Benemérita Universidad Autónoma de Puebla

Facultad de Ciencias de la Computación

T E S I S

INTELIGENCIA ARTIFICIAL PARA EL ANÁLISIS E IMPLEMENTACIÓN DE TÉCNICAS DE DETECCIÓN DE INTRUSIONES

Presenta: *Carolina Maximo Arista*

Para obtener el grado de: Licenciatura en Ciencias de la Computación

Director: *M. C. ANA CLAUDIA ZENTENO VÁZQUEZ*

Codirector: *M.S.C. JUDITH PÉREZ MARCIAL*

Puebla, Pue, Octubre 2025

AGRADECIMIENTOS

En primer lugar, le agradezco a Dios por permitirme llegar a este momento tan especial en mi vida y por haberme brindado la salud para logar mis objetivos, además su infinita bondad y amor. Asimismo, resulta imprescindible agradecer a mi mamá por el esfuerzo tan grande que ha hecho durante toda mi vida y la motivación constante que me ha permitido ser una persona de bien, porque es mi apoyo incondicional, mi madre, mi amiga, para que pedir más. A mis hermanas por la motivación a recordarme la importancia de siempre seguir adelante.

Le agradezco muy profundamente a mi tutora por su dedicación y paciencia, sin sus palabras y correcciones precisas no hubiese podido lograr llegar a esta instancia tan anhelada. Gracias por su guía y todos sus consejos los llevaré grabados para siempre en la memoria en mi futuro profesional. Su experiencia y dedicación fueron fundamentales para superar los desafíos que se presentaron durante este proceso.

Deseo manifestar mi gratitud hacia mis amigos por su apoyo y cariño, ya que siempre estuvieron ahí en los momentos más difíciles para ayudarme a salir adelante ser incondicionales, porque juntos hicimos crecer esta amistad que será para toda la vida.

Finalmente agradezco a todas las personas que de una manera me apoyaron a concluir este proyecto. Ya que siempre me brindaron su apoyo , me animaron y cada palabra de aliento me motivaron durante este camino, se los agradezco de corazón .

RESUMEN

Es conveniente mencionar que en los últimos años los avances entorno a las herramientas diseñadas para la supervisión del tráfico en las redes informáticas, conocidas como Sistemas de Detección de Intrusos (IDS por sus siglas en inglés), han sido notables. Dicha evolución ha sido muy positiva para la Comisión Nacional de Seguridad Nuclear y Salvaguardias de México (CNSNS), ya que desde 2021 a 2024 estas se vio atacada en múltiples ocasiones, lo que demuestra que era necesario implementar mecanismos al respecto con carácter de urgencia.

Los IDS son fundamentales para defender las redes de ataques cada vez más sofisticados. Básicamente, hay dos tipos: unos buscan patrones de ataques conocidos y otros rastrean comportamientos extraños. La verdad es que tanto una como otra estrategia tienen sus debilidades. Por un lado, generan muchas señales falsas, lo que puede ser realmente engorroso. Por otro, es necesario actualizarlas constantemente si queremos seguir el ritmo de las nuevas amenazas que surgen. Esto supone un gran desafío para los equipos de seguridad, pues las amenazas cambian con gran rapidez. Como resultado, es bastante difícil mantener su eficacia.

Como existen problemas en el mundo académico y tecnológico, hemos optado por explorar otras opciones. La forma más interesante de abordar estos problemas implica el uso de inteligencia Artificial. Algunas técnicas, como el aprendizaje automático también llamado Machine Learning o aprendizaje profundo, conocido como Deep Learning, han sido muy útiles. Sin embargo, lo mejor de todo es que estos métodos pueden aprender a detectar patrones raros o extraños. Dicho de otro modo, los sistemas pueden identificar fallos o irregularidades en segundos y con un nivel de precisión jamás posible. En el ámbito de la seguridad informática, el uso de dichos sistemas nos ha permitido ver el uso del (Sistema de Administración de la Seguridad de la Información) SADI y el prototipo BEHOLDER (prototipo de sistema de detección y monitoreo) demostrando cómo las redes neuronales pueden utilizarse para

detectar intrusiones de manera efectiva. Por ejemplo, al implementar estas redes en ambientes complicados, revelan patrones y características precisos que las técnicas anteriores no lograban ver. Por otro lado, y a pesar del progreso, nos enfrentamos a algunos problemas. Uno de los más difíciles de resolver es la selección de las variables correctas para asegurarnos de que nuestros modelos neuronales funcionen correctamente. Desafortunadamente, no es tan sencillo: como en cada paso se plantean millones de posibilidades diferentes, cada elección afecta cómo se detecta lo que queremos. Lo que nos queda es aprender a trabajar en mejorar esta selección para poder detectar de forma incluso más precisa.

Además, los conjuntos de datos como KDD Cup 1999, NSL-KDD y BoT-IoT han contribuido críticamente al progreso en el algoritmo de diseño e implementación de estos sistemas. Es importante mencionar que cada uno de los Dataset se obtuvieron a través de un (tráfico de red simulado en un laboratorio, universidad)**.** Las bases de datos han validado los modelos de IA en un entorno realista, mejorando su capacidad predictiva. También, la adición de técnicas de extracción de características de condensación es necesaria para mejorar el rendimiento de los IDS al reducir la información disponible en los modelos para procesar datos, pero no la calidad percibida de la detección.

Para finalizar, este trabajo considera algunas de las técnicas más recientes para aplicar IA (inteligencia artificial) en la detección de intrusiones. En este trabajo se analizan e implementan algoritmos de ML (aprendizaje automático) y DL (aprendizaje profundo), una evaluación de los datasets y los modelos para elevar la precisión y disminuir el volumen de falsos positivos. De esta manera, un propósito de este trabajo es contribuir al desarrollo de herramientas de ciberseguridad más eficaces en un mundo digital que está volviéndose cada vez más peligroso y vulnerable.

ÍNDICE

CAPÍTULO 1

Introducción a los Sistemas de Detección de Intrusos

1.1 Breve historia de los Sistemas de Detención de Intrusos

Un IDS, por sus siglas en ingles Intrusion Detection System, es un componente esencial en ciberseguridad que permite monitorear los eventos que suceden en el sistema informático para identificar intentos de intrusión. Dichos eventos pueden ser caracterizados como acciones dirigidas contra la confidencialidad, integridad y accesibilidad del sistema informático o apuntar la luz en los contenidos de seguridad que se aplican en esa área. Hay varias características y comportamientos de intrusiones. Entre ellos, especialmente accesos externos no autorizados, es decir, aquellos que llegan vía Internet. Y, simultáneamente, el acceso no autorizado a usuario legítimo al lograr autorización o abuso de derechos concedidos.

En México, la CNSNS (Comisión Nacional de Seguridad Nuclear y Salvaguardias) sufrió una serie de ciberataques que evidenciaron la necesidad de fortalecer sus equipos de protección digital. Entre los años 2021 y 2024, se documentaron al menos cuatro incidentes relevantes. El primero tuvo lugar el 27 de abril de 2021, cuando uno de los empleados recibió un correo electrónico malicioso que, mediante un archivo infectado, permitió un acceso no autorizado por un breve periodo de tiempo. Posteriormente, durante el año 2023, la institución volvió a ser blanco de múltiples intentos de intrusión, en su mayoría vinculados a campañas de *phishing*, las cuales afectaron brevemente los sistemas nucleares antes de ser neutralizadas. Es importante mencionar que México cuenta con siete instalaciones nucleares, muchas de las cuales han permanecido inactivas durante años.

Un nuevo episodio se registró en marzo de 2024, cuando grupos cibercriminales, supuestamente originarios de Brasil, perpetraron una serie de ataques dirigidos a obtener control o sustraer información confidencial de los sistemas. Debido a la gravedad de estos eventos, las autoridades optaron por no divulgar públicamente la magnitud total del impacto [1].

Gracias al intercambio de información digital se ha visto un incremento en desarrollo de las redes de datos por lo que a su vez ha generado nuevas vulnerabilidades ya que la identificación y detección de trafico malicioso hoy en día se ha convertido en

una necesidad para así evitar alteraciones en el funcionamiento de las redes .Es por ello que el desarroll de los IDS proviene de la década de 1980 , ya que surgieron como una herramienta que analizaban en tiempo real los patrones de tráfico para así detectar las anomalías que surgieron en esa época [2]. Su capacidad de análisis los convierte en una pieza fundamental en la arquitectura de seguridad, ya que su funcionamiento es de carácter pasivo, esto quiere decir que no bloquean las amenazas directamente.

La historia de los IDS da origen hasta el año 1972, cuando James P. Anderson, vinculado a la Fuerza Aérea de los Estados Unidos, elaboró un informe pionero sobre la seguridad informática. En su trabajo de 1980, Computer Security Threat Monitoring and Surveillance Denning allanó el camino para la implementación de IDS revisando los archivos de registro o logs para detectar intrusiones [3]. En 1984-1996, Denning y Neumann crearon el sistema IDES Instrusion Detection Expert System; se cree que es el primer IDS basado en reglas [3]. A partir de entonces, se han desarrollado muchas variantes. Según la implementación, los IDS se dividen en sistemas que monitorean el sistema desde adentro, es decir, el IDS controla los sistemas locales; y los monitorea desde el exterior, es decir, la atención se centra en el tráfico de la red.

El crecimiento exponencial de los dispositivos conectados y la mayor dependencia de la tecnología dificultan la tarea de garantizar los niveles apropiados de seguridad. Como consecuencia, se han producido un aumento en los ciberataques por lo que ha provocado el golpe para la economía y la sociedad [4]. Por lo tanto, se han implementado esfuerzos de investigación más activos para producir herramientas preventivas y de detección más efectivas. Los IDS se han vuelto particularmente populares porque les permite identificar cualquier actividad que pueda catalogarse como sospechosa y que cause perjuicios a la red informática o al sistema. A pesar de ser una tecnología relativamente joven, ha seguido progresos significativos y varios enfoques propuestos con el fin de hacer que las tecnologías sean más ingeniosas [2].

Los datos ingresados, por su parte, pueden ser diferenciados de acuerdo con la información sobre la que trabajan y el método de análisis. En este sentido, una de

las opciones más mencionadas es la Intrusión en la Detección Sistemas de Red, pero su enfoque principal es más sobre el método de la detección de anormalidades. En otras palabras, se basa en el principio de que es posible identificar el ingresado al describir su comportamiento normal y cualquier desviación demuestra ser una posible amenaza. Los enfoques para resolver este problema han implicado la aplicación de ANN (Red Neuronal Artificial) y SVM (Máquina de Vectores de Soporte), que son especialmente útiles para la detección de patrones generalizados de anomalías debido el conocimiento de corto plazo características. Sin embargo, es también revelado que el rendimiento de ANN y SVM disminuirá con el número aumento de entradas debido a un mal desequilibrio.

Es por esa razón que una etapa clave en el diseño de un sistema eficiente es una cuidadosa selección de variables hechas para el análisis. Por lo tanto, se puede lograr un equilibrio entre la eficiencia en la detección, que mejora a medida que aumenta la cantidad de datos, y la eficiencia computacional, que posee una mayor eficacia a medida que disminuye la cantidad de entradas. Aquí las técnicas de reducción de funciones sobresalen, ya que ayudan a reducir la cantidad de información en entradas sin eliminar la calidad del análisis.

1.2 Importancia en el contexto actual de ciberseguridad

Hoy en día, la detección de intrusiones es un aspecto indispensable de las estrategias de defensa en profundidad actuales. Como resultado, este enfoque no solo involucra mecanismos individuales de protección, sino que, por un lado, involucra la coordinación de múltiples alturas de seguridad para asegurarse de que las amenazas potenciales sean prevenidas, identificadas y reducidas [5]. Por lo tanto, los sistemas de detección de intrusiones son decisivos. Dado que los intentos de comprometer la seguridad del sistema pueden no estar confinados a un tiempo limitado, los IDS son útiles ya que son capaces de monitorear el tráfico de red en tiempo real. Posteriormente, un IDS generar alertas cuando se dan comportamientos que son sospechosos o representan un riesgo para la seguridad.

- *Respuesta temprana y reducción de los riesgos*. Uno de los beneficios de un sistema de detección de intrusos son los patrones de adquisición de amenazas en el momento justo. Eso permite usar medidas preventivas para que ni una amenaza llegue a un incidente serio. Por ejemplo, en caso de un ataque de ransomware, los patrones raros siguen siendo fácilmente reconocibles, incluso sin un sistema de prevención. Entendido formalmente, un cifrado rápido de los archivos, o una enorme cantidad de fundamentos, es fácil de disminuir. Por lo general, una detección rápida es muy importante para reducir daños en trabajo; a veces, este idea puede salvar una empresa evitando frustraciones en trabajo o pérdidas dinero o problemas entre servicios [6].
- *Salvaguardar la información sensible y los activos críticos*. Dada la frecuencia del robo de datos confidenciales como el objetivo de ciberdelincuentes, no se puede deducirse sobre la naturaleza obligatoria de un IDS. El sistema podría detectar intentos de entrada no autorizados o incluso de extracción antes de que el hacker lograe robar lo que había venido a buscar [7]. Este tema se vuelve aún más serio si queremos hablar de sectores estratégicos o de infraestructura crítica. El ataque de Natanz en Irán también puede mencionarse como un ejemplo perfecto; el nivel de alerta inmediatamente ya que indicó la necesidad de un IDS para solucionar una amenaza complicada que mezclaba un gran número de elementos físicos y digitales [8].
- *Cumplimiento normativo.* Cada vez más, las regulaciones nacionales e internacionales exigen a las organizaciones la implementación de medidas efectivas de seguridad informática. Por ejemplo, el GDPR (Reglamento General de Protección de Datos) en europa, otras pautas reguladoras de NIST (Instituto Nacional de Estándares y Tecnología) en los EUA afirman la necesidad de adoptar tecnologías que puedan ayudar a identificar y mitigar ciberamenazas. En este caso, la introducción de un IDS no solo ayuda a una organización a mejorar su postura defensiva, sino que también facilita la obediencia a la regulación y la demostración de la conformidad legal y técnica con los estándares mencionados.[5].

- *Detectar amenazas internas.* Por último, pero no menos importante, no todas las amenazas vienen de afuera. Los riesgos internos, como el mal uso de los privilegios por parte de los empleados o las acciones intencionales de las personas autorizadas, son un desafío continuo para la seguridad de la organización. Los IDS permiten a las organizaciones monitorear el comportamiento del usuario dentro de la red y alertar de cualquier comportamiento sospechoso, como el acceso de los empleados a archivos sensibles fuera del horario habitual o la transferencia de información sin autorización [9].

Los fundamentos importantes de la ciberseguridad de hoy en día se muestran en la Figura 1 en la que se visualizan de una manera más clara y precisa.

Figura 1 Importancia de la ciberseguridad (Aula21, 2025)

1.3 Evolución de las técnicas y tecnologías aplicación

Durante la década de los noventa, comenzaron a comercializarse los primeros IDS funcionales, basados principalmente en técnicas de detección por firmas (signature-based detection). Estos sistemas eran capaces de reconocer patrones específicos de ataques ya conocidos, gracias a su comparación con una base de datos de firmas previamente almacenadas. Si bien esta estrategia demostró ser eficaz para identificar amenazas preexistentes, presentaba deficiencias importantes frente a variantes desconocidas o ataques novedosos, evidenciando la necesidad de

mecanismos más flexibles y adaptativos [10].

Estas limitaciones dieron origen a la detección de anomalías (anomaly-based detection). Esta técnica cambia el paradigma de un sistema que revisa los registros de intrusión en busca de intentos de ataques conocidos a un sistema que revisa el comportamiento actual del sistema según un perfil de utilización normal, ya que se abre paso. Se utilizaban métodos estadísticos, redes neuronales artificiales y algoritmos de aprendizaje automático para detectar intentos de ataque desconocidos. Además, fue posible disminuir drásticamente los falsos positivos, una forma común de anomalía de los sistemas [11].

En cuanto a los avances recientes en inteligencia artificial, se han creado métodos aún más complejos que utilizan deep learning para aumentar la capacidad de detección y clasificación. Ejemplos populares de tales modelos son las redes neuronales convolucionales y recurrentes. En la bibliografía Kim, Lee y Kim (2016) confirma que las CNN (Red Neuronal Convolucional) y RNN (Red Neuronal Recurrente) muestran un alto rendimiento en la detección de patrones complejos en el tráfico gracias a su capacidad de precisión y adaptabilidad. Otras fuentes confirman que los enfoques que combinan enfoques híbridos de detección de firmas y basados en anormalidades demuestran ser los más útiles al mejorar la cobertura y la precisión simultáneamente [12].

La adopción de nuevas tecnologías como la computación en la nube, el Internet de las cosas y los entornos distribuidos han provocado que existan retos adicionales en términos de ciberseguridad. En cada caso, las mencionadas tecnologías incrementan las superficies atacantes, lo que ha permitido su desarrollo en soluciones mucho más escalables y eficaces. En este mismo orden de ideas, “los IDS modernos han evolucionado hacia esquemas más descentralizados”, por lo que es posible integrarlos con herramientas de análisis en tiempo real, minería de datos e incluso utilizar algoritmos XAI (Inteligencia Artificial Explicable) en sus procesos.

Esto no solo fortalece la eficacia de la detección, sino que también proporciona a los administradores una visión más clara y, por lo tanto, comprensible de la actividad interna del sistema, fomentando la seguridad proactiva [13].

Objetivo General:

1.4 Objetivos generales y específicos del proyecto

Objetivo General:

Desarrollar e implementar un sistema basado en inteligencia artificial que mejore la capacidad de responder a ataques cibernéticos mediante algoritmos de aprendizaje automático y métodos de análisis de datos.

Objetivos Específicos:

- Investigar y analizar las técnicas actuales de detección de intrusiones basadas en inteligencia artificial, evaluando su efectividad y limitaciones en diferentes entornos de red.
- Desarrollar un modelo de aprendizaje automático que permita la identificación y clasificación de patrones de comportamiento anómalos en el tráfico de red,
- Evaluar la precisión y tasa de falsos positivos de los modelos implementados en diferentes escenarios de ataques cibernéticos simulados.

CAPÍTULO 2

Antecedentes de los Sistemas de Detección de Intrusos

2.1 Problemática actual en la detección de intrusos

A pesar de los logros significativos en la implementación y desarrollo de IDS, todavía existen muchos obstáculos que obstaculizan la efectividad de los sistemas de prevención de intrusiones, especialmente para los proveedores de tecnología sofisticada y avanzada que se desarrolla constantemente. Uno de los problemas más comunes aquí es el alto número de falsos positivos, es decir, cuando un sistema identifica una actividad normal y tranquila como un ataque o un comportamiento delincuente. A pesar de la molestia de recibir una gran cantidad de alertas falsas, este fenómeno también puede provocar una falta de confianza en el sistema. Por lo tanto, afecta negativamente la eficiencia operativa [14].

El siguiente desafío es que muchos IDS tradicionales son completamente incapaces de detectar amenazas previamente desconocidas, es decir, ataques de día cero. Dado que los sistemas basados en la firma dependen exclusivamente de lo que ya se ha observado como patrón, son impotentes contra nuevas amenazas o variantes ligeramente modificadas de ataques conocidos. Aunque los enfoques basados en detección de anomalías son una alternativa razonable; solo funcionan cuando hay un modelo preciso de lo que es normal en el sistema. Además, en infraestructuras que son heterogéneas y, por lo tanto, dinámicas impide la creación de un modelo que sea preciso aunque solo sea un poco, en efecto todo el tiempo [15].

Además, en las redes modernas, la escalabilidad y la eficiencia operativa son inquietudes clave, dadas las grandes cantidades de datos que deben manejar los dispositivos. En situaciones como la computación en la nube o el Internet de las cosas, los IDS deben procesar una gran cantidad de información en tiempo real, sin afectar la capacidad del sistema de rendimiento [16]. La velocidad y la escala, muchos de los enfoques utilizados en la actualidad todavía luchan por cumplir con estas demandas. La interpretabilidad de los resultados también es crucial. Con el uso de métodos avanzados como el aprendizaje profundo, los modelos de detección se convierten en cajas negras, es decir, se les hace difícil determinar cómo se toman las decisiones dentro del sistema. Por razones técnicas y legales, esto es una gran barrera para la adopción en do sectores como la trazabilidad y la explicación de los eventos son esenciales para su uso [13].

Por último, los atacantes siguen mejorando técnicas de evasión que intentan evadir los mecanismos de detección de los IDS. Estrategias tales como el cifrado de tráfico, la fragmentación de paquetes o el uso de canales encubiertos plantean desafíos complejos que los sistemas actuales no siempre pueden realizar con eficacia [17]. En este contexto, los desafíos actuales de la detección de intrusos demuestran la necesidad de desarrollar soluciones más robustas, adaptables y explicables. También será vital que puedan rendir eficazmente en entornos diversos y dinámicos, afrontando una amenaza que crece en rapideces y complejidades.

A parte de las visiones ya explicadas con respecto a la eficiencia, precisión y adaptabilidad de los IDS, hay nuevas complejidades que el escenario de la ciberseguridad en 2025 propone y que deberían ser tenidas en cuenta. La sofisticación de los ciberataques ha evolucionado, impulsada por el mal uso de tecnologías emergentes, como la inteligencia artificial generativa, que cambia la naturaleza de las amenazas. Lo anterior mencionado aunado a la capacidad de automatizar acciones, perfeccionar técnicas y analizar grandes bases de datos para acciones más específicas y efectivas. Otro punto de preocupación es una creciente tendencia a los ataques a infraestructuras críticas, particularmente los sistemas de OT (Tecnología Operacional) esenciales para sectores como energía, transporte y telecomunicaciones. Estos están cada vez más conectados a los sistemas IT (Tecnologías de la Información) y son extremadamente vulnerables y en algunos casos, los grupos cibercriminales, algunos de ellos apoyados por los estados.

Otra es la creciente dependencia de la cadena de suministro. Debido a que las organizaciones comparten datos y servicios con múltiples proveedores, están indirectamente expuestas a las brechas de seguridad que vienen de más allá de su perímetro. Un incidente en un tercero puede detener la operación de toda la red interconectada. Ante la urgencia de estas amenazas, los marcos normativos, como la Directiva NIS2 (Directiva sobre la Seguridad de las Redes y Sistemas de Información 2) y el Reglamento DORA (Reglamento sobre Resiliencia Operativa Digital) en Europa, requieren una gestión más proactiva y transparente de los riesgos. Ha habido una mayor responsabilidad y claridad en la notificación de incidentes y en la responsabilidad de respuesta de una institución. Por último, el

crecimiento en la cantidad de dispositivos conectados en IoT (Internet de las Cosas) complica la infraestructura, ya que muchos de ellos carecen de mecanismos adecuados para la autenticación y gestión adecuada, lo que expande la superficie de ataque.

Por lo tanto, los IDS deberían adecuarse a estos entornos dinámicos, no solo sino también como parte de estructuras de seguridad más grandes, como los enfoques de confianza cero y las herramientas que combinan los sistemas de TI (Tecnología de información) y OT (Tecnología Operacional). La inteligencia artificial explicada, la criptografía postcuántica y la cooperación intergubernamental en materia de ciberseguridad resultarán esenciales para descubrir y reducir los riesgos potenciales [18]

2.2 Comparación de Sistemas tradicionales y técnicas basadas en IA

En las últimas décadas, los Sistemas de Detección de Intrusos han evolucionado de enfoques previamente basados en reglas predefinidas tradicionales a soluciones más avanzadas con soporte de IA (Inteligencias Artificial). Este cambio no solo mejoró la manera en que los sistemas detectan amenazas sino que también permitió que los sistemas se adapten a los ambientes cambiantes y nuevos métodos de penetración. Tradicionalmente, los sistemas IDS (Sistema de detención de intrusos) funcionan principalmente utilizando un método de detección por firmas (signature-based), donde el sistema compara el tráfico de la red con un conjunto de firmas conocidas de ataques. Si bien la metodología *signature-based* es eficiente en la detección de amenazas previamente identificadas, tiene limitaciones significativas para omega variants o zero-day attacks. Además, los sistemas basados en reglas solo permiten a los operadores humanos definir que cualquer comportamiento debe considerarse sospechosos, disminuyendo así la escalabilidad de enfoque y haciendo estos sistemas vulnerables a errores humanos [19].

En cambio, las técnicas basadas en IA (Inteligencia artificial), en especial en machine learning y deep learning, se implementan y desarrollan a través de modelos que “aprenden” directamente con datos históricos .Por lo demás, este enfoque permite a los sistemas descontar las anomalías, sin necesidad de que esta detección esté

basada en reglas explícitas, por tanto, se supone una mejor capacidad de detección de ataques nuevos, adaptación a cambios en el entorno y falta de dependencia de bases de datos de firmas [19] [20]. No obstante, estos abordajes de IA (Inteligencia artificial) tienen sus retos, el primero de ellos es, por ejemplo, que es necesario contar con conjuntos de datos amplios y etiquetados para un entrenamiento adecuado. Otros de los problemas que se presentan es los costos computacionales, su capacidad para interpretar las decisiones de modo en que los modelos complejos son, por ejemplo, redes neuronales profundas. Asimismo, cabe mencionar los algoritmos atacables a través denominado ataques adversariales diseñados de tal manera para que engañen al detector y pasen desapercibidos [26]. Sobre la base de otro diferenciador de estos enfoques es su actualización. Mientras que en un modelo tradicional necesita una actuación explícita y manual para aprender reglas nuevas o firmas adicionales, los sistemas soportados por IA (Inteligencia artificial) son actualizables mediante reentrenamiento constante [21].

En definitiva, aunque los métodos tradicionales aún se emplean ampliamente debido a su simplicidad y eficacia frente a amenazas conocidas, las técnicas basadas en inteligencia artificial representan una evolución significativa. Estas ofrecen ventajas sustanciales en cuanto a automatización, capacidad de detección proactiva y escalabilidad. La tendencia actual se orienta hacia el diseño de sistemas híbridos, capaces de combinar la precisión de los enfoques tradicionales con la flexibilidad y adaptabilidad que ofrecen las soluciones basadas en aprendizaje automático.

Este enfoque permite a los sistemas descontar las anomalías, sin necesidad de que esta detección esté basada en reglas explícitas, por tanto, se supone una mejor capacidad de detección de ataques nuevos, adaptación a cambios en el entorno y falta de dependencia de bases de datos de firmas. No obstante, estos abordaje de IA tienen su retos, el primero de ellos es, por ejemplo, que es necesario contar con conjuntos de datos amplios y etiquetados para un entrenamiento adecuado . Otros de los problemas que se presentan es los costos computacionales, su capacidad para interpretar las decisiones de modo en que los modelos complejos son, por ejemplo, redes neuronales profundas. Asimismo, cabe mencionar los algoritmos atacables a través denominado ataques adversariales diseñados de tal manera para

que engañen al detector y pasen desapercibidos. Sobre la base de otro diferenciador de estos enfoques es su actualización . Mientras que en un modelo tradicional necesita una actuación explícita y manual para aprender reglas nuevas o firmas adicionales, los sistemas soportados por IA (inteligencia artificial) son actualizables mediante reentrenamiento constante.

En el mundo conectado de hoy, las organizaciones se enfrentan a un entorno en el que las amenazas cambian rápidamente, no solo en intensidad sino también en sofisticación. Este dinamismo se ha amplificado mediante la introducción de tecnologías emergentes como la inteligencia artificial generativa, que no solo optimiza nuestras herramientas de defensa, sino que también se ha adaptado para ayudar a actuar malintencionados a desarrollar ataques más direcciones, automatizados y difíciles de detectar. Según el *IBM Cost of a Data Breach Report* BDO Global (202**4),** los costes de las filtraciones aumentaron un 10 % con respecto al año anterior, siendo el mayor incremento anual desde la pandemia. Esto se ve agravado por la escasez mundial de personal en ciberseguridad capacitado y experimentado, lo que significa que las organizaciones no tienen la capacidad de detectar y detener eficazmente las amenazas. Sin embargo, se ha mejorado en términos de detección temprana a través de inversiones en planificación de seguridad y tecnologías de IA (inteligencia artificial) diseñadas para compensar operativamente las deficiencias. En 2025, las amenazas que se esperan superen las más importantes serán los ataques patrocinados por el estado. Estos no se basan en la mera ganancia económica, sino en la obtención de una ventaja geoespacial a largo plazo al destinar recursos a la creación de capacidades ofensivas avanzadas. Constantemente los grupos cibernéticos delictivos, organizados a veces apoyados por los estados, han estado reutilizando capacidades ofensivas recién adquiridas para orquestar una serie de ataques en contra.

También se encuentra la amenaza implícita de un ataque realizado dentro de la organización. Un empleado, contratista o socio en un trabajo que tiene acceso a sistemas críticos autorizados puede comprometer la integridad de los datos intencionalmente o por negligencia. Los incidentes internos han llegado a exponer registros un total de cinco veces de aquellos causados por una amenaza exterior.

También, están los ataques sobre la cadena de suministro, los atacantes comprometen a los proveedores externos para infiltrarse en los sistemas de una organización, lo que puede tener consecuencias amplias, afectar a múltiples partes interesadas de la organización y el ecosistema empresarial. En términos de técnicas, el ransomware sigue siendo una de las amenazas más disruptivas. Su rápido crecimiento, a saber, $26 mil millones en ingresos en 2020, ha sido facilitado por la trayectoria de colaboración entre ciberdelincuentes, quienes comparten tácticas y herramientas en espacios clandestinos. Casos recientes como el de la industria de salud, la automotriz y las bibliotecas públicas han demostrado que el ransomware puede cortar operaciones por semanas y causar pérdidas significativas.

Más allá, amenazas como la suplantación de identidad por correo electrónico, apropiación de credenciales, fraude financiero y desinformación; en particular cuando se complementan con ingeniería social y la vulnerabilidad humana seguramente seguirán siendo problemas preocupantes [22]. En este sentido, la ciberseguridad es un negocio de todos, no solo pertenece a los especialistas del área de Tecnologías de la información. De hecho, esta es una estrategia corporativa que se refiere directamente a la administración, ya que es empresarial. Todo debe ser orquestado de manera que los riesgos tecnológicos estén conectados a misión. En 2021 y más allá, la combinación de enfoques multifacéticos con IA (inteligencia artificial) , marcos de gestión de riesgos y monitoreo, conciencia y políticas será esencial para abordar estos grandes problemas de seguridad.

En la Figura 2 se muestra una gráfica de los principales indicadores del año actual y el porcentaje de cada uno de ellos.

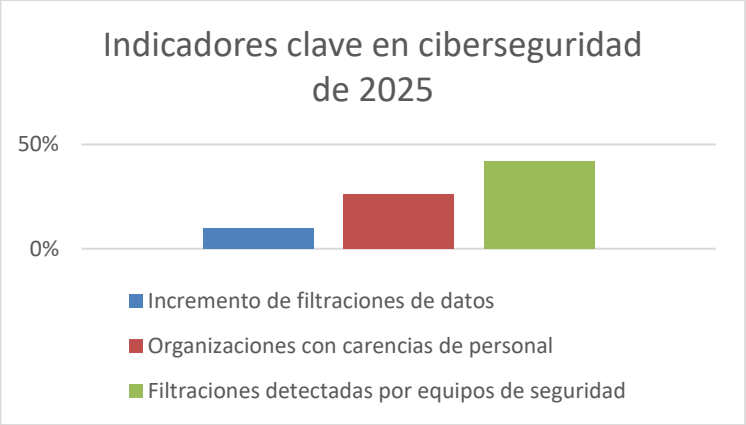

Figura 2 Indicadores clave en ciberseguridad para 2025

2.3 IA para detección de intrusiones

La técnica que ha proliferado como marco utilizado para la detección de intrusiones y detección de anomalías es conocida como inteligencia artificial. Esta técnica tiene la ventaja de otorgar al sistema la capacidad única de procesar cantidades masivas de datos, identificar patrones y adaptarse a nuevos tipos de amenazas, lo que se logra mucho mejor con IA (inteligencia artificial) en comparación con los enfoques clásicos. La IA ha probado ser el enfoque esencial ya que la eficiencia, la automatización y la escalabilidad. Se vuelve crítico en el mundo digital, donde los atacantes pueden controlar la maquinaria prácticamente limitada [19].

Para el primer trimestre de 2022 se registró un aumento de 46% en ataques DDoS (ataque distribuido de denegación de servicio), lo que equivale a casi 4.5 veces en relación con el mismo periodo del año anterior, debido principalmente a la guerra cibernética que se desató entre Rusia y Ucrania. Sin embargo, Estados Unidos sigue siendo el país que ocupa el primer puesto, representando el 45.02% de ataques que se presentaron para el primer trimestre del año. Es importante resaltar que este tipo de ataques se realizaron por medio de inundación UDP con un 53.64% (Gutnikov, Kupreev, & Shmelev, 2022).

Los algoritmos KNN y bosques aleatorios aborda en tiempo real a una velocidad perceptiblemente más alto que cualquier otra IA basada en la detección de amenazas de tráfico, gracias a la capacidad demostrada de distinguir entre una entidad conveniente y una peligrosa [23].

La otra área de interés se refiere al aprendizaje profundo, que ha demostrado ser fundamental para una amplia gama de aplicaciones. Deep learning se refiere a la capacidad de modelar relaciones muy complejas en datos gracias a las redes neuronales multicapa. Se emplearon varias arquitecturas, como redes convolucionales, redes recurrentes y autoencoders para tareas de detección, principalmente cuando se trata de un gran conjunto de datos [24].

La segunda fortaleza que surge del uso de la IA (inteligencia artificial) en la detección de intrusos es la capacidad de identificar el tráfico proveniente de las amenazas desconocidas, también conocidas como ataques de día cero; abordar las brechas críticas de los sistemas basados en firmas convencionales; la reducción de falsos positivos de la detección a través de una diferenciación más detallada del tráfico legítimo y malicioso; y la adaptabilidad al constituir una detección de aprendizaje continuo. En cuanto a las debilidades se alude a la carencia de los recursos en esta área [23]. Uno de los principales es la necesidad de disponer de una muestra de datos de calidad antes de que sean representativos y se hayan marcado correctamente para entrenar a los modelos. Otros retos también incluyen la dificultad para interpretar los resultados en los modelos de caja negra y la capacidad de los algoritmos de ser manipulados por medio de ataques que se han diseñado específicamente para pasar por encima de los mecanismos de detección [21].

Hoy en día: los enfoques híbridos están en desarrollo, lo que implica integrar técnicas de IA (inteligencia artificial) en los métodos tradicionales para combinar las ventajas de ambas. Tal vez incluso más importante: fortalecen el desarrollo de modelos de IA Inteligencia Artificial Explicable. (XAI), para hacer que la interferencia sea más transparente y rastreable. Estas soluciones son esenciales en sectores críticos, incluidos la defensa, las finanzas y las infraestructuras críticas [13].

CAPÍTULO 3
Marco Metodológico

3. Herramientas y Tecnologías utilizadas

El diseño de sistemas efectivos para la detección de intrusiones implica la integración de aprender un sistema de técnicas capaz de reconocer comportamientos no habituales en el tráfico de la red. Los algoritmos de clasificación supervisada son para elegir cuatro de los cuales han demostrado su eficacia demostrada en la protección de la propiedad intelectual por adelantado investigación, sobre todo la aplicación a la capacitación y al juicio. Ya que se llevaron a cabo utilizando los lenguajes de programación R y Python que proporcionaron un entorno para el desarrollo y validación de soluciones presentados.

3.1 Algoritmos

En este contexto el desarrollo de IDS (sistemas de detección de intrusos) basados en IA requiere elegir algoritmos de aprendizaje supervisado con mucho cuidado, garantizando el equilibrio entre precisión, eficiencia y posibilidad de generalización. Cuatro algoritmos, Random Forest, SVM, KNN y Naive Bayes, están investigados y aplicados en este trabajo, ya que son comúnmente utilizados en el campo de la ciberseguridad en proporción a sus características específicas que los hacen más adecuados para diferentes condiciones de detección y análisis de tráfico de red.
En las siguientes secciones se expondrán con mayor detalle estos algoritmos, revisando sus características teóricas, sus características más significativas, los parámetros de configuración más relevantes. Finalmente, una descripción de su función específica en el proceso de clasificación. El propósito de la revisión no es solo para proporcionar una justificación para su uso en el sistema propuesto, sino también para establecer una conveniencia técnica para entender los resultados.

3.1.1 Algoritmo Ramdom Forest

Random Forest, se trata de un método de aprendizaje en conjunto o también conocido como Ensemble learning en el cual se construye una cantidad de árboles de decisión que entrenan mediante un subconjunto del conjunto de datos original

[28]. En términos de predicción, cada uno aporta con un resultado y se escoge una de las siguientes maneras, voto mayoritario en tareas de clasificación y promedio de los resultados en tareas de regresión.

Una de las principales ventajas de este algoritmo es su capacidad de trabajar con conjuntos de datos que presentan un número elevado de variables. Añadir a la capacidad de generalizarse rápidamente del sobreajuste, en particular en comparación con modelos conformados por un único árbol de decisión [25].

En la figura 3 se muestra un diagrama explicativo acerca de la clasificación del algoritmo de Ramdom forest:

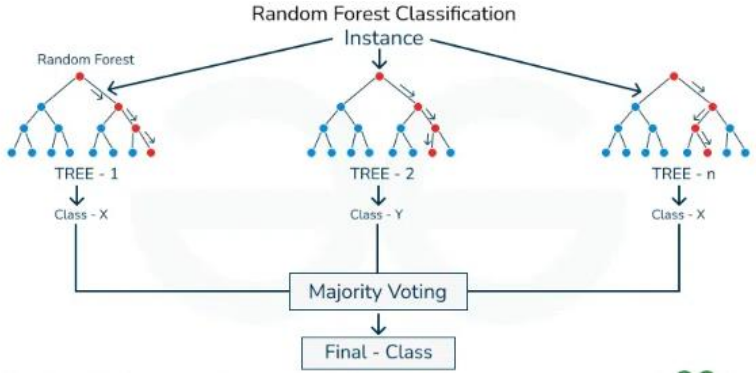

Figura 3 Diagrama de Algoritmo Random Forest (Rahman et al., 2025)

3.1.1.1 Funciones Algoritmo Random Forest

Al respecto, en el contexto de la ciberseguridad, el algoritmo Random Forest tiene varias funciones cuya combinación lo convierte en una herramienta valiosa para la detección de intrusiones. Estos incluyen la clasificación precisa del tráfico de red, lo que significa la capacidad de establecer si un flujo de datos es tráfico legítimo o representa una amenaza. Este caso también se ha demostrado en varios estudios; en particular, los más relevantes son los que se basan en los conjuntos de datos conocidos y ampliamente utilizados, como KDD, NSL-KDD y CICIDS 2017 [31].

En la Detección de Anomalías debido a la arquitectura basada en múltiples árboles, Random Forest tiene la capacidad de identificar el comportamiento inusual en grandes volúmenes de información. Por lo tanto, su uso ha sido beneficioso para reconocer patrones que son diferentes al comportamiento normal, el modelo también

ofrece métricas a través de las cuales se puede determinar qué atributos de tráfico de red o actividad de usuarios deben considerarse más importantes en el proceso de detección. Al mismo tiempo, esto forma la base para la optimización del rendimiento del sistema y la mejor comprensión de los resultados [26].

3.1.1.2 Principales hiperparámetros del Random Forest

Para garantizar un rendimiento óptimo del modelo durante su implementación, una multitud de hiperparámetros clave que influya directamente en la capacidad de generalización y precisión de la modelo asignada a Random Forest fueron considerados. Algunos de los más relevantes son los siguientes:

- *n_estimators*: corresponde al número total de árboles que conforman el bosque. Por defecto, este parámetro escala en 100, lo que significa que se implementarán 100 modelos adicionales. Siendo que su valor por lo general impacta positivamente en la precisión del modelo, requiere más recursos computacionales y tiempo de entrenamiento.
- *max_depth:* define la profundidad máxima permitida para cada árbol. Este parámetro es fundamental para controlar el crecimiento del modelo y evitar problemas de sobreajuste.
- *min_samples_split: establece* el número mínimo de muestras necesarias para dividir un nodo interno. Le permite controlar la complejidad del árbol y fortalecer su desempeño.
- *min_samples_leaf*: indica la cantidad mínima de muestras que debe contener una hoja terminal. Su configuración correcta contribuye a suavizar la estructura del árbol y reduce el riesgo de sobreajuste.
- *max_features*: determina cuántas características deben considerarse al seleccionar la mejor división en cada nodo. Puede expresarse como un número fijo, un porcentaje o mediante valores predefinidos como 'sqrt' o 'log2'.
- *bootstrap:* especifica si se utilizarán muestras con reemplazo durante la construcción de los árboles. Este método es característico de Random Forest

y favorece la diversidad entre árboles.

- *class_weight:* resulta especialmente útil en contextos con clases desbalanceadas, como ocurre frecuentemente en la detección de intrusiones, donde las instancias correspondientes a ataques suelen representar una proporción reducida respecto al tráfico total [25].

3.1.1.3 Ventajas del Random Forest en detección de intrusiones

El algoritmo Random Forest ofrece múltiples ventajas que lo hacen especialmente adecuado para tareas de detección de intrusos en entornos de ciberseguridad. Entre sus principales fortalezas se encuentran:

- *Resistencia al ruido y a datos incompletos*: esta característica lo convierte en una opción robusta para analizar registros de tráfico de red, donde es común encontrar datos faltantes o inconsistentes [27].
- *Buen desempeño en conjuntos de datos* con muchas variables: incluso cuando algunas de estas variables son irrelevantes o presentan alta correlación, el modelo mantiene su efectividad al seleccionar aleatoriamente subconjuntos de características durante el entrenamiento.
- *Alta escalabilidad*: tanto en dimensiones de datos masivos y en número de atributos, lo que permite su aplicación en una línea de producción de redes empresariales de dimensiones medianas y corporativas [28].
- *Capacidad de manejo de clases desbalanceadas*: mediante el uso de parámetros como class_weight o técnicas complementarias como la generación de muestras sintéticas, el algoritmo puede ajustarse para mejorar la detección de clases minoritarias, como los ataques en un IDS.
- *Entrenamiento paralelizable*: cada árbol del bosque puede construirse de manera independiente, lo que permite optimizar el uso de recursos computacionales y reducir significativamente el tiempo de procesamiento [29].

3.1.1.4 Interpretación del modelo

Una de las razones por las cuales Random Forest ha ganado popularidad en aplicaciones prácticas como la ciberseguridad es su capacidad para ofrecer interpretaciones comprensibles de los resultados generados. Esta característica es especialmente valiosa en entornos donde se requiere justificar las decisiones del sistema ante auditorías o situaciones críticas. De hecho, esta característica puede ser valiosa cuando exista una necesidad de autorizar el sistema tao explicar su decisión en un caso en particular.

En consecuencia, las herramientas de análisis más comúnmente utilizadas son, por supuesto, Curva ROC y AUC. Estas métricas son útiles si necesita combinar sensibilidad y especificidad para sistemas de detección: los errores de software en este caso tendrán serias consecuencias.

- *Matriz de confusión*: permite evaluar el rendimiento del modelo a través de métricas como verdaderos positivos, falsos negativos, falsos positivos y verdaderos negativos, proporcionando una visión clara del comportamiento del clasificador.
- *Análisis de importancia de características*: el modelo puede proporcionar información sobre qué atributos del conjunto de datos son más importantes para la regla de decisiones. Esto es especialmente importante para comprender cómo el sistema puede llegar a una determinada conclusión única, transparente para los procesos de evaluación y auditoría visualización de árboles individuales. Ya que el modelo completo se compone de varios de los árboles, estamos interesados en la capacidad de examinarlos de uno en uno para explicar decisiones precisas. Ya que es útil durante la investigación forense cuando queremos reconstruir el razonamiento detrás de una alerta o clasificación.

3.1.1.5 Limitaciones del algoritmo Random Forest

A pesar de la eficacia del algoritmo y su uso generalizado en tareas de clasificación

y regresión, Random Forest tiene una serie de limitaciones que deben tenerse en cuenta al aplicar este enfoque a contextos críticos, como la detección de intrusiones. En primer lugar, aunque el enfoque propuesto se orienta a la reducción del sobreajuste a través de un proceso de voto entre un grupo amplio de árboles de decisión entrenados con subconjuntos aleatorios de datos y características, esto no lo inmuniza de este fenómeno. Por el contrario, en presencia de ruido o atributos irrelevantes, es muy probable que Random Forest sobreajuste los datos, ya que se familiariza con los patrones y no podrá generalizar adecuadamente. En términos abstractos, esto refleja el problema presentado por Srivastava et, H. en el contexto de las redes neuronales de que la dependencia en las combinaciones específicas de las neuronas es riesgosa durante el entrenamiento y esto es precisamente por qué necesitan excluir ciertas neuronas en cada paso de entrenamiento.

Asimismo, Random Forest es un modelo que se considera difícil de interpretar. A medida que aumenta el número de árboles, se limita la posibilidad de observar cómo el modelo está tomando las decisiones, lo cual puede no ser útil en entornos donde la transparencia y la trazabilidad de los resultados son esenciales, como es el caso de los sistemas de seguridad o en los sistemas regulados. Por otro lado, hay que tener en cuenta también que Random Forest puede ser vulnerable a ciertos desequilibrios en los datos. Si hay una clase que contiene muchos más elementos en el conjunto de capacitación, el modelo tiende a marcarla como una última decisión, lo cual en otro caso puede llevar a un comportamiento insuficiente para la detección de los eventos poco frecuentes, como los ataques o las intrusiones en los sistemas de red.

Finalmente, Random Forest no está diseñado para extrapolar más allá del rango de valores observados durante el entrenamiento, lo que puede ser una limitación en escenarios donde se requiere la predicción de valores fuera de la distribución conocida.

3.1.2 Algoritmo Support Vector Machines (SVM)

SVM significa Support Vector Machines, es un algoritmo de machine learning de aprendizaje supervisado usado comúnmente para problemas de regresión y

clasificación. En las Support Vector Machines, dándole al algoritmo un set de datos de entrenamiento denominado training set que ya están clasificados, este mapea los nuevos datos según estén en una categoría o en la otra. En el caso de un problema bidimensional, este hiperplano es una línea que separa los datos a ambos lados de la misma. Se llama Vector Soporte, al vector que está formado por el conjunto de puntos más cercano al hiperplano y son los datos más complicados de clasificar, teniendo una influencia muy elevada en la localización del hiperplano óptimo.
Se conocerán como *atributos* a las variables de predicción y como *característica* al factor principal de clasificación. La metodología para conseguir este hiperplano consistirá en la proyección de los atributos en subespacios de dimensión superior a la de los atributos, consiguiendo una separación de las variables muy eficiente.
En la Figura 4 se ilustra el algoritmo de máquinas de Vector Soporte

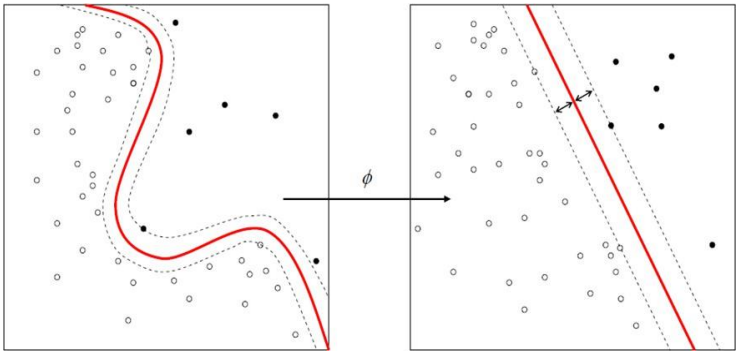

Figura 4 Máquinas de vector (Jiménez, 2016)

3.1.2.1 Funciones principales del modelo

En la implementación por medio de la biblioteca Scikit-learn, el modelo SVM (Máquina de Vectores de Soporte) ofrece un conjunto de funciones esenciales que permiten su entrenamiento, evaluación y aplicación práctica en tareas de clasificación [36]. Entre las más importantes se encuentran:

- *fit [x,y):* Para hacer ajuste con datos se usa esta herramienta. En este caso, se necesita todos los valores de entrada del conjunto de datos y respectivas etiquetas al hiperplano, esta función construirá el mejor separador entre

clases, que es un hiperplano.

- *decision_function(X):*Útil para aprender la distancia de los puntos al hiperplano de decisión y determinar la confianza con la que el modelo hizo la predicción, en otras palabras, se podría usarla para tunning.
- *predict(X):* se utiliza para generar predicciones sobre datos nuevos, permitiendo determinar a qué clase pertenece cada muestra con base en el modelo entrenado.
- *score(X, y):* calcula la precisión del modelo al compararlo con las etiquetas verdaderas y, facilitando la evaluación cuantitativa del desempeño.

Esta funciones son válidas en el caso de escenarios con una separación lineal y para espacios de características más complicados que impliquen el uso de funciones kernel.

3.1.2.2 Principales hiperparámetros

El desempeño de SVM (modelo basado en Máquinas de Vectores de Soporte) depende de la adecuada configuración de sus hiperparámetros, que determinan la forma y la flexibilidad del clasificador [25]. Los hiperparámetros relevantes incluyen:

1. C: un parámetro que regula el equilibrio entre un margen amplio y los errores de clasificación. Bajos valores de C permiten al modelo tolerar ciertos errores, lo que resulta en márgenes más amplios y en generalización. Por otro lado, valores altos obligan al modelo a ajustarse con mayor precisión a los datos de entrenamiento, lo que puede dar lugar a un sobreajuste.

2. *kernel*: define la función que transforma los datos al espacio donde se realiza la separación. Las opciones más comunes incluyen:

    - *'linear':* adecuado cuando las clases son linealmente separables.
    - *'rbf' (Radial Basis Function):* ideal para datos con relaciones no lineales

complejas.

- *'poly':* transforma los datos aplicando un polinomio, útil en situaciones donde la separación se basa en combinaciones no lineales de las variables.

3. gamma: controla la influencia de cada observación individual en la construcción del límite de decisión. Un valor alto hace que solo las muestras cercanas al punto de evaluación influyan en la predicción, mientras que valores bajos generan un efecto más amplio, abarcando regiones más grandes del espacio de datos.
4. *degree:* se utiliza específicamente cuando se selecciona el kernel 'poly' y representa el grado del polinomio empleado. Este valor influye en la complejidad del límite de decisión generado; tener un valor alto, resulta en que solo las muestras cercanas al punto de evaluación afectan a la predicción y, de lo contrario un valor bajo, se cree un efecto más extendido que cubra regiones más grandes del espacio de datos de entrada.
5. *class_weight*: permite asignar distintos pesos a las clases, lo que resulta particularmente útil en contextos de datos desbalanceados, como suele ser el caso en los sistemas de detección de intrusos, donde las instancias positivas (ataques) suelen ser minoritarias.

3.1.2.3 Ventajas del modelo

El uso de Máquinas de Vectores de Soporte (SVM) en sistemas de detección de intrusos presenta diversas ventajas que justifican su implementación en entornos de alta complejidad, como lo destaca la literatura especializada [30]:

- *Eficiencia en espacios de alta dimensionalidad*: SVM mantiene un buen rendimiento incluso cuando se trabaja con conjuntos de datos que contienen un gran número de atributos, lo cual es común en el análisis de tráfico de red.
- *Capacidad para modelar relaciones complejas*: gracias al uso de distintas funciones *kernel*, el modelo puede adaptarse tanto a separaciones lineales

como no lineales, ofreciendo flexibilidad frente a distintos tipos de datos.

- *Resistencia al sobreajuste*: SVM tiende a generalizar bien, incluso en situaciones donde el número de muestras es limitado, lo que reduce el riesgo de que el modelo memorice el conjunto de entrenamiento.
- *Manejo de clases desbalanceadas:* mediante la asignación de pesos diferenciados a cada clase, el algoritmo puede ajustarse adecuadamente en contextos donde las instancias positivas (como los ataques) representan una minoría del total.

3.1.2.4 Interpretación del modelo

La interpretación de los modelos generados por *Máquinas de Vectores de Soporte (SVM)* puede resultar más o menos directa según el tipo de kernel utilizado. No obstante, existen tres elementos fundamentales que permiten comprender cómo se toman las decisiones dentro del modelo:

- *Vectores de soporte*: se trata de las observaciones más cercanas al hiperplano de separación. Estos puntos son determinantes en la construcción del modelo, ya que definen tanto la orientación como la posición del límite de decisión. Su influencia es crítica y afecta directamente al desempeño del clasificador [30].
- *Margen:* corresponde a la distancia entre los vectores de soporte y el hiperplano. Cuanto mayor es este margen, mejor suele ser la capacidad de generalización del modelo, lo que implica una menor probabilidad de sobreajuste.
- *Coeficientes del modelo (en kernels lineales):* en los casos donde se emplea un kernel lineal, es posible interpretar los coeficientes como indicadores del peso que tiene cada variable en el proceso de clasificación. Para modelos no lineales, la interpretación directa no es posible, por lo que se recurre a técnicas complementarias como SHAP (SHapley Additive exPlanations) o LIME (Local Interpretable Model-agnostic Explanations), que permiten estimar la

contribución de cada característica en la decisión del modelo [31].

3.1.3 K-Nearest Neighbors (KNN)

El algoritmo K-Vecinos más Cercanos (KNN) es un método de clasificación que se fundamenta en la cercanía entre datos. Para determinar la clase a la que pertenece una nueva observación, KNN evalúa a sus *k* vecinos más próximos y asigna la categoría que predomina entre ellos, utilizando para ello métricas como la distancia Euclidiana o la distancia de Manhattan [32]

A lo largo del tiempo los parámetros se han ido perfeccionando en los que destacan:

- *n_neighbors (k):* número de vecinos que se toman en cuenta para la clasificación.
- *weights:* esquema de ponderación, ya sea uniforme o basado en la distancia.
- *algorithm:* método empleado para buscar a los vecinos, que puede ser fuerza bruta (brute-force), kd-tree, entre otros.
- *metric:* la función que mide la distancia entre puntos.

3.1.3.1 Hiperparámetros principales

El desempeño del modelo KNN depende en gran medida de la correcta elección de sus hiperparámetros. Entre los más importantes se encuentran:

- *weights:* Determina si todos los vecinos aportan por igual (uniforme) o si se les asigna un peso según su proximidad (distancia). Utilizar ponderación basada en la distancia puede ayudar a mejorar el rendimiento cuando hay vecinos ruidosos en el conjunto de datos [33].
- *metric:* Es la función que se utiliza para medir la distancia entre puntos. La distancia Euclidiana (p=2) es la más frecuente, aunque también se emplean la distancia Manhattan (p=1) o la métrica de Minkowski para obtener una generalización mayor [34].

- *p:* Este parámetro especifica la potencia usada en la métrica de Minkowski.
- *algorithm:* Define el método para la búsqueda de vecinos, con opciones como auto, ball_tree, kd_tree o brute-force. La elección de este algoritmo impacta directamente en la eficiencia del modelo, especialmente cuando se trabaja con grandes conjuntos de datos [31].

3.1.3.2 Ventajas del algoritmo

El algoritmo KNN (K-Vecinos Más Cercanos) presenta varias fortalezas que lo hacen una opción viable en diversas aplicaciones: [32]

- Fácil de aplicar: Dada la sencillez y precisión del algoritmo, es uno de los primeros clasificadores que aprenderá un nuevo científico de datos.
- Se adapta fácilmente: a medida que se agregan nuevas muestras de entrenamiento, el algoritmo se ajusta para tener en cuenta cualquier dato nuevo, ya que todos los datos de entrenamiento se almacenan en la memoria.
- Pocos hiperparámetros: KNN solo requiere un valor 'k' y una métrica de distancia, lo que es bajo en comparación con otros algoritmos de machine learning.

3.1.3.3 Limitaciones del modelo

A pesar de sus fortalezas, el algoritmo KNN (K-Vecinos Más Cercanos) también presenta ciertas limitaciones que es importante tener en mencionar:

- Su fase de predicción resultad muy costosa en términos computacionales, ya que implica calcular la distancia entre la nueva instancia y todas las observaciones almacenadas en el conjunto de datos [35].
- La dimensionalidad afecta su desempeño, pues en espacios con muchas variables las métricas de distancia pierden eficacia, lo que puede degradar la precisión del modelo [36].
- Es sensible a la escala de los atributos, por lo que resulta indispensable aplicar técnicas de normalización o estandarización antes de su uso [37].

- KNN puede ser vulnerable frente a datos ruidosos o conjuntos desbalanceados, especialmente cuando se emplea un número pequeño de vecinos y no se utilizan estrategias de ponderación adecuadas [38].
- Finalmente, la selección adecuada de hiperparámetros como *k* y la métrica de distancia es crucial, requiriendo procesos de validación cruzada para evitar obtener resultados subóptimos [31]

3.1.3.4 Interpretación del modelo

La interpretación del algoritmo KNN (K-Vecinos Más Cercanos) se basa principalmente en el análisis de los vecinos más cercanos a cada muestra. La clase asignada se determina ya sea por mayoría simple o por cualquier otra ponderación basada en la distancia, lo que ofrece una forma intuitiva de entender el proceso de toma de decisiones. Además, visualizar estos vecinos simplifica la narración del modelo para la audiencia no técnica que recibe la explicación posterior con mayor claridad [39]. Para escenarios más complicados, existen técnicas de interpretación post-hoc, que pueden identificar qué atributos afectan el cálculo de la distancia con el conjunto de datos y los vecinos más cercanos. Esto le da una justificación más clara y transparente a la predicción [37].

3.1.4 Algoritmo Naive Bayes

Un algoritmo más crucial aprendizaje automático es el algoritmo Naive Bayes. Se deriva del teorema de la probabilidad de Bayes ya que se utiliza en el análisis de texto y la clasificación de alto conjunto de datos dimensional. Algunos ejemplos del algoritmo Naive Bayes incluyen el análisis sentimental, clasificación de nuevos artículos, y el filtrado de spam. Los algoritmos de clasificación son utilizados para clasificar nuevas observaciones en clases predefinidas para datos no iniciados. El algoritmo Naive Bayes es más utilizado debido a su simplicidad y efectividad. Con los algoritmos Naive Bayes, es rápido construir modelos y hacer predicciones. La teoría de Beyes se formula a partir de un conjunto de evidencias dadas una hipótesis, Se relaciona con dos aspectos: la probabilidad de la hipótesis antes de la evidencia

P(H) y la probabilidad después de la evidencia P(H|E).

La teoría de Bayes se explica mediante la siguiente ecuación que se muestra en la siguiente Figura 5.

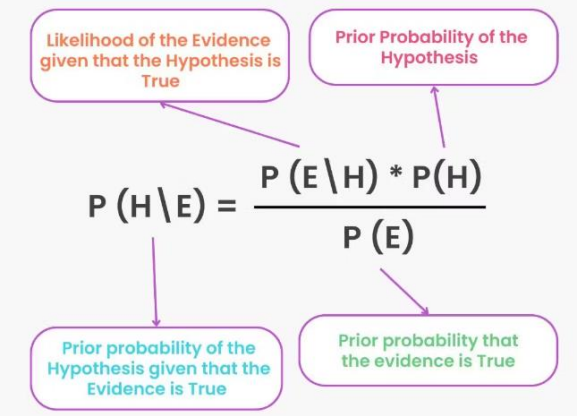

Figura 5 Ecuación de la teoría de bayes (Turing, 2025)

3.1.4 .1 Funciones principales

En bibliotecas de aprendizaje automático como scikit-learn, existen diversas implementaciones del algoritmo Naive Bayes, cada una adaptada a la naturaleza de los datos:

- *GaussianNB:* para atributos continuos que se asumen distribuidos de forma normal.
- *MultinomialNB:* indicado para variables discretas, como el conteo de palabras en análisis de texto.
- *BernoulliNB:* diseñado para variables binarias [40].

Las funciones principales que se utilizan en estas implementaciones incluyen:

- *fit(X, y):* para entrenar el modelo con el conjunto de datos de entrenamiento.
- *predict(X):* que permite realizar predicciones sobre nuevas instancias.
- *predict_proba(X):* calcula las probabilidades de pertenencia de cada muestra a las diferentes clases.
- *score(X, y):* que evalúa la precisión del modelo [40].

3.1.4.2 Hiperparámetros principales

Aunque el clasificador Naive Bayes requiere pocos ajustes, ciertos hiperparámetros

pueden influir significativamente en su desempeño:

- *alpha:* presente en las variantes MultinomialNB y BernoulliNB, este parámetro controla la cantidad de suavizado de Laplace o Lidstone que se aplica para evitar probabilidades nulas. Por ejemplo, un valor de *alpha* igual a 1.0 corresponde al suavizado estándar de Laplace [41].
- *fit_prior*: define si las probabilidades a priori de las clases deben aprenderse automáticamente a partir de los datos o si se proporcionan manualmente. Activar esta opción puede mejorar los resultados cuando se trabaja con conjuntos de datos desbalanceados.
- *var_smoothing:* ajusta la estabilidad de la varianza mediante la adición de una pequeña constante a las varianzas calculadas, lo que contribuye a reducir el riesgo de sobreajuste.

3.1.4 .3 Ventajas del algoritmo

Acentuación se presentara algunas de las principales ventajas que nos ofrece el algoritmo de Naive Bayes [42].

- No requiere grandes cantidades de daos de entrenamiento.
- Es fácil de implementar.
- La convergencia es mas rápida que otros modelos que son discriminativos.
- Es altamente escalable con varios puntos de datos y predictores.
- No es sensible a datos irrelevantes y utiliza predicciones en tiempo real.

3.1.4.4 Limitaciones del modelo

A pesar de sus ventajas, el algoritmo Naive Bayes presenta algunas limitaciones importantes:

- Su rendimiento depende en gran medida del supuesto de independencia condicional entre atributos, una condición que rara vez se cumple en la práctica, lo que puede afectar la precisión del modelo [50].

- Además, es sensible a la presencia de atributos correlacionados, lo que puede llevar a una sobreponderación de ciertas evidencias y afectar las predicciones.

En la siguiente Figura 6 se resume las limitaciones del algoritmo de Naive Bayes

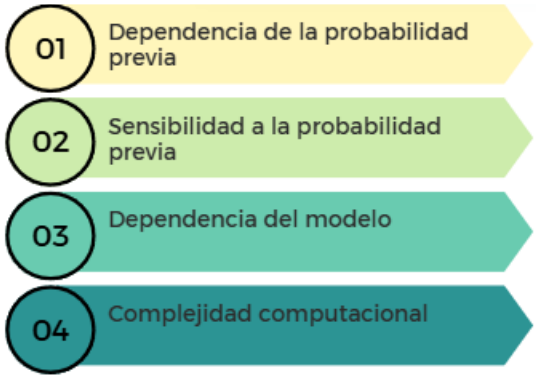

Figura 6 Limitaciones algoritmo de Naive Bayes (FasterCapital, 2025)

3.1.4 .5 Interpretación del modelo

La probabilidad es la base del algoritmo Naive Bayes, ya que los resultados obtenidos a través de la probabilidad que ofrece para problemas más complejos mediante la predicción.
La probabilidad bayesiana permite calcular las probabilidades condicionales. Permite utilizar conocimiento parcial para calcular la probabilidad de ocurrencia de un evento específico. Este algoritmo se utiliza para el desarrollo de modelos de predicción y problemas de clasificación como Naive Bayes.

Así mismo es importante mencionar que existen datos de entrenamiento para entrenar el modelo para así hacerlo funcional. Para continuar con la validación de los datos con el fin de evaluar el modelo y generar nuevas predicciones. Se seleccionaron atributos de entrada que se utilizan como evidencia, asignándoles etiquetas correspondientes como salidas en el conjunto de datos de entrenamiento. A continuación, se muestra un diagrama del teorema de bayes para mejorar comprensión de lo antes mencionado, se muestra que el algoritmo requiere de probabilidad, datos y un estudio previo para poder implementar el algoritmo para asi

obtener una distribución posterior Figura 7 [42].

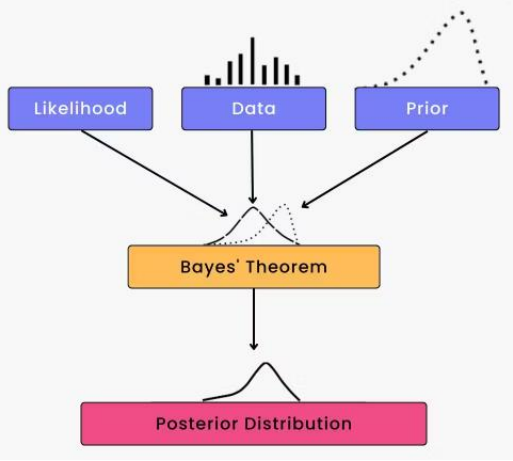

Figura 7 Diagrama de Teorema de Bayes (Turing, 2025)

3.2 Tecnologías

En este proyecto, se emplearon principalmente los lenguajes de programación Python y R, ambos ampliamente reconocidos en la comunidad científica por su versatilidad y potencia en tareas de análisis de datos, modelado estadístico y desarrollo de soluciones basadas en machine learning.

La herramienta principal para utilizar para la implementación de los algoritmos de clasificación fue Python debido a su léxico sencillo y su útil ecosistema de bibliotecas entre las que destacan Scikit-learn, Pandas, NumPy y Matplotlib. Además, su alto enfoque centrado en el desarrollo de modelos de aprendizaje automático permitió construir flujos de trabajo eficientes que abarcan la limpieza y transformación de datos, la validación cruzada y la evaluación de los modelos, todo mediante la integración cualidades pertinentes en sistemas IDS [31]. En contraparte, la herramienta R fue utilizada para exploración preliminar de los datos y visualización estadística. R es una herramienta alineada con la estadística computacional con paquetes afamados como ggplot2, dplyr y caret. Además, su integración con entorno de desarrollo RStudio le otorga la capacidad de realizar análisis interactivos y reproducibles, lo que permitió atenuar el análisis exploratorio por varios lugares [60]. Aquí hubo una robusta implementación de diferentes tecnologías que facilitaron el entrenamiento y la comparación de los algoritmos de clasificación para la detección

de intrusiones en redes.

3.2.1 Paquete estadístico del lenguaje de programación R

La herramienta estadística utilizada en los diferentes análisis que se ha llevado a cabo en este trabajo de fin de grado ha sido R, El lenguaje de programación de R es un programa de computadora que se utiliza en el campo de investigación estadística de asuntos relacionados con la minería de datos, la biomedicina, la bioinformática y las matemáticas financieras. R es una forma corta de leer el sistema GNU Project. Este es un esfuerzo colaborativo. Además, está formado por un amplio conjunto de *paquetes* que permiten utilizar una amplia gama de herramientas estadísticas como modelos lineales, no lineales, análisis de series temporales, algoritmos de clasificación [18].
Otra gran herramienta es la posibilidad de realizar gráficas muy completas. Todas las librerías se obtienen de la página web oficial de *CRAN projects*, sin embargo, se ha utlizado *Rstudio* que es una *IDE (Integrated Development Environment)* que facilita el manejo del entorno de R.

3.2.1. 1 Funciones y parámetros relevantes en preprocesamiento

- *readLines():* Esta función permite leer archivos línea por línea, siendo especialmente útil para filtrar metadatos o eliminar líneas no deseadas antes de realizar una importación estructurada [43].
- read.table(): Utilizada para cargar datos tabulares, cuenta con parámetros como *sep* para definir el separador de columnas, *header* para indicar la presencia de encabezados, y *na.strings* para identificar valores faltantes.
- *na.omit():* Esta función elimina filas que contienen valores nulos (NA), asegurando que el conjunto de datos esté limpio para los análisis posteriores [44].
- *mutate(across()):* Parte del paquete *dplyr*, permite transformar múltiples columnas al mismo tiempo, por ejemplo, para convertir variables categóricas en factores, lo que facilita su manejo estadístico [43].

- *filter():* Se utiliza para filtrar filas basándose en condiciones específicas, como la exclusión de valores atípicos detectados mediante el método del rango intercuartílico (IQR) [44].

3.2.1. 1 Ventajas de utilizar R en el preprocesamiento

- *Amplio ecosistema de paquetes*: Ya que cuenta con colecciones tales como *tidyverse*, R proporciona funciones que facilitan la limpieza y transformación de datos mediante una sintaxis clara y eficiente [43].
- *Integrada visualización:* La capacidad de hacer gráficos ayuda a identificar problemas de la calidad de los datos, como la falta o el valor de los valores atípicos, antes de comenzar su modelado [45].
- *Reproducibilidad*: La posibilidad de volver a ejecutar un análisis completo en cualquier punto: Redundancia de scripts para cada paso de preprocesamiento, proporcionando transparencia y, en consecuencia, una oportunidad de reproducir todos los análisis [46].
- Ser capaz de ser procesado por otras plataformas: Los datos tratados en R pueden exportarse fácilmente en un formato común, como CSV, por lo tanto, utilizarse en el proceso de aprendizaje automático en un entorno como Python.

3.2.2 Python

El Lenguaje de pogramación elegido para escribir los algoritmos de clasificación y realizar otros cálculos fue Python. Se empleó Python porque tiene una sintaxis fácil de entregar y es uno de los lenguajes más comúnmente utilizados en el aprendizaje automático de datos. Tiene muchas librerías y está bien documentado. Las bibliotecas que se utilizan en la programación de Python son *scikit-learn*, que proporciona un conjunto de sencillez para la creación de modelos. *Pandas y NumPy* son bicliotecas donde se utiliza un conjunto, funciones y *dataframe* en arrays de manera funcional. Python fie útil para comparar la exactitud y comparación de los algoritmos KNN y Bayes mediante la implementación del algoritmo decisión forestal

y una técnica de bosque aleatorio. Python es conocido como uno de los lenguajes utilizados en ciencia de datos y aprendizaje automático [47].

3.2.2.1 Funciones y parámetros relevantes en el entrenamiento de modelos

- *pandas.read_csv():* Función fundamental para importar datos desde archivos CSV, que permite especificar parámetros como *filepath* para la ubicación del archivo, *header* para indicar la presencia de encabezados, y *na_values* para identificar datos faltantes [39].
- *train_test_split() (de sklearn.model_selection):* Utilizada para dividir el conjunto de datos en partes de entrenamiento y prueba, con opciones como *test_size* para definir el tamaño del conjunto de prueba y *random_state* para garantizar la reproducibilidad de la división [31].
- *StandardScaler() (de sklearn.preprocessing):* Esta herramienta ajusta las características numéricas para que tengan media cero y varianza uno, un paso esencial para modelos sensibles a la escala, como SVM y KNN (Geron, 2019).
- *fit() y predict() (en modelos como RandomForestClassifier, SVC, KNeighborsClassifier, GaussianNB):* Estas funciones permiten, respectivamente, entrenar el modelo con los datos disponibles y realizar predicciones sobre nuevos datos (Pedregosa et al., 2011).
- *classification_report() y confusion_matrix() (de sklearn.metrics):* Se emplean para evaluar el desempeño del modelo mediante métricas clave como precisión, recall y F1-score, elementos cruciales para validar la efectividad en la detección de intrusiones (Saito & Rehmsmeier, 2015).

3.2.2.2 Ventajas de utilizar Python para el entrenamiento de modelos

El uso de Python en el entrenamiento de modelos de aprendizaje automático presenta múltiples beneficios que lo posicionan como una de las herramientas más utilizadas en el ámbito del análisis de datos y la inteligencia artificial:

- *Sintaxis accesible y legible*: permite implementar algoritmos complejos de forma ágil y comprensible, razón por la que es una excelente salida para las personas con poca experiencia en programación [46].
- *Bibliotecas especializadas:* Scikit-learn, Pandas y NumPy son optimizadas para manipulación de datos, desarrollo de modelos y la evaluación de solución, por lo que el proceso de experimentación toma menos tiempo [45].
- *La escalabilidad y reproducibilidad:* permite automatizar los procesos de prueba de los modelos desarrollados y el ajuste posterior de los mismos, lo que vuelve la repetición de los experimentos y función iterativa desarrollada mucho menos fastidiosa [48].
- *El soporte de una comunidad activa:* los usuarios y los desarrolladores continúan trabajando juntos proporcionando una actualización constante de las capacidades de los trabajadores, la disponibilidad de la documentación necesaria, tutoriales y foros de consulta online.
- *Capacidad de integración:* Python se adapta con facilidad a otros lenguajes, entornos y tecnologías, lo que permite su implementación en soluciones más amplias, como sistemas integrales de detección de intrusiones [48].

CAPÍTULO 4

Implementación y Análisis de Resultados

4.1 Origen de los Datos

Para realización de investigación se trabajó con el conjunto de datos NSL-KDD ya que muestra las conexiones de red simuladas en un entorno de prueba, cada una con un conjunto de características que describen su comportamiento y con una etiqueta que indica si el tráfico es normal o corresponde a algún tipo de ataque

En esta investigación se trabajó con el conjunto de datos NSL-KDD, una versión mejorada del clásico dataset KDD'99, ampliamente empleado en el desarrollo y evaluación de sistemas de detección de intrusiones. El NSL-KDD fue concebido para superar las severas limitaciones del KDD'99, en cuanto a la cantidad excesiva de registros duplicados y la distribución de clases no balanceada; estas propiedades problemáticas, obstaculizan la evaluación objetiva del rendimiento de los modelos de ML. El conjunto fue desarrollado por el grupo de investigación en la Universidad de Nuevo Brunswick como parte de los esfuerzos generales de ofrecer datos más representativos y útiles para la comunidad de investigación en ciberseguridad. Al mantener la estructura general del KDD'99, el NSL-KDD introduce numerosas mejoras, incluidas la eliminación de redundancia y la organización de ejemplos en clases más uniformes, influyendo así a la evaluación más realista y confiable de los sistemas de detección. En cuanto a la implementación, el conjunto de datos para este primer caso se obtuvo de Kaggle, una importante plataforma de ciencia de datos conocida por su gran repositorio en línea de conjuntos públicos. El datset que se empleó fue la versión compartida para el usuario, vista en el siguiente enlace https://www.kaggle.com/datasets/hassan06/nslkdd

Este conjunto de datos incluye varios archivos, entre ellos KDDTrain+.txt y KDDTest+.txt, los cuales contienen registros etiquetados como normales, los distintos tipos de ataques, tales como DoS, Probe, R2L y U2R. A continuación, se exponen de forma detallada los distintos tipos de ataques, con el propósito de comprender mejor su naturaleza y alcance.

- *DoS* (*denegación de servicio) es una categoría de ataque que agota los recursos de la víctima, lo que la incapacita para atender peticiones legítimas.*

- *Probe* (Exploración) el objetivo de la vigilancia y otros ataques de sondeo es obtener información sobre la víctima remota.
- *R2L* (Remoto a Local) es el acceso no autorizado desde una máquina remota, el atacante accede en una máquina remota y obtiene acceso local a la máquina de la víctima.
- *U2R* (Usuario a Root) es el acceso no autorizado a los privilegios del super usuario local el atacante utiliza una cuenta normal para entrar en el sistema de la víctima y obtener privilegios de root/administrador explotando alguna vulnerabilidad de la víctima.

Cada registro está compuesto por 41 atributos que describen diversas características del tráfico de red, además de una etiqueta de clase que identifica el tipo de comportamiento observado.

El NSL-KDD constituye la base sobre la cual se desarrollaron las etapas de preprocesamiento, entrenamiento y evaluación del modelo propuesto para la detección de intrusiones en redes.

Este conjunto de datos incluye varios archivos, entre ellos KDDTrain+.txt y KDDTest+.txt, los cuales contienen registros etiquetados como normal o como distintos tipos de ataques, tales como DoS, Probe, R2L y U2R. Cada registro está compuesto por 41 atributos que describen diversas características del tráfico de red, además de una etiqueta de clase que identifica el tipo de comportamiento observado.

El NSL-KDD constituye la base sobre la cual se desarrollaron las etapas de preprocesamiento, entrenamiento y evaluación del modelo propuesto para la detección de intrusiones en redes.

4.2 Preprocesamiento de datos

Para esta investigación, el preprocesamiento del conjunto de datos NSL-KDD se llevó a cabo utilizando el entorno de desarrollo RStudio y el lenguaje de programación R. Esta etapa resultó fundamental para asegurar la calidad,

coherencia y preparación adecuada de los datos antes de su uso en los modelos de clasificación aplicados a la detección de intrusos. En el contexto del aprendizaje automático, una correcta limpieza, transformación y normalización de los datos es necesaria para optimizar el rendimiento de los algoritmos [48].

El primer paso fue el análisis y depuración del archivo en *formato.arff* contenido en el conjunto de prueba NSL-KDD. El archivo presentaba metadatos, con frecuencia, representados en líneas que inician con el carácter especial @. Dichos metadatos se eliminaron, permitiendo reforzar la estructura de datos presente en una tabla. El archivo fue convertido a formato.csv como archivo temporal, lo que facilitó la carga de este mediante la función read.table. Para ello, se insertaron etiquetas genéricas a las columnas, V1 a Vn, y, adicionalmente, aquellas líneas que contenían variables categóricas, V2, V3 y V4, fueron definidas como factores, siguiendo la recomendación general para el tratamiento de este tipo de datos en R[49].

Durante la fase de limpieza, se eliminaron todas las observaciones con valores faltantes mediante la función na.omit(), y se aplicó un método de detección de valores atípicos basado en el rango intercuartílico (IQR). Este enfoque permite identificar y excluir valores extremos que se encuentran fuera del rango definido por 1.5 veces la diferencia entre el tercer y el primer cuartil, con el fin de reducir su posible impacto negativo durante el entrenamiento de los modelos [49].

A continuación, se normalizaron los atributos numéricos utilizando la técnica de Min-Max Scaling, que transforma los valores de cada variable para que se ubiquen dentro de un rango de 0 a 1. Esta técnica es especialmente relevante en modelos como KNN y SVM, que son sensibles a la escala de las variables [45].

Finalmente, el conjunto de datos preprocesado fue exportado en formato .csv, permitiendo su integración posterior en entornos de Python para llevar a cabo la fase de entrenamiento y evaluación de los modelos de clasificación.

4.2.1 Limpieza de datos

El proceso de limpieza del conjunto de datos comenzó con la lectura del archivo KDDTest+.arff, el cual incluye una cabecera con metadatos. Estas líneas iniciales, que inician con el carácter “@”, fueron filtradas con el fin de conservar únicamente

las instancias correspondientes a los registros de datos reales.

Una vez depurado el contenido, se procedió a convertir el archivo a un formato CSV temporal, lo que facilitó su lectura mediante funciones estándar de R. Posteriormente, el dataset fue cargado e inspeccionado para revisar su estructura, y se asignaron nombres genéricos a las columnas, bajo el esquema V1, V2, ..., Vn, lo cual permitió un tratamiento más uniforme durante las siguientes etapas del preprocesamiento.

El procedimiento implementado en R tiene como propósito transformar el archivo original en un conjunto de datos utilizable para el análisis. El proceso comienza cargando las librerías de soporte y estableciendo la ruta del archivo de entrada. Dado que los ficheros en formato *arff* incluyen información de cabecera que no corresponde directamente a los registros, se realiza una separación: primero se identifican y descartan las líneas de metadatos, y posteriormente se conservan únicamente aquellas que representan datos reales.

El contenido resultante se convierte de forma temporal a un archivo *.csv* para facilitar su manipulación. Una vez cargado, se asignan nombres genéricos a las columnas y se ajusta el tipo de las variables categóricas. Después, se lleva a cabo un tratamiento de calidad: se eliminan filas con valores incompletos y se aplican reglas estadísticas para filtrar observaciones atípicas, utilizando como referencia el rango intercuartílico. En la etapa siguiente, las variables numéricas son escaladas al rango [0,1] mediante la normalización *min–max*, lo cual homogeniza las magnitudes de los atributos. Con estos pasos, el dataset queda limpio y balanceado. Finalmente, se almacena en un archivo *.csv* en el escritorio, junto con un mensaje de confirmación que indica la correcta finalización del proceso.

A continuación, se muestra un fragmento de código 1 que se implementó en esta fase. Si desea verlo con mas detalle continua en el apartado de anexo 1.

Código 1

```
library(tidyverse)
raw_lines <- readLines("C:/Users/wuep/Desktop/KDDTest+.arff")
data_lines <- raw_lines[!grepl("^@", raw_lines)]
```

```
data <- read.table(text = data_lines, sep = ",", header = FALSE, na.strings
= "?", fill = TRUE, strip.white = TRUE)
colnames(data) <- paste0("V", seq_len(ncol(data)))
data <- na.omit(data)
numeric_vars <- data %>% select(where(is.numeric))
data[names(numeric_vars)] <- lapply(numeric_vars, function(x) (x - min(x))
/ (max(x) - min(x)))
write.csv(data, "C:/Users/wuep/Desktop/kddcup_cl.csv", row.names = FALSE)
print("Limpieza de datos completada. Archivo guardado en Desktop.")
```

4.2.2 Eliminación de valores nulos o redundantes.

Durante la fase de inspección, se detectaron valores faltantes identificados por el caracter “?”, registrados como NA en el entorno de R. Con la finalidad de mantener la integridad y cohesionar del dataset, todas las instancias que incluían valores vacíos fueron eliminadas por medio de la *función na.omit().* Esta decisión preventiva se tomó en aras de impedir que registros con datos incompletos que alteren el rendimiento de los modelos en etapas posteriores. El preprocesamiento incluyó el manejo de outliers en las variables numéricas para contrarrestar los posibles sesgos causados por datos extremos. Por ende, se recurrió al método basado en el rango intercuartílico IQR, el cual supone la eliminación de observaciones que difieren del rango comprendido *entre Q1 1.5 × IQR y Q3 + 1.5 × IQR*. Esta técnica permitió limpiar el dataset para promover la estabilidad y la precisión de los algoritmos de aprendizaje automático.

A continuación, se muestra un fragmento de código 2 que se desarrolló en esta fase, así mismo el código completo se encuentra en el apartado de anexo 2. Este programa se encarga de preparar un conjunto de datos antes de trabajar con él. Primero instala y activa las librerías necesarias para poder abrir archivos en formato ARFF, y después carga el archivo desde la carpeta donde se encuentra guardado. Al inicio, muestra cuántos registros contiene el archivo. Posteriormente, realiza una limpieza de la información: elimina las filas con valores vacíos, descarta aquellas que están duplicadas y también filtra valores atípicos utilizando un cálculo basado en el rango intercuartílico. Con esto, se obtiene una

versión más confiable del conjunto de datos. Finalmente, el código compara la cantidad de registros originales con la que queda después de la limpieza, calcula el porcentaje que se eliminó y el que se mantuvo, y lo presenta de manera resumida en la consola.

Código 2

```
# 1. Eliminar valores NA
data <- na.omit(data)
# 2. Eliminar valores duplicados (sin usar dplyr)
data <- data[!duplicated(data), ]
# 3. Filtrar outliers usando Rango Intercuartil (IQR)
numeric_cols <- data[, sapply(data, is.numeric)]   # Seleccionar solo
columnas numéricas
for (col in colnames(numeric_cols)) {
  Q1 <- quantile(data[[col]], 0.25, na.rm = TRUE)
  Q3 <- quantile(data[[col]], 0.75, na.rm = TRUE)
  IQR <- Q3 - Q1
  lower_bound <- Q1 - 1.5 * IQR
  upper_bound <- Q3 + 1.5 * IQR
  data <- data[data[[col]] >= lower_bound & data[[col]] <= upper_bound, ]
}
# Contar registros después de la limpieza
total_final <- nrow(data)
```

4.2.3 Normalización de características

Luego de terminado este proceso de limpieza y purificación del dataset, se procedió con la normalización de las características numéricas. Tal como se explicó en las secciones previas de un flujo o trabajo de aprendizaje automático, la normalización "es un método para modificar la escala de todas las variables a un rango común, es especialmente crítica en algoritmos donde la magnitud de las diferencias entre atributos impacta en el resultado", tales como KNN y SVM. [49].

4.3 Extracción de características

4.3.1 Selección de Características

Otro paso importante en el desarrollo de sistemas de detección de intrusos es la selección de características, que permite reducir la complejidad del conjunto de datos

eliminando atributos redundantes o no relevantes. Además de acelerar el rendimiento computacional, también aumenta la precisión de los modelos predictivos [58] utilizados. En este trabajo, este proceso fue implementado en el lenguaje de programación Python utilizando las bibliotecas especializadas en este sentido, como pandas, numpy y scikit-learn [50].

A continuación, se detallan las etapas implementadas:

1. *Carga de datos:* Se empleó un archivo CSV previamente depurado, que contenía la versión limpia del conjunto de datos.
2. *Depuración de columnas sin valor informativo:* Se detectó la columna V20 como vacía o irrelevante, por lo cual fue eliminada del conjunto de datos.
3. *Reformulación de la variable objetivo:* La columna V42, correspondiente al tipo de conexión, fue codificada de forma binaria, asignando “0” a las conexiones legítimas y “1” a aquellas consideradas intrusivas. Esta transformación simplificó la implementación de algoritmos de clasificación binaria [58].
4. *Separación de atributos y etiquetas:* El conjunto fue dividido en dos partes: una que contiene las variables predictoras (X), y otra que agrupa la variable de salida (y).
5. *Transformación de variables categóricas:* A fin de poder ser procesadas por los modelos, las variables de tipo categórico fueron convertidas a valores numéricos utilizando la técnica de codificación LabelEncoder, que asigna un número entero a cada categoría única [50].
6. *Revisión y tratamiento de datos faltantes:* Se verificó que todos los atributos fueran de tipo numérico y que no existieran valores nulos. En los casos donde se detectaron valores faltantes, estos fueron reemplazados por la media aritmética de cada columna correspondiente.
7. *Estandarización de variables:* Para evitar sesgos debidos a diferentes escalas, las variables numéricas fueron transformadas mediante StandardScaler, técnica que normaliza los datos a una media de cero y una desviación estándar de uno [51].
8. *Reducción de dimensionalidad:* Se aplicaron dos técnicas complementarias:

- *PCA (Principal Component Analysis):* método no supervisado que permitió condensar la información en cinco componentes principales (Jolliffe & Cadima, 2016).
- *LDA (Linear Discriminant Analysis):* técnica supervisada que optimiza la separación entre clases al proyectar los datos sobre una nueva dimensión (Fisher, 1936).

9. *División del conjunto de datos:* Los datos fueron segmentados en un conjunto de entrenamiento (80%) y otro de prueba (20%) utilizando train_test_split, asegurando la representatividad mediante una semilla aleatoria fija [40].
10. *Entrenamiento del modelo:* Se entrenó un clasificador del tipo Random Forest, el cual se basa en un conjunto de árboles de decisión y se configuró con 100 estimadores [23].
11. *Evaluación del modelo:* Una vez entrenado, el modelo fue probado con el conjunto de datos de validación. Se calcularon las métricas de evaluación: precisión, recuperación (recall), puntuación F1 y soporte para cada clase.
12. *Visualización y análisis de resultados:* Los resultados fueron organizados en una tabla resumen, que permite evaluar el desempeño del sistema tanto para conexiones normales como para intentos de intrusión. Además, se calculó la precisión global alcanzada.

Este proceso fue crucial para identificar qué atributos del conjunto de datos aportaban mayor valor al modelo de detección y, al mismo tiempo, permitir una reducción en la carga computacional sin comprometer la eficacia del sistema.

El código carga un archivo CSV con los datos limpios y prepara la variable objetivo V42 para clasificación. Luego divide el conjunto en entrenamiento (70%) y prueba (30%), verificando que ambas clases estén presentes. Con los datos de entrenamiento se ajusta un modelo de Random Forest, y con los de prueba se generan predicciones. Finalmente, se evalúa el rendimiento mediante una **matriz de confusión** y el cálculo de la precisión, además de mostrar la importancia de las variables más relevantes.

A continuación, se muestra un fragmento del código 3 desarrollado si desea verlo

con más detalle continua en el apartado de anexo 3. El código carga el dataset, transforma variables categóricas en numéricas y divide los datos en entrenamiento y prueba. Luego normaliza las características, entrena un modelo de Random Forest y evalúa su desempeño mostrando la precisión, la matriz de confusión y el reporte de clasificación.

Código 3

```
# Cargar librerías necesarias
library(tidyverse)
library(randomForest)
# Cargar y limpiar los datos
data <- read.csv("C:/Users/wuep/Desktop/kddcup_cl.csv")
data_clean <- na.omit(data)
# Definir la variable objetivo como factor
data_clean$V42 <- as.factor(data_clean$V42)
# Dividir datos en entrenamiento (70%) y prueba (30%)
set.seed(123)
train_index <- sample(1:nrow(data_clean), 0.7 * nrow(data_clean))
train_data <- data_clean[train_index, ]
test_data  <- data_clean[-train_index, ]
# Entrenar el modelo Random Forest
model_rf <- randomForest(V42 ~ ., data = train_data, ntree = 100)
# Realizar predicciones en el conjunto de prueba
predictions <- predict(model_rf, newdata = test_data)
# Evaluar el modelo
conf_matrix <- table(Predicted = predictions, Actual = test_data$V42)
print("Matriz de Confusión:")
print(conf_matrix)
accuracy <- sum(diag(conf_matrix)) / sum(conf_matrix)
print(paste("Accuracy:", round(accuracy, 4)))
```

4.4 Entrenamientos de modelos

Durante esta etapa del proyecto, se llevó a cabo el entrenamiento de distintos modelos de aprendizaje automático con la finalidad de analizar y comparar su efectividad en la tarea de detección de intrusiones, utilizando como base el conjunto

de datos procesado NSL-KDD. Para ello, se seleccionaron cuatro algoritmos ampliamente utilizados en el ámbito de la clasificación supervisada: Random Forest, Máquinas de Vectores de Soporte (SVM), K-Nearest Neighbors (KNN) y Naive Bayes.

A fin de garantizar una evaluación justa y consistente, todos los modelos fueron desarrollados siguiendo un mismo procedimiento general.

En el siguiente diagrama se muestra las etapas principales del procesamiento de datos y entrenamiento de los modelos utilizados para la clasificación del tráfico de red. Figura 8.

Figura 8 Diagrama de secuencia de preprocesamiento

4.4.1 Ramdom Forest

El modelo de Ramdom Forest o Bosque Aleatorio es un conjunto de árboles de decisión entrenados sobre subconjuntos aleatorios de los datos. Este enfoque es robusto frente al sobreajuste y ofrece buenos resultados en clasificación binaria.

El modelo se entrenó con 100 árboles (n_estimators=100) y una semilla fija para garantizar la reproducibilidad. Una vez entrenado, se evaluó sobre el conjunto de prueba. Se obtuvo una alta precisión y una buena capacidad de detección tanto para conexiones normales como para intrusiones.

El modelo Random Forest fue incorporado como uno de los clasificadores principales para la detección de intrusiones debido a su alta capacidad para manejar conjuntos

de datos con múltiples características y su robustez ante el sobreajuste. Esta técnica se basa en la construcción de un conjunto de árboles de decisión (conocido como *bagging*) y toma la decisión final mediante votación mayoritaria, lo cual incrementa la precisión y estabilidad del modelo.

Para su implementación se utilizó el lenguaje Python, haciendo uso de bibliotecas como *pandas*, *scikit-learn* y *NumPy*. El conjunto de datos utilizado fue la versión corregida del KDD Cup 99, el cual fue cargado y preprocesado para eliminar una columna vacía (V20) y transformar variables categóricas (V2, V3, V4) mediante codificación *one-hot*. Este paso fue necesario para convertir los datos en un formato numérico compatible con los algoritmos de aprendizaje automático.

Las variables independientes (X) y la variable objetivo (y) fueron separadas, considerando V42 como etiqueta que clasifica las conexiones como "normal" o "anomaly". Posteriormente, los datos se dividieron en subconjuntos de entrenamiento (80%) y prueba (20%) utilizando una semilla aleatoria para garantizar la reproducibilidad de los resultados.

Luego, se aplicó un proceso de estandarización de características sobre las mismas utilizando la clase StandardScaler. Esto se hizo con el propósito de mejorar la eficiencia en el entrenamiento, pero también para mantener la consistencia con los otros modelos que ya estaban implementados. Las entradas se entrenaron utilizando el clasificador RandomForestClassifier, con 100 árboles y una semilla de aleatorización fija. Una vez completado el proceso de entrenamiento, las se recibieron las mismas sobre el conjunto de prueba y se tomaron como métricas de clasificación de rendimiento las clásicas de accuracy, matriz de confusión y reporte que muestra las mismas. Estas métricas, finalmente, permitieron observar cómo el modelo fue capaz de distinguir entre patrones maliciosos y conexiones legales en las entradas de datos.

A continuación, se presenta un fragmento del código Código 4 del algoritmo implementado. Para consultar el código completo, referirse al apartado de Anexo 4. El código carga y prepara el dataset KDD Cup, transformando las variables categóricas y normalizando las numéricas. Luego divide los datos en conjuntos de entrenamiento y prueba para entrenar un modelo Random Forest, y finalmente

evalúa su desempeño mediante precisión, matriz de confusión y reporte de clasificación, diferenciando tráfico normal de anomalías.

Código 4

```
# Cargar y preparar datos
df = pd.read_csv(r"C:\Users\User\Desktop\kddcup_cl.csv")
df = df.drop(columns=['V20'])
df = pd.get_dummies(df, columns=['V2', 'V3', 'V4'])
X = df.drop('V42', axis=1)
y = df['V42']
# Dividir datos en entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
# Escalar características
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
# Entrenar modelo Random Forest
modelo = RandomForestClassifier(n_estimators=100, random_state=42)
modelo.fit(X_train, y_train)
# Predecir y evaluar
y_pred = modelo.predict(X_test)
print("Precisión:", accuracy_score(y_test, y_pred))
print("Matriz de Confusión:\n", confusion_matrix(y_test, y_pred))
print("Reporte de Clasificación:\n", classification_report(y_test,
y_pred, target_names=['normal', 'anomaly']))
```

Así mismo es importante enseñar los resultados al ejecutar el algoritmo por lo que en la Figura 9 se visualiza la consola de la ejecución del código. El modelo Random Forest funcionó de la mejor manera al clasificar el tráfico de red porque logró una precisión global de 94%. La detección de tráfico normal fue muy excelente porque el modelo fue muy confiable y presentó un F1-score de 0.96; sin embargo, la identificación de anomalías también fue buena. De hecho, según las métricas que a continuación se presentan, el modelo tenía un F1 score de 0.87. En general, estos valores reflejan la capacidad del modelo para distinguir entre el tráfico normal y el potencialmente perjudicial. Sin embargo, el modelo tuvo dificultades para clasificar correctamente el tipo de ataque porque algunas veces las clasificó como tráfico normal.

```
======= RESTART: C:/Users/User/OneDrive/Documentos/Tesis/Random Forest.py ======
Modelo: Bosque Aleatorio (Random Forest)
Precisión: 0.9405940594059405
Matriz de Confusión:
 [[667  19]
 [ 35 188]]
Reporte de Clasificación:
              precision    recall  f1-score   support

      normal       0.95      0.97      0.96       686
     anomaly       0.91      0.84      0.87       223

    accuracy                           0.94       909
   macro avg       0.93      0.91      0.92       909
weighted avg       0.94      0.94      0.94       909
```

Figura 9 Resultados de ejecución del algoritmo de Random Forest

4.4.2 Support Vector Machines (SVM)

El algoritmo Support Vector Machines fue utilizado como una técnica de clasificación supervisada en la detección de intrusos en redes. Esta categoría de algoritmos se basa en métodos que buscan un hiperplano óptimo que maximiza el margen entre las clases, lo que lo convierte en una solución eficaz en problemas para los cuales las fronteras de decisión no son lineales. Para la implementación, se optó por el kernel radial, que se destaca por su capacidad para modelar relaciones no lineales entre las variables. La implementación se completó en el lenguaje de programación Python y la biblioteca importada fue *scikit-learn*. que se utilizó para el trabajo. La fuente de datos procesados del KDD Cup 99 (limpio) se preprocesó al eliminar la columna vacía V20. Las variables categóricas se convirtieron en numéricas utilizando la codificación one-hot, que es necesaria para utilizar algoritmos de aprendizaje automático. Luego, los datos se separaron en el X y la variable objetivo y. Esta variable es la columna V42, que se utiliza para identificar si una conexión es normal o una anomalía. Luego, los datos X se dividieron en conjuntos de entrenamiento y prueba, donde el primero consta del 80% de los datos y la prueba el 20%. Aquí, se utilizó una semilla fija para garantizar la aleatoriedad. La estandarización de las funciones numéricas se completó con la clase Importación de *StandardScaler*. Este paso es esencial para tales algoritmos como SVM, ya que son sensibles a la escala de las funciones. Luego, se utilizó el algoritmo de SVM en sí. Las plantillas para entrenamiento y predicción son. Clase SVC que se utiliza para el modelo, *scikiti* se utilizó la semiótica para dividir y escalar.

Ya formado el modelo, se hizo la predicción sobre el conjunto prueba y su respectiva evaluación con base a las métricas accuracy, matriz de confusión y el reporte de clasificación. Las métricas mencionadas permitieron hacer un juicio de valor del desempeño del clasificador en la tarea de detectar tráfico malicioso, demostrando su habilidad para discriminar la clase adversaria frente a las conexiones legales.

A continuación, se muestra un fragmento del código Código 5 utilizado para entrenar y evaluar el modelo Máquina de Vectores de Soporte (SVM). Para consultar el código completo y otros detalles de implementación, favor de referirse al apartado de Anexo 5.

El código realiza la preparación y análisis del dataset KDD Cup para clasificar el tráfico de red usando un modelo SVM. Se eliminan columnas vacías, se codifican variables categóricas y se normalizan las características numéricas. Los datos se dividen en conjuntos de entrenamiento y prueba, se entrena el modelo con un kernel RBF y finalmente se evalúa su rendimiento mediante precisión, matriz de confusión y reporte de clasificación, diferenciando tráfico normal de anomalías.

Código 5

```
# Cargar y preparar datos
df = pd.read_csv(r"C:\Users\User\Desktop\kddcup_cl.csv")
df = df.drop(columns=['V20'])
df = pd.get_dummies(df, columns=['V2', 'V3', 'V4'])
X = df.drop('V42', axis=1)
y = df['V42']
# División de datos en entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
# Escalado de características
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
# Entrenamiento del modelo SVM con kernel RBF
modelo = SVC(kernel='rbf', random_state=42)
modelo.fit(X_train, y_train)
# Predicción y evaluación
y_pred = modelo.predict(X_test)
```

En la Figura 10 se visualiza los resultados de la ejecución del Modelo Máquina de vectores de soporte En la ejecución del modelo de Máquina de Vectores de Soporte

(SVM) se obtuvo una precisión global del 95.8%, lo cual refleja un desempeño sólido en la clasificación del tráfico de red. La matriz de confusión muestra que el modelo logra identificar prácticamente todos los registros normales, mientras que en la detección de anomalías alcanza un 83% de efectividad, presentando algunos falsos negativos. El reporte de clasificación evidencia que la clase *normal* presenta un recall perfecto (1.00), mientras que la clase *anomaly* mantiene una precisión elevada (0.99) y un f1-score aceptable (0.91). En general, los resultados indican que el modelo es altamente confiable para reconocer el comportamiento legítimo en la red, aunque aún puede optimizarse para mejorar la detección de ataques y reducir los falsos negativos.

```
========== RESTART: C:/Users/User/OneDrive/Documer
Modelo: Máquina de Vectores de Soporte (SVM)
Precisión: 0.9581958195819582
Matriz de Confusión:
 [[685   1]
 [ 37 186]]
Reporte de Clasificación:
              precision    recall  f1-score   support

      normal       0.95      1.00      0.97       686
     anomaly       0.99      0.83      0.91       223

    accuracy                           0.96       909
   macro avg       0.97      0.92      0.94       909
weighted avg       0.96      0.96      0.96       909
```

Figura 10 Ejecución del Modelo Maquina de vectores de soporte

4.3.3 K-Nearest Neighbors (KNN)

El algoritmo K-Nearest Neighbors puede encontrarse entre las técnicas de clasificación supervisada que se usaron: KNN. Este clasificador se basa en la suposición de que las instancias similares se encuentran en la misma clase. KNN asigna a una observación una clase que corresponde a la división de la mayoría de las clases extranjeras entre los k vecinos más cercanos a ella, que se mide típicamente mediante la distancia euclidiana. Por lo general, k es igual a 5 o se empieza a experimentar en 5 para problemas de clasificación.

Para su desarrollo se empleó el lenguaje Python, haciendo uso de bibliotecas como

pandas, scikit-learn y NumPy. Se utilizó el conjunto de datos KDD Cup 99 (versión corregida), cargado en un entorno de trabajo adecuado y preprocesado para eliminar la columna vacía V20. Las variables categóricas presentes (V2, V3, V4) fueron transformadas mediante codificación one-hot, con el fin de convertirlas a un formato numérico apto para el algoritmo.

Posteriormente, se realizó la separación entre las variables independientes (X) y la variable objetivo (y), correspondiente a la columna V42, que indica si el tráfico de red es "normal" o representa una "anomalía". Los datos fueron divididos en conjuntos de entrenamiento y prueba, con una proporción de 80% y 20% respectivamente, utilizando una semilla fija para garantizar la reproducibilidad de los resultados.

Dado que KNN es sensible a las escalas de las variables, se estandarizan usando la *clase StandardScaler*. Es crucial llevar a cabo el proceso de normalización para evitar que las distancias en la medida se vean dominadas por variables con valores numéricos superiores. El modelo es entrenado usando la clase *KNeighborsClassifier* de scikit-learn. Se configura el parámetro n_neighbors = 5. Posteriormente, una vez entrenado el modelo, es evaluado usando la precisión, la matriz de confusión y el reporte de clasificación. Estos dos últimos se centran en las clases “normal” y “anomaly”. Dichas pruebas permiten estudiar el comportamiento del modelo en cuanto a su capacidad de detección y clasificación de conexiones maliciosas en el set de test.

A continuación, se presenta un fragmento del código Código 6 utilizado para entrenar y evaluar el modelo K-Vecinos más Cercanos (KNN). Para consultar el código completo y otros detalles de implementación, favor de referirse al apartado de Anexo 6.

El código entrena un modelo de *K-Vecinos más Cercanos (KNN)* para clasificar el tráfico de red. Se preparan los datos eliminando columnas innecesarias, codificando variables y normalizando características. Posteriormente, se entrena el modelo con *k = 5* y se evalúa mediante precisión, matriz de confusión y reporte de clasificación, con el objetivo de diferenciar entre registros normales y anómalos.

Código 6

```
# Cargar y preparar datos
df = pd.read_csv(r"C:\Users\User\Desktop\kddcup_cl.csv")
df = df.drop(columns=['V20'])
df = pd.get_dummies(df, columns=['V2', 'V3', 'V4'])
X = df.drop('V42', axis=1)
y = df['V42']
# División de datos en entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
# Escalado de características
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
# Entrenamiento del modelo K-Vecinos más Cercanos (KNN) con k=5
modelo = KNeighborsClassifier(n_neighbors=5)
modelo.fit(X_train, y_train)
# Predicción y evaluación
y_pred = modelo.predict(X_test)
```

En la Figura 11 se visualiza la ejecución del Modelo K-Vecinos más cercanos. El modelo K-Vecinos más Cercanos alcanzó una precisión global del 94.2%, lo que refleja un buen desempeño en la clasificación del tráfico de red. La matriz de confusión muestra que la mayoría de los registros normales y anómalos fueron correctamente identificados, aunque se presentan algunos falsos negativos en la clase *anomaly*. En el reporte de clasificación se observa que la clase *normal* obtuvo un recall de 0.98, indicando que casi todos los registros normales fueron detectados. Por su parte, la clase *anomaly* logró un recall de 0.84, lo que muestra que algunas anomalías no fueron reconocidas. En conjunto, los resultados confirman que el modelo es efectivo, aunque con margen de mejora en la detección de ataques.

```
========================== RESTART: C:/Users/User/OneDrive/Docum
Modelo: K-Vecinos más Cercanos (KNN)
Precisión: 0.9427942794279428
Matriz de Confusión:
 [[670  16]
 [ 36 187]]
Reporte de Clasificación:
              precision    recall  f1-score   support

      normal       0.95      0.98      0.96       686
     anomaly       0.92      0.84      0.88       223

    accuracy                           0.94       909
   macro avg       0.94      0.91      0.92       909
weighted avg       0.94      0.94      0.94       909
```

Figura 11 Ejecución de Modelos K-Vecinos mas cercanos

4.4.4 Naive Bayes

El algoritmo Naive Bayes es una técnica de clasificación supervisada basada en la aplicación del Teorema de Bayes, ya que sus atributos son estadísticamente independientes entre sí dado el valor de la clase (Murphy, 2006). A pesar de esta suposición simplificadora de ahí el término *naive* o ingenuo, este modelo ha demostrado ser eficaz y robusto en diversos dominios, especialmente cuando se dispone de conjuntos de datos grandes y de alta dimensión (Zhang, 2004).

Naive Bayes es particularmente popular en tareas como la clasificación de textos, detección de spam, análisis de sentimientos y filtrado de correos electrónicos (McCallum & Nigam, 1998). Su bajo costo computacional y su capacidad para manejar grandes volúmenes de datos hacen que sea una opción recurrente como línea base en sistemas de clasificación.

El algoritmo Naive Bayes, específicamente su versión *GaussianNB*, fue implementado como parte del conjunto de modelos supervisados utilizados para la detección de intrusiones. Este modelo se basa en el teorema de Bayes y asume independencia estadística entre las características predictoras. A pesar de esta simplificación, su eficiencia y rapidez lo convierten en una opción sólida para tareas de clasificación en grandes volúmenes de datos, como es el caso de los sistemas de detección de intrusos.

Para su desarrollo, se utilizó el lenguaje de programación Python y las bibliotecas *pandas*, *scikit-learn* y *NumPy*. Inicialmente, se cargó el conjunto de datos KDD Cup

99 (versión corregida), y se procedió a la eliminación de una columna vacía para garantizar la limpieza del conjunto de datos. Posteriormente, se aplicó una codificación *one-hot* a las variables categóricas presentes en los campos V2, V3 y V4, con el fin de convertirlas en un formato numérico adecuado para el modelo.

El conjunto de datos fue segmentado en variables independientes (X) y la variable objetivo (y), correspondiente a la clase V42, que identifica si una conexión es "normal" o representa una "anomalía". A continuación, se realizó una partición del 80% para entrenamiento y el 20% para prueba, con una semilla aleatoria fija para asegurar la reproducibilidad del experimento.

Aunque el algoritmo Naive Bayes no requiere estrictamente la normalización de características, se aplicó una estandarización mediante *StandardScaler* con el objetivo de mantener consistencia con el preprocesamiento realizado en los demás modelos. Posteriormente, se entrenó el modelo utilizando la clase GaussianNB de la biblioteca *scikit-learn*.

Finalmente, se generaron las predicciones y se evaluó el desempeño del clasificador mediante métricas como la *accuracy*, la matriz de confusión y el *classification report*, enfocado en las clases "normal" y "anomaly". Los resultados obtenidos ofrecen una visión clara del comportamiento del modelo frente a patrones normales y maliciosos dentro del tráfico de red.

A continuación, se presenta un fragmento Código 7 representativo del código utilizado para entrenar y evaluar el modelo Naive Bayes (Bayes Ingenuo). Para consultar el código completo y otros detalles de implementación, favor de referirse al apartado de Anexo 7. El código implementa un modelo de clasificación utilizando el algoritmo Naive Bayes para distinguir entre tráfico normal y tráfico anómalo. En primer lugar, después de cargar la base de datos, se elimina la fila sin información. Seguidamente se convierten las variables categóricas en numéricas, dividimos las características y las etiquetas, dividimos los datos de entrenamiento y prueba y normalizamos las características. Luego, entrenamos el modelo, predecimos con nuevas entradas en las características de prueba y evaluamos el rendimiento. Finalmente, evaluamos el rendimiento de la precisión global, la matriz de confusión

y el informe de clasificación.

Código 7

```
# Cargar y preparar datos
df = pd.read_csv(r"C:\Users\User\Desktop\kddcup_cl.csv")
df = df.drop(columns=['V20'])
df = pd.get_dummies(df, columns=['V2', 'V3', 'V4'])
X = df.drop('V42', axis=1)
y = df['V42']
# División de datos en entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
# Escalado de características (opcional para Naive Bayes)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
# Entrenamiento del modelo Naive Bayes
modelo = GaussianNB()
modelo.fit(X_train, y_train)

# Predicción y evaluación
y_pred = modelo.predict(X_test)
```

En la siguiente Figura 12 se muestran los resultados obtenidos en la ejecución del código. El modelo Naive Bayes alcanzó una precisión del 89.3%, mostrando buen desempeño general. Detecta correctamente todas las anomalías (recall = 1.00), aunque clasifica parte del tráfico normal como anómalo. La clase normal tiene alta precisión (1.00) pero recall menor (0.86), mientras que la clase anomalía muestra alta sensibilidad (1.00) pero precisión más baja (0.70). En resumen, el modelo es efectivo para identificar ataques, pero menos confiable para reconocer tráfico legítimo.

```
========== RESTART: C:/Users/User/OneDrive/Documentos
Modelo: Naive Bayes (Bayes Ingenuo)
Precisión: 0.8932893289328933
Matriz de Confusión:
 [[590  96]
 [  1 222]]
Reporte de Clasificación:
              precision    recall  f1-score   support

      normal       1.00      0.86      0.92       686
     anomaly       0.70      1.00      0.82       223

    accuracy                           0.89       909
   macro avg       0.85      0.93      0.87       909
weighted avg       0.92      0.89      0.90       909
```

Figura 12 Ejecución del modelo Naive Bayes

4.4 Métricas de evaluación

Para validar el rendimiento de los modelos de clasificación creados, se emplearon las métricas comunes en problemas de predicción automatizada: precisión, F1-score, record, y el análisis de clasificaciones erróneas o falsos positivos (FP) y falsos negativos (FN). Si bien las métricas lucen como una simple validación numérica del modelo, una que resuma la calidad de las etiquetas pronosticadas, también indican cómo mide el modelo ante problemas y desafíos de conjunto desbalanceado, como la detección de intrusos.

En la Figura 13 se representa la comparación de los valores de accuracy obtenidos para los modelos evaluados. Esta manera de visualizar resultados permite ver diferencias de desempeño entre los clasificadores de una manera clara, destacando el valor superior obtenido por SVM y la posición relativa de KNN, Random Forest y Naive Bayes. Por lo que destacan los siguientes puntos.

- El modelo SVM obtuvo la mayor precisión (95.82%),
- seguido por KNN (94.28%) y Random Forest (94.06%).
- Naive Bayes mostró la menor precisión, alcanzando un 89.33%.

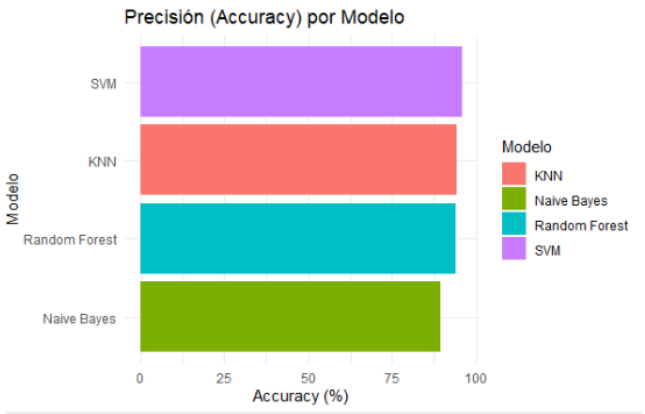

Figura 13 Precisión por Modelo

Estos resultados sugieren que el algoritmo SVM es el más eficaz en términos generales, mientras que Naive Bayes presenta limitaciones en la clasificación global

del conjunto de datos.

4.4.1 F1-Score para la clase "Anomaly"

El *F1-score* es la media armónica entre precisión y recall, y resulta especialmente útil cuando se trabaja con clases desbalanceadas, como en la detección de ataques, donde los falsos negativos pueden ser más costosos que otros errores [46]. En la Figura 14 se comparan los valores de F1-score obtenidos por cada uno de los modelos al evaluar exclusivamente la clase *Anomaly*. La visualización permite apreciar de forma clara las diferencias de rendimiento, resaltando el desempeño superior del modelo SVM frente a las demás técnicas por lo que se destaca los siguientes puntos.

- El algoritmo SVM obtiene el mejor balance (F1 = 0.91),
- El algoritmo Naive Bayes presenta el valor más bajo (F1 = 0.82), lo cual indica un mayor desequilibrio entre precisión y recall.

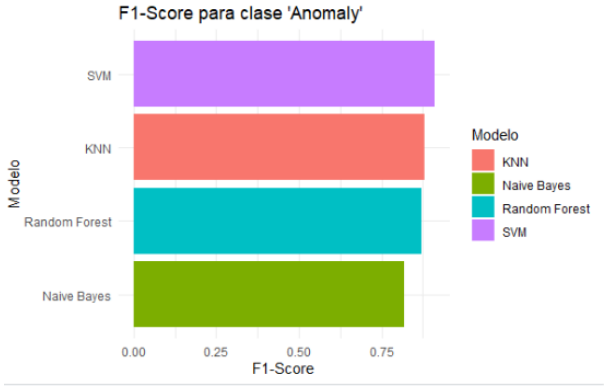

Figura 14 F1 Score para Clase Anomaly

4.4.2 Recall para la clase “Anomaly”

El recall, o retroceso es también sensibilidad o tasa de verdaderos positivos. Es la proporción de instancias positivas, que el modelo clasificó correctamente [52]. En la Figura 15 se muestra los valores de *recall* obtenidos por los modelos evaluados para la detección de la clase *Anomaly*.

- Naive Bayes obtuvo un recall perfecto igual a (1.00), ya que detectó todas las anomalías reales. Sin embargo, su baja precisión significaba muchos falsos positivos. Los modelos Random Forest, SVM y KNN tuvieron valores de recall de 0.83, 0.84, lo que es un buen compromiso entre la tasa de detección y el error.

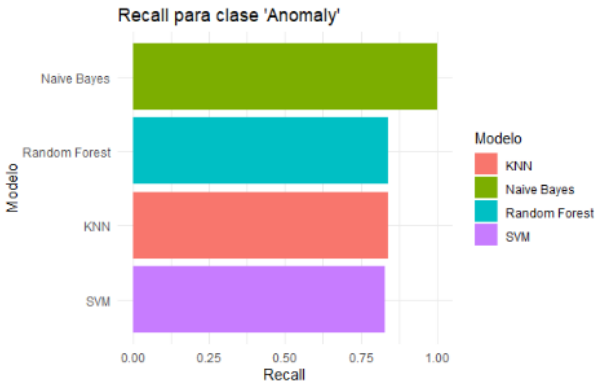

Figura 15 Recall para clase Anomaly

4.5 Resultados de evaluación

Después de seleccionar y completar el entrenamiento de los modelos de clasificación Random Forest, Support Vector Machines, (SVM) K-Nearest Neighbors (KNN) y Naive Bayes, consideramos necesario evaluar su rendimiento sobre el conjunto de prueba. Observamos cómo los modelos trabajan con las métricas comúnmente utilizadas en el ámbito del aprendizaje automático, como la precisión general y la puntuación F1 y recall y para los criterios de error de clasificación que toma la forma de falsos positivos y falsos negativos. Finalmente, en el siguiente paso, los presentamos como visualizaciones comparativas para facilitar la comprensión:

4.5.1 Precisión General "Accuracy"

En términos de la métrica, la evaluación inicial se realizó utilizando la métrica accuracy, que se define como el porcentaje de instancias clasificadas correctamente en el total de instancias clasificables. Si bien los resultados se presentan en la Tabla 15 que podrás visualizar más adelante, el modelo SVM presentó la precisión más

alta de 95.82% y fue el clasificador más eficiente. Lo siguieron KNN y Random Forest con 94.28% y 94.06%, respectivamente. Aunque el algoritmo Naive Bayes, a pesar de su bajo impacto computacional, obtiene 89.33%, es el de peor rendimiento entre los modelos entrenados. Esto podría deberse a la limitación del condicional independientemente de los atributos. Por tanto, estos resultados indican que el SVM muestra la mayor capacidad para generalizar entre los sistemas evaluados. Al mismo tiempo, Naive Bayes sería menos efectivo con datos que no demostrarían la independencia condicional entre atributos.

4.5.2 Evaluación del F1-Score para la Clase (Anomaly)

Sin embargo, dada la inherente naturaleza desequilibrada de los datos, donde las instancias normales superan el número de instancias anómalas por mucho, se decidió calcular el F1-score específicamente para la clase rare. El F1-score es una métrica que armoniza la precisión y la recuperación, y es especialmente relevante en el campo de la ciberseguridad, donde es crucial para identificar el comportamiento malicioso y minimizar los falsos positivos. Los resultados mostraron que el SVM proporcionó el más alto del F1-score alrededor de 0.91 lo que sugiere un rendimiento sólido en términos de detección y precisiones de predicción. Por otro lado, KNN y Random Forest obtuvieron puntajes de alrededor de 0.87, mientras que Naive Bayes mostró el peor F1-score de 0.82 indicando un desequilibrio entre falsos positivos y verdaderos positivos.

4.5.3 Evaluación del Recall

El recall también denominada sensibilidad, es una medida que describe la proporción de instancias anómalas que el modelo fue capaz de identificar con exactitud. Este score posee una relevancia crítica en el contexto de sistemas de detección de intrusiones ya que un bajo recall significa una gran cantidad de falsos negativos, es decir, detecciones que son ataques y no fueron clasificadas correctamente. El modelo Naive Bayes registró un recall perfecto en la totalidad de las anomalías identificadas, es decir, la totalidad de los casos anómalos fueron registrados como

tal. La contracara de esta calidad es la alta cantidad de falsos positivos también detectados por el modelo, lo que resulta en una mayor cantidad de clasificaciones de anomalía que no fueron corroboradas en las evaluaciones globales. En comparación, los modelos SVM, KNN y Random Forest presentaron valores de recall en el intervalo de 0.83 y 0.84, lo que implica un equilibrio más balanceado entre sensibilidad y precisión.

4.5.4 Análisis de Falsos Positivos y Falsos Negativos

En la Figura 16 se representan los errores de clasificación para cada uno de los modelos junto con las tasas de falsos positivos y falsos negativos. La tasa de falsos negativos es crítica para problemas de ciberseguridad, donde Los FN no detectados (insidias) pueden ingresar al sistema comprometiendo su integridad. El modelo de Naive Bayes solo tuvo 1 FN, pero 96 FP, demostrando un sesgo máximo en la clasificación de instancias como anomalía. Por otro lado, SVM fue el modelo más equilibrado, con solo 1 FP y 37 FN, lo que lo hace una buena elección en caso de incertidumbre. RF y KNN gozaron del desempeño más intermedio, con niveles de error que se considerarían aceptables en aplicación IDS.

En la Figura 16, el error de clasificación producido por cada modelo evaluado se analiza en comparación, distinguiendo entre falsos positivos y falsos negativos. Ese tipo de comparación es crítico en el campo de la ciberseguridad, ya que los FN son esencialmente intrusiones que no fueron identificadas y, por lo tanto, representan un riesgo significativo para la integridad del sistema.

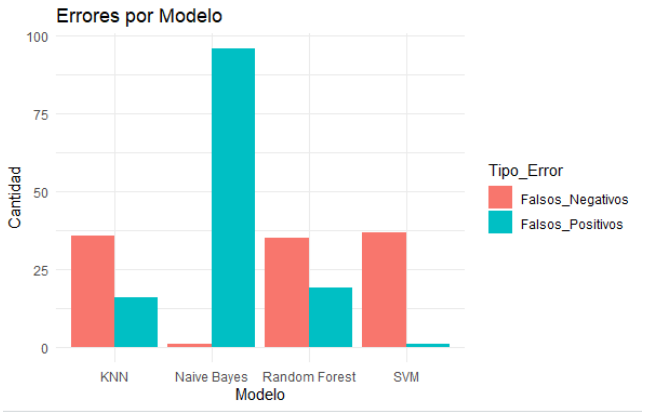

Figura 16 Comparación de Errores por cada Modelo

4.6 Análisis comparativo

A partir de las métricas obtenidas y las gráficas comparativas, se puede observar que, si bien todos los modelos alcanzan un rendimiento aceptable en la detección de intrusiones, presentan diferencias importantes en cuanto al balance entre falsos positivos (FP) y falsos negativos (FN).

El modelo SVM destacó por lograr el mejor equilibrio, con una precisión global de 95.82% y un valor F1 de 0.91 para la clase “Anomaly”, acompañado de 1 falso positivo y 37 falsos negativos. Esto indica que el modelo genera muy pocas alertas falsas y mantiene un nivel razonable de detección de ataques, lo cual lo convierte en el candidato más sólido para una aplicación práctica.

Por el contrario, el modelo Naive Bayes alcanzó un recall de 1.00, detectando todos los ataques, pero con un alto número de 96 falsos positivos, lo que compromete su uso directo en producción. Sin embargo, su alta sensibilidad lo hace ideal como filtro preliminar en un enfoque híbrido.

Los modelos Random Forest y KNN obtuvieron resultados intermedios, con precisiones superiores al 94% y un recall cercano a 0.84, lo que los hace útiles como validadores adicionales o como parte de un esquema de ensamble.

En conjunto, los resultados sugieren la adopción de una estrategia en dos etapas: primero utilizar Naive Bayes como filtro de alta sensibilidad, y posteriormente aplicar SVM como clasificador de precisión para reducir falsos positivos. Este enfoque aprovecha las fortalezas de ambos modelos y garantiza un sistema más confiable en entornos reales.

En la siguiente tabla se mostrará los resultados finales obtenidos de los modelos Tabla 1.

Algoritmo	**Random Forest**	**SVM**	**KNN**	**Naive Bayes**
AccuracyExactitud	94.06%	**95.82%**	94.28%	89.33%
F1 (Anomaly)	0.87	**0.91**	0.88	0.82
Recall (Anomaly)	0.84	0.83	0.84	**1.00**

Falsos Negativos	35	37	36	**1**
Falsos Positivos	19	**1**	16	96

Tabla 1 Resultados obtenidos

4.6.1 Comparación de desempeño global

Los modelos SVM, KNN y Random Forest demostraron un rendimiento competitivo en términos de precisión general, con SVM alcanzando el valor más alto (95.82%). Por lo que demuestra una alta capacidad para clasificar correctamente tanto instancias normales como anómalas. No obstante, al considerar el contexto específico de la detección de intrusos, donde los errores de clasificación tienen consecuencias distintas, es necesario profundizar en otras métricas.

4.6,2 Análisis en la clase minoritaria (Anomaly)

El análisis del *F1-score* para la clase "*Anomaly*" mostró fuertes discrepancias entre los modelos. Aunque ejecutado en una precisión general más alta, SVM también logró el mejor F1-score (0.91), lo que indicaba un compromiso razonable entre la amenaza de no poder detectar y minimizar alarmas falsas. Por otro lado, a pesar de lograr un recuerdo perfecto (1.00), lo que significa que no faltaba una sola anomalía, Naive Bayes demostró un F1-score más bajo (0.82) que se vio limitado por un alto número de falsos positivos. Este patrón pone de manifiesto una debilidad inherentemente ligada de Naive Bayes en contextos donde no se puede suponer la independencia condicional de los datos, algo común en los patrones de tráfico de red. Por otro lado, SVM logró minimizar la frecuencia de ambas clases de errores simultáneamente y, por ende, confirma su resistencia y confiabilidad.

4.6.3 Comportamiento frente a errores críticos

Para los entornos de ciberseguridad, los falsos negativos son extremadamente críticos, ya que son las amenazas no detectadas que pueden llevar al compromiso del sistema. Aunque Naive Bayes combina un número reducido de Falso Negativos (FN), su elevada tasa de Falso Positivos (FP) lo convierte en un predictor ineficiente

para sistemas, donde la generación de alertas incorrectas es operacionalmente costosa. En contraste, SVM ofreció un equilibrio destacado, FC 1 y FN 37, siendo el menos generador de errores. Como se puede ver, Random Forest y KNN tenían un desempeño positivo, pero mostraban una cierta tendencia a generar más errores que SVM, tanto en FPs como en FNs.

4.6.4 Consideraciones prácticas

Desde la perspectiva computacional, Naive Bayes presenta la ventaja de menor carga de entrenamiento, siendo adecuado para entornos donde su propósito sea la rapidez sobre la precisión. Por lo contrario, SVM y Random Forest requieren más tiempo de ejecución y mayor capacidad de procesamiento, pero ofrecen un rendimiento superior cuando la calidad de las predicciones es prioritaria.

4.7 Aplicación web para preprocesamiento

4.7.1 Requerimientos de la Aplicación Web

Para garantizar la correcta ejecución de la aplicación web, el archivo CSV que se desea clasificar debe cumplir con ciertas especificaciones estructurales y de contenido. En primer lugar, el archivo debe estar codificado en formato UTF-8 y contener exactamente 42 columnas con encabezados que sigan el esquema V1, V2, ..., V42, correspondientes a las variables del conjunto de datos NSL-KDD preprocesado.

Entre estas columnas, las posiciones V2, V3 y V4 originalmente representaban variables categóricas (como tipo de protocolo o servicio) que han sido transformadas previamente durante la etapa de limpieza y codificación en el lenguaje R. Además, la columna V42 puede contener la etiqueta de clase (normal o anomaly), aunque esta no es utilizada directamente para la predicción.

El archivo debe estar completamente limpio, es decir, no debe contener valores nulos ni celdas vacías. En particular, se identificó que la columna V20 no aporta información útil, ya que contiene únicamente valores nulos. Esta columna debe ser eliminada durante el preprocesamiento para evitar errores al momento de

escalar las características.
El cumplimiento de estas condiciones es fundamental para que el servicio pueda transformar y escalar correctamente los datos antes de aplicar los modelos KNN o Naive Bayes. De no cumplirse, el servicio puede retornar errores o resultados no válidos.

4.7.2 Procedimiento para el uso de la Aplicación web

La aplicación web desarrollada realiza la clasificación del tráfico de red en tráfico normal o tráfico anómalo mediante dos modelos de aprendizaje automático: K-Vecinos Más Cercanos y Naive Bayes. El sistema se accede y usa a través de una interfaz basada en *FastAPI*, lo cual hace que la interacción con el servidor de clasificación sea fácil y veloz.
El procedimiento de uso es el siguiente:

1. Acceso al sistema
   El usuario ingresa a la dirección web donde se aloja la aplicación. En caso de que se cuente con una interfaz gráfica (HTML) personalizada, esta se muestra al acceder a la raíz del sitio. También se puede interactuar con la API a través de la documentación interactiva disponible en el endpoint /docs.
2. Carga del archivo de datos
   - El usuario debe seleccionar un archivo en formato CSV que contenga registros de tráfico de red previamente preprocesados.
   - El dataset debe contar con 42 columnas nombradas desde V1 hasta V42, en donde las columnas originalmente categóricas (V2, V3 y V4) ya estén codificadas como valores numéricos.
   - Si la columna V20 contiene valores nulos, debe haber sido eliminada previamente.
3. Selección del modelo de clasificación
   Desde la interfaz o mediante el formulario de carga, se elige el modelo de predicción que se desea utilizar:
   - "knn" para el modelo de K-Vecinos Más Cercanos.

- "nb" para el modelo Naive Bayes.

4. Procesamiento y obtención de resultados

   Una vez enviado el archivo y seleccionado el modelo, la aplicación:

   - Preprocesa los datos, asegurando que todas las columnas necesarias estén presentes.
   - Escala los valores utilizando el mismo estándar aplicado en el entrenamiento.
   - Ejecuta el modelo seleccionado para clasificar cada registro. El resultado devuelto incluye:
     - El nombre del modelo utilizado.
     - La lista de predicciones (normal o anómalo).
     - Métricas de rendimiento del modelo (exactitud, F1 para anomalías, recall para anomalías, falsos positivos y falsos negativos).

4.7.3 Desarrollo y pruebas de Aplicación web

El desarrollo de la aplicación se realizó utilizando FastAP*I* como framework principal, lo que permitió implementar un servicio rápido, modular. El flujo general del desarrollo se organizó en las siguientes fases:

1. Preparación y entrenamiento de modelos
   - Se entrenaron dos modelos de clasificación (KNN y Naive Bayes) utilizando datasets de tráfico de red como NSL-KDD.
   - Los modelos se guardaron en formato .pkl mediante *joblib* para su uso posterior.
   - Se entrenó también un escalador para normalizar los datos de entrada, garantizando que la clasificación en producción se realice bajo las mismas condiciones que en el entrenamiento.
2. Diseño de la API
   - Se crearon endpoints en FastAPI, incluyendo uno para la carga de archivos CSV (/clasificar/) y otro para la documentación (/docs).
   - Se definió un proceso de preprocesamiento para asegurar que los datos de entrada cumplan con el formato esperado por el modelo.

3. Implementación del preprocesamiento
    - Se desarrolló una función que:
        - Elimina la columna V20 si contiene valores nulos.
        - Codifica variables categóricas con get_dummies.
        - Garantiza que las columnas estén alineadas con el entrenamiento.
4. Integración y despliegue
    - Los modelos entrenados, el escalador y la lógica de preprocesamiento se integraron en la API.
    - Se configuró la opción de servir una interfaz HTML personalizada mediante archivos estáticos.
5. Pruebas funcionales

    Las pruebas se realizaron cargando datasets de ejemplo con diferentes características para verificar:
    - Que la API rechace archivos con columnas faltantes o datos no codificados.
    - Que el modelo KNN y Naive Bayes devuelvan predicciones consistentes.
    - Que los valores de las métricas se mantengan estables frente a nuevos datos.
6. Validación de resultados

    Finalmente, se verificó que los resultados de la API coincidieran con los obtenidos en el entorno de entrenamiento, confirmando la correcta implementación del preprocesamiento y el uso de los modelos guardados.

A continuación, en la Figura 17 se visualizará el servicio web donde se mostrará la sección de instrucciones del CSV.

Figura 17 La sección de instrucciones del CSV.

En la Figura 18 se muestra el PDF de requerimientos para el uso del Aplicación web

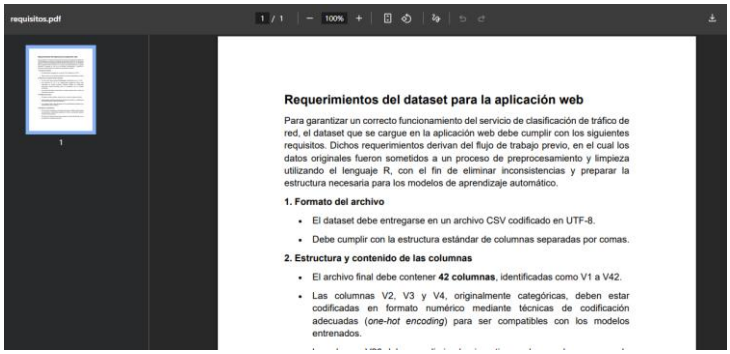

Figura 18 PDF de requerimientos del Aplicación web

En la Figura 19 se muestra la captura de como los usuarios pueden subir archivos CSV a través del formulario web.

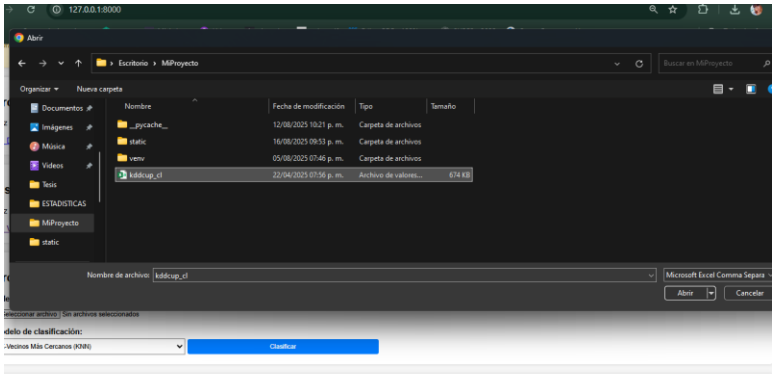

Figura 19 Carga de archivos CVS

En la siguiente Figura 20 se visualiza cuando el usuario selecciona la opción de

Modelo de Naive Bayes

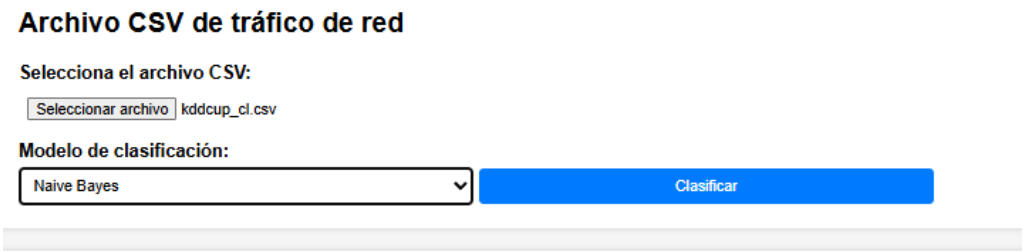

Figura 20 Selección de la opción Naive Bayes

En la siguiente Figura 21 se muestran las predicciones obtenidas en al procesar el modelo.

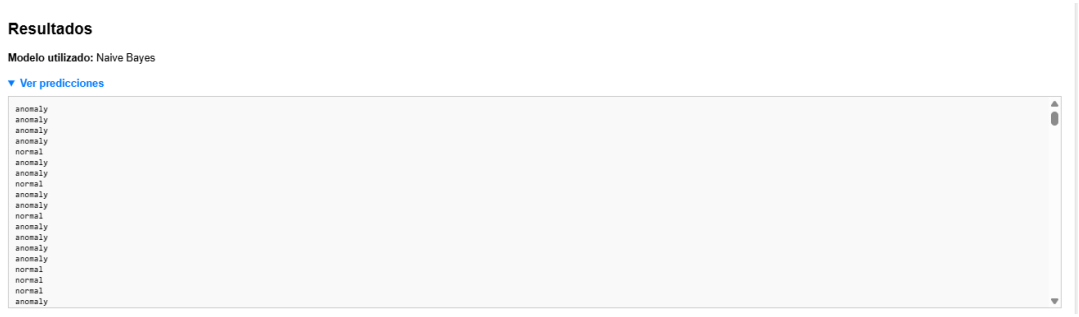

Figura 21 Predicciones del modelo Baive Bayes

A continuación, en la Figura 22 se visualizan los resultados obtenidos al procesar el modelo

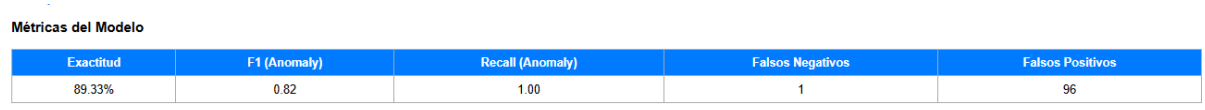

Figura 22 Resultados Naive Bayes

En la siguiente Figura 23 se visualiza cuando el usuario selecciona la opción de procesar el modelo KNN.

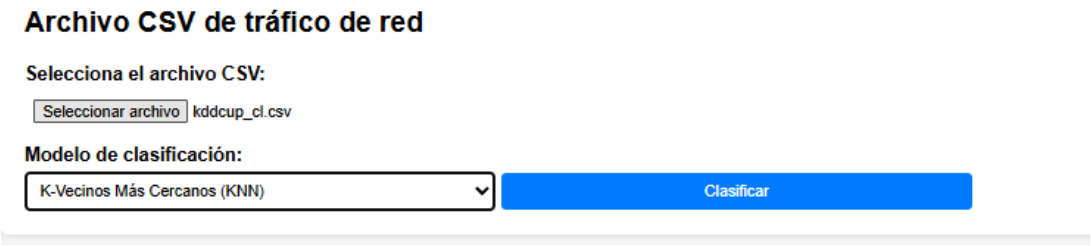

Figura 23 Selección del Modelo KKNN

En siguiente Figura 24 se muestra los resultados obtenidos al procesar el modelo de K-Vecinos Más Cercanos (KNN) en el servicio web. Se observa una lista de predicciones donde cada fila indica si el tráfico de red fue clasificado como “normal” o “anomaly”, permitiendo al usuario identificar patrones de intrusión o comportamiento esperado en la red. La interfaz también ofrece la opción de desplegar u ocultar las predicciones completas, facilitando la revisión de los resultados según sea necesario.

Figura 24 Predicciones del Modelo KNN

En la Figura 25 se presentan los resultados finales obtenidos al completar el proceso de clasificación, permitiendo observar el desempeño del modelo sobre los datos evaluados.

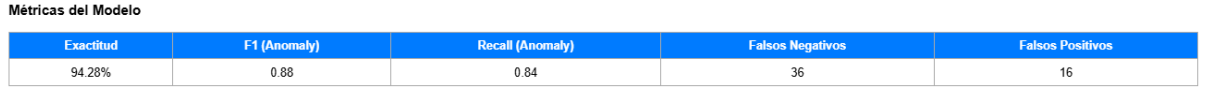

Figura 25 Resultados Finales

CAPÍTULO 5

Conclusiones y Perspectivas Futuras

5.1 Conclusiones generales

Este proyecto de tesis abordó, con un enfoque práctico y académico, cómo la inteligencia artificial puede utilizarse para la detección automatizada de intrusiones en redes informáticas. Con el uso de la base de datos NSL-KDD y la capacitación de modelos supervisados como K Vecinos más Cercanos y Bayes Ingenuo, fue posible desarrollar una solución que pudiera clasificar el tráfico de red como normal o anómalo con un nivel de precisión aceptable.

Según los resultados obtenidos, se observa que ambos son válidos dentro del marco del problema binario. KNN se demuestra eficaz en escenarios en los que los patrones de tráfico son muy unos de otros, mientras que el clasificador Naive Bayes destaca por su velocidad y simplicidad en los casos con buenos datos sindicados. En general, con ambos modelos, pudimos detectar amenazas con buenas métricas de precisión y tasas de falsos positivos y falsos negativos.

Además, como complemento al análisis técnico, se implementó un servicio web interactivo para que los usuarios pudieran cargar archivos de red en formato CSV y obtener de inmediato pronósticos inteligentes utilizando los modelos que se evaluaron aquí. Este trabajo, realizado en FastAPI, presenta un eslabón entre la abstracción teórica y la aplicación práctica que facilita que las herramientas de aprendizaje automático se utilicen en la práctica de la seguridad cibernética. Por lo tanto, en suma, el presente proyecto no solo valida el aprendizaje automático en ciberdefensa sino que también sugiere la viabilidad de soluciones fáciles de usar, precisas y fiables en los escenarios de red del mundo real.

5.2 Limitaciones del trabajo de tesis

Por último, a pesar de los resultados positivos, las limitaciones del trabajo también se destacaron. En primer lugar, solo se utilizó el dataset NSL-KDD. Aunque es un recurso confiable ampliamente utilizado por los investigadores, tiene una estructura estática y no tiene en cuenta todas las condiciones de la red de hoy, como el tráfico cifrado, IoT y los ataques avanzados y persistentes.

Asimismo, el enfoque del modelo fue de clasificación binaria, es decir, distinguir

entre conexiones normales e intrusivas sin identificar el tipo específico de ataque. Esta simplificación ayudó a enfocar el entrenamiento, pero limita la capacidad del sistema para ofrecer diagnósticos más detallados.

Otra restricción fue el tiempo y los recursos computacionales disponibles. Algunos procesos, como la validación cruzada más extensa o el ajuste fino de hiperparámetros, fueron acotados para asegurar la viabilidad del proyecto dentro del periodo establecido.

Por último, aunque se desarrolló un servicio web funcional, su interfaz aún puede mejorarse en aspectos de seguridad, manejo de errores y usabilidad, elementos que son clave en un entorno de producción.

5.3 Trabajo futuro

A partir de esta tesis, se abren varias posibilidades para continuar y enriquecer el desarrollo realizado:

- Extender el modelo hacia una clasificación multicategoría, que no solo detecte la presencia de un ataque, sino que también lo identifique como DoS, R2L, Probe, etc. Esto permitiría una respuesta más precisa y contextual ante las amenazas.
- Actualizar el sistema con nuevos conjuntos de datos, como CICIDS2017 ya que fue desarrollado por el *Canadian Institute for Cybersecurity* (CIC) de la Universidad de New Brunswick (UNB), con el objetivo de proporcionar una base de datos realista y representativa para la evaluación de sistemas de detección de intrusiones (IDS) y prevención de intrusiones (IPS). O el dataset UNSW-NB15 que fue desarrollado por el Cyber Range Lab de la Universidad de Nueva Gales del Sur (UNSW), Australia, con el objetivo de crear un conjunto de datos actualizado que refleje tráfico de red contemporáneo, tanto normal como malicioso, para entrenar y evaluar sistemas de detección de intrusiones (*IDS*)., que incluyen escenarios más recientes y complejos, alineados con las amenazas actuales.

- Integrar otros modelos de aprendizaje profundo, como redes neuronales o enfoques híbridos, que podrían aumentar la capacidad predictiva del sistema sin comprometer la eficiencia.
- Fortalecer el servicio web con una interfaz gráfica completa, autenticación de usuarios, almacenamiento de resultados y alertas en tiempo real, de modo que pueda ser utilizado en entornos empresariales o educativos.
- Finalmente, explorar la incorporación de técnicas como inteligencia artificial explicable (XAI) o sistemas autoentrenables que se adapten continuamente al tráfico real de la red.

Con estas mejoras, se podría consolidar una herramienta robusta y confiable, orientada a contribuir activamente en la protección de infraestructuras digitales frente a ciberamenazas cada vez más sofisticadas.

ANEXOS

En la siguiente sección se muestran los códigos generados para este trabajo con el fin de que le lector puede analizar y observar detalladamente cada uno de ellos.

Anexo 1

En el siguiente código realiza la limpieza de datos obtenidos a través del dataset seleccionado

```
Cargar librerías necesarias
library(tidyverse)
Ruta del archivo (usa doble barra invertida o una sola barra
normal)
file_path <- "C:/Users/wuep/Desktop/KDDTest+.arff"
Verificar las primeras líneas para identificar metadatos
head(readLines(file_path, n = 20))
Saltar las líneas de metadatos manualmente (líneas que comienzan
con "@")
Leer solo las líneas de datos
raw_lines <- readLines(file_path) data_lines <-
raw_lines[!grepl("^@", raw_lines)]
Escribir temporalmente las líneas limpias a un archivo temporal
temp_path <- tempfile(fileext = ".csv") writeLines(data_lines,
temp_path)
Leer los datos (formato CSV)
data <- read.table(temp_path, sep = ",", header = FALSE,
na.strings = "?", fill = TRUE, strip.white = TRUE)
```

```
Verificar estructura del dataset
str(data)
Renombrar columnas como V1, V2, ..., Vn
colnames(data) <- paste0("V", seq_len(ncol(data)))
Verificar nuevamente
str(data)
Convertir columnas categóricas a factor (ajustar según
necesidad)
data <- data %>% mutate(across(c(V2, V3, V4), as.factor)) #
Cambia las columnas si es necesario
Manejo de valores faltantes: eliminar filas con NA
data <- na.omit(data)
Detectar y tratar outliers usando el rango intercuartil (IQR)
numeric_vars <- data %>% select(where(is.numeric))
for (col in colnames(numeric_vars)) { Q1 <- quantile(data[[col]],
0.25, na.rm = TRUE) Q3 <- quantile(data[[col]], 0.75, na.rm =
TRUE) IQR <- Q3 - Q1 lower_bound <- Q1 - 1.5 * IQR upper_bound <-
Q3 + 1.5 * IQR data <- data %>% filter(.data[[col]] >=
lower_bound & .data[[col]] <= upper_bound) }
Normalizar datos numéricos (Min-Max Scaling)
data[names(numeric_vars)] <- lapply(numeric_vars, function(x) (x
- min(x)) / (max(x) - min(x)))
Guardar dataset limpio
write.csv(data, "C:/Users/wuep/Desktop/kddcup_cl.csv", row.names
= FALSE)
Mensaje de confirmación
print("Limpieza de datos completada. Archivo guardado en
Desktop.")
```

Anexo 2

En el siguiente código realiza la eliminación de valores nulos , asi como muestra una comparación final de los datos obtenidos con el dataset de origen

```
# Instalar y cargar paquetes necesarios
install.packages("foreign")   # Para leer archivos .arff
# Cargar librerías
library(foreign)
# Cargar el archivo ARFF
file_path <- "C:\Users\wuep\Desktop\KDDTest+.arff"  # Cambia por
la ruta correcta
data <- read.arff(file_path)
# Contar registros iniciales
total_inicial <- nrow(data)
# --- LIMPIEZA DE DATOS ---
# 1. Eliminar valores NA
data <- na.omit(data)
# 2. Eliminar valores duplicados (sin usar dplyr)
data <- data[!duplicated(data), ]
# 3. Filtrar outliers usando Rango Intercuartil (IQR)
numeric_cols <- data[, sapply(data, is.numeric)]  # Seleccionar
solo columnas numéricas
for (col in colnames(numeric_cols)) {
  Q1 <- quantile(data[[col]], 0.25, na.rm = TRUE)
```

```
  Q3 <- quantile(data[[col]], 0.75, na.rm = TRUE)
  IQR <- Q3 - Q1
  lower_bound <- Q1 - 1.5 * IQR
  upper_bound <- Q3 + 1.5 * IQR
  data <- data[data[[col]] >= lower_bound & data[[col]] <=
upper_bound, ]
}
# Contar registros después de la limpieza
total_final <- nrow(data)
# Calcular porcentaje de mejora
porcentaje_eliminado <- ((total_inicial - total_final) /
total_inicial) * 100
porcentaje_mantenido <- (total_final / total_inicial) * 100
# Mostrar resultados
cat("\nResultados de la limpieza de datos\n")
cat("-------------------------------------------------\n")
cat("Registros iniciales: ", total_inicial, "\n")
cat("Registros después de limpieza: ", total_final, "\n")
cat("Porcentaje de datos eliminados: ",
round(porcentaje_eliminado, 2), "%\n")
cat("Porcentaje de datos mantenidos: ",
round(porcentaje_mantenido, 2), "%\n")
cat("-------------------------------------------------\n")
```

Anexo 3

En el siguiente código realiza la clasificación de características

```
Clasificación de características
Cargar librerías necesarias
library(tidyverse) library(randomForest)
Cargar el archivo CSV limpio
data <- read.csv("C:/Users/wuep/Desktop/kddcup_cl.csv")
Eliminar filas con valores faltantes
data_clean <- na.omit(data)
Verificar cuántas clases hay en V42
print("Distribución de clases en V42:")
print(table(data_clean$V42))
Si solo tienes 'anomaly', es necesario un enfoque diferente
(detección de anomalías)
Convertir V42 a factor (si no es ya un factor)
data_clean$V42 <- as.factor(data_clean$V42)
Verificar que V42 tiene las clases correctas
table(data_clean$V42)
Si solo hay una clase, el modelo no podrá entrenarse. Mostramos
un mensaje de advertencia.
if (length(unique(data_clean$V42)) < 2) { stop("La variable V42
tiene solo una clase. Necesitas más clases para realizar
clasificación.") }
Dividir el dataset en entrenamiento (70%) y prueba (30%)
asegurando que ambas clases estén presentes
set.seed(123) train_index <- sample(1:nrow(data_clean), 0.7 *
nrow(data_clean)) train_data <- data_clean[train_index, ]
test_data <- data_clean[-train_index, ]
Verificar cuántos registros hay en los conjuntos de entrenamiento
y prueba
```

```
print(paste("Filas en entrenamiento:", nrow(train_data)))
print(paste("Filas en prueba:", nrow(test_data)))
Verificar la distribución de clases en el conjunto de
entrenamiento
print("Distribución de clases en el conjunto de entrenamiento:")
print(table(train_data$V42))
Entrenar el modelo Random Forest
model_rf <- randomForest(V42 ~ ., data = train_data, ntree =
100)
Realizar predicciones en el conjunto de prueba
predictions <- predict(model_rf, newdata = test_data)
Evaluar el modelo usando una matriz de confusión
conf_matrix <- table(Predicted = predictions, Actual =
test_data$V42) print("Matriz de Confusión:") print(conf_matrix)
Calcular algunas métricas
accuracy <- sum(diag(conf_matrix)) / sum(conf_matrix)
print(paste("Accuracy:", round(accuracy, 4)))
(Opcional) Mostrar importancia de las variables
print("Importancia de las variables:") importance(model_rf)
varImpPlot(model_rf)
```

Anexo 4

En el siguiente código se realiza la implementación de algoritmo de Ramdom Forest

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing
import StandardScaler
from sklearn.ensemble
 import RandomForestClassifier
from sklearn.metrics
 import classification_report, confusion_matrix, accuracy_score
Cargar el dataset
df = pd.read_csv(r"C:\Users\User\Desktop\kddcup_cl.csv")
Eliminar columna vacía
df = df.drop(columns=['V20'])
Codificar variables categóricas
df = pd.get_dummies(df, columns=['V2', 'V3', 'V4'])
Separar variables independientes y objetivo
X = df.drop('V42', axis=1) y = df['V42']
División en entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
Escalar características
scaler = StandardScaler() X_train = scaler.fit_transform(X_train) X_test =
scaler.transform(X_test)
Modelo Random Forest
modelo = RandomForestClassifier(n_estimators=100, random_state=42)
modelo.fit(X_train, y_train)
#Predicción y_pred = modelo.predict(X_test)
Evaluación
print("Modelo: Bosque Aleatorio (Random Forest)") print("Precisión:",
```

```
accuracy_score(y_test, y_pred)) print("Matriz de Confusión:\n",
confusion_matrix(y_test, y_pred)) print(" Reporte de Clasificación:\n",
classification_report(y_test, y_pred, target_names=['normal',
'anomaly']))
```

Anexo 5

A continuación, se mostrará el código de la implementación del algoritmo support vector Machines (SVM)

```
import pandas as pd
from sklearn.model_selection
import train_test_split from sklearn.preprocessing
import StandardScaler
from sklearn.svm import SVC from sklearn.metrics
import classification_report, confusion_matrix, accuracy_score
Cargar el dataset
df = pd.read_csv(r"C:\Users\User\Desktop\kddcup_cl.csv")
Eliminar columna vacía
df = df.drop(columns=['V20'])
Codificar variables categóricas
df = pd.get_dummies(df, columns=['V2', 'V3', 'V4'])
Separar variables independientes y objetivo
X = df.drop('V42', axis=1) y = df['V42']
División en entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
Escalar características
scaler = StandardScaler() X_train = scaler.fit_transform(X_train) X_test =
scaler.transform(X_test)
Modelo SVM
modelo = SVC(kernel='rbf', random_state=42) # Puedes cambiar el kernel:
'linear', 'poly', 'rbf', 'sigmoid' modelo.fit(X_train, y_train)
Predicción
y_pred = modelo.predict(X_test)
Evaluación
print("Modelo: Máquina de Vectores de Soporte (SVM)") print("Precisión:",
accuracy_score(y_test, y_pred)) print("Matriz de Confusión:\n",
confusion_matrix(y_test, y_pred)) print("Reporte de Clasificación:\n",
classification_report(y_test, y_pred, target_names=['normal',
'anomaly']))
```

Anexo 6

En el siguiente código se implementa el algotritmo Nearest Neighbors (KNN)

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing
import StandardScaler from sklearn.neighbors
import KNeighborsClassifier
from sklearn.metrics
```

```
import classification_report, confusion_matrix, accuracy_score
Cargar el dataset
df = pd.read_csv(r"C:\Users\User\Desktop\kddcup_cl.csv")
Eliminar columna vacía
df = df.drop(columns=['V20'])
Codificar variables categóricas
df = pd.get_dummies(df, columns=['V2', 'V3', 'V4'])
#Separar variables independientes y objetivo X = df.drop('V42', axis=1) y
= df['V42']
División en entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
Escalar características
scaler = StandardScaler() X_train = scaler.fit_transform(X_train) X_test =
scaler.transform(X_test)
Modelo KNN
modelo = KNeighborsClassifier(n_neighbors=5)
# Puedes ajustar el valor de k modelo.fit(X_train, y_train)
#Predicción y_pred = modelo.predict(X_test)
Evaluación
print("Modelo: K-Vecinos más Cercanos (KNN)")
print("Precisión:", accuracy_score(y_test, y_pred))
print("Matriz de Confusión:\n", confusion_matrix(y_test, y_pred))
print("Reporte de Clasificación:\n", classification_report(y_test, y_pred,
target_names=['normal', 'anomaly']))
```

Anexo 7

En el siguiente código se muestra la implementación del algoritmo Nive Bayes

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score
#Cargar el dataset
df = pd.read_csv(r"C:\Users\User\Desktop\kddcup_cl.csv")
#Eliminar columna vacía
df = df.drop(columns=['V20'])
# Codificar variables categóricas
df = pd.get_dummies(df, columns=['V2', 'V3', 'V4'])
#Separar variables independientes y objetivo
X = df.drop('V42', axis=1)
y = df['V42']
# División en entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
#Escalar características (opcional para Naive Bayes, pero se puede
mantener por consistencia)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
#Modelo Naive Bayes
modelo = GaussianNB()
modelo.fit(X_train, y_train)
```

```
#Predicción
y_pred = modelo.predict(X_test)
# Evaluación
print("Modelo: Naive Bayes (Bayes Ingenuo)")
print("Precisión:", accuracy_score(y_test, y_pred))
print("Matriz de Confusión:\n", confusion_matrix(y_test, y_pred))
print("Reporte de Clasificación:\n", classification_report(y_test, y_pred,
target_names=['normal', 'anomaly']))
```

Referencias

[1] Anderson, J. P. (1980). *Computer security threat monitoring and surveillance*. James P. Anderson Company.
[2] Stallings, W. (2017). *Network Security Essentials: Applications and Standards* (6th ed.). Pearson.
[3] Hindy, H., Brosset, D., Bayne, E., Seeam, A., Tachtatzis, C., & Atkinson, R. (2020). A taxonomy and survey of intrusion detection system design techniques, network threats and datasets. *Computer Communications, 160*, 20–40. https://doi.org/10.1016/j.comcom.2020.05.005
[4] Denning, D. E. (1987). An intrusion-detection model. *IEEE Transactions on Software Engineering, SE-13*(2), 222–232. https://doi.org/10.1109/TSE.1987.232894
[5] Gartner. (2021). *Market Guide for Intrusion Detection and Prevention Systems*. Gartner Inc.
[6] OWASP. (2021). *Insider Threats*. Open Web Application Security Project. https://owasp.org/www-project-top-ten/
[7] Debar, H., Dacier, M., & Wespi, A. (1999). Towards a taxonomy of intrusion-detection systems. *Computer Networks, 31*(8), 805–822. https://doi.org/10.1016/S1389-1286(98)00017-6
[8] MITRE. (2021). *Case Study: Cyber Attack on Iran's Natanz Nuclear Facility*. https://www.mitre.org
[9] National Institute of Standards and Technology (NIST). (2018). *Framework for Improving Critical Infrastructure Cybersecurity* (Version 1.1). https://doi.org/10.6028/NIST.CSWP.04162018
[10] SANS Institute. (2021). *Intrusion Detection in the Age of Advanced Persistent Threats*. https://www.sans.org/white-papers/
[11] Liao, H.-J., Lin, C.-H. R., Lin, Y.-C., & Tung, K.-Y. (2013). Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications, 36*(1), 16–24. https://doi.org/10.1016/j.jnca.2012.09.004
[12] Aldwairi, T., & Al-Hammouri, A. (2017). Detecting evasive techniques in intrusion detection systems: Survey and challenges. *Journal of Network and Computer Applications, 80*, 36–57. https://doi.org/10.1016/j.jnca.2016.12.005
[13] Kim, G., Lee, S., & Kim, S. (2016). A novel hybrid intrusion detection method integrating anomaly detection with misuse detection. *Expert Systems with Applications, 41*(4), 1690–1700. https://doi.org/10.1016/j.eswa.2013.08.066
[14] Garcia-Teodoro, P., Diaz-Verdejo, J., Maciá-Fernández, G., & Vázquez, E.

(2009). Anomaly-based network intrusion detection: Techniques, systems and challenges. *Computers & Security, 28*(1–2), 18–28. https://doi.org/10.1016/j.cose.2008.08.003
[15] Rass, S., König, S., & Schauer, S. (2020). Explainable Artificial Intelligence for Intrusion Detection Systems. In *Cyber Security: Analytics, Technology and Automation* (pp. 133–153). Springer. https://doi.org/10.1007/978-3-030-35746-6_8
[16] Sommer, R., & Paxson, V. (2010). Outside the closed world: On using machine learning for network intrusion detection. *IEEE Symposium on Security and Privacy*, 305–316. https://doi.org/10.1109/SP.2010.25
[17] Axelsson, S. (2000). The base-rate fallacy and the difficulty of intrusion detection. *ACM Transactions on Information and System Security (TISSEC), 3*(3), 186–205. https://doi.org/10.1145/357830.357849
[18] Buczak, A. L., & Guven, E. (2016). A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications Surveys & Tutorials, 18*(2), 1153–1176. https://doi.org/10.1109/COMST.2015.2494502
[19] Liu, H., Lang, B., Liu, M., & Yan, H. (2018). CNN and RNN based payload classification methods for attack detection. *Knowledge-Based Systems, 163*, 332–341. https://doi.org/10.1016/j.knosys.2018.10.014
[20] Breiman, L. (2001). Random Forests. *Machine Learning, 45*(1), 5–32. https://doi.org/10.1023/A:1010933404324
[21] Liaw, A., & Wiener, M. (2002). Classification and regression by randomForest. *R News, 2*(3), 18–22. https://cran.r-project.org/doc/Rnews/Rnews_2002-3.pdf
[22] Ho, T. K. (1995). Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition* (Vol. 1, pp. 278–282). IEEE. https://doi.org/10.1109/ICDAR.1995.598994
[23] Biau, G. (2012). Analysis of a random forests model. *The Journal of Machine Learning Research, 13*, 1063–1095. https://www.jmlr.org/papers/volume13/biau12a/biau12a.pdf
[24] Chen, C., Liaw, A., & Breiman, L. (2004). Using Random Forest to Learn Imbalanced Data. University of California, Berkeley. https://statistics.berkeley.edu/sites/default/files/tech-reports/666.pdf
[25] Alenezi, M., Mahmoud, Q. H., & Mahmoud, R. (2020). A Review of Intrusion Detection Systems Using Machine and Deep Learning in Internet of Things: Challenges, Solutions and Future Directions. *Electronics, 9*(7), 1177. https://doi.org/10.3390/electronics9071177
[26] Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning, 20*(3), 273–298. https://doi.org/10.1007/BF00994018
[27] IBM. (s. f.). ¿Qué es el algoritmo de k vecinos más cercanos (KNN)? Think. Recuperado el 8 de agosto de 2025, de https://www.ibm.com/mx-es/think/topics/knn
[28] Srivastava, J. K., & Prakash, J. (2025). Deep learning-enabled energy optimization and intrusion detection for wireless sensor networks. *OPSEARCH, 62*(1), 368–405. https://doi.org/10.1007/s12597-024-00791-z
[29] Azhar, M., Perveen, S., Iqbal, A., & Lee, B. (2024). IDRandom-Forest: Advanced Random Forest for Real-Time Intrusion Detection. *IEEE Access*. https://doi.org/10.1109/ACCESS.2024.3443408
[30] IBM. (s. f.). Introducción a la inteligencia artificial y aprendizaje automático.

Recuperado de https://www.ibm.com/topics/artificial-intelligence
[31] Prasath, R., Kodituwakku, S. R., & Keppitiyagama, C. (2017). The curse of dimensionality in data mining and time series prediction: An analysis. *International Journal of Computer Applications, 165*(3), 1–6.
[32] Real Python. (s. f.). K-Nearest Neighbors (KNN) Algorithm in Python. Recuperado de https://realpython.com
[33] Srivastava, A. (2025). Advanced Concepts in Machine Learning and Pattern Recognition. Springer.
[34] Aggarwal, C. C. (2017). *Data Mining: The Textbook*. Springer.
[35] Han, J., Pei, J., & Kamber, M. (2011). *Data Mining: Concepts and Techniques* (3rd ed.). Elsevier.
[36] Murphy, K. P. (2006). Naive Bayes classifiers. Technical Report, University of British Columbia.
[37] Kennedy, J. (2024). Interpretable Machine Learning: From Theory to Practice. Oxford University Press.
[38] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, É. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research, 12*, 2825–2830. https://www.jmlr.org/papers/v12/pedregosa11a.html
[39] Wickham, H., & Grolemund, G. (2017). *R for Data Science: Import, Tidy, Transform, Visualize, and Model Data*. O'Reilly Media.
[40] Géron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow* (2nd ed.). O'Reilly Media.
[41] Sánchez Gómez-Merelo, M. (2024). Ciberseguridad 2025: Amenazas, desafíos, tendencias y oportunidades. *Seguridad Formación*. https://seguridad-formacion.com/curso-de-ciberseguridad-y-preparacion-del-director-de-seguridad-ciso/
[42] BDO Global. (2024). Principales amenazas para la ciberseguridad y predicciones para 2025. https://www.bdo.global/en-gb/services/advisory/cybersecurity
[43] Jiménez, J. (2016, noviembre 15). Así es cómo el machine learning puede ayudarnos a luchar contra el suicidio, la epidemia silenciosa del siglo XXI. *Xataka*. Recuperado de https://www.xataka.com/medicina-y-salud/asi-es-como-el-machine-learning-puede-ayudarnos-a-luchar-contra-el-suicidio-la-epidemia-silenciosa-del-siglo-xxi
[44] Gutnikov, A., Kupreev, O., & Shmelev, Y. (2022, abril 25). Securelist by Kaspersky. https://securelist.lat/ddos-attacks-in-q12022/96574/
[45] Milenio. (2024, marzo 8). CNSNS sufrió cuatro ciberataques entre 2021 y 2024. https://www.milenio.com/policia/instalaciones-nucleares-de-mexico-sufren-4-ciberataques-desde-2021
[46] Scarfone, K., & Mell, P. (2007). Guide to Intrusion Detection and Prevention Systems (IDPS) (NIST Special Publication 800-94). National Institute of Standards and Technology. https://doi.org/10.6028/NIST.SP.800-94
[47] Rigaki, M., & Garcia, S. (2020). Adversarial attacks and evasion techniques against deep learning models for network intrusion detection: A survey. *arXiv preprint*

arXiv:2007.11530. https://doi.org/10.48550/arXiv.2007.11530
[48] Arya, P. K., & Gupta, S. (2022). Intrusion detection system with layered approach to Internet of Things. En S. Gupta & G. Gabrani (Eds.), *Internet of Things: Security and Privacy in Cyberspace* (pp. 117–132).
[49] Prasath, M. K., & Perumal, B. (2019). A meta-heuristic Bayesian network classification for intrusion detection. *International Journal of Network Management, 29*, e2047. https://doi.org/10.1002/nem.2047
[50] Arya, M. (2022). Machine Learning Algorithms Explained: K-Nearest Neighbors. *DataScience Journal Press*.
[51] Zhang, H. (2004). The Optimality of Naive Bayes. In *Proceedings of the Seventeenth International Florida Artificial Intelligence Research Society Conference* (pp. 562–567).
[52] McCallum, A., & Nigam, K. (1998). A comparison of event models for Naive Bayes text classification. *AAAI-98 Workshop on Learning for Text Categorization, 752*(1), 41–48.
[53] Rivero Pérez, J. (2014). Técnicas de aprendizaje automático para la detección de intrusos en redes de computadora. *Revista Cubana de Ciencias Informáticas*.
[54] Turing.com. (s. f.). An introduction to Naive Bayes algorithm for beginners. Recuperado de https://www.turing.com/kb/an-introduction-to-naive-bayes-algorithm-for-beginners
[55] McKinney, W. (2018). *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython* (2nd ed.). O'Reilly Media.