UNIVERSIDAD SEÑOR DE SIPÁN

FACULTAD DE INGENIERÍA, ARQUITECTURA Y URBANISMO

ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS

TESIS

ANÁLISIS COMPARATIVO DE ALGORITMOS DE MACHINE LEARNING PARA DETECCIÓN DE MALWARE EN APLICACIONES ANDROID

PARA OPTAR EL TÍTULO PROFESIONAL DE

INGENIERO DE SISTEMAS

Autor(a) (es):

Bach. Montenegro Guerrero Victor Agustin

ORCID: https://orcid.org/0000-0002-9218-7823

Asesor(a):

Mg. Aguinaga Tello Juan Adolfo

ORCID: https://orcid.org/0000-0003-2902-9264

Línea de Investigación:

Infraestructura, Tecnología y Medio Ambiente

Pimentel – Perú 2022

APROBACIÓN DEL JURADO

ANÁLISIS COMPARATIVO DE ALGORITMOS DE MACHINE LEARNING PARA DETECCIÓN DE MALWARE EN APLICACIONES ANDROID

_____

Bach. Montenegro Guerrero Víctor Agustín

Autor

_____

Mg. Aguinaga Tello Juan Adolfo

Asesor

_____ _____

Dr. Tuesta Monteza Víctor Alexci **Mg. Cachay Maco Junior Eugenio**

Presidente de Jurado **Secretario de Jurado**

_____

Mg. Aguinaga Tello Juan Adolfo

Vocal de Jurado

Dedicatorias

Dedicado especialmente a Dios por ser siempre la fuente de toda mi fortaleza, y a su vez permitirme en el camino ir venciendo todos los obstáculos, a mi familia y a las personas que están conmigo en cada momento, y también aquellas que desde lo más alto me cuidan, y siempre serán el apoyo y motivación para lograr cada objetivo trazado, alentándome a cumplir todas mis metas tanto personales como profesionales.

Agradecimientos

Un agradecimiento a cada docente que he conocido en mi carrera universitaria por su paciencia y dedicación, en especial aquellos que siempre buscan sacar lo mejor de nosotros como alumnos, gracias por formar parte de mi desarrollo profesional y académico. Asimismo, agradecer a todas las personas que siempre han confiado en mí y que siempre están brindándome su apoyo incondicional, a pesar de que los resultados no sean siempre los esperados.

.

Resumen

El malware en dispositivos móviles Android es muy frecuente hoy en día, ya que este sistema operativo posee un amplio mercado y es el más popular en este ámbito, cabe señalar que en enero de 2021 se obtuvo 71.93% de dispositivos que cuentan con Android, todas sus aplicaciones se basan en el kernel de Linux, el cual es código abierto permitiendo la creación de aplicaciones de fuentes poco confiables. A través de estos dispositivos los usuarios comparten su información personal, confiando en que estas aplicaciones faciliten algunas tareas como pagos, descarga de música, toma de fotos, etc., sin prever que puede ser vulnerable a los ataques de aplicaciones maliciosas que pueden robar esta información. A pesar de los notables esfuerzos de los proveedores de teléfonos Android y Google para implementar mecanismos de seguridad en el software como Bouncer o Google Play Protect, y también en el hardware como Sansung Knox, los autores de malware siempre han encontrado la forma de eludirlos. En este escenario, las técnicas de aprendizaje automático aplicadas en la detección de malware en conjunto con la elaboración de una base de datos con características dinámicas de estas aplicaciones, ha mostrado resultados sobresalientes, superando las limitaciones de métodos de detección tradicionales basadas en firmas, siendo de gran ayuda para la prevención de delitos informáticos. En esta investigación, se propone usar los modelos de aprendizaje automático Random Forest (RF), Decisión Tree (DT) y k-Nearest Neighbor(k-NN) para la detección de malware utilizando características descriptivas del comportamiento dinámico de un malware basado en el tráfico de flujo de red utilizando enfoque propuesto por NetFlowMeter con una cantidad de 15945 muestras entre malware y goodware. La evaluación de los modelos indica que, en cuanto a exactitud, RF tiene un 96%, DT 91.2% y k-NN 85.4% respectivamente. Lo que demuestra que las características de flujo de red generan una base de datos confiable y que el algoritmo Random Forest es el de mejor desempeño en la identificación de malware en aplicaciones android.

Palabras Clave: Clasificación de malware, malware, aprendizaje automático, seguridad android, flujo de red, NetFlowMeter, aplicaciones android.

Abstract

Malware in Android mobile devices is very frequent nowadays, since this operating system has a large market and is the most popular in this area, it is worth noting that in January 2021 71.93% of devices that have Android were obtained, all its applications are based on the Linux kernel, which is open source allowing the creation of applications from unreliable sources. Through these devices users share their personal information, trusting that these applications facilitate some tasks such as payments, downloading music, taking photos, etc., without foreseeing that it can be vulnerable to attacks from malicious applications that can steal this information. Despite the remarkable efforts of Android phone vendors and Google to implement security mechanisms in software such as Bouncer or Google Play Protect, and also in hardware such as Sansung Knox, malware authors have always found ways to circumvent them. In this scenario, machine learning techniques applied in malware detection in conjunction with the development of a database with dynamic features of these applications, has shown outstanding results, overcoming the limitations of traditional detection methods based on foirmas, which will be of great help for the prevention of computer crimes. In this research, it is proposed to use Random Forest (RF), Decision Tree (DT) and k-Nearest Neighbor(k-NN) machine learning models for malware detection using descriptive features of dynamic behavior of a malware based on network flow traffic using Approach proposed by NetFlowMeter with an amount of 15945 samples between malware and goodware. The evaluation of the models indicates that in terms of accuracy, RF has 96%, DT 91.2% and k-NN 85.4% respectively. Which shows that the network Flow features generate a reliable database and Random Forest algorithm is the best performing algorithm in identifying malware in android applications.

Keywords: Malware classification, malware, malware, machine learning, android security, network flow, NetFlowMeter, android apps.

ÍNDICE

I. INTRODUCCIÓN

1.1. Realidad Problemática.

Actualmente existen grandes innovaciones tecnológicas en dispositivos móviles, grandes empresas apuestan por brindar un mejor servicio en la obtención de los mismos. Estos dispositivos conocidos como teléfonos inteligentes son usados de forma amplia para realizar distintas actividades, ya sea para pagos, recibir y enviar correos, descargar música, tomar fotos, etc., todo esto a través de una aplicación determinada que ha sido previamente instalada, el uso de estas aplicaciones de forma masiva se ven reflejados en las estadísticas de las plataformas Android y IPhone donde hoy en día cuentan con más de cinco millones de aplicaciones. (Syrris & Geneiatakis, 2021)

El sistema operativo Android se basa en Linux, y a su vez es el de mayor popularidad a nivel mundial, teniendo una cuota de mercado en enero del 2021 de 71,93%, pero a diferencia de otros sistemas operativos que se encuentra protegidos por diversas leyes o derechos de autor, Android es gratuito y permite que los desarrolladores puedan contribuir en su plataforma, siendo este el motivo principal para que los ataques de malware sean un gran riesgo. El malware se encuentra programado para interrumpir las operaciones o explotar los datos almacenados, y la forma más común de que el malware pueda ingresar al sistema es a través de la descarga de aplicaciones. (Al Saraha, Rifata, Hossainb, & Narmanc, 2021)

En el año 2020 si dieron a conocer que mensualmente aparecen más de 480 000 nuevas muestras de malware, pero estas cifras sólo reflejan el malware que ha llegado a detectarse más no asegura la cantidad total que existen y a pesar de los grandes esfuerzos de los proveedores de teléfonos Android y Google para poder implementar ciertos mecanismos de seguridad en el software y también a nivel del hardware, lo creadores de malware siempre han encontrado la forma de poder eludirlos, esto debido a que las técnicas de antivirus tradicionales basadas en firmas (huella digital) son ineficaces para la detección de malware que no se encuentra en la base de datos. (Guerra-Manzanares, Bahsi, & Nõmm, 2021)

Una de las soluciones a este avance masivo de creación de malware es poder aplicar técnicas de aprendizaje automático, para de esta forma extraer características representativas de las aplicaciones que ofrece Android, con la finalidad de construir modelos capaces de diferenciar entre una aplicación maliciosa o una aplicación benigna, estas técnicas se agrupan en dos grandes categorías, uno es el análisis estático , que utiliza técnicas de desmontaje, descompilación o el análisis de flujo de datos para la detección del malware, pero la ofuscación de código o el código malicioso que se muestra una vez ejecutada la aplicación no permite detectar de la mejor forma si es o no una aplicación maliciosa. Por otro lado, el análisis dinámico registra automáticamente las características del comportamiento de la aplicación, ya sea en un emulador o en un dispositivo real. Aunque algunas aplicaciones preestablecen el entorno donde se ejecutan, es decir si se ejecutan en un emulador no muestran el código malicioso.

Como se puede determinar existe un esfuerzo necesario por combatir el gran aumento de malware en la plataforma Android, y que incluso la gran mayoría de antivirus probados por AV-Comparatives, que es una empresa de seguridad alemana no pueden bloquear, y mucho menos detectar un gran porcentaje de estas aplicaciones maliciosas, peor aún pueden contener malware o vulnerabilidades explotables. Es por ello que debido a estas limitaciones los antivirus deben buscar la mejora de las técnicas basadas en análisis estático o en monitoreo dinámico, siendo herramientas que analizan una aplicación para determinar si su comportamiento cumple con una política de seguridad, y en esta parte se puede encontrar que los algoritmos de aprendizaje automático de machine learning, es específico aquellos que trabajan con datos etiquetados como son los algoritmos supervisados presentan buenos resultados, ya que son relevantes en la clasificación y detección de estas aplicaciones maliciosas, cabe recalcar que estos resultados varían dependiendo de algunos factores, ya sea el algoritmo, la base de datos o el conjunto de características a considerar, lo que abre una ventana para poder realizar un estudio detallado, un análisis del desempeño de estos algoritmos al momento de detectar un malware y poder así brindar un aporte donde se puede determinar qué algoritmo o qué algoritmos

tienen los mejores resultados, y sirva como precedente para mitigar el aumento y el descontrol en la detección de estas aplicaciones maliciosas.

1.2. Trabajos previos.

Lashkari, Kadir, Gonzalez, Mbah, & Ghorbani, (2017),realizaron la investigación, Towards a Network-Based Framework for Android Malware Detection and Characterization, en Canadá. El tráfico de las redes móviles ha aumentado considerablemente debido a las numerosas aplicaciones siempre conectadas, como por ejemplo las redes sociales. Este aumento de aplicaciones en especial en el sistema Android genera un gran campo de acción para los creadores de código malicioso, Malware; muchas investigaciones que buscan dar solución a detección de estos malware han utilizado el dispositivo virtual Android para crear sus conjuntos de datos y han realizado sus experimentos basándose en la generación de tráfico por los emuladores, lo que no es fiable ni similar a los que sucede en la vida real. Por ello se centra en utilizar el tráfico de red generado por las aplicaciones maliciosas instaladas en un dispositivo Android real, para detectar el malware y etiquetar las aplicaciones de adware, este método muestra una exactitud media de 91,41%, una precisión de 91.24% y unos falsos positivos de 8,5% para cinco clasificadores Random Forest, K – Nearest Neighbor, Decisión Tree, Random Tree y Regresión. Estos resultados generan una buena solución a la detección de malware, y para futuros trabajos se planea incluir características, como la información del sistema y del host.

Arora & Peddoju, (2017), realizaron la investigación, Minimizing Network Traffic Features for Android Mobile Malware Detection, en India. Hay muchos programas maliciosos(Malware) para móviles que se controlan de forma remota , ya que reciben órdenes del servidor y pueden filtrar información del usuario o del dispositivo al servidor, Android es el principal sistema en esta clase de dispositivos, la mayoría de estudios para contrarrestar estos malware y buscar sus detección han utilizado un emulador de Android para capturar el tráfico de red de sus muestras , la limitación de estos estudios basados en emuladores es que no admiten eventos como el envío de SMS, la marcación de un número o el reinicio del teléfono, etc. Por ende, algunos programas maliciosos que

esperan estos eventos pueden no activar su carga útil maliciosa y no considerarse como malware. Es por eso que se propuso analizar el comportamiento de tráfico de red del malware de Android capturando su tráfico de smartphone reales, no desde un emulador, y priorizando las características del tráfico de modo que se utilice un subconjunto de características para su detección en lugar de todo el conjunto de características iniciales. El algoritmo de selección de características propuesto extrae 9 características importantes de un total de 22, lo que proporciona un 95.42% de precisión, y una reducción en el tiempo de entrenamiento y prueba de un 50% y 31% respectivamente. A futuro se busca mejorar analizando el tráfico de red cifrado de las muestras de malware utilizando técnicas como el análisis profundo de paquetes.

Wang, y otros, (2017), realizaron la investigación, TrafficAV: An Effective and Explainable Detection of Mobile Malware Behavior Using Network Traffic, en China. El auge del sistema Android se ve muy perjudicado por la prevalencia de malware para Android, las aplicaciones maliciosas utilizan múltiples métodos para evadir los mecanismos de detección existente en el sistema operativo o en el software antivirus. Sin embargo, los comportamientos de red de malware pueden seguir presentando anomalías no triviales que pueden ser identificadas por detectores avanzados, lo que proporciona una visión aguda de detección de Malware. Es por ese caso que se propuso TrafficAV, un método de identificación y clasificación de malware que explota el tráfico de red para poder detectarlo, ya que casi todos los comportamientos maliciosos del malware se llevan a cabo a través de la interfaz de red. TrafficAV emplea una tecnología de duplicación de tráfico para recoger el tráfico de red generado por las aplicaciones móviles, y el tráfico de red generado se transmite a un servidor para el análisis de los datos, en la parte del servidor se extrae las características del tráfico y luego se utilizan modelos de detección basados en el aprendizaje automático para detectar si la aplicación es maliciosa o no. El algoritmo de aprendizaje a utilizar es Decisión Tree, generando resultados de tasas de detección de 98.16% y 99.65%, esto demuestra que, combinando el algoritmo de aprendizaje automático y el análisis de tráfico, se puede detectar eficazmente el tráfico

malicioso y a su vez el malware, la tarea futura es recoger y analizar más muestras de malware y mejorar continuamente los modelos de detección.

Lashkari, Kadir, Taheri, & Ghorbani, (2018), realizaron la investigación, Toward Developing a Systematic Approach to Generate Benchmark Android Malware Datasets and Classification, en Canadá. Gran parte de las técnicas de detección de malware existentes se pueden clasificar en grupos estáticos y dinámicos, la detección estática es el proceso de encontrar características maliciosas o segmentos de código malicioso en una aplicación sin ejecutarla, mientras que la detección dinámica se realiza monitoreando el comportamiento real de la aplicación que se ejecuta en un sandbox como entorno aislado. El entorno aislado es difícil y hace imposible el trabajo para los investigadores, es por ello que utilizar los conjuntos de datos disponibles es una de las mejores soluciones para el análisis dinámico. Actualmente existen numerosos conjuntos de datos disponibles para el malware de Android, pero todos ellos adolecen de deficiencias y limitaciones, debido a que algunos no poseen una cantidad balanceada de malware o las características que se toman son ejecutadas desde un emulador de Android. Es por eso que se traza como objetivo principal proponer un enfoque sistemático para generar conjunto de datos de Malware para Android utilizando smartphones reales y desarrollar un nuevo conjunto de datos, CiCAndMal2017, ofreciendo 80 características de tráfico de red. El método propuesto muestra una precisión media de 85% y un recall de 88% para tres clasificadores, Random Forest, K-Nearest Neighbor y Decisión Tree. A futuro se busca extraer las características útiles de otra fuente de datos, como volcados de memoria, registros y llamadas a la API.

Taheri, Kadir, & Lashkari, (2019), realizaron la investigación, Extensible Android Malware Detection and Family Classification Using Network Flows and API-Calls, en Canadá. Los investigadores proponen cada año un nuevo marco de análisis de Malware en Android para defenderse de las aplicaciones del malware en el mudo real , pero generar un conjunto completo de datos de malware de Android resulta ser un desafío en el campo de escrutinio de malware, siendo el mayor problema la cantidad de aplicaciones que se evalúan

y los ambientes simuladores en los que se ejecutan para su posterior caracterización , puesto que varios malware ejecutados en emuladores no muestran todas sus características, es por ello que se ha realizado una comparación de los conjuntos de datos de malware con respecto a 15 criterios esenciales para definir las principales deficiencias de las base de datos de investigaciones pasadas, generando así la segunda parte de la base de datos CicAndMal2017 que incluye nuevos conjuntos de características como permisos e intenciones en la detección estática y llamadas a las API en la parte dinámica. En conclusión, se mejoró el rendimiento de clasificación de categorías y familias de malware combinando características dinámicas anteriores con relaciones secuenciales de las llamadas de las API, obteniendo resultados de evaluación de hasta 95,3% de recuperación en la clasificación binaria, el 81% de recuperación en la clasificación por categorías y 61,2% de recuperación en la clasificación por familias. A futuro se piensa ampliar el conjunto de datos de Android con más características y un tamaño de muestra masivo.

Wang, Chen, Yan, Yang, & Peng, (2019), realizaron la investigación, A mobile malware detection method using behavior features in network traffic, en China. El sistema operativo Android ocupa una cuota superior en el mercado de dispositivos móviles, su rápido desarrollo y popularidad han generado también una gran cantidad de código malicioso de malware, diseñados para apuntar a teléfonos inteligentes y tabletas. El malware se está volviendo más cruel y difícil de combatir, ya que las aplicaciones maliciosas utilizan varios métodos para evadir los mecanismos de detección existentes proporcionados por el sistema operativo Android, estos métodos pueden ser la ejecución dinámica, ofuscación de código, reempaquetado o cifrado. Sin embargo, los comportamientos de red de los malware aún pueden presentar anomalías no triviales que pueden ser identificadas con detectores avanzados. Es por ello que se ha propuesto un marco para identificar y clasificar malware ligero a nivel de dispositivo, aprovechando el tráfico de red, se trabajan con dos modelos, el modelo de detección HTTP y el modelo de detección de flujo TCP, este método emplea una tecnología de duplicación de tráfico para recopilar el tráfico de red generado por las aplicaciones móviles y el tráfico de red que se transmite a un servidor

para el análisis de datos, y para la clasificación se combina con el algoritmo de aprendizaje automático C4.5, en una base de datos de 8312 goodware y 5560 malware. Asimismo, se concluye que la evaluación experimental demuestra que al combinar los modelos de aprendizaje automático y análisis de tráfico podemos detectar malware de forma eficaz, generando así una tasa de detección de 97,89%.

Sanz, Lopez, Viegas, & Sanches, (2020), realizaron la investigación, A Lightweight Network-based Android Malware Detection System, en Brasil. Se han propuesto muchos enfoques para caracterizar el tráfico de aplicaciones móviles y también para detectar aplicaciones maliciosas, existen dos enfoques , uno es el análisis estático, que examina el código fuente o binario; y otro es el análisis dinámico, que se basa en monitorear el comportamiento del malware, pero si hablamos del enfoque estático el malware utiliza ofuscación de código para evitar la detección, y en el enfoque dinámico los investigadores suelen realizar análisis en un entorno simulado o restringido, donde muchas veces el malware suele ocultar su comportamiento para evitar ser detectado y además este enfoque no tiene en cuenta la naturaleza restringida de los recursos del dispositivo móvil. Es por ello que se propuso un sistema ligero de detección de malware basado en la red para detectar aplicaciones maliciosas en tiempo de ejecución, teniendo en cuenta técnicas ligeras de aprendizaje automático sin afectar el rendimiento del dispositivo móvil, luego se extraen las características ligeras basadas en el buffer de anillo estático y también tráfico de red en características numéricas inferidas directamente del encabezado del paquete TCP/IP, y por último se trabaja con dos algoritmos ligeros de aprendizaje automático, Adaboost y Random Forest, que son aplicados directamente a la salida del buffer de anillo y que clasifican en tiempo real el comportamiento del tráfico de las APK. Los resultados muestran que ambos algoritmos logran un buen equilibrio entre la clasificación y el rendimiento, y es que con sólo 14 características de paquetes TCP/IP, el prototipo fue capaz de clasificar correctamente más del 90% de las APK con una tasa de falsos positivos por debajo del 3%, también se analizó dos aplicaciones que producen muchos paquetes, y se obtuvo como resultado menos del 10% de los recursos de la

CPU incluso en picos de 90 Mb. Como trabajo futuro, se piensa mejorar el volumen mínimo de tráfico, necesario para clasificar correctamente una aplicación.

Manzano, Meneses, & Leger, (2020), realizaron la investigación, An Empirical Comparison of Supervised Algorithms for Ransomware Identification on Network Traffic, en Chile. En el mercado actual de telefonía móvil, Android es el sistema operativo que tiene más popularidad, siendo el uso de aplicaciones muy esencial en el día a día de cada uno de los usuarios, pero a su vez también se han desarrollado masivamente aplicaciones con código malicioso conocidas como malware , siendo su función principal robar información, claro ejemplo es el Ransomware, el cual se desarrolla principalmente para sistemas operativos capturando los datos del usuario a través de la red y los devuelve previo un pago generado por estos usuarios, este caso conlleva la necesidad de buscar alguna manera de contrarrestar este tipo de malware, una de las técnicas que podría ayudar es analizar el tráfico de red generado por estas aplicaciones a través del aprendizaje automático. Es por ello que en esta investigación se evalúa el rendimiento de tres algoritmos Random Forest, Decision Tree y k-NN, desarrollando como primera instancia la creación de una base de datos con características referentes a los tiempos que generan los flujos y también los paquetes bidireccionales del conjunto de datos de tráfico de red determinado por CICAndMal2017, además para la selección de características relevantes se utiliza el método de correlación de Kendall, y una distribución de 80% de muestras de entrenamiento y 20% muestras de prueba. Los resultados obtenidos evidencian que en cuanto a rendimiento en la identificación de Ransomware, Random Forest obtuvo los valores más altos con un promedio de exactitud de 96% y precisión de 91%, seguido por Decision Tree con 94% de exactitud y precisión. Se concluye que la técnica de análisis de red con algoritmos supervisados demuestra buenos resultados de identificación y detección. A futuro por la naturaleza cambiante de la base de datos, se busca implementar los métodos de aprendizaje profundo.

Agrawal & Trivedi, (2020), realizaron la investigación, Evaluating Machine Learning Classifiers to detect Android Malware, en la India. Para realizar la detección de malware existen métodos convencionales como firmas, recursos y componentes, incluso existen antivirus o escáneres que hacen uso de estos métodos, pero a su vez son incapaces de detectar las amenazas de malware y las nuevas firmas que aparecen. Es por ello que existe la necesidad de desarrollar un sistema más eficiente que pueda detectar el malware actual y nuevo, que ayude a superar las deficiencias de los métodos tradicionales. En esta investigación se propone un enfoque de detección de malware usando clasificadores de aprendizaje automático, para lo cual se genera un nuevo conjunto de datos a través de una fase de des compilación y extracción de características, se ha tenido en cuenta 15508 malware y 4000 goodware , utilizando técnicas de reducción de características para escoger aquellas más relevantes, se ha considerado el 75% de muestras para entrenamiento y el 25% para pruebas, los algoritmos supervisados dentro de una evaluación de esta investigación son k-NN, Random Forest, Decision Tree, SVM linear, Regresion Logistic y Naive Bayes. Los resultados de la experimentación determinan que k-NN, Random Forest y SVM lineal son los que mejores rendimientos poseen obteniendo los siguientes porcentajes 90.9%, 91.2% y 90.3% en cuanto a exactitud, y 88.4%, 87.9%,87.3% en precisión respectivamente. Se concluye que la aparición de métodos de aprendizaje automático genera mejores resultados en comparación con los métodos tradicionales, pero todo depende de un conjunto de datos que contenga características relevantes que ayuden a mejorar la detección.

Razgallah, Khoury, Halle, & Khanmohammdi, (2021), realizaron la investigación, A survey of malware detection in Android apps: Recommendations and perspectives for future research, en Canadá. El software antivirus sigue siendo para los usuarios en general y los usuarios de dispositivos móviles una de las primeras líneas defensivas, pero la gran mayoría de estos antivirus suele no detectar las aplicaciones o carga maliciosa, incluso los mismos antivirus pueden tener malware o vulnerabilidades explotables, debido a esas limitaciones de detección los antivirus deben complementarse con

métodos basados en análisis de forma estática y un monitoreo dinámico de código. Es por ello que se ha hecho un análisis de los principales mecanismos y enfoques para detectar malware en aplicaciones Android, identificando las ventajas y limitaciones de cada uno de ellos y la sugerencia para posibles vías de investigación para avanzar en el conocimiento al respecto. Se concluye también que a pesar que se han propuesto una gran cantidad de soluciones quedan varios desafíos por abordar, especialmente debido a la naturaleza de la rápida evolución de los malware. A su vez las recomendaciones que se ofrecen pueden ayudar a guiar la investigación futura y abordar estos desafíos.

1.3. Teorías relacionadas al tema.

1.3.1. Android

Android se ha convertido en una plataforma que ha generado una revolución en el mercado de telefonía móvil a nivel mundial, siendo la primera plataforma de aplicaciones móviles de código abierto que ha iniciado un cambio masivo dentro de este rubro, es por ello que cuando se habla de Android, hay que tener en cuenta una serie de dimensiones técnicas y de mercado (Ableson, Sen, King, & Ortiz, 2012). Sintetizando un concepto fundamental podemos decir que “Android es una plataforma completa de código abierto diseñada para dispositivos móviles” (Gargenta, 2011), y funcionalmente el sistema operativo. Android ha sido desarrollado por Google y está basado en el kernel de Linux, la cual se encarga de la conectividad con el hardware y la funcionalidad básica del sistema operativo, todo diseñado por la arquitectura Avanzada de Máquina RISC (ARM) (Koli, 2018).

1.3.2. Arquitectura de Android

La arquitectura de Android se basa en un entorno de software que ha sido creado para los dispositivos móviles, donde se observa que en su sistema operativo se encuentra como base el núcleo de Linux, las bibliotecas de códigos, la interfaz y aplicaciones de usuario, marcos, soporte multimedia, etc. Además, todos los componentes están escritos en C o C++, las aplicaciones de usuario y las incorporadas están construidas para Android en Java con la excepción de algunos ejercicios exploratorios Linux y el Nativo Developer Kit (NDK), las

cuales usan la directiva de desarrollo de software para Android (SDK). Así mismo se considera como una característica de la plataforma Android que las aplicaciones incorporadas no tienen diferencia alguna y las que se son creadas con el SDK, esto significa que se puede sacar provecho de todos los recursos disponibles que se encuentran en el dispositivo para escribir aplicaciones.

El detalle de la arquitectura de Android se detalla en la Figura 1.

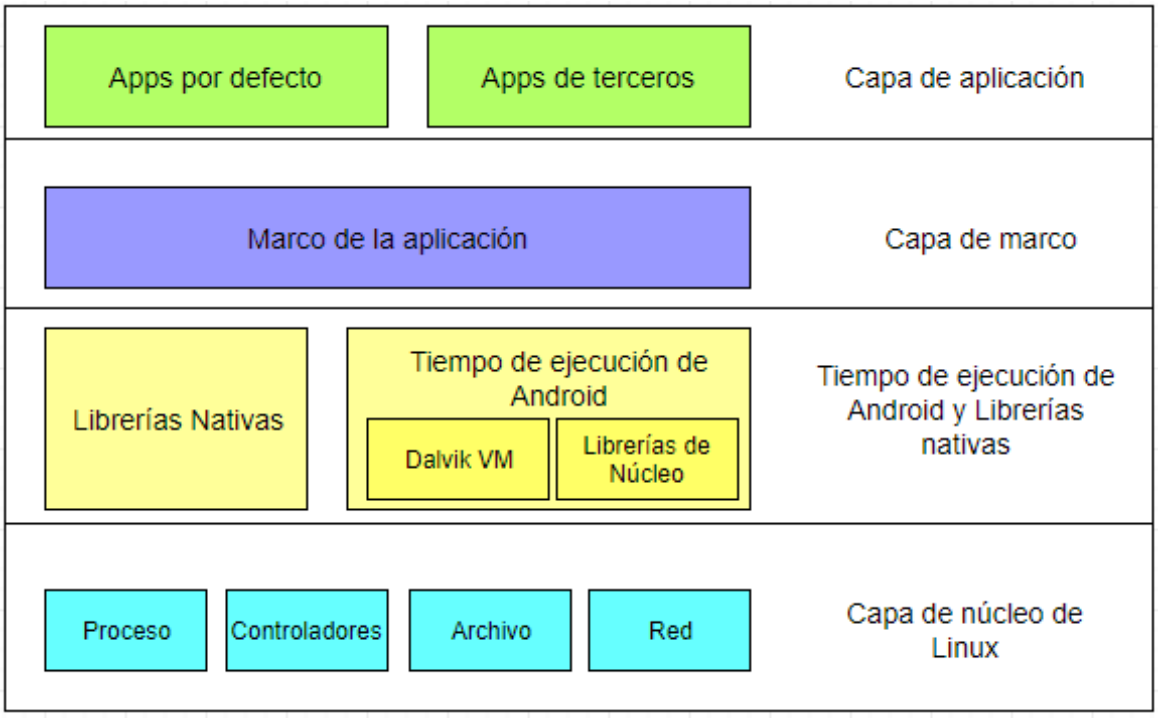

Figura 1. Arquitectura del Sistema Operativo Android.

Fuente: (Koli, 2018)

Seguidamente se describe cada una de las capas que componen la arquitectura:

La primera capa, la capa del núcleo de Linux, es la más importante y se encuentra en la parte inferior; es responsable de la abstracción del hardware y controladores, seguridad, gestión de archivos, procesos y la gestión de la memoria.

La segunda capa, consiste en bibliotecas nativas y tiempo de ejecución de Android. Así mismo las Bibliotecas Nativas están escritas en C/C++ y están expuestas a la estructura de la aplicación y al tiempo de ejecución de Android, y el DVM (Dalvik Virtual Máquina), es una máquina virtual móvil optimizada y es

una instancia del tiempo de ejecución de Android, donde se ejecuta una aplicación.

En la tercera capa, el marco de aplicación facilita la aplicación para acceder a las bibliotecas nativas a través de sus diferentes paquetes.

Por último, la capa de aplicación, que es la capa más alta, donde las funciones del teléfono se proporcionan al usuario final, esta capa consiste en las aplicaciones de Google y la de otros desarrolladores Android.

La aplicación Android tiene una extensión apk, que son las siglas de Android package. Es básicamente un fichero de archivo que contiene todos los archivos y carpetas necesarios en una aplicación, cada aplicación se divide en cuatro componentes principales como se aprecia en la Tabla 1, las cuales son: Actividades, Servicio, Receptores de radiodifusión y Proveedor de contenidos.

Tabla 1.

Componentes de las aplicaciones Android

Componente de la Aplicación	**Descripción**
Actividad	Representa la interfaz de usuario de la aplicación.
Servicio	Funciona como procesos de fondo.
Receptor de emisiones	Responde a los mensajes de difusión. que provienen de otras aplicaciones o del sistema.
Proveedor de contenidos	Es un contenedor de base de datos.

Nota: Tomado de la investigación de (Koli, 2018).

a) La actividad es esencialmente una parte de la interfaz de usuario (UI), así que cualquier pantalla visual que permita al usuario ver e interactuar con una aplicación Android, eso puede consistir en vistas como Vista de botones, Texto ver Vista de tabla, etc.

b) Intención Recibir, que es una manera para que una aplicación pueda registrar algún código que no se ejecuta hasta que se dispara por algún evento externo. El desarrollador puede escribir algo de código a través de XML y registrarlo para

que funcione cuando ocurra algo, por ejemplo, la conectividad de la red se establece a una cierta hora o cuando suena el teléfono.

c) El servicio es una tarea que no tiene ninguna interfaz de usuario, es un componente que se ejecuta en el fondo. Por ejemplo, al almorzar con una música la primera pantalla es una actividad, pero tan pronto como se selecciona una canción para reproducir y pasar a otra aplicación, el servicio sigue corriendo en el directorio fondo.

d) El proveedor de contenido es el almacenamiento de datos, que permite que las aplicaciones puedan compartir los datos con otra aplicación.

1.3.3. Modelo de seguridad de Android

Toda la idea detrás de la plataforma móvil es el hecho de que el usuario puede ejecutar muchas aplicaciones diferentes en el dispositivo. El usuario puede estar instalando y descargando una aplicación bancaria, en donde ingresa algunos datos confidenciales y a su vez, también estar instalando una aplicación de juego justo en paralelo con la aplicación anterior y ejecutándose en el mismo dispositivo, el usuario obviamente no quiere la aplicación del juego para poder acceder a los datos confidenciales que se encuentran en operación de la aplicación bancaria. Entonces para lograr esto, la plataforma Android se asegura de que cualquier aplicación esté aislada en forma individual, básicamente cuando el usuario descarga e instala una aplicación, se le da un UID único. Además, cada aplicación se ejecuta en un proceso separado en una máquina virtual. Por lo tanto, la aplicación no puede leer otra aplicación de datos privados, ya que Android fue construido sobre la parte superior de Linux, por lo que se aplican los permisos de archivos de Linux. El permiso permite al usuario proteger sus datos confidenciales que están almacenados en el dispositivo, y también, protege el acceso al contenido del proveedor, que básicamente es una base de datos en el dispositivo, los permisos son solicitados por una aplicación en el momento de la instalación y se otorgan o niegan una vez en el momento de la instalación que requiere la aprobación del usuario. (Alghamdi, Alfalqi, & Waqdan, 2015)

1.3.4. Malware en Android

1.3.4.1. Malware

Guerra, (2018) ,define al malware como “cualquier tipo de software dañino contra el funcionamiento normal de un dispositivo (esto también incluye dispositivos médicos, como un marcapasos), aplicación o red.”

Entonces, se puede considerar como un malware a cualquier programa o software que presente un comportamiento sin autorización, incorrecto o ilegal. Los usuarios de estos dispositivos móviles no pueden identificar el objetivo de una aplicación solo teniendo en cuenta sus permisos, es por ello que instalan aplicaciones de fuentes desconocidas o no autorizadas e invitan a problemas, esto es aprovechado por los atacantes, pues suelen programar las aplicaciones de malware para robar correos electrónicos, información de pulsaciones de teclas, credenciales de transacciones financieras, imágenes de cámara, información de redes sociales, información del sistema de archivos local y mucho más. Es por ello, que se requiere mayor seguridad móvil para la detección e identificación de aplicaciones genuinas y malignas en función de lo que solicitan las aplicaciones.

1.3.4.2. Análisis de Malware

Guerra, (2018) , lo define como “el arte de diseccionar un software malicioso para comprender su funcionamiento, caracterizarlo para su identificación en otros sistemas con herramientas automatizadas y determinar el método de eliminación en un dispositivo o sistema comprometido.”

El mercado Android es muy grande, por lo que es muy fácil desarrollar malware. Hoy en día hay un gran número de malware disponible en el mercado, se conoce como familia de malware. Estas familias de malware se pueden clasificar de acuerdo a su funcionalidad como: escalamiento de la privacidad, control remoto, intercambio financiero y robo de información privada, además existe una gran cantidad de aplicaciones de spyware que pueden robar su información personal y enviarla a un servidor remoto. Desde el punto de vista forense tenemos que analizar toda esta aplicación y comprobar que se trata de una aplicación de goodware o de malware. Para analizar el malware Android tenemos dos técnicas más desarrolladas, el análisis estático y el análisis dinámico, siendo el

análisis estático donde se da importancia al código de la aplicación para encontrar el código malicioso y corregirlo, pero en el análisis dinámico, la aplicación se ejecuta en el arenero y se descubre el comportamiento anormal de la aplicación.

1.3.4.3. Análisis Estático

El análisis estático no es más que analizar el aplicativo antes de ejecutarse para verificar si es malicioso o no lo es. En el análisis estático, las plataformas de detección incluyen secuencia de bytes n-gramo, firma de cadena, gráfico de flujo de control, celda de biblioteca sintáctica y distribución de frecuencia de código operativo (código de operación), etc. este análisis como un inconveniente importante tiene la carga y ofuscación de código. Antes de hacer el análisis estático, el archivo ejecutable debe ser descifrado y desempaquetado

El formato del archivo apk se detalla en la figura 2.

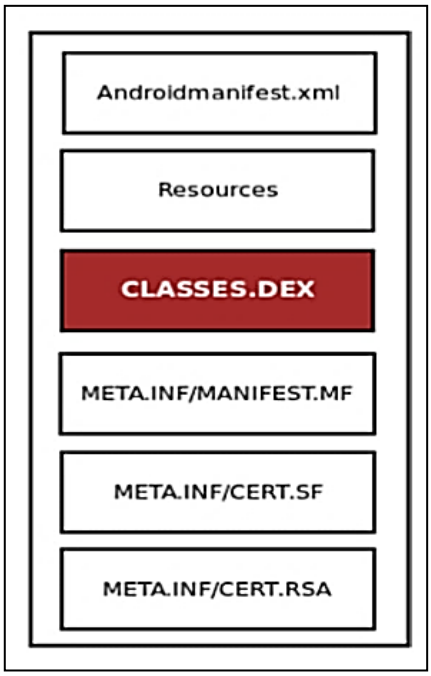

Figura 2. Formato de Archivo Apk. Fuente: (Barsiya, Gyanchandani, & Wadhwani, 2016)

a) **Firma:** Contiene la firma del resumen de mensajes del archivo APK. Si se cambia la firma de cualquier archivo APK eso significa que hay algo mal con el

archivo APK y puede ser fácilmente identificado. El analista también recopila la firma para identificar rápidamente el malware.

b) **Bytecodes:** La codificación Android está escrita en la plataforma Java, por lo que toda la parte ejecutable de la aplicación se almacena en “class.dex”. Esta carpeta contiene todas las clases compiladas del programa en forma de los códigos de byte. Así que se necesita usar ingeniería inversa para entender este código.

c) **Recursos:** Los recursos son la parte no ejecutable del programa. La mayoría de los recursos son componentes de la interfaz de usuario, como mapas de bits, menús y diseño.

d) **Componente:** La actividad, los servicios, el receptor de difusión y el proveedor de contenido son el componente del malware que se ejecuta en segundo plano puede utilizar el componente de servicios, así como un componente de receptor con el fin de obtener el acceso de arranque del sistema.

e) **Permisos:** Para acceder a algunas APIs protegidas de Android necesitamos solicitar algún permiso, como sendTextMessage, getPAckageInfo, getSimCountryIso, READ_Contect, Internet. Si encontramos algún permiso sospechoso en alguna aplicación simple, entonces podemos entender que se trata de un malware.

1.3.4.4. Análisis Dinámico

El análisis dinámico se utiliza para analizar el código malicioso y el comportamiento de la aplicación, esto es posible cuando se interactúa con el sistema y debe hacerse en un entorno controlado. La principal ventaja de este análisis se utiliza para registrar los comportamientos de la aplicación y detecta la carga de código dinámico durante el proceso de tiempo de ejecución debido a la sobrecarga de ejecutar la aplicación, esto es difícil implementar en un análisis comparativamente estático.

Existen algunas características que se extraen de un análisis dinámico:

a) Llamadas a las API: La Interfaz de Programación de Aplicaciones (API) es una agrupación de subrutinas o llamadas de funciones que se utilizan para la comunicación entre dos componentes de software o la comunicación entre componentes de software y hardware. La API puede basarse en el sistema operativo, en el hardware basado en la web y en la biblioteca de software.

b) Uso de CPU y Memoria: Muchos trabajos relacionados han utilizado el comando dumpsys para obtener información de la CPU y la memoria, el uso y características extraídas de ellos, que se representan entre un intervalo.

1.3.4.5. Familia de Malware

La clasificación familiar representa un problema de clasificación multiclase donde el número de clases es igual al número de familias. A su vez utiliza las mismas técnicas de representación que la detección de malware.

Es por ello que modelar las muestras de malware como secuencias de eventos proporciona una forma útil de entender su comportamiento en tiempo de ejecución, lo que permite analizar no sólo cómo las diferentes familias de malware difieren entre sí, sino también para extraer los patrones comunes compartidos en todos ellos.

1.3.5. Algoritmos de Clasificación

a) Random Forest

Lu & Hou, (2018) ,definen el bosque aleatorio como un clasificador que contiene múltiples árboles de decisión, y el resultado del bosque aleatorio está determinado por el número de categorías de salida de árboles individuales. La idea básica del bosque aleatorio se muestra en la figura 3.

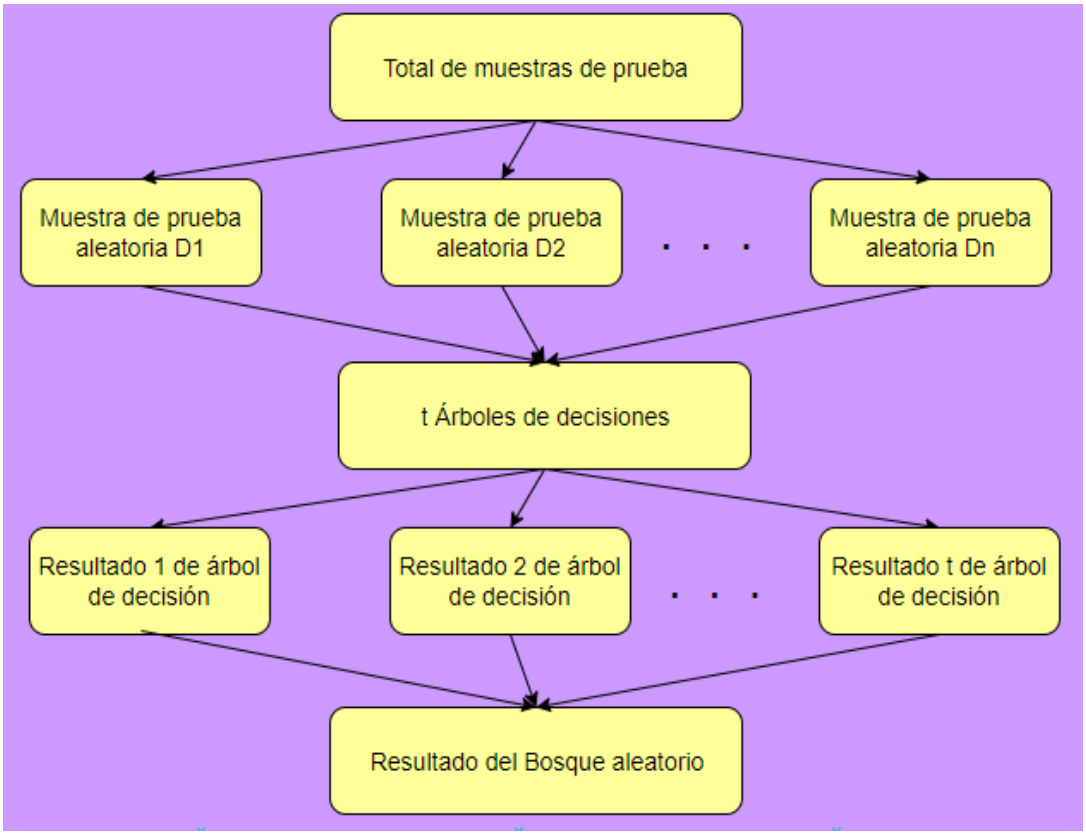

Figura 3. Algoritmo Random Forest. Fuente: (Lu & Hou, 2018).

Detalles de trabajo de un bosque aleatorio

El algoritmo para construir un bosque aleatorio es el siguiente:

1. Crear un subconjunto con los datos originales para que el árbol de decisión se base solo en una muestra del conjunto de datos original.
2. Crear un subconjunto con las variables independientes (características) también mientras construye el árbol de decisión.
3. Cree un árbol de decisión basado en los datos del subconjunto donde el subconjunto de filas y columnas se usa como grupo de datos.
4. Predecir en el grupo de datos de prueba o validación.
5. Repita los pasos 1 a 3 “n” número de veces, donde n es el número de árboles construidos.
6. La predicción final en el conjunto de datos de prueba es el promedio de predicciones de todos los “n” árboles.

b) Decision Tree

Hackeling, (2014),definió a los árboles de decisión como gráficos en forma de árbol que modelan una decisión. Las ramas de un árbol de decisión especifican las secuencias más cortas de variables explicativas que pueden examinarse para estimar el valor de una variable de respuesta. Los árboles de decisión se aprenden comúnmente dividiendo recursivamente el conjunto de entrenamiento en subconjuntos basados en los valores de las instancias para las variables explicativas. El siguiente diagrama muestra un árbol de decisión que veremos con más detalle.

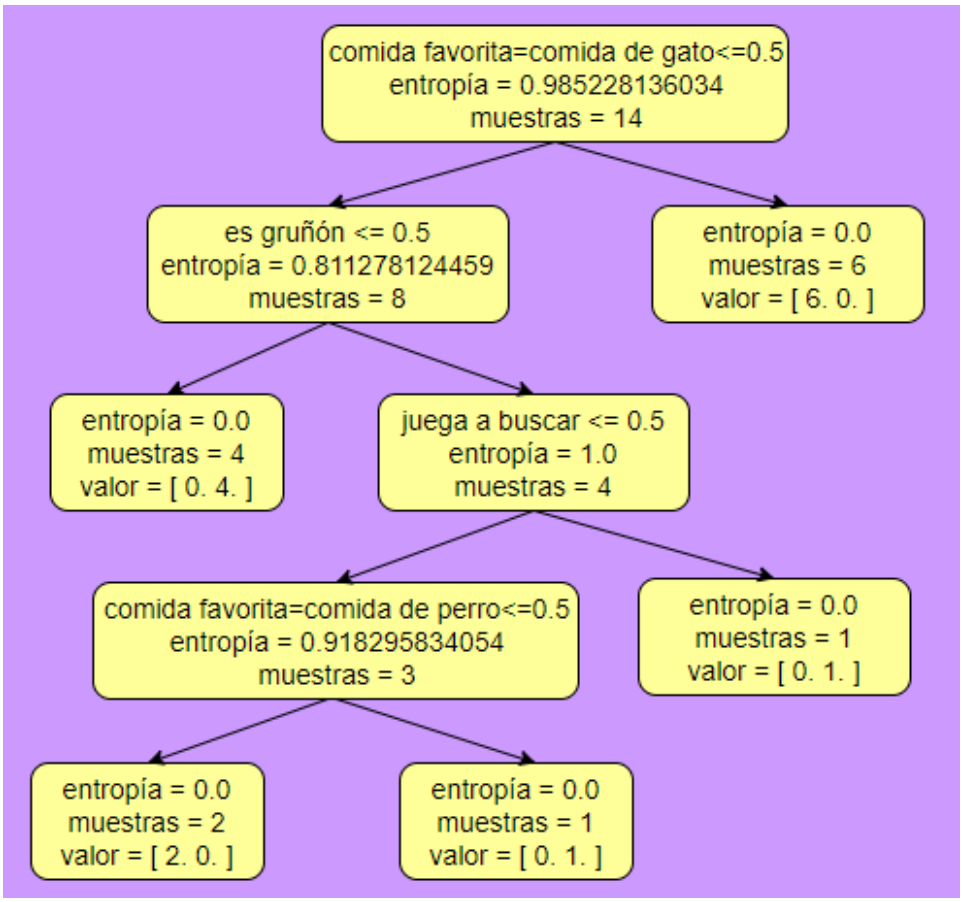

Figura 4. Algoritmo Decision Tree. Fuente: (Hackeling, 2014)

Entrenamiento de un árbol de decisión

Al iniciar se crea un árbol de decisión usando un algoritmo llamado Dicotomizador iterativo 3 (ID3), inventado por Ross Quinlan, ID3 fue uno de los primeros algoritmos utilizados para entrenar la decisión de árboles, suponga que tiene que clasificar a los animales como gatos o perros desafortunadamente no

puede observar a los animales directamente y debe usar solo unos pocos atributos de los animales para tomar tu decisión, te dicen si le gusta o no jugar, ya sea con frecuencia de mal humor o no, y su favorito de los tres tipos de alimentos. Para clasificar nuevos animales, el árbol de decisión examina una variable explicativa en cada nodo, la ventaja que sigue al siguiente nodo depende del resultado de la prueba. Por ejemplo, el primer nodo podría preguntar si al animal le gusta o no jugar, si el animal lo hace, seguiremos el borde hasta el nodo hijo izquierdo; si no, seguiremos el borde al nodo secundario derecho. Finalmente, un borde se conecta a un nodo hoja que indica si el animal es un gato o un perro.

Las siguientes catorce instancias comprenden nuestros datos de capacitación:

Tabla 2.

Datos de capacitación para el algoritmo Decision Tree.

Instancias de Entrenamiento	**Jugar con Pelota**	**Malhumor**	**Comida Favorita**	**Especie**
1	SI	NO	TOCINO	PERRO
2	NO	SI	COMIDA DE PERRO	PERRO
3	NO	SI	COMIDA DE GATO	GATO
4	NO	SI	TOCINO	GATO
5	NO	NO	COMIDA DE GATO	GATO
6	NO	SI	TOCINO	GATO
7	NO	SI	COMIDA DE GATO	GATO
8	NO	NO	COMIDA DE PERRO	PERRO
9	NO	SI	COMIDA DE GATO	GATO
10	SI	NO	COMIDA DE PERRO	PERRO
11	SI	NO	TOCINO	PERRO
12	NO	NO	COMIDA DE GATO	GATO
13	SI	SI	COMIDA DE GATO	GATO
14	SI	SI	TOCINO	PERRO

Nota: Tomada de la investigación de (Hackeling, 2014).

De estos datos podemos ver que los gatos son generalmente más gruñones que los perros. La mayoría de los perros juega a buscar y la mayoría de los gatos se niegan. Los perros prefieren comida para perros y tocino, mientras que los gatos solo como comida para gatos y tocino. Él es gruñón y juega a buscar variables explicativas se puede convertir fácilmente a características con valores binarios. La comida favorita explicativa es una variable categórica que tiene tres valores posibles; se codifica de inmediato. Recordemos que la extracción de características y preprocesamiento, esa codificación única representa una variable categórica con tantas características de valor binario como hay valores para cada variable y así representando la variable categórica con un solo valor entero y la función codifica un orden artificial a sus valores. Ya que la comida favorita tiene tres posibles estados, lo representaremos con tres características de valor binario. De esta tabla, podemos construir manualmente las reglas de clasificación. Por ejemplo, un animal gruñón y le gusta la comida para gatos debe ser un gato, mientras que un animal que juega a buscar y le gusta el tocino debe ser un perro y construir estas reglas de clasificación a mano incluso para un pequeño conjunto de datos es incómodo. En cambio, aprenderemos estas reglas creando un árbol de decisión.

c) k-Nearest Neighbors (KNN)

Brownlee, (2016), define, que KNN no tiene otro modelo que no sea almacenar todo el conjunto de datos, por lo que no se requiere aprendizaje eficiente. Las implementaciones pueden almacenar los datos utilizando estructuras complejas como árboles k-d para realizar búsquedas y la coincidencia de nuevos patrones durante la predicción eficiente, porque todo el conjunto de datos de entrenamiento es almacenado, y puede ser posible que desee pensar cuidadosamente sobre la consistencia de sus datos de entrenamiento. Ya que sería una buena idea para curarlo, actualizarlo a menudo a medida que haya nuevos datos disponibles y eliminar errores y datos atípicos.

Hacer Predicciones con KNN

El algoritmo KNN realiza las predicciones usando de forma directa el conjunto de datos a entrenar y las predicciones se hacen para un punto nuevo con base

en todos los datos de entrenamiento buscando las k instancias similares (vecinos) y resumiendo la variable de salida para esas k instancias. En el caso regresión esto sería la variable de salida media, siendo en la clasificación la clase de modo (o más común) valor.

Para poder hallar la similitud de las “k” instancias dentro del conjunto de datos con referencia a uno nuevo, se usa la medida relativa a la distancia. Siendo la distancia euclidiana la medida más común para las variables de entrada de valor real, esta distancia euclidiana se calcula con la raíz cuadrada de la sumatoria de las diferencias al cuadrado entre un punto a y un punto b en todos los atributos de entrada i.

$$Euclidean\ Distance(a,b) = \sqrt{\sum_{i=1}^{n}(a_1 - b_1)^2}$$

Clasificación con KNN

Para clasificar con KNN, la clase que contenga una mayor frecuencia de k – instancias, siendo el cálculo que se puede considerar como una salida. La función de cada instancia es emitir un voto por clase y por consecuencia la clase que contenga más votos viene a ser una predicción. Las probabilidades de clase se pueden calcular como la frecuencia normalizada de muestras que pertenecen a cada clase en el conjunto de k más similar para una nueva instancia de datos. Por ejemplo, en un problema de clasificación binaria (la clase es 0 o 1):

$$p(class = 0) = \frac{count(class = 0)}{count(class = 0) + count(class = 1)}$$

Si está utilizando k y tiene un número par de clases (por ejemplo, 2), es una buena idea elegir

un valor k con un número impar para evitar un empate. Y a la inversa, use un número par para k cuando tienes un número impar de clases. Los lazos se pueden romper consistentemente al expandir k por 1 y mirando la clase de la siguiente instancia más similar en el conjunto de datos de entrenamiento.

1.4. Formulación del Problema.

¿Cuál es el algoritmo de machine learning que presenta mejor desempeño en la detección de malware en Android?

1.5. Justificación e importancia del Estudio.

A. Justificación Tecnológica

Tecnológicamente el trabajo investigativo aborda una temática que actualmente es una tendencia como lo es el aprendizaje automático, basado en algoritmos que simplifican tareas de grandes dimensiones que el ser humano no podría realizarlas en poco tiempo , y de cómo estos algoritmos que pueden ayudar en la detección de información maliciosa que existe en las aplicaciones de manera que se pueda realizar un etiquetado, para determinar si son malware o no, a través de un entrenamiento de detección basado en características relevantes que presentan estas aplicaciones durante su ejecución.

B. Justificación Social

En la actualidad existen grandes innovaciones tecnológicas en dispositivos móviles, y empresas que apuestan por brindarnos un mejor servicio en la obtención de estos. A su vez se ha estimulado en gran medida la propagación de aplicaciones, ya que proporcionan servicios como redes sociales, juegos, transferencias, etc., pero también debemos tener de conocimiento que existe una gran cantidad de aplicaciones con código malicioso desarrollados con la intención de robar información determinada de un dispositivo móvil, puesto que en la guardamos información personal, laboral y financiera, es por eso que actualmente se dedica un esfuerzo importante a combatir este gran número de aplicaciones a través de diversos análisis de detección de malware, pero siendo algunos de ellos muy ineficientes o irrelevantes al momento de mostrar resultados, es por ello que en esta investigación realizaremos un Análisis Dinámico mediante algoritmos de machine learning.

C. Justificación Económica

La presente investigación no implica costos elevados ya que se han aplicado tecnologías libres para el desarrollo de la misma. Así mismo el resultado de la investigación podría beneficiar a cualquier empresa, puesto que la seguridad informática es fundamental en cualquier proceso, y al identificar los ataques de malware en las aplicaciones podemos generar un registro de estos ataques y estar a la defensiva de cuando aparezca un nuevo ataque o ya relacionado a los anteriores.

1.6. Hipótesis.

El algoritmo Random Forest es el que mejor desempeño muestra para detectar los malware en aplicaciones de dispositivos Android.

1.7. Objetivos.

1.7.1. Objetivo general.

Analizar de forma comparativa el desempeño de los algoritmos supervisados de machine learning en la detección de malware de aplicaciones Android.

1.7.2. Objetivos específicos.

a) Seleccionar tres algoritmos supervisados de machine learning con mayores usos en investigaciones previas para la detección de malware.

b) Caracterizar el comportamiento dinámico basado en tráfico de red de los ataques de malware.

c) Generar la base de conocimiento basándose en las características del malware.

d) Implementar los algoritmos seleccionados utilizando lenguaje de programación Python.

e) Evaluar los resultados del desempeño de los algoritmos.

II. MATERIAL Y MÉTODO

2.1. Tipo y Diseño de Investigación.

a) Tipo de Investigación

Este trabajo se considera Aplicado, Cuantitativo y Tecnológico, porque analiza múltiples elementos cuantificables y medibles, además trabajaremos con técnicas o métodos de investigaciones previas para llevarlos a la práctica.

b) Diseño de la Investigación

El diseño es Cuasi Experimental, debido a que permite manipular la muestra para generar mejores resultados.

2.2. Población y muestra.

a) Población

Se determina una población con 10 algoritmos de machine learning basados en investigaciones para detectar malware que se hallan en la base de datos ScienceDirect e IEEE: Random Forest, Naive Bayes, Support Vector Machine (SVM), Bagging, Multilayer Perceptron, Bayes Net, Decisión Tree, K-Nearest Neighbor (KNN), Multiclass Classifier y Logistic Regression, como se indica en la Tabla 4.

b) Muestra

La muestra del trabajo investigativo se obtiene con base a los algoritmos más utilizados en investigaciones previas para la detección de malware, obteniendo de esta manera los tres algoritmos con más apariciones en varias investigaciones siendo estos: Random Forest, Decisión Tree y K-Nearest Neighbor (KNN)

2.3. Variables, Operacionalización.

Tabla 3.

Operacionalización de variables.

Variable Independiente	**Dimensión**	**Indicador**	**Ítem**	**Técnica e instrumentos de recolección de datos**
Algoritmos de machine learning	Reconocimiento	Tiempo de Respuesta en la ejecución.	$T_{resp} = T_{finanálisis} - T_{inicioanálisis}$	Observación Documentación
	Rendimiento	Consumo de memoria RAM.	$C_{mem} = C_{findetección} - T_{iniciodetección}$	

Variable Dependiente	**Dimensión**	**Indicador**	**Ítem**	**Técnica e instrumentos de recolección de datos**

Detección de Malware	Precisión	Porcentaje de proximidad	$Precisión = \frac{TP}{TP + FP}$	Observación Documentación
	Exactitud	Porcentaje de valor real	$Exactitud = \frac{TP + TN}{FP + FN + TP + TN}$	
	Error	Porcentaje de margen de error	$Error = \frac{FP + FN}{Total}$	
	Recall	Porcentaje de los verdaderos Positivos	$Recall = \frac{TP}{FN + TP}$	

2.4. Técnicas e instrumentos de recolección de datos, validez y confiabilidad.

Técnicas de recolección de datos

a) Observación

Esta técnica viene a ser la forma lógica como se realiza un registro visual y que a su vez se puede verificar centrándose en lo que se va a conocer, en otras palabras, vamos a captar objetivamente lo que está ocurriendo en el entorno real, para luego poder describir, analizar y explicar desde un punto de vista científico.

b) Documentación

Se realizaron consultas bibliográficas a distintas bases de datos con material bibliográfico (libros, artículos, páginas web, etc.)

Instrumentos de recolección de datos

a) Ficha de aceptación

Este instrumento permite poder investigar, evaluar y recolectar todos los datos posibles, direccionados hacía un objetivo determinado, en esta parte vamos a establecer las variables específicas que utilizaremos para registrar algunos datos, y luego ofrecer las recomendaciones para la mejora y análisis que se crea conveniente.

b) Ficha de observación

La ficha de observación es un procedimiento donde se va a utilizar una serie de instrumentos para interrelacionar los hechos reales y la hipótesis por medio de la observación científica y la investigación que tiene que ser ordenada y sistematizada. Los instrumentos utilizados permitieron detallar lo que se va a observar e investigar, permitiendo así la recolección de datos, con base en un objetivo específico para determinar las variables específicas a utilizar.

2.5. Procedimiento de análisis de datos.

a) Observación

Se realizó la extracción de información con un conjunto de datos de malware de Android llamado CICAndMal2017, propuesto por el Instituto Canadiense de Ciberseguridad. En este enfoque, se ejecutan aplicaciones benignas y de malware en teléfonos inteligentes reales para evitar la modificación del comportamiento en tiempo de ejecución de muestras de malware avanzadas que pueden detectar el entorno del emulador, el cual expone los atributos descriptivos de un software para ser considerado un malware. Es por ello que conociendo cada atributo descriptivo la investigación se plantea utilizar algunas de las características más relevantes para de este modo sintetizar la información y poder generar un conjunto de datos óptimo.

b) Documentación

Para la extracción y selección de funciones en la base de datos, se capturan las funciones de tráfico de red (archivos. pcap) y se extrajo más de 80 funciones mediante el uso de CICFlowMeter-V3 durante los tres estados (instalación, antes de reiniciar y después de reiniciar). A su vez se trabajó con un dataset que va considerar un total de 15945 muestras, siendo el 20% de muestras benignas y 80% de muestras malignas, el conjunto de malware contiene dos de las cuatro categorías (Adware y SMS Malware), específicamente para este cuasi experimento se tomó en cuenta 5 familias de ADWARE y 5 familias de SMSMALWARE.

c) Ficha de aceptación

Se elaboró una ficha donde se especifica el tipo de algoritmo a evaluar, la cantidad de datos a analizar, el tiempo de detección y los resultados del funcionamiento en la detección de malware que se ha entregado al asesor del proyecto para verificar el cumplimiento de lo propuesto.

d) Ficha de observación

Se trabajó con una ficha donde se tuvo en cuenta el proceso de detección de los algoritmos a evaluar, especificando la cantidad de datos con los que se va a trabajar gradualmente, donde se puede ir obteniendo diversos resultados según

la cantidad gradual de datos que se vayan analizando, con la finalidad de comprobar la variación de porcentajes según la cantidad de muestras con las que se trabaje.

2.6. Criterios éticos.

a) Criterio de Responsabilidad

Este criterio se ha utilizado ya que existe la responsabilidad de todos los que tenemos una carrera orientada a las tecnologías de la información, de tener el cuidado respectivo con la seguridad informática, en su gran mayoría son los usuarios de dispositivos móviles los que pueden sufrir ataques que permitan robar la información privada, que generen consumo de memoria extra, etc. , esto debido a la cantidad de malware que se está desarrollando, creando desconfianza entre los usuarios hacia las nuevas tecnologías.

Y es por ello que la responsabilidad es fundamental, porque es nuestro compromiso como ingenieros de sistemas, tratar de buscar soluciones a los grandes problemas que afectan el desarrollo del avance tecnológico.

b) Derechos de Autor

Todo buen uso de los datos y la información requerida para el desarrollo de este trabajo está adecuadamente referenciado y citado con sus respectivos autores de desarrollo.

2.7. Criterios de Rigor Científico.

a) Consistencia

Es consistente, ya que se pone a prueba con otras técnicas que son establecidas y también aprobadas por la comunidad científica.

b) Fiabilidad

Para que los resultados sean fiables se establece un ambiente controlado con las condiciones suficientes para llevar a cabo la propuesta teniendo como referencia investigaciones previas similares, es por ello que se utilizó indicadores y procedimientos que sirvan para validar los resultados.

c) Objetividad

Los resultados que se han obtenido del desempeño de los algoritmos supervisados no están condicionados a un juicio parcial del investigador, además las conclusiones se basan en las pruebas realizadas.

III. RESULTADOS.

3.1. Resultados en Tablas y Figuras.

El siguiente trabajo de tesis surge con el propósito de identificar software malicioso basándose en el soporte del aprendizaje automatizado el cual ha tenido un crecimiento constante en uso de propuestas investigativas con una infinita variedad de datos ya sea videos, imágenes, texto, etc. Es por ello que los datos expuestos por un software considerado malware podrían ser caracterizados y correctamente etiquetados por algoritmos supervisados para poder generar un conocimiento que acoplado a la inteligencia artificial supone un beneficio importante proporcionando una alternativa para mejorar el rendimiento identificando este tipo de ataques y obteniendo resultados que permitan comparar los tres algoritmos supervisados seleccionados para esta investigación.

Con base a lo antes expuesto, se propone que los algoritmos Random Forest, K-Nearest Neighbor y Árbol de decisión sean utilizados para llevar a cabo la clasificación de los malwares, la base de datos a evaluar por estos 3 algoritmos consta de 12800 malware y 3145 goodware , los cuales han sido entrenados y puestos a prueba en porcentajes de 80% y 20% respectivamente, estos porcentajes se han derivado después de hacer una revisión de la literatura de investigaciones previas, donde ponen en manifiesto los porcentajes a trabajar, los resultados que se obtienen se han validado por la técnica de Validación Cruzada con ejecución de 10 instancias, de donde se han evaluados los siguientes indicadores : Exactitud , Precisión , Exhaustividad (Recall) , Tiempo de ejecución y Consumo de memoria RAM del algoritmo.

a) Exactitud

La exactitud es un parámetro que permite evaluar en función del número total de predicciones que son correctas, es decir aquellas que se basa solamente en

las predicciones que el algoritmo acertó en la detección del malware dentro de todas las pruebas realizadas, pero cabe recalcar que en datos desbalanceados la exactitud a veces no representa la fiabilidad del algoritmo, puesto que al tener más datos que son ciertos es menos complicado que se equivoque. La figura 5 muestra al Algoritmo k-NN con los resultados más bajos con un porcentaje de 85.4%, seguido de Decision Tree con 91.2% de exactitud, siendo el algoritmo con mejor resultado Random Forest con una exactitud de 96%, esto quiere decir que Random Forest tiene mayor acierto detectando la mayor cantidad de malware que se encuentra en la base de datos.

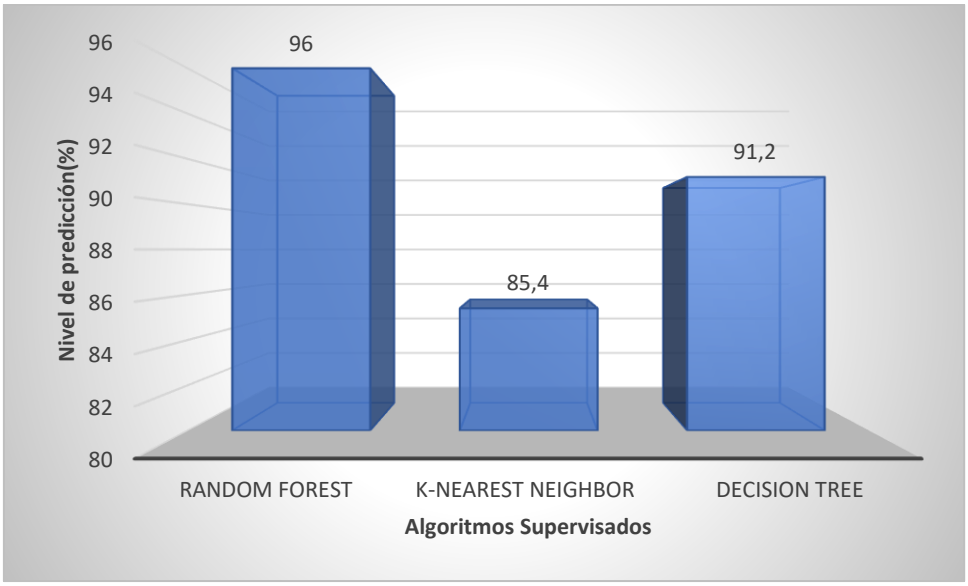

Figura 5. Evaluación de la efectividad del modelo propuesto basado en la Exactitud. Fuente: Elaboración propia.

b) Precisión

La precisión es la relación de los casos considerados como positivos que son predichos correctamente y que son verdaderamente positivos, lo que permite indicar la fiabilidad y calidad de un algoritmo para predecir los malware, todo esto lo podemos resumir en una pregunta ¿Qué porcentaje de los datos pueden ser considerados malware?, esto quiere decir que la precisión funciona de la siguiente manera, de las aplicaciones que se han detectado como malware, va arrojar un porcentaje basado en que si esas aplicaciones detectadas como malware verdaderamente lo son o no, dependiendo del porcentaje veremos qué algoritmo ha detectado mejor y clasificado las aplicaciones . Asimismo, en la

figura 6 se puede observar que el algoritmo con mejores resultados también en el indicador de precisión sigue siendo Random Forest con 92.9%, el algoritmo Decisión Tree es el segundo mejor con un porcentaje de 83.1% y el algoritmo k-NN obtiene el menor resultado ya que mantiene un porcentaje de 73.9%.

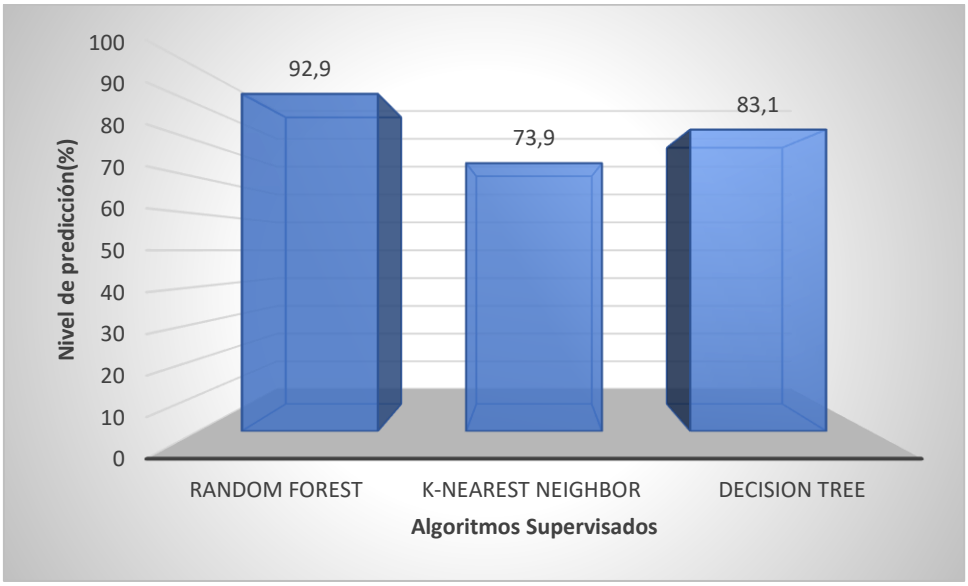

Figura 6. Evaluación de la efectividad del modelo propuesto basado en Precisión. Fuente: Elaboración propia.

c) Exhaustividad (Recall)

El recall es una métrica que indica la cantidad de malware que el algoritmo es capaz de identificar dentro de la base de datos, resumiendo en una pregunta tendríamos ¿Qué porcentaje de los datos que son malware se pueden identificar?, esto quiere decir que de todas las aplicaciones que son sólo malware en la base de datos, cuántas de esas aplicaciones se ha podido detectar como malware con cada uno de los algoritmos. Y como podemos ver en la Figura 7, en cuanto a los resultados basados en identificar malware k-NN tiene un porcentaje por debajo de 73% lo que evidencia ser el peor el algoritmo con el resultado de este indicador, seguido por Decision Tree que se mantiene como el segundo mejor detector con 86.1% y Random Forest con un porcentaje de 87.2, no tan distante de Decision Tree, pero se mantiene como el algoritmo con mejor resultado.

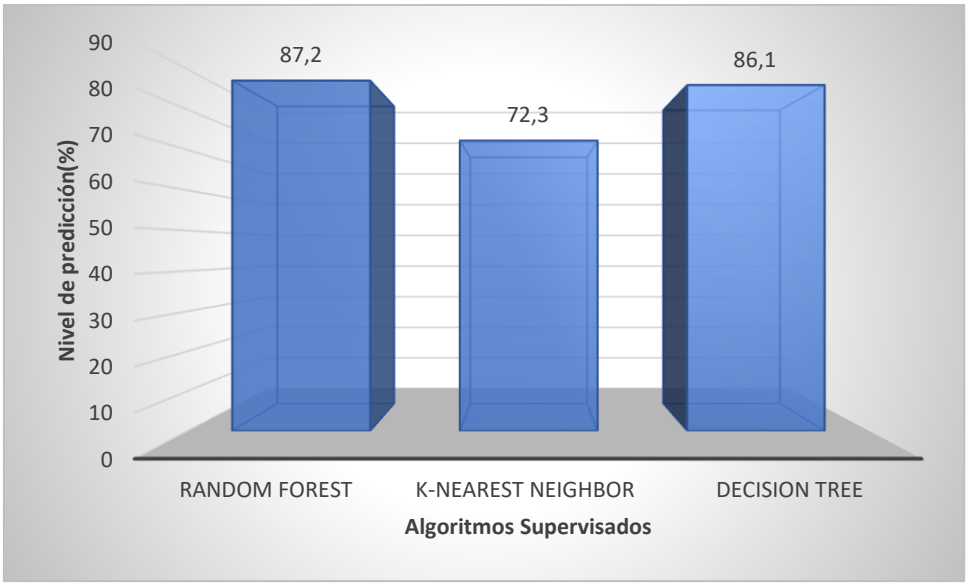

Figura 7. Evaluación de la efectividad del modelo propuesto basado en el Recall. Fuente: Elaboración propia.

d) Error

El análisis de errores es un indicador que va a permitir saber qué hacer para mejorar el rendimiento de un algoritmo que estamos evaluando, además está relacionado con la exactitud , ya que lo calculamos de forma probabilística, restando la unidad que vendría hacer el 100% de acierto menos la exactitud obtenida por algoritmo: 1 – exactitud , es por ello que se evidencia en los resultados que Random Forest tiene menor margen de error con 4% , ya que su exactitud en el indicador anterior fue de 96%, seguido como un algoritmo con menor error está Decisión Tree con 8.8% y k-NN al haber obtenido menor porcentaje de exactitud que los otros dos algoritmos tiene un margen de error mayor de 14.6%

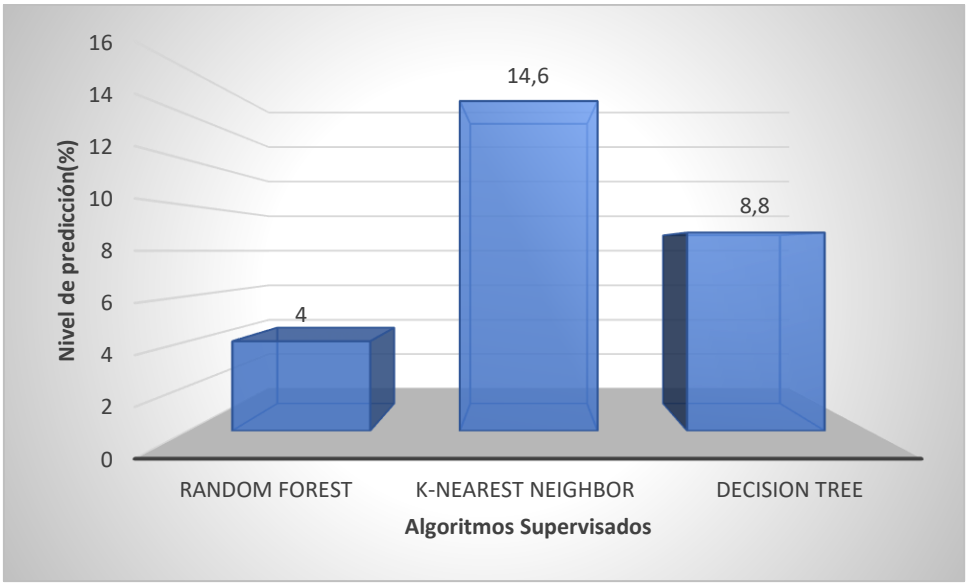

Figura 8. Evaluación de la efectividad del modelo propuesto basado en Error. Fuente: Elaboración propia.

e) Tiempo de respuesta a la ejecución del algoritmo

El tiempo de respuesta permite evidenciar la rapidez de un algoritmo durante el proceso de ejecución para detectar un malware en un conjunto de datos, como podemos ver en la Figura 9 , los menores tiempos son de los algoritmos k-NN y Decisión Tree con 2 y 3 segundos respectivamente, a diferencia de Random Forest que se ejecuta en un tiempo de 24 segundos, esto debido a que Random Forest tiene un parámetro para determinar la cantidad de nodos a evaluar , en esta investigación corresponden a 100 nodos lo que hace más fiable el resultado pero genera más tiempo de espera.

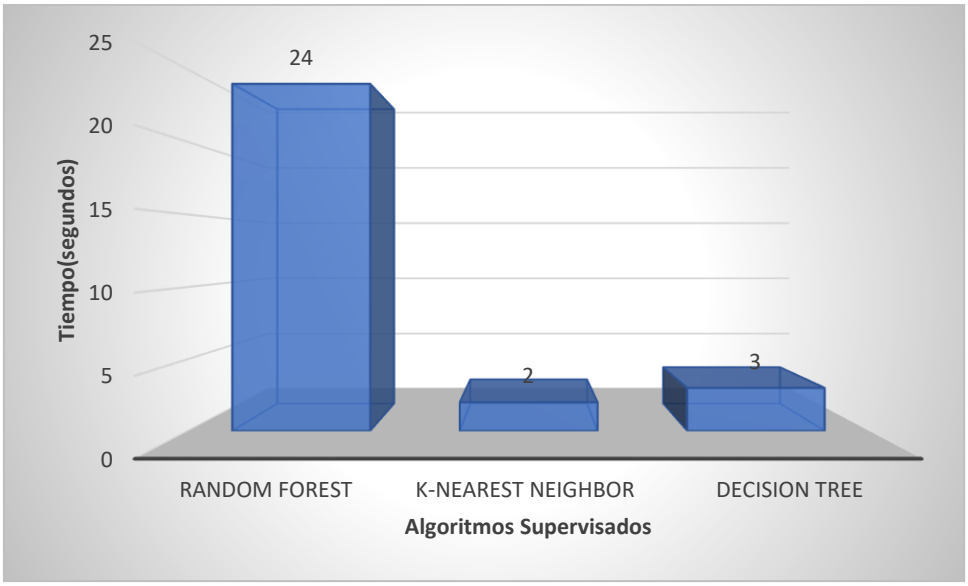

Figura 9. Evaluación del rendimiento del modelo propuesto basado en el Tiempo de Ejecución. Fuente: Elaboración propia.

f) Consumo de memoria RAM

El consumo de memoria RAM es un indicador que permite identificar qué tanto por ciento está exigiendo un proceso determinado a un dispositivo, debemos recordar que la memoria RAM permite gestionar los datos de las aplicaciones que en ese momento están funcionando y por ende afecta directamente al rendimiento del dispositivo, es por ello que mientras más consuma memoria RAM , más va a exigir al dispositivo, en este caso tanto Decision Tree como k-NN son los dos algoritmos que consumen menos memoria RAM durante el proceso de detección de malware con 215.77 y 212.41 mb respectivamente, y como vemos Random Forest brinda buenos resultados en exactitud o precisión, pero al tener parámetros que buscan mejorar estos indicadores, consumen más memoria para encontrar mejores resultados, teniendo un consumo de 295.45 mb, lo que indicaría que este algoritmo no funciona en cualquier dispositivo que cuente con poca memoria RAM.

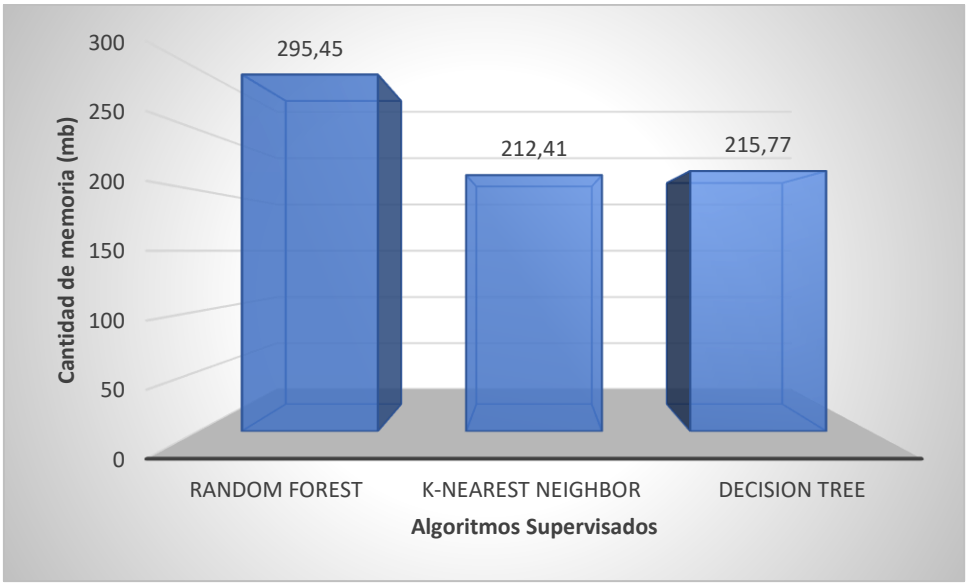

Figura 10. Evaluación del rendimiento del modelo propuesto basado en el Consumo de Memoria RAM. Fuente: Elaboración propia.

3.2. Discusión de resultados.

Una vez entrenados y puestos a prueba cada uno de los algoritmos en la detección de malware con la base de datos de flujo de red, se obtiene que el algoritmo que presenta un desempeño por encima de todos en la clasificación de un malware es Random Forest a diferencia de Decision Tree y k-NN, estos resultados se basan principalmente en los indicadores de Exactitud, Precisión y Recall, pero debemos tener en cuenta que, si evaluamos en función del rendimiento, Random Forest presenta bajos resultados, ya que si a costes computacionales se refiere tanto el tiempo de respuesta en la ejecución y el consumo de memoria RAM, entonces k-NN y Decision Tree fueron el primer y segundo mejor algoritmo en ese orden. A su vez cabe recalcar que la gran mayoría de investigaciones previas de detección de malware no consideran en sus resultados el consumo de recursos o no los evidencian, por lo cual sería ideal que esta información sea referenciada en estas investigaciones, ya que, en la actualidad con el avance de la creación de aplicaciones para la diversidad de dispositivos móviles, evaluar el rendimiento en la detección de malware también ayudaría a comprender la forma de ejecución del algoritmo.

Una vez obtenidos los resultados del desempeño de los algoritmos se puede realizar comparaciones con propuestas hechas en investigaciones previas, que

tuvieron como objetivo la detección de malware con características de flujos de red, y en forma específica trabajaron con algoritmos supervisados. En la investigación de Lashkari, Kadir, Taheri, & Ghorbani, (2018), los resultados obtenidos aplicando validación cruzada de 10 veces en lo que respecta a clasificación binaria de malware muestra a Decision Tree, como el algoritmo de mayor porcentaje de precisión 85.1%, a diferencia del modelo implementado que muestra un mejor resultado de precisión con el algoritmo Random Forest con 92.9%, esto se debe a que Lashkari trabaja con una muestra de datos de 5065 goodware y 429 malware, a diferencia de esta propuesta de investigación donde se considera 15945 muestras , siendo el 20% goodware y 80% malware, además se aplica el mismo porcentaje para entrenamiento y pruebas, lo que le permite al algoritmo tener un buen entrenamiento con diversidad de aplicaciones para así generar mejores resultados con la base de datos, adicional a la cantidad de muestras, en la investigación de Lashkari considera 9 características relevantes, y con base a pruebas realizadas en esta propuesta podemos afirmar que a menor cantidad de características y estas a su vez sean más relevantes generan mejores resultados y reducen consumo de recursos, es por ello de la mejora en la precisión de esta investigación.

Con respecto al porqué de la mejora de resultados entre esta propuesta y la de Lashkari, la investigación de Manzano, Meneses, & Leger, (2020) ratifica lo antes descrito, presentando mejores resultados debido a que también se trabaja con una base de datos de flujo de red de Ransomware donde el 80% son goodware y 20% malware, aumentando la capacidad de aprendizaje para los algoritmos, además que balancea la cantidad de muestras, reafirmando lo antes expuesto, en la investigación de Manzano también se han seleccionado 9 características relevantes para realizar entrenamiento y pruebas, obteniendo en este caso al algoritmo Random Forest como el de mejor desempeño en el indicador de exactitud con 96%. Es por ello que los resultados de Manzano son muy similares a los generados por esta investigación ya que el algoritmo Random Forest también muestra 96% de exactitud, pero a diferencia de lo trabajado con la investigación de Manzano, es que en esta propuesta se trabaja con 7 características relevantes generando similar resultado de exactitud, pero al trabajar con menos características el consumo de memoria RAM es menor.

Los resultados obtenidos muestran un buen desempeño de los algoritmos en la detección de malware y los porcentajes obtenidos tanto de exactitud, precisión o recall determinan que es posible identificar y detectar aplicaciones malware con una base de datos de flujo de red, puesto que al ser características dinámicas de aplicaciones usadas en ambientes reales y no emulados, brindando un conocimiento más certero del comportamiento de un malware, también se comprobó que el número de muestras beneficia en la obtención de buenos resultados, como se sabe la base de datos y la selección de características relevantes son primordiales para trabajar con aprendizaje automático, en este caso 7 características, permitiendo mejorar los resultados y reducir consumo de recursos y tiempo, se determina que Random Forest presenta mejores resultados en cuanto a la detección de Malware llegando a picos de 96% de exactitud y 92.9% de precisión siendo hasta acá el mejor algoritmo de los tres que permite detectar aplicaciones maliciosas en un amplio conjunto de datos.

3.3. Aporte práctico.

Proceso de selección de algoritmos supervisados de machine learning

En esta parte del trabajo se realizó la revisión de artículos científicos publicados en dos bases de datos confiables y validadas, como son IEEE Xplore y ScienceDirect, relacionados a algoritmos de machine learning. Las investigaciones consideradas son el resultado de las pruebas generadas en los últimos cinco años (2016 – 2021) dentro del entorno de la ingeniería, que han sido evaluados para detección de malware en dispositivos Android. Asimismo, las publicaciones que se tomaron en cuenta son aquellas vinculadas con la siguiente cadena de búsqueda “Supervised Algorithms AND Detection AND Malware AND Android”, los resultados obtenidos en la base de datos de IEEE Xplore arrojaron 14 artículos comprendidos en 12 conferencias y 2 revistas ordenados por relevancia, de los cuales se tomaron en cuenta 7 artículos que especifican los resultados de la detección de malware con algoritmos individuales de machine learning supervisados , y de la misma forma en el base de datos de ScienceDirect , se obtuvieron 371 artículos de donde se seleccionaron los 15 primeros ordenados de acuerdo a su relevancia en el

ámbito computacional, de estos 15 artículos siguiendo con el mismo patrón de selección que en la base de datos IEEE , se tomaron en cuenta 8 artículos que especifican los resultados de la detección de malware con algoritmos individuales de machine learning supervisados, obteniendo como un total 15 artículos que permiten generar la población de algoritmos que se muestra en la tabla 4.

Tabla 4.

Evaluación de resultados de los algoritmos en la detección de malware de investigaciones publicadas en 2016-2021.

AÑO	AUTORES	ALGORITMOS	EXACTITUD (%)	PRECISIÓN (%)	RECALL (%)	F1-SCORE (%)	AUC (%)
2016	Nuray Baltaci Kamil Akhuseyinoglu	Naive Bayes	---	72.35	---	---	---
2016	Taniya Bhatia	Decisión Tree	---	85	85.3	---	---
2016		Random Forest	---	88	89	---	---
2016	Marian Kuhnel Ulrike Meyer	K-Nearest Neighbor (KNN)	99.27	---	---	---	---
2016		Decisión Tree (j48)	99.19	---	---	---	---
2016		Random Forest	99.36	---	---	---	---
2017	Sanya Kumar Ari Viinikainem Timo Hamalainen	Random Forest	97.5	---	---	---	---
2017		JRIP	96.4	---	---	---	---
2020	Sebastián Hahn Mykolai Protsenko	Random Forest	96	97	---	---	---

17	Tilo Muller						
2018	Sanya Kumar Ari Viinikainem Timo Hamalainen	Naive Bayes	99.9	---	---	99.9	---
		Decisión Tree (j48)	99.8	---	---	99.9	---
2019	Ming Yang Su Jer Yuan Chang Kek Tung	Bayes Net	80.1	---	---	---	---
		Naive Bayes	80	---	---	---	---
		K-Nearest Neighbor (KNN)	90.3	---	---	---	---
		Decisión Tree (j48)	90.6	---	---	---	---
		Support Vector Machine (SVM)	91.4	---	---	---	---
2019	Mohammamnd Kamel Khaled Mahmound	Extra Tree	87.75	89.35	85.33	---	---
		Random Forest	86.65	89	83.22	---	---
		Decision Tree	86.12	85.76	86.16	---	---
2019	Juan Rodríguez Rama	Decisión Tree	99.9	---	---	---	---
2020	Prerna Agrawal Bhushan Trivedi	Decisión Tree	89.57	85.81	---	---	86.17
		Random Forest	91.21	87.93	---	---	88.54
		K-Nearest Neighbor (KNN)	90.92	87.93	---	---	88.54
		Support Vector Machine (SVM)	90.31	87.39	---	---	87.39
		Regresión Logística	89.87	86.64	---	---	85.66
2018	Tianliang Lu Su Hou	Random Forest	83.7	91	80.7	---	---

2020	Carlos Manzano Claudio Meneses Paúl Leger	Random Forest	91	---	97	94	---
		K-Nearest Neighbor (KNN)	83	---	86	84	---
		Decisión Tree	90	---	93	91	---
2021	Nicolas Kanzig Roland Meier Luca Gambazzi Laurent Venbever	Random Forest	97.10	---	---	---	---
		Support Vector Machine (SVM)	95.31	---	---	---	---
		Extra Tree	96.45	---	---	---	---
		K-Nearest Neighbor (KNN)	96.13	---	---	---	---
2021	Muhammad Ijaz Muhammad Hanif Durad Maliha Ismail	Naive Bayes	91.1	---	---	---	---
2021	Vasileos Syrris Dimitris Geneiatakis	Random Forest	97.4	---	89.1	93.1	---
		Support Vector Machine (SVM)	97.8	---	95.9	96.9	---

Nota: Total de Investigaciones tomadas de la base de datos de ScienceDirect e IEEE Xplore (2016 -2021)

Una vez hecha la revisión de las investigaciones y los resultados obtenidos, se considera para esta tesis comparar los mejores algoritmos en cuanto a desempeño y uso en investigaciones previas, de esta forma, para empezar a determinar la selección por desempeño de clasificación de malware se genera una tabla que contiene todos los resultados porcentuales finales de cada algoritmo supervisado encontradas en las investigaciones al momento de detectar malware en dispositivos Android teniendo en cuenta la Exactitud, Precisión y Recall, como se muestra en la Tabla 5.

Tabla 5.

Resultados del desempeño en la clasificación de malware de cada algoritmo supervisado en diversas investigaciones.

Algoritmo Clasificador	Exactitud (%)	Precisión (%)	Recall (%)
Support Vector Machine	90.31; 91.4; 95.31; 97.8	87.39	95.9
K-Nearest Neighbor	83; 90.3; 90.92; 96.13; 99.27	87.93	86
Naive Bayes	80; 91.1; 99.9	72.35	
Extra Tree	87.75; 96.45	89.35	85.33
Decision Tree	86.12; 89.57; 90; 90.6; 99.19; 99.8; 99.9	85; 85.76; 85.81	85.3; 86.16; 93
Random Forest	83.7; 86.65; 91; 91.21; 96; 97.10; 97.4; 97.5; 99.36	87.93; 88; 89; 91; 97	80.7; 83.22; 89; 97
Regresion Logistica	89.87	86.64	
JRIP	96.4		

Nota: Algoritmos usados en investigaciones previas de la base de datos de ScienceDirect e IEEE Xplore (2016 -2021).

Mostrados los resultados porcentuales de cada algoritmo se realiza una tabla general con los mejores resultados obtenidos en cada métrica, la cual se puede apreciar en la Tabla 6. Asimismo, para finalizar la primera parte del proceso de selección se ha tenido en cuenta elegir aquellos algoritmos que presenten resultados indispensables en las métricas de Exactitud, Precisión y Recall, omitiendo aquellos algoritmos que no cumplan con esta condición, como se muestra en la Tabla 7 , para culminar la selección de estos algoritmos se elabora una tabla que muestre los mejores resultados en cuanto a Exactitud, debido a que en las investigaciones se han encontrado valores muy altos en esta métrica generando fiabilidad, es por ello que se ha optado por los algoritmos que tengan resultados mayores de 99% de exactitud, lo que permite elegir tres algoritmos que se muestran en la Tabla 8.

Tabla 6.

Mejor desempeño en la clasificación de malware de cada algoritmo supervisado.

Algoritmo Clasificador	Exactitud	Precisión	Recall
Support Vector Machine	97.80 %	87.39 %	95.9 %
K-Nearest Neighbor	99.27 %	87.93 %	86.00 %
Naive Bayes	99.90 %	72.35 %	-
Extra Tree	96.45 %	89.35 %	85.33 %
Decision Tree	99.90 %	85.76 %	93.00 %
Random Forest	99.36 %	91. 00 %	97.00 %
Regresion Logistica	89.87 %	86.64 %	-
JRIP	96.40 %	-	-

Nota: Algoritmos usados en investigaciones previas de la base de datos de ScienceDirect e IEEE Xplore (2016 -2021).

Tabla 7.

Algoritmos supervisados con resultados de exactitud, precisión y recall

Algoritmo Clasificador	Exactitud	Precisión	Recall
Support Vector Machine	97.80 %	87.39 %	95.9 %
K-Nearest Neighbor	99.27 %	87.93 %	86.00 %
Extra Tree	96.45 %	89.35 %	85.33 %
Decision Tree	99.90 %	85.76 %	93.00 %
Random Forest	99.36 %	91. 00 %	97.00 %

Nota: Algoritmos usados en investigaciones previas de la base de datos de ScienceDirect e IEEE Xplore (2016 -2021).

Tabla 8.

Algoritmos con mejores resultados en cuanto a exactitud.

Algoritmo Clasificador	Exactitud
K-Nearest Neighbor	99.27 %
Decision Tree	99.90 %
Random Forest	99.36 %

Nota: Algoritmos usados en investigaciones previas de la base de datos de ScienceDirect e IEEE Xplore (2016 -2021).

Como se puede apreciar en la primera parte del proceso de selección se ha podido reducir a 3 algoritmos tomando en cuenta el desempeño, de la misma forma se muestra una tabla general donde se especifica que algoritmos son los más usados en investigaciones previas, con la finalidad de consolidar y dar falibilidad a la decisión de seleccionar estos tres algoritmos, como se muestra en la Tabla 9, este proceso permite tener la seguridad de trabajar con los algoritmos Random Forest, Decision Tree y K-Nearest Neighbor, siendo estos los que van a servir para el entrenamiento y futuras pruebas.

Tabla 9.

Total de investigaciones por Algoritmo individual supervisado.

Algoritmo Clasificador	**Investigaciones**
Support Vector Machine	4
K-Nearest Neighbor	5
Naive Bayes	4
Extra Tree	2
Decision Tree	8
Random Forest	10
Regresion Logistica	1
JRIP	1

Nota: Algoritmos usados en investigaciones previas de la base de datos de ScienceDirect e IEEE Xplore (2016 -2021).

Características del comportamiento de los ataques malware

La caracterización de los malware se realiza utilizando estrictamente el enfoque propuesto por NetFlowMeter el cual es un generador de flujo de tráfico de red que se ha escrito en Java y ofrece una enorme flexibilidad en términos de elegir las características que desea calcular, agregar nuevas y también tener un mejor control de la duración del tiempo de espera del flujo (University New Brunswick, 2017), la base de datos de entrenamiento y pruebas, que en total son 15945 muestras se genera con dos categorías de malware: ADWARE y SMSMALWARE que representan el 80% y la categoría de archivos benignos llamada BENIGWARE que representa el 20%.

Los ADWARE, son software no deseados diseñados para mostrar anuncios en la pantalla, normalmente en un explorador, en esta categoría se trabaja con cinco familias: DOWGIN, EWIND, FEIWO, GOOLIGAN, KEMOGE. Asimismo, los SMSMALWARE, es un malware para hacer llamadas no autorizadas o enviar mensajes de textos no autorizados sin que el usuario conozca o consienta el uso, es esta categoría tenemos también cinco familias: BEANBOT, BIIGE, FAKEINST, FAKEMART y FAKENOTIFY. Además, NetFlowMeter genera flujos bidireccionales (Biflow), donde el primer paquete determina las direcciones hacia adelante (fuente a destino) y hacia atrás (destino a fuente), de ahí las 83 características estadísticas como Duración, Número de paquetes, Número de bytes, Longitud de paquetes, etc. también se calculan por separado en la dirección hacia adelante y hacia atrás (UNB, 2017). El resultado de la aplicación es el formato de archivo CSV con seis columnas etiquetadas para cada flujo, a saber, FlowID, SourceIP, DestinationIP, SourcePort, DestinationPort y Protocol con más de 80 características de tráfico de red como se evidencia en la Tabla 10.

Tabla 10.

Características dinámicas de flujo de red de la ejecución de una aplicación.

Nombre de la Característica	**Descripción**
Duración de flujo	Tiempo de duración del flujo de red (µs).
Paquete Fwd total	Total de paquetes en dirección de subida.
Paquetes totales de Bwd	Total de paquetes en dirección de bajada.
Longitud total del paquete Fwd	Dimensión total de los paquetes en subida.
Longitud total del paquete Bwd	Dimensión total de los paquetes en bajada.
Fwd Longitud del paquete Mín.	Mínima dimensión del paquete en subida.
Fwd Longitud del paquete Máx.	Máxima dimensión del paquete en subida.
Fwd Paquete Longitud Media	Dimensión media del paquete en subida.

Fwd Longitud del paquete estándar	Dimensión de la desviación estándar de los paquetes en subida.
Bwd Longitud del paquete Mín.	Mínima dimensión del paquete en bajada.
Bwd Longitud del paquete Máx.	Máxima dimensión del paquete en bajada.
Bwd Longitud del paquete Media	Dimensión media del paquete en bajada.
Bwd Longitud del paquete Estándar	Dimensión de la desviación estándar de los paquetes en bajada.
Byte de flujo / s	Cantidad de bytes en un flujo de red(seg.).
Paquetes de flujo / s	Cantidad de paquetes en un flujo de red(seg.).
Flow IAT Mean	Media de tiempo de 2 paquetes que se han enviado en el flujo.
Flow IAT Std	Duración de la desviación estándar de 2 paquetes que se han enviado en el flujo
Flow IAT Max	Duración máxima de 2 paquetes que se han enviado en el flujo.
Flow IAT Min	Duración mínima de 2 paquetes que se han enviado en el flujo.
Fwd IAT Min	Duración mínima de 2 paquetes enviados en subida.
Fwd IAT Max	Duración máxima de 2 paquetes enviados en subida
Fwd IAT Mean	Duración media de 2 paquetes enviados en subida.
Fwd IAT Std	Duración de desviación estándar de 2 paquetes enviados en subida.
Fwd IAT Total	Duración total de dos paquetes enviados de subida.
Bwd IAT Min	Duración mínima de 2 paquetes enviados de bajada.
Bwd IAT Max	Duración máxima de 2 paquetes enviados de bajada.
Bwd IAT Mean	Duración media de 2 paquetes enviados de bajada.
Bwd IAT Std	Duración de desviación estándar de 2 paquetes enviados de bajada.
Bwd IAT Total	Duración total de 2 paquetes enviados de bajada.
Fwd PSH flag	Cantidad de ocasiones que el indicador PSH se configuró en paquetes de subida (0 para UDP)
Bwd PSH Flag	Cantidad de ocasiones que el indicador PSH se configuró en paquetes de bajada (0 para UDP)

Fwd URG Flag	Cantidad de ocasiones que se estableció el indicador URG en paquetes de subida (0 para UDP)
Bwd URG Flag	Cantidad de ocasiones que se estableció el indicador URG en paquetes de bajada (0 para UDP)
Longitud del encabezado de Fwd	Cantidad de bytes usados para encabezados de subida.
Longitud del encabezado Bwd	Cantidad de bytes usados para encabezados de bajada.
Paquetes Fwd / s	Cantidad de paquetes reenviados de subida.
Paquetes Bwd / s	Cantidad de paquetes reenviados de bajada.
Longitud mínima del paquete	Dimensión mínima de un paquete.
Longitud máxima del paquete	Dimensión máxima de un paquete.
Longitud de paquete Media	Dimensión media de un paquete.
Longitud del paquete estándar	Dimensión de la longitud estándar de un paquete.
Variación de longitud de paquete	Dimensión de variación de un paquete.
FIN Flag Count	Cantidad de paquetes FIN
SYN Flag Count	Cantidad de paquetes SYN
RST Flag Count	Cantidad de paquetes RST
PSH Flag Count	Cantidad de paquetes PUSH
ACK Flag Count	Cantidad de paquetes ACK
URG Flag Count	Cantidad de paquetes URG
CWR Flag Count	Cantidad de paquetes CWE
ECE Flag Count	Cantidad de paquetes ECE
Ratio abajo / arriba	Descargar y cargar un ratio
Tamaño promedio de paquete	Dimensión promedio de un paquete
Tamaño segmento AVG Fwd	Dimensión media del segmento en subida
Tamaño de segmento AVG Bwd	Cantidad media de velocidad de transferencia de bytes en subida.

Longitud del encabezado de avance	Dimensión del encabezado para un paquete de forma directa.
Fwd Promedio Bytes / Granel	Cantidad de velocidad media de transferencia de bytes en subida.
Fwd AVG Paquete / Granel	Cantidad media de paquetes de tasa masiva en subida.
Fwd AVG Bulk Rate	Cantidad media de tasa masiva en subida.
Bwd Promedio Bytes / Granel	Cantidad media de bytes de tasa masiva en bajada.
Bwd AVG Paquete / Granel	Cantidad media de la tasa masiva de paquetes en bajada.
Bwd AVG Bulk Rate	Cantidad media de tasa masiva en bajada.
Subflujo Paquetes Fwd	Cantidad media de paquetes en un subflujo en subida.
Subflujo Bytes Fwd	Cantidad media de bytes en un subflujo en subida.
Paquetes de Subflujo Bwd	Cantidad media de paquetes en un subflujo en bajada.
Subflujo Bytes Bwd	Cantidad media de bytes en un subflujo en bajada.
Init_Win_bytes_forward	Total de bytes enviados en la ventana de inicio en subida.
Init_Win_bytes_backward	Total de bytes enviados en la ventana de inicio en bajada.
Act_data_pkt_forward	Recuento de paquetes con al menos 1 byte de carga de datos TCP en subida.
min_seg_size_forward	Dimensión mínima de segmento durante la subida.
Min activo	Duración mínima de un flujo que pasó de activo a inactivo.
Media activa	Duración media de un flujo que pasó de activo a inactivo.
Max activo	Duración máxima de un flujo que pasó de activo a inactivo.
Std activo	Duración de desviación estándar de un flujo que pasó de activo a inactivo.
Idle Min	Duración mínima de un flujo que pasó de inactivo a activo.
Media inactiva	Duración media de un flujo que pasó de inactivo a activo.

Idle Max	Duración máxima de un flujo que pasó de inactivo a activo.
Idle Std	Duración de desviación estándar que pasó de inactivo a activo.

Nota: Tomado de la investigación de la University New Brunswick (Lashkari, 2018).

Generación de la base de conocimiento con base en las características de malware.

Cuando se trabaja con algoritmos de aprendizaje automático se tiene que tener en cuenta que los datos que sirvan para el aprendizaje, deben contener una información limpia , lo que genera un conocimiento puro y que ayuda a mejorar la base de datos , para ello se inicia un procedimiento de limpieza que abarca eliminar valores nulos, penalización de las variables y muchos más datos atípicos, siendo esta tarea importante al momento de generar una predicción ya que el conjunto da datos predictores tiene que ser el mejor (Girones, Casas, Minguillón, & Caiguelas, 2017). Es por ello que en varias situaciones de detección el objetivo principal no es sólo hacer las predicciones más precisas de la respuesta, si no también identificar que variables predictoras son las más importantes o relevantes para generar resultados fiables.

Conociendo claramente la importancia de este procedimiento se han eliminado las características correlativas o irrelevantes aplicando un esquema recursivo de eliminación de características basado en el algoritmo Recursive Feature Elimination.

Algoritmo Recursive Feature Elimination (RFE)

La técnica de selección de características tiene como objetivo principal hallar un grupo reducido de las variables originales que permita mejorar, o en todo caso no empeorar, el desempeño del método del modelo que se aplica en una base de datos en estudio, atenuando el problema de la dimensionalidad, así como también sintetiza los modelos utilizados, obteniendo como resultado una representación más simple de los datos.

Realizar la selección mediante el algoritmo RFE es principalmente trabajar con un proceso recursivo que clasifica las características de la base de datos dependiendo de la importancia de alguna medida, como se puede mostrar en el pseudocódigo en la Figura 11 (Granito, Furlanello, Biasioli, & Gasperi, 2006), en este proceso se aprecia que en cada iteración que se realiza el algoritmo RFE es capaz de medir la importancia de las características y de esa forma eliminar aquella que sea menos relevante hasta alcanzar el número especificado de características excluyendo la colinealidad y dependencias que pueda existir en el modelo.

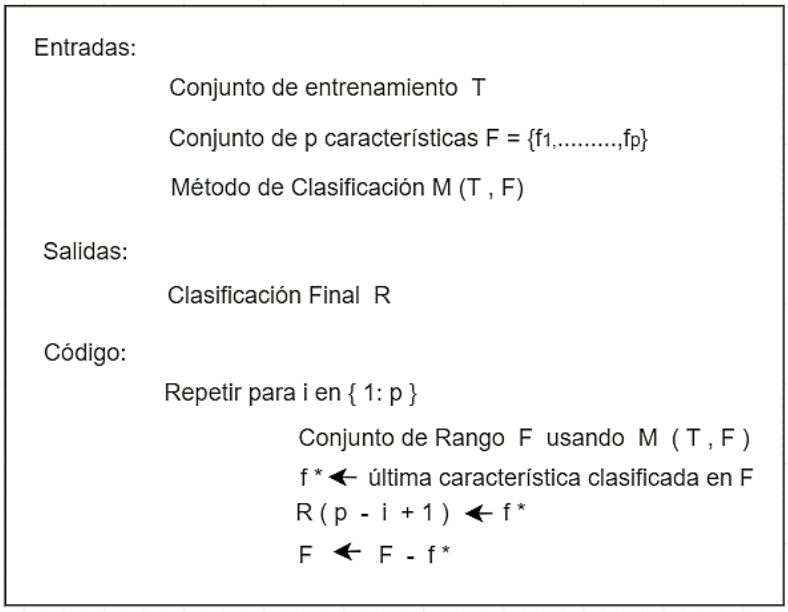

Figura 11. Pseudocódigo para el algoritmo Recursive Feature Elimination (RFE). Fuente: (Granito, Furlanello, Biasioli, & Gasperi, 2006)

A su vez para hallar la cantidad óptima de características, se aplica la validación cruzada con RFE para poder generar puntuaciones en distintos subconjuntos de características, permitiendo seguidamente la selección del grupo de características con la mejor puntuación, en esta parte del proceso la recursividad es esencial porque cuando se evalúa en relación a un subconjunto distinto de características durante el proceso de eliminación gradual , la importancia correspondiente de cada característica puede sufrir cambios relacionados a ciertas medidas , y de esta forma se genera una clasificación final basada en el orden en que se eliminan las características. Finalizando el proceso tomando las n primeras características de este ranking (7

características en este trabajo), en la Figura 12 se visualiza el proceso principal del algoritmo RFE , de la misma manera se muestra en el Anexo 5 y Anexo 6 la codificación del algoritmo, en cuyo procedimiento se ha capacitado a este clasificador Random Forest con el conjunto de datos y todas las características, eliminando aquellas con la puntuación más baja en cuanto a importancia, obteniéndose la selección de 7 características relevantes para el trabajo de detección de malware con Random Forest y 7 características descriptivas relevantes para Decisión Tree y KNN, basándose en el trabajo propuesto por (Bhadauria & Kumar, 2020).

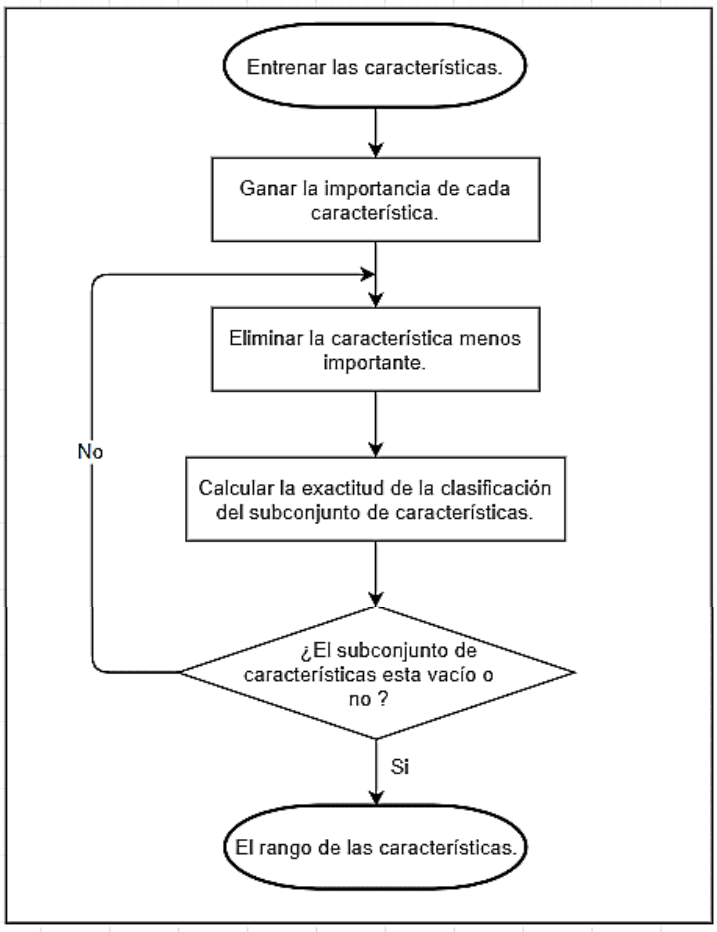

Figura 12. El procedimiento principal del método de eliminación recursiva de características (RFE). Fuente: (Shi, y otros, 2021)

Tabla 11.

Características relevantes generadas con el Algoritmo RFE para trabajar con Random Forest.

Nombre de la Característica	Descripción
Flow IAT Max	Duración máxima de 2 paquetes que se han enviado en el flujo.
Flow IAT Min	Duración mínima de 2 paquetes que se han enviado en el flujo.
Init_Win_bytes_backward	Total de bytes enviados en la ventana de inicio en bajada.
Duración de flujo	Tiempo de duración del flujo de red (µs.).
Puerto de inicio	Puerto de inicio del flujo de red
Paquetes FWD / s	Cantidad de paquetes reenviados de subida.
Paquetes Bwd / s	Cantidad de paquetes reenviados de bajada.

Nota: Tomado de la investigación de la UNB (Lashkari, 2018).

Tabla 12.

Características relevantes generadas con el Algoritmo RFE para trabajar con Decisión Tree y KNN.

Nombre de la Característica	Descripción
Flow IAT Mean	Media de tiempo de 2 paquetes que se han enviado en el flujo.
Flow IAT Min	Duración mínima de 2 paquetes que se han enviado en el flujo.
Fwd IAT Min	Duración mínima de 2 paquetes enviados en subida.
Variación de longitud de paquete	Dimensión de variación de un paquete.
Init_Win_bytes_backward	Total de bytes enviados en la ventana de inicio en bajada.
Duración de flujo	Tiempo de duración del flujo de red (µs).
Puerto de inicio	Puerto de inicio del flujo de red

Nota: Tomado de la investigación de la UNB (Lashkari, 2018).

A continuación, se describe la función Python utilizada para poder extraer las características descriptoras del programa maligno (malware) expuestas en la Tabla 11 y Tabla 12.

a) **Selección de las características relevantes antes definidas.**

En esta parte se define una función para la extracción de la información utilizando los datos descriptivos.

Tabla 13.

Función que recibe un parámetro correspondiente al dataframe para que se lea de la base de datos y seleccione las características descriptivas.

```
def get_dataset(df, opc='a'):
    colums_clean = df.columns.str.strip()
    df.columns = colums_clean
    columnas_selec = ['Source Port','Flow Duration','Flow IAT Mean',
                      'Flow IAT Min','Fwd IAT Min','Packet Length
                      Variance','Init_Win_bytes_backward','Label']
    df = df[columnas_selec]
    return df
```

Nota: Elaboración propia.

b) **Análisis Exploratorio Inicial**

Se realiza un procedimiento para leer la base de datos que se encuentra en extensión csv utilizando la librería panda con su método para estos archivos llamado red_csv, y también se puede trabajar con un archivo en excel con extensión .xlsx, pero usando la función read_excel ().

Tabla 14.

Función para la lectura de datos en extensión csv.

```
data = pd.read_csv("data/datafinal.csv")
```

Nota: Elaboración propia.

Implementar en un determinado lenguaje de programación cada uno de los algoritmos seleccionados.

La implementación de la investigación es desarrollada sobre un computador con las siguientes características.

Tabla 15.

Descripción del hardware y software donde se implementan los modelos de inteligencia artificial.

Python implementation	: CPython
Python version	: 3.8.0
IPython version	: 7.28.0
Compiler	: MSC v.1916 64 bit (AMD64)
OS	: Windows
Release	: 10
Machine	: AMD64
Processor	: Intel64 Family 6 Model 142 Stepping 9, GenuineIntel
CPU cores	: 4
Architecture	: 64bit

Nota: Elaboración propia.

a) Cuantificación del tiempo de ejecución

En esta parte se utiliza una función para calcular el tiempo de ejecución, donde tenemos que indicar dos parámetros, la hora inicio y la hora de fin, donde se calcula la diferencia, devolviendo el resultado en segundos.

Tabla 16.

Función para calcular el tiempo de ejecución.

```
def get_time_seconds(hora_fin, hora_inicio):
    diferencia = hora_fin - hora_inicio
    return diferencia.days * 24 * 3600 + diferencia.seconds
```

Nota: Elaboración propia.

b) Segmentación uniforme de los datos

La segmentación uniforme de los datos tiene como finalidad dividir los datos descriptivos y la clase objetivo para poder trabajar con algoritmos clasificadores supervisados, de donde se determinar que la variable “X” pertenece a los datos descriptivos, en este caso los malware y la variable “y” pertenece a la clase objetivo, en este caso la etiqueta (Label) correspondiente al tipo de malware que le corresponde cada vector.

Tabla 17.

Función que extrae los datos descriptivos y la clase objetivo.

```
def get_X_y(data, opc=7):
    col = data.columns.values.tolist()
    pred = col[:opc] # Valores descriptores
    obj = col[opc] # Objetivo
    X = data[pred]
    y = data[obj]
    return X, y
```

Nota: Elaboración propia.

c) División de los datos, en entrenamiento y prueba.

En esta parte se segmenta la información de acuerdo al tamaño establecido para el entrenamiento y las pruebas de la base de datos, obteniendo como como resultado 4 conjuntos de datos:

X_train = Datos descriptivos para entrenamiento.
X_test = Datos descriptivos para pruebas.
y_train = Datos objetivo para entrenamiento.
y_test = Datos objetivo para pruebas.

Tabla 18.

Función para segmentar la información.

```
def get_train_test(data, t_size, len_data=4):
    from sklearn.model_selection import train_test_split
    X,y = get_X_y(data=data, opc=len_data)
    X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=t_size)
    return X_train, X_test, y_train, y_test
```

Nota: Elaboración propia.

Validación Cruzada

Es una técnica que recibe dentro de un modelo datos descriptivos y la clase objetivo, siendo fundamental determinar una variable "k", que es la correspondiente a la cantidad de ejecuciones que va a realizar esta técnica. A su vez se ha optado por escoger esta técnica ya que permita generar un análisis estadístico basado en una serie de repeticiones con datos de entrenamiento y prueba, además la base de datos con la que se trabaja no es muy extensa y los datos están desbalanceados, por lo cual el uso de Validación Cruzada se realiza para que los resultados obtenidos sean validados mediante la media aritmética con los valores de cada repetición. En la Tabla 19 se muestra la función para generar la validación cruzada, y además los indicadores a utilizarse junto con los resultados de este modelo establecido.

Tabla 19.

Función para generar la validación cruzada

```
def print_accuracy_report(modelo, X, y, k=10):

    Vector con los indicadores:
    puntua = ['accuracy', 'precision_m', 'recall_m']

    Resultados de los indicadores:
    resultado = cross_validate(estimator=modelo,
                               X=X,
                               y=y,
                               cv=cv_kfold,
                               puntua=puntua,
```

```
                                    return_train_score=False)
    fit_tiempo = np.around(results['fit_tiempo'], 3)
    score_tiempo = np.around(results['score_tiempo'], 3)
    prueba_exactitud = np.around(results['prueba_exactitud'], 2)
    prueba_precision_m = np.around(results['prueba_precision_m'],
    2)
    prueba_recall_m = np.around(results['´prueba_recall_m'], 2)
```

Nota: Elaboración propia.

Algoritmo Random Forest

La configuración del modelo para realizar la prueba se describe a continuación:

a) Se segmenta la información de los datos descriptores y de la clase objetivo (X,y), teniendo en cuenta la base de datos generada con la selección de características con el algoritmo RFE .

Tabla 20.

Segmentación de la base de datos generada por el algoritmo RFE.

```
data = pd.read_csv("data_RFE_RF.csv")

col = data.columns.values.tolist()
pred = col[:-1]      # Valores descriptores

obj = col[-1]      # Objetivo
X = data[pred]
y = data[obj]
```

Nota: Elaboración propia.

b) Se establece el tiempo de inicio de la ejecución del algoritmo Random Forest.

Tabla 21.

Paquete datetime para dar inicio al tiempo de ejecución.

```
hora_inicio = datetime.now()
```

Nota: Elaboración propia.

c) Luego se crea el modelo y se entrena con las instancias generadas anteriormente para generar resultados como se observa en la Tabla 24.

Tabla 22.

Código para la creación y entrenamiento del modelo Random Forest

```
modelo_bosque = RandomForestClassifier(n_jobs=2, oob_score=True,
n_estimators=100)

modelo_bosque.fit(X.values,y)
```

Nota: Elaboración propia.

d) Se establece el tiempo final, para luego calcular el tiempo general de la ejecución del algoritmo Random Forest.

Tabla 23.

Código para calcular el tiempo de ejecución del algoritmo Random Forest

```
hora_fin = datetime.now()

tiempo_rf = get_time_seconds(hora_fin, hora_inicio)
print("\nTiempo de de ejecusión para Random forest = %d segundos"
%(tiempo_rf))
```

Nota: Elaboración propia.

Tabla 24.

Resultados del entrenamiento y prueba del algoritmo Random Forest

Tiempo de entrenamiento	: 2.005ms
Tiempo de la prueba	: 0.061ms
Exactitud	: 96%
Precisión	: 92.90%
Recall	: 87.120%
Error	: 4

Nota: Elaboración propia.

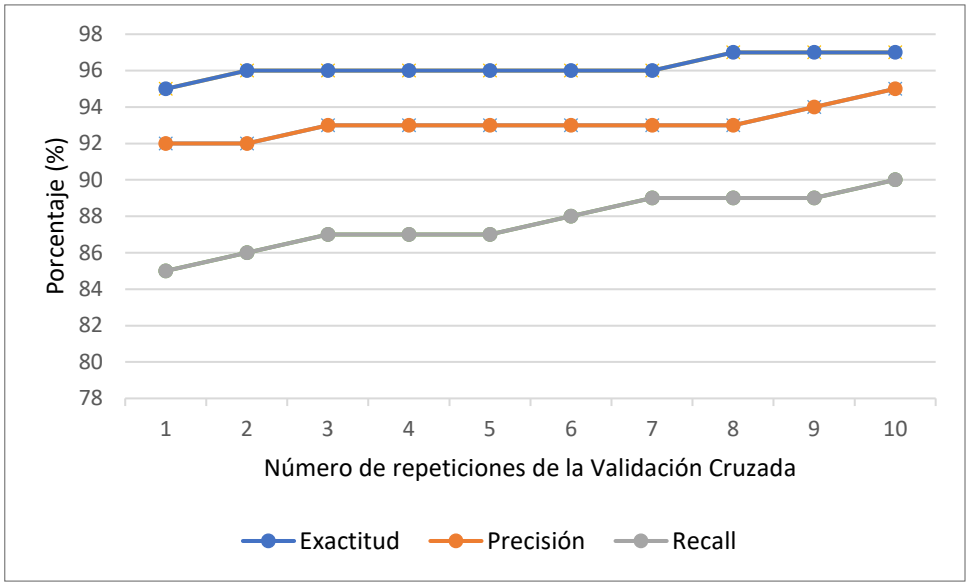

Figura 13. Resultados de los indicadores durante las repeticiones de la Validación Cruzada con el algoritmo Random Forest.

Algoritmo K-Nearest Neighbor (KNN)

La configuración del modelo para realizar la prueba se describe a continuación:

a) Se segmenta la información de los datos descriptores y de la clase objetivo (X,y), tomando como referencia la base de datos inicial donde se seleccionan las 7 características más relevantes.

Tabla 25.

Segmentación de la base de datos con las 7 características más relevantes.

```
datos_ini =  pd.read_csv("data/datafinal.csv")
datos = get_dataset(datos_ini)
X,y = get_X_y(data=datos, opc=7)
X_train, X_test, y_train, y_test = get_train_test(data=datos,
t_size=0.2)
```

Nota: Elaboración propia.

b) Se establece el tiempo de inicio de la ejecución del algoritmo KNN.

Tabla 26.

Paquete datetime para dar inicio al tiempo de ejecución.

```
hora_inicio = datetime.now()
```

Nota: Elaboración propia.

c) Luego se crea el modelo y se entrena con las instancias generadas anteriormente para generar resultados como se observa en la Tabla 29.

Tabla 27.

Código para la creación y entrenamiento del modelo KNN

```
modelo_vecino = KNeighborsClassifier(n_neighbors=16)
modelo_vecino = modelo_vecino.fit(X.values, y)
```

Nota: Elaboración propia.

d) Se establece el tiempo final, para luego calcular el tiempo general de la ejecución del algoritmo k-NN.

Tabla 28.

Código para calcular el tiempo de ejecución del algoritmo Random Forest

```
hora_fin = datetime.now()

tiempo_knn = get_time_seconds(hora_fin, hora_inicio)
print("\nTiempo de de ejecusión para KNN = %d segundos"
%(tiempo_rf))
```

Nota: Elaboración propia.

Tabla 29.

Resultados del entrenamiento y prueba del algoritmo KNN

Tiempo de entrenamiento	: 0.037ms
Tiempo de la prueba	: 0.089ms
Exactitud	: 85.4%
Precisión	: 73.9%
Recall	: 72.3%
Error	: 14.6

Nota: Elaboración propia.

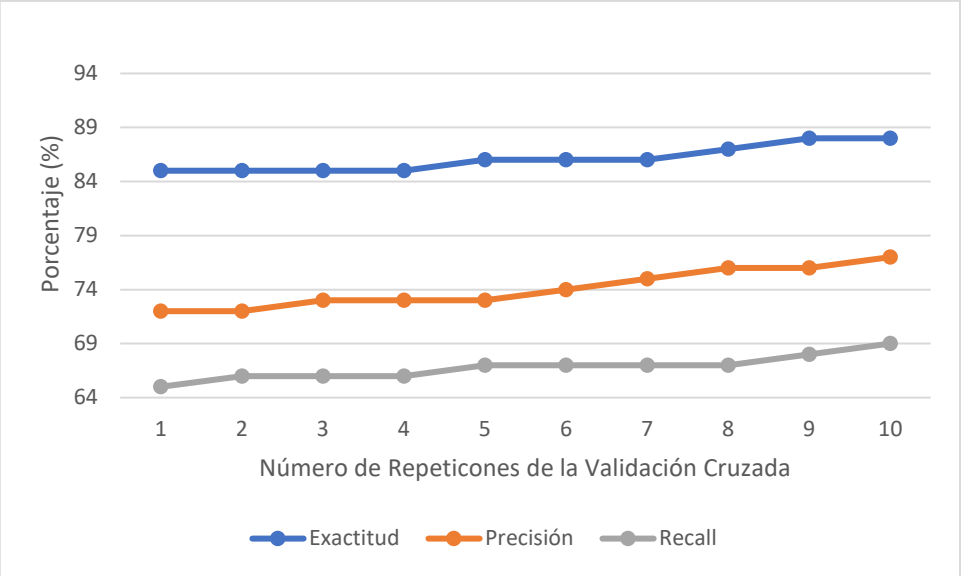

Figura 14. Resultados de los indicadores durante las repeticiones de la Validación Cruzada con el algoritmo k-NN.

Algoritmo Decision Tree

La configuración del modelo para realizar la prueba se describe a continuación:

a) Se segmenta la información de los datos descriptores y de la clase objetivo (X,y), tomando como referencia la base de datos generada con la selección de características con el algoritmo RFE .

Tabla 30.

Segmentación de la base de datos generada por el algoritmo RFE.

```
data = pd.read_csv("data_RFE_DT.csv")

col= data.columns.values.tolist()
pred = col[:-1]      # Valores descriptores

obj = col[-1]      # Objetivo
X = data[pred]
y = data[obj]
```

Nota: Elaboración propia.

b) Se establece el tiempo de inicio de la ejecución del algoritmo Decision Tree.

Tabla 31.

Paquete datetime para dar inicio al tiempo de ejecución.

```
hora_inicio = datetime.now()
```

Nota: Elaboración propia.

c) Luego se crea el modelo y se entrena con las instancias generadas anteriormente para generar resultados como se observa en la Tabla 34.

Tabla 32.

Código para la creación y entrenamiento del modelo KNN

```
model_arbol = tree.DecisionTreeClassifier()
model_arbol = modelo.arbol.fit(X, y)
```

Nota: Elaboración propia.

d) Se establece el tiempo final, para luego calcular el tiempo general de la ejecución del algoritmo Decision Tree.

Tabla 33.

Código para calcular el tiempo de ejecución del algoritmo Random Forest

```
hora_fin = datetime.now()

tiempo_knn = get_time_seconds(hora_fin, hora_inicio)
print("\nTiempo de de ejecusión para RF = %d segundos"
%(tiempo_rf))
```

Nota: Elaboración propia.

Tabla 34.

Código para calcular el tiempo de ejecución del algoritmo Random Forest

Tiempo de entrenamiento	: 0.128ms
Tiempo de la prueba	: 0.027ms
Exactitud	: 91.20%
Precisión	: 83.10%
Recall	: 86.10%
Error	: 8.8%

Nota: Elaboración propia.

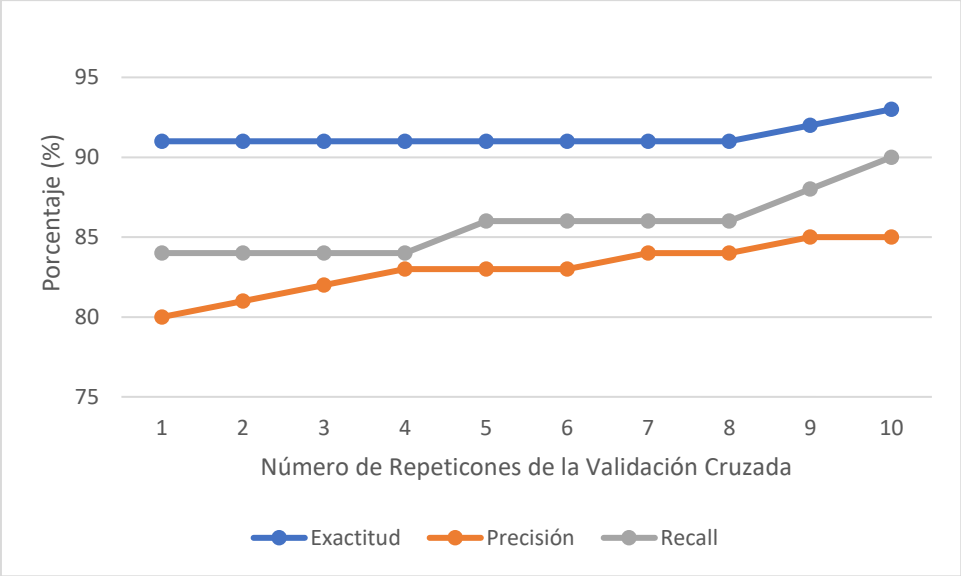

Figura 15. Resultados de los indicadores durante las repeticiones de la Validación Cruzada con el algoritmo Decisión Tree.

IV. CONCLUSIONES Y RECOMENDACIONES

4.1. Conclusiones.

Teniendo en cuenta los resultados obtenidos en la ejecución del método propuesto para la detección del malware analizando algoritmos supervisados de machine learning Random Forest, Decisión Tree y K-Nearest Neighbor (k-NN), podemos determinar que:

a) La selección de los algoritmos de aprendizaje automático presentados en la propuesta está meticulosamente dirigida con énfasis en su uso en trabajos publicados anteriormente en relación con la regla de búsqueda específica “Supervisad Algorithms AND Detection AND Malware AND Android”. Por

este motivo, se seleccionaron Random Forest, Decision Tree y k-NN, que son los algoritmos más utilizados para la detección de malware en diversas propuestas.

b) La caracterización del comportamiento dinámico del malware se basa estrictamente en el tráfico de red basándose en el enfoque propuesto por NetFlowMeter.

c) Para la propuesta se tuvo a bien utilizar un conjunto de 15945 datos de muestras descriptivas de malware de las cuales se segmenta el 80% para entrenamiento y el 20% para pruebas, además se utilizó el algoritmo REF para la extracción de las características más relevantes.

d) La implementación de la propuesta se realiza utilizando el lenguaje de programación Python, utilizado comúnmente en el desarrollo de aplicaciones relacionadas con el aprendizaje automático. Con respecto al resultado obtenido, se determina que satisface cada uno de los objetivos que se logró plantear: selección de los algoritmos supervisados, caracterización del ataque de malware, generación de la base de conocimiento, implementación de los algoritmos y evaluación de resultados.

e) La evaluación de la propuesta de identificación del malware a través flujo de red demostró que utilizando Random Forest (RF) obtuvo un 96% de exactitud, Decision Tree (DT) con 91.2% de exactitud y K-Nearest Neighbor (k-NN) con 85.4% de exactitud, con un promedio error 4%, 8.8% y 14.6% respectivamente, en un tiempo de 24s, 3s y 2s respectivamente. Demostrando que el algoritmo RF es el más eficiente en la detección de aplicaciones maliciosas en android.

4.2. Recomendaciones.

a) Se recomienda validar la selección de algoritmos supervisados, para lo cual se debe generar una tabla de los algoritmos donde se especifique el uso de estos en las diferentes investigaciones previas, antes de que decidamos con cuál de ellos debemos trabajar.

b) A fines de obtener mejores características, se recomienda trabajar con una base de datos de ataques de malware basados en análisis dinámico y que no hayan sido ejecutados en ambientes simulados, si no reales.

c) Durante el proceso de limpieza de datos, es recomendable eliminar cualquier dato atípico, como nulos, variables textuales, etc., que pueden influir en el aprendizaje del algoritmo, debemos trabajar con métodos de selección de caracteres para poder generar una base de datos reducida aportado el mismo valor descriptivo de un aplicativo android sin incurrir en demasiados costes computacionales.
d) En la implementación de los algoritmos, se recomienda trabajar con un lenguaje de programación que permita instalar todos los paquetes y librerías que se requiere en los algoritmos supervisados, tanto para entrenamiento como pruebas de la base de datos. Así también, contar con un ambiente controlado (software y hardware) que permitan trabajar con una cantidad mayor a 10000 muestras.
e) Con la finalidad de incrementar los valores de los indicadores se recomienda trabajar con cantidades de muestras superior a 10000 y reducir la cantidad de características seleccionando las más relevantes, para lograr un mejor desempeño en la identificación del malware y un mejor rendimiento del algoritmo.

REFERENCIAS.

Ableson, W., Sen, R., King, C., & Ortiz, E. (2012). *Android in Action.* New York: Manning Publications Co.

Abuthawabeh, M., & Mahmoud, K. (2019). Android Malware Detection and Categorization Based on Conversation-level Network Traffic Features. *International Arab Conference on Information Technology* , 42-47.

Agrawal, P., & Trivedi, B. (2020). Evaluating Machine Learning Classifiers to detect Android Malware. *International Conference for Innovation in Technology*, 1-6.

Akhuseyinoglu, N., & Akhuseyinoglu, K. (2016). AntiWare: An automated Android malware detection tool based on machine learning approach and official market metadata. *th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference*, 1-7.

Al Saraha, N., Rifata, F., Hossainb, S., & Narmanc, H. (2021). An Efficient Android Malware Prediction Using Ensemble machine learning algorithms. *International Conference on Mobile Systems and Pervasive Computing*, 184-191.

Alghamdi, R., Alfalqi, K., & Waqdan, M. (2015). Android Platform Malware Analysis. *International Journal of Advanced Computer Science and Applications*, 140-146.

Arora, A., & Peddoju, S. (2017). Minimizing Network Traffic Features for Android Mobile Malware Detection. *18th International Conference on Distributed Computing and Networking*, 1-10.

Barsiya, T., Gyanchandani, M., & Wadhwani, R. (2016). Android Malware Analysis : a Survey Paper. *International Journal of Control, Automation, Communication and Systems*, 35-42.

Bhadauria, P., & Kumar, J. (2020). INTRUSION DETECTION USING GRNN AND RANDOM FOREST. *https://www.bibliomed.org/?mno=137933*.

Brownlee, J. (2016). *Master Machine Learning Algorithms.* Australia: Jason Brownlee.

Frenklach, T., Cohen, D., Shabtai, A., & Puzis, R. (2021). Android malware detection via an app similarity graph. *Computers & Security*.

Gargenta, M. (2011). *Learning Android.* EE.UU.: O'Reilly Media; 1er edición.

Girones, J., Casas, J., Minguillón, J., & Caiguelas, R. (2017). *Minería de datos: modelos y algoritmos.* Barcelona: Editorial UOC.

Granito, P., Furlanello, C., Biasioli, F., & Gasperi, F. (2006). Recursive feature elimination with random forest for PTR-MS analysis of agroindustrial products. *Chemometrics and Intelligent Laboratory Systems 83*, 83-90.

Guerra, M. (2018). *Análisis de Malware para Sistemas Windows.* Madrid: RA-MA Editorial.

Guerra-Manzanares, A., Bahsi, H., & Nõmm, S. (2021). KronoDroid: Time-based Hybrid-featured Dataset for Effective Android Malware Detection and Characterization. *Computer & Security*, 1-32.

Hackeling, G. (2014). *Mastering Machine Learning with scikit-learn.* New York: Packt Publishing Ltd.

Hansen, S., Larsen, T., Stevanovic, M., & Pedersen, J. (2016). An Approach for Detection and Family Classification of Malware Based on Behavioral Analysis. *International Conference on Computing, Networking and Communications (ICNC)*, 1-5.

Jamil, Q., & Shah, M. (2016). Analysis of machine learning solutions to detect malware in android. *Sixth International Conference on Innovative Computing Technology*, 226-232.

Koli, J. (2018). RanDroid: Android malware detection using random machine learning classifiers. *Technologies for Smart-City Energy Security and Power (ICSESP)*, 1-6.

Kühnel, M., & Meyer, U. (2016). Classification of Short Messages Initiated by Mobile Malware. *1th International Conference on Availability, Reliability and Security* , 618-627.

Kumar, S., Viinikainen, A., & Hamalainen, T. (2017). Evaluation of ensemble machine learning methods in mobile threat detection. *International Conference for Internet Technology and Secured Transactions*, 261-268.

Lashkari, A. (2018). *Network Traffic Flow Analyzer.* Obtenido de CICFlowMeter: https://github.com/ahlashkari/CICFlowMeter/blob/master/ReadMe.txt

Lashkari, A., Kadir, A., Gonzalez, H., Mbah, K., & Ghorbani, A. (2017). owards a Network-Based Framework for Android Malware Detection and

Characterization. *5th Annual Conference on Privacy, Security and Trust (PST)*, 233-23309.

Lashkari, A., Kadir, A., Taheri, L., & Ghorbani, A. (2018). Toward Developing a Systematic Approach to Generate Benchmark Android Malware Datasets and Classification. *International Carnahan Conference on Security Technology (ICCST)*, 1-7.

Lu, T., & Hou, S. (2018). A Two-Layered Malware Detection Model Based on Permission for Android. *IEEE International Conference on Computer and Communication Engineering Technology*, 239-243.

Manzano, C., Meneses, ,., & Leger, P. (2020). An Empirical Comparison of Supervised Algorithms for Ransomware Identification on Network Traffic. *39th International Conference of the Chilean Computer Science Society* , 1-7.

Razgallah, A., Khoury, R., Halle, S., & Khanmohammdi, K. (2021). A survey of malware detection in Android apps: Recommendations and perspectives for future research. *Computer Science Review*, 1-17.

Sanz, I., Lopez, M., Viegas, E., & Sanches, V. (2020). A Lightweight Network-based Android Malware Detection System. *IFIP Networking Conference (Networking)*, 695-703.

Saraha, N., Rifata, F., Hossainb, S., & Narman, H. (2021). An Efficient Android Malware Prediction Using Ensemble machine learning algorithms. *18th International Conference on Mobile Systems and Pervasive Computing*, 184-191.

Shah, K., & Singh, D. (2015). A Survey on Data Mining approaches for Dynamic Analysis of Malwares. *International Conference on Green Computing and Internet of Things (ICGCIoT)*, 495-499.

Shi, H., Pan, Y., Yang, F., Cao, J., Yang, A., Xinlong, B., . . . Jiang, J. (2021). Nano-SAR Modeling for Predicting the Cytotoxicity of Metal Oxide Nanoparticles to PaCa2. *Environmental Toxicology*.

Syrris, V., & Geneiatakis, D. (2021). On machine learning effectiveness for malware detection in Android OS using static analysis data. *Journal of Information Security and Applications 59*.

Syrris, V., & Geneiatakis, D. (2021). On machine learning effectiveness for malware detection in Android OS using static analysis data. *Journal of Information Security and Applications*, 1-22.

Taheri, L., Kadir, A., & Lashkari, A. (2019). Extensible Android Malware Detection and Family Classification Using Network-Flows and API-Calls. *International Carnahan Conference on Security Technology (ICCST)*, 1-8.

University New Brunswick. (2017). *Instituto Canadiense de Ciberseguridad*. Obtenido de Instituto Canadiense de Ciberseguridad : https://www.unb.ca/cic/datasets/andmal2017.html

Wang, S., Chen, Z., Yan, Q., Yang, B., & Peng, L. (2019). A mobile malware detection method using behavior features in network traffic, en China. *Revista de aplicaciones informáticas y de red*, 15-25.

Wang, S., Chen, Z., Zhang, L., Yans, Q., Yang, B., Peng, L., & Jia, Z. (2017). TrafficAV: An effective and explainable detection of mobile malware behavior using network traffic. *24th International Symposium on Quality of Service (IWQoS)*, 1-6.

ANEXOS.

Anexo 1. Resolución de aprobación de tema de tesis.

Universidad Señor de Sipán

FACULTAD DE INGENIERÍA, ARQUITECTURA Y URBANISMO

RESOLUCIÓN N°0571-A-2022/FIAU-USS

Pimentel, 19 de septiembre de 2022

VISTO:
El Acta de reunión N°0509-2022 del Comité de investigación de la Escuela profesional de INGENIERÍA DE SISTEMAS remitida mediante Oficio 0195-2022/FIAU-IS-USS de fecha 5 de septiembre de 2022, y;

CONSIDERANDO:

Que, de conformidad con la Ley Universitaria N° 30220 en su artículo 48º que a letra dice: "La investigación constituye una función esencial y obligatoria de la universidad, que la fomenta y realiza, respondiendo a través de la producción de conocimiento y desarrollo de tecnologías a las necesidades de la sociedad, con especial énfasis en la realidad nacional. Los docentes, estudiantes y graduados participan en la actividad investigadora en su propia institución o en redes de investigación nacional o internacional, creadas por las instituciones universitarias públicas o privadas.";

Que, de conformidad con el Reglamento de grados y títulos en su artículo 21° señala: "Los temas de trabajo de investigación, trabajo académico y *tesis* son *aprobados por el Comité de Investigación* y derivados a la facultad o Escuela de Posgrado, según corresponda, para la emisión de la resolución respectiva. El *periodo de vigencia de los mismos será de dos años*, a partir de su aprobación. En caso un tema perdiera vigencia, el Comité de Investigación evaluará la ampliación de la misma.

Facultad de Ingeniería, Arquitectura y Urbanismo
UNIVERSIDAD SEÑOR DE SIPÁN SAC.

Que, de conformidad con el Reglamento de grados y títulos en su artículo 24º señala: La tesis es un estudio que debe denotar rigurosidad metodológica, originalidad, relevancia social, utilidad teórica y/o práctica en el ámbito de la escuela profesional. Para el grado de doctor se requiere una tesis de máxima rigurosidad académica y de carácter original. Es individual para la obtención de un grado; *es individual o en pares para obtener un título profesional*. Asimismo, en su artículo 25° señala: "El tema debe responder a alguna de las líneas de investigación institucionales de la USS S.A.C.".

Que, mediante documentos de vistos, el Comité de investigación de la referida Escuela profesional acordó aprobar de tema tesis que se detallan en el Acta de reunión N° 0509 - 2022, de la línea de investigación de INFRAESTRUCTURA, TECNOLOGÍA Y AMBIENTE, a a cargo de los estudiantes y /o egresados del Programa de estudios INGENIERÍA DE SISTEMAS, hasta la fecha que indica la presente resolución.

Estando a lo expuesto, y en uso de las atribuciones conferidas y de conformidad con las normas y reglamentos vigentes;

SE RESUELVE:

ARTÍCULO ÚNICO: APROBACION, de tema la Tesis a cargo de los estudiantes y /o egresados del Programa de estudios de **INGENIERÍA DE SISTEMAS** que se detallan en el anexo de la presente Resolución.

Universidad Señor de Sipán

FACULTAD DE INGENIERÍA, ARQUITECTURA Y URBANISMO

RESOLUCIÓN N°0571-A-2022/FIAU-USS

Pimentel, 19 de septiembre de 2022

ANEXO

APROBACION DE TEMA DE TESIS

APELLIDOS	TESIS
BARRERA TORRES JORGE LUIS	GESTIÓN DEL SERVICIO BASADO EN ITIL V3 PARA EVALUAR LA CALIDAD DEL SOFTWARE DE RECAUDACIÓN DE AGUA POTABLE Y ALCANTARILLADO DE LA EPS JUCUSBAMBA E.I.R.L.
ALDANA MANZANARES DAVID BENJAMIN FERNANDEZ COLINA KATTIA MILAGROS	IMPLEMENTACIÓN DE ITIL V.4.0 PARA EL PROCESO DE GESTIÓN DE INCIDENCIAS EN EL ÁREA DE INFORMÁTICA DE LA COOPERATIVA DE AHORRO Y CRÉDITO SAN MARTIN
MARTINEZ CASUSOL ENRIQUE VALENTIN MENDOZA HUANGAL WALTER	APLICACIÓN DE PENTESTING PARA EL DIAGNÓSTICO DE VULNERABILIDADES DE LA CIBERSEGURIDAD EN EL SITIO WEB DE LA EMPRESA OPTIMA SECURITY SAC
MONTENEGRO GUERRERO VÍCTOR AGUSTÍN	ANÁLISIS COMPARATIVO DE ALGORITMOS DE MACHINE LEARNING PARA DETECCIÓN DE MALWARE EN APLICACIONES ANDROID
MOYA CARNERO EDISON CLAUDIO	ANÁLISIS DE RENDIMIENTO DE TECNOLOGÍAS PARA REDES DE ÁREA LOCAL DE ALTA DEMANDA DE VELOCIDAD DE CONEXIÓN

DR. VICTOR ALEXCI TUESTA MONTEZA
DECANO (E) FACULTAD DE INGENIERÍA, ARQUITECTURA Y URBANISMO
UNIVERSIDAD SEÑOR DE SIPÁN SAC.
CHICLAYO

DR. HALYN ALVAREZ VÁSQUEZ
SECRETARIO ACADÉMICO | FACULTAD DE INGENIERÍA, ARQUITECTURA Y URBANISMO
UNIVERSIDAD SEÑOR DE SIPÁN SAC.
CHICLAYO

REGÍSTRESE, COMUNÍQUESE Y ARCHÍVESE

Cc: Interesado, Archivo

Anexo 2. Ficha de observación del desempeño de los tres algoritmos en la detección de malware en aplicaciones Android.

Resultados de la Detección de Malware

Aplicaciones Procesadas :		15945			
% Entrenamiento	**:**	80%	**% Prueba**	**:**	20%
Fecha	**:**	26/11/2021			

Evaluación de los Indicadores

Algoritmo / Indicador	**Random Forest**	**K-Nearest Neighbor**	**Decision Tree**
Exactitud	96%	85.4%	91.2%
Precisión	92.9%	73.9%	83.1%
Recall	87.2%	72.3%	86.1%
Error	4%	14.6%	8.8%
Consumo de memoria RAM	295.45 mb	212.41 mb	215.77 mb
Tiempo de respuesta en la ejecución	24 seg.	2 seg.	3 seg.

Glosa:

Nota: Elaboración Propia.

Anexo 3. Código fuente del esquema recursivo de eliminación de características basado en el algoritmo RFE para Random Forest

```
# Información de las características seleccionadas por RFE
from sklearn.feature_selection import RFE
from sklearn.ensemble import RandomForestClassifier
  # definir dataset
  # definir RFE
  rfe = RFE(estimator=RandomForestClassifier(),
  n_features_to_select=7)
  # ajustar RFE
  rfe.fit(X, y)
  # resumir las características
  index_column = []
  for i in range(X.shape[1]):
      if rfe.support_[i] == True:
          index_column.append(i)
          print('Column: %d, Selected %s, Rank: %.3f' % (i,
  rfe.support_[i], rfe.ranking_[i]))

  print(f'indexs: {index_column}')
  columns = data.columns.values.tolist()
  columns_select = []
  for i in index_column:
      columns_select.append(columns[i])

  data_result = data[columns_select]
  data_result.head()
```

Nota: Elaboración propia.

Anexo 4. Código fuente del esquema recursivo de eliminación de características basado en el algoritmo RFE para Decisión Tree y K-Nearest Neighbor.

```
# Información de las características seleccionadas por RFE
from sklearn.feature_selection import RFE
from sklearn.tree import DecisionTreeClassifier
  # definir dataset
  # definir RFE
  rfe = RFE(estimator=DecisionTreeClassifier(),
  n_features_to_select=7)
  # ajustar RFE
  rfe.fit(X, y)
  # resumir las características
  index_column = []
  for i in range(X.shape[1]):
      if rfe.support_[i] == True:
          index_column.append(i)
          print('Column: %d, Selected %s, Rank: %.3f' % (i,
  rfe.support_[i], rfe.ranking_[i]))

  print(f'indexs: {index_column}')
  columns = data.columns.values.tolist()
  columns_select = []
  for i in index_column:
      columns_select.append(columns[i])

  data_result = data[columns_select]
  data_result.head()
```

Nota: Elaboración propia.

Anexo 5. Características relevantes de las 30 últimas muestras de la base de datos para el algoritmo Random Forest.

	Source Port	Flow Duration	Flow IAT Max	Flow IAT Min	Fwd Packets/s	Bwd Packets/s	Init_Win_bytes_backward	Label
15915	42291	1734869	1134281	5	10.375423	8.069774	357.0	SI
15916	51339	483363	77065	21	24.826062	24.826062	349.0	SI
15917	51339	36705	36705	36705	27.244245	27.244245	349.0	SI
15918	44752	58978760	58361160	289	0.101732	0.067821	31.0	SI
15919	44753	58951519	58347899	216	0.101779	0.067852	31.0	SI
15920	46503	5583242	4997643	257	0.716430	0.716430	30.0	SI
15921	55229	10263186	10087919	87524	0.194871	0.194871	29.0	SI
15922	55231	10262338	10088131	86691	0.194887	0.194887	29.0	SI
15923	33178	5585192	4997569	505	0.716179	0.716179	30.0	SI
15924	48228	955241	240792	5	14.655987	25.124550	160.0	SI
15925	57377	10057821	9992750	13	0.596551	0.596551	31.0	SI
15926	34409	588836	264158	12855	5.094797	5.094797	125.0	SI
15927	36882	605692	263961	10	6.604017	6.604017	160.0	SI
15928	36881	608284	263639	45	6.575876	6.575876	160.0	SI
15929	36883	609827	263671	14	6.559237	6.559237	160.0	SI
15930	57376	10903302	9997104	9	1.375730	1.375730	31.0	SI
15931	36880	1910185	1043242	648	2.617548	2.094038	160.0	SI
15932	36884	1913900	1042824	35	2.612467	2.089973	160.0	SI
15933	55014	11801836	10022099	1737	0.423663	0.169465	14600.0	SI
15934	46504	10970965	10672654	52572	0.273449	0.091150	29200.0	SI
15935	33170	10396094	10060666	86982	0.288570	0.096190	29200.0	SI
15936	57371	14025098	9998255	5	15.187060	21.960631	31.0	SI
15937	36885	3407058	2882576	71	1.174033	0.880525	160.0	SI
15938	36890	536441	263412	44	7.456552	7.456552	160.0	SI
15939	36892	536540	263447	121	7.455176	5.591382	160.0	SI
15940	36891	543594	263546	43	7.358433	7.358433	160.0	SI
15941	36893	531991	264122	73	7.518924	5.639193	160.0	SI
15942	55228	16472390	15885165	139	0.242831	0.242831	30.0	SI
15943	58116	9128442	9106529	1878	0.219095	0.219095	29.0	SI
15944	58112	10198584	9991439	5	4.216272	6.569539	27.0	SI

Anexo 6. Características relevantes de las 30 últimas muestras de la base de datos para el algoritmo Decisión Tree y K-Nearest Neighbor.

	Source Port	Flow Duration	Flow IAT Mean	Flow IAT Min	Fwd IAT Min	Packet Length Variance	Init_Win_bytes_backward	Label
15915	42291	1734869	5.596352e+04	5	10.0	177318.903409	357.0	SI
15916	51339	483363	2.101578e+04	21	122.0	174789.273333	349.0	SI
15917	51339	36705	3.670500e+04	36705	0.0	176.333333	349.0	SI
15918	44752	58978760	6.553196e+06	289	1207.0	6518.454545	31.0	SI
15919	44753	58951519	6.550169e+06	216	846.0	6518.454545	31.0	SI
15920	46503	5583242	7.976060e+05	257	257.0	31819.111111	30.0	SI
15921	55229	10263186	3.421062e+06	87524	175267.0	0.000000	29.0	SI
15922	55231	10262338	3.420779e+06	86691	174207.0	0.000000	29.0	SI
15923	33178	5585192	7.978846e+05	505	505.0	92165.111111	30.0	SI
15924	48228	955241	2.581732e+04	5	8.0	508135.923077	160.0	SI
15925	57377	10057821	9.143474e+05	13	13.0	167914.435897	31.0	SI
15926	34409	588836	1.177672e+05	12855	12855.0	90965.142857	125.0	SI
15927	36882	605692	8.652743e+04	10	10.0	313010.611111	160.0	SI
15928	36881	608284	8.689771e+04	45	225.0	313010.611111	160.0	SI
15929	36883	609827	8.711814e+04	14	35.0	313010.611111	160.0	SI
15930	57376	10903302	3.759759e+05	9	9.0	384895.929032	31.0	SI
15931	36880	1910185	2.387731e+05	648	648.0	337812.900000	160.0	SI
15932	36884	1913900	2.392375e+05	35	35.0	337812.900000	160.0	SI
15933	55014	11801836	1.966973e+06	1737	321073.0	0.000000	14600.0	SI
15934	46504	10970965	3.656988e+06	52572	298311.0	0.000000	29200.0	SI
15935	33170	10396094	3.465365e+06	86982	335428.0	0.000000	29200.0	SI
15936	57371	14025098	2.697134e+04	5	5.0	506504.615060	31.0	SI
15937	36885	3407058	5.678430e+05	71	71.0	360280.214286	160.0	SI
15938	36890	536441	7.663443e+04	44	249.0	328108.527778	160.0	SI
15939	36892	536540	8.942333e+04	121	121.0	360280.214286	160.0	SI
15940	36891	543594	7.765629e+04	43	70.0	328108.527778	160.0	SI
15941	36893	531991	8.866517e+04	73	73.0	360509.125000	160.0	SI
15942	55228	16472390	2.353199e+06	139	177308.0	32811.444444	30.0	SI
15943	58116	9128442	3.042814e+06	1878	21913.0	0.000000	29.0	SI
15944	58112	10198584	9.356499e+04	5	8.0	474292.777887	27.0	SI

Anexo 7. Código fuente de la implementación del algoritmo Random Forest.

```
# Se segmenta la información X, y
datos = pd.read_csv("data_RFE_RF7.csv")

columns = data.columns.values.tolist()
predictores = columns[:-1] # Valores descriptores

obj = columns[-1] # Objetivo
X = data[pred]
y = data[obj]

# X.head()
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2)


# Se establece el inicio de la ejecución para random utilizando el
paquete datetime con su función now ()

hora_inicio = datetime.now()

## Se importa el paquete para Random Forest

from sklearn.ensemble import RandomForestClassifier

# Se crea el modelo utilizando los parámetros para ajustar y
mejorar los resultados obtenidos

Modelo_bosque = RandomForestClassifier(n_jobs=2, oob_score=True,
                                n_estimators=100)

# Se entrena el modelo utilizando la instancia creada anteriormente
Modelo_bosque.fit(X.values,y)
get_confusion_matrix_report(modelo_bosque, X_test.values, y_test)
```

Nota: Elaboración propia.

Anexo 8. Código fuente de la implementación del algoritmo K-Nearest Neighbor.

```
## Se importa el paquete para K - Nearest Neighbors

from sklearn.neighbors import KNeighborsClassifier

#Se establece el inicio de la ejecución para k-NN utilizando el
paquete datetime con su función now ()

hora_inicio = datetime.now()

# Se segmenta la información X, y

data_r =  pd.read_csv("data_RFE_DT7.csv")
datos = get_dataset(data_r, knn=True)
X,y = get_X_y(data=datos, opc=7)

X_train, X_test, y_train, y_test = get_train_test(data=datos,
t_size=0.2)
```

Nota: Elaboración propia.

Anexo 9. Código fuente de la implementación del algoritmo Decision Tree.

```
## Se importa el paquete para Decision Tree

from sklearn import tree

#Se establece el inicio de la ejecución para Decision Tree
utilizando el paquete datetime con su función now ()

hora_inicio = datetime.now()


# Se segmenta la información X,y

data = pd.read_csv("data_RFE_DT7.csv")

columns = data.columns.values.tolist()
pred = columns[:-1] # Valores descriptores

obj = columns[-1] # Objetivo
X = data[pred]
y = data[obj]

# X.head()
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2)

# Se crea el modelo utilizando los parámetros para ajustar y
mejorar los resultados obtenidos

modelo_arbol = tree.DecisionTreeClassifier()
modelo_arbol = modelo_arbol.fit(X, y)

get_confusion_matrix_report(modelo_arbol, X_test, y_test)
```

Nota: Elaboración propia.