molina en [15], un sistema inteligente se define como “una máquina diseñada para ejecutar tareas de utilidad humana, como la conducción automática de vehículos o el diagnóstico médico”. La inteligencia de dicha máquina se evalúa en función de su comportamiento en un entorno determinado y su capacidad de interacción con otros agentes. Esta definición, sin embargo, quizá no sea universalmente aceptada, dado que, como se expuso en el capítulo 1 de este documento, el concepto de Inteligencia Artificial se fundamenta en una diversidad de propuestas teóricas.

A pesar de que no es estrictamente necesario que un sistema inteligente realice tareas como el diagnostico de enfermedades o la conducción automática para que sea considerado inteligente, es esencial que pueda interactuar con agentes inteligentes según lo dice Molina. En este contexto, Russell y Norvig en [2] , definen como un agente inteligente a una entidad capaz de percibir su ambiente mediante sensores y actuar sobre él a través de actuadores.

Se puede ver ilustrado gráficamente en la Figura 3.1 un sistema inteligente según lo propone Molina en [15], demostrando que un sistema inteligente percibe su entorno utilizando sensores y actúa mediante actuadores, teniendo en cuenta la capacidad de poder interactuar con otros agentes a través de dispositivos de entrada y salida (I/O).

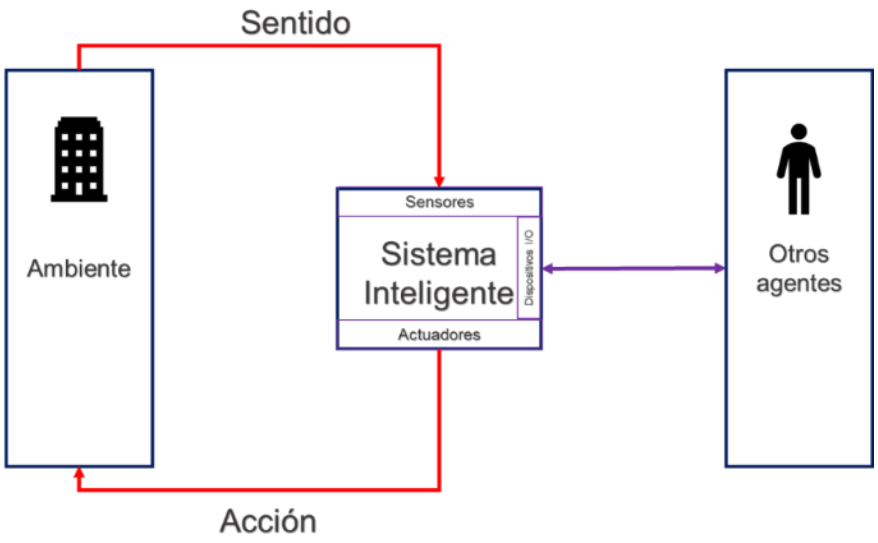


Figura 3.1 Un sistema Inteligente, según la definición de M. Molina. Fuente [15]

Se infiere que, de manera general, un sistema inteligente se comporta como un agente, lo que implica su capacidad para actuar basándose en información obtenida de su entorno mediante sensores y actuadores, que pueden se tanto físicos como virtuales.

Por ejemplo, un sensor virtual podría detectar un posible ataque en el tráfico de una red de datos, que constituye su entorno, y el sistema podría responder con medidas de mitigación apropiadas. La utilización de estos dispositivos marca la distinción del sistema inteligente del resto del entorno, un concepto conocido como “embodiment” o en español “personalización”, pero no es común este término en español. Además, se describe que un agente está “situado” en un entorno dado su interacción continua y directa con este, en un ciclo constante de percepción y acción.

3.2 Herramientas IDS

Para que un sistema inteligente perciba su entorno, como se mencionó anteriormente, es a través del uso de sensores, que pueden ser tanto físicos como virtuales. No es imperativo desarrollar sensores virtuales desde cero; en cambio, es una propuesta viable la adaptación de herramientas preexistentes para funcionar como tal.

Un ejemplo destacado de esta adaptación es el Sistema de Detección de Intrusos (SDI), conocido comúnmente por su denominación en inglés, como Intrusion Detection System (IDS). Estos sistemas pueden representar una aplicación práctica de sensores

virtuales, procesando información de la red para identificar actividades sospechosas o no autorizadas y de esta manera el sistema inteligente tome una acción.

3.2.1 ¿Qué es un Sistema Detector de Intrusos?

En [16] J. Beale y B. Caswell, definen una intrusión se define como el “acto de ingresar sin autorización a un lugar o estado”. Las intrusiones pueden ser perpetradas tanto por actores con intenciones maliciosas como por individuos descuidados sin la intención de dañar.

De acuerdo con la empresa IBM [17], un Sistema de Detección de Intrusiones es “una herramienta de seguridad de redes que monitorea el tráfico y los dispositivos de red para identificar actividades maliciosas, sospechosas o violaciones a las políticas de seguridad”. Es importante subrayar que un IDS no actúa como un sistema de respuesta ante ataques. En realidad, los IDS no funcionan de manera aislada; más bien, son parte del arsenal de herramientas a disposición de los administradores de red. Un sencillo ejemplo de su relación con otras herramientas es ilustrado en la Figura 3.2.

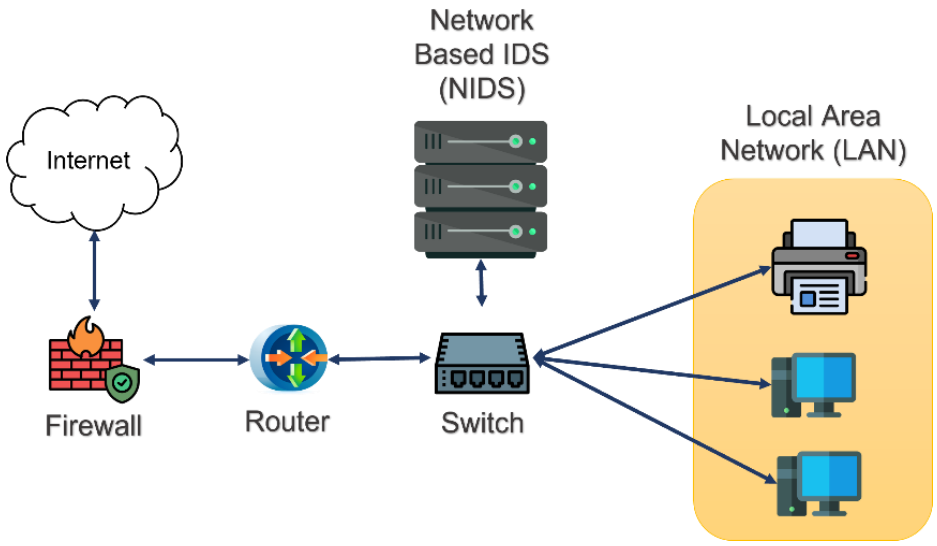


Figura 3.2 Red Local protegida mediante un Firewall y un IDS basado en red. Fuente: Elaboración propia

En [18], Z. Ahmad, S. Khan, W. Shiang, y otros autores proponen una clasificación para las herramientas IDS, en la Tabla 3.1, se explica de manera simplificada y resumida, esta clasificación.

Tabla 3.1 Clasificación de los IDS. Fuente [18]

<table>
<tr><th colspan="4">Clasificación de los IDS</th></tr>
<tr><th colspan="2">Método de despliegue</th><th colspan="2">Modo de detección</th></tr>
<tr><th>Host Based IDS (HIDS)</th><th>Network Based IDS (NIDS)</th><th>Signature Based IDS (SIDS)</th><th>Anomaly Detecion Based IDS (AIDS)</th></tr>
<tr>
<td>Sistemas basados en un HOST, se implementan en un equipo, y monitorizan la actividad interna del sistema.

Ventajas:
Monitoreo detallado.
Capacidad de verificar dispositivos conectados, aplicaciones, y servicios.
Detección precisa de actividades sospechosas en la computadora.
Desventajas:
Necesidad de implementación en cada host que requiera protección.
Sobrecarga potencial.
Disminución del rendimiento general del equipo.</td>
<td>Sistemas basados en red, detectan amenazas analizando el tráfico de la red. Estos sistemas requieren acceso promiscuo a la red y trabajan de manera pasiva.

Ventajas:
Monitoreo integral del tráfico de red.
Protección de los dispositivos en la red.
No impacta negativamente en el rendimiento de los hosts individuales.
Desventajas:
Puede no detectar intrusiones que no generen tráfico de red anómalo.
Menos efectivo en redes cifradas sin acceso a las claves de cifrado.
Requiere hardware y software capaz de manejar grandes volúmenes de datos de red.</td>
<td>IDS basado en firmas, funcionan mediante la identificación de patrones de ataques conocidos almacenados en una base de datos. Son eficientes para detectar ataques previamente identificados.

Ventajas:
Alta eficiencia en detectar ataques conocidos.
Capacidad para relacionar patrones específicos de ataques con firmas almacenadas.
Desventajas:
Ineficiencia para identificar ataques nuevos sin firmas previas
Dependencia de actualizaciones de la base de datos de firmas.
Posible vulnerabilidad ante tácticas de evasión avanzadas por parte de los ataques.</td>
<td>Los IDS basados en la detección de anomalías, identifican comportamientos inusuales, comparándolos con un patrón de actividad normal preestablecido. Una desviación significativa del comportamiento normal se considera una anomalía.

Ventajas:
Capacidad para detectar ataques nuevos.
Versatilidad para su uso en diversas aplicaciones y dispositivos.
Desventajas:
Dificultad en establecer límites claros entre comportamientos normales y anormales.
Posibles falso positivos debido a la variabilidad del comportamiento normal.</td>
</tr>
</table>

En la Figura 3.3, se puede ilustrar de manera grafica la clasificación propuesta, y como proponen los autores, como estas tecnologías puede relacionarse con las técnicas de IA.

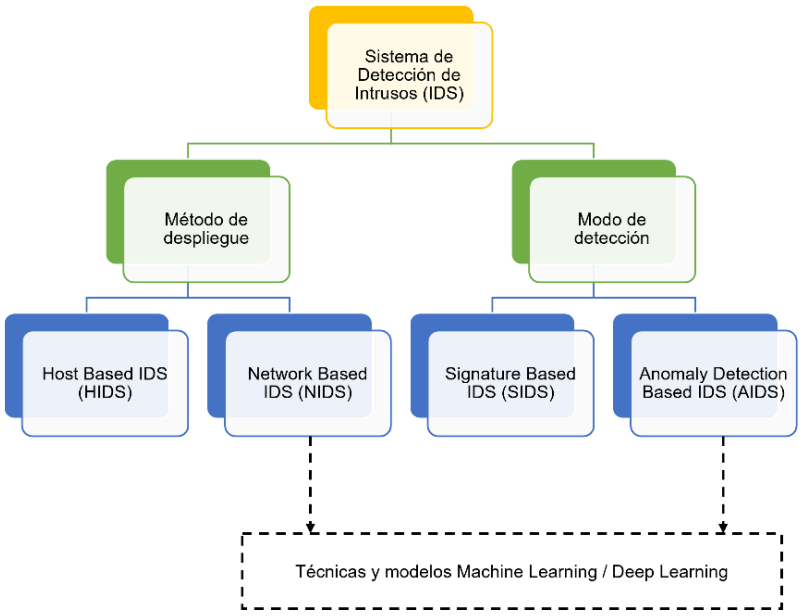


Figura 3.3 Clasificación de los IDS y su posible relación con la Inteligencia Artificial. Fuente [18]

3.2.2 Tecnologías IDS en la actualidad

Las tecnologías actuales presentan un abanico diverso de soluciones en Sistemas de Detección de Intrusos (IDS), cada una con sus propias características distintivas, incluyendo sus ventajas y limitaciones. Estas soluciones abarcan sistemas de código abierto, así como sistemas de tipo propietarios. En la Tabla 3.2, se detalla un resumen comparativo de estas tecnologías, proporcionando una descripción de cada una, facilitando así una elección de la solución más adecuada según las necesidades de esta investigación.

Tabla 3.2 Comparación de herramientas IDS. Fuente: Elaboración propia

HERRAMIENTA	LICENCIA	CAPACIDAD DE DETECCIÓN	FACILIDAD DE USO	SOPORTE Y COMUNIDAD	REQUISITOS
SNORT	Código abierto	• Basada en firmas • Anomalías • Protocolos	Media	Amplia comunidad Soporte limitado	Configuración Ajustes detallados
SURICATA	Código abierto	• Multihilos • Basada en firmas • Anomalías	Media - Alto	Comunidad activa Extensa documentación	Requiere hardware adecuado
BRO (ZEEK)	Código abierto	• Trafico en profundidad • Scripts personalizados	Alto	Amplia comunidad Documentación extensa	Configuración y ajuste avanzado
KISTMET	Código abierto	• Wireless • Drones	Medio	Poca comunidad Soporte limitado	Configuración y ajustes detallados

OSSIM	Código abierto	• Eventos • Anomalías	Medio	Amplia comunidad Soporte limitado	Configuración y ajustes detallados
MCAFEE NSP	Propietario	• Basada en firmas • Anomalías • Protocolos • Multihilos • Trafico en profundidad	Medio - Alto	Comunidad activa Soporte de pago	Pago de servicio Instalación por técnicos capacitados
AZURE SENTINEL	Propietario	• Basada en firmas • Anomalías • Protocolos • Multihilos • Trafico en profundidad	Alto	Comunidad activa Soporte de pago	Subscripción a Azure Pago por uso

En vez de focalizar el análisis en cada una de las herramientas descritas en la Tabla 3.2, este estudio se concentrará exclusivamente en la herramienta Snort. Esta ha sido seleccionada por su proceso de instalación relativamente sencillo, su compatibilidad con sistemas operativos basados en el Kernel de Linux, como Ubuntu, además de contar con una extensa y activa comunidad.

Snort en un sistema que consta de varios componentes, inicialmente, tiene un decodificador y preprocesador de paquetes que interpreta y transforma los datos de la red para su análisis conforme a las reglas preestablecidas. Luego un motor de detección compara los paquetes con reglas definidas para identificar posibles amenazas y finalmente, incorpora un sistema de registro para documentar y alertar sobre las detecciones realizadas.

Un aspecto destacado es su capacidad de integración con múltiples sistemas a través de plugins de salida, adaptándose a diferentes entornos, un ejemplo sería a una conexión MySQL, o generación de datos en formato JSON. Snort opera en tres modos: como sniffer para monitorizar en tiempo real, registrador de paquetes para almacenar datos para luego analizarlos, y por último como sistema IDS para aplicar reglas de detección a los datos recopilados y reconocer patrones específicos.

3.3 Inteligencia artificial y redes neuronales

Un IDS debe trabajar en conjunto con otras tecnologías que puedan tomar medidas capaces de responder ante amenazas, el conjunto de estas tecnologías se conoce como Sistema de Prevención de Intrusos mayormente conocido por sus siglas en ingles de

Intrusion Prevention System (IPS). Este sistema no solo detecta posibles ataques, sino también es capaz de mitigar. En este contexto para que un sistema sea inteligente debe tomar acción mediante un actuador por ejemplo como un IPS, con la integración de técnicas de inteligencia artificial en el proceso de análisis para la mejora y precisión en respuesta a las posibles amenazas detectadas.

En [2] , Russell y Norvig presentan una sección que explica las diversas disciplinas que han contribuido al desarrollo de la IA, se presentan en la Tabla 3.3, de una manera resumida. Todas estas han proporcionado perspectivas, ideas y técnicas clave que han ayudado a formar lo que hoy se conoce como Inteligencia Artificial.

Tabla 3.3 Disciplinas fundamentales de la Inteligencia Artificial. Fuente [2]

Disciplinas fundamentales de la Inteligencia Artificial por Russell y Norvig	
Filosofía	Explica como la filosofía presenta una consideración de la mente humana como una máquina que opera con conocimiento y toma de decisiones, los conceptos relacionados son: • Racionalismo • Dualismo • Materialismo • Empirismo • Inducción • Etcétera
Matemáticas	Herramientas para la lógica y su relación con la probabilidad, así como los fundamentos para la computación y los algoritmos, los conceptos relacionados son: • Algoritmos • Teorema de incompletitud • Computación • Tractability • NP-Completeness • Probabilidad
Economía	Formalización de la toma de decisiones para maximizar los resultados esperados, los conceptos relacionados son: • Utilidad • Teoría de decisión • Teoría de juegos • Investigación de operaciones
Neurociencias	Descubrimientos sobre el funcionamiento cerebral y su relación con las computadoras así, los conceptos relacionados son: • Neurociencia • Neurona • Singularidad
Psicología	Tratamiento de humanos y animales como máquinas de procesamiento de información, los conceptos relacionados son: • Conductismo • Psicología cognitiva
Ingeniería en Computación	Proveer los conocimientos de programación a los equipos de cómputo, que tienen gran capacidad de procesamiento para crear modelos de IA, los conceptos relacionados son: • Operaciones • Programación
Teoría de control y cibernética	El diseño de dispositivos que actúan óptimamente basados en retroalimentación del entorno, los conceptos relacionados son:

	• Teoría de control • Cibernética • Función objetivo
Lingüística	La integración del uso del lenguaje en el modelo de procesamiento de información, el concepto que se relacionan es: • Lingüística computacional

3.3.1 Una breve historia de la Inteligencia Artificial

En el año de 1936 el matemático Alan Turing propuso teorías fundamentales para el posterior estudio de la computación [19], Warren McCulloch y Walter Pitts en 1943, publicaron una teoría acerca de para trabajar con neuronas mediante circuitos eléctricos [2] en donde se realizó el análisis, de la teoría de la lógica proposicional propuesta por B. Russell y Whitehead, con la teoría de la computación propuesta por Turing.

Propusieron un modelo de neuronas artificiales que operan en modos “encendido” y “apagado”, donde sugirieron que una función computable podía ser realizada por una red de estas neuronas ilustrado en la Figura 3.4, también propusieron que esta red podría ser capaz de aprender, pero lograron demostrarlo.

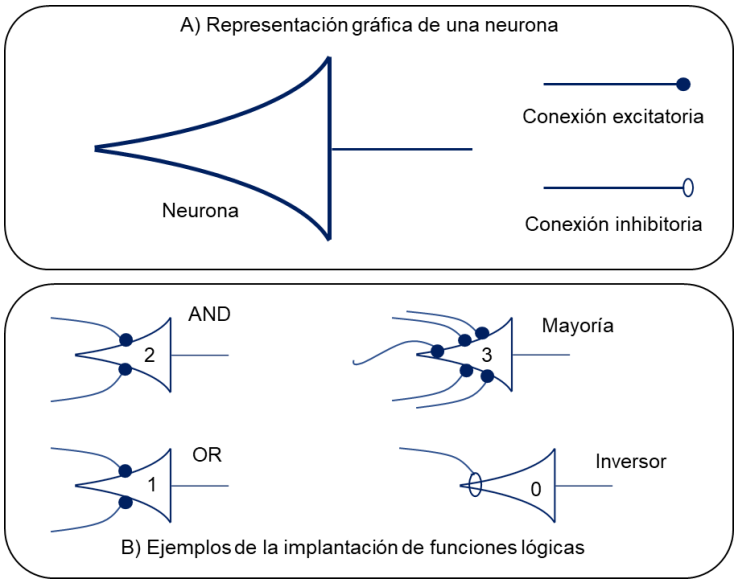


Figura 3.4 El modelo Neuronal de McCulloch y Pitts. Fuente [17]

En 1949, Donal Hebb propuso una teoría fundamental sobre los procesos de aprendizaje desde una perspectiva psicológica [17], proponiendo una regla fundamental para el aprendizaje neuronal. Este trabajo sentó las bases de lo que hoy conocemos como teoría de redes neuronales.

El nacimiento de la IA se sitúa en 1956, en un taller de dos meses en Dartmouth College, considerado el lugar del nacimiento oficial de la IA [2] . John McCarthy, una influyente figura de la universidad de Princeton en Estados Unidos, junto con, Minsky, Shannon y Rochester, fueron los organizadores de este taller. En el taller se propuso explorar la posibilidad de que todas las características del aprendizaje y la inteligencia pudieran describirse con tanta precisión que una máquina pudiera simularlas.

Posteriormente en 1957, Frank Rosenblatt desarrollo el concepto del perceptrón, la más antigua de las redes neuronales [21], un modelo capaz de aprender y reconocer patrones, aunque con limitaciones como su incapacidad de resolver el problema de la función OR-exclusiva y clasificar clases no linealmente separables. La estructura del modelo se ilustra en la Figura 3.5.

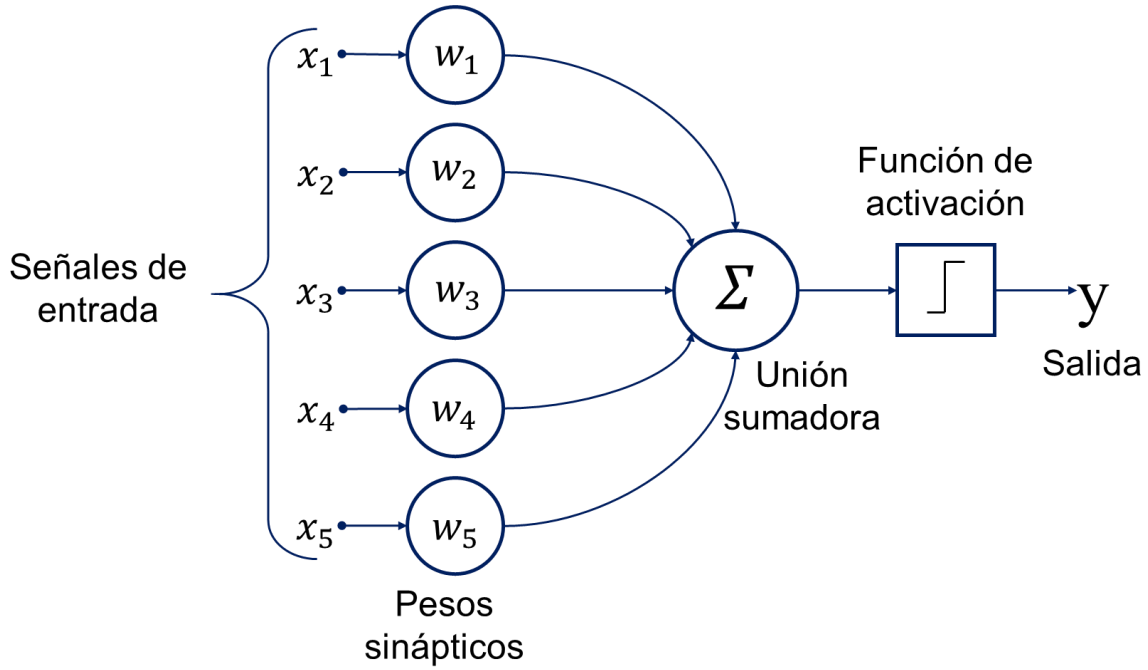


Figura 3.5 Modelo del Perceptrón propuesto por F. Rosenblatt. Fuente [21]

En 1974 Paul Werbos, desarrolla la idea de backpropagation [19], que proponía un algoritmo de aprendizaje hacia atrás, pero fue hasta 1985 que el significado quedó aclarado. Durante el año 1996 la super computadora Deep Blue, creada por la empresa IBM, vence al campeón del mundo de ajedrez [20].

En 2015 donde la empresa Google desarrollo “AlphaGo” donde en octubre de ese mismo año, logro derrotar a jugadores profesionales de Go [20]. Ese mismo año en los laboratorios de investigación de la empresa OpenIA, se comienza el desarrollo del modelo Generative Pre-trained Transformers (GPT) [22].

Se publica en 2018 el articulo “Improving Language Understanding by Generative Pre-Training” publicado por OpenIA, en donde se sientan las bases para sus modelos GPT [23], en donde se describe un método para mejorar la compresión del lenguaje en modelos de IA, basado en el preentrenamiento generativo donde un modelo se entrena primero en una tarea de generación de lenguaje y luego se ajusta para realizar tareas específicas de procesamiento del lenguaje natural.

Para noviembre de 2022 el modelo GPT-3 es accesible para todas las personas mediante la herramienta Chat GPT, el cual usa un modelo de lenguaje autorregresivo basado en aprendizaje profundo del lenguaje natural para realizar tareas relacionadas con la comprensión y generación de texto [22]. En marzo de 2023 GPT-4 sale a la luz como versión de paga de Chat GPT, un modelo capaz de procesar texto, pero también imágenes, y acceso a todas las herramientas de Inteligencia Artificial Generativa de OpenIA [24].[25]. Según Zhang y Lu [26], las aplicaciones de la IA se extienden a industrias como la automotriz, los mercados financieros, la salud, el entretenimiento, los sistemas de pago inteligente y los hogares inteligentes.

Tras este pequeño recorrido histórico de la Inteligencia Artificial, en el siguiente punto de esta sección, se revisa una perspectiva teórica de las redes neuronales.

3.3.2 Redes neuronales

Como se ha documentado previamente, el campo de la computación emergió a partir de las teorías propuestas por Turing [19]. Sin embargo, el desarrollo de las redes neuronales, fundamentales en los modelos de inteligencia artificial, se originó en un esfuerzo por emular las funciones del cerebro humano, considerado el procesador más avanzado y potente hasta la fecha.

La investigación de Paul Broca (1824-1880), sobre la afasia, un trastorno del habla, en pacientes con lesiones cerebrales en 1861, proporciono evidencia empírica de la

existencia de áreas específicas del cerebro dedicadas a distintas funciones cognitivas [2] . Este estudio reveló que la generación del lenguaje estaba confinada a una región del hemisferio izquierdo, posteriormente denominada área de Broca. Además, se estableció que el cerebro humano estaba compuesto por células nerviosas, que más adelante serían conocidas como neuronas.

En 1888, el distinguido científico español Santiago Ramón y Cajal, realizó un descubrimiento crucial, demostrando que el sistema nervioso constaba de una compleja red de células interconectadas, conocidas como neuronas [27]. A través de observaciones microscópicas, identificó pequeños espacios vacíos entre las neuronas, y postulo que el flujo de información dentro de una neurona se produce desde las dendritas hacia el axón, pasando por el soma [27].

Russell y Norvig proporcionan una exposición general sobre la estructura y funcionamiento básico de una neurona, como se ilustra en la Figura 3.6, extraída de [2] . Este análisis se puede sintetizar en varios puntos a continuación:

- **Estructura de la Neurona**: Las neuronas están compuestas por un cuerpo celular (soma) que contiene el núcleo, con múltiples dendritas y un axón que se extienden desde el. Las dendritas reciben señales, y el axón, que puede ser muy largo trasmite señales a otras neuronas.
- **Conexiones Sinápticas**: Las neuronas se conectan entre sí en puntos llamados sinapsis, formando redes complejas. Cada neurona puede conectar con hasta 100,000 otras neuronas, facilitando una extensa red de comunicación neuronal.
- **Transmisión de Señales y aprendizaje**: Las neuronas transmiten señales a través de procesos electroquímicos en las sinapsis. Estas señales son esenciales para las operaciones cerebrales inmediatas y también contribuyen a cambios a largo plazo en las conexiones neuronales, fundamentales para el aprendizaje y la memoria.
- **Procesamiento de Información en la Corteza Cerebral**: La mayor parte del procesamiento de información en el cerebro ocurre en la corteza cerebral, compuesta por columnas de tejido que contiene miles de neuronas. Estas

columnas son cruciales para las funciones cognitivas superiores y abarcan toda la profundidad de la corteza.

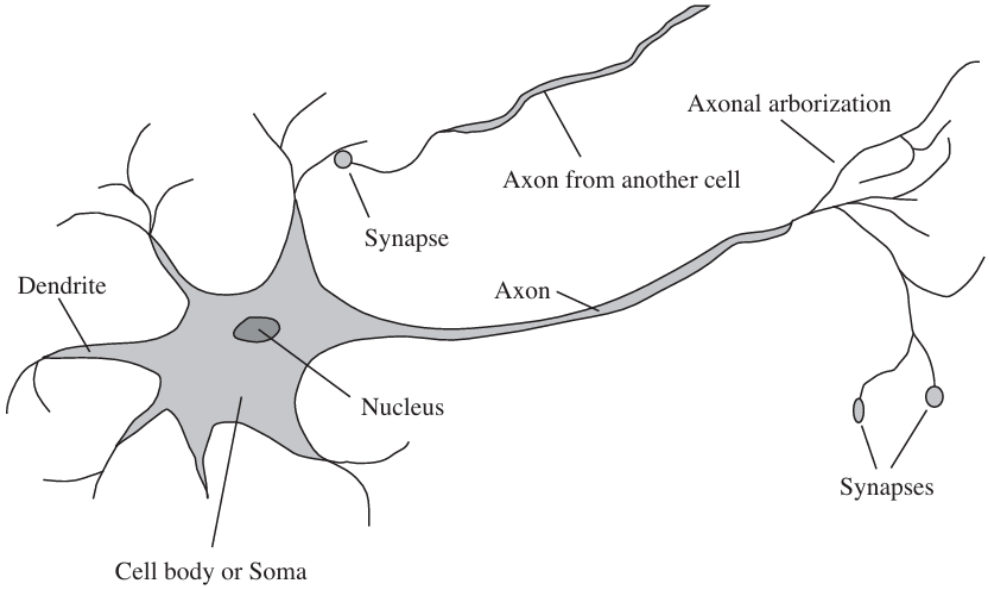


Figura 3.6 Estructura básica de una Neurona biológica. Fuente [2]

En el marco de la presente investigación, se centra en el análisis teórico de las redes neuronales artificiales. Estas estructuras matemáticas comúnmente referidas como Redes Neuronales Artificiales (ANNs, por sus siglas en inglés) [28], constituyen una parte esencial de este estudio.

En [29] Haykin, describe una neurona artificial como una “unidad de procesamiento de información”. A continuación, se describe el funcionamiento de una neurona artificial, en donde se puede ver una gran similitud, con una neurona biológica. Esta similitud no es fortuita, sino que resulta de un diseño intencionado para emular las funciones y comportamientos de su contraparte biológica. Esta se puede apreciar en la Figura 3.7.

- **Sinapsis y pesos sinápticos**: Cada neurona artificial posee múltiples sinapsis, que actúan como enlaces de conexión con otras neuronas. Cada sinapsis está caracterizada por un peso sináptico, el cual determina la fuerza o importancia de esa conexión especifica. Las señales x_j de entrada de la sinapsis j de la neurona k se multiplican por esto pesos w_{kj}. Los pesos sinápticos pueden tener valores

tanto positivos como negativos, lo cual difiere de las sinapsis en el cerebro biológico.

- **Sumador Lineal**: La neurona contiene un sumador que agrega todas las señales de entrada ponderadas. Esto significa que cada señal de entrada es multiplicada por su respectivo peso sináptico y luego todas estas señales ponderadas se suman. Esta operación se conoce como un combinador lineal.
- **Función de Activación**: Finalmente, la neurona utiliza una función de activación para limitar la salida a un rango específico. Esta función es importante para controlar la amplitud de la señal de salida es decir limita la señal a un rango finito, en algunos textos se puede encontrar como función de aplastamiento.

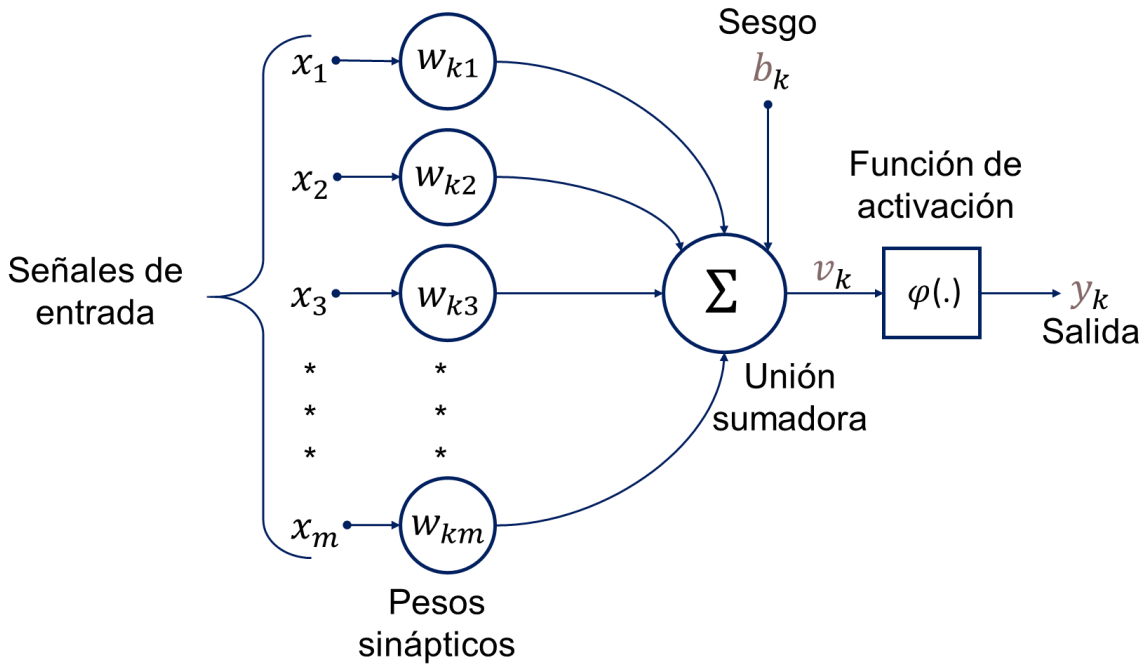


Figura 3.7 Modelo de una neurona artificial no lineal. Fuente [26]

La salida de una neurona artificial típicamente se normaliza a un intervalo cerrado de: [0, 1] o [-1, 1]. Esto significa que la función de activación de la neurona ajusta su salida para que caiga dentro del rango esperado.

La neurona incluye un sesgo (en inglés “bias”) denotado como b_k. Este sesgo ajusta el nivel de entrada a la función de activación hacia arriba o hacia abajo,

dependiendo de si es positivo o negativo. Esto permite que la neurona se active (o no) bajo diferentes condiciones de entrada.

En términos matemáticos se puede representar la neurona k representada en la Figura 3.7 mediante la ecuación (1) que representa la suma ponderada de las entradas. Donde x_j son las señales de entrada, y w_{kj} son los pesos sinápticos asociados a estas entradas para la neurona k, y $\mathbf{u_k}$ es la salida del combinador lineal de la neurona, que básicamente suma todas las entradas multiplicadas por sus respectivos pesos.

$$u_k = \sum_{j=1}^{m} w_{kj} x_j \tag{1}$$

La ecuación (2) muestra cómo se calcula la salida final de la neurona. Se aplica la función de activación φ a la suma de u_k (la suma ponderara) y el sesgo b_k, entonces y_k es la señal de salida final de la neurona.

$$y_k = \varphi(u_k + b_k) \tag{2}$$

En la ecuación (3) el sesgo realiza una transformación afín en la salida del combinador lineal. Esto indica que la entrada total a la función de activación es la suma ponderada más el sesgo. Esta transformación permite que la gráfica de la relación entre v_k y u_k, ilustrada en la Figura 3.8 no pase por el origen necesariamente, lo que da a la neurona mayor flexibilidad para ajustarse durante el proceso de aprendizaje.

$$v_k = u_k + b_k \tag{3}$$

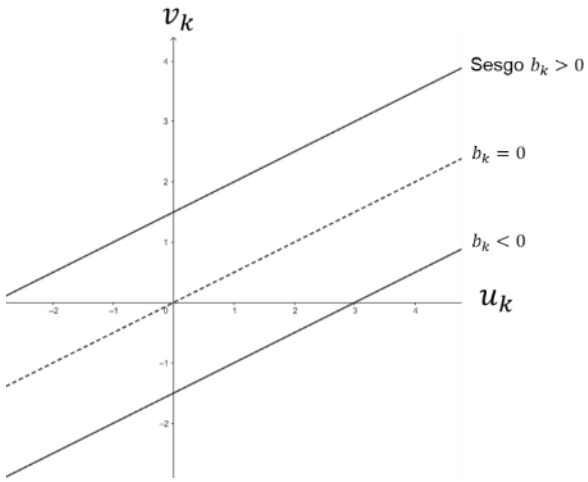


Figura 3.8 Transformación afín producida por la presencia de un sesgo. Fuente [26]

Lo visto anteriormente es un acercamiento inicial a la teoría de redes neuronales artificiales, fundamentado por Haykin en [29]. Esta perspectiva, aunque simplificada, ofrece una base enriquecedora y esencial para dar continuación a este análisis.

3.3.3 Funciones de activación

Las funcione de activación, que se denota por $\varphi(.)$, en el ejemplo de la Figura 3.7 definen la salida de una neurona en un límite de rango. A continuación, en la Figura 3.9 se presenta una posible propuesta de clasificación de las funciones de activación.

Figura 3.9 Una propuesta de la clasificación de las funciones de activación. Fuente: Elaboración propia

3.3.3.1 Función lineal

Una de las más sencillas funciones de activación en la función lineal. Un ejemplo que propone F. Berzal en [27], es (4) que se ilustra en la Figura 3.10.

$$y = f_{lin}(z) = z \tag{4}$$

La característica principal de neuronas artificiales con esta función de activación, es que, si se conectan varias capas de neuronas lineales en serie, el resultado siempre será un equivalente a una única capa de neuronas, de lo cual se abordará más adelante en la arquitectura de redes neuronales. Para ejemplificar lo anterior se propone en [30] una capa de neuronas con una matriz de pesos W_1, la salida será la entrada a una segunda capa de neuronas, con pesos W_2, que se representa en (5).

$$y = f_2\big(f_1(x)\big) = W_2(W_1 x) = (W_2 W_1)x \tag{5}$$

Es decir que se puede sustituir ambas capas por una única capa con una matriz de pesos $W = W_2 W_1$.

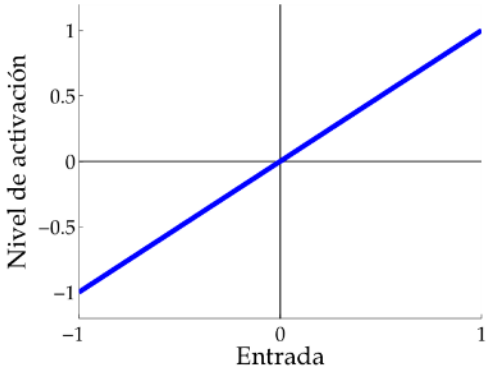


Figura 3.10 Función de activación lineal. Fuente [27]

3.3.3.2 Función escalón

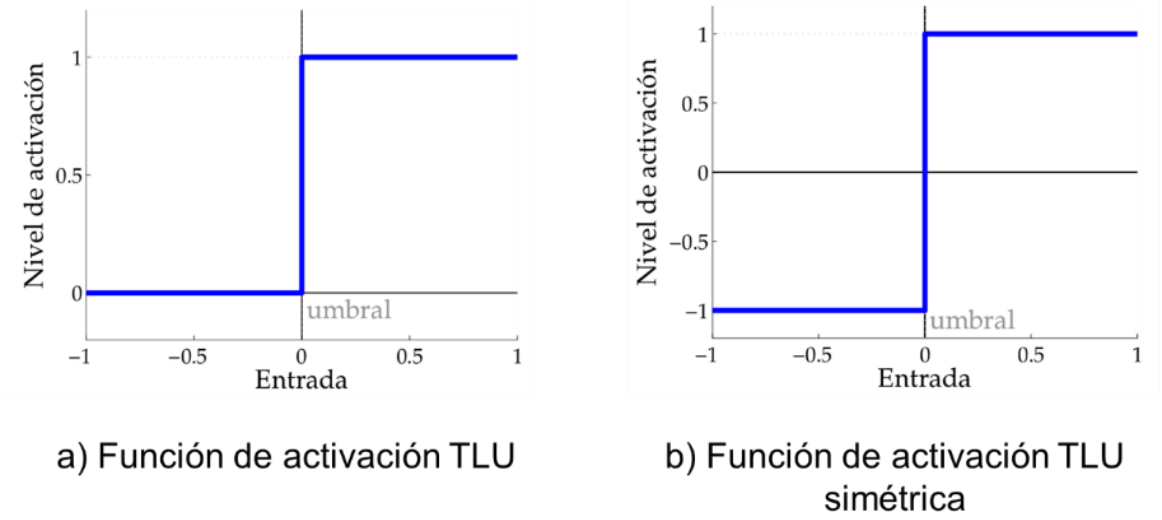
La función escalón es quizá la función de activación más antigua usada, es la que implementaron McCulloch y Pitts [2] así como en el perceptrón de F. Rosenblatt [21] visto en la Figura 3.5. Es una función no lineal según lo describe Berzal [30], también es conocida como función umbral, se puede ver representada formalmente en (6). En redes neuronales es la función basada en TLU (Threshold Logic Units).

$$y = f_{tlu}(z) = u(z) = 1_z \geq 0 = \begin{cases} 1 \; si \; z \geq 0 \\ 0 \; si \; z < 0 \end{cases} \quad (6)$$

Se puede observar que se trata de una función de activación binaria como se observa en la Figura 3.11 a), de la que se derivó una versión simétrica, la función de signo o SGN (signum funtion). También conocida como limitador estricto simétrico, que se puede ver en (7).

$$y = f_{sgn}(z) = sng(z) = \begin{cases} 1 & si \; z \geq 0 \\ -1 & si \; z < 0 \end{cases} \quad (7)$$

Esta función es simétrica con respecto al origen como se puede observar en la Figura 3.11 b), a pesar de ser una función más primitiva, y no eficiente para problemas complejos, es ideal para comenzar con el estudio de redes neuronales artificiales, con problemas sencillos que no requieran soluciones no binarias.



a) Función de activación TLU

b) Función de activación TLU simétrica

Figura 3.11 Funciones de activación escalón TLU. Fuente [27]

3.3.3.3 Función sigmoidal

Lo que interesa de una función de activación de una neurona artificial, es que cumpla con algunos requisitos, como que sea no lineal, estrictamente creciente, continua, y derivable, se puede ver un de manera general algunos de estos términos en la Figura 3.9. Para ello las funciones de activación sigmoidales, satisfacen todos estos requisitos.

Su utilidad es marcada con el algoritmo de Backpropagation [16], este algoritmo es un mecanismo central para el aprendizaje en las redes neuronales, permitiendo que la red aprenda de sus errores y ajuste sus pesos para mejorar sus predicciones.

Existe diversidad considerable de funciones sigmoidales, todas caracterizadas por su distintiva forma de “S”. Cuando existe una relación sencilla entre el valor de la función en un punto y el valor de su derivada en ese punto, así se puede aprovechar esa relación para reducir el coste computacional del entrenamiento de la red neuronal. Una vez calculado el valor de activación de la neurona, una sencilla expresión aritmética permite obtener su derivada de forma eficiente.

Una de las funciones sigmoidales más comunes es la función de activación logística, que se puede expresar en (8) y ver ilustrada en la Figura 3.12 a).

$$y = f_{logistic}(z) = \sigma(z) = \frac{1}{1 + e^{-z}} \tag{8}$$

La función logística es simétrica, en el sentido de que:

$$\sigma(-z) = 1 - \sigma(z) \tag{9}$$

La derivada (resumiendo el procedimiento) de la función logística es (10), por tanto, es un sigmoide “binaria”, con rango [0, 1]. Se puede ver ilustrada en la Figura 3.12 b).

$$\frac{d\sigma}{dz} = \frac{d}{dz}\left[\frac{1}{1 + e^{-z}}\right] = \sigma(z)(1 - \sigma(z)) \tag{10}$$

En la Figura 3.12 se pueden ver gráficamente la función logística.

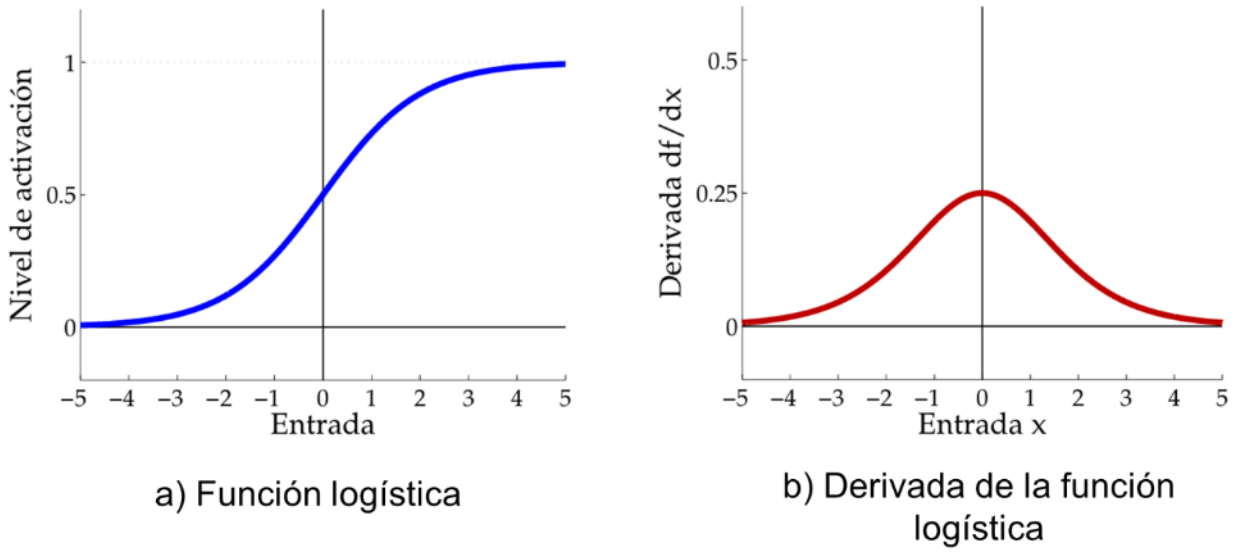


Figura 3.12 Funciones de activación sigmoidales. Fuente [27]

3.3.4 Arquitectura de las redes neuronales artificiales

La esencia de una red neuronal artificial, reside en la integración colectiva de múltiples neuronas para formar modelos de aprendizaje avanzados. A continuación, se realiza el análisis de las distintas arquitecturas que permiten comprender como estas estructuras cooperan para facilitar el procesamiento, análisis y aprendizaje de datos más complejos. En [30], Berzal, brinda tres tipos de arquitecturas de redes neuronales, las Redes feed-forward, competitivas y recurrentes.

3.3.4.1 Redes feed-forward

Son las arquitecturas más populares, quizá por su antigüedad, este tipo de arquitecturas, emplean redes con múltiples capas de neuronas. Su topología más habitual consiste en que las neuronas de cada capa suelen ser independientes entre sí, y operan en paralelo. Las distintas capas se conectan entre sí de tal forma que la salida de la capa i se utiliza como entrada en la capa $i+1$. Estas redes reciben su nombre al carecer de algún mecanismo de retroalimentación, existen varios tipos de redes feed-forward.

- **Redes simples, con una capa**

Esta conjuración representa el caso más básico de una red neuronal y tiene aplicaciones prácticas significativas. Aunque para problemas más complejos podría no

ser la solución óptima, resulta ideal para introducir los principios fundamentales de las redes neuronales. En esta arquitectura, las neuronas se sitúan únicamente en la capa de entrada, cuya función se limita a recibir señales externas. Estas señales de entrada son luego redistribuidas a las neuronas de la capa de salida, la cual constituye la única capa en la red que realiza operaciones de procesamiento. Esta estructura se puede visualizar en la Figura 3.13.

.

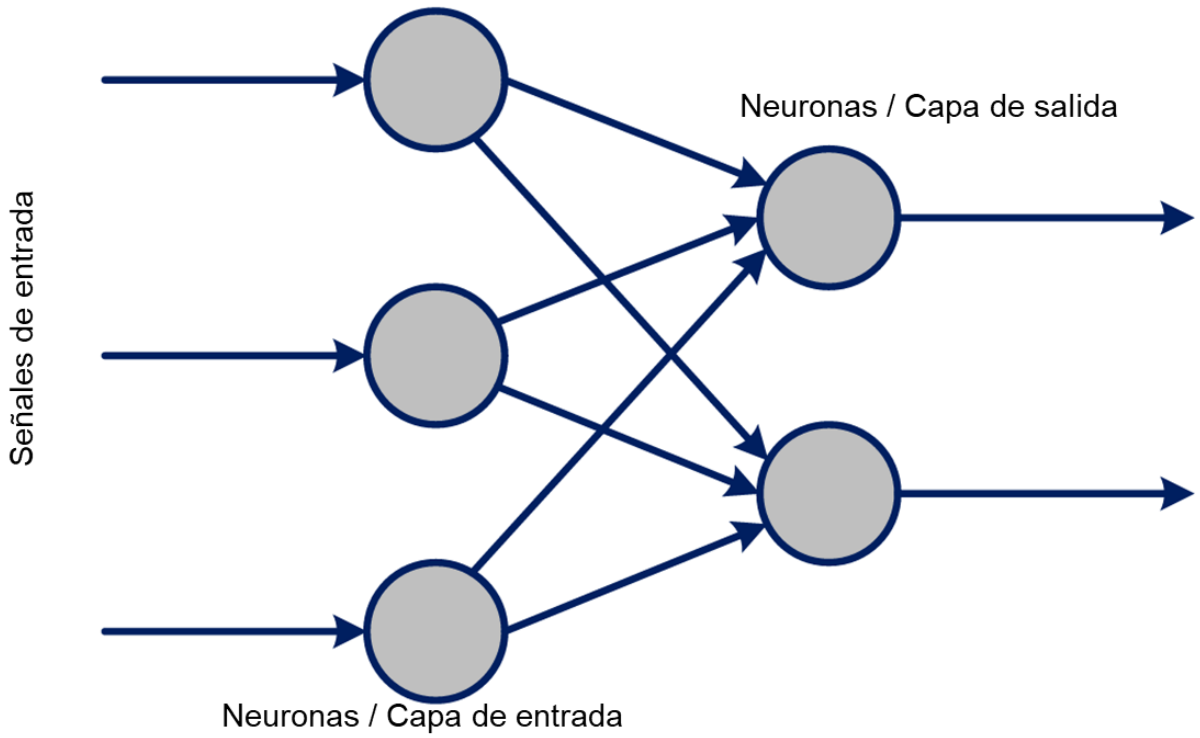


Figura 3.13 Arquitectura de neurona simple, sin capas ocultas. Fuente [30]

- **Redes multicapa, con una capa**

Es una red que contiene capas internas, es decir además de la capa de entrada y salida (que son visibles), contiene capas adicionales internas, estas no serán visibles, y las llamaremos capas ocultas. Estas capas tienen muchas aplicaciones, como por ejemplo el uso del algoritmo Backpropagation para ajustar parámetros (proceso de aprendizaje). Se puede observar en la Figura 3.14.

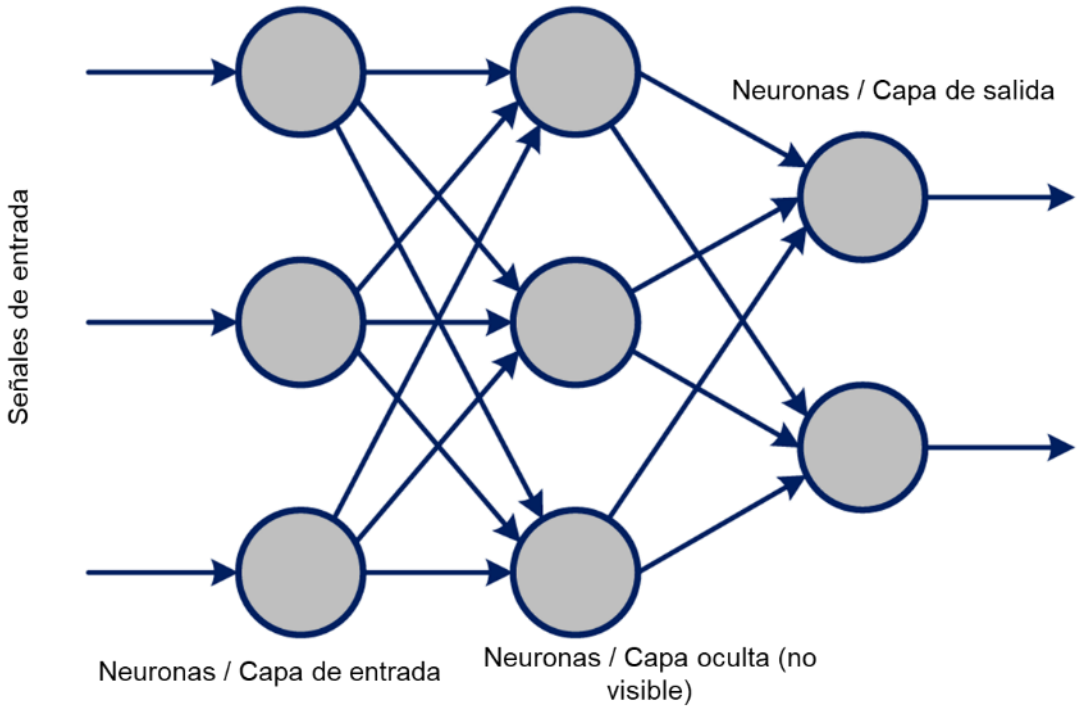


Figura 3.14 Red neuronal con una capa oculta. Fuente [30]

- **Redes profundas, con varias capas ocultas**

Este tipo de redes tiene una más de una profunda, que al igual que la anterior tampoco son es visible. Su importancia radica en el campo del aprendizaje profundo "Deep learning", se puede ver ilustrada en la Figura 3.15.

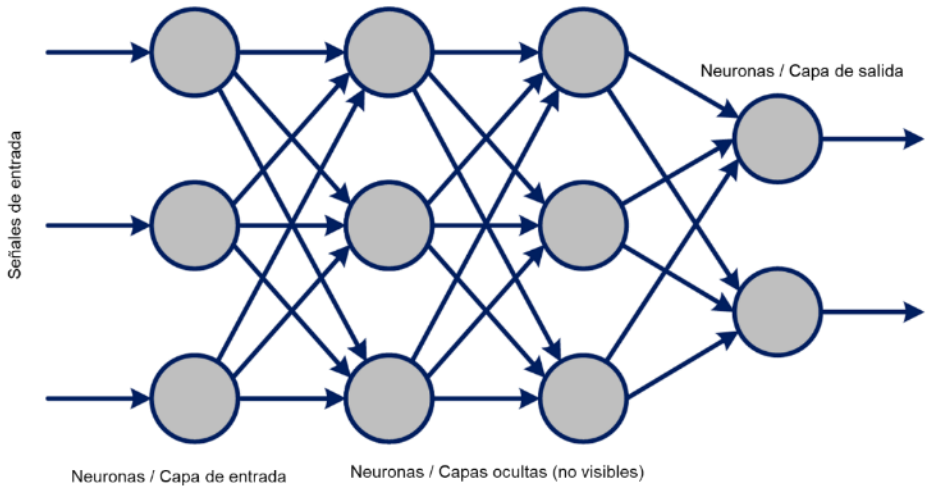


Figura 3.15 Red neuronal multicapa (2 capas profundas para este ejemplo). Fuente [30]

3.3.4.2 Redes competitivas

En este tipo de red, las neuronas compiten entre sí para determinar cuál se activará. Esta competencia se basa generalmente en la fuerza de la señal de entrada y los pesos sinápticos de las neuronas, la neurona con la respuesta más fuerte inhibe la activación de las otras neuronas en la capa. A menudo este tipo de redes se emplean para el aprendizaje no supervisado, debido a que aprenden a identificar patrones o características comunes en los datos de entrada. Un ejemplo grafico se ilustra en la Figura 3.16.

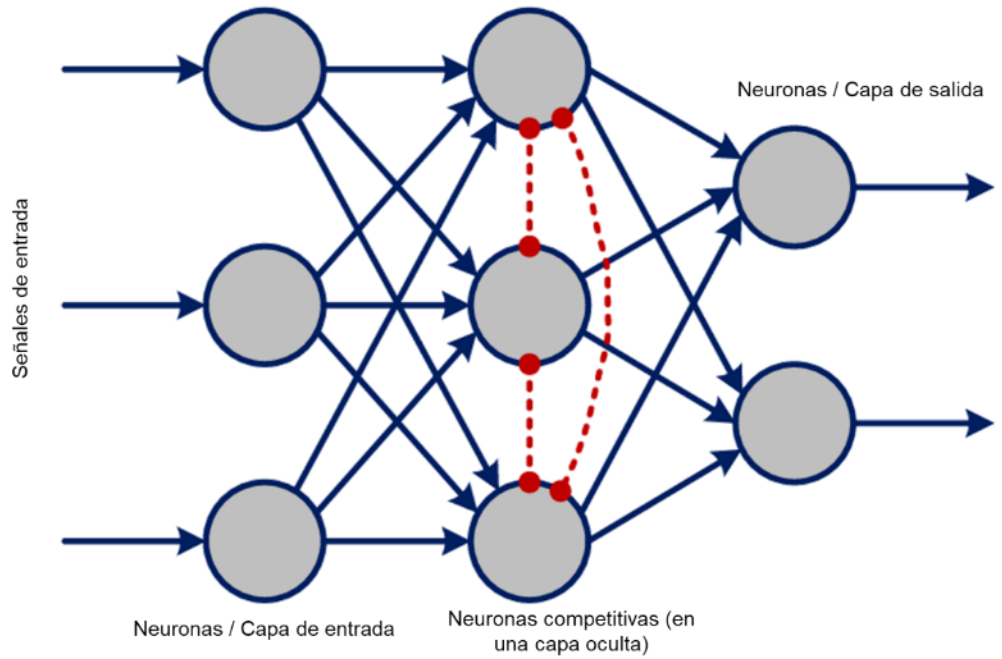


Figura 3.16 Red Neuronal competitiva. Fuente [30]

Un ejemplo famoso de redes competitivas son los mapas autoorganizativos (Self-Organizing Maps, SOMs) de Kohonen, que son útiles para la visualización y análisis de datos de alta dimisión [27].

3.3.4.3 Redes recurrentes

Las redes recurrentes son un tipo de red neuronal diseñada para procesar secuencias de datos, aprovechando su naturaleza temporal o secuencial. A diferencia de las redes feed-forward, donde la información fluye en una sola dirección, las RNN (redes

neuronales recurrentes) tienen conexiones que forman ciclos, permitiendo que la información persista. Se ilustra en la Figura 3.17.

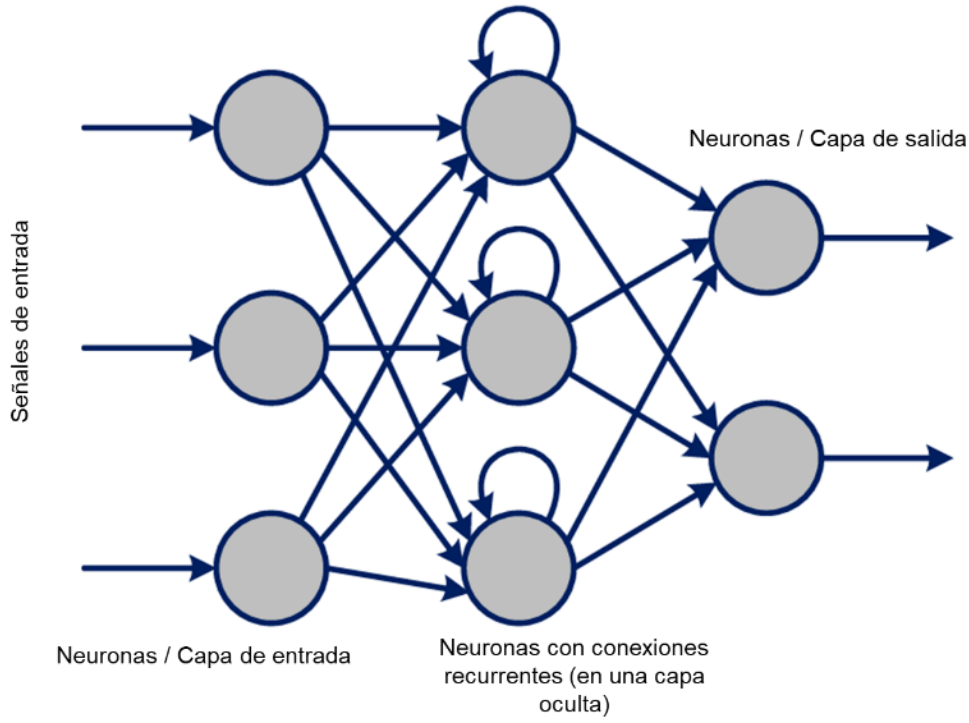


Figura 3.17 Red neuronal recurrente. Fuente [30]

En una red recurrente, las salidas de ciertas neuronas se retroalimentan a la misma capa o a capas anteriores. Esto permite que la red mantenga un estado o memoria de las entradas anteriores, lo cual es crucial para tareas en donde el contexto y el orden de los datos son importantes.

Este tipo de redes tiene una amplia gama de aplicaciones, incluyendo el reconocimiento de voz, la generación de texto, la traducción automática y la predicción de series temporales.

3.4 Algoritmos y técnicas de Machine Learning

Una aplicación práctica relevante en el campo de la inteligencia artificial y las redes neuronales, discutidas en el tema 3.3, es el aprendizaje automático o machine learning. En este apartado, se lleva a cabo un análisis más detallado de las diversas técnicas y algoritmos asociados con el machine learning.

La primera cuestión a abordar es la naturaleza del aprendizaje en sí. Recordando la analogía con las redes neuronales biológicas, ilustrada en la Figura 3.6, en donde el aprendizaje ocurre cuando las neuronas establecen conexiones a través de las sinapsis, podemos trasladar este concepto al ámbito de las redes neuronales artificiales. En este contexto, existen varios tipos de aprendizaje [31], que se pueden observar en la Tabla 3.4. En esta clasificación no se ha incluido el aprendizaje automático o machine learning, y el aprendizaje profundo o Deep, ya que técnicamente estas dos, hacen uso de estas clasificaciones.

Tabla 3.4 Clasificación de los tipos de aprendizaje. Fuente [31]

Clasificación de los tipos de aprendizajes		
Supervisado	**No supervisado**	**Por refuerzo**
La red se entrena con un conjunto de datos etiquetados. Cada entrada en el conjunto de entrenamiento tiene una salida o etiqueta asociada.	La red se entrena con datos no etiquetados. El objetivo es encontrar patrones o estructuras intrínsecas en los datos de entrada.	La red aprende a tomar decisiones mediante la realización de acciones en un entorno para maximizar alguna noción de recompensa o beneficio acumulado.
La red aprende a mapear entradas a salidas y se ajusta hasta que puede hacer predicciones precisas sobre datos no vistos	Se utiliza para agrupamiento, reducción de dimensionalidad y aprendizaje de características.	Se utiliza en videojuegos, robótica, y en problemas de toma de decisiones secuenciales.
Ejemplo: Clasificación, regresión.	Ejemplos: agrupación (clustering), análisis de componentes principales.	Ejemplo: Q-Learning, políticas de gradiente.

En el apartado “1.1 Introducción al trabajo de investigación”, de este documento, se analizó una breve explicación del concepto de machine learning, una ampliación descrita de [5] sobre machine learning es que el “machine learning puede definirse de manera amplia como métodos computacionales que utilizan la experiencia para mejorar el rendimiento y predicciones más precisas”, en donde la experiencia se refiere a la información disponible para el algoritmo computacional.

3.4.1 Lugar de Machine Learning en la investigación

Aquí surge una cuestión del ¿porque utilizar técnicas de machine learning para la detección de trafico de red malicioso?, si en muchas herramientas es posible, como por ejemplo con un firewall, un antivirus, un antimalware, y sistemas IDS e IPS en la DMZ (zona desmilitarizada). ¿Cuál es el beneficio de utilizar estas técnicas y si es viable?, estas cuestiones se abordan a continuación.

La importancia de realizar una investigación de carácter científico que utilice técnicas de ML, presenta 6 puntos que pretenden responder a estas cuestiones, justificando el uso de estas técnicas, que serán analizadas más adelante.

- **Detección de amenazas nuevas y evolutivas**: Los métodos tradicionales como IDS, IPS, antivirus y firewalls a menudo se basan en firmas conocidas para detectar amenazas. Sin embargo, estas herramientas pueden ser menos efectivas contra amenazas nuevas o en rápido cambio, como el malware de día cero (zero day atacks). El machine learning puede analizar patrones de tráfico y comportamientos en la red anómalos para identificar patrones que podrían indicar una nueva forma de ataque. En el reporte de 2022 de FortiGuard Labs informa que en el año 2022 el Ransomware Emotet desarrollo al menos 92 variantes nuevas [32]. Según este informe el volumen mensual de todos los tipos de Ransomware detectados solo por los equipos de FortiGuard a un promedio de 391.1 M de intentos de Ransomware detectado, en sus múltiples variantes. Véase Figura 3.18, fuente [32].

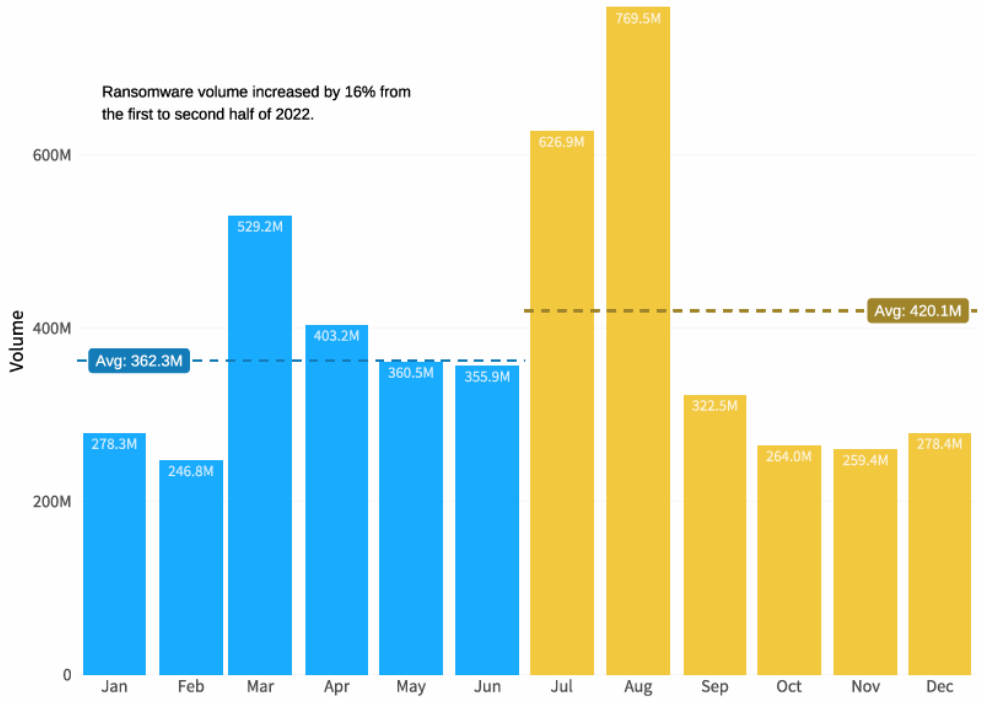


Figura 3.18 Volumen mensual de ransomware para 2022. Fuente [32]

- **Adaptabilidad y aprendizaje continuo**: Las técnicas de machine learning pueden adaptarse y aprender continuamente de los nuevos datos, lo que les permite mantenerse al día con las tácticas cambiantes de los ciberatacantes. Esto contrasta con algunas herramientas tradicionales que pueden requerir actualizaciones manuales y regulares.
- **Análisis de comportamiento y contexto**: El machine learning puede analizar el tráfico de red en un contexto más amplio, observando patrones de comportamiento a lo largo del tiempo y en diferentes partes de la red. Esto permite una detección más precisa y menos falsos positivos en comparación con los métodos basados únicamente en firmas.
- **Complementariedad con herramientas existentes**: Las técnicas de machine learning no necesariamente reemplazan a las herramientas tradicionales como IDS, IPS, antivirus y firewalls entre otras. Sino que pueden complementarlos, proporcionando una capa adicional de seguridad en la red.
- **Personalización y flexibilidad**: El machine learning permite desarrollar soluciones personalizadas que se ajusten a las necesidades específicas y a la arquitectura de una red particular, ofreciendo así una protección más específica y adaptada.

3.4.2 Principales retos del machine learning

En su obra [33], A. Gerón, describe los principales retos del machine learning, para este estudio solo se hace mención de los 6 más significativos en los siguientes puntos.

- **Cantidad insuficiente de los datos de entrenamiento**: Gerón explica que por ejemplo para que un niño pequeño aprenda que es una manzana, basta con decirle y enseñar si mucho un par de veces que es una manzana, y sabrá por el resto de su vida que forma, como se ve, y que sabor tiene una manzana, en cambio para una máquina, eso es imposible, esta solo puede aprender algunas de las características de una manzana, esta necesita muchos ejemplos de que es una manzana, en cuanto más ejemplos tenga, será mucho mejor.

  Se presenta el ejemplo presentando por investigadores de la empresa Microsoft, en un estudio que presenta la cantidad de datos necesarios para un problema complejo

de procesamiento del lenguaje natural, con diferentes técnicas de machine learning, que se ilustra en la Figura 3.19.

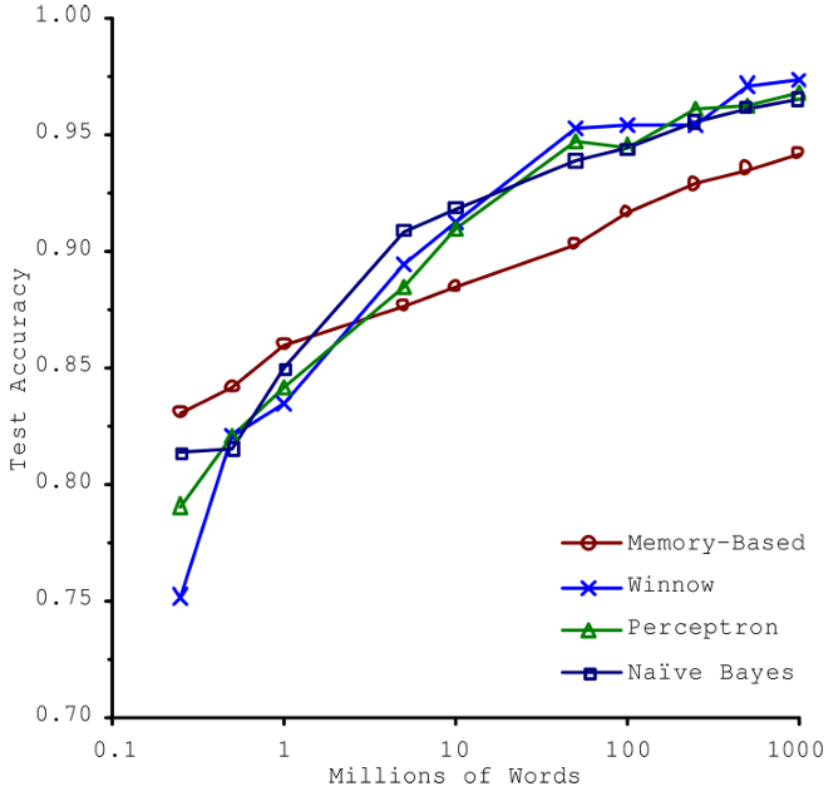


Figura 3.19 La importancia de los datos en contraste con los algoritmos. Fuente [33]

- **Datos no representativos en el entrenamiento**: Es crucial que los datos de entrenamiento sean lo suficientemente representativos. Al utilizar un conjunto de datos no representativo, se puede entrenar un modelo que probablemente no hará las predicciones más precisas. Un ejemplo dado se refiere a un modelo lineal entrenado para predecir la felicidad en diferentes países, ilustrado en la Figura 3.20. Esto ilustra cómo un conjunto de entrenamiento no representativo ilustrado por la línea punteada puede llevar a conclusiones erróneas, particularmente en casos extremos (como países muy obres o muy ricos en este ejemplo) al contrario de un modelo con más datos representativo (línea recta).

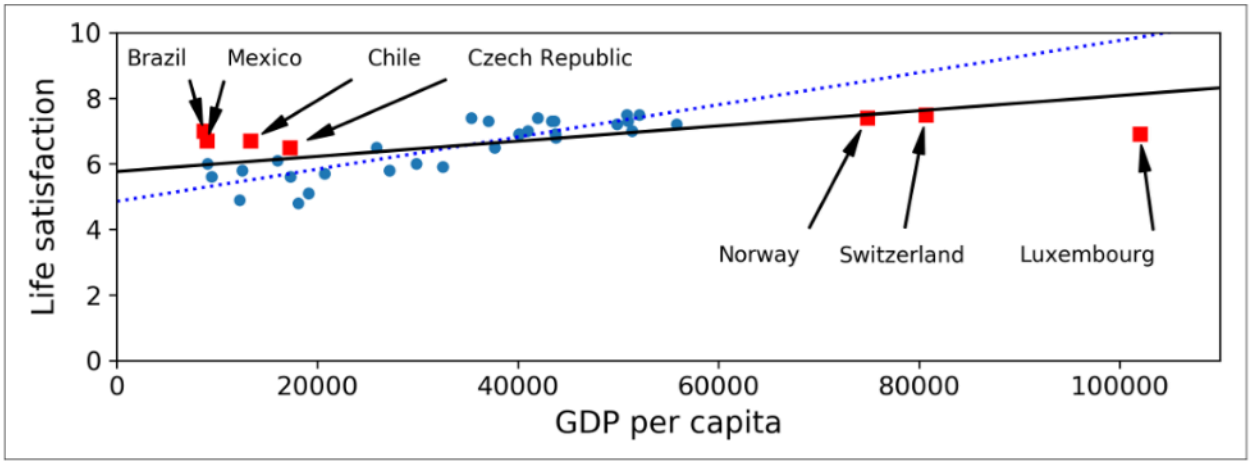


Figura 3.20 Ejemplo de la importancia de datos representativos. Fuente [33]

- **Mala calidad en los datos**: Si los datos contienen muchos errores, valores atípicos (outliers) o ruido, será más complicado para el algoritmo de aprendizaje automático identificar los patrones correctos, lo que afectará negativamente el rendimiento del modelo. Se destacan dos puntos importantes, el primero es el manejo de valores atípicos, es decir se debe identificar y tratar los valores atípicos, ya que pueden distorsionar los resultados del modelo, y el segundo es el tratamiento de datos faltantes, como descartar esos datos o utilizar la media para continuar con el entrenamiento.
- **Características irrelevantes**: Si los datos de entrada son irrelevantes, el modelo también será irrelevante, así como sus resultados, los puntos que se deben tomar en cuenta son una buena selección de características, esto implica elegir las características más significativas y relevantes, es decir hallar aquellas que contribuyen de manera más efectiva al proceso de aprendizaje y predicción. Luego es la extracción de características, esto es aplicar técnicas de reducción de dimensionalidad que ayudan a simplificar los datos sin perder información crucial. Por último, es la posible creación de nuevas características, en ocasiones las características que requiere el modelo no se encuentran presentes en los datos obtenido, y se debe generar derivándola de los datos existentes.
- **Sobreajuste de los datos de entrenamiento (Overfitting)**: Situación en la que un modelo se ajusta demasiado a los datos específicos con los que se entrena, hasta el punto de perder capacidad de generalización a datos nuevos o no vistos. Puede ser

causado por varios factores como tener un modelo demasiado complejo para la cantidad de datos de entrenamiento disponibles, no tener suficiente variedad en los datos de entrenamiento, o entrenar el modelo por demasiado tiempo. Se puede prevenir al utilizar las cantidades adecuadas de datos, utilizar técnicas de regularización y al emplear métodos de validación cruzada para evaluar la capacidad del modelo para generalizar datos no vistos.

- **No ajuste en los datos de entrenamiento (Underfitting)**: El modelo no tiene suficiente complejidad o flexibilidad para aprender de los datos. Puede deberse a un modelo que no tiene suficientes parámetros o características para entender las relaciones en los datos, o que falta más tiempo de entrenamiento. Puede ser causado por un modelo excesivamente simple, poco tiempo de entrenamiento, y datos de baja calidad. Se puede prevenir, adoptando un modelo más complejo, más ciclos de entrenamiento, y revisando la calidad de los datos.

3.4.3 Técnicas de aprendizaje

En [30], se presentan una perspectiva teórica detallando cómo al igual que en otras disciplinas científicas, el campo del aprendizaje automático ha visto el desarrollo de diversas corrientes de pensamiento, cada una proporcionando su propia perspectiva y enfoque metodológico. Este análisis se basa en una clasificación propuesta por Pedro Domingos [34], profesor de la Universidad de Washington en Seattle, que describe cinco corrientes de pensamiento. Véase la Tabla 3.5.

Tabla 3.5 Clasificación sobre las corrientes de pensamiento en ML. Fuente [31]

Clasificación de P. Domingos, sobre las corrientes de pensamiento en el ML	
Enfoque	Descripción
Simbólicos	Enfatizan aspectos filosóficos, lógicos y psicológicos del aprendizaje interpretándolo como inducción, el inverso de deducción.
Analógicos	Fundamentados en la extrapolación de ejemplos conocidos, usando juicios de similitud, a menudo conducen a problemas de optimización matemática.
Bayesianos	Concentrándose en la inferencia estadística y probabilística, basándose en el teorema de Bayes, como en las redes bayesianas.
Evolutivos	Basan sus métodos en la teoría de la evolución y la genética para desarrollar técnicas como los algoritmos genéticos.
Conexionistas	Inspirados en el funcionamiento del cerebro humano, colaborando con físicos y neurocientíficos para realizar ingeniería inversa de sus procesos.

3.4.4 Algoritmos de machine Learning

En el artículo “Comparative analysis of Machine Learning algorithms for Intrusion Detection” publicado en el año 2021 en la “IOP Conference Series: Materials Science and Engineering” [12], revisado en el “CAPÍTULO 2 TRABAJOS RELACIONADOS” se presentan los algoritmos de machine learning más populares para el análisis de detección de intrusiones. De los cuales se realiza un análisis a continuación.

3.4.4.1 Algoritmo Random Forest

El algoritmo Random Forest es un método de aprendizaje en conjunto (ensemble learning) utilizado tanto en clasificación como en regresión. Es uno de los algoritmos más populares y potentes, conocido por su simplicidad y eficacia.

Su funcionamiento se basa em crear un “bosque” que es un conjunto de árboles de decisión, generalmente entrenados con un método de “bagging” o “boostrap aggregating”. El termino bagging hace referencia a que, en cada árbol en el bosque, se construye a partir de una muestra aleatoria con reemplazo de los datos de entrenamiento.

Para realizar predicciones, agrega las predicciones de todos los árboles del bosque. En el caso de la clasificación, esto se hace por “votación mayoritaria”, en el caso de la regresión, se toma el promedio de las predicciones.

En términos matemáticos, se puede explicar suponiendo que se tiene un conjunto de entrenamiento X con N instancias y M características. Random Forest construye un conjunto de K árboles de decisión, $T1, T2, ..., T_k$.

Para cada árbol T_k.

- Se selecciona una muestra aleatoria de los datos de entrenamiento, dígase X_k.
- Durante la construcción del árbol, para cada división, se selecciona un subconjunto aleatorio de las características.
- Cada árbol T_k, genera una predicción y_k para una entrada dada.

La predicción final del Random Forest para una entrada dada es:

- En clasificación: La clase más votada entre todas las predicciones y_k, de los árboles.
- En regresión (11): El promedio de todas las predicciones y_k, de los árboles:

$$y_{RF} = \frac{1}{k}\sum_{k=1}^{K} y_k \quad (11)$$

El algoritmo Random Forest se podría clasificar dentro de los enfoques conexionistas en el aprendizaje automático. Aunque su funcionamiento se basa en árboles de decisión, que son generalmente considerados como métodos simbólicos debido a su naturaleza interpretable y basada en reglas.

3.4.4.2 Algoritmo Naive Bayes

El algoritmo de Naive Bayes es un método de clasificación basado en el teorema de Bayes, con una "ingenuidad" (naive) subyacente que asume que las características que se utilizan para la predicción son independientes entre sí. A pesar de esta simplificación, Naive Bayes puede ser sorprendentemente efectivo, en particular para los grandes retos del Big Data.

Su funcionamiento se basa en utilizar el teorema de Bayes, para calcular la probabilidad de que un dato pertenezca a una clase determinada dada sus características. Además, asume que todas las características son independientes entre sí, lo que a menudo resulta no ser cierto, el algoritmo sigue funcionado bien en muchas aplicaciones prácticas.

Su interpretación matemática, se formula de la siguiente manera en (12).

$$P(C_k|X) = \frac{P(X|C_k)P(C_k)}{P(X)} \quad (12)$$

Donde:

- $P(C_k|X)$ es la probabilidad posterior de la clase C_k dado el predictor X.
- $P(X|C_k)$ es la probabilidad de que el predictor pertenezca a la clase C_k.
- $P(C_k)$ es la probabilidad previa de la clase C_k.
- $P(X)$ es la probabilidad total de que el predictor sea observado

En la práctica, cuando el interés es en la clase más probable, a menudo solo se necesita calcular $P(X|C_k)P(C_k)$ para cada clase y seleccionar la clase con mayor probabilidad.

Naive Bayes se clasificaría en la categoría de enfoques bayesianos en el aprendizaje automático de la Tabla 3.5. En base a que ese enfoque se basa en el uso de técnicas estadísticas de inferencia probabilística y, como su nombre lo indica se basa en el teorema de Bayes.

3.4.4.3 Algoritmo J48

El algoritmo J48, es un método de aprendizaje supervisado utilizado para la clasificación y la generación de árboles de decisión. Los árboles de decisión son estructuras de árbol donde cada nodo interno representa una "prueba" en un atributo, cada rama representa el resultado de la prueba y cada nodo hoja representa una clase.

El funcionamiento del algoritmo J48 implica la construcción de un árbol de decisión de manera recursiva, dividiendo el conjunto de datos en subconjuntos más pequeños basados en los atributos que mejor separan las clases. La medida utilizada para evaluar la calidad de la división puede ser la ganancia de información o la reducción de la impureza, entre otras.

El proceso de construcción del árbol se detiene cuando se cumple alguna condición de parada, como cuando todos los ejemplos en un nodo pertenecen a la misma clase, o cuando no hay más atributos disponibles para dividir, o cuando se alcanza un límite predefinido en la profundidad del árbol.

Una vez construido el árbol de decisión, se utiliza para clasificar nuevos ejemplos. Cada ejemplo sigue el camino a través del árbol, comenzando desde la raíz y siguiendo las ramas correspondientes a los valores de atributo del ejemplo, hasta llegar a un nodo hoja que determina la clase a la que pertenece el ejemplo.

Desde una perspectiva matemática, el algoritmo J48 busca construir un árbol que maximice la homogeneidad de las clases en los nodos hoja, mientras minimiza la profundidad del árbol y el número de divisiones necesarias.

En resumen, el algoritmo J48 es una herramienta poderosa y versátil para la clasificación de datos, que produce modelos interpretables en forma de árboles de decisión, adecuados para una amplia gama de problemas de aprendizaje supervisado.

3.4.4.4 Algoritmo de regresión logística

El algoritmo de regresión logística es un modelo estadístico utilizado para la clasificación binaria. Se pudo observar un ejemplo de este tipo en la sección, "3.3.3.3 Función sigmoidal" de este documento, en donde se analizó una función de activación sigmoidal, presentando su ecuación y su derivada, con sus respectivas graficas.

El funcionamiento básico del algoritmo de regresión logística, se puede generalizar en dos puntos, el primero es el modelado de probabilidades en donde se modela la probabilidad de que una entrada pertenezca a una clase particular. Se utiliza la función logística para asignar valores de entrada reales a una probabilidad entre 0 y 1. El segundo punto es el umbral de decisión, es convertir la probabilidad en una decisión binaria utilizando un umbral (comúnmente 0.5), entonces si la probabilidad estimada es mayor que el umbral, la entrada se clasifica en la clase "1", de lo contrario en la clase "0".

De manera formal se puede interpretar matemáticamente con la ecuación (13):

$$P(y = 1|x) = \frac{1}{1 + e^{-(b_0 + b_1 x)}} \qquad (13)$$

Donde:

- $P(y = 1|x)$ es la probabilidad de que la variable dependiente y sea 1 dado el predictor x.
- b_0 y b_1 son los parámetros del modelo, donde b_0 es el término de intercepción y b_1 es el coeficiente del predictor x.
- e es la base del logaritmo natural.

El modelo utiliza el método de máxima verosimilitud para estimar los parámetros b_0 y b_1 de manera que se maximice la probabilidad de los datos observados.

De acuerdo con la Tabla 3.5 este algoritmo se podría clasificar como enfoque simbólico, a pesar de ser basado en ecuaciones y relaciones matemáticas explicitas, se alinea con los métodos simbólicos. Además, produce un modelo que es interpretable en términos de como las características afectan la probabilidad de estar en una clase particular.

3.4.4.5 Algoritmo árbol de decisiones

Este algoritmo es un modelo predictivo que utiliza conjuntos de reglas binarias para calcular una decisión. Es representado en estructuras del tipo árbol, donde cada nodo interno representa una pregunta sobre una característica, cada rama representa la respuesta a la pregunta, y cada hoja representa una etiqueta de clase (en una clasificación) o un valor continuo (en una regresión). El funcionamiento de este algoritmo se puede expresar en tres puntos:

- **División basada en características**: cada nodo del árbol, elige una característica y un punto de corte para dividir el conjunto de datos en dos o más subconjuntos homogéneos. Esta decisión se basa en criterios como la impureza de Gini, la entropía (en clasificación) o el error cuadrático medio (en regresión).
- **Construcción recursiva del árbol**: El proceso se repite de manera recursiva en cada subconjunto hasta que se cumple algún criterio de parada. Algunos ejemplos pueden ser en el que todos los datos en un nodo tengan la misma etiqueta, que el nodo contenga un número mínimo de datos, o que se alcance una profundidad máxima del árbol.
- **Poda del árbol**: Para evitar el sobreajuste (“overffiting”, revisado en la sección “3.4.2 Principales retos del machine learning”), los árboles de decisión suelen ser podados. Esto significa que se deben eliminar algunas secciones del árbol para simplificarlo.

La interpretación matemática de un árbol de decisiones puede ser basada en un grado, pero este dependerá de la naturaleza del problema, es decir depende del problema el número de nodos, que puede llegar a tener para poder representar el grafo.

Con respecto a la clasificación de enfoques vista en la Tabla 3.5, el algoritmo de árbol de decisiones, podría clasificarse como enfoque simbólico. Se debería a que hace uso de un conjunto de reglas lógicas para la toma de decisiones, lo que facilita la interpretación de las mismas.

3.4.4.6 Algoritmo IBK

El algoritmo IBk, también conocido como k-Nearest Neighbors (k-NN), es un método de aprendizaje supervisado utilizado para la clasificación de datos. A diferencia de los algoritmos de construcción de modelos, como los árboles de decisión o las máquinas de vectores de soporte, el algoritmo IBk no construye un modelo explícito durante la fase de entrenamiento. En cambio, clasifica las instancias de prueba basándose en las etiquetas de las instancias vecinas más cercanas en el espacio de características.

La formulación matemática del algoritmo IBk es relativamente sencilla y se basa en el cálculo de la distancia entre instancias en un espacio de características X. Suponiendo un conjunto de datos de entrenamiento $D = \{(x_1, y_1), (x_2, y_2), \dots (x_n, y_n)\}$ donde x_i es el vector de características de la instancia i y y_i es su etiqueta de clase, la clasificación de una nueva instancia de prueba x^* se realiza de la siguiente manera:

- Calcular la distancia entre la instancia de prueba x^* y todas las instancias de entrenamiento en D utilizando alguna medida de distancia, como la distancia euclidiana o la distancia de Manhattan.
- Encontrar los k vecinos más cercanos se seleccionan las k instancias más cercanas a x^* en función de la distancia calculada.
- Clasificar por mayoría se determina la clase de x^* mediante votación mayoritaria entre las etiquetas de los k vecinos más cercanos. Es decir, asigna a x^* la etiqueta de clase más frecuente entre los k vecinos.

Esta formulación se puede expresar matemáticamente de la siguiente manera:

Dada una nueva instancia de prueba x^*, la clase y^* se calcula como:

$$y^* = argmax_{y_i} \sum_{i=1}^{k} \delta(y_i, y)$$

donde $\delta(y_i, y)$ es una función delta que devuelve 1 si $y_i = y$ y 0 en caso contrario, y k es el número de vecinos más cercanos.

3.4.5 Métricas de evaluación

En esta sección de describen las principales métricas de evaluación del rendimiento de los algoritmos de machine learning.

- **True Positive (TP, Positivos verdaderos)**: son las instancias que el clasificador del modelo predice correctamente como ataques.
- **False Positive (FP, Falsos positivos)**: son las instancias erróneamente predichas como ataques por el clasificador del modelo de IDS, ya que el modelo ha fallado y se trata de tráfico de red normal.
- **True Negative (TN, negativos verdaderos)**: son las instancias que el modelo predice correctamente como tráfico de red normal.
- **False Negative (FN, falsos negativos)**: son las instancias que el clasificador del modelo predice erróneamente como tráfico normal cuando en realidad son ataques.

A continuación, se ilustra en la Figura 3.21, los posibles resultados de las predicciones de una instancia de conjuntos en la matriz de confusión.

		Instancia predicha	
		Ataque	Tráfico normal
Instancia actual	Ataque	True Positive (TP)	False Negative (FN)
	Tráfico normal	False Positive (FP)	True Negative (TN)

Figura 3.21 Matriz de confusión de instancias predichas. Fuente: Elaboración propia

Por lo tanto, las métricas en la detección de intrusiones son las siguientes:

- **Precision**: Consisten en el ratio de instancias correctamente predichas como ataques (TP) frente al total de clases predichas como ataques (TP+FP). Por lo tanto, la formula se representa en (14):

$$Precision = \frac{TP}{TP + FP} \tag{14}$$

- **Accuracy**: es la ratio que mide las instancias predichas correctamente frente al número total de instancias. La fórmula se define como en (15):

$$Accurancy = \frac{TN + TP}{TN + TP + FN + FP} \tag{15}$$

- **Detection rate**: Es la ratio de detección que mide la cantidad de ataques detectados (TP) en comparación con el total de ataques del conjunto de datos (TP + FN). La fórmula se representa en (16):

$$DectetionRate = \frac{TP}{TP + FP} \tag{16}$$

- **False alarm rate**: Es la ratio que mide la cantidad de tráfico normal que el modelo predice como ataque cuando no lo es (FP) frente al total de tráfico normal (TN+FP). La fórmula se representa en (17):

$$Recall = \frac{TP}{TP + FN} \tag{17}$$

- **ROC**: Es la curva que mide la relación entre los verdaderos positivos y los falsos positivos. Es utilizado para comparar modelos óptimos frente a otros subóptimos.
- **F Measure (F-Score)**: es una medida definida como la medida armónica de la precisión y la ratio de detección, utilizada para medir el rendimiento de un modelo, la formula se representa en (18):

$$F - Score = 2\left(\frac{Precision * DetectionRate}{Precision + DetectionRate}\right) \tag{18}$$

3.5 Ciberataques

En este apartado se pretende describir los más populares tipos de ataques a una red de datos, describiendo su principal funcionamiento. En algunos casos a modo de ejemplo, se ilustran, mediante scripts específicamente diseñados para ejecutar pruebas de concepto. Sin embargo, es pertinente subrayar que el alcance de este estudio no abarca la explicación técnica de cómo prevenir estos ataques, ya que esto excede el marco de la investigación actual.

En el artículo “Cyber Attacks and Its Differents Types” de J. M. Biju y otros autores [35], se describe a un ciberataque como “*un intento de acceder al sistema de información de un individuo o una organización utilizando código o herramientas con intenciones maliciosas”*. Se puede intuir entonces que un ciberataque repercute en la intención de un actor (persona u organización) que ejecuta alguna herramienta (o código) con malas intenciones.

3.5.1 Breve repaso de redes y protocolos

En su obra [36], J. Forshaw, describe a una red de datos, como un “*conjunto de dos o más computadoras o equipos de informáticos, conectados entre sí, con el fin de compartir información”.* Para que pueda existir la comunicación entre estos dispositivos, existen reglas que llevan a cabo esa comunicación en distintas fases, a estas reglas se les llama “protocolos”, un protocolo es el medio de comunicación de un dispositivo con otro, existen diferentes tipos de protocolos, entre los más famosos están TCP, UDP, IP, ICMP, ARP, entre muchos más. Tienen una extensa cantidad de aplicaciones y funciones.

El protocolo más utilizado es TCP/IP, es la combinación de dos protocolos, TCP de Transmission Control Protocol e IP de Internet Protocol, estos dos protocolos forman parte de la suite IPS (Internet Protocol Suite), un modelo conceptual de cómo trabajan en conjunto los protocolos para proveer de tráfico a los dispositivos de cómputo. Se ilustra este modelo en la Figura 3.22.

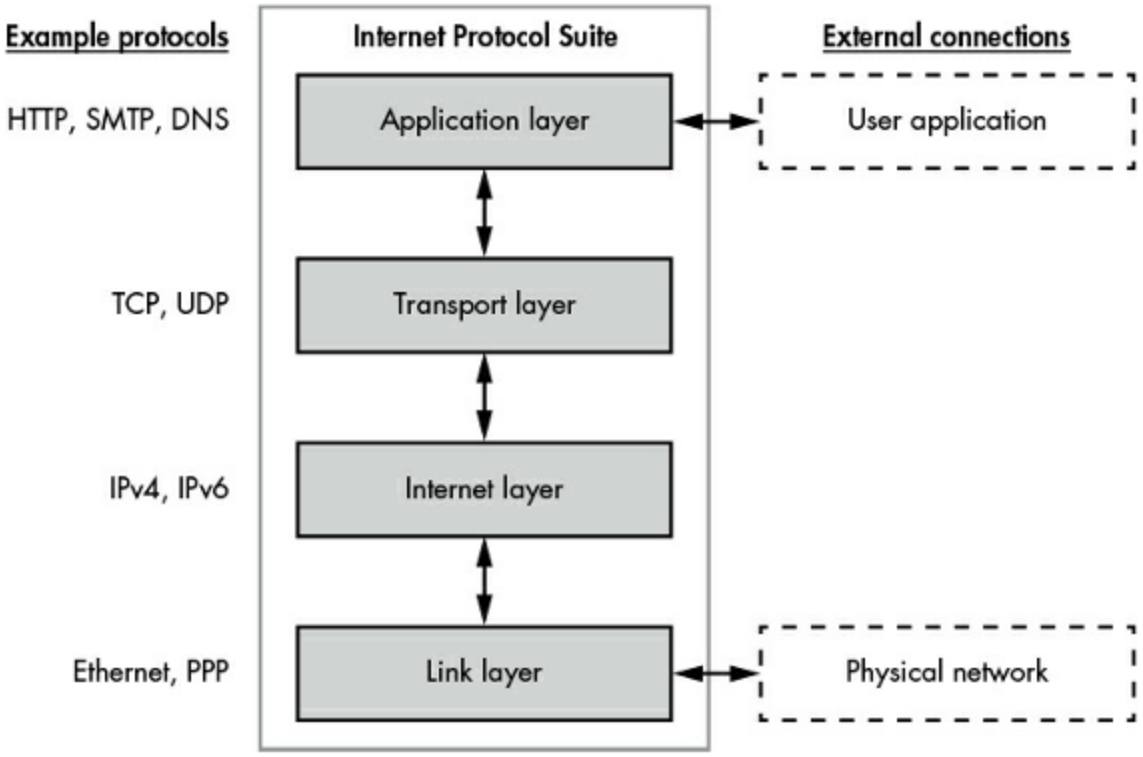


Figura 3.22 Modelo Internet Protocol Suite (IPS). Fuente [36]

Se observa en la Figura 3.22 que el modelo IPS cuenta con cuatro capas, estas tienen distintas funciones descritas a continuación, pero es importante recordar que el modelo IPS, tiene sus orígenes en el Modelo OSI, un modelo conceptual con siete capas, diseñado para abstraer los conceptos de redes y que los administradores de redes o personal que necesita entender el campo, puedan entender el funcionamiento de los protocolos y sus funciones de manera teórica.

Capas del modelo IPS

- **Capa de Enlace**: Esta capa describe los mecanismos físicos utilizados para la transferencia de información entre los distintos nodos o equipos.
- **Capa de Internet**: Esta capa proporciona los mecanismos para el direccionamiento de los distintos nodos de la red. En su versión más actual hace uso del protocolo IPv6, sin quitar relevancia a su antecesor IPv4, pero aún queda bastante para esté.

- **Capa de Transporte**: Esta capa es la responsable de las conexiones entre los clientes y los servidores, verificando el correcto orden de los paquetes y su distribución. Sirve para el soporte de múltiples servicios asignados a distintos números, estos números son llamados puertos.
- **Capa de Aplicación**: Contiene los protocolos en donde el usuario interactúa con las aplicaciones.

El proceso de cómo se transmite información de un nodo a otro se llama **"encapsulamiento de datos"**, es parte del modelo IPS, consiste en un protocolo en donde cada paquete enviado, da y recibe información a cada capa, a este proceso se le llama PDU (Protocol Data Unit).

El PDU es enviado a través de las capas, con una carga útil denominada **payload**, esta es transmitida tomando y dando información de las distintas capas durante su trayecto, se puede ver ilustrado este proceso en la Figura 3.23.

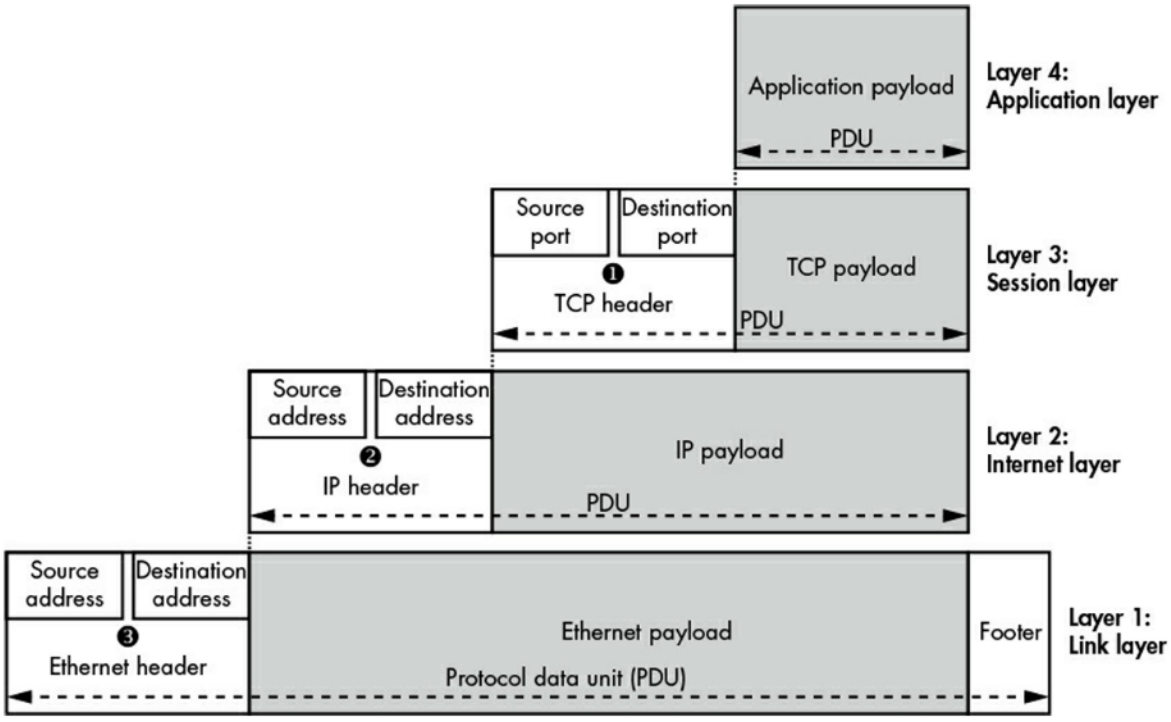


Figura 3.23 Encapsulación de datos en el modelo IPS. Fuente [36]

3.5.2 DoS y DDoS

Un ataque de denegación de servicio DoS por sus siglas en inglés (Denial Of Service) según la empresa Cloudflare, [37] especializada en la prevención y mitigación de este tipo de ataques, lo describe como un ataque en donde un actor con intenciones maliciosas pretende que un servicio o equipo de cómputo, no se encuentre disponible para los clientes (usuarios u otros servicios). Este tipo de ataque normalmente funcionan enviando una cantidad de peticiones superior a la esperada a un servicio, abrumándolo hasta el punto de desbordar su capacidad de respuesta.

De manera general se puede clasificar un ataque de tipo DoS, dos tipos; Buffer Overflow, y Flood attacks, términos muy conocidos en el campo de la ciberseguridad. El primero conocido en español como ataque de desbordamiento, se centra en sobrecargar las capacidades de cómputo de un dispositivo o red, mediante la carga masiva de tareas en memoria, o ficheros en la unidad de almacenamiento. El segundo es quizá el más popular por los medios de comunicación, popularizado por ataques famosos como Mirai [38], consiste en que el actor malicioso realice múltiples envíos de paquetes a un servidor, con el objetivo de inundar (Flood) el servicio. Siendo este tipo de ataque elegido para ser simulado dentro del marco de esta investigación.

Este tipo de ataques evolucionó, utilizando las mismas técnicas, pero de manera distribuida, consiste en replicar el ataque DoS, pero en más de un dispositivo atacante, generalmente los cibercriminales hacen uso de redes comprometidas llamadas Botnet. Un ejemplo histórico es la botnet Mirai [38], desarrollada en el año 2012, popularizándose en el año 2016 ejecutando un ataque de denegación distribuida (DDoS), afectando dos continentes y cientos de servicios [39]. En ese momento la botnet Mirai tenía un aproximado de 600,000 dispositivos infectados, listos para ejecutar cientos de peticiones por segundo cada uno. Un ejemplo de la posible distribución global de los dispositivos infectados por Mirai, se ilustra en la Figura 3.24.

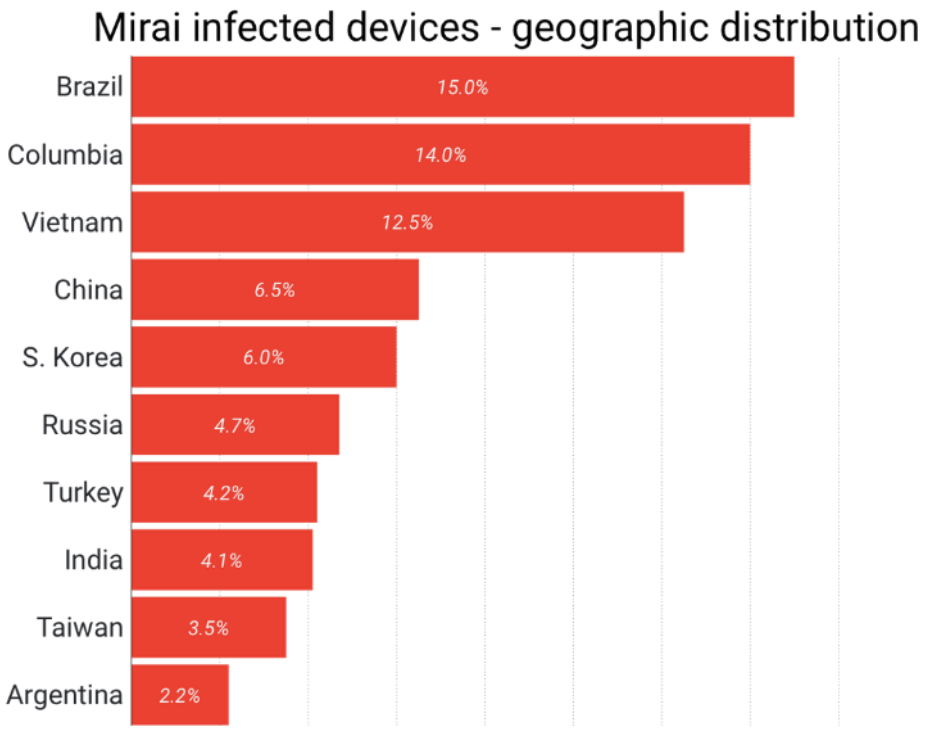


Figura 3.24 Distribución geografíca de los dispositivos infectados por botnet Mirai. Fuente [38]

3.5.3 Escaneo de puertos

Los investigadores de la empresa Malwarebytes Labs [40] describen que un ataque de escaneo de puertos consiste en la fase de reconocimiento de un ciberataque, en donde un actor malicioso escanea sistemáticamente un nodo (host o red), en busca de información que sea de utilidad para perpetrar el ataque, mediante los puertos que posibiliten esta acción, estos pueden estar abiertos o filtrados.

Se pueden clasificar dos tipos de escaneo de puertos, descritos por la empresa Kaspersky [41], como tipo horizontal (también llamado de red) y tipo vertical (de host). El primero consiste en realizar reconocimiento enviando solicitudes al mismo puerto de distintos hosts en una misma red, este tipo de ataque quizá también podría ser considerado como un ataque DoS, dependiendo de la cantidad de solicitudes enviadas. El segundo consiste en que el actor malicioso realice un escaneo de un determinado host.

Los métodos de escaneo de puertos más comunes son:

- **Escaneo SYN**: El atacante envía una solicitud SYN (de sincronización) al puerto a través del protocolo TCP, en donde si el puerto está abierto, se devuelve un paquete que indica SYN-ACK donde confirma la sincronización, o de lo contrario un paquete RST que indica que el puerto está cerrado o filtrado por una herramienta de monitoreo, como un firewall.
- **Escaneo TCP**: Consiste en encontrar puertos abiertos con servicios que hagan uso del protocolo TCP.
- **Escaneo UDP**: Es muy parecido al escaneo TCP, la diferencia es el protocolo UDP como vector de ataque, y si el puerto está abierto los paquetes enviados se entregan y no hay necesidad de respuesta del servicio.

El ataque de escaneo vertical, es de interés particular de esta investigación, ejecutado como prueba de concepto y simulación en un entorno controlado, para la creación de un data set. Existen una extensa cantidad de herramientas para realizar este tipo de actividades, la más popular por su capacidad es Nmap (Network Mapper) es una herramienta muy útil, popular entre el arsenal de los profesionales de la ciberseguridad [42].

3.5.4 Ataques SSH de fuerza bruta

El protocolo SSH o Secure Shell es un popular protocolo criptográfico utilizado por usuarios para conectarse a un equipo de manera remota, mediante el acceso con credenciales, generalmente se ejecuta el servicio SSH en el puerto 22, entonces si un actor malicioso realiza un escaneo es posible detectar el puerto 22 abierto o filtrado.

El ataque de fuerza bruta según lo describe un artículo publicado en 2015 en la 14th International Conference on Machine Leaning and Application de IEEE [43], "*es cuando un actor malicioso intenta ganar acceso al servidor remoto SSH, probando distintas credenciales, mediante herramientas especializadas o scripts específicos"*.

Este tipo de ataques es de interés la presente investigación, en donde se simulo un ataque de diccionario y fuerza bruta a un servicio SSH.

3.5.5 Os Finger Printing

El ataque OFP es parte del reconocimiento para un posterior ciberataque más sofisticado, en donde el actor malicioso, intenta detectar que sistema operativo ejecuta el nodo víctima. Existen una gran variedad de herramientas para ejecutar este reconocimiento. La más sencilla, es mediante el comando ping, dependiendo del tiempo de vida de la respuesta TTL, es como se podría detectar que sistema operativo, se ejecuta, igualmente se puede desarrollar una herramienta que ejecute este reconocimiento más específico, siendo este el caso de la presente investigación, en donde se desarrolló en el lenguaje de programación Python un script que determina el posible sistema operativo de la víctima. Se ilustra en la Figura 3.25.

```python
from scapy.all import *

# Función para realizar OS Fingerprinting
def os_fingerprinting(target_ip):
    # Envío de paquete ICMP Echo Request
    packet = IP(dst=target_ip)/ICMP()
    response = sr1(packet, timeout=1, verbose=0)

    # Análisis de la respuesta recibida
    if response:
        if response.ttl <= 64:
            print("Probablemente es un sistema operativo basado en Unix (Linux, macOS, etc.)")
        else:
            print("Probablemente es un sistema operativo basado en Windows")
    else:
        print("No se recibió respuesta")

# Dirección IP del dispositivo objetivo
target_ip = "192.168.93.135"

# Llamada a la función de OS Fingerprinting
os_fingerprinting(target_ip)
```

Figura 3.25 Script en Python, que simula un ataque OFP. Fuente: Elaboración propia

3.6 Herramientas para el despliegue del laboratorio

El siguiente segmento de este documento, justifica, el uso de distintas herramientas en el despliegue del laboratorio y pruebas experimentales. Se resalta el uso de herramientas de tipo software libre como sistemas operativos basados en el Kernel de Linux, y herramientas opensource para el análisis de datos.

3.6.1 Hipervisor

Un hipervisor o también conocido como un monitor de máquina virtual (VMM), es un software capaz de crear y ejecutar máquinas virtuales o VM (Virtual Machine) en un host [44]. Este le permite al host admitir más de un sistema operativo distinto al suyo o no, compartiendo los recursos de cómputo y red del mismo.

Estos se pueden dividir en dos tipos, ilustrados en la Figura 3.26. El primer tipo se ejecutan en un sistema operativo host, en donde ejecuta a su vez otros sistemas como parte de la aplicación del hipervisor, el tipo 2 es el que distribuye los recursos de hardware entre distintos sistemas.

Para el propósito de esta investigación, y con fines prácticos, se eligió el primer tipo de hipervisor, en donde se simulo un laboratorio virtual.

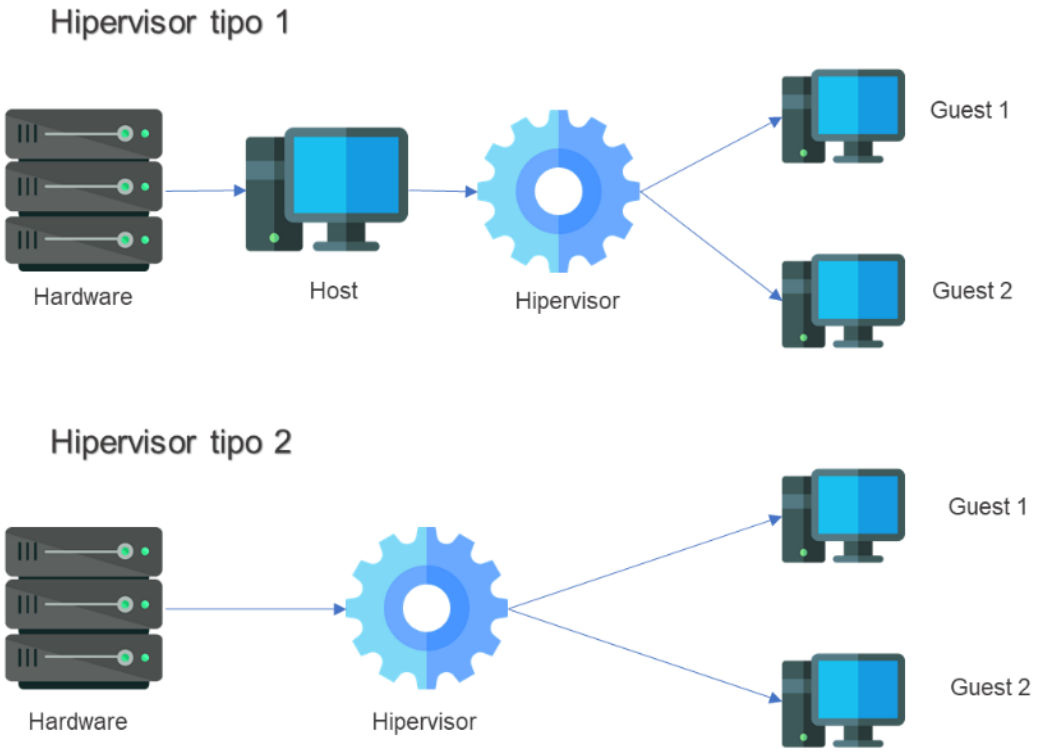


Figura 3.26 Tipos de hipervisor. Fuente: Elaboración propia

La elección del hipervisor utilizado para esta investigación, se basó en la fácil configuración de una red virtual, para simular un entorno controlado, el hipervisor elegido fue VMware Workstation Pro 17, si bien es un software de tipo propietario, así como de

paga, al comienzo de esta investigación, se contaba con el mismo, por lo que no representó un gasto económico en la investigación. La interfaz del hipervisor utilizado se ilustra en la Figura 3.27.

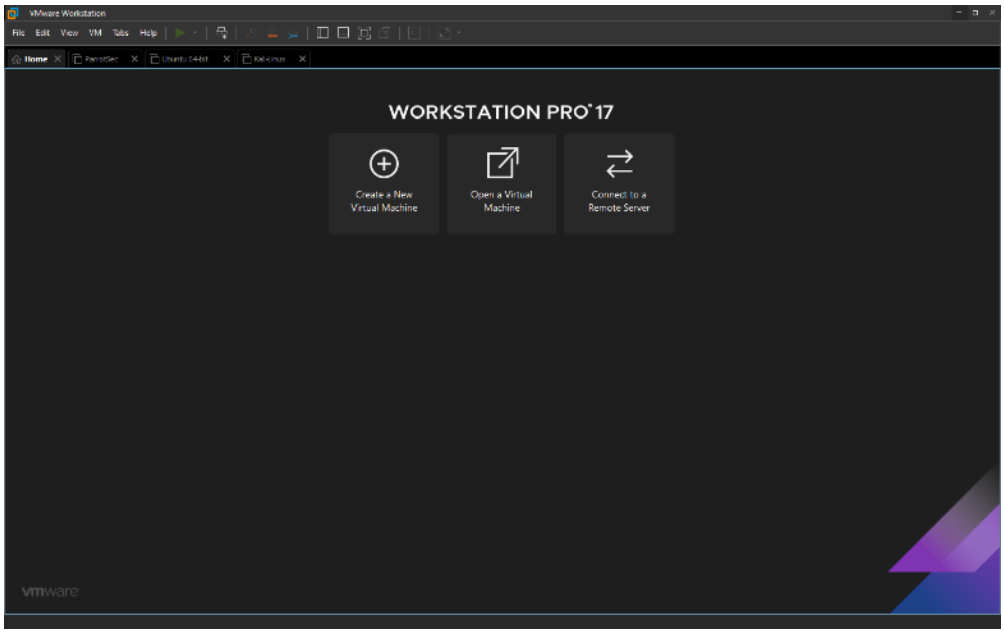


Figura 3.27 Laboratorio virtual mediante el Hipervisor VMware PRO 17. Fuente: Elaboración propia

Existen otras alternativas para un hipervisor además VMware Pro 17, como, Hyper-V, Oracle Virtual Box, Oracle VM, Gnome Boxes, entre muchos más, incluso la versión gratuita del mismo VMware. Esta opción fue la elegida, por su fácil manejo de redes virtuales, redes NAT, y facilidad de instalación.

3.6.2 Sistemas Linux

A continuación, se explica brevemente los sistemas operativos desplegados en el laboratorio de pruebas, desatacando que todos los sistemas son software libre. Es importante mencionar que el laboratorio se desplegó en un entorno controlado.

3.6.2.1 Ubuntu

El sistema operativo Ubuntu, es un sistema basado en el Kernel del sistema Debian GNU/Linux, desarrollado y patrocinado por la organización canonical. Es un sistema muy estable, compatible con software de tipo propietario, amigable interfaz y una curva de aprendizaje muy baja, con una gran comunidad.

Este sistema se utilizó principalmente en el marco experimental para la implementación de un sistema IDS, configurado para recibir ataques de otros equipos en el mismo segmento de red, el sistema IDS instalado y configurado con reglas personalizadas fue Snort. Además de fungir como un sistema IDS, Ubuntu, fue utilizado para el análisis de datos, con la herramienta Elasticsearch y Kibana, así como la ejecución de simulaciones de clasificación con Weka, herramientas de las que se detalla más adelante. La interfaz del sistema Ubuntu, se presenta en la Figura 3.28.

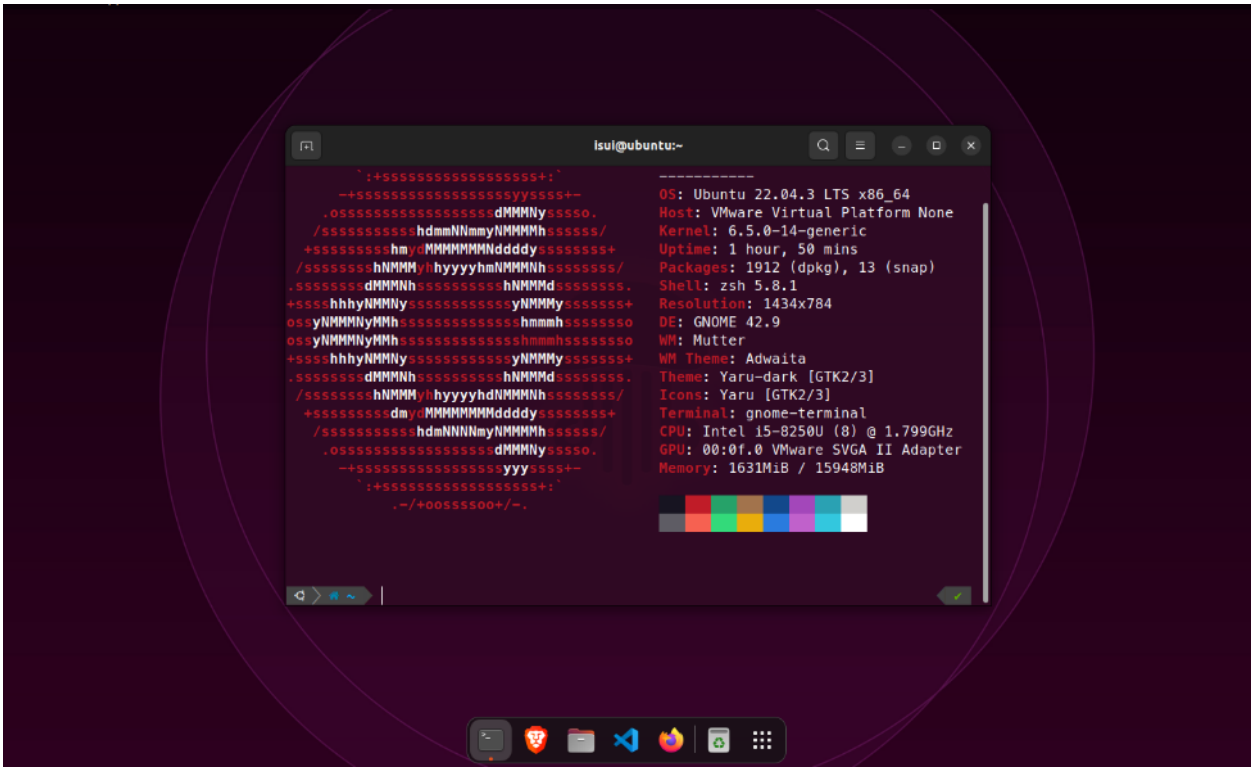


Figura 3.28 Interfaz del sistema operativo Ubuntu. Fuente: Elaboración propia

3.6.2.2 Kali Linux y Parrot Security

Los sistemas para simular a un actor malicioso, elegidos en esta investigación, fueron Kali Linux y Parrot Security, ambos sistemas orientados al uso de auditorías de seguridad.

Kali Linux es el sucesor del sistema operativo BackTrack [45], actualmente Kali Linux es patrocinado completamente por la empresa OffSec antes conocida como Offensive Security, una muy importante organización, en donde además de llevar el desarrollo de Kali, es promotora de las más reconocidas certificaciones para profesionales en el campo

de la ciberseguridad ofensiva, algunos ejemplos de estas certificaciones, son: OSCP, OSEP, OSWE3, OSMR, entre muchas otras más.

Tanto Parrot Security como Kali Linux son sistemas basados en el Kernel del sistema Debian GNU/Linux, así como de un sistema de ficheros FHS-compliant [45]. De manera general el sistema Kali, es una herramienta esencial para los profesionales de la seguridad, su principal enfoque se centra en la seguridad ofensiva o Red Teaming (equipo rojo de seguridad ofensiva), cuenta con herramientas para auditorias de red, auditorias de redes inalámbricas, de radio frecuencias, análisis forenses, análisis de muestras de malware, y muchas herramientas preinstaladas, así como para el desarrollo en entornos Linux. La implementación de Kali Linux en el laboratorio se ilustra en la Figura 3.29.

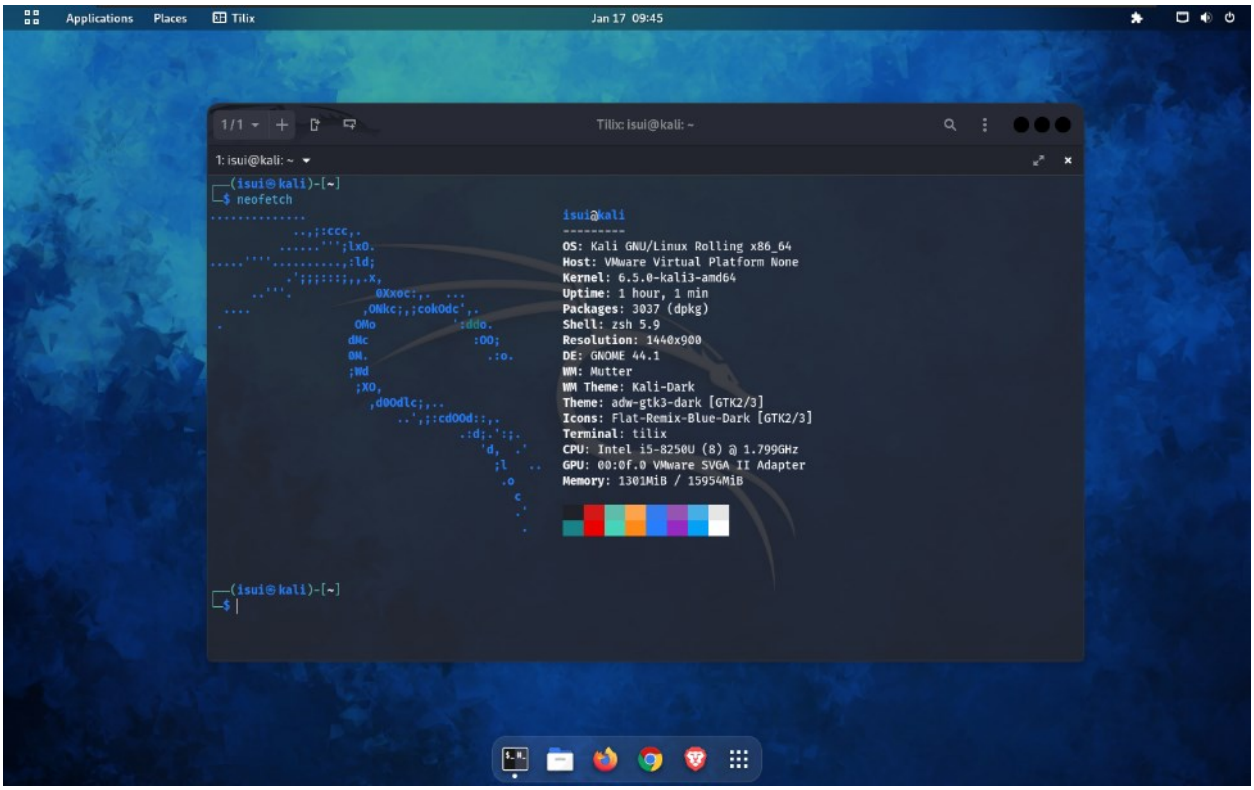


Figura 3.29 Interfaz del sistema operativo Kali Linux. Fuente: Elaboración propia

El siguiente sistema operativo, implementado en el laboratorio, fue el sistema Parrot Security, este es un sistema orientado para los profesionales de la seguridad, al igual que Kali Linux, son herramientas que todo profesional de la seguridad emplea en su dia a dia. Parrot Security es la versión para auditorias del Proyecto Parrot OS, baso en el

Kernel del sistema Debian GNU/Linux. Cuenta con una amplia cantidad de herramientas para auditorias de seguridad. Para el propósito de esta investigación, se realizó la instalación en un entorno controlado de ambos sistemas Parrot y Kali, en el hipervisor VMware. La interfaz del sistema Parrot se puede ver en la Figura 3.30.

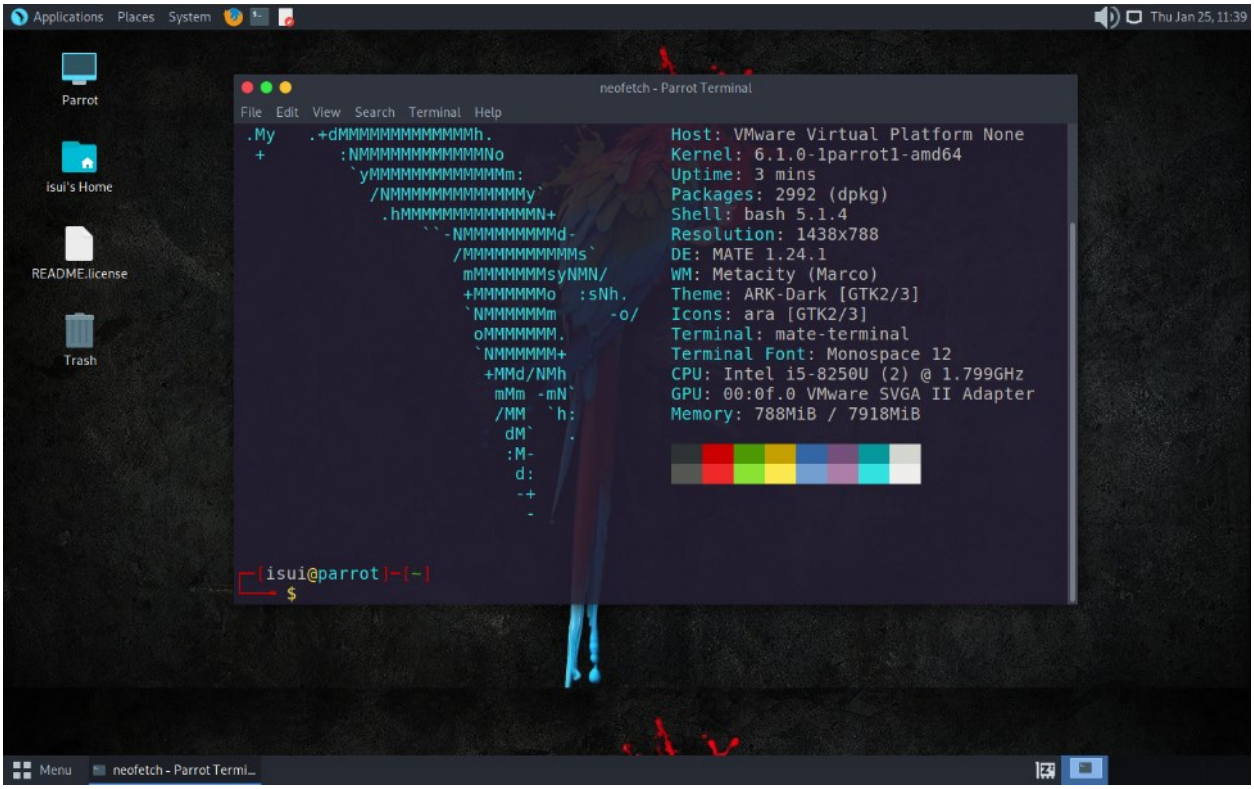


Figura 3.30 Interfaz del sistema operativo Parrot Security. Fuente: Elaboración propia

3.6.3 Elasticsearch para el análisis de datos

Elasticsearch es una herramienta para el análisis de datos, basado en la biblioteca de Apache Lucene, es open Source, pero su verdadero potencial es como un motor de análisis junto con otras herramientas de paga, como Logstach y Beats [46]. Elasticsearch es desarrollado y patrocinado por la empresa Elastic, una empresa enfocada en el análisis de datos.

Elasticsearch hace uso de Kibana, un software también de Elastic, que muestra gráficamente los datos, y permite al usuario una mejor visualización de los datos, además implementa muchas otras opciones en donde se pueden crear Dashboards, análisis estadísticos, e implementación de modelos de inteligencia artificial, aunque esto último

solo para la versión empresarial de Elasticsearch. La versión libre, permite crear tareas simples de predicción, y detección de anomalías en grandes cantidades de datos.

El funcionamiento de Elasticsearch, se basa en organizar los documentos, conformados por registros o datos generalmente en formato JSON, o CSV, los cuales representan entidades y se agrupan en índices.

El apartado de interés de esta investigación sobre Elasticsearch y Kibana, son las funciones de Machine Learning que realizan, si bien la versión empresarial cuenta con muchas opciones, en la versión de prueba, se pueden realizar trabajos (Jobs) un Job, es lo que se conoce en Elasticsearch al procesamiento sobre los datos utilizando Machine Learning. En la versión de prueba, solo es posible hacer uso predicciones y trabajos simples, que son útiles para el propósito de esta investigación, donde se pueden ver los falsos positivos y negativos de nuestros datos en base a predicciones, así como la detección de anomalías, y predicciones del comportamiento del tráfico durante un intervalo de tiempo determinado. La interfaz de inicio de sesión de Elasticsearch se ilustra en la figura Figura 3.31.

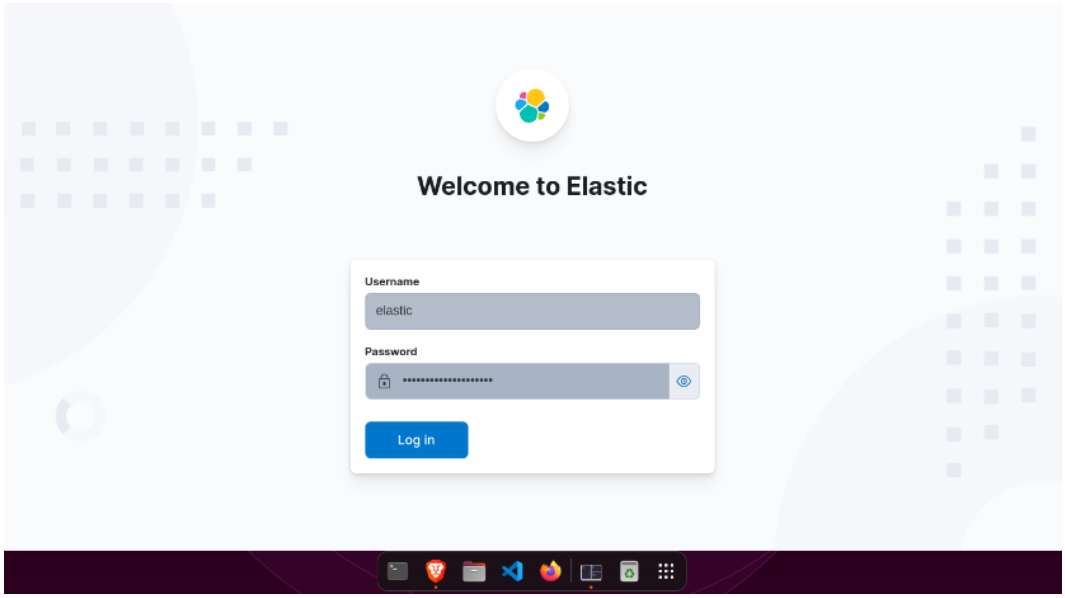


Figura 3.31 Log in de Elasticsearch en Local. Fuente: Elaboración propia

3.6.4 Weka para la utilización de algoritmos de ML

El software utilizado en esta investigación para analizar diferentes modelos de clasificación, regresión, y agrupación utilizando técnicas de Machine Learning, fue

principalmente Weka. Este software es desarrollado por la universidad Waikato, en Nueva Zelanda.

Tiene como objetivo poner a disposición las técnicas de Machine Learning, aplicar problemas prácticos, y el desarrollo e impulso de nuevos algoritmos de aprendizaje. Se encuentra disponible para distintos sistemas operativos, principalmente para sistemas basados en el Kernel de Unix, pero de igual manera hay para Windows. La interfaz de Weka, se ilustra en la figura 3.32.

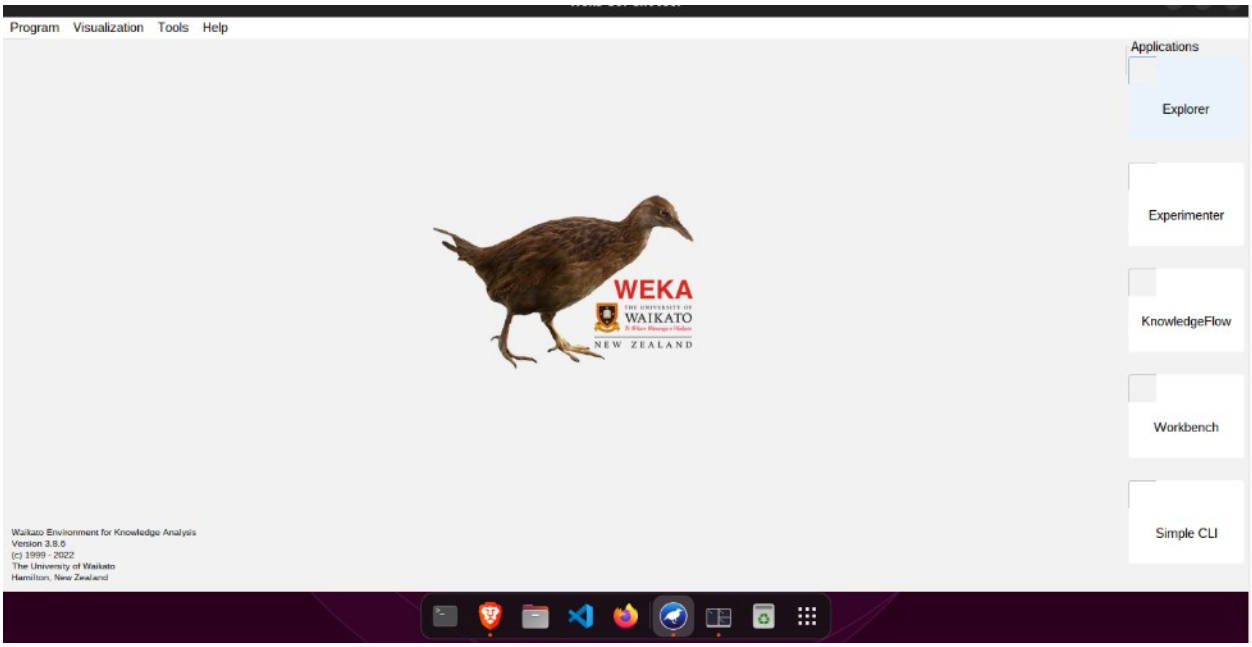


Figura 3.32 Interfaz gráfica de Weka. Fuente: Elaboración propia

3.6.5 Tensor Flow

TensorFlow es una biblioteca de código abierto diseñada para el lenguaje de programación Python, concebida con el objetivo primordial de servir como una herramienta versátil en ámbitos que abarcan desde la investigación hasta la producción, pasando por el prototipado y otros usos especializados. Esta biblioteca fue desarrollada por la empresa tecnológica Google [48].

Figura 3.33 Ejemplo ilustrativo de Tensor Flow. Fuente: github.com/tensorflow

Es importante destacar que TensorFlow hace uso de la API Keras, una interfaz de red neuronal de alto nivel, también implementada en el entorno Python. Además, integra la librería NumPy, reconocida por su capacidad para brindar soporte en el manejo de matrices y arreglos multidimensionales de gran escala, ampliando así la funcionalidad y eficiencia.

CAPÍTULO 4 MARCO EXPERIMENTAL

El presente capitulo aborta el marco experimental del proyecto de investigación, donde se detalla la implementación y despliegue de un laboratorio de pruebas virtualizado. El detalle y explicación de los ataques simulados a un servidor con distintos servicios ejecutándose.

Una preparación de los datos adquiridos para la creación del Data Set, el entendimiento de los mismos, la preparación, modelado y ejecución de los modelos de machine learning sobre el data set.

4.1 Arquitectura del proyecto

El primer paso en la ejecución práctica del proyecto, es la elaboración de la arquitectura del mismo, ilustrada en la Figura 4.1.

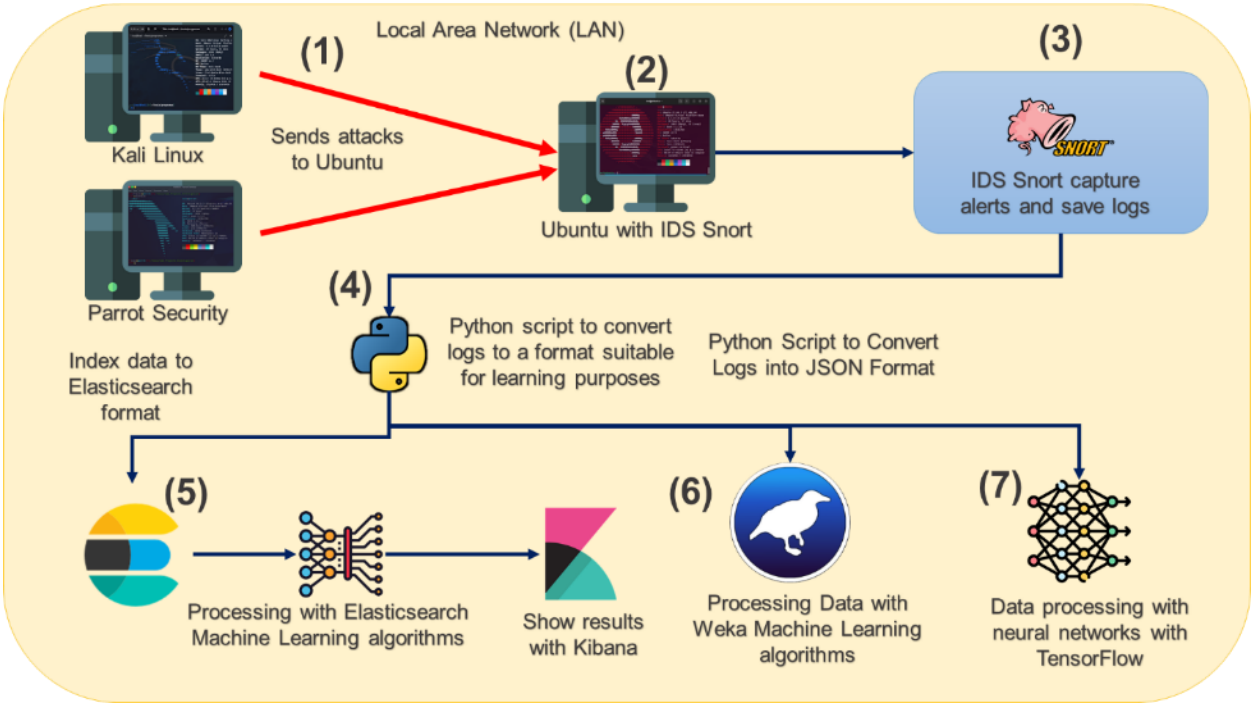


Figura 4.1 Arquitectura del proyecto. Fuente: Elaboración propia

El esquema de la Figura 4.1, refleja las etapas del desarrollo del marco experimental. Comenzando con el despliegue de un laboratorio de pruebas en una red virtual controlada. Estas etapas se muestran en un orden ascendente, explicado a continuación.

La primera etapa (1), especifica la fase en que se simulan los ciberataques a un servidor IDS, este servidor se especifica en (2), luego en (3), se describe como el sistema IDS Snort, mediante reglas personalizadas, ejecuta las alertas, y guarda los registros.

En (4), se comienza con el tratamiento de los datos, donde mediante de Scripts en el lenguaje de programación Python, se realiza un tratamiento de los registros, convirtiéndolos en formato JSON, y CSV, eliminando los datos que no proporcionan conocimiento para su posterior análisis.

En (5) es procesado el conjunto de datos mediante el motor de indexación Elasticsearch y Kibana, para aplicar modelos básicos de ML, con predicciones y detección de anomalías en base a una línea temporal.

En (6), el conjunto de datos, es analizado por medio de clasificadores (algoritmos de ML) en Weka, para su posterior comparación.

En (7), el conjunto de datos, fue analizado por medio de las redes neuronales desarrolladas con TensorFlow.

4.2 Preparación del laboratorio

En primera instancia, se acordó con la institución la asignación de un equipo de cómputo, en sus instalaciones, para la ejecución de la investigación, dadas ciertas circunstancias, no previstas por ninguna parte tanto del autor como de la institución, fue imposible hacer uso de dicho equipo por un largo periodo de tiempo. Una vez resuelta dicha situación, se determinó que el equipo asignado no contaba con la suficiente capacidad de cómputo para el despliegue del laboratorio. Dada esta decisión, el laboratorio de pruebas, fue ejecutado en el equipo de cómputo del autor, del cual se describen las características técnicas en la Tabla 1.1.

El primer paso, fue configurar una red virtual, con acceso a internet, por ello se optó por implementar una red NAT. La configuración de la red virtual, consto de pasos muy sencillos, gracias al Hipervisor en uso (VMware Workstation Pro 17), esto significo, un fácil entendimiento, configuración y administración de la red, como se ilustra en la Figura 4.2.

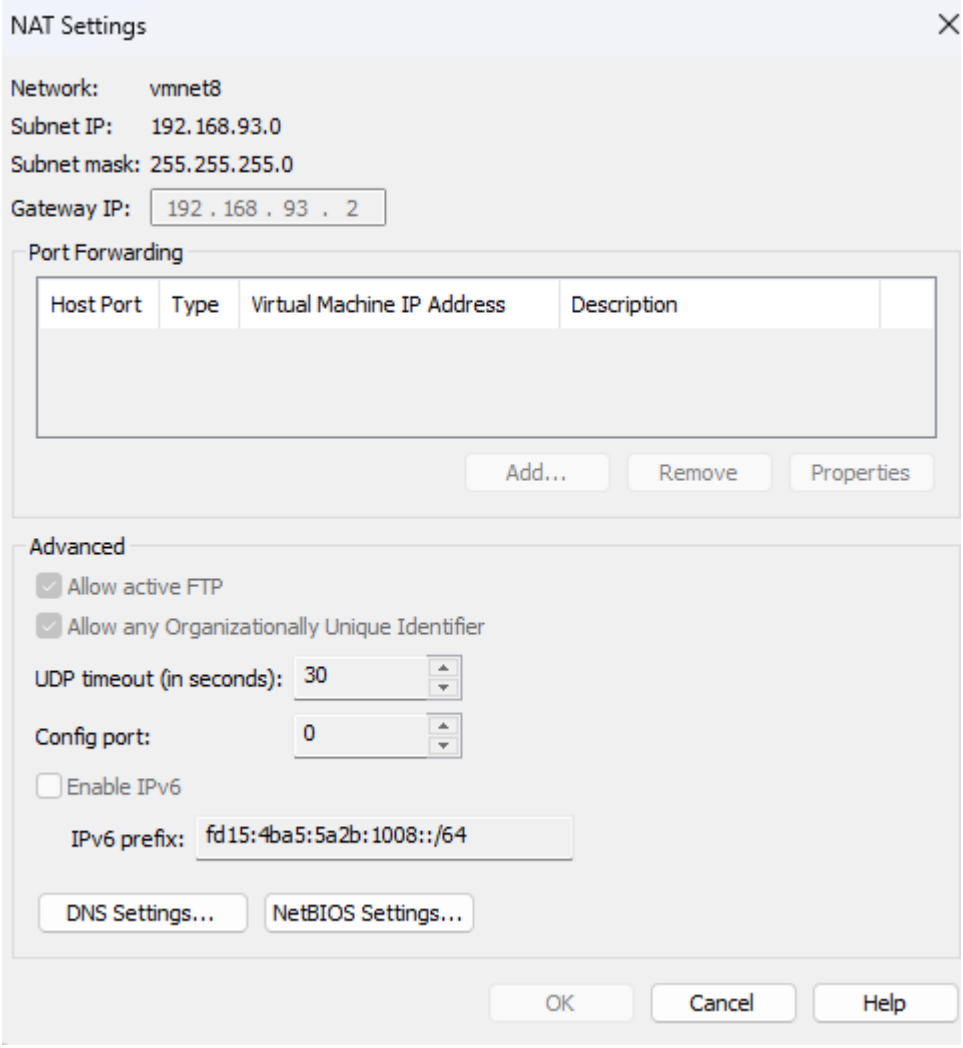


Figura 4.2 Configuración de la red virtual. Fuente: Elaboración propia

Una vez establecidas la configuración de la red, se decidió, que la asignación de IP mediante el protocolo IPv4, se asignara de manera dinámica, mediante el servicio DHCP, esta asignación de IP se ilustra en la Tabla 4.1.

Tabla 4.1 Asignación de direcciones IP a host. Fuente: Elaboración propia

Dirección IPv4	Host
192.168.93.128	Kali Linux
192.168.93.137	Parrot Security
192.168.93.135	Ubuntu

La asignación de recursos de hardware, se realizó de manera que el servidor Ubuntu, obtuviera la mayor cantidad de recursos disponibles, para las posteriores fases de

análisis de datos. Además de ejecutar más de un servicio al momento de las simulaciones de posibles ciberataques. Las especificaciones técnicas de los tres sistemas se presentan en la Tabla 4.2.

Tabla 4.2 Especificaciones tecnicas de sistemas del laboratorio. Fuente: Elaboración propia

Recursos	Ubuntu 22.04.3 LTS	Kali Linux	Parrot Security
Núcleos de procesador	4	4	4
Memoria RAM	16 GB	16 GB	8 GB
Capacidad de almacenamiento	100 GB	60 GB	60 GB

4.2.1 Instalación de Snort

La instalación del sistema IDS, constó de sencillos pasos. Esta se realizó en el sistema Ubuntu, para poder implementar las posteriores simulaciones. La instalación se llevó a cabo desde los repositorios oficiales de Canonical, la empresa que patrocina Ubuntu. La instalación se ejecutó mediante el sistema de paquetes de APT. La versión utilizada de Snort fue la 2.9.15.1, si bien es posible utilizar versiones más recientes, lo recomendado es utilizar una versión lo bastante estable para poder crear modificaciones en el sistema. Una ilustración del sistema Snort ejecutándose en el entorno de pruebas de presenta en la Figura 4.3.

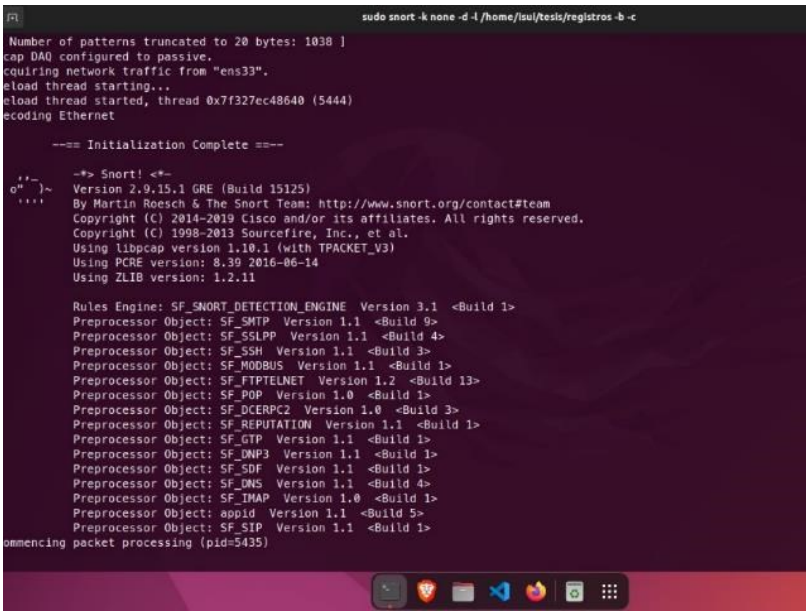


Figura 4.3 Ejecución del sistema IDS Snort. Fuente: Elaboración propia

4.2.2 Configuración de reglas de Snort

Una forma abstracta de observar el funcionamiento de Snort, es mediante el esquema de la Figura 4.4. No obstante, es importante resaltar que no es marco de investigación el funcionamiento de Snort, solo se describirá brevemente, de acuerdo a la importancia de este trabajo, en donde se resalta la fase del motor de detección, en donde se generan las alertas y logs, que posteriormente formaron parte del conjunto de datos.

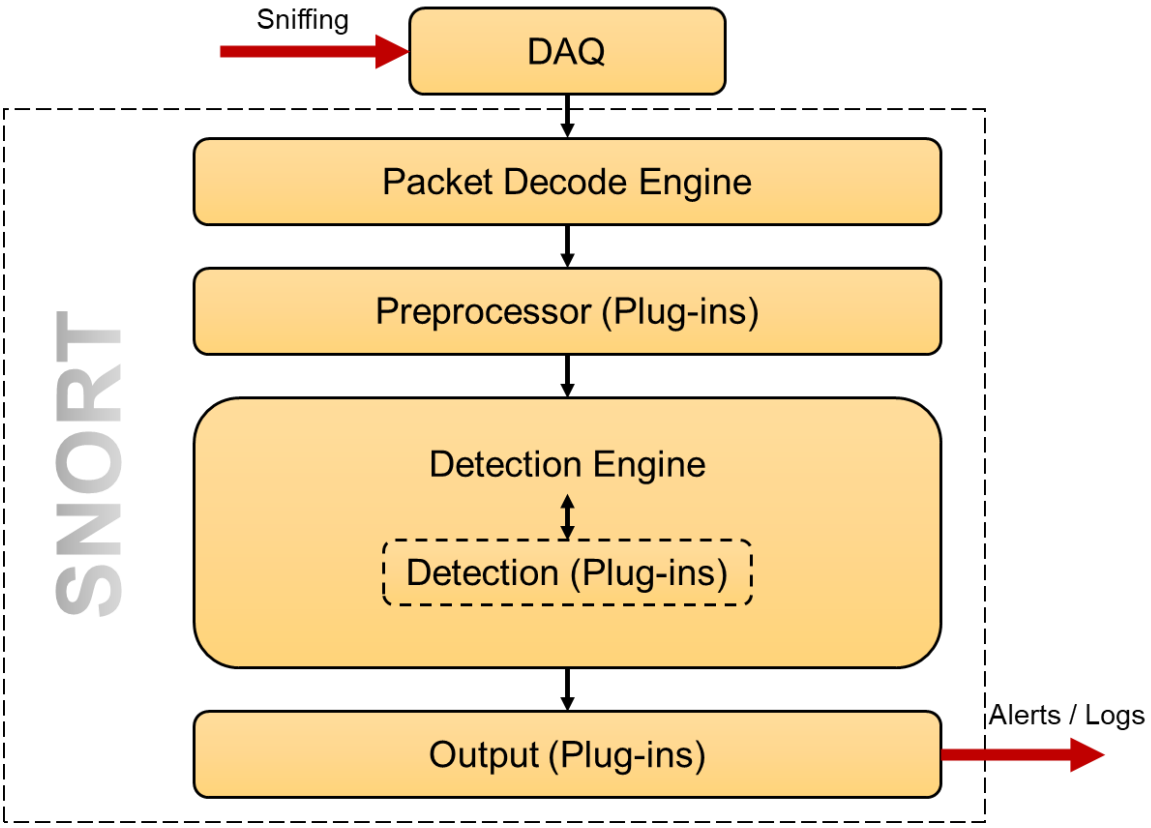


Figura 4.4 Funcionamiento de Snort. Fuente [49]

El funcionamiento generalizado del sistema IDS Snort, se trata de 5 pasos básicos, desde que entran los paquetes, hasta que se activan las alertas y generan logs, el primer paso se basa en la detección del paquete, mediante la librería DAQ (Data Acquisition), luego el motor decodificador de paquetes (Packet Decode Engine) recibe el paquete, que se encarga de organizar la estructura del mismo, para ser enviado al preprocesador, que analiza el paquete y agrega información del análisis, luego el paquete pasa a la fase del motor de detección (Detection Engine), en donde se analiza la información de las fases

anteriores en base a firmas, y se decide si activar o no una alerta, por último se da una respuesta y si el paquete es detectado como sospechoso (Output), se activa una alerta.

En cuanto todos los paquetes han sido preprocesados, pasan a la fase del motor de detección, en donde se comprueba si son posibles paquetes maliciosos o no. Una manera abstracta de ver el funcionamiento del funcionamiento del motor de detección, como se muestra en la Figura 4.5.

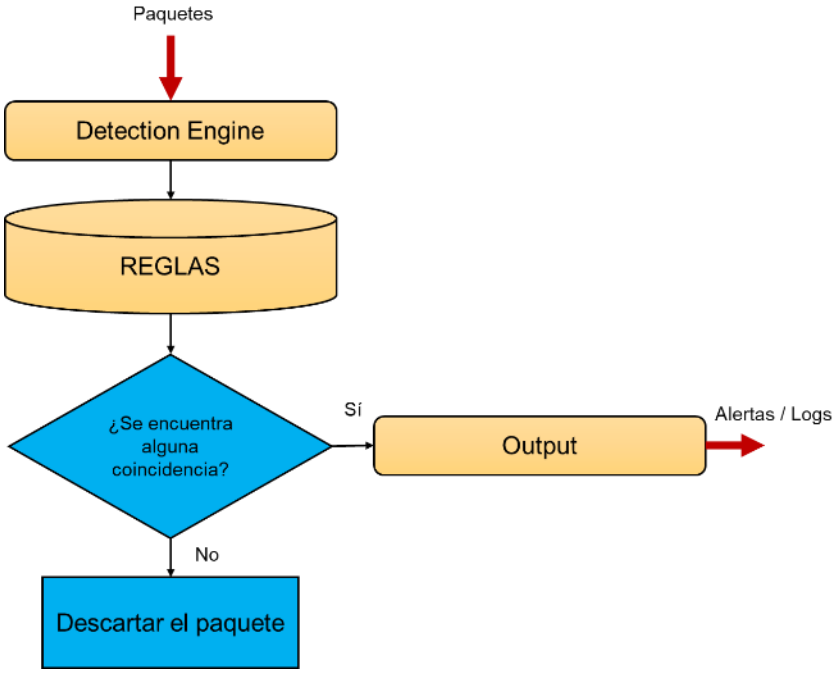


Figura 4.5 Funcionamiento del motor de detección de Snort. Fuente [49]

El interés de esta explicación, se basa en las reglas, un archivo de configuración, que se ejecuta junto con Snort, estas pueden ser o no, reglas personalizadas, Snort, ya cuenta con una gran cantidad de reglas, y su comunidad aporta muchas más. El propósito de este trabajo es crear reglas personalizadas, para crear un conjunto de datos a través de los registros que generan alertas que contienen las reglas personalizadas.

La sintaxis de las reglas de Snort es muy amplia, incluso se podría llevar un curso completo de solo Snort, bastantes horas, es por ello que se resume, y solo te toma en cuenta la sintaxis básica para crear reglas personalizadas. Con se muestra en la Figura 4.6.

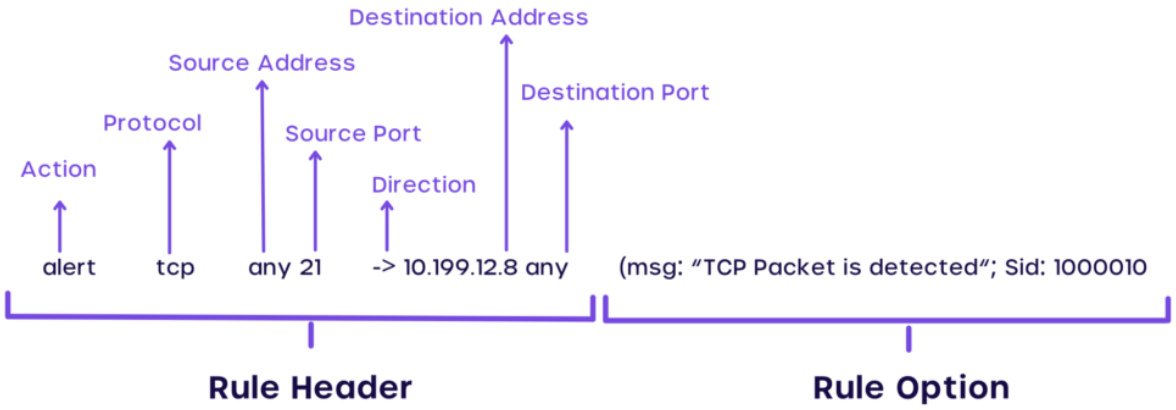


Figura 4.6 Sintaxis básica de las reglas de Snort. Fuente [50]

Una regla de manera general se dividen en dos apartados, el encabezado (header) y las opciones (option), el primer apartado dice que acción tomar en caso de que el paquete que se esté analizando, coincida con la regla definida, luego que protocolo de red se está analizando, la dirección IP de donde proviene el paquete, el puerto destino, la dirección IP destino y el puerto destino, luego de ello sigue el segundo apartado en donde el administrador escribe un mensaje que identifique al paquete en caso de que se active la regla, así como un identificador. Para las reglas personalizadas, los identificadores de red, pueden comenzar a partir de los dos millones.

A continuación, se presenta la Tabla 4.3, que expone las 14 reglas creadas para este trabajo de investigación, para efectos prácticos, en esta tabla no se explican en detalle, solo se expresan de manera general, y como fueron escritas. Mas adelante se expone una tabla con todo el detalle de cada una de estas reglas.

Tabla 4.3 Reglas personalizadas de Snort. Fuente: Elaboración propia

Reglas personalizadas de Snort
`alert tcp any any -> $HOME_NET any (msg:"ARP Poisoning Detected"; content:"ff:ff:ff:ff:ff:ff"; nocase; sid:2000001;)`
`# Inyeccion de comandos en solicitudes Web` `alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:"Inyección de comandos detectada"; content:"|3B|cmd|3D|"; http_uri; sid:2000003;)`

```
# Deteccion de fuerza bruta en SSH
alert tcp $EXTERNAL_NET any -> $HOME_NET 22 (msg:"Múltiples intentos de login SSH fallidos, posible ataque de fuerza bruta"; flags:A+; threshold:type threshold, track by_src, count 10, seconds 60; sid:2000004;)
# Escaneo de puertos SYN
alert tcp any any -> $HOME_NET any (flags:S; msg:"Escaneo de puertos SYN detectado"; flow:stateless; threshold:type threshold, track by_src, count 5, seconds 2; sid:2000005;)
# Regla para detectar escaneos de puertos con Nmap
alert tcp any any -> $HOME_NET any (msg:"Escaneo de puertos Nmap detectado"; flags:S; threshold:type threshold, track by_src, count 5, seconds 2; sid:2000006;)
# Reglas para detectar ataques DoS/DDoS
alert tcp any any -> $HOME_NET any (msg:"Posible ataque DoS"; flags:S; flow:stateless; threshold:type threshold, track by_dst, count 100, seconds 1; sid:2000008;)
alert icmp any any -> $HOME_NET any (msg:"Posible ataque ICMP Flood"; itype:8; threshold:type threshold, track by_dst, count 100, seconds 1; sid:2000009;)
# Regla para detectar OS Fingerprinting con ICMP
alert icmp any any -> $HOME_NET any (msg:"OS Fingerprinting con ICMP detectado"; itype:8; icode:0; sid:2000010;)
# Detección de agotamiento de direcciones IP en DHCP (DHCP Starvation)
alert udp any any -> any 67 (msg:"Posible ataque de agotamiento DHCP detectado"; content:"DHCP Discover"; sid:2000011;)
# Detección de servidor DHCP falso
alert udp any any -> $HOME_NET 68 (msg:"Respuesta DHCP desde servidor no autorizado"; content:"DHCP Offer"; sid:2000012;)
# Posibles conexiones exitosas a servicio SSH
alert tcp any any -> $HOME_NET 22 (msg:"Posible inicio de sesión SSH exitoso"; flags:PA; sid:2000013;)
# Posibles solicitudes legitimas a servidor HTTP
alert tcp any any -> $HOME_NET 80 (msg:"Solicitud HTTP detectada"; content:"GET"; http_method; sid:2000014;)
alert tcp any any -> $HOME_NET 80 (msg:"Solicitud HTTP detectada"; content:"POST"; http_method; sid:2000015;)
# Posible conexion exitosa a servidor MySQL
alert tcp any any -> $HOME_NET 3306 (msg:"Conexión MySQL detectada"; flags:S; sid:2000016;)
```

Estas reglas se crearon en el directorio ***etc/snort/rules/reglas_custom.rules***, con el objetivo de que el sistema Snort, hiciera uso de las mismas. En la Tabla 4.4, se describe con más detalle cada una de las reglas. Se podrá notar que no existe las reglas con identificador 2000002 y 2000007, dado que cuando se configuro el archivo de reglas

personalizadas, no se detectó ese error, el cual no tiene ningún impacto en el marco experimental.

Tabla 4.4 Descripción de las reglas personalizandas. Fuente: Elaboración propia

ID Regla	Descripción	Detalles
2000001	Posible ataque ARP Poisoning	Alerta cuando se detecta tráfico TCP desde cualquier origen y puerto hacia el host en cualquier puerto. Se busca el contenido “ff:ff:ff:ff:ff:ff” en los paquetes.
2000003	Inyección de comandos en solicitudes web	Alerta en caso de tráfico TCP desde la red externa hacia la red local en puertos HTTP.
2000004	Detección de fuerza bruta en servicio SSH	Alerta cuando se detectan múltiples intentos de inicio de sesión SSH fallidos desde la red externa hacia la red local. Utiliza la bandera “A+” que indica un intento de autenticación.
2000005	Escaneo de puertos SYN	Alerta de tráfico TCP desde cualquier origen y puerto hacia la red local con la bandera SYN. Busca identificar escaneos de puerto SYN mediante el uso de paquetes staless y aplicando un umbral.
2000006	Regla para detectar escaneos de puertos Nmap	Similar a la regla anterior, pero específicamente diseñada para detectar escaneos de puertos utilizando la herramienta Nmap
2000008	Regla para detectar ataques DoS y DDoS en protocolo TCP	Alerta en caso de tráfico TCP desde cualquier origen y puerto hacia la red local con la bandera SYN, indicando un posible ataque de denegación de servicio DoS, basado en la cantidad de paquetes en un intervalo de tiempo.
2000009	Regla para detectar ataques DoS y DDoS en protocolo ICMP	Alerta en caso de tráfico ICMP desde cualquier origen y puerto hacia la red local, con tipo de ICMP con peticiones ECHO_REQUEST, indicando un posible ataque de inundación ICMP. Aplica un umbral para detectar posibles ataques basados en la cantidad de paquetes en un intervalo de tiempo.
20000010	Reglas para detectar herramientas OS	Alerta en caso de tráfico ICMP desde cualquier origen y puerto hacia la red local, con tipo de ICMP. Busca detectar intentos de fingerprinting del sistema operativo mediante ICMP.
2000011	Detección de agotamiento de direcciones IPO en DHCP	Alerta en caso de tráfico UDP desde cualquier origen y puerto hacia el puerto 67 (DHCP Server) con el contenido “DHCP Discover”, indicando un posible ataque de agotamiento DHCP.

2000012	Detección de servidor DHCP falso	Alerta en caso de tráfico UDP desde cualquier origen y puerto hacia la red local en le puerto 68 (DHCP Client), indicando una posible respuesta DHCP desde un servidor no autorizado
2000013	Posibles conexiones exitosas a servicio SSH	Alerta en el caso de tráfico TCP desde cualquier origen hacia la red local en el puerto 22 (SSH) con la bandera PA indicando una conexión establecida.
2000014	Posibles solicitudes legitimas a servidor HTTP con el método GET	Alerta en caso de tráfico TCP desde cualquier origen y puerto hacia el puerto 80 HTTP con la cadena GET. Indicando posibles solicitudes legitimas al servidor HTTP
2000015	Posibles solicitudes legitimas a servidor HTTP con el método POST	Alerta en caso de tráfico TCP desde cualquier origen y puerto hacia el puerto 80 HTTP con la cadena POST. Indicando posibles solicitudes legitimas al servidor HTTP
2000016	Posible conexión exitosa a servidor MySQL	Alerta en caso de tráfico TCP desde cualquier origen y puerto hacia el puerto 3306 con la bandera SYN, indicando posibles conexiones exitosas al servidor MySQL

La configuración completa del sistema Snort se puede revisar en el apartado de anexos.

4.2.3 Configuración se servicios para tráfico de red normal

Los servicios básicos para que el servidor simule tráfico de red normal, fueron: MySQL, para base de datos, Apache para servidor HTTP, y Servidor SSH, para conexiones remotas.

En principio, se realizó la instalación de estos tres servicios, en la maquina Ubuntu, todo mediante el sistema de paquete APT. Se realizaron configuraciones básicas, como crear una base de datos de prueba, implementar una página web en el servidor HTTP, configurar el servicio SSH.

Con el propósito de que sea lo más automática posible la simulación, se crearon scripts en el lenguaje de programación Python, para simular el tráfico de red normal.

El primer script que se presenta simula la conexión a la base de datos por un tiempo aleatorio, luego se desconecta, espera 5 segundos, y vuelve a conectarse por un tiempo

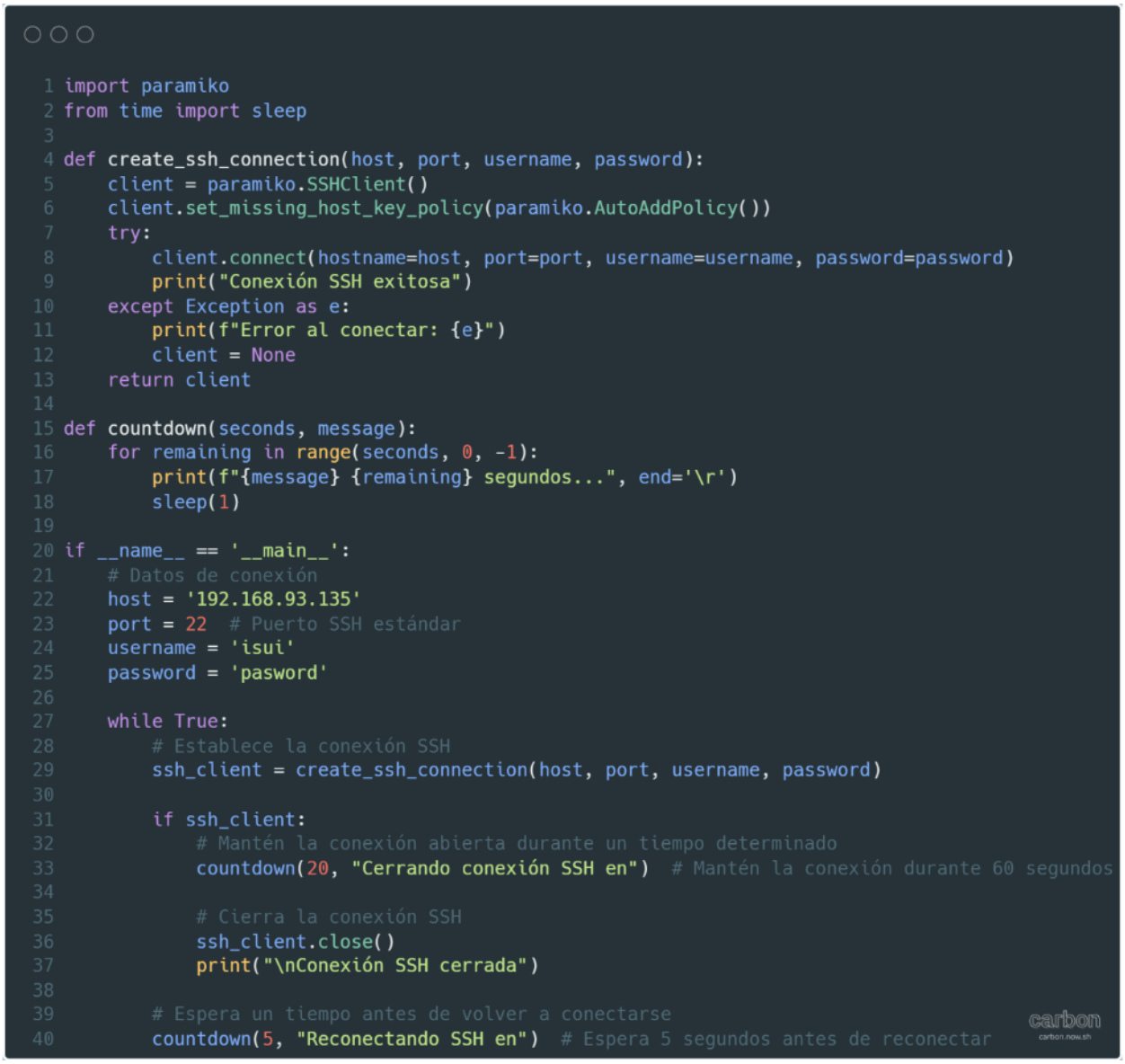
aleatorio nuevamente, para repetir el proceso. El script que simula este proceso se muestra en la Figura 4.7.

```python
import mysql.connector
from time import sleep

def connect_to_database(host, port, username, password, database):
    connection = None
    try:
        connection = mysql.connector.connect(
            host=host,
            port=port,
            user=username,
            passwd=password,
            db=database
        )
        print("Conexión exitosa")
    except mysql.connector.Error as error:
        print(f"No se pudo establecer la conexión: {error}")
    finally:
        return connection

def countdown(seconds, message):
    for remaining in range(seconds, 0, -1):
        print(f"{message} {remaining} segundos...", end='\r')
        sleep(1)

if __name__ == '__main__':
    # Datos de conexión directamente en el código
    host = '192.168.93.135'
    port = 3306
    username = 'isui'
    password = 'pasword'
    database = 'DataBase'

    print("\nIntentando conectarse a la base de datos...\n")

    while True:
        connection = connect_to_database(host, port, username, password, database)
        if connection:
            try:
                countdown(10, "Cerrando conexión en")  # Mantener la conexión abierta durante 30 segundos
            finally:
                connection.close()
                print("\nConexión cerrada")
        countdown(5, "Reconectando en")  # Espera cinco segundos antes de intentar conectarse nuevamente
```

Figura 4.7 Script en Python que automatiza la conexion a una base de datos. Fuente: Elaboración propia

El siguiente servicio instalado, para la simulación de ataques y tráfico normal, fue un servidor SSH. Las siglas SSH, hace referencia al servicio Secure Shell, un protocolo utilizando para la conexión remota cifrada de un un sistema, a otro. Es común que este protocolo sea utilizado para conexiones remotas, pero también es útil, para transferencia de archivos.

El script en Python que simula una conexión exitosa a un servicio SSH, se ilustra en la Figura 4.8. Este script conecta por un tiempo aleatorio al cliente con el servidor, cierra la conexión, espera 5 segundos, y vuelve a crear una conexión exitosa.



```python
import paramiko
from time import sleep

def create_ssh_connection(host, port, username, password):
    client = paramiko.SSHClient()
    client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    try:
        client.connect(hostname=host, port=port, username=username, password=password)
        print("Conexión SSH exitosa")
    except Exception as e:
        print(f"Error al conectar: {e}")
        client = None
    return client

def countdown(seconds, message):
    for remaining in range(seconds, 0, -1):
        print(f"{message} {remaining} segundos...", end='\r')
        sleep(1)

if __name__ == '__main__':
    # Datos de conexión
    host = '192.168.93.135'
    port = 22  # Puerto SSH estándar
    username = 'isui'
    password = 'pasword'

    while True:
        # Establece la conexión SSH
        ssh_client = create_ssh_connection(host, port, username, password)

        if ssh_client:
            # Mantén la conexión abierta durante un tiempo determinado
            countdown(20, "Cerrando conexión SSH en")  # Mantén la conexión durante 60 segundos

            # Cierra la conexión SSH
            ssh_client.close()
            print("\nConexión SSH cerrada")

        # Espera un tiempo antes de volver a conectarse
        countdown(5, "Reconectando SSH en")  # Espera 5 segundos antes de reconectar
```

Figura 4.8 Script en Python que automatiza la conexion a un servicio SSH. Fuente: Elaboración propia

El script de la Figura 4.8. simuló tráfico normal, por medio del protocolo TCP, creando conexiones exitosas a un servicio SSH, generando registros normales en el IDS.

Una conexión, que simuló tráfico de red de normal, mediante el protocolo TCP, fue mediante peticiones REQUEST a un servicio HTTP, por el puerto 80. Se desarrolló un

script en Python que simula peticiones GET en un tiempo aleatorio, simulando a usuarios. El script que simula el tráfico HTTP normal se ilustra en la Figura 4.9.

```python
import requests
import time
import random

def make_request(url):
    try:
        response = requests.get(url)
        response.raise_for_status()
        print(f"Solicitud exitosa: {response.status_code}")
    except requests.exceptions.RequestException as err:
        print(f"Fallo la solicitud: {err}")

def countdown(interval):
    for remaining in range(interval, 0, -1):
        print(f"Tiempo restante: {remaining} segundos", end='\r')
        time.sleep(1)

if __name__ == '__main__':
    url = 'http://192.168.93.135'

    while True:
        make_request(url)
        interval = random.randint(5, 60)  # Genera un número aleatorio entre 5 y 60 segundos
        countdown(interval)
```

Figura 4.9 Script en Python que simula peticiones GET a un servicio HTTP. Fuente: Elaboración propia

4.3 Simulación de ciberataques

La simulación de ciberataques, se realizó en un entorno controlado, en una red aislada. El propósito de realizar esta simulación, consistió en generar registros mediante el IDS Snort, para su posterior análisis.

Esta simulación constó de la ejecución de la herramienta NMAP, donde se realizó un escaneo activo, y el desarrollo de Scripts para simular ataques de fuerza bruta a un servicio SSH, reconocimiento OFP, así como la simulación de ataques DoS y DDoS.

La primera simulación, consistió en un reconocimiento activo, con la herramienta NMAP, haciendo uso de comandos avanzados. La ejecución se llevó a cabo desde las dos máquinas atacantes, en donde se destaca él envió de 500 paquetes por segundo. Se presenta una captura de pantalla, en donde se ejecuta la herramienta desde la máquina Kali Linux, como se ilustra en la Figura 4.10.

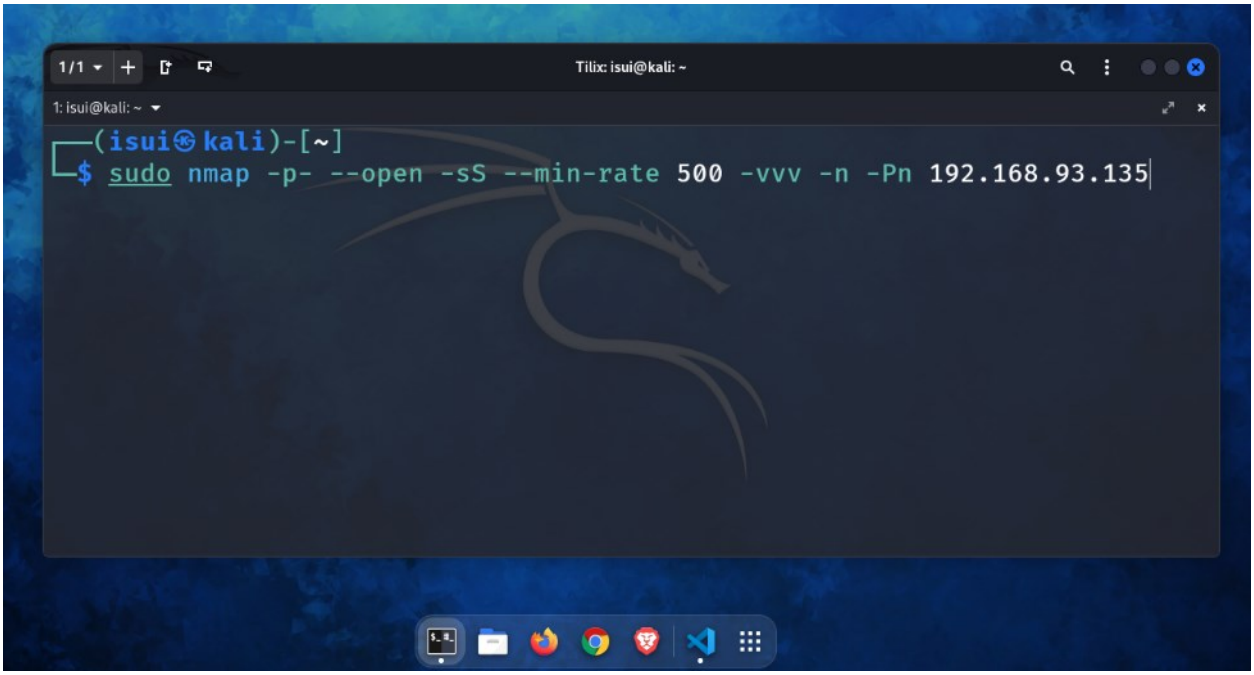


Figura 4.10 Ejecución de escaneo activo con Nmap desde Kali Linux. Fuente: Elaboración propia

La ejecución del comando en cuestión de la Figura 4.10, se explica en la Tabla 4.5. para mayor compresión de los parámetros utilizados.

Tabla 4.5 Expliación de parametros de la ejeción de la herramienta NMAP. Fuente: Elaboración propia

Comando	Explicación
sudo	Comando para ejecutar la herramienta con privilegios de administrador
nmap	Comando para ejecutar la herramienta Nmap
-p-	Parámetro para escanear todos los puertos existentes, desde el 1 hasta el 65535
--open	Parámetro para mostrar solo los puertos abiertos
-sS	Parámetro que especifica el tipo de escaneo, en donde es de tipo SYN, que envía paquetes SYN para determinar la respuesta del sistema
--min-rate 500	Parámetro que establece la cantidad de paquetes que se debe de enviar por segundo, en este caso 500 paquetes por segundo
-vvv	Parámetro para mostrar el detalle de la ejecución en terminal

-n	Parámetro que indica a Nmap que no realice resolución DNS durante el escaneo
-Pn	Parámetro que indica a Nmap que el HOST objetivo, está en línea e ignore las peticiones ECHO_REPLY
192.168.92.135	Dirección IP del HOST objetivo

Este escaneo fue detectado mediante las reglas 2000005 y 2000006 de la Tabla 4.4 que representan un posible escaneo con la herramienta NMAP.

El siguiente tipo de simulación de ciberataque, fue el desarrollo de una herramienta que simula el reconocimiento del sistema mediante técnicas de Os Finger Printing, este script fue desarrollado con el lenguaje de programación Python, el cual se ilustra en la Figura 3.25. Este script se ejecutó múltiples veces desde las maquinas atacantes, el cual fue registrado por el IDS, mediante la regla 20000010 de la tabla Tabla 4.4.

El siguiente ataque que fue simulado, fue un ataque de fuerza bruta, para intentar autenticar el servicio SSH. Tanto en Kali Linux como con Parrot, tienen ya herramientas existentes, para llevar a cabo ataques de fuerza bruta como por ejemplo con Hydra, GoBuster, entre otros.

Para el caso de esta experimentación, se desarrolló un script en Python, que simula un ataque de fuerza bruta a un servicio SSH, en donde además de desarrollar el script, se desarrolló un diccionario con distintas posibles credenciales. Este script se puede ver en la Figura 4.11.

```python
import paramiko
import time

def connect_ssh(host, port, username, password):
    max_retries = 3
    for attempt in range(max_retries):
        try:
            ssh_client = paramiko.SSHClient()
            ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
            ssh_client.connect(hostname=host, port=port, username=username, password=password)
            print(f"Conexión exitosa a {host} con usuario {username}")
            return ssh_client
        except paramiko.AuthenticationException:
            print(f"Autenticación fallida para {username} en {host}")
            return None
        except paramiko.SSHException as e:
            if "Error reading SSH protocol banner" in str(e) and attempt < max_retries - 1:
                print(f"Reintentando conexión a {host} con usuario {username} (intento {attempt + 1})")
                time.sleep(5)  # Esperar 5 segundos antes de reintentar
            else:
                print(f"Error al conectar a {host} con usuario {username}: {e}")
                return None
        except Exception as e:
            print(f"Error al conectar a {host} con usuario {username}: {e}")
            return None

# Configuración del servidor SSH
host = "192.168.93.135"
port = 22  # Puerto SSH, por defecto es el 22

# Leer las credenciales desde un archivo
ruta_archivo_credenciales = "/home/isui/tesis/programas/credenciales.txt"
ssh_client = None

with open(ruta_archivo_credenciales, "r") as file:
    for line in file:
        username, password = line.strip().split()  # Asegúrate de que el archivo tiene el formato correcto
        ssh_client = connect_ssh(host, port, username, password)
        if ssh_client:
            break  # Salir del bucle si la conexión es exitosa

# No olvides cerrar la conexión cuando termines
if ssh_client:
    ssh_client.close()
```

Figura 4.11 Script en Python que simula un ataque de fuerza bruta a un servicio SSH. Fuente: Elaboración propia

Esta simulación activó la regla 2000004, de la tabla Tabla 4.4, que registró el IDS, esta regla alerta sobre un posible ataque de fuerza bruta, mediante múltiples intentos de inicio de sesión del servicio SSH.

El siguiente ataque consistió en simular ataques de tipo DoS, desde ambos equipos atacantes, además de simulando distintas direcciones IP, para la simulación del ataque DDoS. El script que simula este ataque se encuentra en la Figura 4.12.

```python
import requests
import threading
import time
import random

def attack(url, run_event):
    while run_event.is_set():
        try:
            response = requests.get(f"http://{url}")
            print("Status Code: ", response.status_code)
        except Exception as e:
            print(e)

if __name__ == '__main__':
    target_url = input("Introduce la URL del servidor HTTP: ")
    num_threads = int(input("Introduce el número de hilos para realizar el ataque: "))

    run_event = threading.Event()
    run_event.set()

    threads = []
    for _ in range(num_threads):
        t = threading.Thread(target=attack, args=(target_url, run_event))
        t.start()
        threads.append(t)

    try:
        while True:
            # Ejecuta durante un tiempo aleatorio entre 15 segundos y 1 minuto
            time.sleep(random.randint(15, 60))
            print("\nPausando el ataque...\n")
            run_event.clear()  # Indica a los hilos que se detengan

            # Espera 15 segundos antes de reiniciar
            time.sleep(15)
            print("\nReiniciando el ataque...\n")
            run_event.set()  # Indica a los hilos que continúen

    except KeyboardInterrupt:
        print("\nDeteniendo todos los hilos...")
        run_event.clear()

    for t in threads:
        t.join()
```


Figura 4.12 Script en Python que simula un ataque DDoS a un servicio HTTP. Fuente: Elaboración propia

Esta simulación debería haber activado las reglas 2000008 y 2000009 de la Tabla 4.4 en donde se detectan múltiples peticiones al servicio HTTP en un intervalo de tiempo muy corto.

4.4 Entendimiento de los datos

Los registros generados, se guardaron en un directorio específico, con privilegios de administrador. Una ilustración grafica se puede presentar en la Figura 4.13.

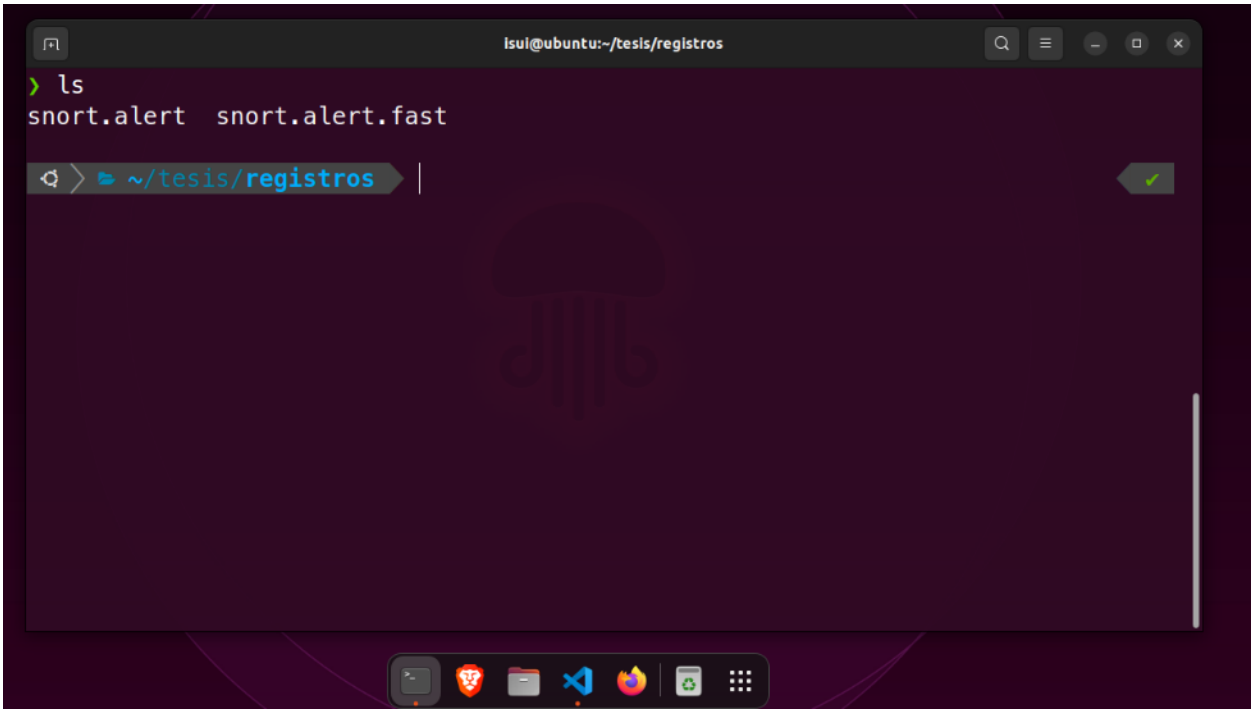


Figura 4.13 Directorio para guardar los registros generados por el IDS Snort. Fuente: Elaboración Propia

Los registros generados por el IDS Snort proporciona un archivo en formato tcpdump, este formato, incluye la información descrita en la Tabla 4.6.

Tabla 4.6 Expliación de los registros generados por el IDS Snort. Fuente: Elaboración propia

Campo	Explicación
Fecha y hora	La fecha y hora exactas en que se registró la entrada
ID de la regla	Identificador único de la regla que se activo
Descripción (etiqueta)	Descripción de la etiqueta asignada a la regla
Prioridad	Nivel de prioridad o gravedad asociado a la detección, en el caso de la presente investigación, este valor no será tomado en cuenta
Protocolo	El protocolo de red utilizado
Dirección origen	La dirección IP asociada al registro de la petición o solicitud origen

Puerto origen	El número de puerto asociado al registro de la petición o solicitud de origen
Dirección destino	La dirección IP del host con el servicio e IDS
Puerto destino	El número de puerto asociado al puerto del host

Un ejemplo de un registro se puede ver en la Figura 4.14.

Figura 4.14 Ejemplo de un registro generado por el IDS Snort en formato TCPDUM. Fuente: Elaboración propia

Inicialmente, se propuso la creación de un conjunto de datos compuesto por aproximadamente 200,000 registros con el objetivo de garantizar una muestra significativa.

Se procedió a realizar la simulación de todos los servicios previamente mencionados, así como la simulación de ataques, durante un intervalo de tiempo definido, con la finalidad de generar los 200,000 registros planeados. Como resultado de estas simulaciones, se obtuvo un total de 199,782 registros. Las alertas activadas por el Sistema de Detección de Intrusiones (IDS) Snort se detallan en la Tabla 4.7.

Tabla 4.7 Reglas que activaron la alerta del IDS y generaron registros, en la ejecución de la simulación. Fuente Elaboración propia

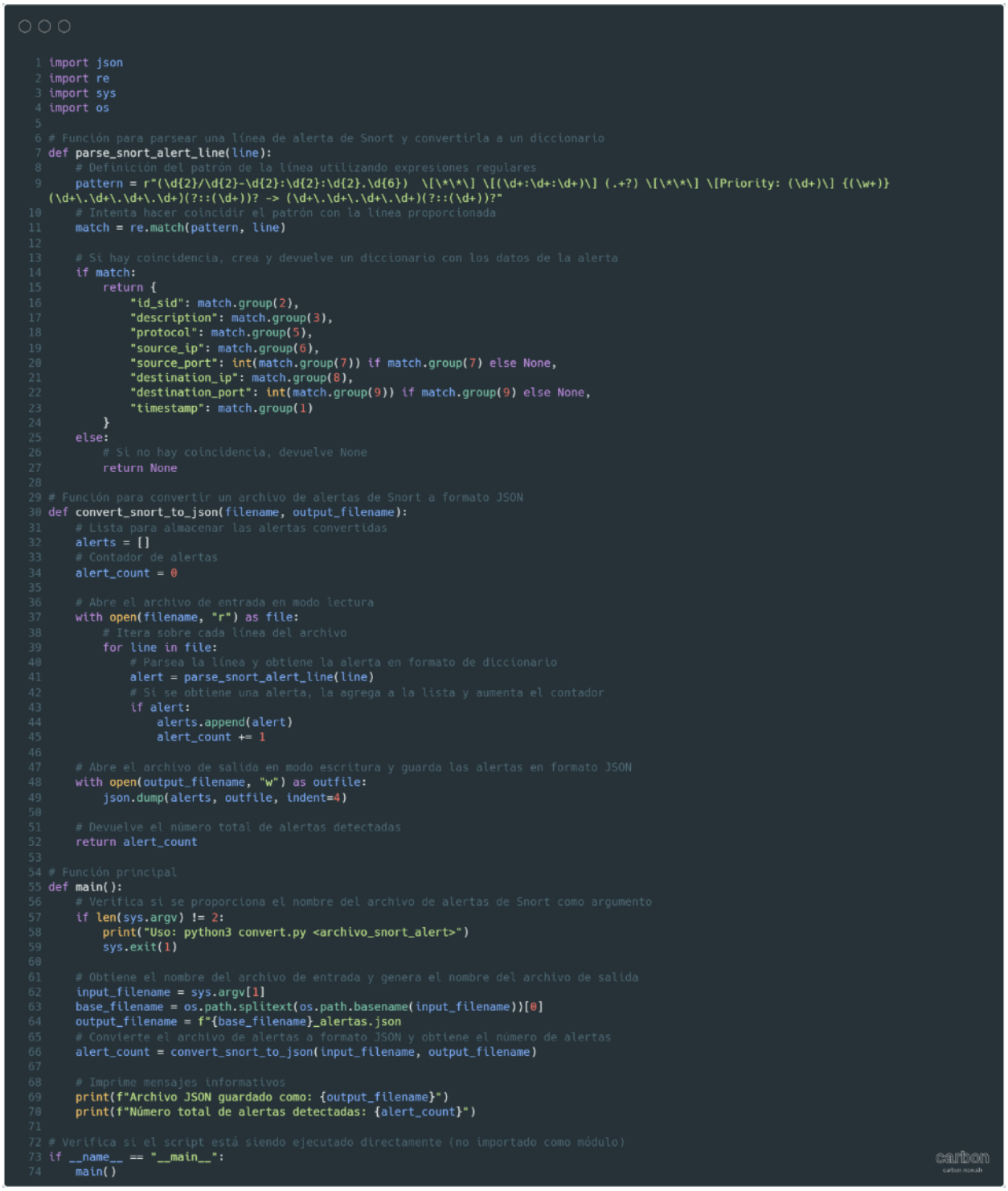
Regla activada	Explicación
2000004	Posible ataque de fuerza bruta a un servicio SSH
2000005	Posible escaneo de puertos SYN
2000006	Posible escaneo con herramienta NMAP
2000008	Posible ataque DoS y DDoS en a servicio HTTP

20000010	Posible detección de reconocimiento con escaneo OFP
2000013	Posibles conexiones exitosas a servicio SSH
2000014	Posibles solicitudes legitimas a servicio HTTP
2000016	Posibles conexiones exitosas a servidor MySQL

En una subsiguiente simulación, se generó un total de 53,804 registros, que se determinó sería el conjunto de datos de pruebas de ahora en adelante. Al ejecutar los análisis con Weka y Elasticsearch, se concluyó que la muestra óptima sería esta última con los 53,804 registros en base a las capacidades del software utilizado, así como de las capacidades de hardware, garantizando un manejo eficiente de los datos y evitando desbordamientos de memoria.

4.5 Preparación de los datos

Los registros producidos por el Sistema de Detección de Intrusiones (IDS) fueron almacenados en formato tcpdump. Posteriormente, se inició un proceso de tratamiento de datos, que implicó la creación de un script en Python diseñado para eliminar la columna de prioridad identificada en la Tabla 4.6. Como se indicó previamente, este campo no fue tomado en cuenta para los objetivos específicos de la investigación. La representación gráfica del script utilizado para llevar a cabo este proceso de tratamiento de datos se exhibe en la Figura 4.15.



```python
import json
import re
import sys
import os

# Función para parsear una línea de alerta de Snort y convertirla a un diccionario
def parse_snort_alert_line(line):
    # Definición del patrón de la línea utilizando expresiones regulares
    pattern = r"(\d{2}/\d{2}-\d{2}:\d{2}:\d{2}.\d{6})  \[\*\*\] \[(\d+:\d+:\d+)\] (.+?) \[\*\*\] \[Priority: (\d+)\] {(\w+)} (\d+\.\d+\.\d+\.\d+)(?::(\d+))? -> (\d+\.\d+\.\d+\.\d+)(?::(\d+))?"
    # Intenta hacer coincidir el patrón con la línea proporcionada
    match = re.match(pattern, line)

    # Si hay coincidencia, crea y devuelve un diccionario con los datos de la alerta
    if match:
        return {
            "id_sid": match.group(2),
            "description": match.group(3),
            "protocol": match.group(5),
            "source_ip": match.group(6),
            "source_port": int(match.group(7)) if match.group(7) else None,
            "destination_ip": match.group(8),
            "destination_port": int(match.group(9)) if match.group(9) else None,
            "timestamp": match.group(1)
        }
    else:
        # Si no hay coincidencia, devuelve None
        return None

# Función para convertir un archivo de alertas de Snort a formato JSON
def convert_snort_to_json(filename, output_filename):
    # Lista para almacenar las alertas convertidas
    alerts = []
    # Contador de alertas
    alert_count = 0

    # Abre el archivo de entrada en modo lectura
    with open(filename, "r") as file:
        # Itera sobre cada línea del archivo
        for line in file:
            # Parsea la línea y obtiene la alerta en formato de diccionario
            alert = parse_snort_alert_line(line)
            # Si se obtiene una alerta, la agrega a la lista y aumenta el contador
            if alert:
                alerts.append(alert)
                alert_count += 1

    # Abre el archivo de salida en modo escritura y guarda las alertas en formato JSON
    with open(output_filename, "w") as outfile:
        json.dump(alerts, outfile, indent=4)

    # Devuelve el número total de alertas detectadas
    return alert_count

# Función principal
def main():
    # Verifica si se proporciona el nombre del archivo de alertas de Snort como argumento
    if len(sys.argv) != 2:
        print("Uso: python3 convert.py <archivo_snort_alert>")
        sys.exit(1)

    # Obtiene el nombre del archivo de entrada y genera el nombre del archivo de salida
    input_filename = sys.argv[1]
    base_filename = os.path.splitext(os.path.basename(input_filename))[0]
    output_filename = f"{base_filename}_alertas.json
    # Convierte el archivo de alertas a formato JSON y obtiene el número de alertas
    alert_count = convert_snort_to_json(input_filename, output_filename)

    # Imprime mensajes informativos
    print(f"Archivo JSON guardado como: {output_filename}")
    print(f"Número total de alertas detectadas: {alert_count}")

# Verifica si el script está siendo ejecutado directamente (no importado como módulo)
if __name__ == "__main__":
    main()
```

Figura 4.15 Script en Python, El principal para realizar el tratamiento de los registros generados con el IDS Snort. Fuente: Elaboración propia

Este script no solo elimina el campo mencionado, sino que también transforma el archivo generado por el Sistema de Detección de Intrusiones (IDS) a formato JSON. Esta

conversión facilita su manipulación posterior mediante diversas herramientas y, además, se encarga de generar etiquetas en idioma español para mejorar la comprensión. De esta manera, el ejemplo presentado en la Figura 4.14 experimenta una transformación significativa hacia un nuevo formato en la salida de datos tratados, ahora representado en la Figura 4.16.

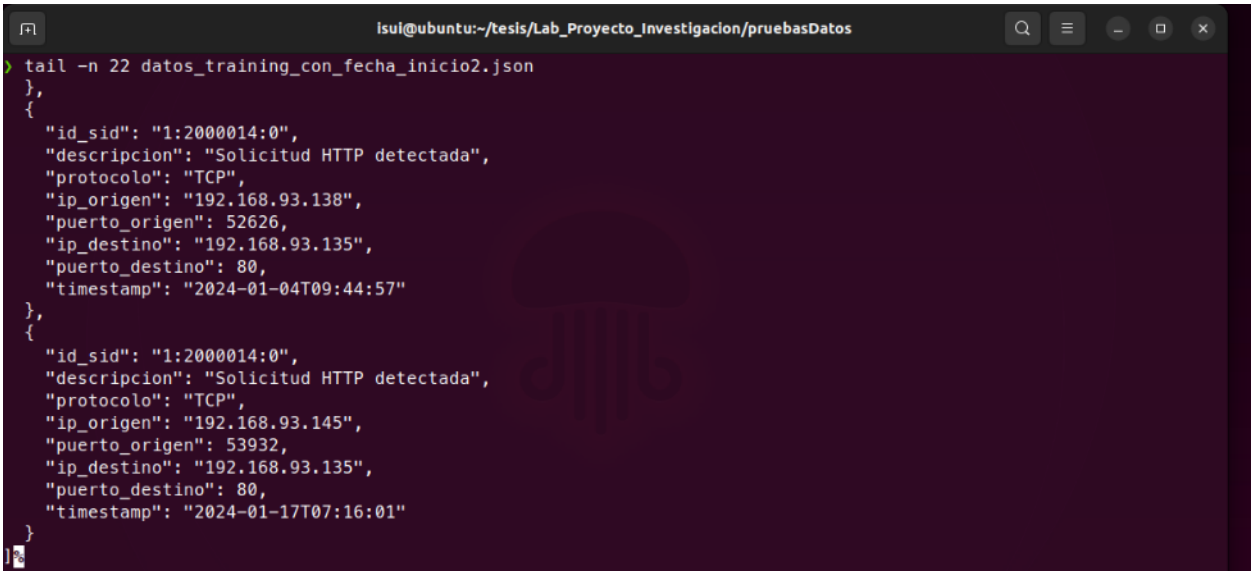


Figura 4.16 Ejemplo de conjunto de datos generados y tratados en formato JSON, desde terminal. Fuente: Elaboración propia

El siguiente paso fue convertir los archivos JSON a formato CSV. Principalmente se planteó ejecutar todos los análisis con los archivos en formato JSON, tanto en Elasticsearch como en Weka, pero en la ejecución de las pruebas, ninguna de las dos aplicaciones fue capaz de procesar dicho formato.

4.6 Clasificadores

Inicialmente, los clasificadores propuestos fueron los identificados en la sección 3.4.4 “Algoritmos de machine Learning”, de este documento. Sin embargo, durante la fase de experimentación, se concluyó que no todos eran aplicables debido a las características particulares del conjunto de datos analizado. Los clasificadores seleccionados para la aplicación en Weka fueron principalmente árboles de decisión, considerando las particularidades del conjunto de datos para realizar tareas de clasificación y predicción.

A continuación, se enumeran los clasificadores utilizados, respaldados por una captura de pantalla que ilustra la interfaz de la aplicación, como se muestra en la Figura 4.17:

- NaiveBayes
- J48
- RandomTree
- RandomForest
- MultilayerPerceptron
- IBK

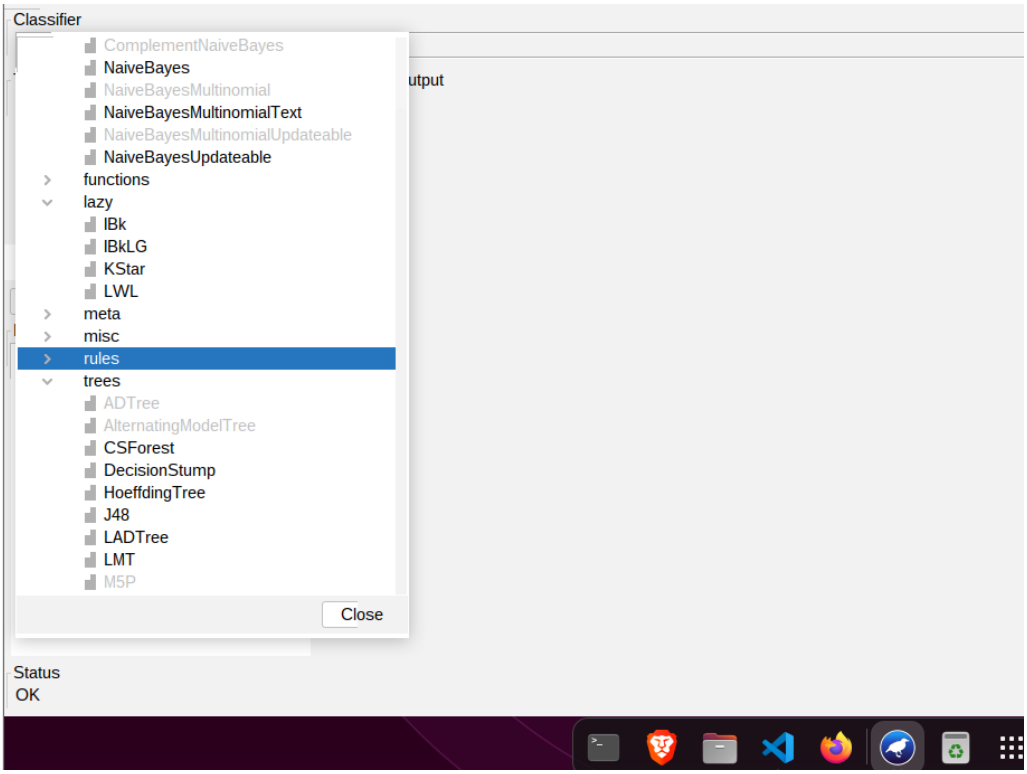


Figura 4.17 Captura de pantalla de los posibles clasificadores en Weka. Fuente: Elaboración propia

La elección de estos clasificadores se fundamentó en su capacidad demostrada para abordar la naturaleza del conjunto de datos, asegurando así un enfoque efectivo y adecuado para las tareas de clasificación necesarias. La captura de pantalla proporciona una visión visual de la interfaz utilizada en Weka para configurar y ejecutar estos clasificadores, facilitando la comprensión de la metodología empleada en el proceso de experimentación y evaluación.

Por otro lado, la aplicación de Elasticsearch ha brindado herramientas esenciales para la identificación y revisión de anomalías, centrándose especialmente en la marca temporal de las solicitudes al servidor. Este enfoque permite una evaluación detallada de los eventos en función del tiempo, proporcionando valiosa información para detectar patrones inusuales o actividades sospechosas en la red.

La aplicación también ha proporcionado resultados estadísticos significativos, los cuales serán objeto de análisis detallado en el próximo capítulo. Estos resultados estadísticos arrojarán luz sobre tendencias, comportamientos atípicos y otros aspectos cruciales que contribuirán a una comprensión más profunda de la seguridad del sistema.

Es esencial destacar que la comparación directa entre ambas aplicaciones se ha visto limitada por la necesidad de contar con la versión empresarial de Elasticsearch para personalizar clasificadores. Esta limitación ha impedido una evaluación completamente equitativa de las capacidades de personalización y adaptabilidad de ambos sistemas. En este sentido, se ha incluido una captura de pantalla ilustrativa en la Figura 4.18, que proporciona una visión visual de la interfaz y las opciones disponibles en la versión utilizada.

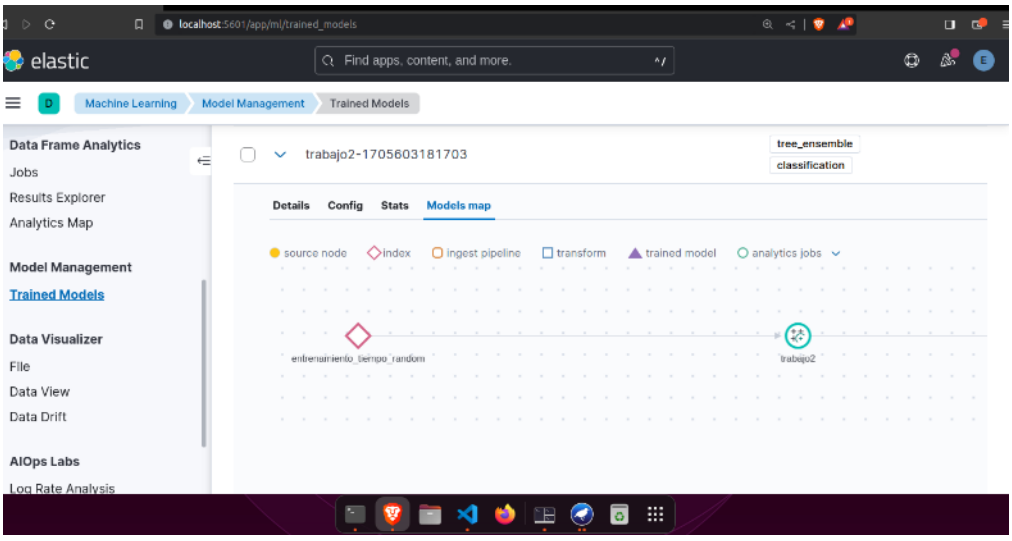


Figura 4.18 Apartado de Machine Learning en Elasticsearch. Fuente: Elaboración propia

4.7 Red neuronal con Tensor Flow

A continuación, se presentan dos redes neuronales propuestas con Tensor Flow, la primera es una red neuronal densa vista en el apartado 3.3.2. “Redes neuronales” Para

practicidad y facilidad de entendimiento, se presenta la tabla que explica los parámetros de la primera red neuronal.

Tabla 4.8 Parámetros para la red neurona densa con Tensor Flow. Fuente: Elaboración propia

Parámetro	**Valor**	**Detalles**
Capas de la red Neuronal	Embedding, Flatten, Dense	La red consta de tres capas
Función de activación	ReLU en la primera capa Dense, Softmax en la última capa	Se utiliza ReLU en la capa oculta para introducir no linealidades en la red y Softmax en la capa de salida para obtener las probabilidades de pertenencia a cada clase.
Tamaño de Embedding	50	La capa de embedding convierte las entradas dispersas en vectores densos de tamaño 50.
Tamaño de capa oculta	64	La capa oculta de la red neuronal tiene 64 unidades.
Función de perdida	Sparce categorical cross-entropy	Se utiliza para problemas de clasificación con múltiples clases, donde las etiquetas son enteros.
Optimizador	Adam	Algoritmo de optimización popular que combina el concepto de momentum y RMSProp, adecuado para una amplia gama de problemas de optimización.
Épocas	5	Cada ciclo de corrección de propagación hacia atrás y hacia adelante para reducir la pérdida.
Tamaño de Lote	32	El tamaño del lote para el entrenamiento del modelo. Un tamaño de lote más grande puede acelerar el entrenamiento, pero puede requerir más memoria.
Evaluación del rendimiento	Accuracy	Se evalúa la exactitud del modelo en el conjunto de validación para medir su rendimiento.

De manera formal se puede expresar la red neuronal de la siguiente manera:

Dada una entrada ***x*** que representa un documento de texto, y asumiendo que se está utilizando una función de activación ***softmax*** en la capa de salida para la clasificación multiclase, la salida de la red neuronal se calcularía en (21) como:

$$\text{softmax}(W_3 * \text{ReLu}(W_2 * \text{flatten}(E * I_x + b_1) + b_2) + b_3) \quad (19)$$

Donde:

- **E** es la matriz de incrustación (Embedding) que mapea palabras a vectores de características densas.
- $\mathbf{I_x}$ es una matriz one-hot encoding que representa el documento de entrada **x.**
- $\mathbf{W_1}$, $\mathbf{W_2}$, y $\mathbf{W_3}$ son las matrices de pesos de capas densas
- $\mathbf{b_1}$, $\mathbf{b_2}$, y $\mathbf{b_3}$ son los sesgos de las capas densas.
- **Flatten(*)** es la operación de aplanado que convierte los resultados de la capa de incrustación en un vector unidimensional.
- **ReLu(*)** es la función de activación rectificada aplicada a la segunda capa densa.
- **Softmax(*)** es la función de activación softmax aplicada en la capa de salida para obtener la probabilidades de pertenencia a cada clase.

A continuación, en la Tabla 4.9 se presenta los parámetros para la red neuronal recurrente, para el análisis del conjunto de datos:

Tabla 4.9 Parámetros para la red neurona recurrente con Tensor Flow. Fuente: Elaboración propia

Parámetro	**Valor**	**Detalles**
Capas de la red Neuronal	Embedding, SimpleRNN, Dense	La red consta de tres capas
Función de activación	ReLU en la primera capa SimpleRNN, Softmax en Dense	Se utiliza ReLU en la capa oculta para introducir no linealidades en la red y Softmax en la capa de salida para obtener las probabilidades de pertenencia a cada clase.
Tamaño de Embedding	50	La capa de embedding convierte las entradas dispersas en vectores densos de tamaño 50.
Tamaño de capa oculta	64	La capa RNN de la red neuronal tiene 64 unidades.

Función de perdida	Sparce categorical cross-entropy	Se utiliza para problemas de clasificación con múltiples clases, donde las etiquetas son enteros.
Optimizador	Adam	Algoritmo de optimización popular que combina el concepto de momentum y RMSProp, adecuado para una amplia gama de problemas de optimización.
Épocas	5	Cada ciclo de corrección de propagación hacia atrás y hacia adelante para reducir la pérdida.
Tamaño de Lote	32	El tamaño del lote para el entrenamiento del modelo. Un tamaño de lote más grande puede acelerar el entrenamiento, pero puede requerir más memoria.
Evaluación del rendimiento	Accuracy	Se evalúa la exactitud del modelo en el conjunto de validación para medir su rendimiento.

La manera formal de representar la red neuronal es:

Dada una entrada ***x*** que representa un documento de texto, la salida de la red neurona con una capa SimpleRNN se calcularía en (22) como:

$$\text{softmax}(\text{W}_3 * \text{ReLu}(\text{W}_2 * h_t + \text{b}_2) + \text{b}_3) \quad (20)$$

Donde:

- $\mathbf{h_t}$ es la salida de la capa SimpleRNN en paso de tiempo ***t*** que se calcula en (23) como:

$$h_t = \tanh\left(W_1 * x_t + U * h_{t-1} + b_1\right) \quad (21)$$

  Donde $\mathbf{W_1}$ son los pesos que se aplican a la entrada $\mathbf{x_t}$, **U** son los pesos que aplican a la salida anterior $\boldsymbol{h_{t-1}}$ y $\mathbf{b_1}$ son los sesgos.
- $\mathbf{W_2}$, y $\mathbf{W_3}$ son las matrices de pesos de capas densas
- $\mathbf{b_2}$, y $\mathbf{b_3}$ son los sesgos de las capas densas.
- **ReLu(*)** es la función de activación rectificada aplicada a la segunda capa SimpleRNN.

- **Softmax(*)** es la función de activación softmax aplicada en la capa de salida para obtener la probabilidades de pertenencia a cada clase.

Al comparar ambas tablas, se puede observar que son muy similares, encontrando la diferencia en la primera capa.

CAPÍTULO 5 EVALUACIÓN

Este capítulo expone la evaluación de la fase experimental, en donde se presenta, distintas evaluaciones, como el rendimiento de hardware, anomalías en un intervalo de tiempo, estadísticas y clasificadores.

5.1 Evaluación del rendimiento del hardware en la simulación

La ejecución de las simulaciones, fue realizada en un laboratorio virtual, mediante un hipervisor, al equipo host, donde se virtualizo el laboratorio, se le exigió el suficiente poder de cómputo para llevar a cabo la ejecución de dicho laboratorio.

La descripción técnica del equipo host, se documentó en la Tabla 1.1, la figura que se expone es la carga de CPU, realizando simulación de los 3 equipos del laboratorio.

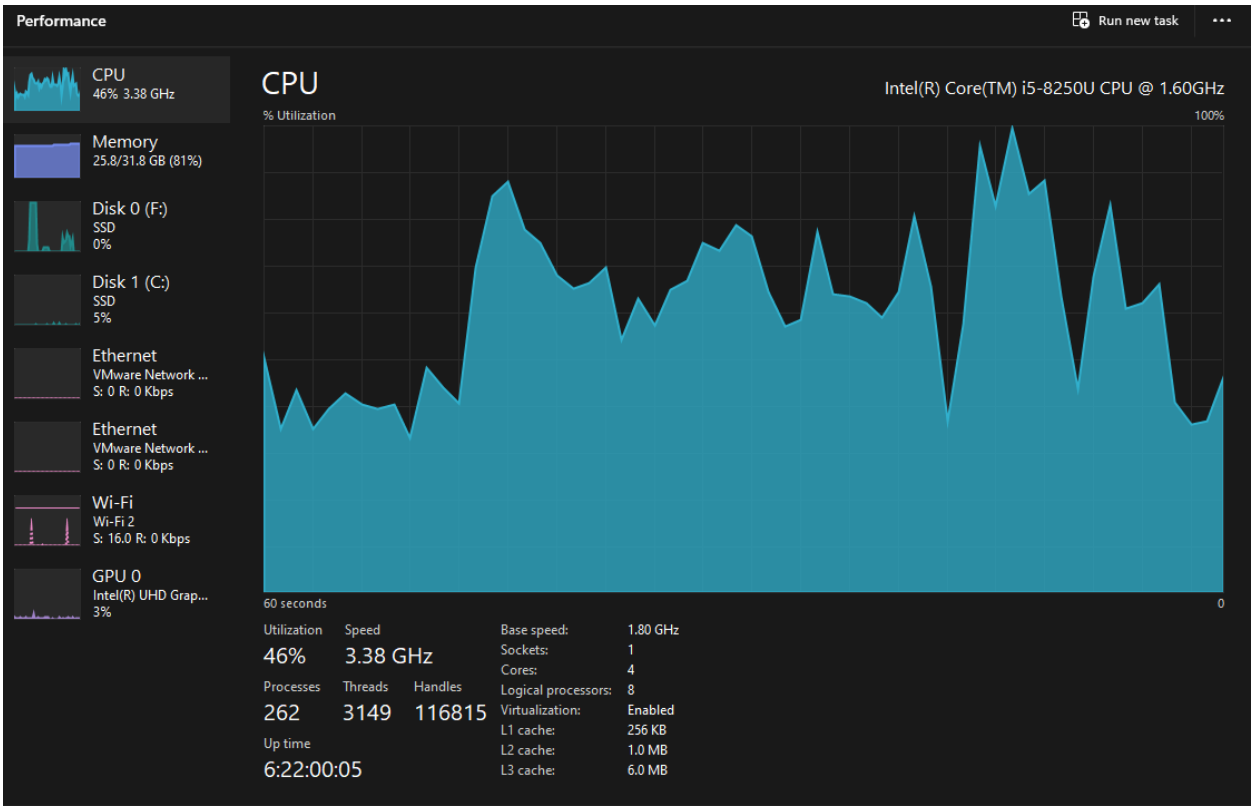


Figura 5.1 Evaluación de carga de trabajo del CPU host simulando los 3 equipos GUETS del laboratorio. Fuente: Elaboración propia

La Figura 5.1, muestra una carga de CPU alta, si bien no en el límite de su capacidad, muy cerca de llegar a este. El poder de cómputo del CPU, represento significativamente,

en la posible muestra que se quería obtener, siendo mucho menos a la principalmente planteada.

Luego se evaluó la capacidad de memoria RAM, del equipo, contando con una capacidad de 32 GB disponibles para el despliegue del laboratorio, la Figura 5.2, representa que esta capacidad se encontraba casi en su límite, al momento de ejecutar las simulaciones.

Figura 5.2 Evaluación de carga de trabajo de memoria RAM del host simulando los 3 equipos GUETS del laboratorio. Fuente: Elaboración propia

Estos son los dos principales componentes del Hardware, que limitaron la investigación, dada las circunstancias ya revisadas en el CAPÍTULO 1, en la sección de limitaciones.

5.2 Evaluación de anomalías con Elasticsearch

La siguiente evaluación, fue la detección de anomalías con la herramienta de Machine Learning de Elasticsearch, esta presentó distintos picos en la gráfica, que representan posibles ataques, en base a la cantidad de peticiones al servidor, en un intervalo de tiempo corto. Esta grafica se presenta en la Figura 5.3.

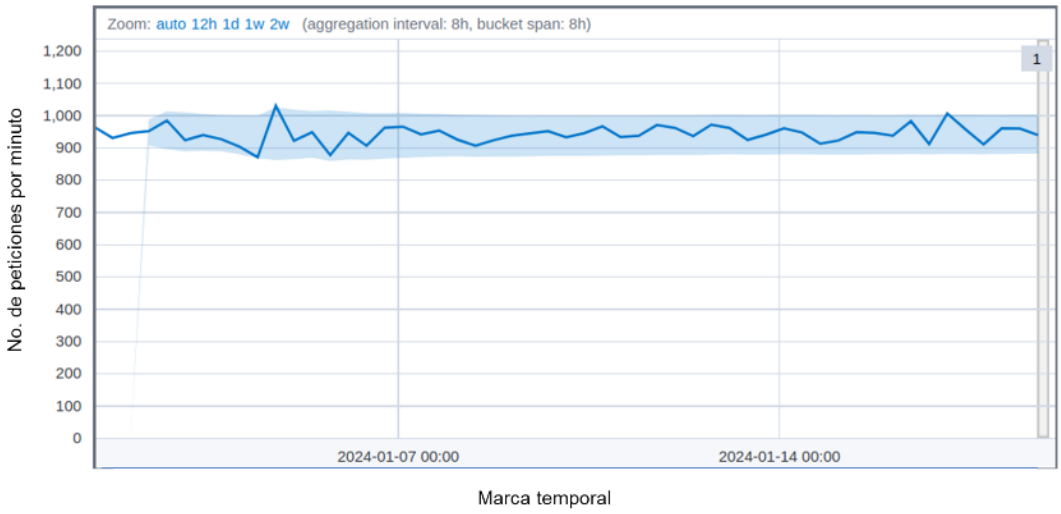


Figura 5.3 Detección de anomalias con Elasticsearch en un intervalo de tiempo. Fuente: Elaboración propia

Seguida de esta detección de anomalías, se utilizó, la herramienta de Machine Learning, para predecir un intervalo de tiempo, del posible tráfico en base al conocimiento obtenido previamente de los registros. Este se presenta en la Figura 5.4.

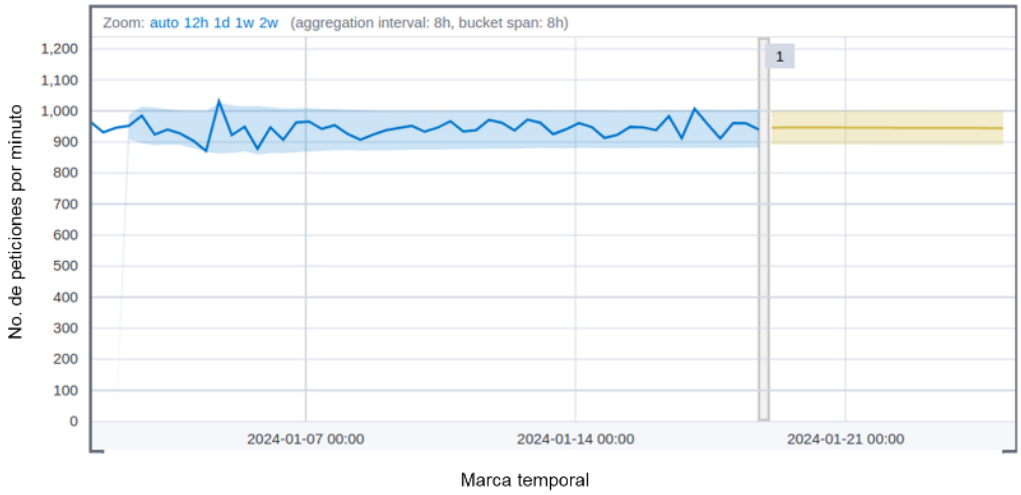


Figura 5.4 Predicción del posible comportamiento del trafico de red, en un proximo intervalo de tiempo. Fuente: Elaboración propia

Si bien esta predicción, puede sugerir que en las próximas horas el tráfico ser comportará de manera normal, es importante recordar que un actor malicioso, no trabaja en base a las estadísticas, sino más bien, en base a la sorpresa del objetivo, para que este no se encuentre preparado.

5.3 Clasificación con Elasticsearch

Los modelos básicos de machine Learning, de Elasticsearch, permitieron representar una matriz de confusión en donde se clasifico el atributo ID Regla. Esta se presenta en la Figura 5.5.

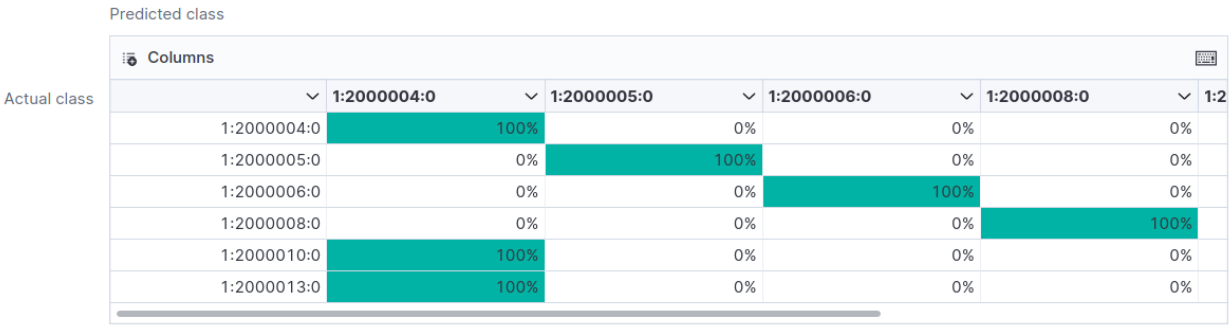


Figura 5.5 Matriz de confusión para el conjunto de datos. Fuente: Elaboración propia

La matriz de confusión multiclase resume el rendimiento del análisis de clasificación al mostrar la proporción de puntos de datos correctamente e incorrectamente clasificados. Las etiquetas reales están en el lado izquierdo, mientras que las etiquetas predichas están en la parte superior. La proporción de predicciones correctas e incorrectas se detalla para cada clase, permitiendo analizar las confusiones entre las clases durante las predicciones.

Con el aumento del número de clases, la complejidad de la matriz de confusión también aumenta. Las celdas más oscuras indican un mayor porcentaje de predicciones, proporcionando una visión más clara del rendimiento. Puede utilizar el selector de columnas para alternar entre mostrar u ocultar algunas o todas las columnas, facilitando la visualización de la información deseada.

Luego se presentó una evaluación de las métricas de calidad, presentadas en la Figura 5.6.

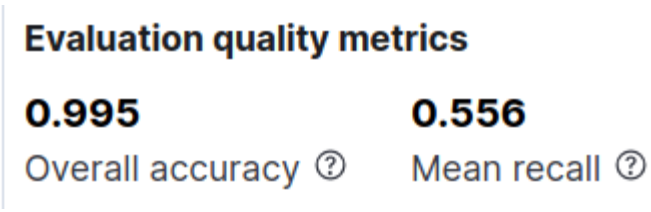


Figura 5.6 Metricas de calidad de la clasificación con Elasticsearch. Fuente: Elaboración propia

Estas métricas indican primeramente "Overall Accuracy" que se refiere a la precisión general de un modelo de clasificación y representa la proporción de instancias clasificadas correctamente sobre el total de instancias. Se calcula como el número total de predicciones correctas dividido por el número total de instancias en el conjunto de datos.

Por otro lado, "Mean Recall" (Recall Promedio) es una métrica de evaluación de modelos de clasificación que se centra en la capacidad del modelo para identificar todas las instancias positivas en el conjunto de datos. El Recall se calcula como el número de verdaderos positivos dividido por la suma de verdaderos positivos y falsos negativos.

Se expone en la Figura 5.1 la precisión de las clases.

Tabla 5.1 Recall y precesición por clase. Fuente: Elaboración propia

Class	Accuracy	Recall
1:2000004:0	0.995	1
1:2000005:0	1	1
1:2000006:0	1	1
1:2000008:0	1	1
1:2000010:0	1	0
1:2000013:0	0.995	0
1:2000014:0	1	1
1:2000015:0	1	0
1:2000016:0	0.999	0

Seguido de esto se presentó una curva ROC un gráfico que representa el rendimiento del proceso de clasificación en diferentes umbrales de probabilidad predicha. Se presenta en la Figura 5.7.

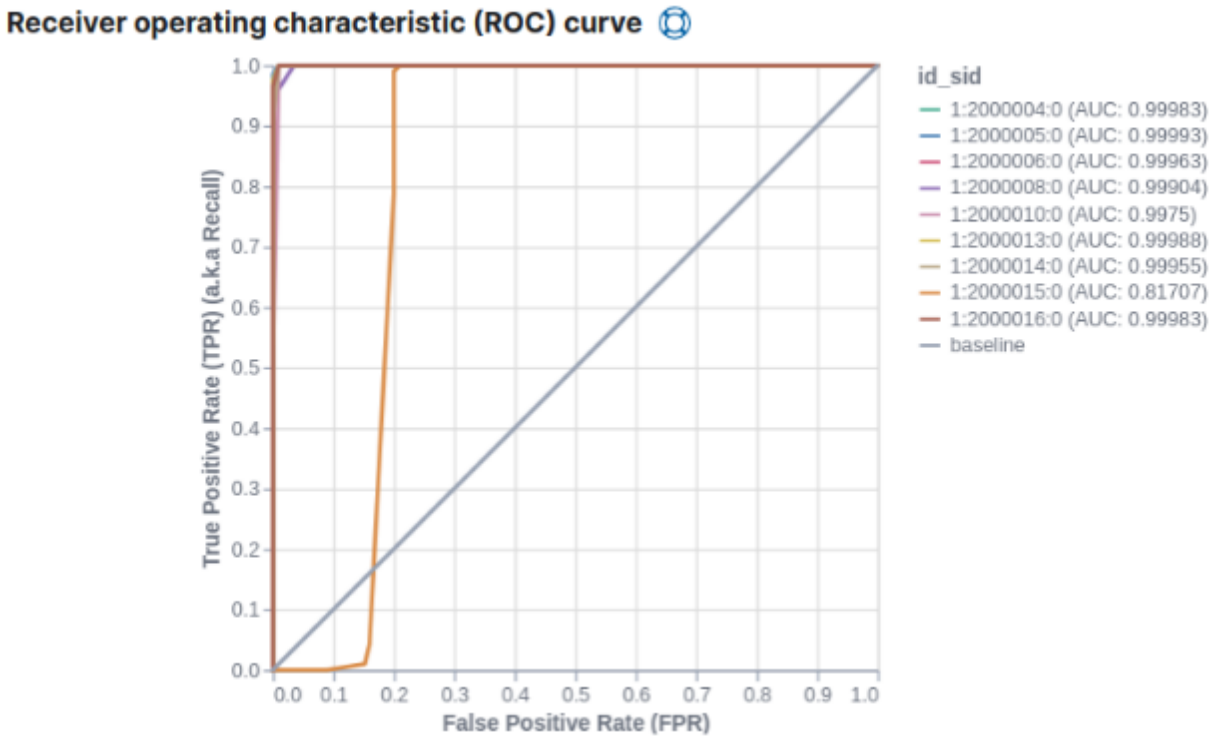


Figura 5.7 Curva ROC para la detección de falsos positivos y verdaderos positivos. Fuente: Elaboración propia

Esta curva compara la tasa de verdaderos positivos (eje y) para una clase específica con la tasa de falsos positivos (eje x) en diferentes niveles de umbral para crear la curva. A partir de este gráfico, se puede calcular el valor del área bajo la curva (AUC), que es un número entre 0 y 1. Cuanto más cercano a 1, mejor es el rendimiento del algoritmo.

Seguido de esto, se presenta la matriz de diagrama de dispersión de los distintos puertos utilizados, recordando que en una gran cantidad mayor serán los puertos denominados como efímeros, dado que son lo que envían las solicitudes al servidor. Este diagrama se presenta en la Figura 5.8.

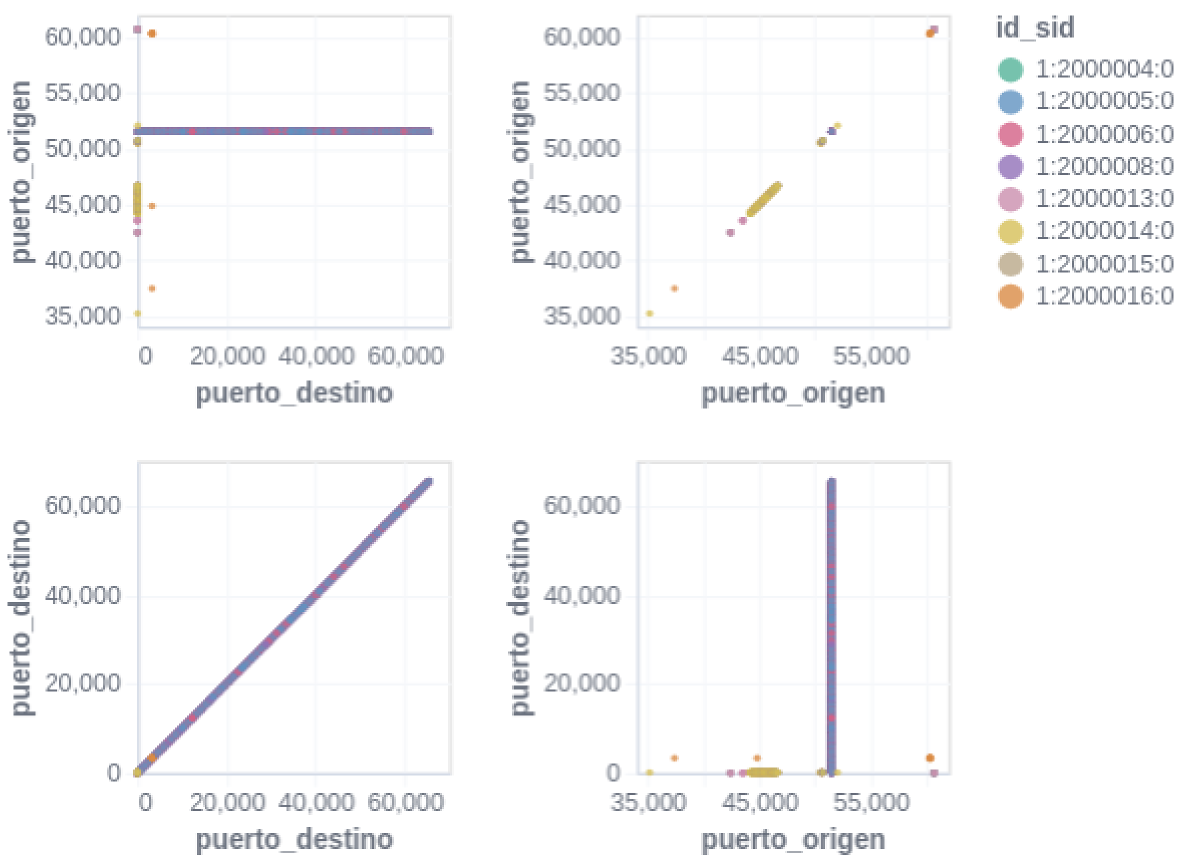


Figura 5.8 Matriz de diagrama de dispersión. Fuente: Elaboración propia

Luego de esto se presentan una tabla de la clasificación total del conjunto de datos, en donde se puede observar la predicción en la columna 2 y el valor real en la columna 3, que presenta muy alta precisión. Se presenta un fragmento de esta tabla en la Figura 5.9.

8 columns hidden   Sort fields   Histogram charts

ml.is_training	ml.prediccion_trafico	id_sid	ml.prediction_probabi...	ml.prediction_score	ml.top_classes	descripc
true	1:2000006:0	1:2000006:0	0.988	0.076	[{"class_score":[0.07575...	Escaneo
true	1:2000005:0	1:2000005:0	0.999	0.076	[{"class_score":[0.07606...	Escaneo
true	1:2000006:0	1:2000006:0	0.988	0.076	[{"class_score":[0.07575...	Escaneo
true	1:2000005:0	1:2000005:0	0.999	0.076	[{"class_score":[0.07606...	Escaneo
true	1:2000006:0	1:2000006:0	0.988	0.076	[{"class_score":[0.07575...	Escaneo
true	1:2000005:0	1:2000005:0	0.999	0.076	[{"class_score":[0.07606...	Escaneo
true	1:2000006:0	1:2000006:0	0.988	0.076	[{"class_score":[0.07575...	Escaneo
true	1:2000005:0	1:2000005:0	0.999	0.076	[{"class_score":[0.07606...	Escaneo
true	1:2000006:0	1:2000006:0	0.988	0.076	[{"class_score":[0.07575...	Escaneo
true	1:2000005:0	1:2000005:0	0.999	0.076	[{"class_score":[0.07606...	Escaneo

Rows per page: 10   1 2 3 4 5 ... 1000

Figura 5.9 Fragmento de la tabla con los resultado generales. Fuente: Elaboración propia

5.4 Evaluación de clasificadores con Weka

A continuación, se presentan la clasificación con los algoritmos de machine learning que Weka ofrece. Estos, presentan sus resultados en base a las métricas de evaluación vista en la sección 3.4.5, “Métricas de evaluación”, se presentaron algunos inconvenientes, expuestos por la capacidad de cómputo del equipo de pruebas, así como el posible diseño de los algoritmos.

El primer algoritmo probado fue Naive Bayes, este obtuvo un 88.86% de efectividad. En este y todos los clasificadores, se realizó un Split del 66% para el entrenamiento y predicción. En la Figura 5.10, se presentan los principales resultados.

```
=== Summary ===

Correctly Classified Instances       16256               88.8646 %
Incorrectly Classified Instances      2037               11.1354 %
Kappa statistic                          0.8351
Mean absolute error                      0.023
Root mean squared error                  0.1336
Relative absolute error                 15.2523 %
Root relative squared error             48.6056 %
Total Number of Instances            18293

=== Detailed Accuracy By Class ===

               TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
               1.000    0.000    0.989      1.000    0.994      0.994    1.000     1.000     1:2000013:0
               1.000    0.000    0.929      1.000    0.963      0.964    1.000     1.000     1:2000004:0
               1.000    0.000    0.917      1.000    0.957      0.957    1.000     1.000     1:2000016:0
               1.000    0.171    0.760      1.000    0.863      0.794    1.000     1.000     1:2000014:0
               0.000    0.000    ?          0.000    ?          ?        1.000     1.000     1:2000010:0
               0.824    0.000    1.000      0.824    0.904      0.874    1.000     1.000     1:2000006:0
               0.829    0.000    1.000      0.829    0.907      0.877    1.000     1.000     1:2000005:0
               0.821    0.000    1.000      0.821    0.902      0.905    0.988     0.972     1:2000008:0
               ?        0.000    0.000      ?        ?          ?        ?         ?         1:2000015:0
Weighted Avg.  0.889    0.060    ?          0.889    ?          ?        1.000     0.999
```

Figura 5.10 Evaluación del clasificador NaiveBayes con un split del 66%. Fuente: Elaboración propia

En la Figura 5.11 se presenta la matriz de confusión de este clasificador, obteniendo resultados los suficientemente aceptables.

```
=== Confusion Matrix ===

    a    b    c    d    e    f    g    h    i   <-- classified as
   89    0    0    0    0    0    0    0    0 |    a = 1:2000013:0
    0   26    0    0    0    0    0    0    0 |    b = 1:2000004:0
    0    0   11    0    0    0    0    0    0 |    c = 1:2000016:0
    0    0    0 6416    0    0    0    0    0 |    d = 1:2000014:0
    0    0    0    0    0    0    0    0    1 |    e = 1:2000010:0
    1    1    1 1002    0 4718    0    0    0 |    f = 1:2000006:0
    0    1    0  979    0    0 4766    0    1 |    g = 1:2000005:0
    0    0    0   50    0    0    0  230    0 |    h = 1:2000008:0
    0    0    0    0    0    0    0    0    0 |    i = 1:2000015:0
```

Figura 5.11 Matriz de confusión del clasificador NaiveBayes. Fuente: Elaboración propia

Luego se presenta la MaginCurve, esta representa una gráfica de los márgenes de decisión o confianza de un clasificador para diferentes instancias en un conjunto de datos. En el caso de Naive Bayes, que es un clasificador probabilístico, el margen podría interpretarse como la diferencia entre las probabilidades asignadas a la clase predicha y la segunda clase más probable en donde Y son las instancias y X la probabilidad de efectividad. Esta grafica se presenta en la Figura 5.12.

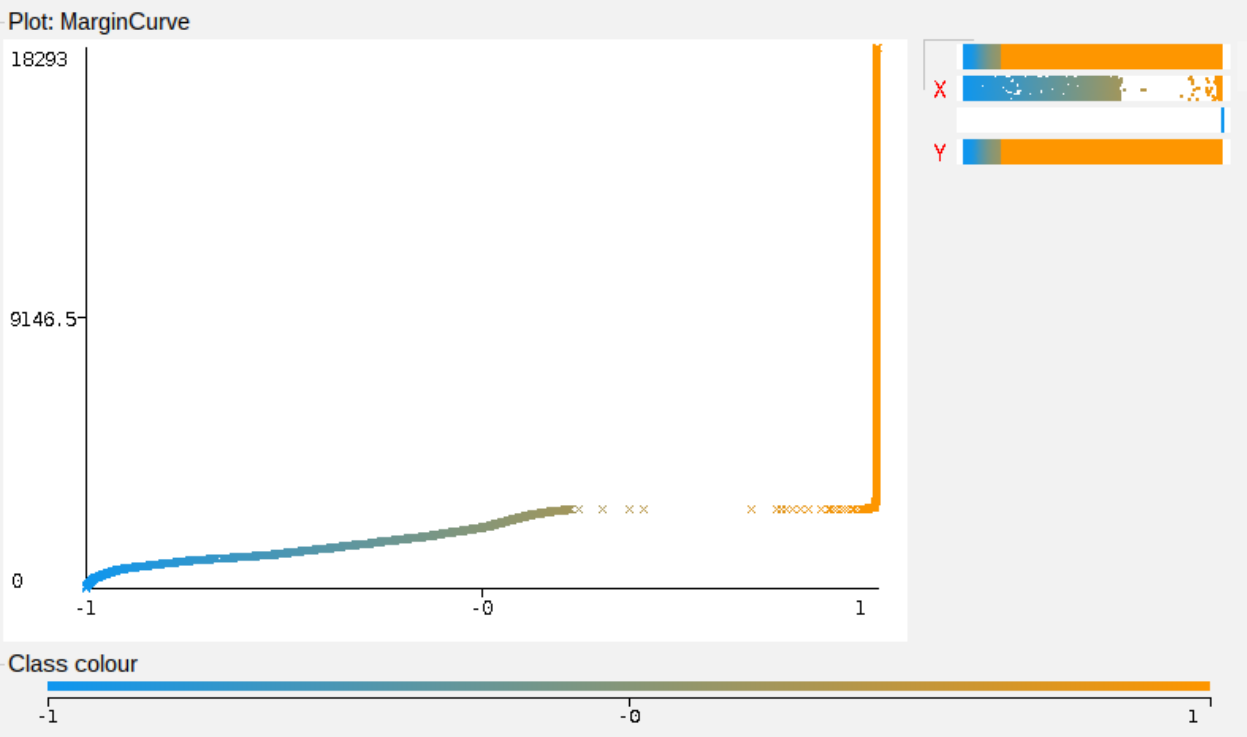


Figura 5.12 MarginCurve del clasificador Naive Bayes. Fuente: Elaboración propia

Seguido de esta evaluación, se continuo con el clasificador de tipo árbol de decisión, J48, el resumen de los resultados se presenta en la Figura 5.13.

Classifier output

```
=== Summary ===

Correctly Classified Instances       18292               99.9945 %
Incorrectly Classified Instances         1                0.0055 %
Kappa statistic                          0.9999
Mean absolute error                      0
Root mean squared error                  0.0028
Relative absolute error                  0.0163 %
Root relative squared error              1.0304 %
Total Number of Instances            18293

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     1:2000013:0
                 1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     1:2000004:0
                 1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     1:2000016:0
                 1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     1:2000014:0
                 0.000    0.000    ?          0.000    ?          ?        0.500     0.000     1:2000010:0
                 1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     1:2000006:0
                 1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     1:2000005:0
                 1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     1:2000008:0
                 ?        0.000    ?          ?        ?          ?        ?         ?         1:2000015:0
Weighted Avg.    1.000    0.000    ?          1.000    ?          ?        1.000     1.000
```

Figura 5.13 Evaluación del clasificador J48. Fuente: Elaboración propia

Este clasificador obtuvo una muy alta precisión en sus resultados destacando un 99.99%, y con una gran velocidad de ejecución. La matriz de confusión se presenta en la Figura 5.14.

```
=== Confusion Matrix ===

    a    b    c    d    e    f    g    h    i   <-- classified as
   89    0    0    0    0    0    0    0    0 |    a = 1:2000013:0
    0   26    0    0    0    0    0    0    0 |    b = 1:2000004:0
    0    0   11    0    0    0    0    0    0 |    c = 1:2000016:0
    0    0    0 6416    0    0    0    0    0 |    d = 1:2000014:0
    0    0    0    1    0    0    0    0    0 |    e = 1:2000010:0
    0    0    0    0    0 5723    0    0    0 |    f = 1:2000006:0
    0    0    0    0    0    0 5747    0    0 |    g = 1:2000005:0
    0    0    0    0    0    0    0  280    0 |    h = 1:2000008:0
    0    0    0    0    0    0    0    0    0 |    i = 1:2000015:0
```

Figura 5.14 Matriz de confusión del clasificador J48. Fuente: Elaboración propia

Se puede observar un grafica del árbol J48 en la Figura 5.15.

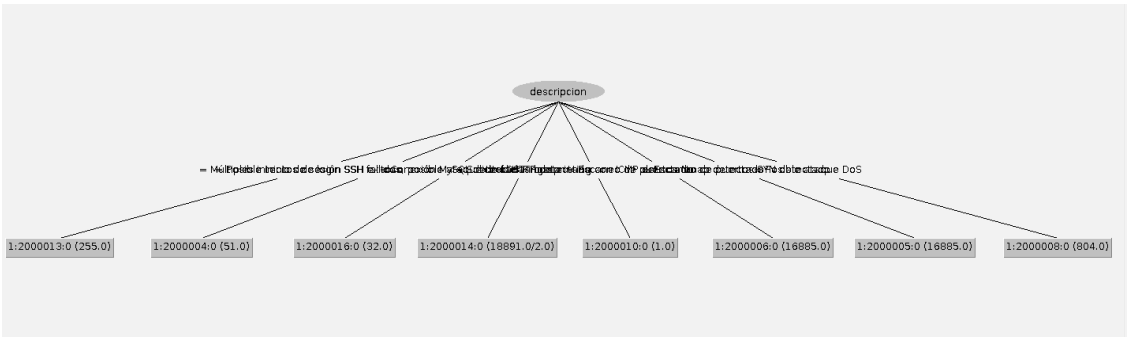


Figura 5.15 Árbol de decisión del clasificador J48. Fuente: Elaboración propia

El siguiente clasificador fue el RandomTree, que presento menor efectividad en la clasificación de instancias, con un total de 89.37%. El resumen de resultados se presenta en la Figura 5.16.

```
=== Summary ===

Correctly Classified Instances       16349               89.373  %
Incorrectly Classified Instances      1944               10.627  %
Kappa statistic                          0.8423
Mean absolute error                      0.0365
Root mean squared error                  0.1361
Relative absolute error                 24.1506 %
Root relative squared error             49.5222 %
Total Number of Instances            18293

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.169    0.000    1.000      0.169    0.288      0.410    0.846     0.180     1:2000013:0
                 0.269    0.000    1.000      0.269    0.424      0.519    0.874     0.272     1:2000004:0
                 0.636    0.000    1.000      0.636    0.778      0.798    0.937     0.637     1:2000016:0
                 0.999    0.163    0.768      0.999    0.868      0.801    0.943     0.838     1:2000014:0
                 0.000    0.000    ?          0.000    ?          ?        0.500     0.000     1:2000010:0
                 0.840    0.000    1.000      0.840    0.913      0.885    0.965     0.922     1:2000006:0
                 0.845    0.001    0.999      0.845    0.915      0.887    0.965     0.923     1:2000005:0
                 0.875    0.000    1.000      0.875    0.933      0.935    0.978     0.880     1:2000008:0
                 ?        0.000    ?          ?        ?          ?        ?         ?         1:2000015:0
Weighted Avg.    0.894    0.057    ?          0.894    ?          ?        0.957     0.887
```

Figura 5.16 Evaluación del clasificador RandomTree. Fuente: Elaboración propia

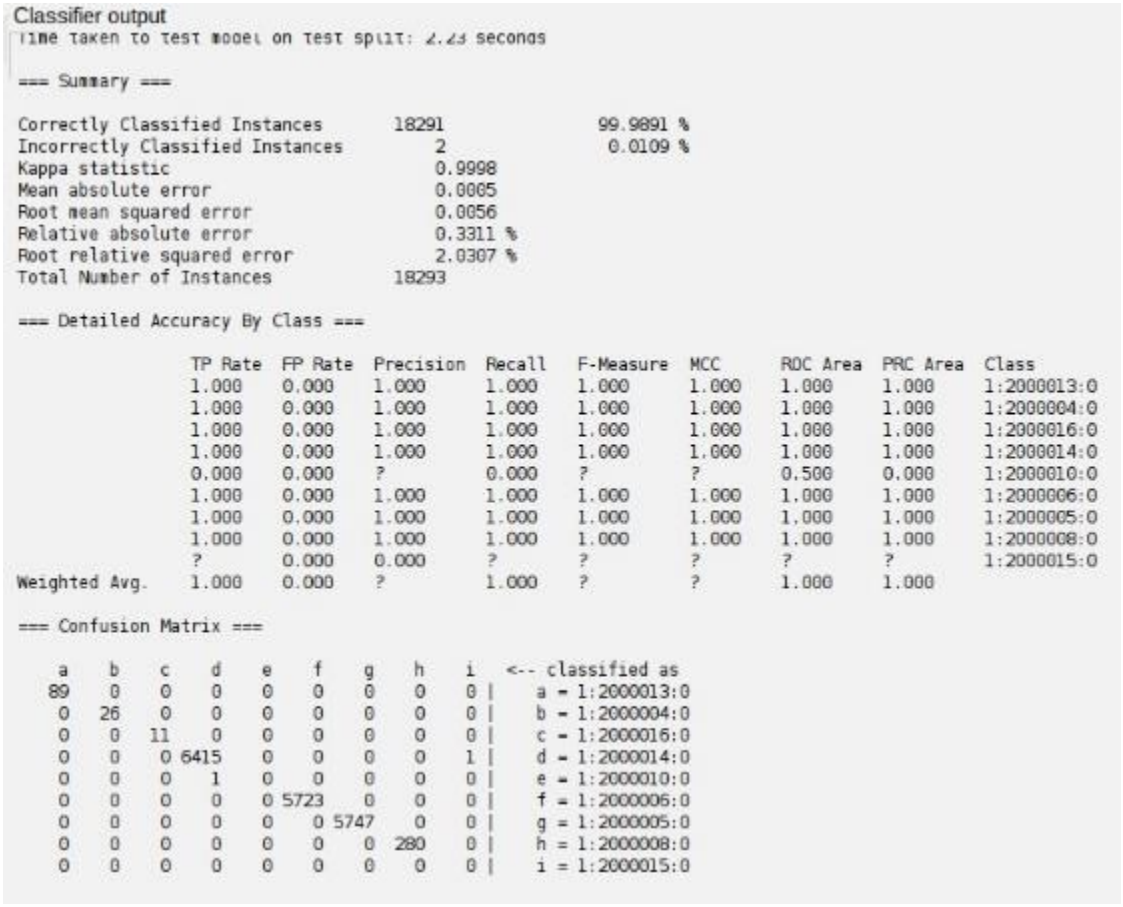
La Matriz de confusión se presenta en la Figura 5.17.

```
=== Confusion Matrix ===

    a    b    c    d    e    f    g    h    i   <-- classified as
   15    0    0   74    0    0    0    0    0 |    a = 1:2000013:0
    0    7    0   19    0    0    0    0    0 |    b = 1:2000004:0
    0    0    7    4    0    0    0    0    0 |    c = 1:2000016:0
    0    0    0 6410    0    0    6    0    0 |    d = 1:2000014:0
    0    0    0    1    0    0    0    0    0 |    e = 1:2000010:0
    0    0    0  914    0 4808    1    0    0 |    f = 1:2000006:0
    0    0    0  889    0    1 4857    0    0 |    g = 1:2000005:0
    0    0    0   35    0    0    0  245    0 |    h = 1:2000008:0
    0    0    0    0    0    0    0    0    0 |    i = 1:2000015:0
```

Figura 5.17 Matriz de confusión del clasificador RandomTree. Fuente: Elaboración propia

La grafica del árbol de decisiones del clasificador RandomTree no fue posible capturar debido a la limitación de cómputo necesaria.

El siguiente clasificador fue RandomForest, este clasificador obtuvo un muy alto porcentaje de efectividad del 99.98%, el resumen de los resultados y matriz de confusión se presentan en la Figura 5.18.



Classifier output

```
Time taken to test model on test split: 2.23 seconds

=== Summary ===

Correctly Classified Instances       18291               99.9891 %
Incorrectly Classified Instances         2                0.0109 %
Kappa statistic                          0.9998
Mean absolute error                      0.0005
Root mean squared error                  0.0056
Relative absolute error                  0.3311 %
Root relative squared error              2.0307 %
Total Number of Instances            18293

=== Detailed Accuracy By Class ===

               TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
               1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     1:2000013:0
               1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     1:2000004:0
               1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     1:2000016:0
               1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     1:2000014:0
               0.000    0.000    ?          0.000    ?          ?        0.500     0.000     1:2000010:0
               1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     1:2000006:0
               1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     1:2000005:0
               1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     1:2000008:0
               ?        0.000    0.000      ?        ?          ?        ?         ?         1:2000015:0
Weighted Avg.  1.000    0.000    ?          1.000    ?          ?        1.000     1.000

=== Confusion Matrix ===

    a    b    c    d    e    f    g    h    i   <-- classified as
   89    0    0    0    0    0    0    0    0 |    a = 1:2000013:0
    0   26    0    0    0    0    0    0    0 |    b = 1:2000004:0
    0    0   11    0    0    0    0    0    0 |    c = 1:2000016:0
    0    0    0 6415    0    0    0    0    1 |    d = 1:2000014:0
    0    0    0    1    0    0    0    0    0 |    e = 1:2000010:0
    0    0    0    0    0 5723    0    0    0 |    f = 1:2000006:0
    0    0    0    0    0    0 5747    0    0 |    g = 1:2000005:0
    0    0    0    0    0    0    0  280    0 |    h = 1:2000008:0
    0    0    0    0    0    0    0    0    0 |    i = 1:2000015:0
```

Figura 5.18 Evaluación del clasificador RandomForest. Fuente: Elaboración propia

Al igual que el anterior clasificador, el poder de cómputo, se vio limitado al ejecutar la gráfica del árbol de decisión de este clasificador.

El siguiente clasificador en ser evaluado fue el algoritmo IBK, este tomo mucho más tiempo que todos los anteriores en su ejecución, así como resulto con un gran porcentaje de efectividad del 99.94%, el resumen de los resultados se presenta en la Figura 5.19.

Classifier output

```
Time taken to test model on test split: 151.64 seconds

=== Summary ===

Correctly Classified Instances       18283               99.9453 %
Incorrectly Classified Instances        10                0.0547 %
Kappa statistic                          0.9992
Mean absolute error                      0.0002
Root mean squared error                  0.011
Relative absolute error                  0.1135 %
Root relative squared error              4.0089 %
Total Number of Instances            18293

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   MCC      ROC Area  PRC Area  Class
               1.000    0.000    0.978      1.000    0.989      0.989    1.000     0.979     1:2000013:0
               0.731    0.000    1.000      0.731    0.844      0.855    0.866     0.731     1:2000004:0
               0.909    0.000    1.000      0.909    0.952      0.953    0.955     0.909     1:2000016:0
               1.000    0.000    0.999      1.000    1.000      0.999    1.000     0.999     1:2000014:0
               0.000    0.000    ?          0.000    ?          ?        0.501     0.000     1:2000010:0
               1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     1:2000006:0
               1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     1:2000005:0
               1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     1:2000008:0
               ?        0.000    0.000      ?        ?          ?        ?         ?         1:2000015:0
Weighted Avg.  0.999    0.000    ?          0.999    ?          ?        1.000     0.999

=== Confusion Matrix ===

    a    b    c    d    e    f    g    h    i   <-- classified as
   89    0    0    0    0    0    0    0    0 |    a = 1:2000013:0
    2   19    0    4    0    1    0    0    0 |    b = 1:2000004:0
    0    0   10    1    0    0    0    0    0 |    c = 1:2000016:0
    0    0    0 6415    0    0    0    0    1 |    d = 1:2000014:0
    0    0    0    0    0    0    1    0    0 |    e = 1:2000010:0
    0    0    0    0    0 5723    0    0    0 |    f = 1:2000006:0
    0    0    0    0    0    0 5747    0    0 |    g = 1:2000005:0
    0    0    0    0    0    0    0  280    0 |    h = 1:2000008:0
    0    0    0    0    0    0    0    0    0 |    i = 1:2000015:0
```

Figura 5.19 Evaluación y matriz de confusión del clasificador IBK. Fuente: Elaboración propia

El ultimo clasificador en ser evaluado fue el MultilayerPerceptron, en donde no fue posible completar su ejecución, dado que la aplicación de Weka, llego a un punto de desbordamiento de memoria, lo que impidió su correcta ejecución. El mensaje de error se puede observar en la Figura 5.20.

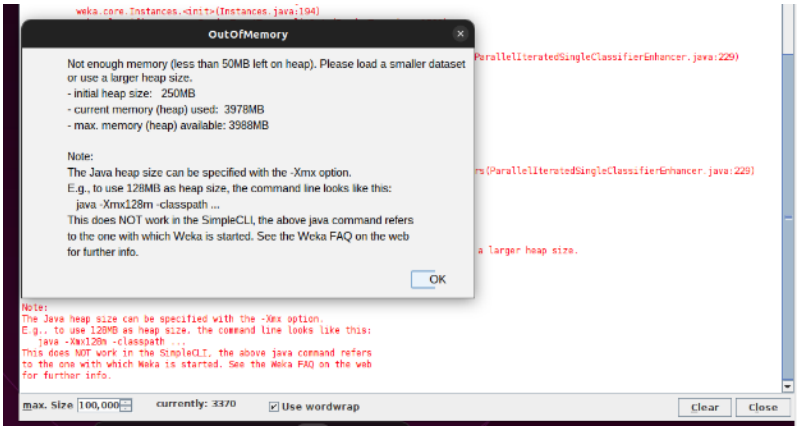


Figura 5.20 Error en la ejecución del clasificador MultilayerPerceptron. Fuente: Elaboración propia

Para abordar este problema, se implementó una solución que consistió en la reducción del tamaño del conjunto de datos a un total de 100 documentos. Aunque idealmente se buscaba una muestra más representativa, las limitaciones de recursos computacionales en el entorno del laboratorio virtual no permitieron alcanzar este objetivo de manera óptima. Esta restricción tuvo como consecuencia que la muestra final estuviera compuesta únicamente por un conjunto de 4 atributos disponibles para la clasificación, como se detalla en la Figura 5.21.

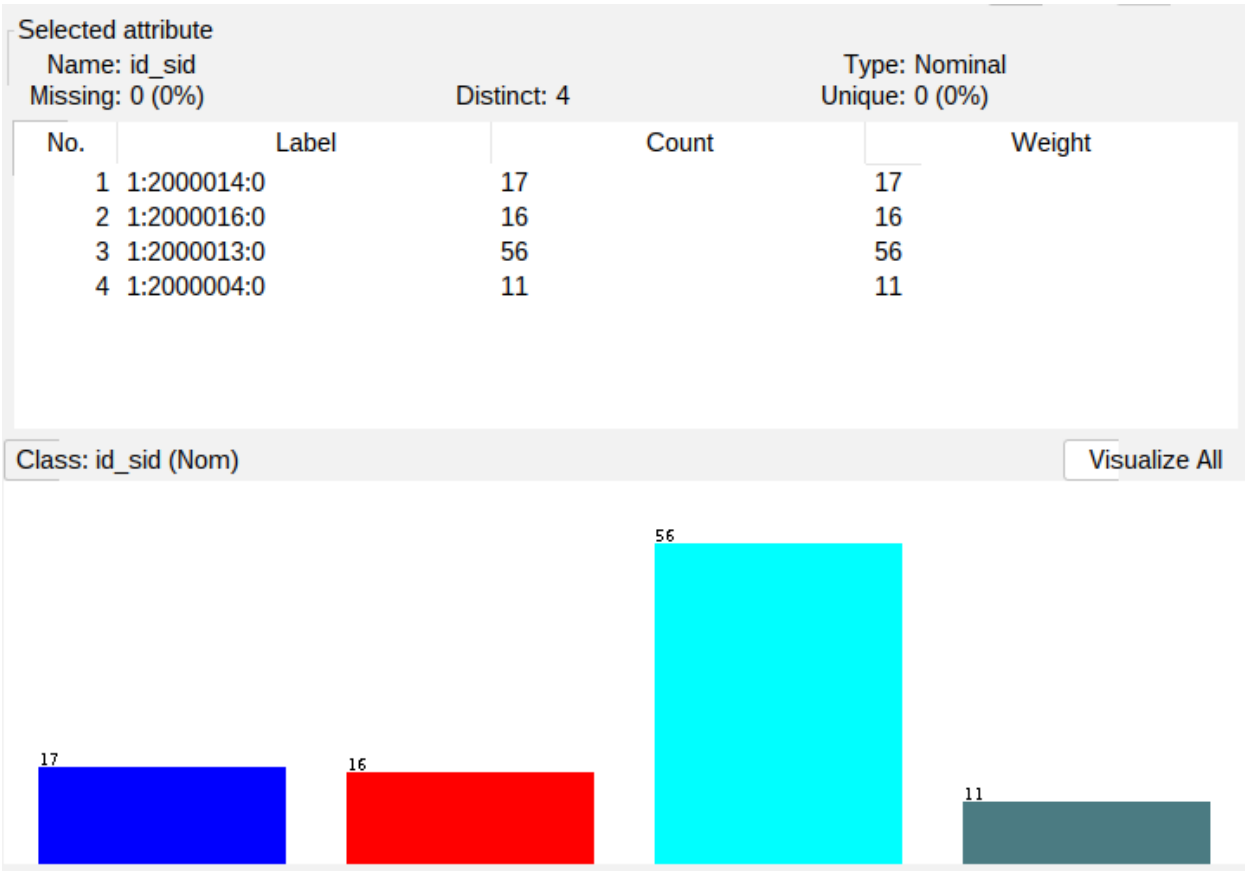


Figura 5.21 Cambios en la muestra del conjunto de datos, para poder ejecutar el clasificador MultilayerPerceptron. Fuente: Elaboración propia.

La evaluación de este clasificador obtuvo muy buenos resultados, pero se debe tener en cuenta lo anterior mencionado sobre la muestra, lo que significa que no es posible compararlo con los anteriores clasificadores en Weka. El resumen de resultados se presenta en la Figura 5.22.

```
=== Summary ===

Correctly Classified Instances          34               100      %
Incorrectly Classified Instances         0                 0      %
Kappa statistic                          1
Mean absolute error                      0.0069
Root mean squared error                  0.0093
Relative absolute error                  2.1304 %
Root relative squared error              2.2223 %
Total Number of Instances               34

=== Detailed Accuracy By Class ===

               TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
               1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     1:2000014:0
               1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     1:2000016:0
               1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     1:2000013:0
               1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     1:2000004:0
Weighted Avg.  1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000

=== Confusion Matrix ===

  a  b  c  d   <-- classified as
  7  0  0  0 |  a = 1:2000014:0
  0  8  0  0 |  b = 1:2000016:0
  0  0 16  0 |  c = 1:2000013:0
  0  0  0  3 |  d = 1:2000004:0
```

Figura 5.22 Evaluación y matriz de confusión del clasificador MultilayerPerceptron. Fuente: Elaboración propia

Este clasificado nos permite ver gráficamente, el modelo de red neuronal, con un lote de 100, a comparación de Tensor Flow que se eligió de 32 así como 500 épocas en comparación con las 5 elegidas en Tensor Flow. La Figura 5.23, representa una captura de pantalla de este modelo gráficamente.

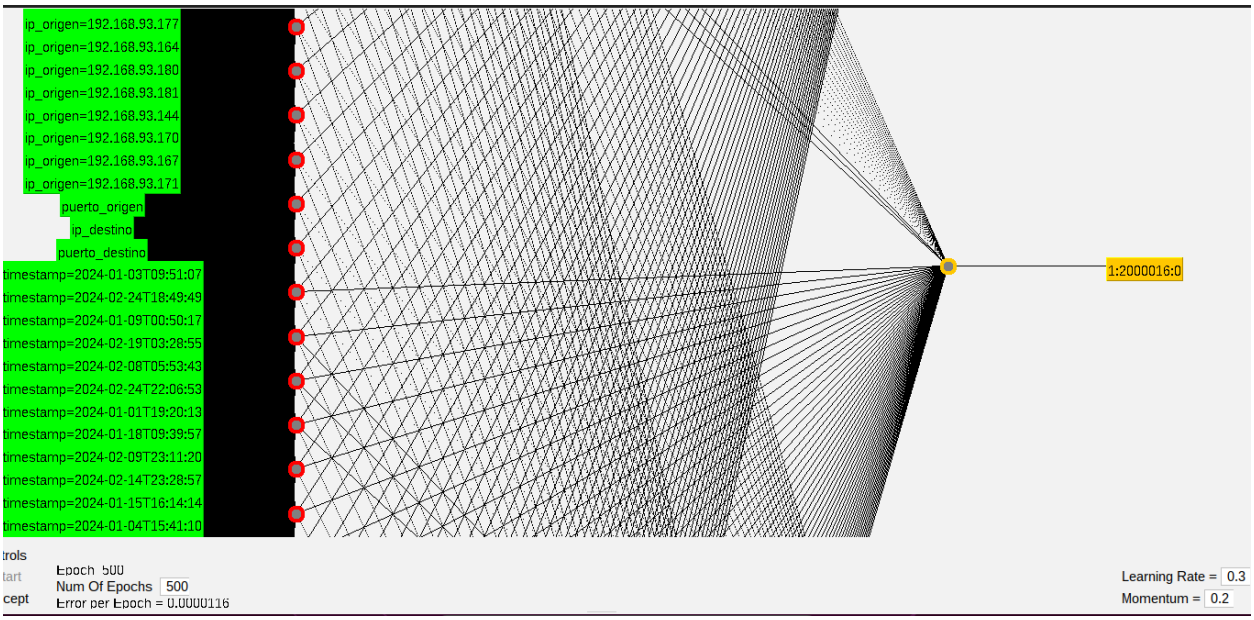


Figura 5.23 Ejemplo gráfico del clasificador MultilayerPerceptron. Fuente: Elaboración propia.

5.5 Evaluación de red neuronal convolucional con Tensor Flow

La evaluación de las redes neuronales con Tensor Flow, presentaron una exactitud de 1, lo que significa que son quizá los parámetros adecuados, para este conjunto de datos.

La primera red en ser evaluada fue la red neuronal densa, con 5 épocas, que se presenta en la Figura 5.24.

```
Epoch 1/5
1407/1407 [==============================] - 12s 7ms/step - loss: 0.0338 - accuracy: 0.9919 - val_loss: 8.9896e-05 - val_accuracy: 1.0000
Epoch 2/5
1407/1407 [==============================] - 12s 8ms/step - loss: 4.1827e-05 - accuracy: 1.0000 - val_loss: 1.7767e-05 - val_accuracy: 1.0000
Epoch 3/5
1407/1407 [==============================] - 10s 7ms/step - loss: 1.0553e-05 - accuracy: 1.0000 - val_loss: 5.6081e-06 - val_accuracy: 1.0000
Epoch 4/5
1407/1407 [==============================] - 10s 7ms/step - loss: 3.6691e-06 - accuracy: 1.0000 - val_loss: 2.0830e-06 - val_accuracy: 1.0000
Epoch 5/5
1407/1407 [==============================] - 10s 7ms/step - loss: 1.4630e-06 - accuracy: 1.0000 - val_loss: 9.0198e-07 - val_accuracy: 1.0000
157/157 [==============================] - 1s 3ms/step
Accuracy: 1.0
```

Figura 5.24 Evaluación de la red neuronal densa con TensorFlow. Elaboración propia

El resume y matriz de confusión se pueden observar en la Figura 5.25.

```
Confusion Matrix:
[[   1    0    0    0    0    0    0]
 [   0  689    0    0    0    0    0]
 [   0    0  706    0    0    0    0]
 [   0    0    0   23    0    0    0]
 [   0    0    0    0    9    0    0]
 [   0    0    0    0    0 3564    0]
 [   0    0    0    0    0    0    8]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         1
           1       1.00      1.00      1.00       689
           2       1.00      1.00      1.00       706
           3       1.00      1.00      1.00        23
           4       1.00      1.00      1.00         9
           5       1.00      1.00      1.00      3564
           6       1.00      1.00      1.00         8
```

Figura 5.25 Evaluación y matriz de confusión de la red neuronal densa en Tensor Flow. Fuente: Elaboración propia

A continuación, se presentan los resultados de la red neuronal recurrente, en la Figura 5.26, se puede observar las 5 épocas de entrenamiento de la red.

```
Epoch 1/5
1407/1407 [==============================] - 25s 16ms/step - loss: 0.0504 - accuracy: 0.9879 - val_loss: 6.6397e-06 - val_accuracy: 1.0000
Epoch 2/5
1407/1407 [==============================] - 24s 17ms/step - loss: 3.9526e-06 - accuracy: 1.0000 - val_loss: 1.2378e-06 - val_accuracy: 1.0000
Epoch 3/5
1407/1407 [==============================] - 23s 16ms/step - loss: 8.7593e-07 - accuracy: 1.0000 - val_loss: 3.8327e-07 - val_accuracy: 1.0000
Epoch 4/5
1407/1407 [==============================] - 21s 15ms/step - loss: 3.0044e-07 - accuracy: 1.0000 - val_loss: 1.7266e-07 - val_accuracy: 1.0000
Epoch 5/5
1407/1407 [==============================] - 23s 16ms/step - loss: 1.1634e-07 - accuracy: 1.0000 - val_loss: 6.8545e-08 - val_accuracy: 1.0000
157/157 [==============================] - 1s 6ms/step
```

Figura 5.26 Evaluación de la red neuronal recurrente con TensorFlow. Fuente: Elaboración propia

En la Figura 5.27, se presenta el resumen y matriz de confusión de la red neuronal.

```
Confusion Matrix:
[[   1    0    0    0    0    0    0]
 [   0  689    0    0    0    0    0]
 [   0    0  706    0    0    0    0]
 [   0    0    0   23    0    0    0]
 [   0    0    0    0    9    0    0]
 [   0    0    0    0    0 3564    0]
 [   0    0    0    0    0    0    8]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         1
           1       1.00      1.00      1.00       689
           2       1.00      1.00      1.00       706
           3       1.00      1.00      1.00        23
           4       1.00      1.00      1.00         9
           5       1.00      1.00      1.00      3564
           6       1.00      1.00      1.00         8

    accuracy                           1.00      5000
   macro avg       1.00      1.00      1.00      5000
weighted avg       1.00      1.00      1.00      5000
```

Figura 5.27 Evaluación y matriz de confusión de la red neuronal recurrente en Tensor Flow. Fuente: Elaboración propia

CAPÍTULO 6 RESULTADOS

Este estudio se centró primordialmente en el análisis de técnicas de inteligencia artificial aplicadas a la detección de anomalías. Dichas anomalías fueron reproducidas en un entorno de laboratorio controlado mediante la generación de tráfico de red, con el propósito de generar un conjunto de datos para su posterior análisis, esté tráfico de red fue categorizado en dos tipos: tráfico normal y tráfico anómalo.

El tráfico normal fue diseñado para simular la interacción típica de usuarios con un servidor, abarcando diversas actividades como solicitudes a servicios tales como bases de datos, conexiones remotas SSH, peticiones HTTP, así como el intercambio de información mediante el protocolo ICMP.

Por otro lado, el tráfico anómalo fue concebido para representar potenciales ciberataques perpetrados por agentes malintencionados. Estos incluyeron las actividades como el reconocimiento y análisis de puertos y servicios a través de ataques de fuerza bruta utilizando la herramienta NMAP, así como ataques de denegación de servicio dirigidos a un servidor HTTP, ataques de denegación de servicio distribuidos contra el servidor HTTP y el protocolo ICMP, y ataques de fuerza bruta dirigidos al servicio SSH. La simulación de estos ataques se llevó a cabo empleando un software desarrollado por el autor de la investigación, denominado “TK_Attack”, esta herramienta se ilustra en la Figura 6.1.

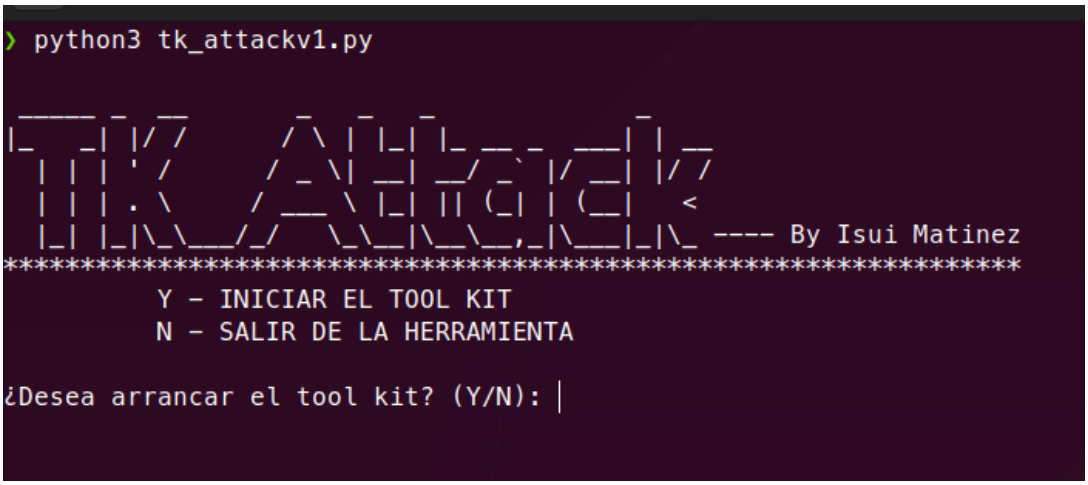


Figura 6.1 Caputa de pantalla del Tool Kit en ejecución. Fuente: Elaboración propia

Durante estas simulaciones realizadas, se obtuvo un conjunto de datos que se presenta gráficamente en la Figura 6.2. Este conjunto consta de un total de poco más de 53,000 registros, los cuales fueron capturados por el sistema de detección de intrusos Snort. Posteriormente, estos registros fueron sometidos a un proceso de transformación y ajuste con el fin de facilitar su comprensión y análisis mediante diversas herramientas de inteligencia artificial.

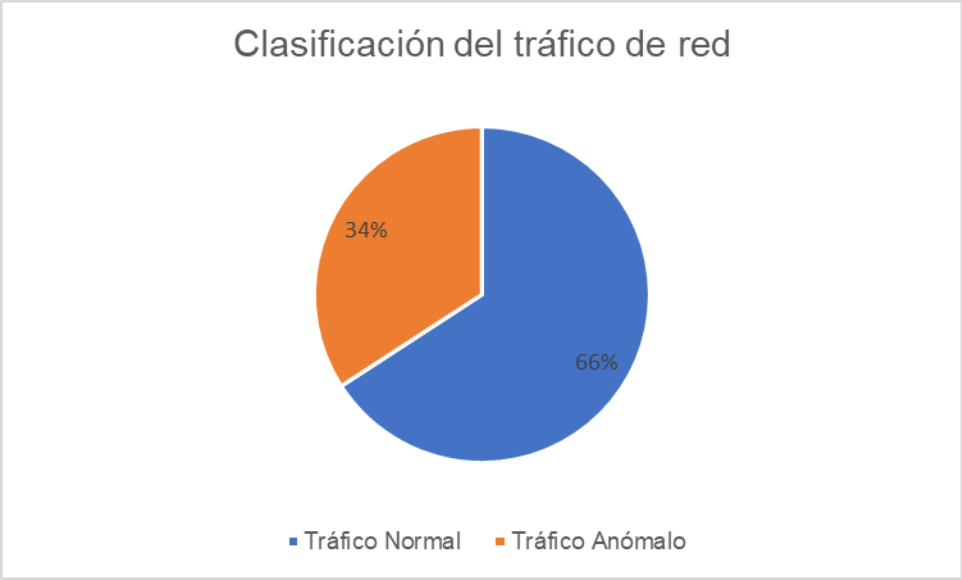


Figura 6.2 Clasificación de tráfico de red, derivado de las simulaciones. Fuente: Elaboración propia

Como se evidencia en la Figura 6.2, del total de registros, el 66% corresponde a tráfico normal, mientras que el restante 34% corresponde a tráfico anómalo.

A continuación, se exponen los resultados derivados de las evaluaciones de los algoritmos de clasificación utilizando las herramientas como Weka y redes neuronales implementadas con Tensor Flow.

Es importante destacar que no se logró realizar una evaluación equitativa con Elasticsearch. La versión gratuita de esta herramienta carece de la capacidad para seleccionar algoritmos de clasificación, ya que dicha funcionalidad está reservada exclusivamente para la versión empresarial, de la cual evidentemente no se dispuso.

En la Tabla 6.1 se ofrece una comparación de los diversos clasificadores, acompañada de sus respectivas métricas de evaluación. Para el caso de la herramienta

Weka todos los modelos fueron evaluados utilizando una división de datos del 66% de entrenamiento y el 34% para la evaluación del clasificador.

Tabla 6.1 Comparación de la efectividad de los clasificadores con Weka. Fuente: Elaboración propia

Clasificador	Efectividad (Accuracy)
NaiveBayes	88.86%
J48	99.99%
RandomTree	89.37%
RandomForest	99.98%
IBK	99.94%
MultilayerPerceptron	99.99
Red Neuronal Densa	100%
Red Neuronal Recurrente	100%

La tabla precedente revela que el algoritmo de clasificación J48 sobresalió con un destacado 99.99% de efectividad en la detección de registros que posiblemente corresponden a ciberataques, en comparación, por ejemplo, con el clasificador NaiveBayes, que obtuvo un 88.86% de efectividad. Esto posiciona a J48 como el algoritmo más eficaz en la clasificación, mientras que las redes neuronales implementadas con TensorFlow alcanzaron una efectividad del 100%. No obstante, al contrastar los resultados del clasificador de Elasticsearch con los obtenidos mediante Weka para propósitos prácticos, se observan similitudes notables. Es crucial señalar que esta comparación se ve limitada por las restricciones inherentes a Elasticsearch, las cuales restringen la elección del algoritmo y la modificación de sus parámetros.

Estos resultados, alentadores y significativos, brindan la oportunidad de aplicar técnicas de inteligencia artificial en la creación de un sistema para la detección de posibles ataques maliciosos o anomalías en el tráfico de red. La eficacia demostrada por el clasificador J48 y la semejanza de resultados entre los clasificadores en Weka y Elasticsearch sugieren la viabilidad de integrar la inteligencia artificial en estrategias de seguridad.

En este contexto, la aplicación de Elasticsearch emerge como una herramienta sumamente útil, ofreciendo perspectivas valiosas para la posible implementación de un Sistema de Gestión de Eventos e Información de Seguridad (SIEM) en una organización.

La capacidad de analizar en tiempo real los registros y logs de diversos sistemas proporciona una valiosa ventaja para identificar y responder de manera proactiva a posibles amenazas, fortaleciendo la postura de seguridad de la infraestructura organizativa.

CONCLUSIÓN Y TRABAJOS FUTUROS

A continuación, se exponen las conclusiones derivadas del análisis de las tres hipótesis planteadas en el primer capítulo.

Hipótesis H1: Identificar el algoritmo de clasificación más efectivo para la detección de ataques en una red de datos.

Basándose en el estudio llevado a cabo y en la revisión de la literatura pertinente, se concluye que el algoritmo J48 es el más idóneo para llevar a cabo este tipo de análisis. Este resultado se sustenta en los datos presentados en la Tabla 6.1, la cual fue basada para la creación de la gráfica comparativa de los diferentes algoritmos de clasificación y redes neuronales analizadas, tal como se muestra en la Figura 6.3.

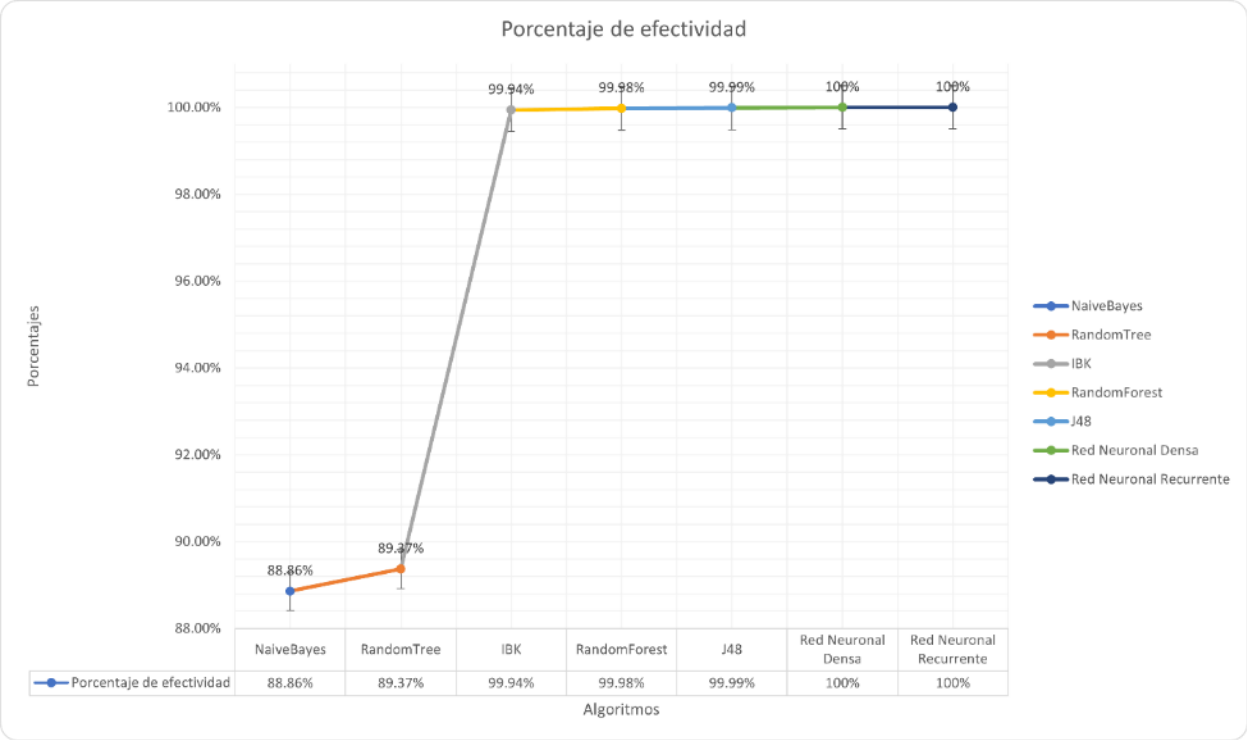


Figura 6.3 Porcentaje de efectividad de los algoritmos y redes neuronales. Fuente: Elaboración propia

Se concluye que el algoritmo de clasificación óptimo es J48, y que la implementación de redes neuronales resulta aún más efectiva. En este sentido, las redes neuronales lograron una efectividad del 100% en este estudio.

Hipótesis H2: Creación de un conjunto de datos mediante la simulación de trafico de red, para su análisis, este tráfico simulara peticiones normales y posibles ciberataques.

El proceso de generación de este conjunto de datos se llevó a cabo gracias a la implementación de un sistema de detección de intrusos conocido como "Snort", el cual sirvió como la base para capturar los registros del tráfico generado durante las simulaciones. Es importante destacar que, aunque se trató de simulaciones, se implementaron y configuraron servicios reales durante este estudio, incluyendo la instalación, así como configuración de un servicio de base de datos, un servidor HTTP, un servicio SSH, entre otros.

Para simular los posibles ciberataques, se desarrolló un software denominado "tool kit attack" en Python, licenciado bajo la MIT License. El propósito de esta licencia es facilitar su uso por parte de otros investigadores y permitir que la comunidad contribuya al desarrollo del mismo. Este programa automatiza ataques de fuerza bruta, escaneos y ataques de denegación de servicio, proporcionando así un entorno controlado para la generación de datos de interés en el ámbito de la seguridad informática.

Se concluye que fue posible cumplir con la hipótesis planteada, dado que se logró la generación de un conjunto de datos basado en las simulaciones expuestas, como resultado de esta hipótesis de desarrollo un programa que automatiza las simulaciones de ciberataques.

H3: Identificar de anomalías de una red de datos, mediante técnicas de monitoreo e Inteligencia Artificial.

Por medio de en análisis de posibles intrusiones con el software Snort, un sistema de detección de intrusos como técnica de monitoreo fue de ayuda para que los registros generados, formaran parte de un conjunto de datos para el análisis con técnicas de inteligencia artificial. Las técnicas de inteligencia artificial, utilizadas fue la utilización de algoritmos de clasificación mediante minería de datos, además de la ejecución de redes neuronales con TensorFlow.

Cabe destacar que las alertas fueron personalizadas para este trabajo de investigación. Sin embargo, no todas las alertas generadas se activaron, ya que ciertos tipos de ciberataques requerían simulaciones más sofisticadas y complejas, como por ejemplo los ataques de envenenamiento ARP o DNS.

Aunque inicialmente se planteó la generación de un conjunto de datos con 200,000 registros, las limitaciones de hardware llevaron a realizar una simulación con 53,000 registros. A pesar de esta reducción, la muestra resultante se considera significativa, aunque menor de la proyectada inicialmente. En la Figura 6.4, se presenta la composición grafica del conjunto de datos por tipos de solicitud.

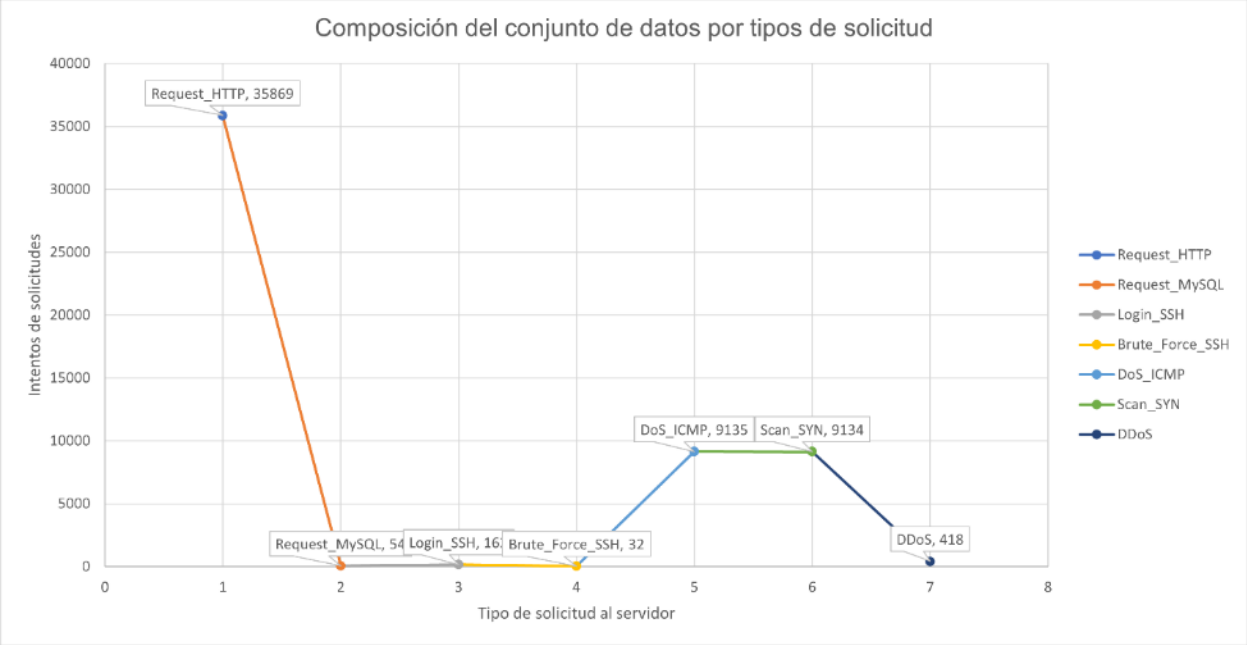


Figura 6.4 Composición del conjunto de datos por tipos de solicitud. Fuente: Elaboración propia

Como se evidencia en la figura anterior, la cantidad de solicitudes normales, como el tráfico HTTP, es notablemente superior en comparación con las conexiones a bases de datos o las conexiones remotas por SSH. Además, las solicitudes identificadas como posibles ataques también representan una proporción considerable en relación con las demás categorías.

Este trabajo no solo permitió una exploración teórica de diversos modelos de clasificación de Machine Learning, sino que también llevó a cabo su implementación

práctica. A pesar de la inicial consideración de distintos clasificadores, la naturaleza del conjunto de datos presentó ciertas restricciones.

Durante la ejecución de este trabajo, se hizo un amplio uso del conocimiento adquirido a lo largo de la carrera, incluyendo habilidades como el manejo de sistemas operativos Linux, la programación de diversos scripts para la realización y automatización de simulaciones, así como la ejecución y configuración de diversos servicios, como bases de datos y sesiones remotas, entre otros.

Como perspectiva a futuro, se contempla la implementación de un Sistema de Gestión de Información y Eventos de Seguridad (SIEM), que permita la visualización y análisis en tiempo real de los registros generados, utilizando técnicas de machine learning para mejorar la detección y respuesta ante posibles amenazas.

Un posible trabajo a futuro, es la continuación del desarrollo del programa elaborado para la simulación de ciberataques automatizados, con una variedad más amplia de posibles intrusiones como backdoor, y keylooger.

Este trabajo de investigación ha sido un impulsor determinante para el autor, quien ha encontrado en él una motivación significativa para continuar su carrera científica. Este proyecto ha servido como un catalizador en su decisión de considerar la posibilidad de postularse a un programa de postgrado en ciencias de la computación o disciplinas afines, con énfasis en la investigación en ciberseguridad

REFERENCIAS

[1] Forbes México. (2022). “Hackeo masivo a Sedena evidencia vulnerabilidad de ciberseguridad; así fue el ataque”. [En línea]. Disponible en: https://www.forbes.com.mx/hackeo-masivo-a-sedena-evidencia-vulnerabilidad-de-ciberseguridad-asi-fue-el-ataque/.

[2] Russell, S., & Norvig, P. (2021). Artificial intelligence: A modern approach (3a ed.). Pearson Education.

[3] Nilsson, N. J. (1998). Artificial intelligence: A new synthesis. Morgan Kaufmann.

[4] Dunning, T., & Friedman, E. (2014). Practical machine learning - A new look at anomaly detection (1a ed.). O’Reilly Media.

[5] Mohri, M., Rostamizadeh, A., & Talwalkar, A. (2018). Foundations of machine learning (2a ed.). MIT Press.

[6] CISA. (2021). “What is cybersecurity?”. [En línea]. Disponible en: https://www.cisa.gov/news-events/news/what-cybersecurity.

[7] ENISA. (2023). “Artificial Intelligence and cybersecurity research”. [En línea]. Disponible en: https://www.enisa.europa.eu/publications/artificial-intelligence-and-cybersecurity-research.

[8] Gartner. (2023) “The Gartner it security approach for the digital age”. [En línea]. Disponible en: https://www.gartner.com/smarterwithgartner/the-gartner-it-security-approach-for-the-digital-age.

[9] Gartner. (2023) “Using the Predict, Prevent, Detect, Respond Framework to communicate your security program strategy”. [En línea]. Disponible en: https://www.gartner.com/en/documents/3286317.

[10] Fortinet. (2023). “Informe del Panorama Segundo semestre de 2022 Global de Amenazas de FortiGuard Labs”. [En línea]. Disponible en: https://www.fortinet.com/lat/corporate/about-us/newsroom/press-

releases/2023/fortiguard-labs-reports-destructive-wiper-malware-increases-over-50-percent.

[11] IFT (2023). “Encuesta Nacional sobre Disponibilidad y Uso de Tecnologías de la Información en los Hogares (ENDUTIH) 2022. (Comunicado de prensa)”. [En línea]. Disponible en: https://www.ift.org.mx/comunicacion-y-medios/comunicados-ift/es/encuesta-nacional-sobre-disponibilidad-y-uso-de-tecnologias-de-la-informacion-en-los-hogares-endutih-0.

[12] Pai, V. D., & Adesh, N. D. (2021). “Comparative analysis of Machine Learning algorithms for Intrusion Detection”. IOP Conf. Ser. Mater. Sci. Eng., 1013(1), 012038.

[13] Abdallah, E. E., Eleisah, W., & Otoom, A. F. (2022). “Intrusion Detection Systems using Supervised Machine Learning Techniques: A survey”. Procedia Comput. Sci., 201, 205–212.

[14] Almseidin, M., Alzubi, M., Kovacs, S., & Alkasassbeh, M. (2017). “Evaluation of machine learning algorithms for intrusion detection system”. In 2017 IEEE 15th International Symposium on Intelligent Systems and Informatics (SISY).

[15] Molina, M. (2022). “What is an intelligent system?”. Universidad Politécnica de Madrid.

[16] Beale, J., & Caswell, B. (2014). Snort 2.1 intrusion detection (2a ed.). Syngress Publishing.

[17] IBM (2023) “¿Qué es un sistema de detección de intrusiones (IDS)?”. [En línea]. Disponible en: https://www.ibm.com/mx-es/topics/intrusion-detection-system.

[18] Ahmad, Z., Khan, A. S., Shiang, C. W., Abdullah, J., & Ahmad, F. (2021). “Network intrusion detection system: A systematic study of machine learning and deep learning approaches”. Trans. Emerg. Telecommun. Technol., 32(1).

[19] Matich, D. J. (2001). Redes Neuronales: Conceptos Básicos y Aplicaciones. Universidad Tecnológica Nacional – Facultad Regional Rosario.

[20] National Geographic. (2019). “Breve historia visual de la inteligencia artificial”.

[21] Meléndez, P., Herrera, R., Pérez, A., & Godínez, J. P. (2000). “EL modelo neuronal de Mcculloch y Pitts Interpretación Comparativa del Modelo”. Research Gate.

[22] Telefónica. (2023). “¿Qué es GPT-3?: la democratización de la inteligencia artificial”.

[23] Open AI. (2023). “Improving language understanding with unsupervised learning”. [En línea]. Disponible en: https://openai.com/research/language-unsupervised.

[24] Riquelme, R. (2023). “ChatGPT hipnotiza y consterna a 6 meses de su lanzamiento”. [En línea]. Disponible en: https://www.eleconomista.com.mx/tecnologia/ChatGPT-hipnotiza-y-consterna-a-6-meses-de-su-lanzamiento-20230614-0045.html.

[25] Open AI. (2023). “GPT-4”. [En línea]. Disponible en: https://openai.com/research/gpt-4.

[26] Zhang, C., & Lu, Y. (2021). “Study on artificial intelligence: The state of the art and future prospects”. J. Ind. Inf. Integr., 23, 100224.

[27] Brío, B. M., & Molina, A. S. (2007). Redes Neuronales y Sistemas Borrosos (3a Edición). Alfa Omega Grupo Editor.

[28] Gershenson, C. (2003). “Artificial Neural Networks for beginners”. arXiv [cs.NE].

[29] Haykin, S. (2009). Neural Networks and Learning Machines. Prentice Hall.

[30] Berzal, F. (2018). Redes neuronales & deep learning. deep-learning.ikor.org.

[31] François-Lavet, V., Henderson, P., Islam, R., Bellemare, M. G., & Pineau, J. (2018). “An introduction to deep reinforcement learning”. Found. Trends Mach. Learn., 11(3–4), 219–354.

[32] ForitguardLabs. (2023). “Febrero de 2023 Informe global del panorama de amenazas Un informe semestral de FortiGuard Labs”. Centro de investigación de Fortiguard Labs, pp. 16–21.

[33] Géron, A. (2022). Hands-on machine learning with scikit-learn, keras, and tensorflow (3a ed.). O’Reilly Media.

[34] Domingos, P. (2016). The master algorithm. Basic Civitas Books.

[35] Biju, J. M., Gopal, N., & Prakash, A. J. (2023). “Cyber attacks and its different types”. [En línea]. Disponible en: https://www.irjet.net/archives/V6/i3/IRJET-V6I31244.pdf.

[36] Forshaw, J. (2017). Attacking network protocols. No Starch Press.

[37] Cloudflare. (2022). “What is a denial-of-service (DoS) attack?”. [En línea]. Disponible en: https://www.cloudflare.com/learning/ddos/glossary/denial-of-service/.

[38] Bursztein, E. (2017). “Inside Mirai the infamous IoT Botnet: A Retrospective Analysis”. Elie Bursztein’s site.

[39] Gentile, N. (2020). “El CIBER-ATAQUE más grande de la historia Ciberseguridad EP 2”. [En línea]. Disponible en: https://youtu.be/8iJLbYNsIYQ.

[40] Cozens, B. (2023). “Port scan attacks: Protecting your business from RDP attacks and Mirai botnets”. [En línea]. Disponible en: https://www.malwarebytes.com/blog/business/2023/04/port-scan-attacks-protecting-your-business-from-rdp-attacks-and-mirai-botnets.

[41] Kaspersky. (2020). “Port scanning”. [En línea]. Disponible en: https://encyclopedia.kaspersky.com/glossary/port-scanning/.

[42] Allen, L. (2023). Advanced penetration testing for highly-secured environments: The ultimate security guide. Packt Publishing.

[43] Najafabadi, M. M., Khoshgoftaar, T. M., Calvert, C., & Kemp, C. (2015). “Detection of SSH brute force attacks using aggregated netflow data”. In 2015

IEEE 14th International Conference on Machine Learning and Applications (ICMLA).

[44] UCL. (2018, enero 24). “What is a hypervisor?”. [En línea]. Disponible en: https://www.ucl.ac.uk/isd/what-ssh-and-how-do-i-use-it.

[45] Lakhani, A., & Muniz, J. (2015). Web Penetration Testing with Kali Linux. Createspace.

[46] Elastic. (2023). “What is elasticsearch?”. [En línea]. Disponible en: https://www.elastic.co/guide/en/elasticsearch/reference/current/elasticsearch-intro.html.

[47] Waikato. (2023). “Weka 3 - data mining with open source machine learning software in java”. [En línea]. Disponible en: https://www.cs.waikato.ac.nz/ml/weka/.

[48] V. Silaparasetty. (2020) Deep learning projects using TensorFlow 2: Neural network development with python and keras, 1a ed. Berlín, Alemania: APress.

[49] Biblus.us.es. (2023). “4. Snort”. [En línea]. Disponible en: https://biblus.us.es/bibing/proyectos/abreproy/12077/fichero/memoria%252Fpor_capitulos%252F04.snort.pdf.

[50] CYVATAR.AI. (2022). “Writing snort rules with examples and cheat sheet”. [En línea]. Disponible en: https://cyvatar.ai/write-configure-snort-rules/.

[51] VM Ware. (2018). “What is SSH and how do I use it?”. [En línea]. Disponible en: https://www.vmware.com/topics/glossary/content/hypervisor.html.

ENLACES RELACIONADOS

https://www.flaticon.com/free-icons/firewall" title="firewall icons" Firewall icons created by Ida Desi Mariana - Flaticon

https://www.flaticon.com/free-icons/installer" title="installer icons" Installer icons created by Freepik - Flaticon

https://www.flaticon.com/free-icons/network-switch" title="network-switch icons" Network-switch icons created by Chattapat - Flaticon

https://www.flaticon.com/free-icons/server" title="server icons" Server icons created by Roundicons - Flaticon

https://www.flaticon.com/free-icons/computer" title="computer icons" Computer icons created by Freepik - Flaticon

https://www.flaticon.com/free-icons/printer" title="printer icons" Printer icons created by Freepik - Flaticon

https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.stickpng.com%2Fimg%2Ficons-logos-emojis%2Ftech-companies%2Fsnort-logo&psig=AOvVaw3wz7CykKRnaog2SB9-jFja&ust=1705089340783000&source=images&cd=vfe&opi=89978449&ved=0CBMQjRxqFwoTCLDq6fWO1oMDFQAAAAAdAAAAABAD

https://www.flaticon.com/free-icons/deep-learning" title="deep learning icons"

https://www.flaticon.com/free-icons/python" title="python icons"

https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html

https://www.kali.org/docs/

https://parrotsec.org/docs/

https://help.ubuntu.com/

https://www.udemy.com/course/elasticsearch-para-todos/

https://youtu.be/CU24iC3grq8

https://youtu.be/MwqdKevOas0

https://youtu.be/uU_GwKbGuZk