UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

TRABAJO FIN DE GRADO

Detección de ataque DDoS mediante el uso de aprendizaje automático

Autor: Alejandro de la Morena Vázquez
Tutor: Luis de Pedro Sánchez

julio 2025

Todos los derechos reservados.
Queda prohibida, salvo excepción prevista en la Ley, cualquier
forma de reproducción, distribución comunicación pública
y transformación de esta obra sin contar con la autorización
de los titulares de la propiedad intelectual.
La infracción de los derechos mencionados puede ser constitutiva de
delito contra la propiedad intelectual *(arts. 270 y sgts. del Código Penal).*

DERECHOS RESERVADOS
© 3 de Noviembre de 2017 por UNIVERSIDAD AUTÓNOMA DE MADRID
Francisco Tomás y Valiente, n^o 1
Madrid, 28049
Spain

Alejandro de la Morena Vázquez
Detección de ataque DDoS mediante el uso de aprendizaje automático

Alejandro de la Morena Vázquez
C\ Francisco Tomás y Valiente Nº 11

IMPRESO EN ESPAÑA – PRINTED IN SPAIN

Agradecimientos

Quisiera agradecer a mis padres, Rafi y Migui, por su compresión y cariño, en los buenos y malos momentos. Y a mi hermano Daniel por su apoyo constante y pundonor.

También dar las gracias a mis compañeros de universidad y amigos por hacer esta camino más llevadero y memorable y a los profesores de la universidad que han aportado significativamente a mi desarrollo académico y personal.

Resumen

En la última década, se ha visto intensificado las conexiones a Internet y los servicios en nube, pero a la vez también ha habido un incremento de los ciberataques. Por esto mismo surge la necesidad de herramientas y procesos para protegerse ante estos peligros. Este trabajo se basa en identificar ataques de denegación de servicio distribuido (*Distributed Denial of Service*, DDoS) los cuales consisten en saturar un servidor o red con una gran cantidad de tráfico, haciendo imposible su acceso a usuarios legítimos.

En este estudio se ha llevado a cabo la identificación de dichos ataques mezclados con tráfico benigno a través de diferentes algoritmos de aprendizaje, en específico Regresión Logística, Máquina de Vectores de Soporte (*Support Vector Machine*, SVM), *Random Forest* y *Gradient Boosting*. El conjunto de datos que se ha analizado proviene de diferentes flujos de red y abarca distintos tipos de tráfico. Los resultados obtenidos demuestran la eficacia de cada algoritmo y su potencial uso en casos reales.

Palabras clave

Ciberataques, denegación de servicio distribuido (DDoS), aprendizaje automatico, *gradient boosting*, *random forest*, regresión logística.

Abstract

Over the past decade, internet connections and cloud services have intensified, but at the same time, cyberattacks have also increased.This has led to a need for tools and processes to protect against these dangers. This work is based on identifying distributed denial of service (DDoS) attacks, which consists of saturating a server or network with a huge amount of traffic, making it impossible for legitimate users to access it.

In this study, these attacks were identified mixed with benign traffic using different learning algorithms specifically Logistic Regression, Support Vector Machine (SVM), Random Forest, and Gradient Boosting. The dataset analyzed comes from different network flows and includes various types of traffic. The results obtained demonstrate the effectiveness of each algorithm and their potential application in real-world scenarios.

Keywords

Cyberattacks, distributed denial-of-service(DDoS), machine learning, gradient boosting, random forest, logistic regression.

ÍNDICE

Listas

Lista de códigos

Lista de ecuaciones

Lista de figuras

Lista de tablas

1

Introducción

En este apartado se mostrarán las razones por las que se llevó a cabo este trabajo y los objetivos a alcanzar, así como las diferentes etapas del trabajo que se han realizado y un resumen de la estructura del documento.

1.1. Motivación

En un mundo cada vez más digitalizado, el uso de Internet ha aumentado exponencialmente y con ello los datos que se transmiten por la red. Esta creciente dependencia de los servicios en línea da lugar a la necesidad de protegerse ante posibles ataques e implementar herramientas capaces de detectarlos.

La expansión de Internet ha provocado también un aumento de los ciberataques, tanto las empresas como los individuos son objetivos frecuentes de estos ataques, lo que demuestra la urgencia de soluciones. No solo ha habido un aumento, sino que también cada vez son más complejos, dificultando su detección y poniendo en riesgo información sensible, como datos personales, financieros o confidenciales, cuya exposición puede causar pérdidas económicas y comprometer la privacidad de los usuarios. La complejidad de estos ataques en línea limita la capacidad con la que los sistemas tradicionales pueden detectarlos, ya que no se adaptan bien a patrones nuevos o desconocidos. Es aquí donde se encuentra uno de los mayores potenciales del aprendizaje automático la adaptabilidad, es capaz de ajustarse a las nuevas amenazas sin necesidad de redefinir nuevas pautas. Por ello, surgió este trabajo para desarrollar una herramienta capaz de analizar el tráfico de red y clasificarlo automáticamente, identificando si es tráfico benigno o malicioso.

1.2. Objetivos

El objetivo es desarrollar una herramienta capaz de detectar ataques DDoS analizando el tráfico de red mediante aprendizaje automático con el fin de identificar de forma instantánea si es una amenaza o no. El trabajo tendrá los siguientes puntos clave:

O-1.– Importar y preparar los conjunto de datos.

O-2.– Normalizar y separar los conjuntos en entrenamiento y prueba.

O-3.– Entrenar y evaluar el conjunto con diferentes modelos de *machine learning*.

O-4.– Representar y analizar los resultados de los distintos modelos.

1.3. Fases del trabajo

Este trabajo se inició en julio de 2024 con una etapa de búsqueda del tema a desarrollar, durante la cual se propusieron diversos temas relacionados con la ciberseguridad. Finalmente, se decidió por un tema enfocado en la detección de ciberataques con el uso de aprendizaje automático, en particular, en la detección de ataques DDoS. La decisión se tomó considerando la relevancia de los ataques en línea en la actualidad y el auge del aprendizaje automático en aplicaciones como la ciberseguridad.

Este se basa en una continuación al trabajo realizado por [1] el Sr. Abhiram B. S., el Sr. B. S. Sumanth, y el Sr. Kushal R. Para este trabajo se partió del código desarrollado por los autores ya mencionados, adaptándolo para después evaluarlo sobre su propio *dataset* llamado `dataset_sdn`. Después dicho código se evaluó para un nuevo *dataset*,`dataset_Canada`. La adaptación de este nuevo *dataset* fue la parte más desafiante debido a su mayor tamaño, la incorporación de nuevas cabeceras y una estructura de datos diferente respecto al `dataset_sdn`. Tras la evaluación de ambos conjuntos por separado, se procedió a la combinación de ambos con el objetivo de observar y analizar los nuevos resultados. A partir de estos resultados se llegaron a unas conclusiones sobre la compatibilidad de ambos, las limitaciones y dependencias del algoritmo empleado. Para representar las fases del trabajo se realizó un diagrama de Gantt como se muestra en la figura 1.1.

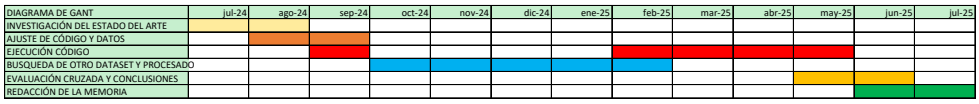

Figura 1.1: Diagrama de Gantt

1.4. Organización de la memoria

El documento presenta la siguiente estructura:

- **Capítulo 1 - Introducción:** se presentan los motivos por los cuales se llevó a cabo trabajo, el enfoque adoptado y las etapas principales que componen este proyecto.
- **Capítulo 2 – Estado del arte:** se describen los conceptos fundamentales a tener en cuenta para entender el proyecto, incluyendo temas como el aprendizaje automático o ciberataques.

- **Capítulo 3 – Diseño e implementación:** en este apartado se describen los pasos seguidos con detalle desde la recopilación del conjunto de datos hasta la implementación de los modelos.
- **Capítulo 4 – Pruebas y resultados:** se representan y analizan los resultados obtenidos tras las diferentes pruebas.
- **Capítulo 5 – Conclusiones y trabajo futuro:** análisis de los resultados mencionados anteriormente y posibles líneas de trabajo a continuar o mejorar el estudio.

2

Estado del arte

2.1. Introducción

En este apartado se mostrarán los puntos clave para entender y seguir el desarrollo del trabajo, proporcionando así una base sólida para comprender los conceptos esenciales. Se presentará la definición de ciberataque, así como los diferentes algoritmos de aprendizaje automático empleados en este trabajo. Se tratarán los siguientes temas:

- **Ataques en la red (Ciberataques).**
- ***Machine Learning.***
- **Regresión Logística.**
- **Máquina de vectores de soporte.**
- ***Random Forest.***
- ***Gradient Boosting.***
- **Estudios anteriores relacionados.**

2.2. Ataques en la red (Ciberataques)

Un ataque de red [2], o ciberataque, es una acción maliciosa diseñada con el objetivo de comprometer, dañar, robar o interrumpir sistemas informáticos. Estos ataques son dirigidos tanto a usuarios individuales como a grandes compañías. Estos ataques han ido en aumento a lo largo de los últimos años debido a la digitalización de los negocios. Existe una gran cantidad de ataques, como los ataques de *malware* (software malicioso como virus), también los ataques de *phishing* (suplantación de identidad) o del que se va a basar este estudio, el ataque DDoS (denegación de servicio distribuido).

El ataque DDoS [3] (*Distributed Denial of Service*) es una versión ampliada del ataque DoS (*Denial of Service*) en la cual los ataques ya no provienen de una sola fuente, sino que tienen origen desde varios puntos de conexión generando un gran tráfico en el servidor. Esto provoca que se sobrecargue y no pueda continuar prestando su servicio habitual. El hecho de que sean múltiples dispositivos ubicados en diversos puntos los que inyecten solicitudes maliciosas hace que sea mucho más potente y difícil de detener. A nivel global, este tipo de ataque ha ido en aumento tanto por su facilidad para crear

el ataque como por el número de dispositivos mal configurados.

Dentro de los ataques DDoS se pueden dividir en [4]:

- Ataques volumétricos: cuyo objetivo es sobrecargar el ancho de banda del sistema o servicio objetivo. Al desbordar el ancho de banda, lo que se consigue es ralentizar o impedir el tráfico legítimo de los usuarios.
- Ataques de protocolo: abusan del funcionamiento interno de las comunicaciones para saturar el sistema.
- Ataques a la capa de aplicación: cuyo principal propósito es provocar la caída del servidor mediante solicitudes supuestamente reales.

Para protegerse de los ataques DDoS, se pueden tomar dos tipos de enfoques: estrategias preventivas y estrategias de respuesta, como el uso de *firewalls* o sistemas de detección de intrusos (IDS) para reconocer y parar el tráfico maligno. Otra medida es distribuir el tráfico entre diferentes servidores, evitando que uno solo se desborde.

2.3. Machine Learning

Machine Learning [5], o aprendizaje automático, es una subdisciplina de la informática y una rama de la inteligencia artificial, cuyo objetivo es desarrollar herramientas capaces de construir sistemas que aprendan a partir de la experiencia. Un agente es capaz de aprender si su rendimiento mejora con el tiempo y los datos recibidos, sin que la habilidad adquirida este programada de forma explícita desde el inicio. Hay diversos tipos de algoritmos de aprendizaje automático:

- **Aprendizaje supervisado:** es un método que consiste en usar datos etiquetados para que puedan clasificarse nuevas entradas.
- **Aprendizaje no supervisado:** en este caso no se entrena al algoritmo con respuestas, sino que el algoritmo debe ser capaz de analizar los datos y descubrir por sí mismo patrones
- **Aprendizaje semisupervisado:** combina los dos algoritmos mencionados, para ello utiliza tantos datos etiquetados como no etiquetados.
- **Aprendizaje por refuerzo:** el algoritmo aprende a través de prueba y error, interactuando con el entorno y recibiendo recompensas o castigos según su toma de decisión.

Dentro del aprendizaje supervisado podemos encontrar diferentes tipos como los que se llevan a cabo en este trabajo, como regresión logística, *gradient boosting*, *support vector machine* o *random forest*.

2.4. Regresión Logística

La regresión logística [6] calcula la probabilidad de que ocurra un evento en función de un conjunto de datos con características conocidas e independientes. Este modelo se utiliza frecuentemente para clasificación y predicción. Su resultado es una probabilidad entre 0 y 1. Para llevar a cabo este método, se emplea una transformación llamada función *logit*, que devuelve la probabilidad de que ocurra o no

un escenario. Para ello se emplea la siguiente función sigmoide 2.1:

$$f(z) = \frac{1}{1 + e^{-z}} = \frac{1}{1 + e^{-(b_1 \cdot x_1 + ... + b_k \cdot x_k + a)}} \tag{2.1}$$

f(z) es la probabilidad estimada de que ocurra el resultado (en este caso 0 o 1) y z es la combinación lineal de las variables predictoras. Una vez obtenido el resultado f(z), se compara la probabilidad con un umbral de 0.5 para determinar si es 0 o 1 (malicioso o benigno), como se muestra en la figura 2.1.

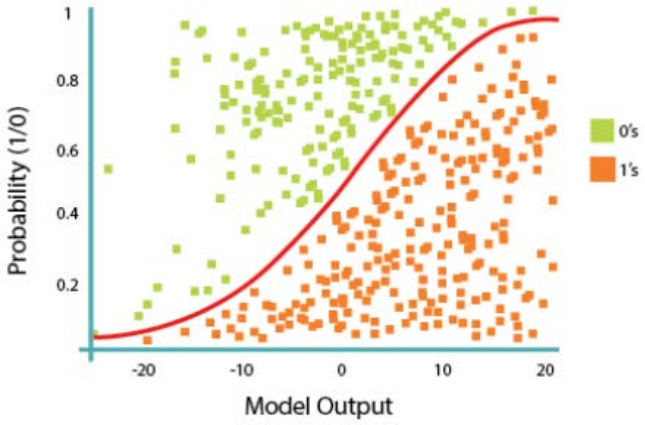

Figura 2.1: Regresión logística [7].

2.5. Support Vector Machine

La máquina de vectores de soporte [8] (SVM, *support vector machine*) tiene como objetivo principal encontrar un hiperplano óptimo, es decir, espacio que divide un espacio de múltiples dimensiones en dos partes separando los datos en diferentes clases. Además, busca que este hiperplano este separado lo máximo posible de los puntos más cercanos de cada clase. Estos puntos más próximos a la "frontera" se les llaman vectores de soporte.

Si los datos pueden separarse por una línea recta (2D) o por un plano (3D), entonces la máquina de vectores de soporte puede hallar esa clasificación sin problemas, pero cuando nuestros datos dependen de muchas características se emplea una técnica llamada *kernel*, que permite transformar los datos a un nuevo espacio con más dimensiones, donde es más probable que puedan clasificarse de manera correcta. Este proceso se ve en la siguiente figura 2.2

El rendimiento de la SVM depende mucho del tipo de *kernel* que se utilice y aunque son algoritmos potentes y precisos cuando se trabaja con un gran volumen de datos, su entrenamiento puede ser lento.

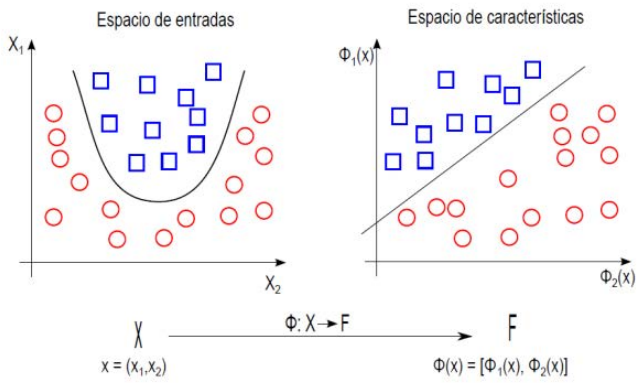

Figura 2.2: Support vector machine [9].

2.6. Random forest

Random forest [10] o bosque aleatorio es una técnica empleada para clasificar datos, consiste en crear muchos árboles de decisión que dividen la información en partes para tomar una decisión. Este método puede contar con decenas e incluso centenas de árboles de decisión donde cada uno se entrena y prueba con un subconjunto de datos y características.

Al final, para predecir una clase, este algoritmo combina las respuestas de todos los árboles y toma la predicción en función de la mayoría. Esto se ejemplifica en la siguiente figura 2.4. La combinación de todas las decisiones de los árboles hace que sea un método muy robusto y eficaz comparado con la de solo un árbol. Además, también se puede visualizar la importancia de cada característica a la hora de tomar una decisión viendo qué factores son las más influyentes.

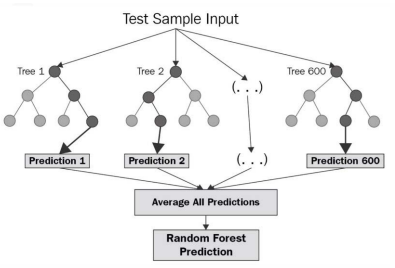

Figura 2.3: Random forest [11].

2.7. Gradient Boosting

Gradient boosting [12], o potenciación del gradiente, consiste en una técnica de aprendizaje automático basada en la construcción de un modelo robusto a partir de modelos más débiles. A diferencia de Random Forest, en donde los árboles se creaban de manera independiente, en este método cada árbol se crea en función del árbol anterior, de tal manera que se intenta corregir los errores anteriores.

El proceso comienza con un modelo sencillo que realiza una primera predicción. A continuación, se mide el error de dicha clasificación y con esta información se entrena otro árbol en donde se corrigen los errores anteriores. Este árbol no reemplaza al primero, sino que se suma. Este procedimiento, como se muestra en la figura 2.4, se repite varias veces mejorando cada vez más la precisión del modelo final.

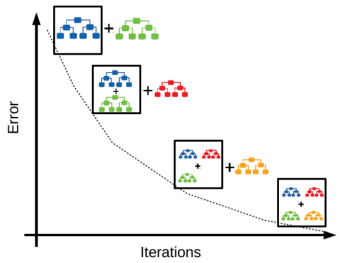

Figura 2.4: Gradient boosting [13].

2.8. Estudios anteriores relacionados

Durante el desarrollo de este trabajo se ha llevado a cabo un proceso de estudio, análisis e implementación con el objetivo de diseñar un sistema eficaz para la detección de ataques DDoS, utilizando algoritmos de aprendizaje automático.

El presente trabajo se basa en una continuación y desarrollo del proyecto [1] realizado por el Sr. Abhiram B. S., el Sr. B. S. Sumanth, y el Sr. Kushal R. Este proyecto supuso un punto de partida clave para entender los desafíos y posibilidades que presenta la detección de tráfico malicioso, así como para comprender los diferentes algoritmos de aprendizaje automático que se muestran en él y analizar y evaluar la eficiencia de cada uno de ellos con la base de datos que se obtuvo en el desarrollo del trabajo.

El estudio [1] “DDos Detection Using Machine Learning Algorithms” muestra de forma clara como la técnica del aprendizaje automático es una herramienta eficaz para mejorar la detección y clasificación de ataque DDoS. A diferencia de los métodos tradicionales que presentan reglas ya definidas con *machine learning*, se pueden reconocer nuevos patrones de manera más precisa y con mayor adaptación a cambios.

En la fase de análisis se llevó a cabo una representación visual detallada para entender de manera clara cómo se comportaba el tráfico tanto en situaciones habituales como en casos de ataque. A lo largo del estudio se muestran gráficas acerca de direcciones IP origen, protocolos utilizados, duración de los eventos, distribución de las solicitudes maliciosas y benignas. Además, se identificaron valores nulos y se llevó a cabo normalización y selección de rasgos para mejorar el rendimiento de los modelos.

En dicho proyecto además se detalla el uso entornos como *Jupyter Notebook* para llevar a cabo el

análisis y la representación gráfica de los datos, junto con el uso de librerías como *Pandas*, *NumPy*, *Seaborn*, *Matplotlib* y *Scikit-Learn*. A lo largo de este trabajo se muestra una perspectiva completa que abarca desde la recolección de los datos, depuración y análisis de los mismos y evaluación y comparación de resultados obtenidos.

En cuanto a los resultados dados, destaca la precisión del *Random Forest* con un 100 % seguido del modelo *Gradient Boosting* con un 99.54 % y del *Support Vector Machine* con un 97 % y por último la regresión logística obtuvo el peor resultado con un 76.64 %.

Cabe señalar que el código proporcionado de este estudio en *GitHub* no se encontraba completamente finalizado, por lo que fue necesario revisarlo, modificarlo y completarlo para obtener resultados concluyentes.

Este estudio además tiene una base sólida bibliográfica donde se recopilan investigaciones que han empleado técnicas como LSTM, *Naive Bayes* y modelos basados en *Random Forest* en contextos como redes definidas por *software* (SDN) y sistemas IoT. Todo esto pone de contexto de por qué se han elegido dichos algoritmos de aprendizaje para el proyecto.

El trabajo también muestra las consecuencias fatales que conllevan los ataques DDoS, como pérdidas económicas, deterioro de la reputación o como técnica de distracción para realizar otros ataques. Lo cual refuerza la necesidad de un modelo capaz de detectar estos ataques a tiempo.

2.9. Conclusiones

A lo largo de este trabajo se ha evidenciado de forma clara la problemática del aumento de los ciberataques, en concreto de los ataques DDoS, que presentan una amenaza tanto para organizaciones como individuos particulares. Con el objetivo de reconocerlos y aplacarlos se ha demostrado la eficacia del uso de métodos como algoritmos de aprendizaje automático, los cuales dan una solución automatizada y adaptable.

Se han escogido cuatro algoritmos de aprendizaje automático regresión logística, máquinas de vectores de soporte, *random forest* y *gradient boosting*. Cada uno de sus métodos presenta diferentes características, como el bajo coste computacional de la regresión logística como alta capacidad de clasificación de *random forest* y *gradient boosting*.

3

Diseño y desarrollo

3.1. Introducción

A lo largo de este capítulo se explicarán las etapas que componen el proyecto, desde la obtención del *dataset* y del código base de un repositorio en *GitHub* [14] hasta la evaluación y entrenamiento cruzados de los dos *datasets*. Todo este proceso se muestra en la figura 3.1

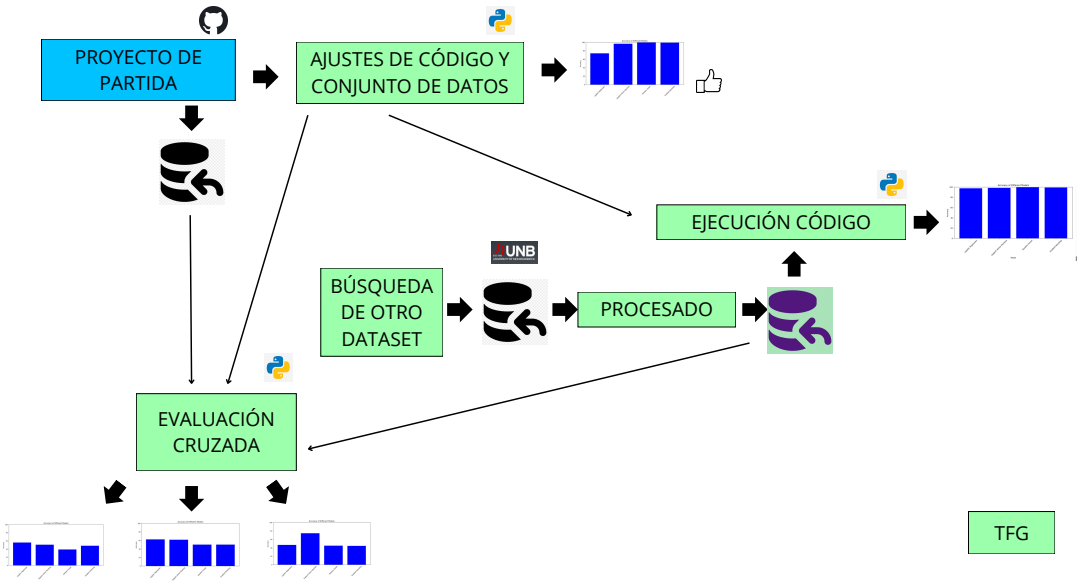

Figura 3.1: Flujo de trabajo

En primer lugar, se obtuvo el conjunto de datos junto con el código correspondiente desde *GitHub* [14]. Luego se realizaron las modificaciones necesarias, tanto para el código como para el conjunto de datos, para el correcto funcionamiento de los modelos, estos ajustes se realizaron a través de *Python* utilizando el entorno *Jupyter Notebook* dando lugar al `dataset_sdn`. Una vez completada esta etapa inicial, se buscó un nuevo conjunto de datos con el objetivo de ampliar la base de análisis y evaluar el modelo en diferentes condiciones. Para este nuevo *dataset*, el cual se obtuvo a partir del conjunto de datos *CICDDoS2019* [15] también hubo que realizar ciertas modificaciones tanto en el

mismo como en el código de igual manera en *Python* a través de *Jupyter Notebook*, el conjunto de datos originado a partir de estos cambios se llamó `dataset_Canada`. Una vez evaluados los dos *datasets* por separado, se procedió a su evaluación cruzada.

3.2. Preprocesamiento del dataset original y resolución de errores iniciales

Como se mencionó anteriormente, tanto el conjunto de datos como el código provienen de un repositorio de *GitHub* [14], como parte del trabajo realizado por estudiantes de la universidad BNM Institute of Technology. Durante este proceso, se detectó un error.

El problema se debía a que el *dataset* contenía algunas columnas de tipo texto, para ello en el código estas columnas se eliminaban, ya que se intentaba realizar operaciones matemáticas sobre palabras en vez de valores numéricos. Pero en el código original faltaba por eliminar la columna 'protocolo', por lo que se introdujo la siguiente línea 3.1 para eliminar dicha característica.

Código 3.1: Eliminación de protocolo

```
1 X = df.drop(['dt','src','dst','label','Protocol'], axis=1)
```

También se observó que el conjunto de datos original presentaba un desequilibrio, puesto que tenía mayor cantidad de tráfico malicioso que de tráfico benigno, en concreto un 60.9 % y un 39.1 % respectivamente. Esta diferencia de porcentajes podría afectar a la calidad de la clasificación por lo que se ajustó el tamaño para que hubiese una igualdad entre ambas clases. Esto se consiguió gracias al siguiente código 3.2, en donde se iguala la cantidad de datos maliciosos a los benignos y se mezclan para formar un conjunto equilibrado.

Código 3.2: Balanceo de clases

```
1 benign_count = len(data[data.etiqueta == 1])
2 malicious_data = data[data.etiqueta == 0 ]
3
4 malicious_sample = malicious_data.sample(n=benign_count, random_state=42)
5
6 balanced_data = pd.concat([data[data.etiqueta == 1 ], malicious_sample])
7 balanced_data = balanced_data.sample(frac=1, random_state=42).reset_index(drop=True)
8 print(balanced_data.etiqueta.value_counts())
```

Después de llevar a cabo este ajuste con éxito, como se muestra en la figura 3.2, el código se ejecutó sin ningún problema y los modelos comenzaron a funcionar correctamente. Gracias a esto, fue

posible continuar con las etapas de análisis y evaluación de la eficacia del modelo confirmando que los ajustes al código funcionaron correctamente.

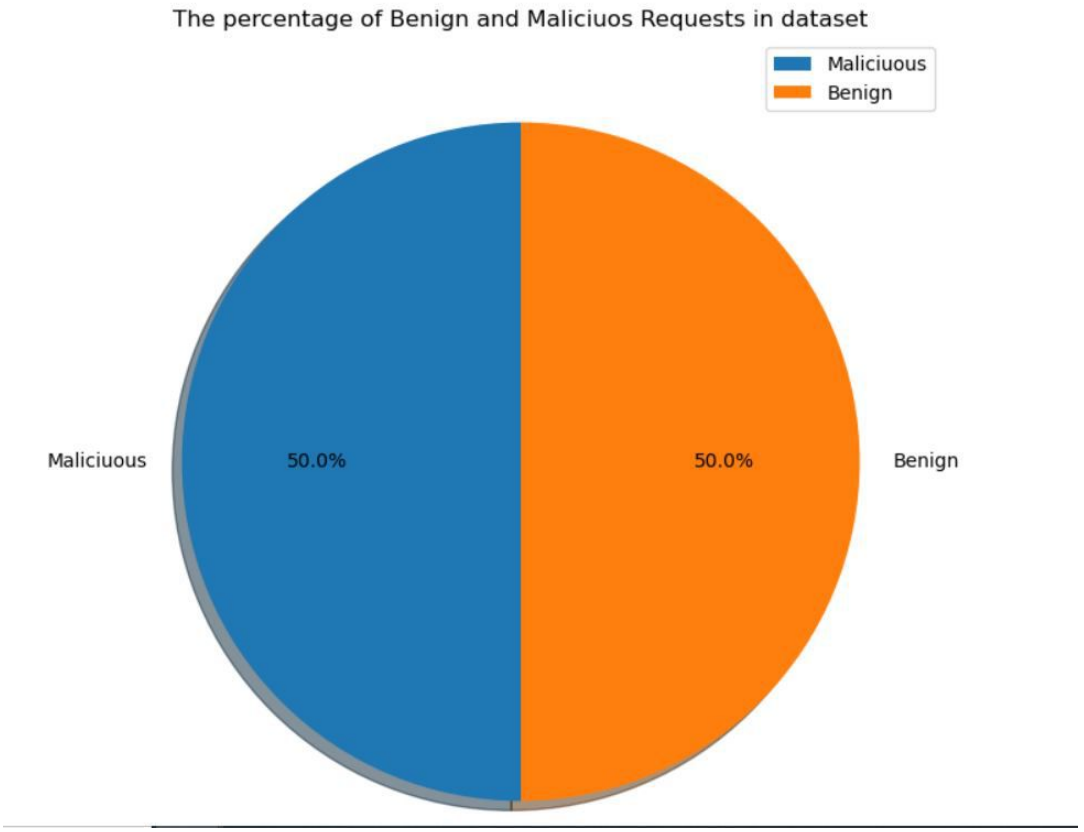

Figura 3.2: Gráfico equilibrio de clases.

3.3. Integración del dataset CICDDoS2019

Con el fin de continuar con la evaluación del modelo y analizar su capacidad para adaptarse a diferentes entornos y tipos de ataque, se escogió una nueva base de datos: el *CICDDoS2019* [15]. Este conjunto de datos fue generado por el Canadian Institute for Cybersecurity (CIC), este *dataset* tiene un gran reconocimiento en el campo de la investigación en la ciberseguridad por su alta calidad, diversidad de ataques y su importancia en el análisis de ataques en la actualidad.

Esta base de datos canadiense se caracteriza por albergar una gran cantidad de tráfico de red, simulando un entorno realista e incluyendo una gran diversidad de tipos de ataques DDoS que aprovechan protocolos como *UDP*, *TFTP* o *DNS* entre otros. Además, cada tipo de ataque está identificada mediante etiquetas, lo que facilite el entrenamiento supervisado en *machine learning*. Este *dataset* además presenta tanto tráfico malicioso como benigno presentando condiciones similares a situaciones reales. Este conjunto de datos se seleccionó debido a una serie de características:

- Diversidad de ataques: Al ofrecer una gran variedad de ataques permite al modelo entrenar con diferentes tipos y aprender a reconocer diferentes patrones.

- Entorno realista: El tráfico fue obtenido en condiciones que simulan situaciones reales lo que hace que se practicó y aplicable.
- Reconocimiento en el ámbito académico: Este conjunto de datos es aceptado y utilizado en estudios académicos, lo que aumenta su relevancia y fiabilidad.

Después de descargar el conjunto de datos *CICDDoS2019* se llevó a cabo un proceso de preprocesamiento. Para este proyecto se utilizaron los siguientes subconjuntos: *MSSQL, DrDoS_DNS, Portmap, DrDoS_NTP, DrDoS_SSDP, DrDoS_UDP, TFTP*. Debido al gran tamaño de estos archivos, se decidió balancearlos por separado, a través del código 3.2 ya antes visto, asegurando una distribución equitativa del 50 % para el tráfico benigno y 50 % para el tráfico malicioso como se puede apreciar en la figura 3.3

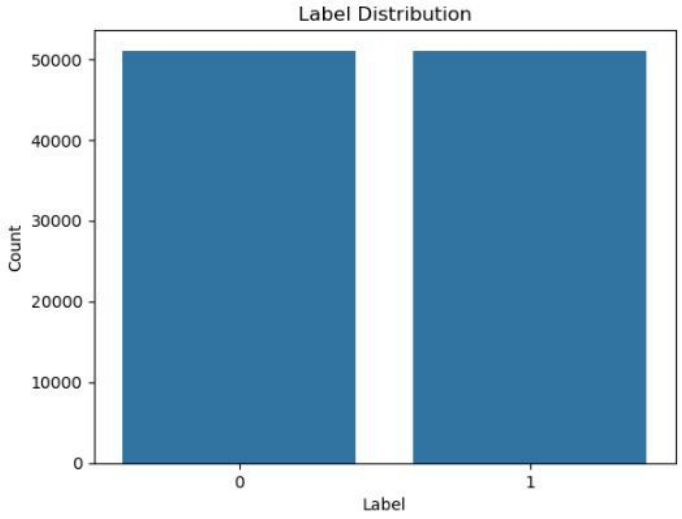

Figura 3.3: Equilibrio de clases Canada.

Una vez se consiguió balancear todos los archivos, fueron concatenados utilizando el código 3.3 para crear un único conjunto de datos llamado `dataset_Canada`.

Código 3.3: Concatenación de datos

```
1 dataset_Canada=
      pd.concat([data_MSSQL,data_DrDoS_DNS,data_Portmap,data_DrDoS_NTP,data_DrDoS_SSDP
2                ,data_DrDoS_UDP,data_TFTP], ignore_index=True)
```

Este proceso fue sencillo para la mayoría de conjuntos a excepción del *TFTP*, debido a su gran tamaño. Primero se dividió el archivo en fragmentos más pequeños, de unos 5 millones de filas, para poder manejarlo con más facilidad, después se llevó a cabo un balanceo de cada una de estas divisiones para tener una proporción adecuada entre tráfico malicioso y benigno y ya por último una vez equilibrados todos los fragmentos, se formó un nuevo conjunto completo pero ahora más ligero y equilibrado.

Después de concatenar los subconjuntos balanceados del *CICDDoS2019* en un único conjunto llamado `dataset_Canada`, se realizó una comparación estructural con el conjunto de datos original, `dataset_sdn`, con el propósito de identificar las características similares y garantizar la compatibilidad para su análisis y evaluación utilizando el mismo modelo.

El primer paso de este proceso comenzó con la identificación y selección de las columnas que debían conservarse. Para ello, se llevó a cabo un análisis sobre el contenido y relevancia de las columnas que conformaban tanto el *dataset* original, `dataset_sdn`, como las presentes en el nuevo conjunto de datos generado,`dataset_Canada`.

A lo largo de este estudió se observó una gran diferencia en la estructura de dichos conjuntos. Por un lado, el `dataset_sdn` estaba formado por un total de 23 columnas mientras que el conjunto `dataset_Canada` estaba compuesto por 90 columnas, muchas de las columnas no guardaban correspondencia directa o presentaban diferentes nombres o escalas diferentes.

Aunque existía una diferencia significativa, se logró encontrar 11 columnas que coincidían o presentaban muchas similitudes entre ambos conjuntos. Estas equivalencias fueron claves para definir una base común entre ambos *datasets*, permitiendo así reutilizar el modelo sin tener que modificar completamente su arquitectura.

A continuación, se presentan los nombres de dichas columnas del `dataset_sdn`, junto con su respectiva explicación y su correspondencia establecida con el *dataset* originado a partir del de Canadá a través de la tabla 3.1:

Bytecount: refleja la cantidad total de bytes transmitidos durante la comunicación en la red. Mide el volumen de datos en bytes.

packetins: indica el número de paquetes introducidos en la red durante el evento. Rastrea el número de paquetes añadidos a la red.

dur: representa la duración del evento de red en segundos. Señala el intervalo de tiempo que abarco la actividad.

byteperflow: calcula la media de bytes transmitidos por cada flujo.

pktrate: muestra la tasa de transmisión de paquetes por segundo. Indica la velocidad a la que se propagan los paquetes.

tx_bytes: indica la cantidad de bytes transmitidos durante el flujo de red. Mide el volumen de datos enviados desde el origen.

rx_bytes: registra la cantidad de bytes recibidos durante el flujo de red. Mide el volumen de datos recibidos por el destino.

tot_kbps: indica la tasa global de transferencia de datos en kilobits por segundo (kbps). Representa

la velocidad de envío como la de recepción.

Pairflow: indica el número de flujos bidireccionales entre pares de origen y destino.

label: indica la etiqueta de clasificación. Diferencia entre tráfico normal o malicioso.

pktperflow: calcula el número medio de paquetes transmitidos por cada flujo. Proporciona información sobre la densidad de tráfico por flujo.

La columna *bytecount* es la suma de las columnas *Total Length of Fwd Packets* y *Total Length of Bwd Packets*. Al igual que la columna *packetins* es la suma de *Total Fwd Packets* junto a *Total Backward Packets*.

Dataset_Canada	**Dataset_sdn**
Flow Duration	dur
Packet Length Mean	pktperflow
Average Packet Size	byteperflow
Flow Packets/s	pktrate
Total Length of Fwd Packets	tx_bytes
Total Length of Bwd Packets	rx_bytes
Flow Bytes/s	tot_kbps
pairflow	Pairflow
Total Length of Fwd Packets + Total Length of Bwd Packets	bytecount
Total Fwd Packets + Total Backward Packets	packetins
Etiqueta	Label

Tabla 3.1: Correspondencia de características Dataset_Canada y Dataset_sdn

Después de establecer la correspondencia entre las columnas de ambos conjuntos de datos, fue obligatorio la adaptación de algunas de estas al nuevo *dataset*, al que se llamara `dataset_Canada`. para que fueran compatibles con el modelo y la estructura del conjunto original, `dataset_sdn`. Estas modificaciones fueron las siguientes, en primer lugar se crearon las columnas *Bytecount* y *packetins*, puesto que eran columnas que se encontraban en `dataset_sdn`, pero no explícitamente en `dataset_Canada` al ser la suma de dos columnas ya existentes, esto se logró mediante la implantación del código 3.4.

Código 3.4: Creación de bytecount y packetins

```
1 data['bytecount'] = data['Total_Length_of_Fwd_Packets'] + data['_Total_Length_of_Bwd_Packets']
2 data['packetins'] = data['_Total_Fwd_Packets'] + data['_Total_Backward_Packets']
```

También se observó que algunas columnas se detectaban como texto en lugar de numéricas, lo que impedía su uso para cálculos y la limitación para usarlas en el entrenamiento del modelo. Es por eso

que se llevó a cabo la conversión de estas características a formato numérico, para poder manejarlas adecuadamente, como se muestra en estas líneas de código 3.5, en concreto se transformaron las siguientes tres columnas: *Packet Length Mean*, *Flow Bytes/s*, *Flow Packets/s* y *Average Packet Size*.

Código 3.5: Conversión númerica

```
1  data['␣Packet␣Length␣Mean'] = pd.to_numeric(data['␣Packet␣Length␣Mean'], errors='coerce')
2  data['Flow␣Bytes/s'] = pd.to_numeric(data['Flow␣Bytes/s'], errors='coerce')
3  data['␣Flow␣Packets/s'] = pd.to_numeric(data['␣Flow␣Packets/s'], errors='coerce')
4  data['␣Average␣Packet␣Size'] = pd.to_numeric(data['␣Average␣Packet␣Size'], errors='coerce')
```

Este proceso de conversión fue un punto clave en el preprocesamiento, ya que aseguró que los datos se encontraran el formato adecuado para su posterior análisis, previniendo errores en etapas posteriores y críticas como el entrenamiento y evaluación de los modelos.

Y por último hubo que cambiar los nombres de ciertas columnas para que coincidieran con los nombres del `dataset_sdn`, se cambiaron las siguientes variables: *Flow Duration*, *Packet Length Mean*, *Average Packet Size*, *Flow Packets/s*, *Total Length of Fwd Packets*, *Total Length of Bwd Packets*,*Flow Bytes/s* y *pairflow*. La correspondencia a estas variables se encuentra en la tabla 3.1. Para la modificación de dichas columnas se empleó el siguiente código 3.6.

Código 3.6: Renombramiento de columnas

```
1  balanced_data_df.rename(columns={
2      '␣Flow␣Duration': 'dur',
3      '␣Packet␣Length␣Mean': 'pktperflow',
4      '␣Average␣Packet␣Size': 'byteperflow',
5      '␣Flow␣Packets/s': 'pktrate',
6      'Total␣Length␣of␣Fwd␣Packets': 'tx_bytes',
7      '␣Total␣Length␣of␣Bwd␣Packets': 'rx_bytes',
8      'Flow␣Bytes/s': 'tot_kbps',
9      'pairflow': 'Pairflow'
10 })
```

Una vez realizado estos cambios se procedió a la ejecución del modelo entrenado y evaluado originalmente con `dataset_sdn`, ahora ejecutado por el conjunto modificado `dataset_Canada`.

También se ejecutó el conjunto `dataset_sdn`, pero esta vez solo con las columnas exclusivamente presentes en el `dataset_Canada` quedando así un conjunto de once columnas de las veintitrés que presentaba originalmente este conjunto, eliminando columnas como*switch*, *pktcount*, *dur_nsec*, *tot_dur*, *flows*, *port_no*,*tx_kbps*, *rx_kbps* debido a la incompatibilidad con el `dataset_Canada`.

3.4. Evaluación cruzada entre modelos

Con el propósito de comprobar la adaptación del modelo a nuevos entornos de red se llevó a cabo una evaluación cruzada empleando dos conjuntos de datos diferentes. Esta estrategia pone en contexto un escenario más realista, donde un sistema de detección entrenado para un *dataset* en concreto debe ser capaz de identificar ataques en otro *dataset* con características diferentes.

Analizar el rendimiento del modelo bajo estas condiciones es clave para asegurar que el modelo siga siendo eficaz en un entorno diferente al que fue entrenado.

Se llevaron a cabo dos configuraciones, la primera se entrenó el modelo con el conjunto `dataset_Canada` y se evaluó el conjunto `dataset_sdn`. Después se realizó lo contrario, se entrenó con `dataset_sdn` y se evaluó para `dataset_Canada`. Estos dos experimentos se realizaron con los *datasets* con las once características en común definidas anteriormente, garantizando la compatibilidad entre ambos. Para ello en el modelo se definieron cuatro entradas dos de entrenamiento y dos de evaluación las cuales correspondieron a `dataset_Canada` o a `dataset_sdn` dependiendo del caso. Como se muestra en el código 3.7 también se llevó a cabo un proceso de normalización a las variables con el fin de que todas estuvieran en una misma escala.

Código 3.7: Configuración de datos para entrenamiento y evaluación

```
1 self.X_train = scaler.fit_transform(X_train)
2 self.y_train = y_train
3 self.X_test = scaler.transform(X_eval_data)
4 self.y_test = y_eval
```

Además, se realizó una evaluación del modelo en el que se entrenó con el conjunto `dataset_sdn` y para evaluarlo se llevó a cabo una mezcla del tráfico benigno del `dataset_Canada` y el tráfico malicioso del `dataset_sdn`. Para ello lo que se realizó fue una división de ambos conjuntos en tráfico benigno y malicioso para después obtener cuatro porciones, posteriormente se concatenó el tráfico benigno del `dataset_Canada` y el tráfico malicioso del `dataset_sdn` como se puede apreciar en el siguiente fragmento de código 3.8.

Después a dicha concatenación se le realizó una mezcla aleatoria con el fin de asegurar una distribución equilibrada y evitar que el modelo se viese influenciado por un orden secuencial de los datos. El objetivo de esta prueba era observar como el modelo se adaptaba a las variaciones del tráfico legítimo sin modificar el tráfico malicioso.

Código 3.8: Configuración de datos para entrenamiento y evaluación

```
1 X_total = pd.concat([X_benign_Canada, X_malignant_sdn], axis=0, ignore_index=True)
2 y_total = pd.concat([y_benign_Canada, y_malignant_sdn], axis=0, ignore_index=True)
```

3.5. Conclusión

A lo largo de este trabajo se ha abordado con profundidad procesos como la depuración o comparación de conjuntos de datos destinados al entrenamiento de modelos de detección de ataques en red. Estas operaciones han demostrado la importancia de la equivalencia entre estructuras de diferentes *datasets* así como el preprocesamiento de los mismos.

Durante el desarrollo de este trabajo se llevó a cabo cambios en la estructura de los datos con el fin de mejorar el rendimiento de del modelo y la adaptabilidad de los conjuntos para poder realizar una evaluación cruzada.

Un punto relevante fue la identificación de once características comunes entre los dos conjuntos, esto fue fundamental para posteriormente llevar a cabo la evaluación cruzada entre los datasets y llegar a conclusiones sobre su rendimiento entre distintos contextos. Esto es especialmente importante en el mundo de la ciberseguridad, puesto que los sistemas deben ser capaces de adaptarse a diferentes tipos de ataques y entornos.

4

Pruebas y resultados

4.1. Introducción

Una vez completado la etapa de procesamiento y definido los modelos, se dio paso a la fase de pruebas y analizar los resultados.

En esta sección se presentarán los resultados obtenidos tanto para el `dataset_sdn` como para el `dataset_Canada` así como la evaluación cruzada entre el `dataset_sdn` y el `dataset_Canada`. También se mostrarán distintas tablas y gráficas con el fin de visualizar los resultados alcanzados.

4.2. Resultados de las predicciones

4.2.1. Descripción y configuración de los modelos evaluados

Como paso previo a analizar los resultados obtenidos en la detección del tráfico malicioso, es importante presentar brevemente las configuraciones implementadas para cada modelo de aprendizaje automático. El objetivo fue comparar su precisión y eficacia.

En el caso del modelo de regresión logística, se pusieron a prueba varios algoritmos de optimización llamados *solvers*, entre estos se encuentran, *newton-cg*, *lbfgs*, *liblinear*, *sag* y *saga*. Cada uno de estos métodos emplea diferentes técnicas para calcular los coeficientes adecuados, lo que determina la velocidad de convergencia y la precisión en el resultado final del modelo.

Para las máquinas de vectores de soporte (SVM), se probaron distintas funciones *kernel*, las cuales se caracterizan por realizar una transformación de los datos para encontrar un hiperplano óptimo de separación. Se evaluaron los *kernels linear*, *poly* (polinómico), *rbf* (*radial basis function*) y *sigmoid*.

En cuanto al *random forest*, se aplicó una optimización de hiperparámetros mediante la función *GridSearchCV*. Esto permitió ajustar variables como el número de árboles, el criterio de división o la profundidad máxima.

Por último, para el modelo *gradient boosting* se establecieron parámetros ya predefinidos como 100 árboles, una tase de aprendizaje de 0.1 y una profundidad máxima de 3 niveles.

4.2.2. Predicciones de los porcentajes de ataque con el dataset_sdn

En esta sección se presentan los resultados obtenidos en la clasificación de tráfico malicioso en los distintos conjuntos de datos.

Como se mencionó anteriormente, para el `dataset_sdn` fue necesario completar y modificar el código y los resultados cambiaron con respecto al publicado en *GitHub* [14], por lo que las predicciones obtenidas fueron las siguientes:

En cuanto a la regresión logística obtuvo un 74.13 % de media, entre los *solvers* destaca el *lbfgs* como el más efectivo. La máquina de vectores de soporte alcanzó un 97.0 % de media, con mejor rendimiento el kernel el *rbf*. El modelo *random forest* logró un 99.99 % siendo el que mayor porcentaje obtuvo. Y por último el *gradient boosting* consiguió un 99.25 %. A continuación se muestra la figura 4.1 donde se ven dichos porcentajes.

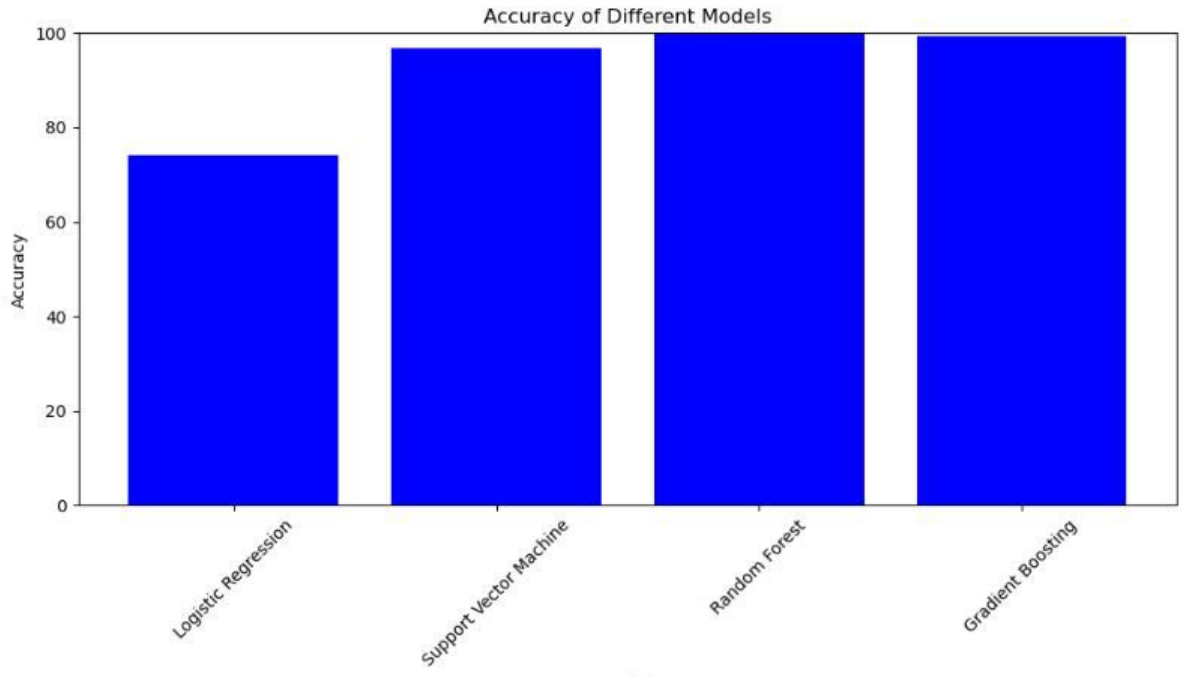

Figura 4.1: Diagrama de barras dataset_sdn.

El modelo de regresión logística destaca por su bajo tiempo de procesamiento, unos dos segundos, lo que permite realizar predicción de manera ágil y eficaz. Por otro lado, la *SVM* requiere mayor tiempo ejecución, en torno a 413 segundos, aunque el que mayor tiempo de ejecución presenta es el *random forest* con unos 771 segundos. Finalmente, el *gradient boosting* si bien es cierto que presenta un tiempo mayor que la regresión logística no alcanza el tiempo de ejecución del *random forest* o *SVM*

con 12 segundos.

Además, se extrajeron las siguientes métricas, como se observa en las siguientes tablas 4.1, 4.2, 4.3 y 4.4 para cada uno de los modelos de aprendizaje, en donde cada uno de los elementos representa:

- ***Recall* (Sensibilidad)**: Indica qué proporción de los casos reales positivos el modelo logró identificar adecuadamente.
- ***Accuracy* (Exactitud)**: Mide el total de predicciones correctas que hizo el modelo, considerando todas las clases.
- ***F1-Score***: Media que combina la precisión y la sensibilidad.

Las matrices de confusión, representadas en las figuras 4.2, 4.3, 4.4 y 4.5 son una herramienta fundamental para comprender como se comporta un modelo de clasificación, ya que muestra los diferentes tipos de clasificaciones obtenidas y están formadas por:

- **Verdaderos positivos (TP)**: Muestras que el modelo clasificó correctamente como positivas.
- **Falsos positivos (FP)**: Muestras que el modelo clasificó como positivas, pero que en realidad no lo son.
- **Verdaderos negativos (TN)**: Muestras negativas correctamente clasificadas.
- **Falsos negativos (FN)**: Muestras que el modelo clasificó como negativas, pero que en realidad son positivas.

Métrica	Valor
Accuracy	0.74
Recall	0.74
F1-Score	0.74
TP (Verdaderos Positivos)	9012
FP (Falsos Positivos)	3297
TN (Verdaderos Negativos)	9033
FN (Falsos Negativos)	3001

Tabla 4.1: Tabla con las métricas de evaluación del modelo regresión logística para el `dataset_sdn`.

Métrica	Valor
Accuracy	0.97
Recall	0.97
F1-Score	0.97
TP (Verdaderos Positivos)	11783
FP (Falsos Positivos)	562
TN (Verdaderos Negativos)	230
FN (Falsos Negativos)	41118

Tabla 4.2: Tabla con las métricas de evaluación del modelo de support vector machine para el `dataset_sdn`.

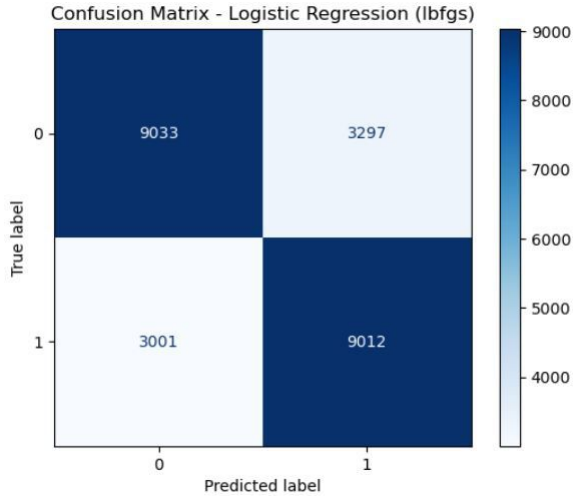

Figura 4.2: Matriz de confusión del modelo regresión logística para el `dataset_sdn`.

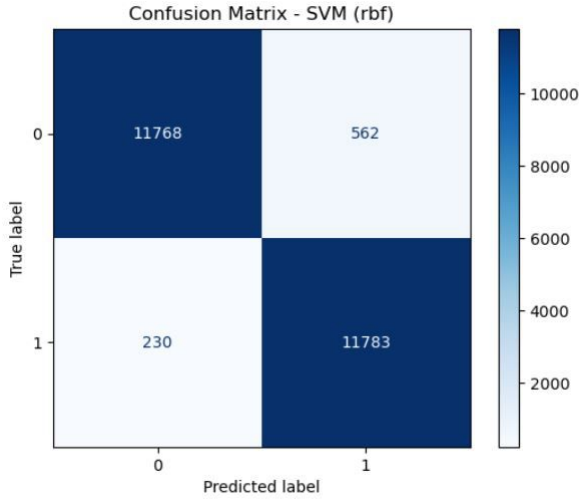

Figura 4.3: Matriz de confusión del modelo support vector machine para el `dataset_sdn`.

Métrica	**Valor**
Accuracy	1.00
Recall	1.00
F1-Score	1.00
TP (Verdaderos Positivos)	12012
FP (Falsos Positivos)	3
TN (Verdaderos Negativos)	0
FN (Falsos Negativos)	41118

Tabla 4.3: Tabla con las métricas de evaluación del modelo random forest para el `dataset_sdn`.

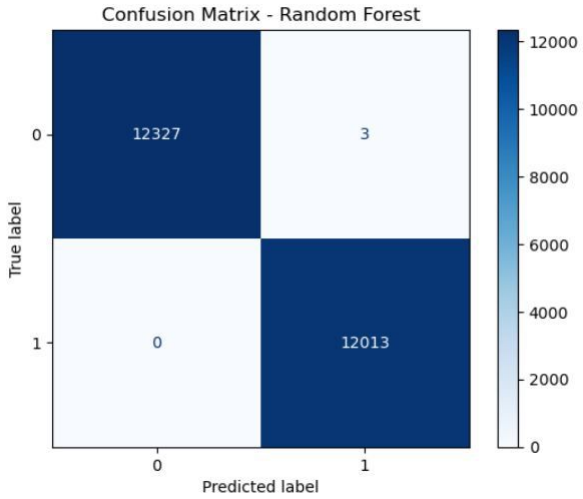

Figura 4.4: Matriz de confusión del modelo random forest para el `dataset_sdn`.

Métrica	**Valor**
Accuracy	0.99
Recall	0.99
F1-Score	0.99
TP (Verdaderos Positivos)	11990
FP (Falsos Positivos)	159
TN (Verdaderos Negativos)	12171
FN (Falsos Negativos)	23

Tabla 4.4: Tabla con las métricas de evaluación del modelo gradient boosting para el `dataset_sdn`.

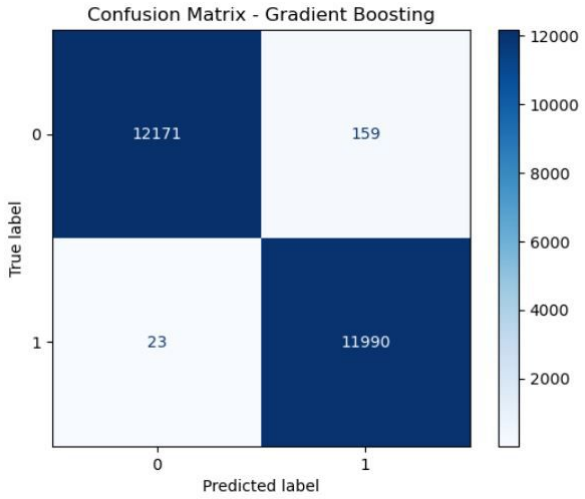

Figura 4.5: Matriz de confusión del modelo gradient boosting para el `dataset_sdn`.

Como se mencionó anteriormente, también se ejecutó el modelo con el `dataset_sdn` pero esta vez solo con las once columnas mantenidas en el `dataset_Canada`. A pesar de reducir el número de características los resultados apenas sufrieron cambios como se puede apreciar en la figura 4.6 siendo muy similares a los del `dataset_sdn` sin eliminación de columnas. Y los tiempos de ejecución también fueron casi idénticos.

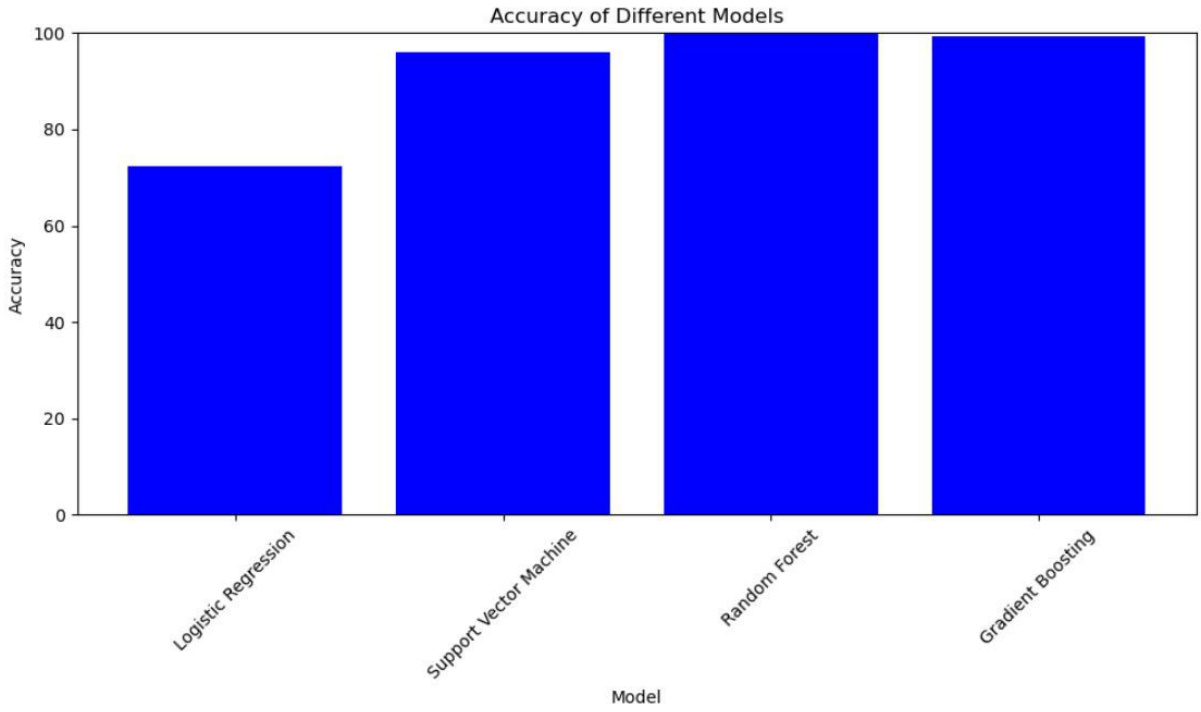

Figura 4.6: Diagrama de barras dataset_sdn nuevas características.

4.2.3. Predicciones de los porcentajes de ataque con el dataset_Canada

En este apartado se presentan los resultados obtenidos tras aplicar los distintos modelos de clasificación al `dataset_canada` como se aprecia en la figura 4.7.

En general se obtuvieron resultados mejores que para el anterior dataset, sin embargo, ninguno de los modelos alcanzó el 100 % como si lo hizo el anterior. La regresión logística alcanzó un 97.57 %, destacando el *solver liblinear*. La SVM, logró un 98 % con mejor rendimiento de *kernel poly*. Por otro lado, el *random forest* consiguió un 99.61 %. Y finalmente, el *gradient boosting* obtuvo un 99.15 %.

En cuanto a los tiempos de ejecución no hay mucha diferencia entre el dataset_sdn y dataset_Canada. Como se expuso anteriormente, la regresión logística sigue destacando por su velocidad, con unos cuatro segundos, mientras que la SVM y el *Random forest* son los que más tiempo de ejecución demandan, siendo 192 y 718 segundos respectivamente, destacando este último como el mayor. Y en un punto intermedio se encuentra el *gradient boosting*, con siete segundos.

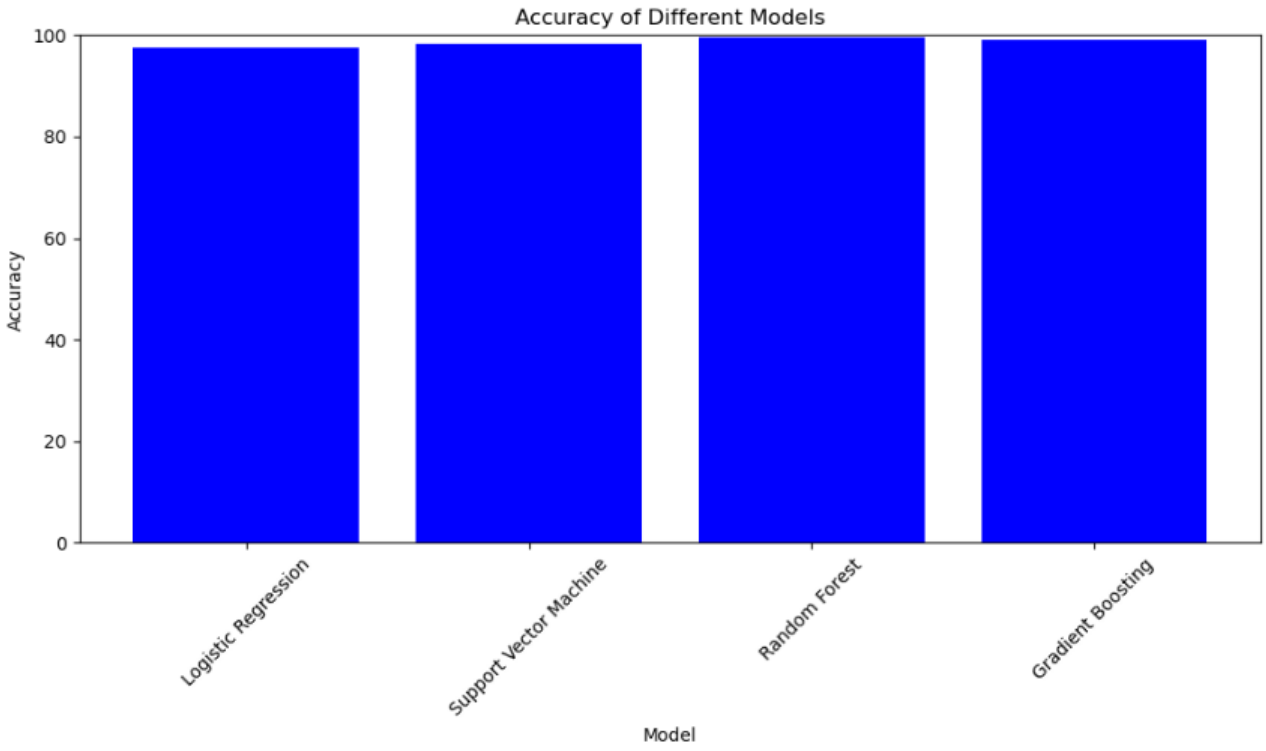

Figura 4.7: Diagrama de barras dataset_canada.

A continuación, se muestran las gráficas 4.8, 4.9, 4.10 y 4.11 correspondientes a las matrices de confusión y las tablas 4.5, 4.6, 4.7 y 4.8 con las métricas para evaluar el rendimiento de los modelos en el `dataset_canada`, muy parecidas a las obtenidas para el `dataset_sdn` con una diferencia notable en el modelo de regresión logística mucho mayor para el `dataset_canada`.

Métrica	**Valor**
Accuracy	0.98
Recall	0.98
F1-Score	0.98
TP (Verdaderos Positivos)	14593
FP (Falsos Positivos)	451
TN (Verdaderos Negativos)	14468
FN (Falsos Negativos)	274

Tabla 4.5: Tabla con las métricas de evaluación del modelo regresión logística para el `dataset_canada`.

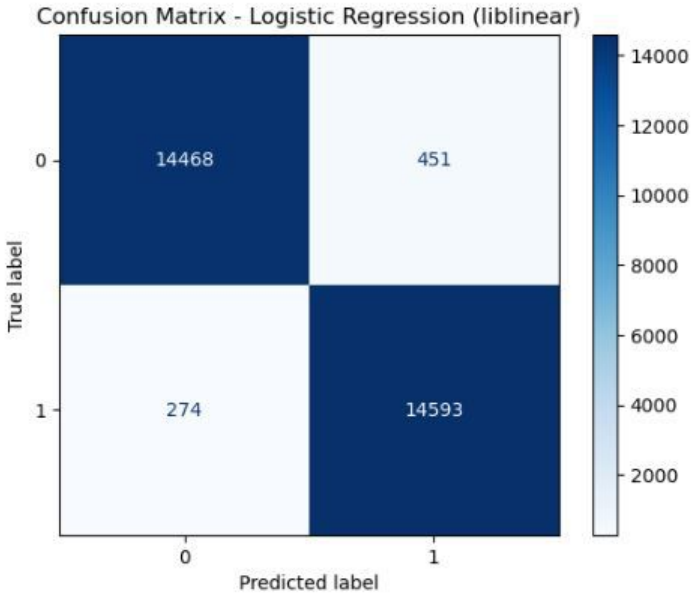

Figura 4.8: Matriz de confusión del modelo regresión logística para el `dataset_canada`.

Métrica	Valor
Accuracy	0.98
Recall	0.98
F1-Score	0.98
TP (Verdaderos Positivos)	14802
FP (Falsos Positivos)	431
TN (Verdaderos Negativos)	14488
FN (Falsos Negativos)	65

Tabla 4.6: Tabla con las métricas de evaluación del modelo support vector machine para el `dataset_canada`.

Métrica	Valor
Accuracy	1.00
Recall	1.00
F1-Score	1.00
TP (Verdaderos Positivos)	14849
FP (Falsos Positivos)	102
TN (Verdaderos Negativos)	14817
FN (Falsos Negativos)	18

Tabla 4.7: Tabla con las métricas de evaluación del modelo random forest para el `dataset_canada`.

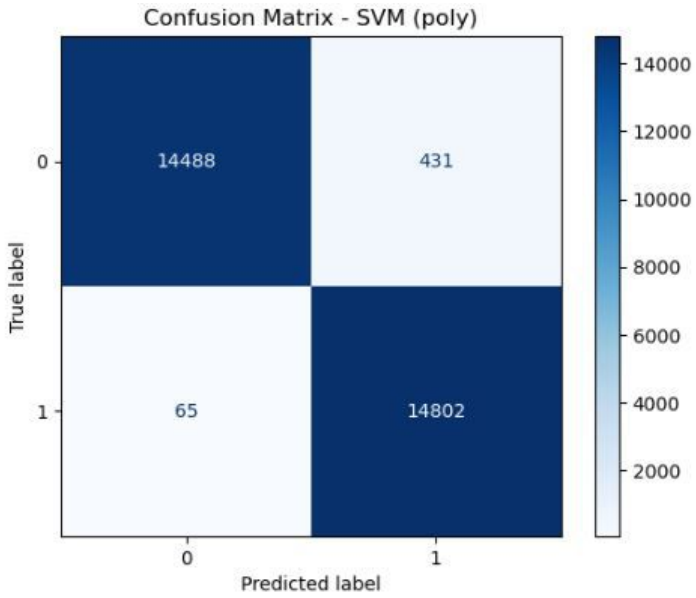

Figura 4.9: Matriz de confusión del modelo support vector machine para el `dataset_canada`.

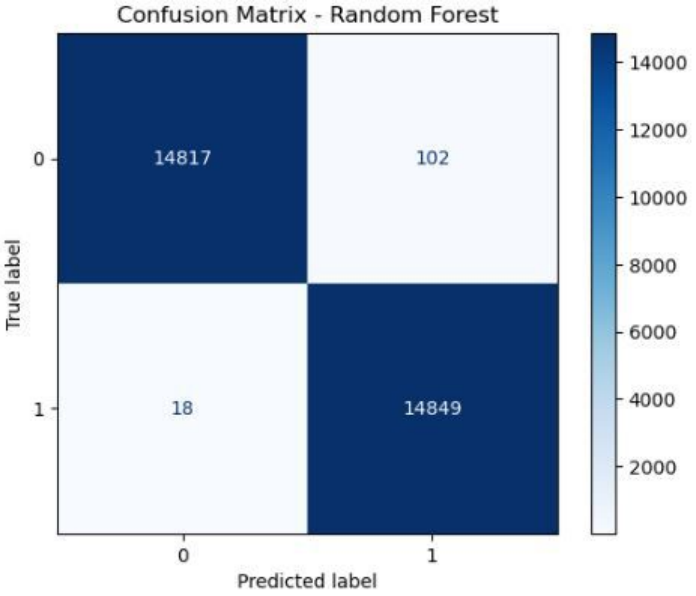

Figura 4.10: Matriz de confusión del modelo random forest para el `dataset_canada`.

Métrica	Valor
Accuracy	0.99
Recall	0.99
F1-Score	0.99
TP (Verdaderos Positivos)	14829
FP (Falsos Positivos)	213
TN (Verdaderos Negativos)	14706
FN (Falsos Negativos)	38

Tabla 4.8: Tabla con las métricas de evaluación del modelo gradient boosting para el `dataset_canada`.

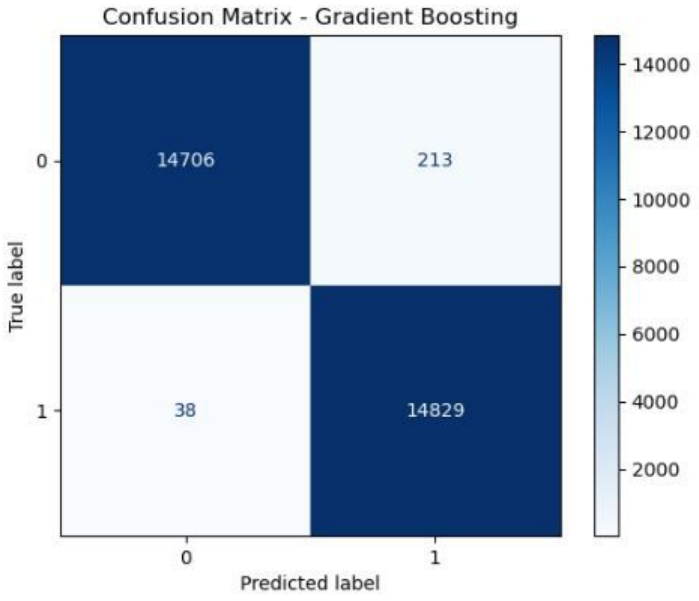

Figura 4.11: Matriz de confusión del modelo gradient boosting para el `dataset_canada`.

4.2.4. Predicciones de los porcentajes de ataque entre el dataset_sdn y el dataset_Canada

En esta sección se muestran los resultados obtenidos tras entrenar el modelo con el `dataset_Canada` y evaluarlo con el `dataset_sdn`.

Como se observa en la figura 4.12 la regresión logística llegó a un 56.04 % siendo el que mayor porcentaje tiene de los cuatro modelos, destacando el *solver saga*. El SVM alcanzó un 51 % con mejor precisión el *kernel linear*. El *random forest* fue el que mejor porcentaje obtuvo con un 39 % seguido del *gradient boosting* con un 48.13 %.

En cuanto a los tiempos de ejecución la regresión logística invirtió seis segundos, mientras que el SVM unos 485 segundos y el *random forest* 1108 segundos. Por último el *gradient boosting* consumió 10 segundos.

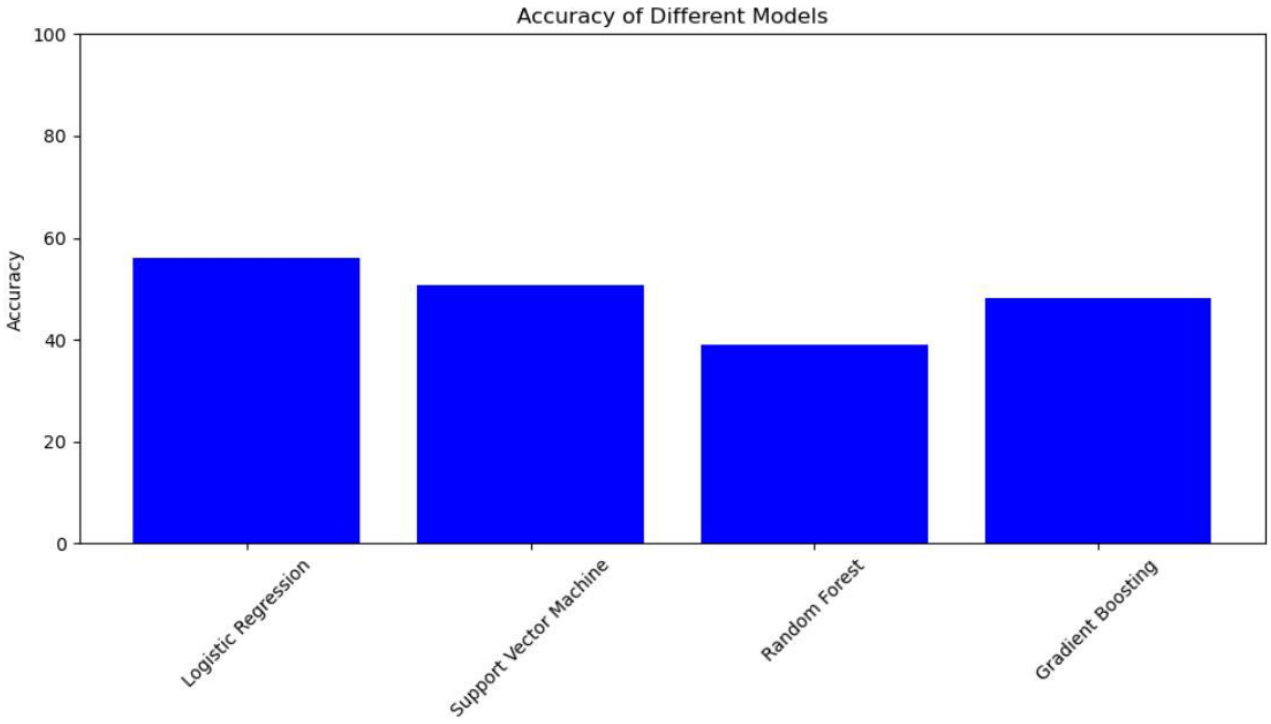

Figura 4.12: Diagrama de barras evaluación cruzada entrenada con dataset_Canada y evaluado con dataset_sdn

También se exponen las matrices de confusión de los diferentes modelos en las figuras 4.13, 4.14, 4.15 y 4.16.

Para el modelo entrenado con el `dataset_sdn` y evaluado con el `dataset_Canada` estos fueron los resultados. De acuerdo a lo que se aprecia en la figura 4.17 la regresión logística y el SVM fueron los modelos que alcanzaron mayor rendimiento con un 62.47 % y un 62 % respectivamente y destacando como *solver newton-cg* y como *kernel poly*. Tanto el *random forest* como el *gradient*

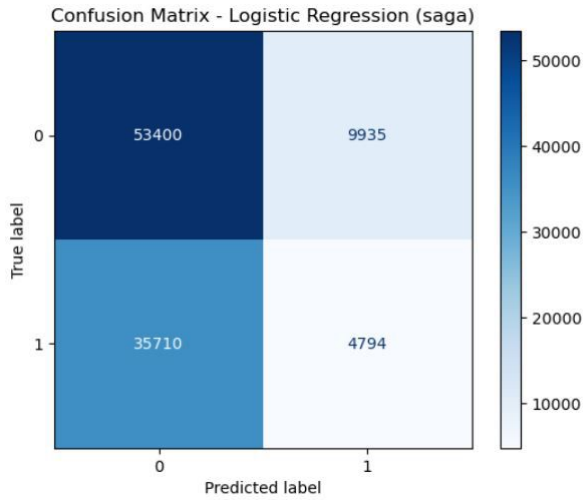

Figura 4.13: Matriz de confusión del modelo regresión logística entrenado con `dataset_canada` y evaluado con `dataset_sdn`.

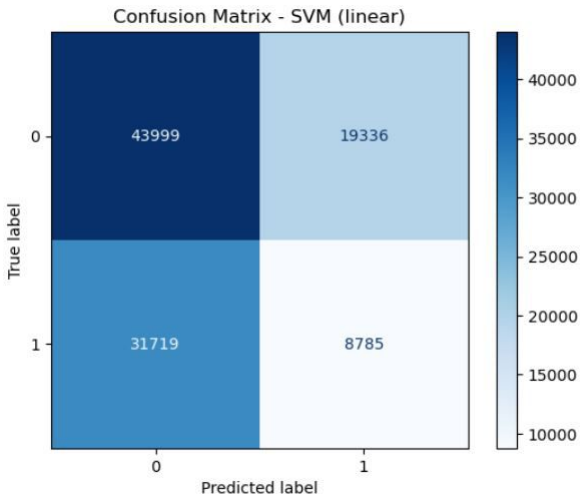

Figura 4.14: Matriz de confusión del modelo support vector machine entrenado con `dataset_canada` y evaluado con `dataset_sdn`.

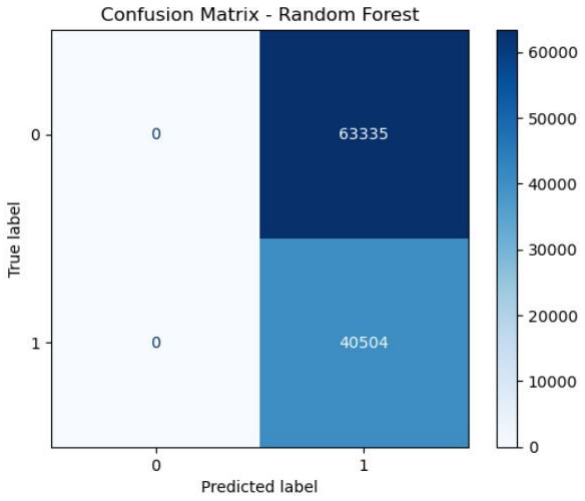

Figura 4.15: Matriz de confusión del modelo random forest entrenado con `dataset_canada` y evaluado con `dataset_sdn`.

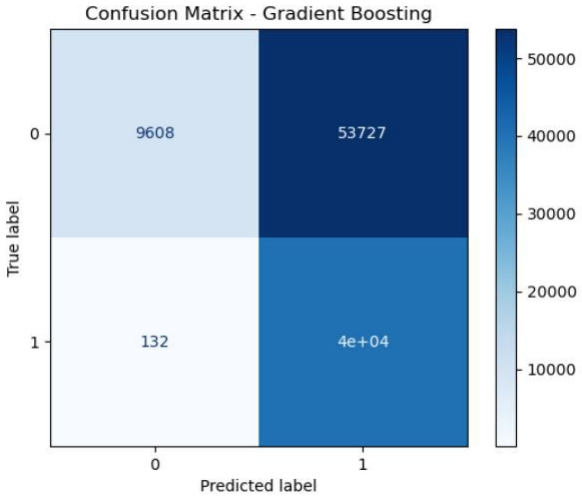

Figura 4.16: Matriz de confusión del modelo gradient boosting entrenado con `dataset_canada` y evaluado con `dataset_sdn`.

boosting fueron los que obtuvieron peor porcentaje ambos con un 50.1 %.

Los tiempos de ejecución continuaron siendo elevados, para la regresión logística alcanzó casi los tres segundos, el SVM tardó 1263 segundos. También el *random forest* llegó a los 966 segundos y por último el *gradient boosting* invirtió 12 segundos.

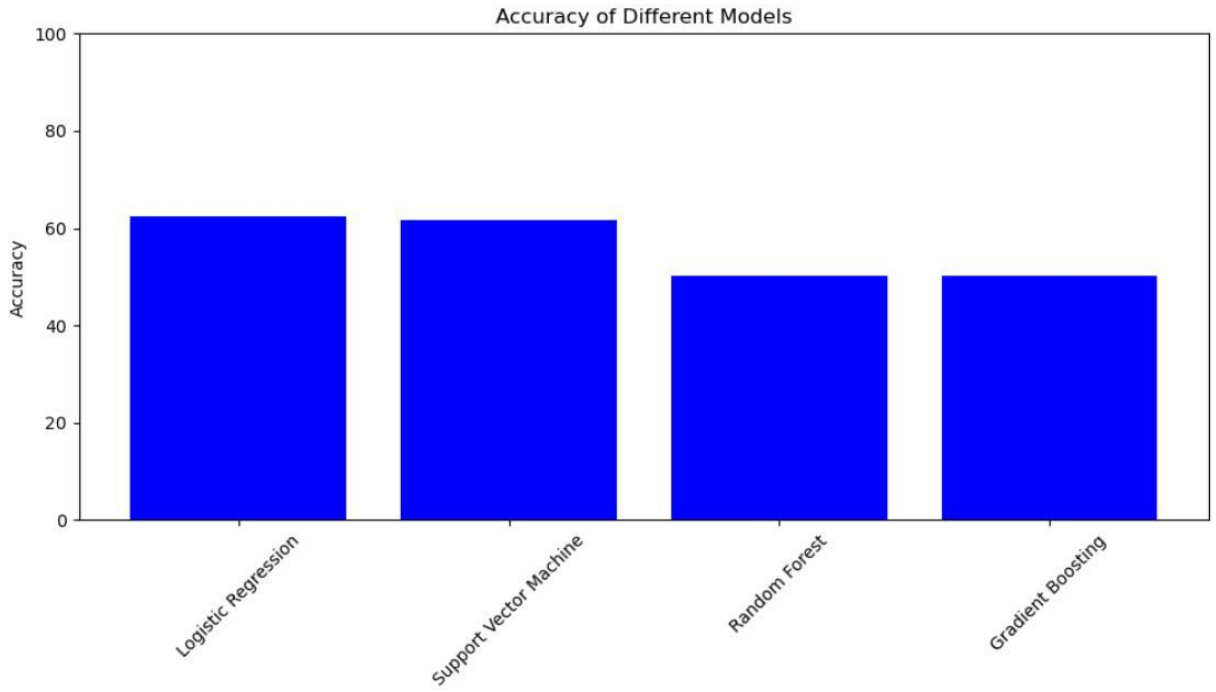

Figura 4.17: Diagrama de barras evaluación cruzada entrenada con dataset_sdn y evaluado con dataset_Canada

Para la evaluación en donde el modelo era entrenado con el `dataset_sdn` y evaluado con una combinación de tráfico benigno de `dataset_Canada` y tráfico malicioso de `dataset_sdn` se dan los siguientes resultados que se muestran en la figura 4.18.

En cuanto a la regresión logística alcanzó un 46.92 % y tuvo un tiempo de procesamiento de tres segundos el *solver* que destaco fue *liblinear*, el SVM invirtió un tiempo de 1674 segundos siendo el mejor *kernel* el *sigmoid* y obtuvo el mayor porcentaje de los cuatro modelos con un 75 %. El *random forest* llegó a los 1000 segundos y logró un 45.07 % siendo el peor junto al *gradient boosting* con un 44.56 % y un tiempo de 12 segundos.

Se observa que para ambos hay una considerable disminución en el rendimiento del modelo al ser evaluado con un *dataset* distinto al del entrenamiento, comparándolo con los resultados obtenidos al entrenar y evaluar con el mismo modelo. Además, los tiempos de ejecución aumentaron hasta llegar a ser casi el doble o el triple en algunos casos con respecto al de las anteriores pruebas, aunque esto tendría sentido, puesto que ahora el conjunto de datos es mucho mayor.

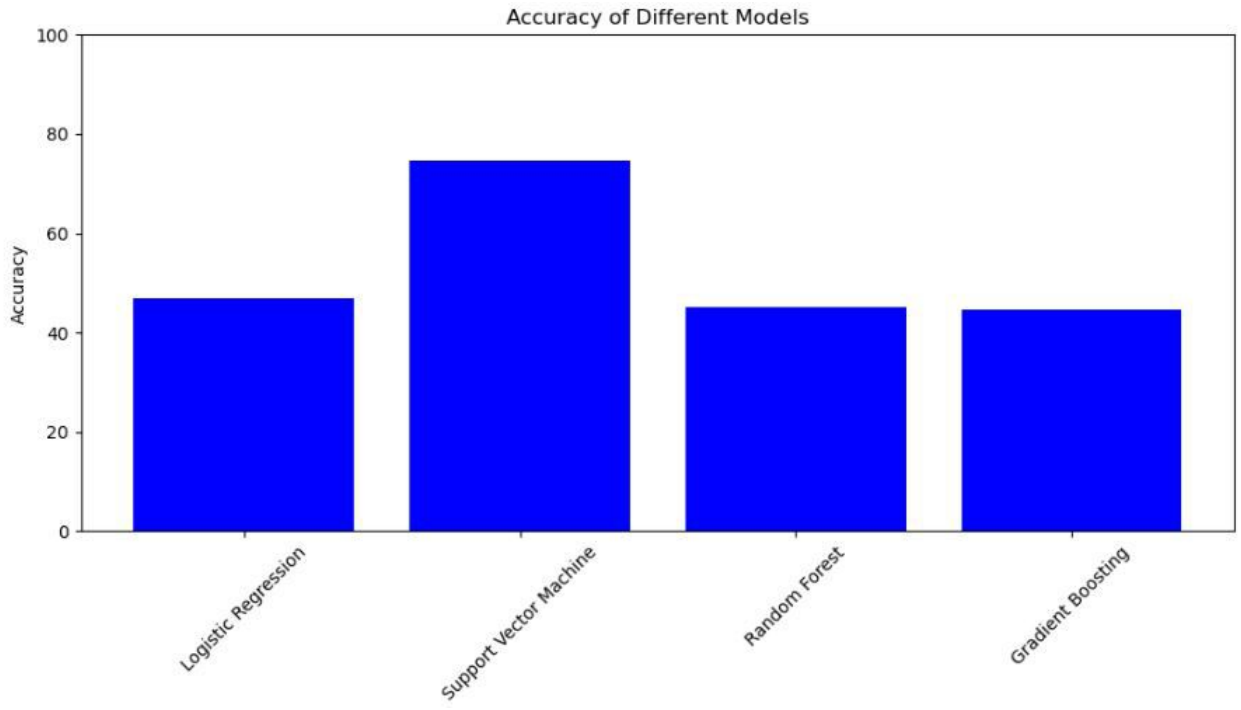

Figura 4.18: Diagrama de barras evaluación cruzada entrenada con dataset_sdn y evaluado con dataset_sdn y dataset_Canada.

4.3. Conclusión

Cada modelo evaluado presenta distintas características adecuadas según las necesidades del problema a resolver. Por ejemplo, la regresión logística es la mejor opción si lo que se busca es un modelo con un bajo coste computacional y que muestre resultados de manera simple pero eficaz. En contraste, Random Forest es un modelo con un alto coste computacional pero con una precisión muy alta, parecido al caso de la *SVM*. En cambio, *gradient boosting* ofrece una precisión muy alta y un tiempo de ejecución no tan alto como el *random forest* o *SVM*, por lo que puede ser una muy buena opción si lo que se busca es un equilibrio entre eficacia y rendimiento.

También se ha de tener en cuenta que un aspecto fundamental en la detección de ataque DDoS es el tiempo de respuesta del modelo, donde es clave detectar rápidamente si el tráfico es malicioso o benigno. Por esto mismo, no basta solo con alcanzar el mayor porcentaje, sino también conseguir un modelo lo suficientemente rápido como para detectar amenazas de manera ágil.

Los resultados obtenidos tras entrenar y evaluar un modelo con el mismo *dataset* son altamente positivos. Sin embargo, como se observó al entrenar y evaluar los modelos con distintos *datasets* los resultados no son adecuados. Esto puede deberse a diversos factores: en primer lugar aunque puedan parecer similares, existen diferencias estructurales entre ellos. Por otro lado, cada conjunto de datos presenta distintos tipos de ataques, por un lado, el `dataset_sdn` por ejemplo incluye ataques que utilizan protocolos como *ICMP* o *TCP*, mientras que `dataset_Canada` *NTP* O *DNS*. Estas diferencias

afectan a los modelos a la hora de identificar patrones comunes y predecir correctamente. También un modelo puede predecir con eficacia un conjunto de entrenamiento, pero cuando se evalúa sobre otro conjunto que no presenta el mismo patrón su rendimiento disminuye.

Un claro ejemplo de las diferencias entre ambos *datasets* se refleja en los árboles de decisión generados tras la ejecución del algoritmo *random forest*. Aunque ambos modelos hayan sido entrenados con las mismas once características, la profundidad y la estructura de ambos árboles varía completamente. Para el `dataset_Canada` se observa que en la figura 4.19 las características dominantes son *pktperflow*, *packetins* o *pairflow*. Por otro lado, para el `dataset_sdn` si bien es cierto que comparte la variable *pktperflow* como una de las más importantes, destacan otras como *dur* o *byteperflow* como se puede ver en la figura 4.20.

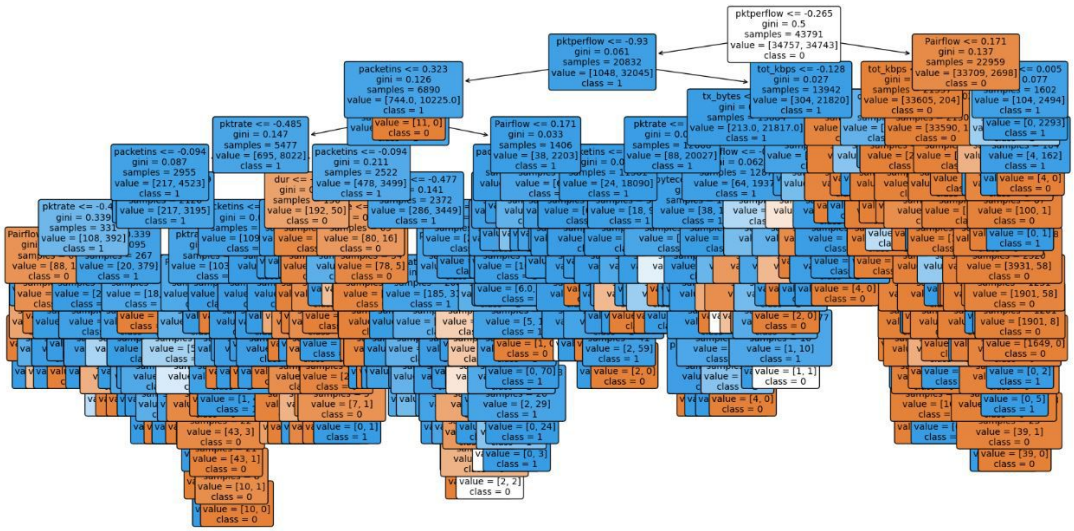

Figura 4.19: Árbol de decisión del dataset_canada

Esto demuestra que los patrones y estructuras del tráfico de red depende del entorno y del tipo de ataque que se este llevando a cabo. Puesto que a pesar de estar utilizando el mismo algoritmo, el árbol que se genera se adapta a las variables específicas de cada conjunto, lo que demuestra que los modelos son sensibles a las variaciones de contenido y composición de los *datasets*

Además, como se mencionó anteriormente se llevó a cabo una prueba en la que se entrenó el modelo con el `dataset_sdn` y se evaluó utilizando una combinación de tráfico benigno del `dataset_Canada` y tráfico malicioso `dataset_sdn`, en donde los resultados obtenidos fueron realmente bajos. Esto evidencia que no solo el tráfico malicioso, sino también el benigno varía de estructura y patrón dependiendo del entorno y la fuente de captura.

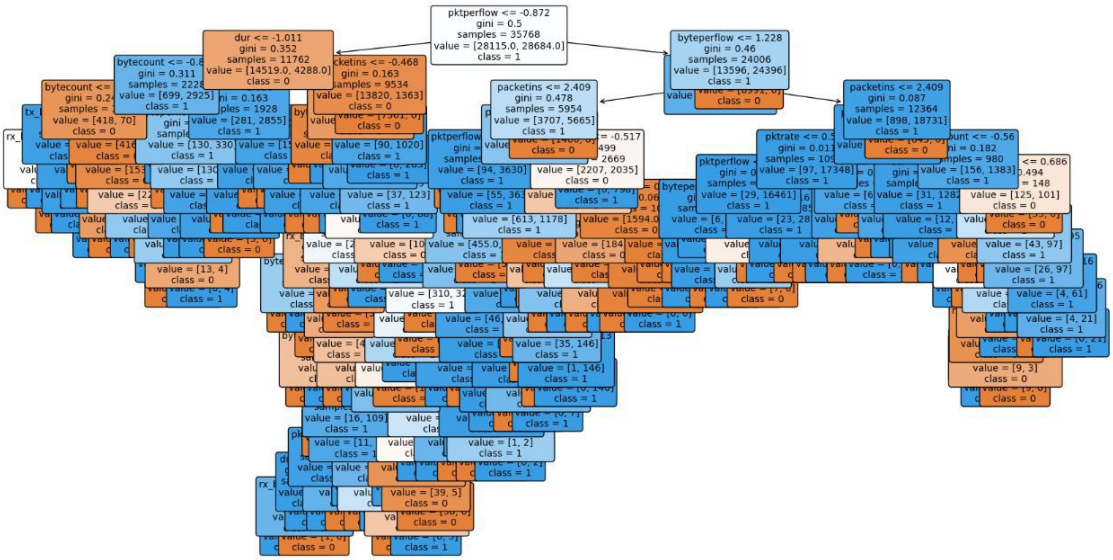

Figura 4.20: Árbol de decisión del dataset_sdn

5

Conclusiones y trabajo futuro

5.1. Introducción

En esta última sección se exponen las conclusiones a raíz del trabajo presentado, resaltando los resultados más importantes y se plantean nuevas vías de investigación con el fin de ampliar y mejorar el trabajo ya realizado.

5.2. Conclusiones

Tras finalizar este trabajo, se ha podido constatar la eficacia de uso de modelos de aprendizaje automático a la hora de detectar ataques DDoS, alcanzando en algunos casos una muy alta precisión como es el caso de los algoritmos *random forest* o *gradient boosting* llegando a los 99 % de exactitud. Los resultados presentan que la regresión logística es el modelo con menor porcentaje, pero con menor tiempo de procesamiento, mientras *random forest* y *gradient boosting* fueron los algoritmos más eficaces siendo este último el que más destaco por su rapidez y por último el SVM logró buenos resultado pero un muy alto tiempo de procesamiento.

Si bien esto es cierto mientras se entrenaba y evaluaba con el mismo conjunto de datos al realizar una evaluación cruzada los rendimientos bajaron considerablemente lo que demuestra una dependencia de la estructura y tipos de ataques. Además, se mostró la importancia del preprocesamiento y normalización de los datos antes de su uso en los modelos.

5.3. Trabajo futuro

En próximos trabajos se podría realizar lo siguiente:

- Llevar a cabo técnicas de aprendizaje no supervisado, para hallar patrones sin necesidad de un entrenamiento previo.
- Emplear nuevos *datasets* para evaluar la capacidad de los modelos a nuevas estructuras y entornos.

- Optimizar los recursos con el fin de reducir los tiempos de procesamiento en particular en *random forest* y *SVM*.
- Desarrollar una herramienta capaz de detectar ataques en tiempo real.
- Implementar modelos híbridos, en donde es utilicen diferentes modelos en donde se aprovechen las ventajas de cada uno.

Bibliografía

[1] Abhiram B. S., B. S. Sumanth, Kushal R, *DDos Detection Using Machine Learning Algorithms*. BNMIT Bangalore, 2023.

[2] Fortinet. Tipos de ciberataques: ataque ddos, ransomware y más. Visitado: 24.06.2025. [Online]. Available: https://www.fortinet.com/lat/resources/cyberglossary/types-of-cyber-attacks

[3] Wikipedia. Ataque de denegación de servicio. Visitado: 30.07.2024. [Online]. Available: https://es.wikipedia.org/wiki/Ataque_de_denegaci%C3%B3n_de_servicio

[4] Stackscale. Ataques ddos: tipos, protección y mitigación. Visitado: 24.06.2025. [Online]. Available: https://www.stackscale.com/es/blog/ataques-ddos/

[5] Wikipedia. Aprendizaje automático. Visitado: 30.07.2024. [Online]. Available: https://es.wikipedia.org/wiki/Aprendizaje_autom%C3%A1tico

[6] IBM. ¿qué es la regresión logística? Visitado: 24.06.2025. [Online]. Available: https://www.ibm.com/es-es/think/topics/logistic-regress

[7] Geekscoach. Regresión logística usando sklearn| logic regression. Visitado: 27.06.2025. [Online]. Available: https://geekscoach.medium.com/regresi%C3%B3n-log%C3%ADstica-usando-sklearn-logic-regression-2955e8668aaf

[8] Wikipedia. Máquina de vectores de soporte. Visitado: 24.06.2025. [Online]. Available: https://es.wikipedia.org/wiki/M%C3%A1quina_de_vectores_de_soporte

[9] Elena Campo León, *Introducción a las máquinas de vector soporte (SVM) en aprendizaje supervisado*. Universidad de Zaragoza, 2016.

[10] DataScientest. Random forest: Bosque aleatorio. definición y funcionamiento. Visitado: 24.06.2025. [Online]. Available: https://datascientest.com/es/random-forest-bosque-aleatorio-definicion-y-funcionamiento

[11] Darío Alarcón Hunter. Analítica predictiva 2: Regresión multivariable con random forest. Visitado: 27.06.2025. [Online]. Available: https://www.quality.cl/analitica-predictiva-2-regresion-multivariable-con-random-forest/

[12] Wikipedia. Gradient boosting. Visitado: 24.06.2025. [Online]. Available: https://es.wikipedia.org/wiki/Gradient_boosting

[13] hemashreekilari. Understanding gradient boosting. Visitado: 27.06.2025. [Online]. Available: https://medium.com/@hemashreekilari9/understanding-gradient-boosting-632939b98764

[14] Abhiram B.S. DDos-Detection-Using-Machine-Learning-Algorithms-Python. Visitado: 26.09.2024. [Online]. Available: https://github.com/Abhirambs-08/DDos-Detection-Using-Machine-Learning-Algorithms-Python/tree/main/Docs

[15] C. I. for Cybersecurity (CIC), “Ddos evaluation dataset (cic-ddos2019),” Visitado: 27.10.2024. [Online]. Available: https://www.unb.ca/cic/datasets/ddos-2019.html

Apéndices

A

GITHUB

Para probar el código usado durante el desarrollo del proyecto se ha habilitado un repositorio de GitHub `https://github.com/Aledan43/TFG_AMV`


UAM
UNIVERSIDAD AUTONOMA
DE MADRID