ESCUELA POLITECNICA NACIONAL

FACULTAD DE INGENIERÍA ELECTRÍCA Y ELECTRÓNICA / DEPARTAMENTO DE ELECTRÓNICA, TELECOMUNICACIONES Y REDES DE INFORMACIÓN

SEGURIDAD PARA REDES IOT USANDO MACHINE LEARNING.

IMPLEMENTACIÓN DE UN SISTEMA DE DETECCIÓN DE AMENAZAS UTILIZANDO REDES DEFINIDAS POR SOFTWARE.

TRABAJO DE INTEGRACIÓN CURRICULAR PRESENTADO COMO REQUISITO PARA LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN TECNOLOGÍAS DE LA INFORMACIÓN

JUAN JOSÉ NOBOA ROMO

juan.noboa@epn.edu.ec

DIRECTOR: DR. TARQUINO FABÍAN SÁNCHEZ ALMEIDA

tarquino.sanchez@epn.edu.ec

Quito, septiembre 2023

CERTIFICACIONES

Yo, JUAN JOSÉ NOBOA ROMO declaro que el trabajo de integración curricular descrito aquí es de mi completa autoría; este no ha sido presentado previamente para ningún tipo de grado o calificación profesional; y, he consultado personalmente cada una de las referencias bibliográficas que constan en este documento.

_____

JUAN JOSÉ NOBOA ROMO

Certifico que el presente trabajo de integración curricular fue desarrollado por JUAN JOSÉ NOBOA ROMO, bajo mi supervisión.

_____

DR. TARQUINO FABIÁN SÁNCHEZ ALMEIDA

DECLARACIÓN DE AUTORÍA

Por medio de la presente declaración, afirmamos que el trabajo de integración curricular aquí descrito, así como el (los) producto (s) resultantes del mismo, son públicos y estarán a disposición de la comunidad a través del repositorio institucional de la Escuela Politécnica Nacional; sin embargo, la titularidad de los derechos patrimoniales nos corresponde a los autores que hemos contribuido en el desarrollo del presente trabajo; observando para el efecto las disposiciones establecidas por el órgano competente en propiedad intelectual, la normativa interna y demás normas.

JUAN JOSÉ NOBOA ROMO

DR. TARQUINO FABIÁN SÁNCHEZ ALMEIDA

DEDICATORIA

Dedico este Trabajo de Integración Curricular a mis padres, abuelos y amigos, quienes me han brindado su incondicional apoyo a lo largo de toda mi trayectoria académica.

AGRADECIMIENTO

Agradezco en primer lugar a Dios por darme la oportunidad de culminar mi carrera y darme la fortaleza necesaria para seguir adelante todos los días, en segundo lugar quiero agradecer a mis padres y abuelos que me han brindado su incondicional apoyo y han creído en mis capacidades, también agradezco al Dr. Tarquino Sánchez por su paciencia y comprensión durante el desarrollo del presente proyecto y finalmente agradezco a mis mejores amigos José Murillo y Christian Morán, con quienes tuve la oportunidad de compartir maravillosas experiencias durante mi formación académica.

ÍNDICE DE CONTENIDO

RESUMEN

En este estudio se expone el proceso de la implementación de un sistema de detección de amenazas para una red SDN simulada en Cisco Packet Tracer, a partir de una topología que favorezca la clasificación de las tramas enviadas y monitoreadas por un controlador SDN. Para ello se emplea el lenguaje de programación Python, tanto para la comunicación con la API del controlador de la red, así como para el tratamiento de los conjuntos de datos utilizados para el diseño e implementación de los modelos de predicción con Machine Learning. Dentro del primer capítulo de este Trabajo de Integración Curricular, se desarrolla e indagan todos los fundamentos teóricos relacionados a redes IoT, seguridad de redes y algoritmos de Machine Learning orientado a modelos de predicciones para dar pie a la segunda parte que abarca la metodología del proyecto, donde se definen las herramientas, recursos y scripts en Python para lograr capturar los datos del simulador, poder conformar un conjunto de datos y evaluarlo en los diferentes modelos creados. Finalmente, en el último capítulo, se realiza una comparación de las pruebas realizadas con los algoritmos y modelos propuestos, empatando con lo que se ha consultado en las etapas teóricas, sacando conclusiones y recomendaciones de acuerdo a la experiencia adquirida durante el desarrollo de este proyecto.

PALABRAS CLAVE: Cisco Packet Tracer, Machine Learning, Python, Redes IoT, Seguridad de redes, Regresión Logística, SVM, Random Forest.

ABSTRACT

This study presents the process of the design of a threat detection system for simulated SDN network in Cisco Packet Tracer, starting with a topology that facilitates the classification of frames which are sent and monitored by an SDN controller. Python programming language is used for communication with the network controller's API and datasets processing that will be used for designing and implementing prediction models based on Machine Learning. In the first chapter of this project, all theorical foundations about IoT networks, cybersecurity and Machine Learning oriented towards prediction models are studied and explored to start the second chapter that encompasses the project methodology, defining tools, resources, and Python scripts to capture traffic data from the simulator, creating a dataset and evaluating it with different prediction models. Finally, in the last chapter, a comparison is made between all proposed algorithms and models according with the theorical stages, drawing conclusions and providing recommendations based on the gained knowledge during the development of this project.

KEYWORDS: Cisco Packet Tracer, Machine Learning, Python, IoT networks, network security, Logistic Regression, Support Vector Machine, Random Forest.

1. INTRODUCCIÓN

Con el constante avance de la tecnología, día a día nos enfrentamos a innovaciones que buscan simplificar muchas tareas de nuestra vida cotidiana, siendo IoT una de las ramas más reconocidas dentro del campo de conectividad, permitiendo al usuario interconectar varios objetos uso diario como dispositivos que se encuentran bajo una misma red, siendo administrados y controlados de manera remota por el usuario generalmente desde una aplicación en su dispositivo móvil, lo cual si bien presenta un gran número de facilidades, podría llegar a presentar un alto riesgo a nivel de seguridad de datos y privacidad del usuario [1], por lo que es necesario ser consciente de los riesgos que puede abarcar el manejo de este tipo de comunicaciones, ya que, generalmente los dispositivos asociados con redes IoT presentan características limitadas y poca memoria lo cual incrementa su vulnerabilidad, dejando ver así que los nodos IoT son débiles.

Por lo tanto en este TIC se propone la implementación de un sistema de detección de amenazas empleando redes definidas por software, dejando así a los dispositivos IoT de limitadas características la función únicamente de transmitir los datos, mientras que un controlador SDN será el encargado de clasificar las trazas que circulen dentro de la red IoT, con la ayuda de algoritmos de clasificación y decisión utilizando Machine Learning para la detección de intrusión, clasificando las trazas como “buenas” o “malas”, buscando conseguir así fortalecer la seguridad de la red y volverla más robusta frente ataques.

Para esto debemos tener en cuenta que a medida que más crecen las redes se tornan más complejas de administrar y gestionar, donde los equipos tradicionales con los que se trabajan actualmente no cuentan con la capacidad de responder en tiempo real a las demandas actuales, servicios como IoT, inteligencia artificial, aprendizaje continuo, entre otros, demandan mayor desempeño en la red, consumiendo grandes cantidades de recursos, cosa con la que los dispositivos IoT no cuentan, tornando así a estas redes como vulnerables, siendo este el punto donde entran las redes definidas por software, las cuales nos permiten separar las funciones de control y reenvío de los dispositivos de la red, teniendo a un controlador SDN encargado de las funciones de administración y control del tráfico, asegurándose de que las trazas transmitidas sean

"buenas", fortaleciendo así la seguridad de la red, mientras que se deja a dispositivos IoT y a equipos de red de capa 2 y 3 como switches y routers a los encargados de retrasmitir los datos, consumiendo así una menor cantidad de recursos y a su vez brindando una mayor seguridad en la red IoT.

1.1. Objetivo general:

- Implementar un sistema de detección de amenazas utilizando redes definidas por software, con una aproximación a una red neuronal convolucional en base a un controlador SDN desarrollado en lenguaje de programación Python.

1.2. Objetivos específicos:

- OE1: Recopilar información acerca de las redes IoT, el tráfico, protocolos de transmisión, seguridad, vulnerabilidades y compatibilidad con dispositivos de bajo consumo de recursos.
- OE2: Analizar los distintos modelos y algoritmos de clasificación de trazas y determinar la mejor opción para la posterior implementación de un controlador SDN capaz de etiquetar las trazas buenas o malas de acuerdo con su contenido.
- OE3: Implementar el sistema de detección de amenazas utilizando un modelo de Machine Learning desarrollado en lenguaje de programación Python.
- OE4: Realizar las pruebas experimentales, constatando que se clasifiquen correctamente las tramas de acuerdo con el comportamiento definido en el controlador, empleando entornos de simulación y virtualización de redes como Cisco Packet Tracer y otros.

1.3. Alcance

Para el desarrollo del presente proyecto, se ha planteado emplear una metodología de investigación basada tanto en el enfoque de la ciencia del diseño, la cual se centra en la resolución de problemas y planteamiento de ideas que permitan mediante tareas de análisis y comprensión la implementación y uso de sistemas de información, para que pueda ser afectivo, eficiente y en el caso del proyecto, que presente una mayor robustez en lo que respecta a la seguridad de las redes IoT. Para ello se va a partir de la comprensión del problema, para mediante ideas y criterios poder llegar a diseñar modelos que se presenten como la solución a dicho problema. Se ha separado el proyecto en dos componentes fundamentales, siendo estos asignados a cada uno de los estudiantes participantes del proyecto. En el

presente trabajo nos centraremos en el componente denominado "*Implementación de un sistema de detección de amenazas utilizando redes definidas por software*", cuyo desarrollo puede simplificarse en 5 etapas principales, las cuales se pueden visualizar en el siguiente diagrama de bloques presentado en la Figura 1 a continuación:

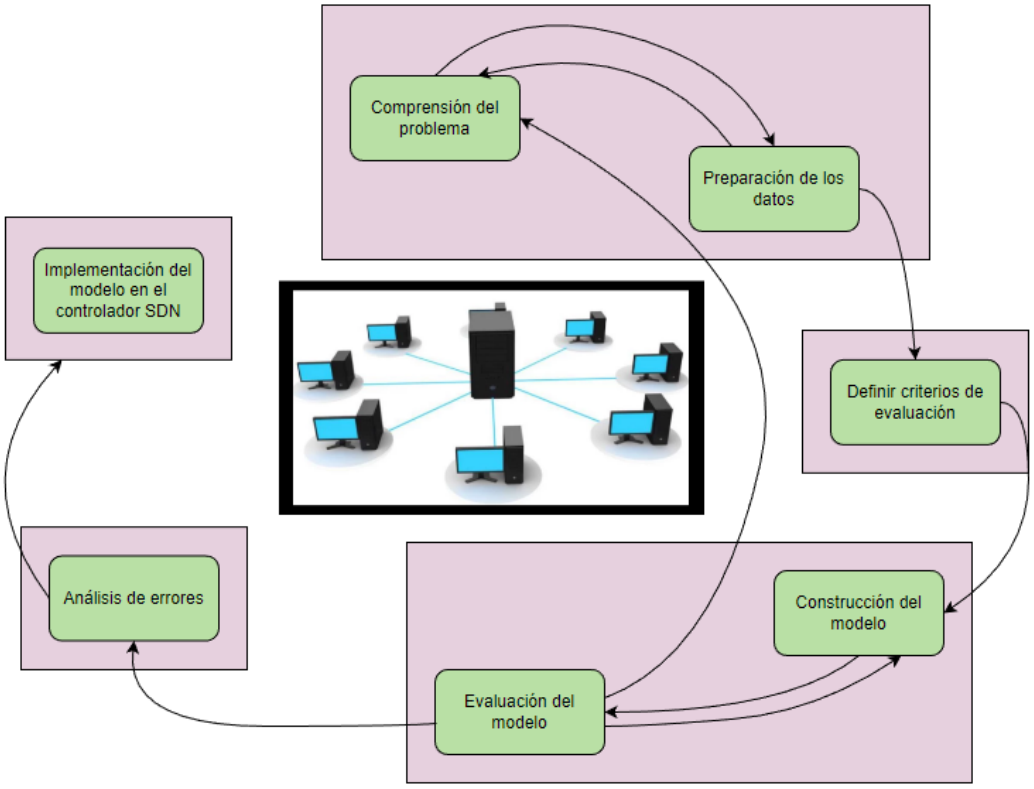


Figura 1. Diagrama simplificado de la metodología del proyecto.

En el gráfico diseñado anteriormente, el cual se encuentra basado en "Las 7 Fases del Proceso de Machine Learning" [2], se han planteado cinco etapas, las cuales serán descritas brevemente, así como los requerimientos necesarios para la elaboración de cada una de ellas, a continuación:

1.3.1. Etapa E1: Comprensión del problema y preparación de datos.

Dentro de esta etapa se va a realizar el estudio del problema propuesto, analizando cada uno de los requerimientos, así como la exploración de la o las bases de datos a analizar, las cuales van a contener tráfico generado de una simulación de una red SDN en Cisco Packet Tracer, donde, haciendo uso de la API del controlador SDN

se puede mediante Python extraer cada uno de los campos de las tramas que circulan por la red y armar un *dataset* o conjunto de datos para trabajar. Para la parte de preparación de los datos se emplea la minería de datos ya que, es esencial al momento de filtrar y normalizar los datos del conjunto de datos empleado, limpiando y dejando únicamente los atributos y registros necesarios para la elaboración del modelo de clasificación empleando Machine Learning, teniendo en cuenta la relación de tramas benignas y tramas maliciosas, de modo que la base de datos se encuentre balanceada para conseguir una mayor eficiencia del modelo desarrollado.

La base de datos [3] a emplear en el proyecto es un conjunto de datos en formato CSV, el mismo que será construido en base a los datos extraídos de las tramas de la red simulada y organizados empleando el lenguaje Python para la comunicación con la API del controlador SDN. Brevemente se puede destacar que el conjunto de datos cuenta con aproximadamente un total de 105 mil registros, cada uno de ellos con 23 campos de atributos, de los cuales algunos son extraídos de los switches, mientras que otros son calculados, de estos, el eje central será el campo denominado “label” el cual está compuesto por 1 y 0, indicando si se trata de una trama “maliciosa” para el número 0 y “benigna" para el número 1. El conjunto de datos contiene tramas ICMP, TCP y UDP generados en un simulador de redes destinado como fuente para el entrenamiento y clasificación de tráfico empleando algoritmos de aprendizaje automático y es generado a partir de una topología de red conectada a un controlador RYU, donde la simulación ejecuta tráfico benigno y malicioso, como ataques TCP Syn, inundación de paquetes UDP (UDP Flood) y ataques ICMP. Este conjunto de datos puede ser obtenido en el siguiente enlace: https://data.mendeley.com/datasets/jxpfjc64kr/1

1.3.2. Etapa E2: Definición de los criterios de evaluación.

Con una base de datos ya preparada para el desarrollo del modelo, es necesario identificar bajo qué criterios se va a desarrollar dicho modelo, para ello se necesita de la comprensión del problema realizada anteriormente donde de acuerdo a los parámetros establecidos en la base de datos, se analiza cada uno de los campos y se determina que el más relevante es “label”, el mismo que nos indicará si la trama en cuestión es maliciosa o benigna a través de los valores 0 y 1 respectivamente,

de modo que esta clasificación sea la base para la construcción del modelo de Machine Learning.

1.3.3. Etapa E3: Construcción y evaluación del modelo.

Empleando el lenguaje de programación Python con la ayuda de varias bibliotecas orientadas a Machine Learning, se va a elaborar el modelo de clasificación de tramas con ayuda de los criterios de evaluación definidos previamente, siendo este modelo capaz de separar dentro de un intervalo de tráfico las tramas “buenas o benignas” de las tramas “malas o infectadas” de un ataque DDoS, afectando así la seguridad de la red, siendo aquí donde el modelo cumple la función de clasificar el tráfico y aumentar así la robustez de la red y la confiabilidad de la misma.

1.3.4. Etapa E4: Análisis de errores.

Una vez que el modelo haya sido completamente desarrollado, este será sometido a varias pruebas con todo tipo de tráfico para poder evaluar su desempeño y analizar los potenciales errores que llegue a tener al momento de clasificar las tramas, para lo cual será necesario realizar varios escenarios de prueba, dentro de un ambiente simulado con todo tipo de datos y corregir los errores que el modelo pueda llegar a tener, donde una vez corregidos tendrán que nuevamente pasar por el proceso de evaluación del modelo, hasta que tenga un margen de error mínimo y aceptable.

1.3.5. Etapa E5: Interpretación de los resultados.

Con el modelo listo y corregido, será el momento de probar varios algoritmos aproximados a una red neuronal convolucional para evaluar los resultados de cada uno de ellos, tratando así de determinar el modelo más eficiente siendo capaz de clasificar correctamente las tramas, manteniendo así una seguridad robusta en la red simulada gracias a la implementación del modelo de clasificación de tramas.

1.4. Marco Teórico:

1.4.1. Redes IoT:

Actualmente se ha manejado con mucha frecuencia el término IoT (Internet of Things), el cual tiene como eje central los conocidos dispositivos IoT, los mismos que se pueden definir como objetos capaces de interconectarse por medio de internet, teniendo la capacidad de intercambiar datos entre sí. Las redes IoT tienen diferentes campos de aplicación, tales como en la industria, donde los dispositivos IoT, son sensores o máquinas interconectadas que intercambian datos para

potenciar su funcionalidad. Por otro lado, tenemos los smart home, los cuales se caracterizan por trabajar a nivel local y como su nombre lo dice, dentro del hogar, donde los dispositivos IoT pueden ser sensores o electrodomésticos que se encuentran interconectados para intercambiar datos que pueden ser administrados desde un software instalado en un dispositivo de control como una tablet o smartphone.

Las redes IoT se encuentran actualmente en auge, lo que ha permitido realizar la incorporación de diferentes tipos de tecnologías y funcionalidades con la finalidad de mejorar u optimizar la experiencia del usuario con la ayuda de diferentes técnicas como: análisis y tratamiento de datos, implementación de modelos de Machine Learning e inteligencia artificial, lo cual ha permitido el desarrollo de esta tecnología, aumentando así los campos de aplicación y despertando el interés de varios sectores en donde se puede implementar las funcionalidades que nos ofrece esta tecnología. [4]

1.4.2. Machine Learning:

Es considerada una disciplina científica, como rama de la inteligencia artificial, la cual se encarga de la creación de sistemas capaces de aprender de forma automatizada, por medio de la revisión e identificación de patrones complejos, resultado del análisis de millones de datos. Dejando así que la máquina sea la encargada de aprender un algoritmo con la finalidad de lograr predecir comportamientos futuros, dando la capacidad a estos sistemas de mejorar con el tiempo de manera autónoma y lo más importante sin la necesidad de que intervenga el hombre.

Machine Learning a lo largo del tiempo se está abriendo paso por diferentes sectores empresas e industrias, ya que para muchas entidades la extracción de información de información valiosa supone una ventaja competitiva. Una de las más grandes ventajas que supone el empleo de Machine Learning es que no es necesario ser un experto en ciencias de datos para lograr aprovechar todo lo que nos ofrecen este tipo de tecnologías, ya que, gracias a un amplio mercado de herramientas disponibles, de fácil uso y accesible para todo tipo de empresas. La incorporación y uso de estas tecnologías ha supuesto una simplificación enorme en el proceso de obtención de datos de calidad, los cuales mediante su análisis tenemos la posibilidad de crear sistemas o modelos de predicción de comportamientos,

proporcionándonos resultados rápidos, precisos y lo más importante sin la intervención humana. Teniendo como resultado predicciones de alto valor para la toma de mejores decisiones y desarrollo de mejores acciones de negocio. [5]

Además de tomar una posición en el campo de los negocios, se ha relacionado el Machine Learning con la redes IoT, dentro de las cuales también tenemos un sinfín de aplicaciones donde podemos hacer uso de información útil para la optimización y simplificación de procesos, sin dejar de lado uno de los pilares más importantes de las redes, la seguridad, campo en el cual ha incursionado el Machine Learning con la finalidad de darle una mayor robustez a las redes IoT frente a ataques, bien sea mediante el análisis del tráfico de la red o mediante la implementación de modelos de clasificación de trazas, los cuales por medio de un entrenamiento sean capaces de detectar de manera automática las trazas maliciosas dentro de la red o aquellas que pueden suponer un riesgo para las mismas, para el proceso de clasificación, se pueden aplicar varios modelos o algoritmos, entre los cuales caben destacar:

1.4.3. Algoritmo de Regresión Lineal:

Es una técnica de aprendizaje automático perteneciente al campo estadístico, se trata de un método de clasificación donde se obtienen valores binarios (1s y 0s), lo cual nos permitirá identificar si una determinada acción u operación es viable o no, dejando el margen únicamente dos posibilidades, midiendo así la relación existente entre una variable dependiente, que generalmente se traduce como el comportamiento que se desea predecir, con variables independientes que se definen como el conjunto de características disponibles para el modelo, de modo que se emplea una función logística que será la encargada de determinar dicha probabilidad de que ocurra la variable dependiente. [6]

1.4.4. Modelo de clasificación K-NN:

Es un modelo que se basa en una instancia del tipo “supervisado” dentro del campo de Machine Learning. Este algoritmo es empleado generalmente para clasificación de muestras, a manera de valores discretos para poder predecir valores continuos, dentro de los varios métodos de clasificación este se caracteriza por ser uno modelo sencillo, siendo ideal para incursionar dentro del campo del aprendizaje automático. Es empleado para clasificar valores empleando puntos de datos similares, basados en la cercanía de las muestras proporcionadas, las cuales son tomadas previamente

de una etapa de entrenamiento, basando su modelo en conjeturas de nuevos puntos de clasificación aprendidos automáticamente. [7]

1.4.5. Redes convolucionales:

Denominadas redes neuronales convolucionales, son un tipo de redes neuronales artificiales las cuales como su nombre lo indica están compuestas por “neuronas”, las cuales poseen campos de recepción de información, similar a las neuronas del cerebro humano. Estas redes son consideradas como una variación de los conocidos perceptrón “multicapa”. Su campo de aplicación se basa en matrices del tipo bidireccionales, las cuales son consideradas de gran desempeño para tareas de visión artificial, partiendo de ahí para la clasificación, así como la segmentación de información, como imágenes, tramas y demás datos en varios campos. Estas redes se encuentran compuestas por múltiples filtros convolucionales de varias dimensiones, donde en cada capa se añade una función específica para realizar un proceso denominado “mapeo causal no-lineal”. [8]

1.4.6. Redes Definidas por Software (SDN):

Con el constante crecimiento de las redes, estas se tornan más complejas de administrar y gestionar, donde los equipos tradicionales con los que se trabajan actualmente no cuentan con la capacidad de responder a tiempo real a las demandas actuales, servicios como IoT, inteligencia artificial, aprendizaje continuo, entre otros, demandan mayor desempeño en la red, consumiendo grandes cantidades de recursos, cosa con la que los dispositivos IoT no cuentan, tornando así a estas redes como vulnerables, siendo este el punto donde entran las redes definidas por software, las cuales nos permiten separar las funciones de control y reenvío de los dispositivos de la red, teniendo a un controlador SDN encargado de las funciones de administración y control del tráfico, asegurándose de que las trazas transmitidas sean “buenas” fortaleciendo asó la seguridad de la red, mientras que se deja que a dispositivos IoT y equipos de red de capa 2 y 3 como switches y routers que sean los encargados de únicamente retrasmitir los datos, consumiendo así una menor cantidad de recursos y a su vez brindando una mayor seguridad en la red IoT. [9]

En la Figura 1 a continuación, se muestra gráficamente la arquitectura del modelo tradicional, mientras que en la Figura 2, se muestra la arquitectura de un modelo de una SDN.

Figura 2. Modelo tradicional de una red. [5]

Figura 3. Modelo de una red SDN. [5]

2. METODOLOGÍA

2.1. Comprensión del problema y procesamiento de los datos.

Para una mejor comprensión del problema y una orientación más clara al desarrollo del presente proyecto, se presenta una tabla de metodología de acuerdo a las etapas planteadas en el alcance, obteniendo como resultado la Tabla 1.

Tabla 1. Tabla de metodología. Fuente: Autor

OBJETIVOS	ETAPAS
OE1: Recopilar información acerca de las redes IoT, el tráfico, protocolos de transmisión, seguridad, vulnerabilidades y compatibilidad con dispositivos de bajo consumo de recursos.	**E1:** Comprensión del problema y preparación de datos.
OE2: Analizar los distintos modelos y algoritmos de clasificación de trazas y determinar la mejor opción para la posterior implementación de un controlador SDN capaz de etiquetar las trazas buenas o malas de acuerdo con su contenido.	**E2:** Definición de los criterios de evaluación. **E3:** Construcción y evaluación del modelo.
OE3: Implementar el sistema de detección de amenazas utilizando un modelo de Machine Learning desarrollado en lenguaje de programación Python.	**E3:** Construcción y evaluación del modelo.
OE4: Realizar las pruebas experimentales, constatando que se clasifiquen correctamente las tramas de acuerdo con el comportamiento definido en el controlador, empleando entornos de simulación y virtualización	**E4:** Análisis de errores. **E5:** Interpretación de los resultados.

de redes como Cisco Packet Tracer y otros.	

2.1.1. Selección de la base de datos.

Tras un período de análisis y búsqueda de un conjunto de datos que se acoplen a las necesidades del proyecto, se ha logrado dar con una base de datos denominada “DDOS attack SDN Dataset”, que fue creada como parte de un trabajo de investigación perteneciente a la Universidad de Bennet en la India, la cual contiene información relacionada al tráfico de una red capturado en el año 2020 y generado intencionalmente para la clasificación de tráfico empleando algoritmos de Machine Learning y Deep Learning con eventos de red maliciosos enfocados en ataques DDoS para un posterior análisis. [3]

2.1.2. Descripción de la base de datos.

El conjunto de datos fue generado a partir de una topología de red, donde cada uno de los switches se encuentran conectados a un solo controlador RYU SDN en el software de emulación Mininet. La simulación genera tráfico TCP, UDP e ICMP tanto benigno como malicioso, siendo este último, la colección de ataques de tipo TCP Syn, inundaciones de paquetes UDP (UDP Flood) y ataques ICMP, amoldando así un conjunto de datos que cuenta con 23 atributos, los cuales algunos son extraídos directamente de los switches de la topología, mientras que otros son calculados en base a las condiciones de la simulación, dentro de los atributos extraídos se incluyen campos como: switch-id, packet_count, duration_sec, ip de origen, ip de destino, números de puertos, etc., mientras que dentro del campo de los atributos calculados tenemos el campo “Packet per flow”, el cual, como su nombre indica es un contador de paquetes dentro de un solo flujo, así mismo el campo “Byte per flow” indica el conteo de bytes en un solo flujo, entre otras características más, las cuales para un mejor análisis se detallan a continuación en la tabla 2.

Tabla 2. Descripción de atributos de la base de datos. Fuente: Autor

Campo	Descripción
dt	Fecha y hora convertidos a número, empleado para monitoreo con un intervalo de 30 segundos.

switch	Switch de la red identificado por número.
src	Dirección de host de origen.
dst	Dirección de host de destino.
pktcount	Contador de paquetes.
bytecount	Contador de bytes.
dur	Duración de la trama en segundos.
dur_nsec	Duración de la trama en nanosegundos.
tot_dur	Suma de dur y dur_nsec.
flows	Número de flujos transmitidos.
packetins	Número de paquetes de entrada en el switch.
pktperflow	Número de paquetes por flujo.
byteperflow	Número de bytes por flujo.
pktrate	Cantidad de paquetes transmitidos por segundo.
Pairflow	Flujo par (0 si, 1 no).
Protocol	Protocolo de la trama.
port_no	Número de puerto.
tx_bytes	Número de bytes transmitidos.
rx_bytes	Número de bytes recibidos.
tx_kbps	Tasa de transferencia de bytes transmitidos.
rx_kbps	Tasa de transferencia de bytes recibidos.
tot_kbps	Suma de tx_kbps y rx_kbps.
label	Tipo de tráfico (0 malicioso, 1 benigno).

2.1.3. Arquitectura de la red de simulación.

Para la creación del dataset, se parte de la creación de una topología aleatoria en el software de simulación Mininet donde se va a generar tráfico mediante el envío de paquetes entre los diferentes hosts de la red, encontrando paquetes benignos TCP, UDP e ICMP y tráfico malicioso por medio de paquetes TCP Syn, UDP Flood y ataques ICMP. La simulación es ejecutada en un tiempo de 250 minutos, donde se han recolectado un total de 104.354 registros de datos, esta topología puede ser simulada por diferentes intervalos de tiempo para una mayor cantidad de datos recolectados.

Para poder capturar el flujo del tráfico generado en la topología, se requiere de un fichero desarrollado en el lenguaje de programación Python, en el cual se especifica cada uno de los campos y estadísticas del tráfico de la red que se desea extraer para el análisis y futuro desarrollo del modelo de clasificación de tráfico empleando Machine Learning. [3]

A continuación, se presenta en la Figura 4, una topología aleatoria donde se pueden recolectar datos y generar un dataset similar al utilizado.

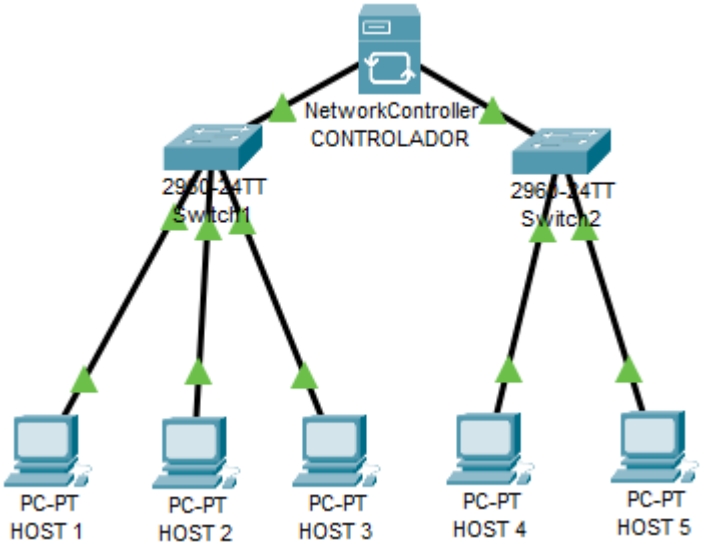


Figura 4. Arquitectura de red de datos. Fuente: Autor

2.1.4. Extracción de datos de simulación.

Como se mencionó previamente, el proceso de extracción de los datos está basado en un script de Python, el cual se comunica con la API del controlador SDN, en el caso del dataset “DDOS attack SDN Dataset” [3], los datos son extraídos de la herramienta de simulación Mininet, sin embargo, se seleccionó la herramienta Cisco Packet Tracer, donde, se crea una topología aleatoria con la finalidad de generar la mayor cantidad de tráfico, la topología empleada en Cisco Packet Tracer, se presenta en la Figura 5 a continuación.

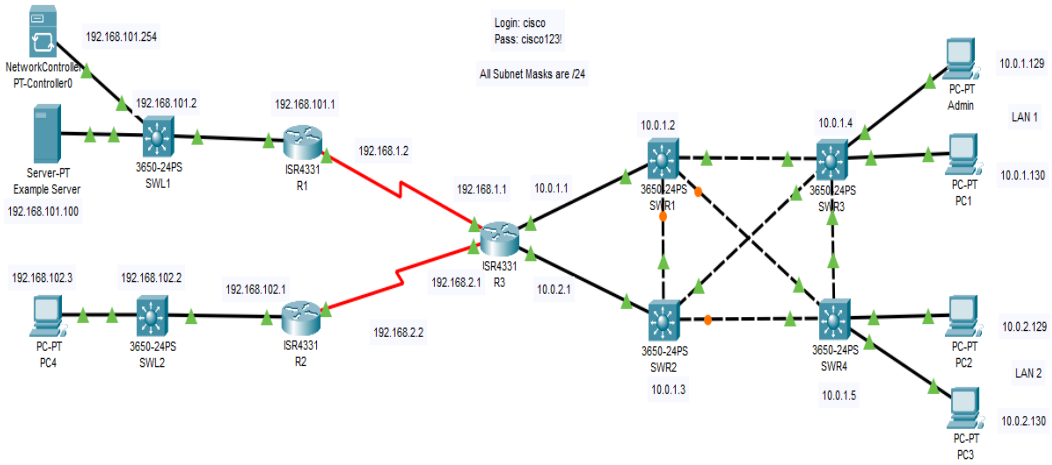


Figura 5. Topología de red simulada en Cisco Packet Tracer. Fuente: Autor

En la Figura 6, se puede observar la interfaz de direccionamiento que se crea cada vez que se abre la topología en Cisco Packet Tracer, donde el controlador muestra información de los dispositivos de la red como sus interfaces, las direcciones IP y sus nombres correspondientes.

Packet Tracer - Implement REST APIs with an SDN Controller

Addressing Table

Note: All subnet masks are /24 (255.255.255.0).

Device	Interface	IP Address
R1	G0/0/0	192.168.101.1/24
	S0/1/0	192.168.1.2
R2	G0/0/0	192.168.102.1
	S0/1/1	192.168.2.2
R3	G0/0/0	10.0.1.1
	G0/0/1	10.0.2.1
	S0/1/0	192.168.1.1
	S0/1/1	192.168.2.1
SWL1	VLAN 1	192.168.101.2
SWL2	VLAN 1	192.168.102.2

Time Elapsed: 37:19:49

Top Dock Check Results Back 1/1 Next

Figura 6. Interfaz de direccionamiento del controlador. Fuente: Autor

Una vez implementada la topología y estableciendo la conectividad de esta se puede acceder al controlador SDN por medio del navegador web de cualquiera de los hosts, haciendo uso de la dirección IP del controlador, donde se presentará una interfaz, donde nos solicitará usuario y contraseña como se muestra a continuación en la Figura 7.

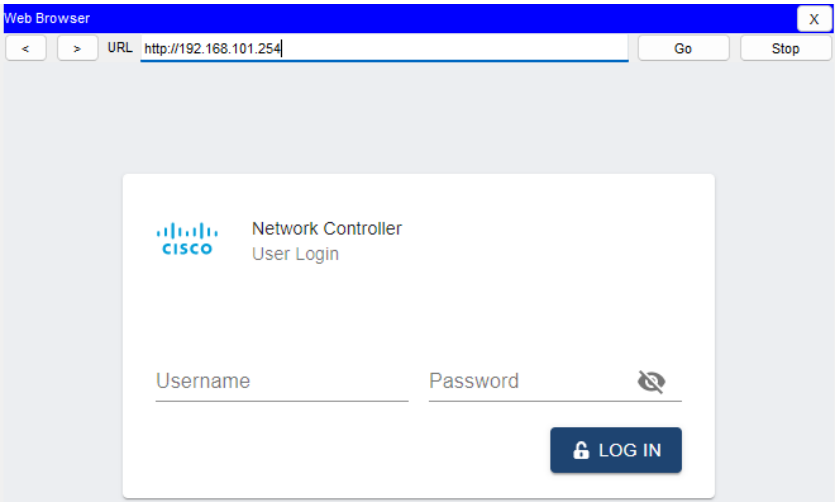
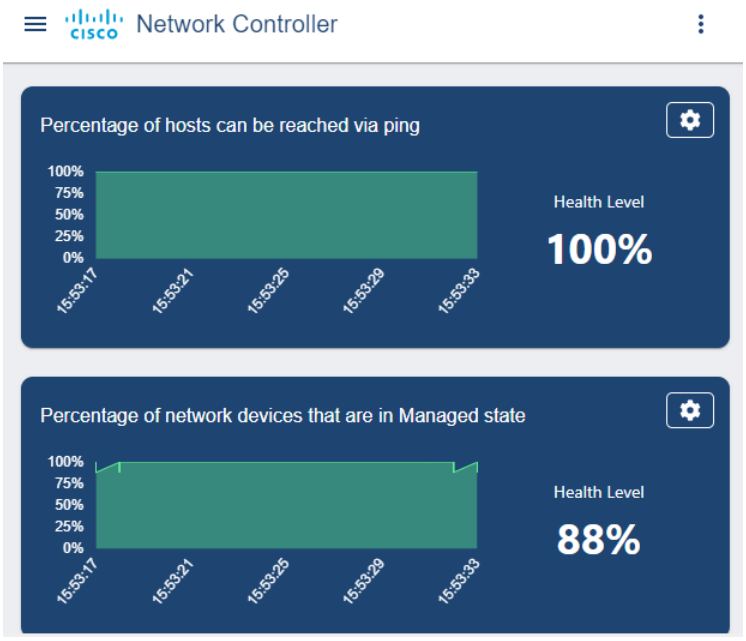


Figura 7. Interfaz de registro del controlador. Fuente: Autor

En el controlador, se puede ver varias características de la red sobre la que este actúa, tales como: porcentaje de los hosts que interactúan mediante ping, porcentaje de dispositivos que son administrados por el controlador, así como los diferentes estados de los hosts, dispositivos de red y QoS, y el tiempo en el que se efectúa la simulación, como se observa en la Figura 8.



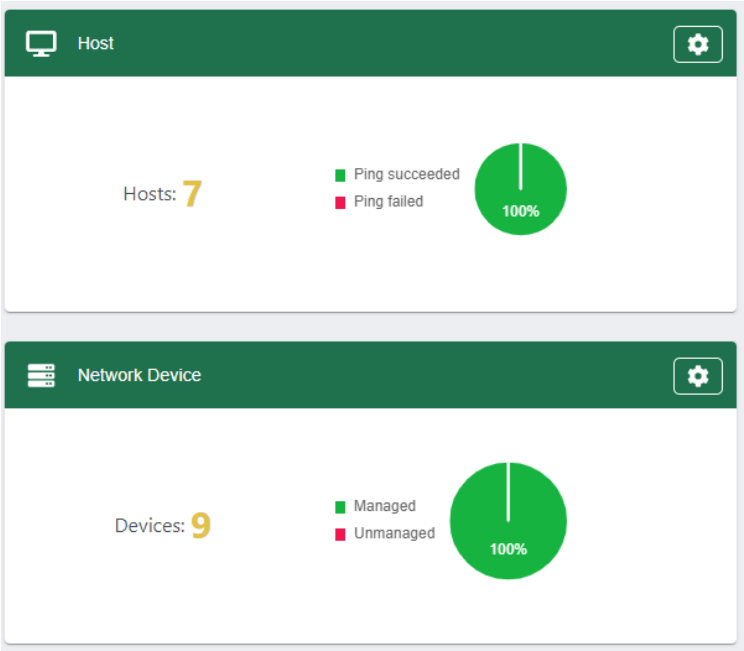


Figura 8. Estadísticas de la red proporcionadas por el controlador. Fuente: Autor

Los parámetros estadísticas, si bien provee información de la red, los datos indicados en la Figura 8, son aquellos que un controlador SDN, en un entorno de simulación puede proveer, no obstante, Cisco nos brinda la oportunidad de hacer uso de una API, donde, empleando el lenguaje de programación Python se puede mediante una comunicación por tickets interactuar con los parámetros de la red, a través de las funciones que proporciona “Cisco Packet Tracer Northbound API for Network Controller Device”, siendo algunas de estas presentadas en la Figura 9.

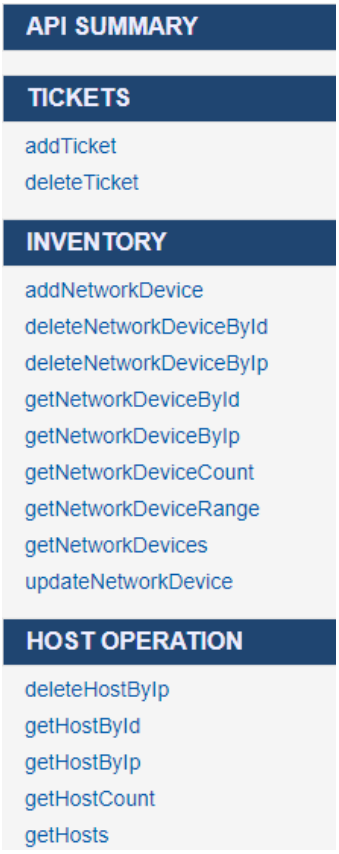


Figura 9. Algunas de las funciones de la API de Cisco. Fuente: Autor

Para extraer los datos necesarios y armar un dataset con las características de DDOS attack SDN Dataset [3], se hacen uso de varias de las funciones que nos provee la API de Cisco, las cuales se han organizado de manera distribuida empleando dos scripts desarrollados en Python, que son detallados individualmente a continuación:

2.1.4.1. Obtención de ticket:

Como bien se mencionó previamente, la API permite a los usuarios acceder a los diferentes controladores de la red a través de una interfaz REST mediante HTTP, siendo así posible acceder directamente a sus características mediante el siguiente script de la Figura 10:

```python
import json
import requests
api_url = "http://localhost:58000/api/v1/ticket"

headers = {
    "content-type" : "application/json"
}

body_json = {
    "username"  : "cisco",
    "password" : "cisco123!"
}
resp = requests.post(api_url,json.dumps(body_json),headers=headers,verify=False)
print("Ticket request status" , resp.status_code)
response_json = resp.json()

serviceTicket = response_json["response"] ["serviceTicket"]
print("El número de ticket de servicio es:" , serviceTicket)
```

Figura 10. Código de obtención del ticket. Fuente: Autor

El código ejecutado nos dará un resultado similar al mostrado en la Figura 11, este es el que debe ser actualizado en el campo “X-Auth-Token” para que pueda comunicarse el script de Python con la API.

```
 c:/Users/juanj/OneDrive/Escritorio/SDN_V2/SDN/01_get-ticket.py
Ticket request status 201
El numero de ticket de servicio es: NC-101-4b4ee4411d1346aa9f80-nbi
```

Figura 11. Generación del número de ticket. Fuente: Autor

2.1.4.2. Extracción de campos:

Con el número de ticket generado ya se tiene paso para la comunicación con el controlador y la extracción de campos para el dataset, todo ello a través de otro script de Python, donde se emplea las funciones que proporciona la API para extraer los campos necesarios para conformar el conjunto de datos en el

formato deseado. En la Figura 12 se presenta a continuación, el script de Python, donde se hace uso de varias de las funciones de la API, que nos van a permitir extraer distintas características de las tramas que circulan por la red.

```
1  api_url = "http://localhost:58000/api/v1/host"
2  headers = {"X-Auth-Token": "NC-101-4b4ee4411d1346aa9f80-nbi"}
3  resp = requests.get(api_url,headers=headers,verify=False)
4  print("Request status: " , resp.status_code)
5  response_json = resp.json()
6  hosts = response_json["response"]
7  for  host in hosts:
8      print(host["hostName"] , "\t",host["hostIp"] , "\t" , host["hostMac"], "\t" , host["connectedInterfaceName"])
```

Figura 12. Segmento de código de extracción de campos. Fuente: Autor

Haciendo uso de estas funciones se puede ir conformando el conjunto de datos, donde se extraen las características de la red, como se muestra en la línea 8 del código de la Figura 12, donde se especifican campos como: nombre de host, la dirección IP de los hosts, la dirección MAC, así como el nombre de las interfaces conectadas, arrojando un resultado que se indica en la Figura 13:

```
PC4      192.168.102.3   00E0.F96C.155B          GigabitEthernet1/0/24
PC3      10.0.2.130      0004.9A42.C245          GigabitEthernet1/0/24
Admin    10.0.1.129      0050.0FCE.B095          GigabitEthernet1/0/21
PC2      10.0.2.129      0060.47C1.A4DB          GigabitEthernet1/0/23
PC1      10.0.1.130      00E0.A330.3359          GigabitEthernet1/0/22
Example Server   192.168.101.100         000A.413D.D793          GigabitEthernet1/0/3
PC3      10.0.2.129      0004.9A42.C245          GigabitEthernet1/0/24
```

Figura 13. Resultado del código de extracción de la Figura 12.

De este modo se pueden extraer varios datos y características de la red con la ayuda de la API de Cisco y almacenarlos en un dataset para un posterior análisis.

2.1.5. Generación de tráfico malicioso:

Si bien se obtienen los datos del tráfico de la red por medio de las funciones de la API de Cisco, no se han generado iniclamente tramas maliciosas dentro de la

topología, para conseguir esto, se hará uso del apartado de programación que ofrecen cada uno de los hosts de la red de Cisco Packet Tracer, donde permite dentro del host embeber código en lenguaje Python para generar tráfico del tipo malicioso a tiempo real en la red. Para ello, es necesario acceder dentro del host a la pestaña de programación donde se crea un nuevo script para la generación del código malicioso. Para este propósito se emplea un código en lenguaje Python de ataque DDoS, el cual se encuentra basado en la denegación de servicio utilizando múltiples hilos.

```
1  def conectar():
2      global url
3      global activo
4      global estado
5
6      while activo:
7        if activo == 0:
8          sys.exit()
9          raise
10         break
11       else:
12         try:
13           requests.get(url,timeout=8,headers=headers)
14           estado = '+'
15         except KeyboardInterrupt:
16           activo = 0
17           sys.exit()
18         except:
19           estado = '-'
```

Figura 14. Parte1. Establecimiento de conexión con el objetivo. Fuente: Autor

Como parte del código, se incluye una función denominada “conectar” que será la encargada de realizar una solicitud a una URL objetivo a través del modo “request” como se observa en la línea 8, de modo que, si la solicitud es exitosa, la variable estado tiene un valor de “+”, caso contrario queda como “-”. Posteriormente, se crea una función denominada “principal”, la cual es la encargada de crear una lista de hilos y llama a la función “conectar” para cada hilo.

```
1  principal()
2    while activo:
```

```
3       seg = seg + 1
4       try:
5         if (estado == '+'):
6           print(" [Status] Online - " + str(seg) + " second/s"),
7         else:
8           print(" [Status] Offline - " + str(seg) + " second/s"),
9         time.sleep(1)
10        sys.stdout.flush()
11        print ("\r"),
12      except KeyboardInterrupt:
13        activo = 0
14        print ('\n\n[Closing threads]')
15        sys.exit()
16      except:
17        seg = seg + 1
```

Figura 15. Parte 2. Bucle e iteración de solicitudes. Fuente: Autor

En la Figura 15, se configura el bucle de iteración de solicitudes donde, dependiendo del valor del estado, indicará si el servicio aún sigue en línea o ya está fuera de línea, así mismo establece la posibilidad de cortar la iteración empleando el teclado del ordenador.

```
PS C:\Users\juanj\OneDrive\Escritorio\SDN_V2\SDN> python ddos.py "192.168.101.254" 20
[Creating threads]

 [Status] Online - 1 second/s

 [Status] Online - 2 second/s

 [Status] Online - 3 second/s

 [Status] Online - 4 second/s

 [Status] Online - 5 second/s
```

Figura 16. Resultado del ataque DDoS al controlador de la red SDN. Fuente: Autor

2.2. Definición de criterios de evaluación.

2.2.1. Algoritmos de Machine Learning a utilizar.

Para la etapa de entrenamiento se ha considerado el uso de una red neuronal convolucional o también conocida como CNN, las cuales pertenecen a un subconjunto o tipo de Machine Learning, compuesto por capas; una capa de entrada, una o múltiples capas intermedias ocultas y una capa de salida, donde cada uno de los nodos se encuentran interconectados mediante un peso y un umbral asociado, de modo que, cuando la salida de un nodo específico se

encuentre por encima del peso de umbral determinado, éste se activará, enviando así datos a los nodos restantes de la red. Para una mejor comprensión, se detalla en la Figura 17 la arquitectura de una red neuronal convolucional.

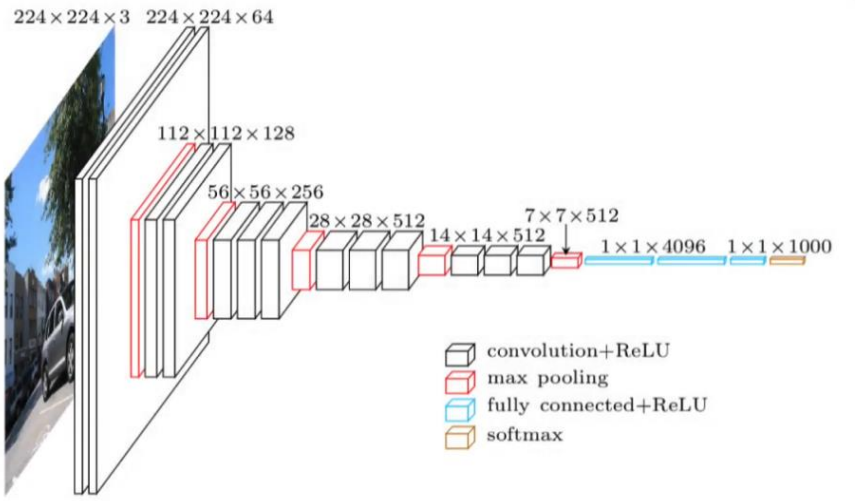


Figura 17. Arquitectura de una red neuronal convolucional. Fuente: [10]

Con base en la arquitectura de una CNN, se debe asociar al plano de trabajo sobre el cual se aplica un modelo basado en este algoritmo, y como se nota, se trata de un modelo orientado al trabajo con imágenes, mas no con datos numéricos ni categóricos, por lo que se van a desarrollar tres modelos de aprendizaje supervisado basados en redes convolucionales, estos son: regresión lineal, support vector machine (SVM) y árbol de decisión, los cuales se desglosan brevemente a continuación:

2.2.1.1. Regresión Logística:

La regresión logística, es empleada para modelar la probabilidad de que un evento pueda o no ocurrir, es decir que éste debe ser binario o constar de dos categorías, como es el caso del campo “label” de nuestro dataset, que nos arroja valores entre 0 y 1 para definir si la trama es maliciosa o benigna, siendo la regresión logística una de las técnicas que más se adecúa para la estimación de las probabilidades de ocurrencia de uno de estos dos resultados en particular.

2.2.1.2. Support Vector Machine (SVM):

Por otro lado, el algoritmo Support Vector Machine o mejor conocido como SVM, se desenvuelve en un conjunto de datos donde una instancia puede pertenecer a una de dos clases posibles, para ello hace uso de un hiperplano dentro de un espacio dimensional, el cual es seleccionado a través de la maximización del margen de las dos clases posibles, en nuestro caso serían maliciosas y benignas, este margen será la distancia perpendicular más corta entre el hiperplano y los puntos más cercanos que tendrá cada clase.

Otro concepto que se maneja como su nombre lo indica son los vectores de soporte, los cuales son los puntos o muestras de datos más cercanos al hiperplano. Con el hiperplano óptimo seleccionado el algoritmo SVM hace uso de éste para clasificar las instancias del conjunto de datos, en para este caso, determinar si las tramas son benignas o maliciosas dependiendo del lado del hiperplano en la que estas se encuentren.

2.2.1.3. Random Forest:

Los árboles de decisión son algoritmos de aprendizaje automático que son usados generalmente para problemas de clasificación, su nombre se basa en la estructura de árbol, donde cada una de las ramificaciones representa una decisión en base a un atributo específico, para el caso del conjunto de datos, se basaría en la columna “label”, representando así cada una de las hojas del árbol como una clase o un valor numérico, bien sea 0 y 1, lo cual se traduce en tramas maliciosas y benignas.

Si bien, los árboles de decisión pueden llegar a ser fáciles de analizar e interpretar en cuestión de tramas buenas y malas, tienen una gran probabilidad de sobreajustar los datos de las tramas si no son controlados de forma adecuada, por tal motivo existe la probabilidad de trabajar con varios árboles de decisión de manera conjunta, conociendo a este método como Random Forest, el cual es utilizado esencialmente para problemas de clasificación por medio de la combinación de múltiples árboles de decisión, garantizando así una mejora en la precisión del algoritmo y una considerable reducción del sobreajuste de los árboles de decisión, proporcionando así una mayor robustez del modelo de aprendizaje, motivo por el cual se ha optado por trabajar con

este algoritmo, ya que para la cantidad de datos con la que se va a trabajar se necesita un equilibrio entre precisión y generalización en sus predicciones.

2.3. Construcción del modelo basado en los algoritmos de clasificación propuestos.

2.3.1. Herramientas para la construcción del modelo de clasificación.

Google Colab.

Como se mencionó anteriormente, para la etapa del procesamiento, tratamiento de los datos y elaboración de los modelos de aprendizaje automático, se empleó el Lenguaje de programación Python, con la ayuda de la extensión pandas y demás librerías de Machine Learning dentro del entorno de Google Colab, donde se realizaron tareas de análisis, etiquetado, normalización e interpretación gráfica de los registros obteniendo, así como resultado una serie de gráficos y estadísticas que nos ayudará para una mejor percepción de la estructura del conjunto de datos con el que se va a trabajar, todo esto con la finalidad de facilitar la visualización y entendimiento de los mismos para el futuro desarrollo de los diferentes modelos de Machine Learning propuestos, los mismos que son desarrollados en el mismo entorno, ya que Google Colab soporta todas las librerías necesarias para el diseño y prueba de estos modelos.

Cisco Packet Tracer.

Para la etapa de la implementación de la red, se va a emplear el simulador de Cisco Packet Tracer, el cual se caracteriza por ser una poderosa herramienta de simulación de redes que permite analizar y experimentar el comportamiento de las redes, además soporta la implementación y configuración de controladores SDN [10], lo cual es un requisito fundamental para el desarrollo del presente proyecto.

Como bien se conoce, Cisco Packet Tracer es una herramienta que exige de una cuenta de Cisco Academy para poder acceder a sus funcionalidades, por lo que se accedió a la plataforma mediante las credenciales proporcionadas por la EPN. La interfaz del programa es simple e intuitiva en donde se dispone de todos los elementos de red necesarios para armar la topología deseada, la misma que se muestra en la Figura 18 a continuación:

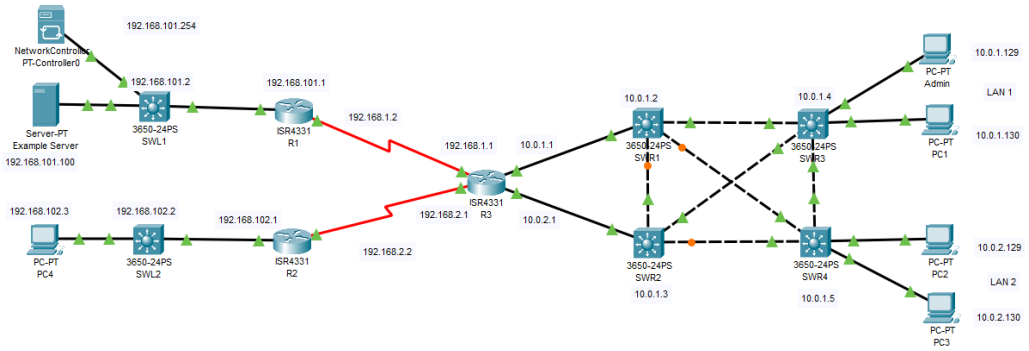


Figura 18. Topología de una red SDN en Cisco Packet Tracer. Fuente: Autor

Visual Studio Code:

Se trata de una poderosa herramienta de edición de código fuente desarrollada por Microsoft cuya principal ventaja es ser software libre y multiplataforma que cuenta con el soporte necesario para la depuración de código en prácticamente todos los lenguajes de programación existentes gracias a su amplio número de extensiones y su buena integración con Git.

Para este proyecto se ha optado por el uso de esta herramienta para las diferentes pruebas de código de extracción de datos y conexión con Cisco Packet Tracer, si bien, herramientas como Google Colab nos permitía trabajar o realizar tareas similares, se consideró que era una mejor alternativa optar por trabajar en un entorno aislado del análisis de datos y qué mejor herramienta que VS Code que con su gran número de extensiones, permite trabajar en la conexión con la API de Cisco Packet Tracer así como con las diferentes pruebas realizadas en la generación de tráfico malicioso DDoS. [11]

2.3.2. Librerías consideradas.

Durante el desarrollo del presente trabajo se hicieron uso de múltiples librerías para diferentes propósitos, dentro de las áreas de tratamiento de datos, representación de gráficas, Machine Learning, entre otros, las cuales para un mejor entendimiento se detallan en la Tabla 3 a continuación:

Tabla 3. Librerías utilizadas. Fuente: Autor

Librería	Descripción
pandas	Creación de datasets y tratamiento de datos.
numpy	Creación de vectores y matrices multidimensionales.
matplotlib	Generación de gráficos estadísticos.
seaborn	Visualización de datos.
sklearn	Aprendizaje automático gratuito para Python.

2.3.3. Visualización de los datos.

Previo al desarrollo de los modelos de aprendizaje es necesario conocer bien el conjunto de datos con el que se va a trabajar, si bien tenemos una visualización completa de los datos, es necesario trabajar con las herramientas y funciones que nos ofrecen las librerías de la Tabla 3 para una mejor concepción de los datos, entre las funciones más relevantes empleadas para este fin se utilizaron:

2.3.3.1. Análisis de datos:

Data.head(): Para poder tener una visión más amplia de la estructura del conjunto de datos, se tiene la opción de usar varias funciones que viene incluidas en la librería de Pandas, como lo es head (), cuya función es mostrar las primeras filas del dataset, de modo que se puede tener una idea de los campos y del tipo de dato que superficialmente contiene el conjunto de datos. En la Figura 19 a continuación, se muestra la estructura inicial de los datos dentro de la librería Pandas.

	dt	switch	src	dst	pktcount	bytecount	dur	dur_nsec	tot_dur	flows	...	pktrate
0	11425	1	10.0.0.1	10.0.0.8	45304	48294064	100	716000000	1.010000e+11	3	...	451
1	11605	1	10.0.0.1	10.0.0.8	126395	134737070	280	734000000	2.810000e+11	2	...	451
2	11425	1	10.0.0.2	10.0.0.8	90333	96294978	200	744000000	2.010000e+11	3	...	451
3	11425	1	10.0.0.2	10.0.0.8	90333	96294978	200	744000000	2.010000e+11	3	...	451
4	11425	1	10.0.0.2	10.0.0.8	90333	96294978	200	744000000	2.010000e+11	3	...	451

Figura 19. Estructura inicial del conjunto de datos. Fuente: Autor

Data.shape(): Si lo que se requiere es conocer información aún más detallada, tales como las dimensiones del conjunto de datos, rango de los registros y tipos de datos por cada campo, lo adecuado sería emplear las funciones info () y shape (), que nos indicarán la dimensión del dataset y un resumen respectivamente. Todos los datos mostrados como resultado de estas funciones se pueden ver en la Figura 20.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 104345 entries, 0 to 104344
Data columns (total 23 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   dt           104345 non-null  int64
 1   switch       104345 non-null  int64
 2   src          104345 non-null  object
 3   dst          104345 non-null  object
 4   pktcount     104345 non-null  int64
 5   bytecount    104345 non-null  int64
 6   dur          104345 non-null  int64
 7   dur_nsec     104345 non-null  int64
 8   tot_dur      104345 non-null  float64
 9   flows        104345 non-null  int64
 10  packetins    104345 non-null  int64
 11  pktperflow   104345 non-null  int64
 12  byteperflow  104345 non-null  int64
 13  pktrate      104345 non-null  int64
 14  Pairflow     104345 non-null  int64
 15  Protocol     104345 non-null  object
 16  port_no      104345 non-null  int64
 17  tx_bytes     104345 non-null  int64
 18  rx_bytes     104345 non-null  int64
 19  tx_kbps      104345 non-null  int64
 20  rx_kbps      103839 non-null  float64
 21  tot_kbps     103839 non-null  float64
 22  label        104345 non-null  int64
dtypes: float64(3), int64(17), object(3)
memory usage: 18.3+ MB
```

Figura 20. Información resumida del conjunto de datos.

Data.describe(): Si se requiere ahondar aún más en las características del conjunto de datos, se puede emplear la función describe (), la cual muestra un resumen del dataset con varios parámetros estadísticos por cada campo, como: conteo de registros, media, desviación estándar, mínimos, máximos, entre otros. Al aplicar esta función en el dataset, se obtuvo el resultado mostrado en la Figura 21.

	dt	switch	pktcount	bytecount	dur	dur_nsec	tot_dur
count	104345.000000	104345.000000	104345.000000	1.043450e+05	104345.000000	1.043450e+05	1.043450e+05
mean	17927.514169	4.214260	52860.954746	3.818660e+07	321.497398	4.613880e+08	3.218865e+11
std	11977.642655	1.956327	52023.241460	4.877748e+07	283.518232	2.770019e+08	2.834029e+11
min	2488.000000	1.000000	0.000000	0.000000e+00	0.000000	0.000000e+00	0.000000e+00
25%	7098.000000	3.000000	808.000000	7.957600e+04	127.000000	2.340000e+08	1.270000e+11
50%	11905.000000	4.000000	42828.000000	6.471930e+06	251.000000	4.180000e+08	2.520000e+11
75%	29952.000000	5.000000	94796.000000	7.620354e+07	412.000000	7.030000e+08	4.130000e+11
max	42935.000000	10.000000	260006.000000	1.471280e+08	1881.000000	9.990000e+08	1.880000e+12

Figura 21. Información estadística del conjunto de datos. Fuente: Autor

Estas y otras funciones más son proporcionadas por la librería de pandas y son empleadas de acuerdo con el nivel de profundidad de análisis que se requiera para los diferentes fines.

2.3.3.2. Visualización de datos:

Porcentaje de solicitudes benignas y maliciosas del dataset:

Si bien las funciones de análisis de los datos son muy empleadas, podemos tener adicionalmente otra perspectiva de los mismos para otro enfoque de análisis por medio de la visualización de los datos, para este caso, la primera representación que se requiere es la del índice porcentual de tramas maliciosas y tramas benignas, con la finalidad de tener una noción de en qué relación se encuentran estos dos tipos de tramas en base a los cuales girará el desarrollo de los modelos de clasificación y predicción. Para ello, se emplea la librería “plot” y haciendo uso del campo “label” se puede clasificar las tramas y ejemplificarlas en una gráfica de pastel como la que se muestra en la Figura 22 a continuación:

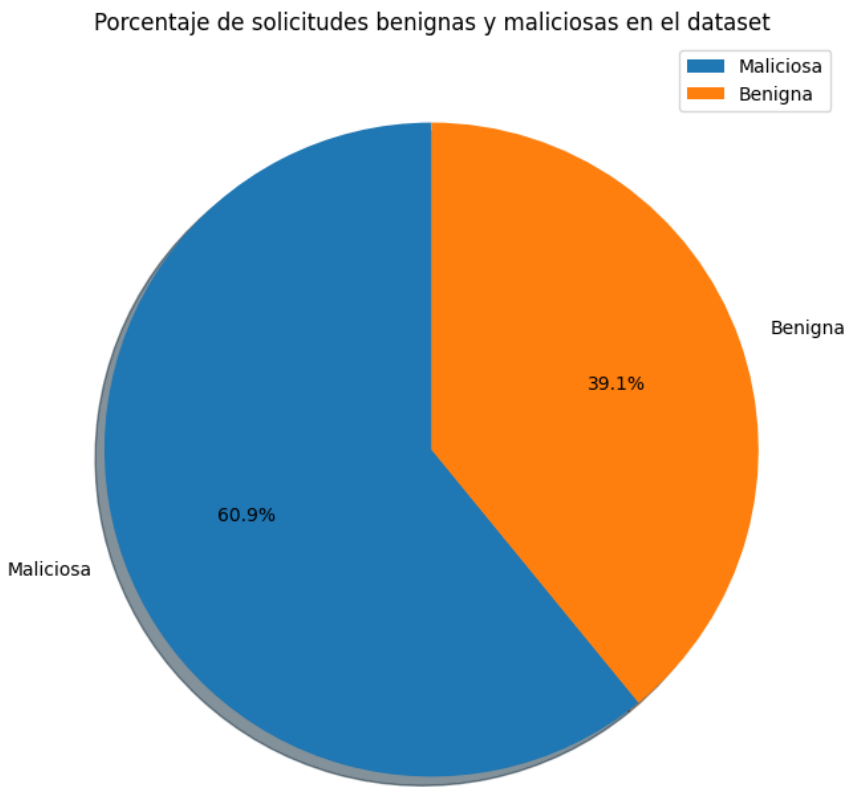


Figura 22. Gráfica porcentual de solicitudes maliciosas y benignas en el dataset. Fuente: Autor

Número de solicitudes de las diferentes direcciones IP:

Es necesario también hacer un análisis de las diferentes solicitudes de la res hacia las diferentes direcciones IP de los hosts de la red, donde haciendo uso de la librería “plot”, en la Figura 23, se logra por medio de un gráfico de barras mostrar el número total de solicitudes en color verde y para contrastar visualmente se ha logrado extraer por medio de otras barras de color azules superpuestas el número total de solicitudes maliciosas, de modo que, se pueda contemplar en qué relación se han transmitido las tramas maliciosas con las tramas totales, teniendo así una apreciación visual que puede ser de gran ayuda para el o los criterios de desarrollo de los modelos de Machine Learning.

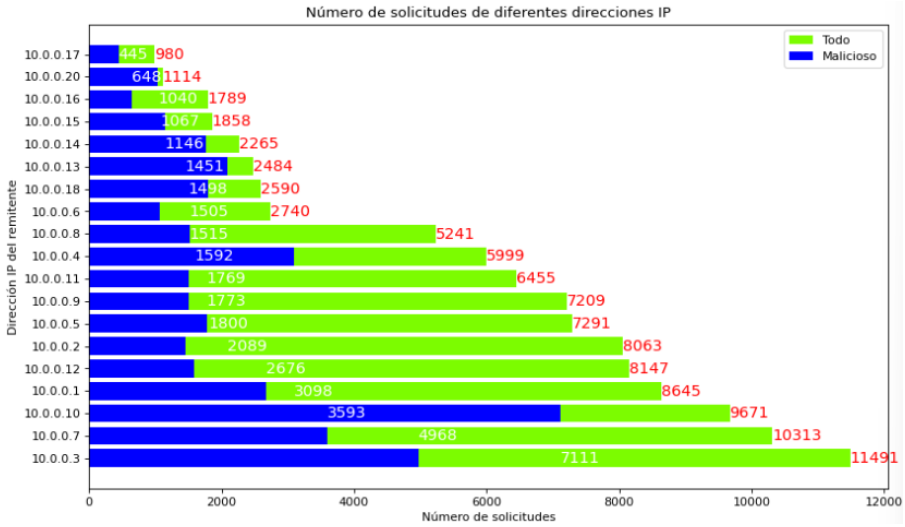


Figura 23. Número de solicitudes totales vs maliciosas. Fuente: Autor

Número de solicitudes de diferentes protocolos:

Como se observó previamente en la sección de análisis de datos, el tráfico capturado de la red cuenta con tres protocolos: UDP, TCP e ICMP, información que, si bien se pudo visualizar en el resumen del dataset, se va a constatar a través de una gráfica de barras, las misma que al igual que la Figura 23, se va a superponer las solicitudes totales divididas por protocolo a las del tipo maliciosas, como se muestra en la Figura 24 a continuación:

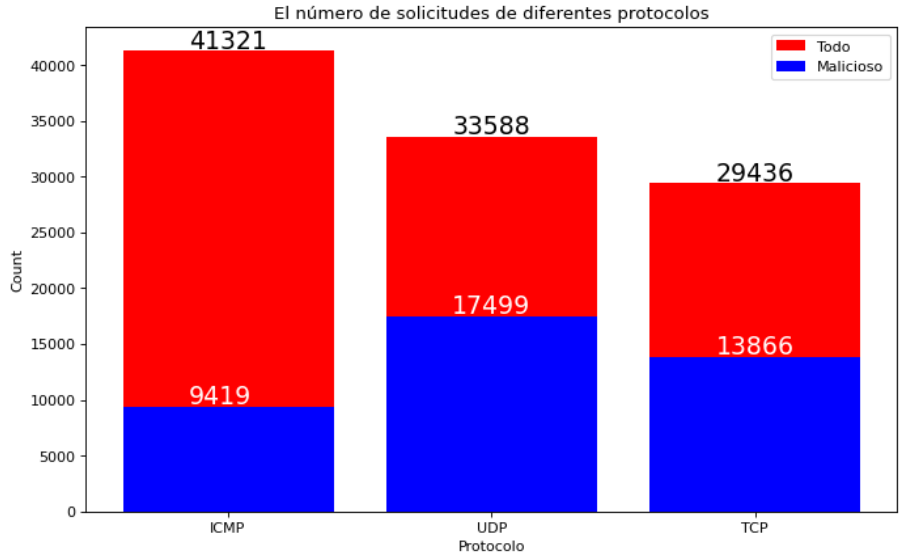


Figura 24. Número total de solicitudes vs maliciosas por protocolo. Fuente: Autor

Estas y otras gráficas más se pueden exponer con el objetivo de tener cada vez una mejor perspectiva de los datos con lo que se está trabajando, las cuales dependiendo la orientación del proyecto podrían proporcionarnos diferente información de acuerdo a otras características como: el tiempo de duración de los paquetes, los bytes transmitidos y el tiempo de duración de las tramas por switch que se muestran en las Figura s 25, 26 y 27 respectivamente gracias a las librerías proporcionadas por Pandas. Dejando así listo el campo para la definición y creación de los modelos de Machine Learning.

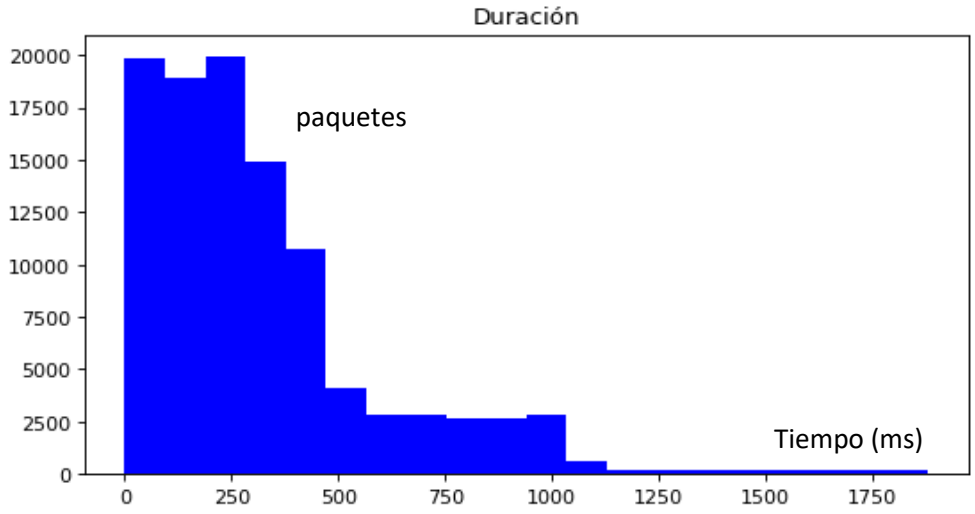


Figura 25. Tiempo de duración por número de paquetes. Fuente: Autor

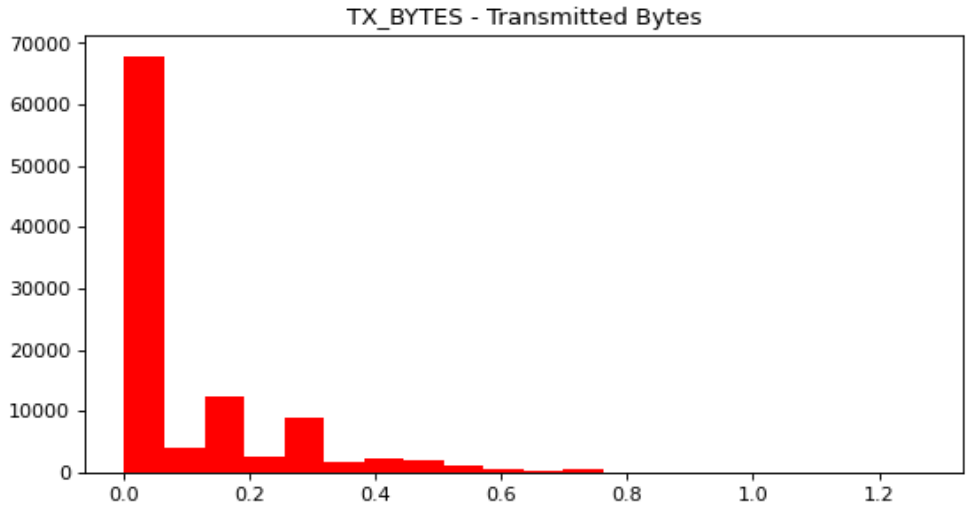


Figura 26. Bytes transmitidos por número de paquetes. Fuente: Autor

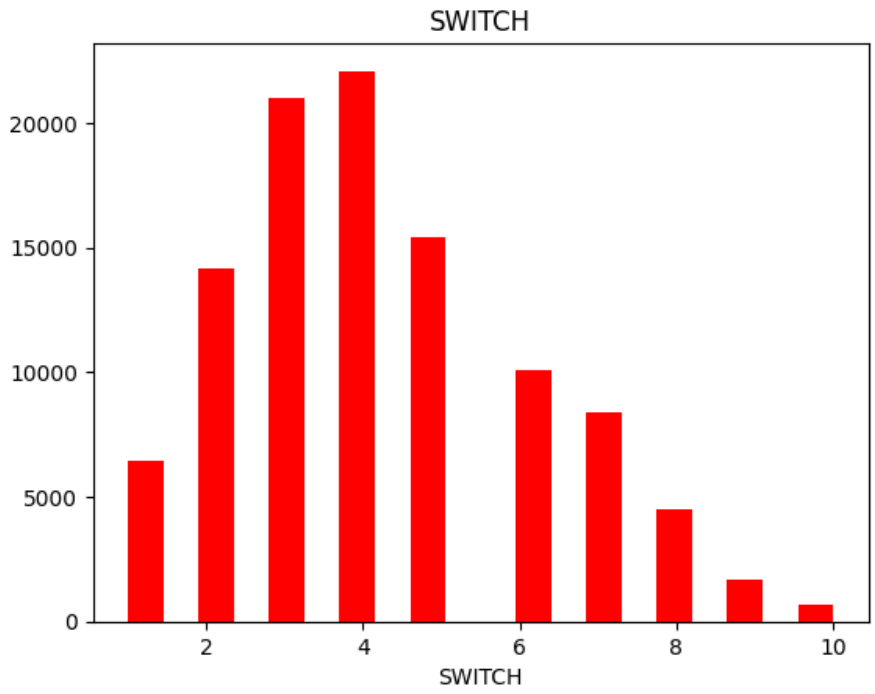


Figura 27. Duración en milisegundos de cada ping en cada uno de los switches de la red. Fuente: Autor

2.3.4. Definición de los modelos.

Todos los modelos son desarrollados en el entorno de Google Colab, donde cada uno de ellos va a ser definido a modo de función, con la finalidad de que, al momento de ejecutarlos, cada uno pueda ser llamado de manera independiente sin entorpecer ni involucrar variables en el desarrollo de los otros modelos.

2.3.4.1. Regresión logística:

Para su desarrollo vamos a enfocarnos en la Figura 28, donde a partir de una función denominada “LogisticRegression”, dentro de la cual se crea una lista denominada “solvers”, que hace alusión a los distintos tipos de solucionadores para trabajar con la regresión logística, de modo que permitan encontrar eficientemente los coeficientes óptimos para el modelo. Dentro de los solucionadores se ha incluido “newton-cg, lbfgs, liblinear, sag y saga” como se observa en la línea 3 de la Figura 28. Posteriormente se realizan las preparaciones correspondientes para la generación de los conjuntos de entrenamiento y prueba y se ejecuta el modelo.

```
1  def LogisticRegression(self):
2      #arreglo de solucionadores de tipo newton-cg, lbgfs, sg, saga, etc.
3      solvers = ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']
4
5      start_time = time.time()
6      results_lr = []
7      accuracy_list = []
8      for solver in solvers:
9          LR = LogisticRegression(C=0.03, solver=solver).fit(self.X_train,
   self.y_train)
10         predicted_lr = LR.predict(self.X_test)
11         accuracy_lr = accuracy_score(self.y_test, predicted_lr)
12
13         results_lr.append({'solver' : solver, 'accuracy':
   str(round(accuracy_lr * 100, 2)) + "%",
14                            'Coefficients': {'W' : LR.coef_, 'b': LR.intercept_}})
15
16         accuracy_list.append(accuracy_lr)
17
18     solver_name = solvers[accuracy_list.index(max(accuracy_list))]
19     LR = LogisticRegression(C=0.03,
   solver=solver_name).fit(self.X_train,self.y_train)
20     predicted_lr = LR.predict(self.X_test)
21     accuracy_lr = accuracy_score(self.y_test, predicted_lr)
```

Figura 28. Segmento de código del modelo de regresión logística. Fuente: Autor

Este proceso se repetirá para cada uno de los solucionadores y sus resultados serán almacenados en “results_lr” con la finalidad de realizar una comparación y encontrar el resultado que presente la mayor precisión, siendo este el que se mostrará en los resultados finales.

2.3.4.2. Support Vector Machine:

Centrándonos en la Figura 29, podemos contemplar que SVM es planteado como una función, la cual es denominada como: “SupportVectorMachine”, la cual cuenta con el argumento “self”, que hace referencia a los conjuntos de entrenamiento previamente planteados, de la línea 2 a la 4, se define el temporizador “start_time” y otros parámetros como “accuracy_list” y “result_list” que cumplen los mismos fines que en el código de regresión lineal, sin embargo en la línea 5, se crea una lista de “kernels”, que cumplen una función similar a la de los “solvers”, con la diferencia de que estos son funciones matemáticas que miden la similitud entre varios puntos determinados en el plano. Desde la

línea 7 hasta la 14, se define un bucle for para recorrer la lista de kernels, para cada uno de ellos se va a crear un modelo SVM empleando la función “svm.SVC” como se muestra en la línea 8, posteriormente se ajusta el modelo al conjunto de entrenamiento para después realizar las predicciones con los datos de prueba y calcular la precisión del modelo, datos que así mismo van a ser almacenados en “result_svm” para la respectiva generación del reporte de clasificación final.

```
1  def SupportVectorMachine(self):
2          start_time = time.time()
3          accuracy_list = []
4          result_svm = []
5          kernels = ['linear', 'poly','rbf', 'sigmoid']
6
7          for kernel in kernels:
8              SVM = svm.SVC(kernel=kernel).fit(self.X_train, self.y_train)
9              predicted_svm = SVM.predict(self.X_test)
10             accuracy_svm = accuracy_score(self.y_test, predicted_svm)
11             result_svm.append({"kernel" : kernel, "accuracy":
f"{round(accuracy_svm*100,2)}%"})
12             print("Accuracy: %.2f%%" % round((accuracy_svm * 100.0),2))
13 print('#########################################################')
14             accuracy_list.append(accuracy_svm)
15
16         kernel_name = kernels[accuracy_list.index(max(accuracy_list))]
17         SVM = svm.SVC(kernel=kernel_name).fit(self.X_train, self.y_train)
18         predicted_svm = SVM.predict(self.X_test)
19         accuracy_svm = accuracy_score(self.y_test, predicted_svm)
```

Figura 29. Segmento de código del modelo de Support Vector Machine (SVM). Fuente: Autor

2.3.4.3. Random Forest:

Como se mencionó en el apartado teórico, el modelo de Random Forest propone trabajar con varios árboles de decisión, por lo que se empieza con la implementación de un método de árbol de decisión, así mismo será definido como una función como se muestra en la primera línea de la Figura 30, donde igualmente se le pasará como argumento “self” referenciando a los conjuntos

de datos de prueba y entrenamiento. En las líneas 2 y 3 se registra una variable temporal para monitorear la duración del modelo, seguidamente, se crea una instancia denominada “DecisionTreeClassifier” para representar el modelo de árbol de decisión acompañado de otra instancia “GridSearchCV” que será la encargada de encontrar mediante validación cruzada los mejores parámetros para el desarrollo del modelo, basado en el conjunto de datos con el que se está trabajando. Una vez que se encuentran los mejores parámetros de entre “criterion”, “max_deph” y “max_leaf_nodes”, se entrena un nuevo árbol de decisión empleando los parámetros ya optimizados previamente, como se puede observar en las líneas 11 a la 17.

```
1  def DecisionTree(self):
2          start_time = time.time()
3          tree = DecisionTreeClassifier()
4          dt_search = GridSearchCV(tree, param_grid={'criterion' : ['gini', 'entropy'],
5                                      'max_depth' : [2,3,4,5,6,7,8, 9, 10],
6                                      'max_leaf_nodes' : [2,3,4,5,6,7,8,9,10, 11]},
7                           n_jobs=-1, cv=5, scoring='accuracy', verbose=2)
8
9          dt_search.fit(self.X_train, self.y_train)
10
11         criterion = dt_search.best_params_['criterion']
12         max_depth = dt_search.best_params_['max_depth']
13         max_leaf_nodes = dt_search.best_params_['max_leaf_nodes']
14
15         dtree = DecisionTreeClassifier(criterion=criterion,
16                                        max_depth=max_depth,
17                                        max_leaf_nodes=max_leaf_nodes).fit(self.X_train,
self.y_train)
18         predicted_dt = dtree.predict(self.X_test)
19         accuracy_dt = metrics.accuracy_score(self.y_test, predicted_dt)
```

Figura 30. Segmento de código de modelo de árbol de decisión. Fuente: Autor

Finalmente, ya con el árbol de decisión ajustado se realizan las predicciones con los datos de prueba “self.X.test”, calculando valores como la precisión mediante la función “metrics.accuracy_score” comparando los valores ingresados como prueba con los valores reales. Y así mismo como en el resto de los modelos se genera un reporte de clasificación final para analizar las métricas resultantes de la ejecución del modelo. Sin embargo, el objetivo final era la elaboración de un modelo Random Forest, el cual se detalla en la Figura 31 a continuación:

```
1  def RandomForest(self):
2      start_time = time.time()
3      RF = RandomForestClassifier(criterion='gini',
4                                  n_estimators=500,
5                                  min_samples_split=10,
6                                  #min_samples_leaf=1,
7                                  max_features='auto',
8                                  oob_score=True,
9                                  random_state=1,
10                                 n_jobs=-1).fit(self.X_train, self.y_train)
11
12     predicted_rf = RF.predict(self.X_test)
13     svm_accuracy = accuracy_score(self.y_test, predicted_rf)
```

Figura 31. Segmento de código del modelo Random Forest. Fuente: Autor

Dentro de este modelo se van a usar varios árboles de decisión, a través de la creación de una instancia “RandomForestClassifier”, donde se van a conFigura r varios parámetros, como se observa de la línea 3 a la 10, de entre los cuales destacan: “criterion”: utilizado para medir la calidad de una división o ramificación, “n_estimators”: que indica el número de árboles de decisión que tendrá el bosque, “n_jobs”: hace referencia al número de ejecuciones en paralelo para el ajuste de la predicción, entre otros. Posteriormente, se realiza el entrenamiento del modelo mostrado en la línea 10 y se realiza la predicción y la evaluación del modelo, como se observa en las líneas 12 y 13, generando finalmente un reporte de clasificación para la interpretación de los resultados.

2.4. Primeras impresiones.

Con cada uno de los modelos desarrollados, se realiza un análisis de cada uno de los reportes de clasificación generados, donde se pueden observar los resultados para cada uno de los modelos.

2.4.1. Resultados iniciales:

2.4.1.1. Regresión Logísitica:

Para el modelo de regresión logística, como se observa en la Figura 32, el reporte nos muestra una exactitud de predicción del 76.64%, teniendo como mejor "solver" a liblinear, acompañado de otras características del reporte como precisión y recall para las tramas maliciosas y benignas (1 y 0 respectivamente), así como el tiempo que ha tomado la ejecución del modelo.

```
M.LogisticRegression()

Accuracy: 76.64%

################################################################################
Best solver is :  liblinear
################################################################################
              precision    recall  f1-score   support

           0       0.84      0.79      0.81     20024
           1       0.66      0.72      0.69     11128

    accuracy                           0.77     31152
   macro avg       0.75      0.76      0.75     31152
weighted avg       0.77      0.77      0.77     31152


################################################################################
--- 4.343384027481079 seconds --- time for LogisticRegression
```

Figura 32. Resultado inicial de modelo de regresión logística. Fuente: Autor

2.4.1.2. Support Vector Machine:

Para SVM también se ha generado un reporte, el cual puede ser visualizado en la Figura 33, donde se muestra un resumen de la exactitud del modelo evaluado para cada kernel, arrojándonos en promedio una exactitud del 97%, teniendo como mejor opción de kernel a "rbf", acompañado de su respectiva precisión, recall para los dos tipos de tramas y el tiempo de ejecución del modelo.

```
M.SupportVectorMachine()

Accuracy: 78.40%
######################################################################
Accuracy: 96.53%
######################################################################
Accuracy: 96.67%
######################################################################
Accuracy: 54.52%
######################################################################
Accuracy of SVM model 97.0%

######################################################################
best kernel is :  rbf
######################################################################
              precision    recall  f1-score   support

           0       0.97      0.98      0.97     18750
           1       0.97      0.95      0.96     12402

    accuracy                           0.97     31152
   macro avg       0.97      0.96      0.97     31152
weighted avg       0.97      0.97      0.97     31152

######################################################################
--- 1042.876233100891 seconds ---
```

Figura 33. Resultado inicial de modelo de Support Vector Machine (SVM). Fuente: Autor

2.4.1.3. Árbol de decisión:

En lo que respecta al modelo de árbol de decisión, el reporte de la Figura 34 muestra un total de 5 “folds” o divisiones que se han realizado del conjunto de datos para trabajar con la validación cruzada, concretando así un porcentaje de exactitud de predicción del 98.22%, acompañado también de los parámetros de precisión y recall junto al tiempo de ejecución del modelo.

```
M.DecisionTree()

Fitting 5 folds for each of 180 candidates, totalling 900 fits
criterion: gini, max depth: 8, max_leaf: 11
The Accuracy is : 98.22%
######################################################################
              precision    recall  f1-score   support

           0       0.98      0.99      0.99     18743
           1       0.99      0.97      0.98     12409

    accuracy                           0.98     31152
   macro avg       0.98      0.98      0.98     31152
weighted avg       0.98      0.98      0.98     31152

######################################################################
--- 122.13890886306763 seconds ---
```

Figura 34. Resultado inicial del modelo de árbol de decisión. Fuente: Autor

2.4.2. Determinación de las características más importantes (peso).

Si bien, los resultados iniciales arrojados por los modelos son bastante buenos, estos podrían ser aún mejorados, por tal motivo, se propone modificar el conjunto de datos, con la finalidad de poder determinar las características más importantes y así evitar sobrecargar los modelos desarrollados con información que no aporten en el proceso de predicción y clasificación de los modelos.

Para lograr determinar los atributos más importantes o con mejor ponderación, se vuelve a emplear funciones de tratamiento de datos, donde a través de la función "weighted_features" conoceremos cual es la ponderación en peso de cada atributo del dataset, como se ilustra en la Figura 35.

	features	weights
0	src	17.87
1	pktcount	15.16
2	dst	13.64
3	byteperflow	12.97
4	pktperflow	11.35
5	pktrate	11.35
6	tot_kbps	9.68
7	rx_kbps	9.66
8	flows	8.95
9	bytecount	4.92
10	dt	2.33
11	Protocol	1.31
12	dur	1.11
13	tot_dur	1.11

Figura 35. Ponderación de pesos de los atributos del conjunto de datos. Fuente: Autor

La eliminación de atributos de acuerdo a su peso es un criterio que puede variar de acuerdo al modelo, ya que, el peso se encuentra directamente relacionado con la capacidad de contribución que tiene una variable para la predicción del modelo, si bien puede parecer un buen criterio, esto no siempre garantiza una mejora en el modelado, sin embargo basándose en los pesos mostrados de la Figura 35,

podemos ver que entre los pesos más elevados se encuentran “src” que hace referencia a la dirección IP de origen y “dst” que referencian a la dirección IP de destino, estos datos al tener un alto peso y poca relevancia en la clasificación de tramas, podrían estar ocasionando un sobreajuste en los modelos desarrollados, por lo que se ha optado por eliminarlos del conjunto de datos, del mismo modo el atributo “dt”, que si bien su eliminación no contribuye por su poco peso, se considera que es un atributo irrelevante, ya que, como nos indica la Tabla 2, se trata de un formato de fecha y hora que tampoco aporta en la clasificación de las tramas y su eliminación puede reducir un poco la complejidad del modelo y lograr aumentar la eficiencia de procesamiento de este.

2.4.3. Matriz de correlación.

La matriz de correlación se caracteriza por ser una herramienta estadística útil para el análisis de relación entre las variables o atributos de un conjunto de datos, de modo que se pueda comprender cómo los diferentes atributos se relacionan entre sí, así como el tipo de dependencia que existe entre ellos. Es utilizada para varios propósitos entre los cuales destacan: la identificación de relaciones lineales, visualización de patrones, selección de atributos, entre otros. [12]

Esta matriz puede ser de gran ayuda al momento de seleccionar atributos, ya que, si dos variables presentan una correlación elevada podría ser redundante y poco útil tenerlas a ambas dentro del conjunto de datos en el modelo, ya que, estaría proporcionando información que otro atributo ya se encuentra brindando al modelo.

Para esta situación, se propone una matriz de correlación con el nuevo conjunto de datos habiendo eliminado los atributos “src, dst y dt” que, por peso fueron descartados, obteniendo una matriz de correlación que se muestra en la Figura 36 a continuación:

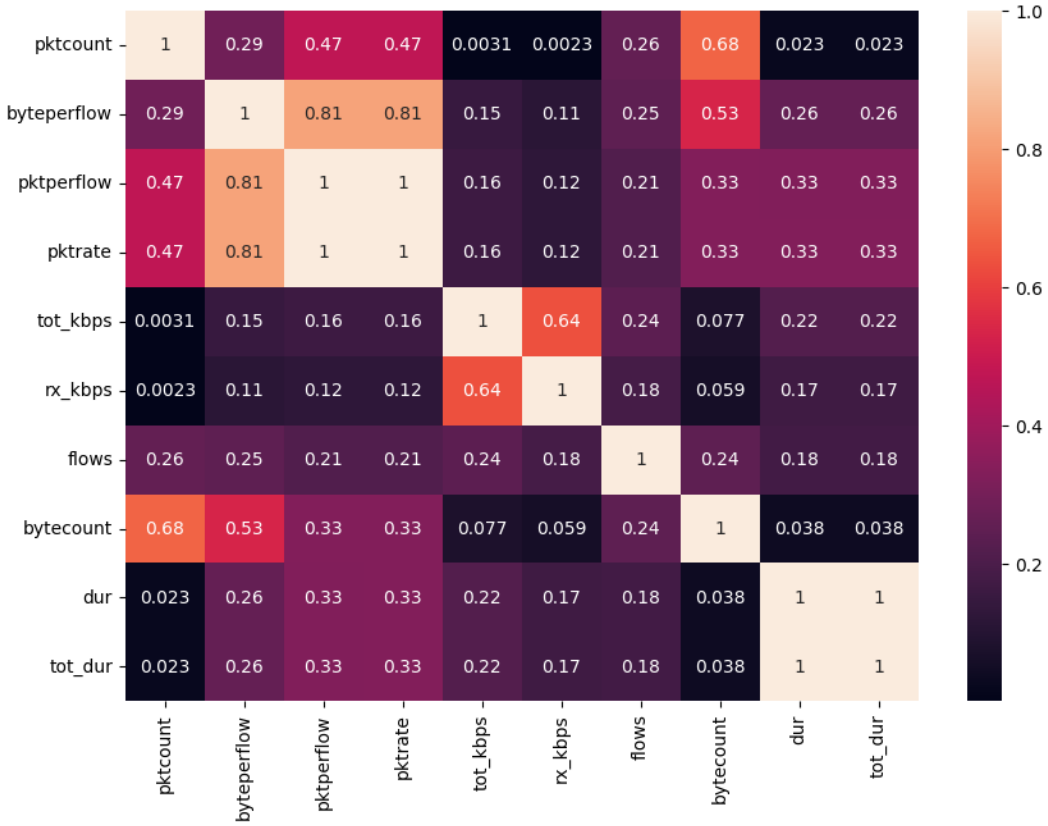


Figura 36. Matriz de correlación con primeros atributos eliminados. Fuente: Autor

Teniendo en cuenta que mientras los valores son más cercanos a 1, los atributos tienen una mayor correlación, se puede observar que los campos “dur” y “pktrate” tienen valores de 1, mientras que “pktperflow” de 0.81 indicando así que existe una alta correlación de estos con otros campos del conjunto de datos, por lo que, para evitar redundancias se decide eliminarlos.

Una vez eliminados los atributos que se han considerado innecesarios para el desarrollo de los modelos planteados, se comprueba una vez más empleando la matriz de correlación si una vez eliminados los campos presenta una menor correlación y por ende una menor redundancia. La nueva matriz de correlación se muestra en la Figura 37 a continuación:

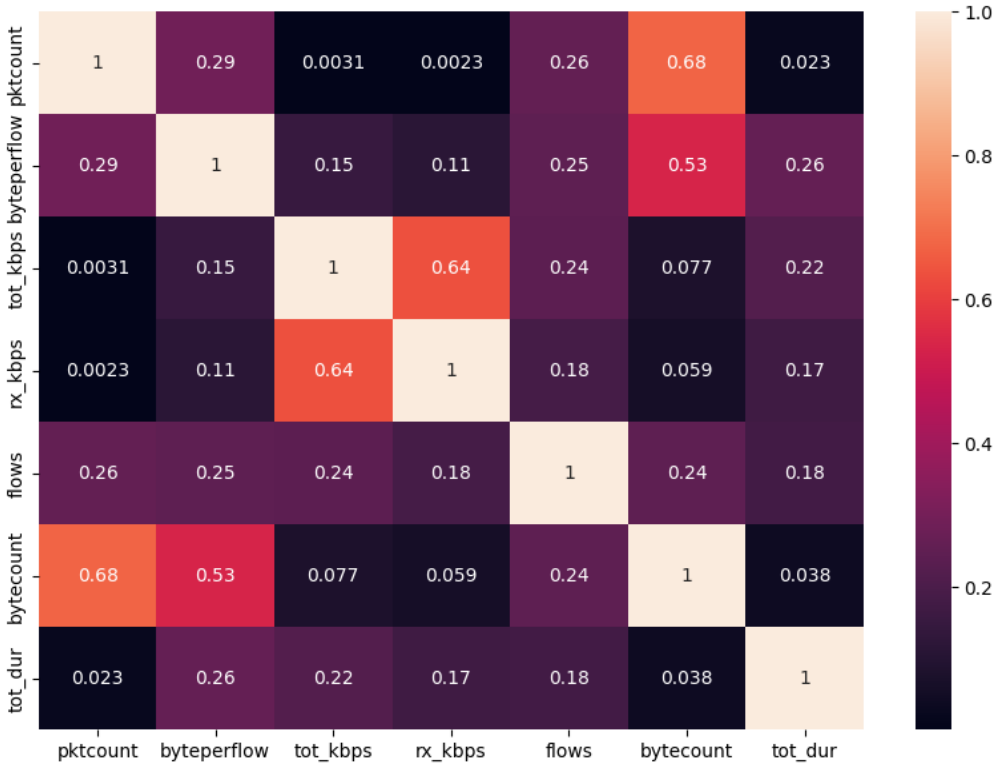


Figura 37. Matriz de correlación con los atributos finales eliminados. Fuente: Autor

2.4.4. Optimización de los modelos.

Como se puede apreciar en la Figura 37, habiendo eliminado los campos innecesarios que podían ocasionar redundancia o sobreajuste en el desarrollo del modelo no presenta más correlación entre los atributos del conjunto de datos, por lo que se pretende así reducir el margen de error en los modelos desarrollados previamente, si bien, este criterio no aplica en general para todos los modelos, se considera que si presentara una mejora en su reporte de clasificación. A continuación, se presentan los reportes de cada modelo, ejecutados con el nuevo dataset generado.

2.4.4.1. Regresión logística:

Como resultado del modelo tenemos ahora una precisión del 75.23% y adicionalmente cambia el mejor “solver” y pasa a ser “sag”, toda esta información y el reporte de predicción se puede ver en la Figura 38 a continuación:

```
M.LogisticRegression()

/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:416:
  warn(msg, LineSearchWarning)
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:306:
  warn('The line search algorithm did not converge', LineSearchWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/optimize.py:203: Use
  warnings.warn("Line Search failed")
Accuracy: 75.23%

######################################################################
Best solver is :  sag
######################################################################
              precision    recall  f1-score   support

           0       0.85      0.77      0.81     20974
           1       0.60      0.72      0.65     10178

    accuracy                           0.75     31152
   macro avg       0.73      0.74      0.73     31152
weighted avg       0.77      0.75      0.76     31152


######################################################################
--- 2.0529685020446777 seconds --- time for LogisticRegression
```

Figura 38. Resultado final de modelo de Regresión Logística. Fuente: Autor

2.4.4.2. Support Vector Machine (SVM):

Analizando el resultado final de SVM se observa que ahora cuenta con una exactitud promedio del 92% y también ha sido seleccionado como el mejor kernel “rbf”, toda esta información y su reporte se puede visualizar en la Figura 39 a continuación:

```
M.SupportVectorMachine()

Accuracy: 75.65%
######################################################################
Accuracy: 91.69%
######################################################################
Accuracy: 91.77%
######################################################################
Accuracy: 56.07%
######################################################################
Accuracy of SVM model 92.0%

######################################################################
best kernel is :  rbf
######################################################################
              precision    recall  f1-score   support

           0       0.90      0.96      0.93     17757
           1       0.95      0.86      0.90     13395

    accuracy                           0.92     31152
   macro avg       0.92      0.91      0.91     31152
weighted avg       0.92      0.92      0.92     31152

######################################################################
--- 733.0782959461212 seconds ---
```

Figura 39. Resultado final de modelo de Support Vector Machine (SVM). Fuente: Autor

2.4.4.3. Árbol de decisión:

Por otro lado, como resultado final del modelo del árbol de decisión, se tiene una exactitud del 94.19%, se muestra el tiempo de ejecución acompañado del resto de características que forman parte del reporte de clasificación, las cuales pueden observarse en la Figura 40.

```
M.DecisionTree()

Fitting 5 folds for each of 180 candidates, totalling 900 fits
criterion: gini, max depth: 6, max_leaf: 11
The Accuracy is : 94.19%
########################################################################
              precision    recall  f1-score   support

           0       0.91      1.00      0.95     17287
           1       1.00      0.87      0.93     13865

    accuracy                           0.94     31152
   macro avg       0.95      0.94      0.94     31152
weighted avg       0.95      0.94      0.94     31152

########################################################################
--- 54.60580563545227 seconds ---
```

Figura 40. Resultado final de modelo de árbol de decisión. Fuente: Autor

Finalmente, como se muestra en la Figura 41, se ejecutó el modelo de Random Forest, donde nos muestra un valor de exactitud elevado del 99.42%, acompañado de los demás parámetros que forman parte del reporte de predicción, siendo este uno de los que más alto porcentaje de exactitud ha proporcionado.

```
M.RandomForest()

/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:424:
  warn(
Accuracy of RF is : 99.42%

########################################################################
              precision    recall  f1-score   support

           0       0.99      1.00      1.00     18922
           1       1.00      0.99      0.99     12230

    accuracy                           0.99     31152
   macro avg       0.99      0.99      0.99     31152
weighted avg       0.99      0.99      0.99     31152

########################################################################
--- 23.608413457870483 seconds ---
```

Figura 41. Resultado final de modelo de Random Forest. Fuente: Autor

3. RESULTADOS, CONCLUSIONES Y RECOMENDACIONES.

3.1. Resultados.

El principal objetivo del presente TIC consiste en la implementación de un sistema de detección de amenazas en una red SDN, a partir de una arquitectura de una red que permita favorecer la clasificación de tramas administradas por un controlador SDN, el modelo debe ser programado en Python y debe ser basado en algoritmos de Machine Learning. Todo esto se ha logrado exitosamente usando Cisco Packet Tracer como principal herramienta de simulación, donde se pudo armar una topología de red con un controlador acorde a los requerimientos establecidos y con un número de hosts necesarios para la generación de una buena cantidad de tráfico, que nos permita obtener un conjunto de datos o "dataset" con las características y campos necesarios para tomarlo como base para el desarrollo del o los modelos de Machine Learning.

Al trabajar en el entorno de Cisco Packet Tracer, la generación del tráfico malicioso se ve muy limitado, ya que, estas herramientas de simulación son diseñadas para trabajar en ambientes de prueba y orientados a conexión, sin embargo las pruebas de conexión son realizadas por medio de la transmisión de paquetes TCP, UDP e ICMP, que si bien, son protocolos básicos de comunicación, no quedan aislados de poder ser utilizados como base de ataques y generación de tramas maliciosas, como lo son TCP Syn, UDP Flood y ataques ICMP que constan en la generación de peticiones con el objetivo de saturar la red y dejarla vulnerable afectando gravemente a su disponibilidad. Para la generación de este tipo de tramas se ha empleado uno de los hosts de la red simulada, la cual brinda la posibilidad de ejecutar código en Python, logrando así infectar la red SDN simulada. Si bien se puede generar todos estos datos desde el simulador y extraerlos empleando un script que permita guardar el tráfico en formato CSV, se ha optado por la utilización del dataset "DDOS attack SDN Dataset" [3], el cual cuenta ya con una sólida estructura de datos generados en Mininet, conteniendo tramas maliciosas y benignas con más de cien mil registros, por lo que fue idóneo, ya que cumple con la clase de ataques que se pueden generar y son soportados por las herramientas de simulación.

Una vez lista la base de datos se pudo en marcha el diseño del modelo de Machine Learning aproximado a una red neuronal convolucional, la cual después de varios análisis se determinó que es un modelo orientado más hacia el campo de imágenes y visión artificial, mas no al manejo de datos cualitativos y cuantitativos como los que proporcionan los conjuntos de datos generados en simuladores, por este motivo se optó por desarrollar tres modelos de predicción y clasificación. Previo a este desarrollo, se realizó un análisis profundo de los datos con los que se trabajó, donde empleando librerías de tratamiento de datos como Pandas, se pudieron generar gráficas que nos proporcionen información y una perspectiva global de los datos como el porcentaje de tramas maliciosas y benignas, número de solicitudes generadas a los diferentes hosts de la red, así como el número de solicitudes maliciosas y benignas realizadas por protocolo, representadas en las Figura s 42, 43 y 44 respectivamente.

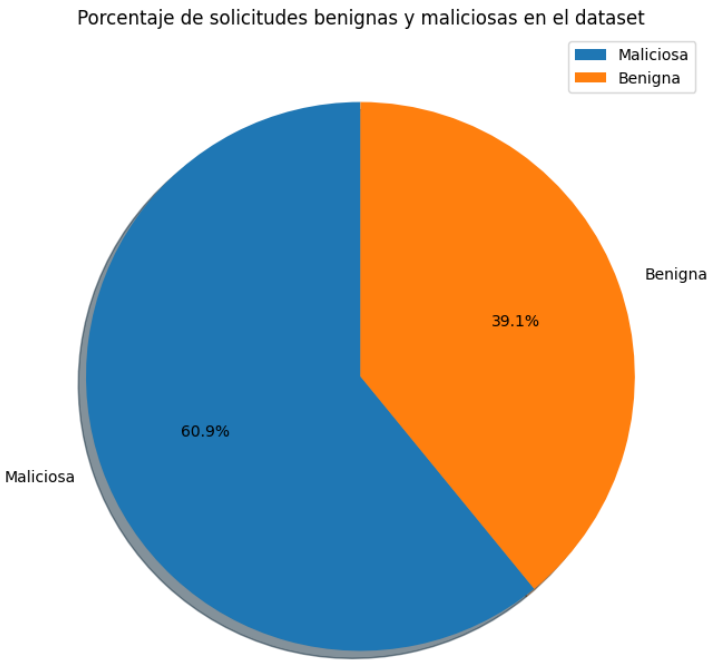


Figura 42. Porcentaje de solicitudes benignas y maliciosas en el dataset. Fuente: Autor

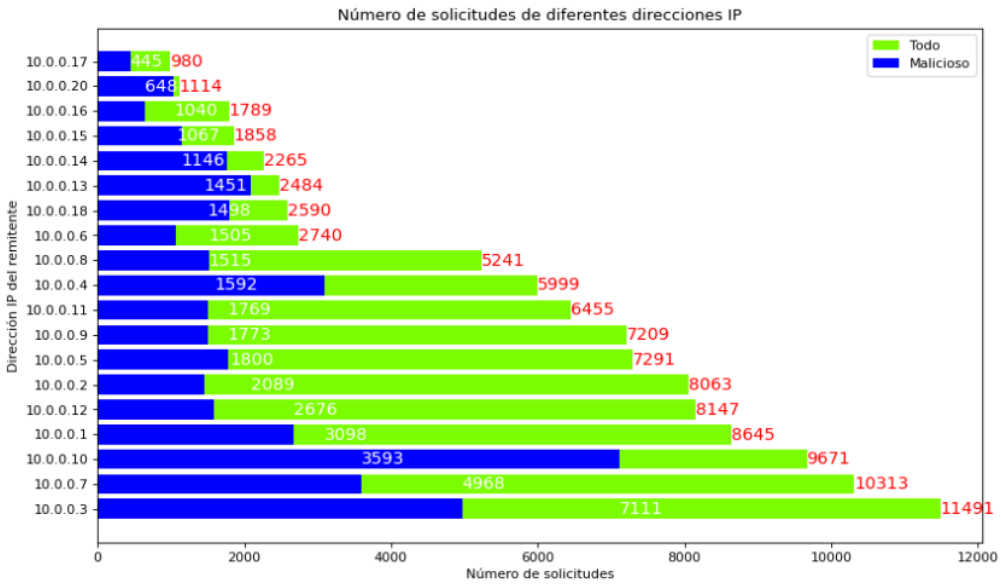


Figura 43. Cantidad de solicitudes maliciosas y benignas por host. Fuente: Autor

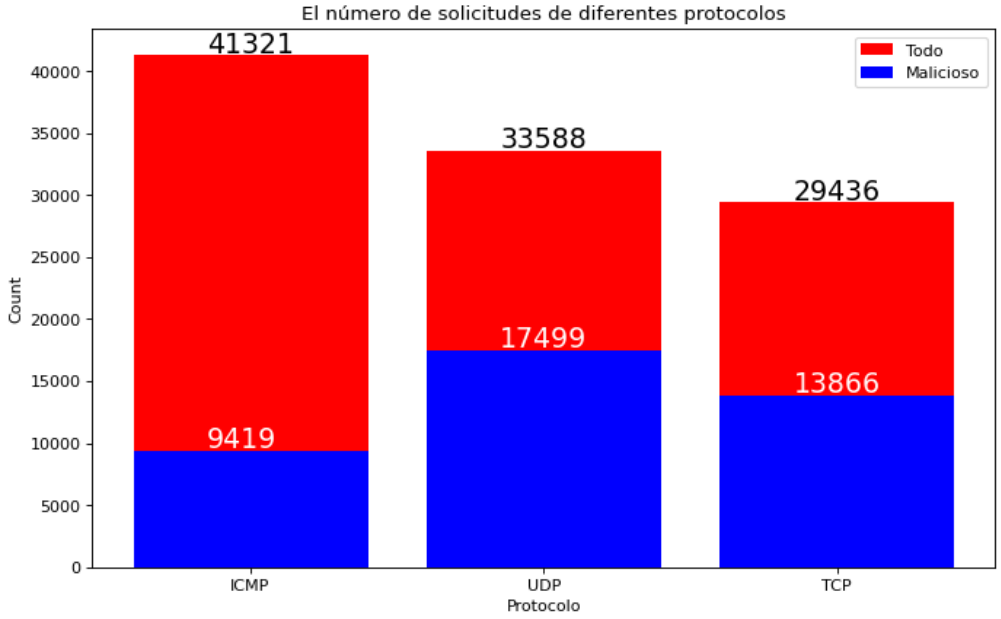


Figura 44. Número de solicitudes por protocolo. Fuente: Autor

Todas estas estadísticas nos dan una mejor perspectiva y una visión más completa de los datos con los que se va a trabajar, siendo esto fundamental para el desarrollo de los modelos de clasificación. Se ha decidido optar por la implementación de tres modelos: regresión lineal, support vector machine y árboles de decisión con random forest, donde se realizaron dos pruebas por cada modelo.

Regresión Lineal:

Para la elaboración de este modelo se trabajó con 3 tipos de solucionadores (newton, lbfgs, liblinear, sag y saga). Como primer resultado obtuvimos que el mejor "solver" o solucionador es "liblinear" arrojándonos una exactitud de predicción del 76.64% en un tiempo de ejecución de apenas 4.34 segundos aproximadamente, como se puede ver en la Figura 45 a continuación:

```
Accuracy: 76.64%

########################################################################
Best solver is :  liblinear
########################################################################
              precision    recall  f1-score   support

           0       0.84      0.79      0.81     20024
           1       0.66      0.72      0.69     11128

    accuracy                           0.77     31152
   macro avg       0.75      0.76      0.75     31152
weighted avg       0.77      0.77      0.77     31152


########################################################################
--- 4.343384027481079 seconds --- time for LogisticRegression
```

Figura 45. Primer resultado de modelo de Regresión Lineal. Fuente: Autor

Después de correr el modelo y obtener el primer resultado, se realizó una limpieza de los datos mediante la eliminación de campos o características de una copia del dataset original, todo esto basado en criterios de ponderación de pesos y matrices de correlación, eliminando así un total de 6 campos entre los que se encontraban: "dur", "pktrate", "pktperflow" entre otros más, obteniendo el resultado de la Figura 46.

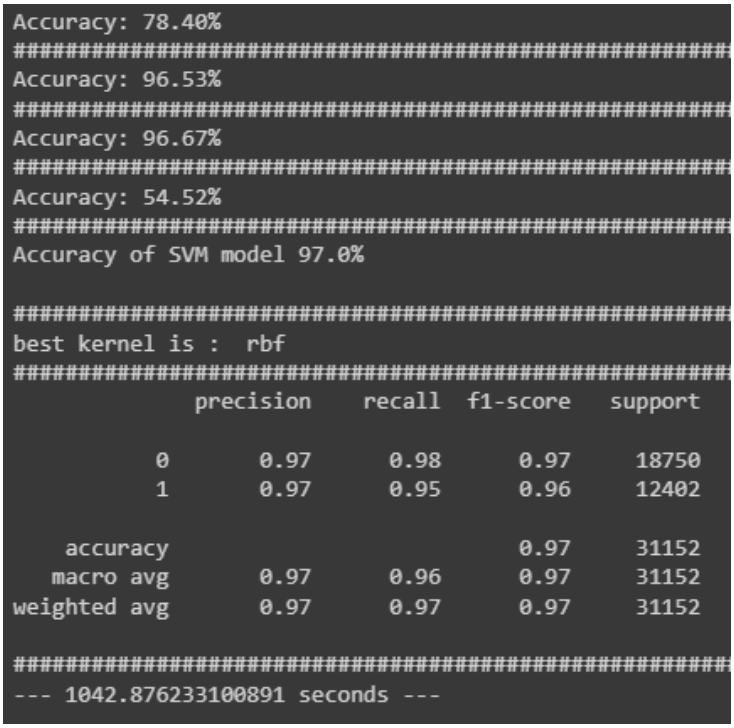
```
Accuracy: 75.23%

########################################################################
Best solver is :  sag
########################################################################
              precision    recall  f1-score   support

           0       0.85      0.77      0.81     20974
           1       0.60      0.72      0.65     10178

    accuracy                           0.75     31152
   macro avg       0.73      0.74      0.73     31152
weighted avg       0.77      0.75      0.76     31152


########################################################################
--- 2.0529685020446777 seconds --- time for LogisticRegression
```

Figura 46. Segundo resultado del modelo de Regresión Lineal. Fuente: Autor

Si bien, se puede observar una pequeña caída en la exactitud del 76.64% al 75.23% no existe mucho margen de pérdida, sin embargo, con el segundo resultado en la Figura 46, se ha conseguido disminuir el tiempo de ejecución de 4.35 segundos a 2.05 segundos, por lo que se ha conseguido optimizar el modelo y aumentar su eficiencia.

Support Vector Machine (SVM):

Para SVM también se emplearon tres kernels, de los cuales “rfb” fue el resultado de mejor kernel para ambos escenarios, antes y después de la eliminación de campos, consiguiendo los resultados mostrados en las Figura s 47 y 48.



```
Accuracy: 78.40%
##########################################################
Accuracy: 96.53%
##########################################################
Accuracy: 96.67%
##########################################################
Accuracy: 54.52%
##########################################################
Accuracy of SVM model 97.0%

##########################################################
best kernel is :  rbf
##########################################################
              precision    recall  f1-score   support

           0       0.97      0.98      0.97     18750
           1       0.97      0.95      0.96     12402

    accuracy                           0.97     31152
   macro avg       0.97      0.96      0.97     31152
weighted avg       0.97      0.97      0.97     31152

##########################################################
--- 1042.876233100891 seconds ---
```

Figura 47. Primer resultado del modelo de Support Vector Machine. Fuente: Autor

De igual modo que en el modelo de Regresión Lineal, SVM ha conseguido una exactitud menor con una diferencia del 5% en relación al primer resultado, sin embargo, se ha conseguido disminuir considerablemente el tiempo de 1042 a 733 segundos, dando así una muestra más del aumento de eficiencia y optimización de los modelos con la limpieza de campos.

```
Accuracy: 75.65%
##################################################
Accuracy: 91.69%
##################################################
Accuracy: 91.77%
##################################################
Accuracy: 56.07%
##################################################
Accuracy of SVM model 92.0%

##################################################
best kernel is :  rbf
##################################################
              precision    recall  f1-score   support

           0       0.90      0.96      0.93     17757
           1       0.95      0.86      0.90     13395

    accuracy                           0.92     31152
   macro avg       0.92      0.91      0.91     31152
weighted avg       0.92      0.92      0.92     31152

##################################################
--- 733.0782959461212 seconds ---
```

Figura 48. Segundo resultado del modelo de Support Vector Machine (SVM). Fuente: Autor

Random Forest:

Este modelo fue uno particular, ya que tiene como base los árboles de decisión, modelos que, fueron igualmente puestos a prueba antes y después del criterio de eliminación de los campos, obteniendo los resultados mostrados a continuación en las Figura s 49 y 50.

```
Fitting 5 folds for each of 180 candidates, totalling 900 fits
criterion: gini, max depth: 8, max_leaf: 11
The Accuracy is : 98.22%
##################################################################
              precision    recall  f1-score   support

           0       0.98      0.99      0.99     18743
           1       0.99      0.97      0.98     12409

    accuracy                           0.98     31152
   macro avg       0.98      0.98      0.98     31152
weighted avg       0.98      0.98      0.98     31152

##################################################################
--- 122.13890886306763 seconds ---
```

Figura 49. Primer resultado del modelo de Árbol de decisión. Fuente: Autor

Evidentemente en el modelo desarrollado del árbol de decisión también se observa un menor porcentaje de exactitud del algoritmo de un inicial 98.22% a un 94.19%, sin embargo, hay que notar el incremento de la eficiencia del algoritmo, ya que, si nos fijamos en el tiempo de ejecución, se pasa de 122.13 segundos en la Figura 49 a 54.60 segundos de ejecución después de la eliminación de campos.

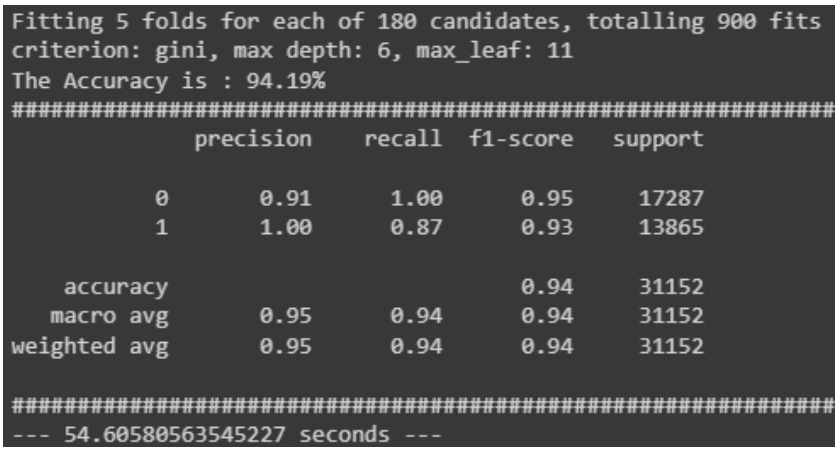


Figura 50. Segundo resultado del modelo de Árbol de decisión. Fuente: Autor

Comprobando así finalmente que la limpieza del conjunto de datos nos puede ayudar a eliminar redundancias y sobreajustes en el desarrollo de los modelos de Machine Learning, sin embargo, hay que tener en cuenta que se debe tratar de buscar un equilibrio entre la eficiencia y la calidad del modelo, ya que este criterio puede variar dependiendo de las aplicaciones a las que sea destinado el algoritmo.

3.2. Conclusiones.

La demanda de la seguridad de las redes con el paso del tiempo va creciendo exponencialmente, y es que tanto empresas como usuarios de dispositivos electrónicos son cada vez más conscientes de los riesgos que pueden llegar a tener por la falta de conocimiento, más aún en el área de las redes IoT, al ser uno de los campos que supone una mayor innovación, lleno de dispositivos inalámbricos, generalmente de bajos recursos, comunicándose entre sí para satisfacer las necesidades del usuario e intercambiando información sensible, requiere de una mayor atención, cuidado y capacitación.

Para lograr cumplir los objetivos del presente trabajo se recopiló información detallada sobre el funcionamiento de las rede IoT y SDN, profundizando en los aspectos de seguridad, los protocolos de transmisión que estas redes manejan así como las principales vulnerabilidades que tienen, analizando a la par diferentes algoritmos y modelos de clasificación de tramas, concluyendo que el uso de una red neuronal convolucional no era la mejor opción para trabajar con conjuntos de datos como los que proporciona el tráfico de red, ya que, estas se encuentran destinadas más al campo de las imágenes y la visión artificial.

Se ha logrado satisfactoriamente implementar varios sistemas de detección de amenazas mediante la creación de modelos de aprendizaje automático como lo son: regresión lineal, support vector machine (SVM) y random forest, a partir de la creación de una topología de red en Cisco Packet Tracer, administrada por un controlador SDN capaz de gestionar, analizar el tráfico de la red e interactuar con aplicaciones externas por medio de una API, siendo esta la que hace posible la integración de Python para el monitoreo y análisis externo de la red, lo cual fue fundamental para la incorporación de los modelos de Machine Learning así como para la clasificación efectiva de las tramas maliciosas y benignas. Comprobando cada uno de dichos modelos entre sí y evaluando parámetros como exactitud, precisión y tiempo de ejecución, concluyendo que, si bien un menor tiempo de ejecución aumenta la eficiencia del modelo, no hay que dejar de lado la exactitud y la precisión, ya que dependiendo de las aplicaciones que se le dé al modelo necesitaremos darle prioridad a una u otra.

3.3. Recomendaciones.

Analizar minuciosamente todos los detalles y alcance del TIC, tales como, información necesaria, aplicaciones o programas que se van a emplear en el desarrollo, lenguajes de programación, entro otros, con la finalidad de comprender la complejidad real del proyecto y dimensionar bien los tiempos y las tareas a realizar en el período de trabajo.

En caso de existir algún programa o lenguaje de programación con el que no se tenga mucha experiencia, adquirir preparación por medio de libros y cursos en línea para así fortalecer las debilidades y poder trabajar de manera óptima y sin retrasos por falta de conocimiento.

Al momento de realizar pruebas con códigos maliciosos como es el caso del script desarrollado para ataques DDoS, tener en cuenta únicamente entornos controlados, como máquinas virtuales o servidores privados, ya que, al ser una herramienta que puede perjudicar un servicio, la mala utilización de esta podría llegar a acarrear problemas legales. El desconocimiento de la ley no exime de culpabilidad.

Emplear un entorno de programación colaborativo y fácil de usar, como es el caso de Google Colab, que además de contar con múltiples extensiones y herramientas, permite la ejecución de código por bloques, lo cual facilita enormemente el desarrollo de cada uno de los modelos de predicción, ya que al encontrarse aislados es mucho más sencillo hacer pruebas sin afectar a los modelos vecinos. Además de contener un sinnúmero de librerías que basta con llamarlas correctamente para poder utilizarlas, mas no es necesario estar instalando por paquetes por separado como si sucede con otros entornos.

Tener conocimiento básico sobre las principales funciones de tratamiento y análisis de datos en pandas, ya que contiene varias funciones tanto de estudio como gráficas, las cuales pueden ampliamente ayudarnos en el desarrollo de un modelo de Machine Learning.

4. REFERENCIAS BIBLIOGRÁFICAS.

[1] H. D. M. R. R. C. Cartuche J, http://revistas.uap.edu.pe/ojs/index.php/HAMUT/article/view/2192/2295, Lima, Perú: Hamutay, 2020.

[2] J. M. Heras, «IArtificial.net,» 19 septiembre 2020. [En línea]. Available: https://www.iartificial.net/fases-del-proceso-de-machine-learning/. [Último acceso: 03 enero 2023].

[3] N. Ahuja, G. Singal y D. Mukhopadhyay, «DDOS attack SDN Dataset,» 27 septiembre 2020. [En línea]. Available: https://data.mendeley.com/datasets/jxpfjc64kr/1. [Último acceso: 12 agosto 2023].

[4] A. Sandoval, «Think Big/Empresas,» abril 25 2020. [En línea]. Available: https://empresas.blogthinkbig.com/las-redes-utilizadas-en-el-iot/. [Último acceso: 03 enero 2023].

[5] A. Gonzales, «Cleverdata,» 2021. [En línea]. Available: https://cleverdata.io/que-es-machine-learning-big-data/.

[6] D. Rodríguez, «Analytics Lane,» 23 julio 2018. [En línea]. Available: https://www.analyticslane.com/2018/07/23/la-regresion-logistica/.

[7] Na8, «Aprende Machine Learning,» 10 julio 2018. [En línea]. Available: https://www.aprendemachinelearning.com/clasificar-con-k-nearest-neighbor-ejemplo-en-python/.

[8] J. Barrios, «Health Big Data,» 22 abril 2022. [En línea]. Available: https://www.juanbarrios.com/redes-neurales-convolucionales/.

[9] T. C. R. 200-125, «Mis Libros de Networking,» 11 septiembre 2016. [En línea]. Available: http://librosnetworking.blogspot.com/2016/09/sdn-software-defined-network.html. [Último acceso: 2022 noviembre 22].

[10] erickosvaldovg, «Packet Tracer en sencillo,» 30 Septiembre 2014. [En línea]. Available: https://erickosvaldovg.wordpress.com/2014/09/30/que-es-packet-tracer/. [Último acceso: 4 julio 2023].

[11] F. Flores, «Open Webinars,» 22 julio 2022. [En línea]. Available: https://openwebinars.net/blog/que-es-visual-studio-code-y-que-ventajas-ofrece/. [Último acceso: 16 agosto 2023].

[12] D. S. Team, «Data Science,» 03 mayo 2020. [En línea]. Available: https://datascience.eu/es/matematica-y-estadistica/que-es-una-matriz-de-correlacion/. [Último acceso: 17 julio 2023].

5. ANEXOS

ANEXO I. Enlace al proyecto completo de implementación de los modelos de predicción.

https://colab.research.google.com/drive/17TfwPzXrN_dklVko-8l9Nkf8erDb3A0d?usp=drive_link