ESING

REVISTA INGENIEROS MILITARES
Número 17, Año 2022
ISSN 2215-7107 / Bogotá-Colombia

Implementación de Modelos de Aprendizaje Supervisado para la Detección de Tráfico Malicioso

Miguel Alfonso Negrete Romero[1], José Francisco Ortiz Morales[2], Jorge Gómez Gómez[3]

RESUMEN

sistemas ha aumentado las amenazas cibernéticas y el tráfico malicioso, motivando el desarrollo de algoritmos basados en Inteligencia Artificial para su detección. Este estudio investiga métodos avanzados de Aprendizaje Automático para mejorar la seguridad, enfocándose en la eficacia de diferentes modelos en la detección de amenazas como ataques de denegación de servicio e intrusiones. Se utilizan datasets como KDD Cup 99 y Network Traffic Data-Malicious Activity Detection para experimentar distintos algoritmos. El análisis detallado de datos y la evaluación de algoritmos destacan la viabilidad de estas técnicas para fortalecer la ciberseguridad.

PALABRAS CLAVE: Inteligencia artificial, malware, aprendizaje computacional, modelos de ML, detección de intrusos, aprendizaje supervisado.

ABSTRACT

The increasing interconnectedness of devices and systems has heightened cyber threats and malicious traffic, prompting the development of Artificial Intelligence-based algorithms for detection. This study investigates advanced Machine Learning methods to enhance security, focusing on the effectiveness of different models in detecting threats such as Denial of Service attacks and intrusions. Datasets like KDD Cup 99 and Network Traffic Data-Malicious Activity Detection are used to experiment with various algorithms. Detailed data analysis and algorithm evaluation highlight the viability of these techniques to strengthen cybersecurity.

KEYWORDS: Artificial intelligence, malware, machine learning, Machine learning models, Intrusion Detection System, supervised learning.

1 Pregrado en ingeniería de sistemas. Montería – Colombia. mnegreteromero00@correo.unicordoba.edu.co

2 Pregrado en ingeniería de sistemas. Montería – Colombia.

3 Departamento de Ingeniería de Sistemas y Telecomunicaciones, Facultad de Ingeniería, Universidad de Córdoba, jeliecergómez@correo.unicordoba.edu.co

Introducción

La creciente interconexión de dispositivos y sistemas ha traído consigo una proliferación de amenazas cibernéticas y tráfico malicioso en redes. La protección de la integridad y seguridad de sistemas críticos y datos sensibles se ha convertido en una prioridad apremiante.[1] En este contexto, la aplicación de la Inteligencia Artificial y técnicas de Aprendizaje Automático (ML) para la detección de tráfico malicioso emerge como una solución prometedora. Estos métodos algorítmicos juegan un papel crucial en la identificación y neutralización de actividades sospechosas en tiempo real.[2] Este estudio se centra en explorar el uso de técnicas de ML en la detección de tráfico malicioso. Se busca crear un modelo que pueda analizar diversos datos para identificar patrones y comportamientos anómalos, permitiendo así la detección temprana de posibles ataques de denegación de servicio (DoS), intrusiones no autorizadas y otras actividades maliciosas.

Además, se examinarán las vulnerabilidades comunes presentes en estas redes y cómo los métodos algorítmicos pueden ofrecer soluciones efectivas para mitigar estos riesgos. Aspectos como la autenticación robusta de usuarios, la confidencialidad de datos y la protección contra intrusiones serán discutidos en detalle.[3] El propósito de este estudio es proporcionar una base sólida para la implementación de medidas preventivas y correctivas efectivas en materia de seguridad cibernética. Al enfocarse en el uso de técnicas de ML y algoritmos para la detección de tráfico malicioso, se busca equipar a las organizaciones con las herramientas necesarias para fortalecer la seguridad de sus sistemas y proteger sus operaciones en el panorama actual. Varios modelos de aprendizaje supervisado han sido aplicados con éxito al problema de la detección de tráfico malicioso. Algunos de los modelos más comúnmente utilizados incluyen árboles de decisión, bosques aleatorios, máquinas de vectores de soporte (SVM), regresión logística y redes neuronales.

La implementación de modelos de aprendizaje supervisado para la detección de tráfico malicioso típicamente involucra la recolección y preprocesamiento de datos, la selección y ajuste de modelos, el entrenamiento y evaluación de los modelos, y su despliegue y monitoreo.[4] La aplicación de IA ofrece diversas ventajas dependiendo del método empleado en los sistemas de detección y prevención de intrusiones (IDPS), como el procesamiento de información en paralelo en redes neuronales artificiales, la adaptación al entorno y las preferencias del usuario en agentes inteligentes, la autoadaptibilidad y autoorganización en sistemas inmunológicos artificiales, la optimización con algoritmos genéticos y la interoperabilidad en lógica difusa. A pesar de sus ventajas, la implementación de IA en ciberseguridad también plantea desafíos. Las preocupaciones de privacidad y ética deben abordarse con cautela, y la confiabilidad de los algoritmos debe evaluarse constantemente.

La sinergia entre la inteligencia artificial (IA) y el malware crea una realidad extremadamente compleja y desafiante para las empresas. La IA, concebida para facilitar nuestras vidas, también la simplifica para los cibercriminales. Para las PYME, con recursos más modestos y limitados que las grandes corporaciones, estas amenazas representan un riesgo significativo, ya que a menudo son el objetivo de los cibercriminales.[5] Como se ha podido observar, hay una gran necesidad del uso de la inteligencia artificial para la detección del tráfico malicioso, por lo que es necesario desarrollar modelos para detectar estas amenazas y poder identificar que las está causando [13] ,12] ,[11] ,[10] ,[9] ,[8] ,[7] ,[6].

Fundamentos teóricos

Aprendizaje supervisado. El aprendizaje supervisado es una técnica de machine learning en la cual un algoritmo aprende de un conjunto de datos etiquetados para realizar predicciones o clasificaciones sobre datos no etiquetados. Este proceso implica dos etapas principales: la fase de entrenamiento y la fase de prueba. Durante la fase de entrenamiento, el algoritmo se entrena con un conjunto de datos que incluye tanto las entradas como las salidas deseadas. En la fase de prueba, el algoritmo se evalúa con nuevos datos para medir su precisión y capacidad de generalización. [14] El objetivo del aprendizaje supervisado es construir un modelo que pueda predecir con precisión la etiqueta de salida para cualquier entrada dada, basándose en los ejemplos que ha visto durante el entrenamiento. Esto se logra minimizando una función de error o pérdida, que mide la discrepancia entre las predicciones del modelo y las etiquetas reales en el conjunto de datos de entrenamiento [9]. Los algoritmos de aprendizaje supervisado pueden clasificarse en dos categorías principales: regresión y clasificación. La regresión se utiliza para predecir valores continuos, mientras que la clasificación se utiliza para predecir categorías discretas. [15]

IDS. Un sistema de detección de intrusos (IDS, por sus siglas en inglés) es una solución de seguridad que monitorea las actividades de la red o del sistema en busca de comportamientos maliciosos o violaciones de políticas. Su objetivo principal es identificar y alertar sobre posibles intrusiones o ataques que puedan comprometer la integridad, confidencialidad o disponibilidad de los recursos informáticos. [16] Los IDS pueden clasificarse en dos categorías principales: sistemas basados en host (HIDS) y sistemas basados en red (NIDS). Los HIDS se enfocan en la supervisión y análisis de actividades dentro de un solo host, incluyendo logs del sistema, archivos de auditoría y otras fuentes de datos locales. Por otro lado, los NIDS supervisan el tráfico de red en tiempo real, analizando paquetes para detectar patrones sospechosos o conocidos de ataque. [17]

El funcionamiento de un IDS puede basarse en firmas (detección de patrones conocidos) o en anomalías (identificación de comportamientos inusuales). Los sistemas basados en firmas comparan el tráfico de red o los eventos del sistema contra una base de datos de firmas predefinidas, mientras que los sistemas basados en anomalías establecen una línea base de comportamiento normal y detectan desviaciones significativas de esta línea base. [18] A pesar de su utilidad, los IDS enfrentan varios desafíos, incluyendo la alta tasa de falsos positivos y negativos, la capacidad de manejar grandes volúmenes de datos en tiempo real y la necesidad de actualizaciones constantes para mantener la efectividad contra nuevas amenazas. Además, los IDS por sí solos no pueden prevenir intrusiones, sino que actúan como una capa adicional de defensa que debe integrarse con otros mecanismos de seguridad. [19]

Malware. Se refiere a cualquier programa o código diseñado con la intención de dañar, interrumpir, robar o causar un impacto negativo en los datos, hosts o redes. Los tipos de malware incluyen virus, gusanos, troyanos, ransomware, spyware y adware, entre otros. Cada tipo de malware tiene un mecanismo de funcionamiento y un objetivo específico, pero todos comparten la característica común de realizar actividades no autorizadas y perjudiciales en los sistemas donde se ejecutan [20].

Virus: Un virus es un tipo de malware que se adjunta a un programa o archivo legítimo y se propaga a otros archivos y sistemas cuando el programa infectado se ejecuta. Los virus pueden corromper o borrar datos, causar interrupciones en el sistema y realizar otras acciones dañinas. [21]

Gusanos: Los gusanos son programas maliciosos que se replican a sí mismos y se propagan automáticamente a través de redes. A diferencia de los virus, los gusanos no requieren de la intervención humana para propagarse y pueden causar daños significativos al consumir recursos de red y sobrecargar los sistemas. [22]

Troyanos: Los troyanos se presentan como software legítimo o inofensivo para engañar a los usuarios y hacer que los instalen. Una vez dentro del sistema, los troyanos pueden crear puertas traseras, robar información, registrar teclas y realizar otras actividades maliciosas sin el conocimiento del usuario. [23]

Ransomware: El ransomware cifra los datos del usuario y exige un rescate para proporcionar la clave de descifrado. Este tipo de malware ha ganado notoriedad en los últimos años debido a varios ataques de alto perfil que han paralizado empresas e infraestructuras críticas. [24]

Spyware: El spyware monitorea las actividades del usuario sin su consentimiento, recopilando información como datos de navegación, credenciales de inicio de sesión y otra información personal. Esta información se envía a terceros que pueden utilizarla con fines malintencionados. [25]

Adware: El adware es software que presenta anuncios no deseados al usuario, a menudo dentro de un navegador web. Aunque no siempre es malicioso, el adware puede degradar el rendimiento del sistema y comprometer la privacidad del usuario. [26]

El malware se distribuye de diversas maneras, incluyendo correos electrónicos de phishing, descargas de software de sitios web no seguros, exploits de vulnerabilidades de software y unidades USB infectadas. Los atacantes utilizan técnicas de ingeniería social para engañar a los usuarios y hacer que ejecuten el malware en sus sistemas. [27]

El impacto del malware puede variar desde molestias menores hasta pérdidas financieras significativas y daño a la reputación. Las organizaciones y usuarios pueden protegerse contra el malware mediante el uso de software antivirus, la aplicación de parches de seguridad, la educación sobre la ciberseguridad y la implementación de políticas de seguridad robustas. [28]

Protocolos de red. Conjuntos de reglas y convenciones que permiten la comunicación entre dispositivos en una red. Estos protocolos definen cómo se deben formatear, transmitir y recibir los datos para garantizar una comunicación eficiente y precisa. Los protocolos de red abarcan una amplia gama de funciones, desde la transmisión de datos a través de medios físicos hasta la gestión de conexiones y el enrutamiento de paquetes. [29]

Protocolo de Control de Transmisión/Protocolo de Internet (TCP/IP):

El conjunto de protocolos TCP/IP es el estándar fundamental para las comunicaciones en Internet. TCP se encarga de establecer y mantener conexiones fiables, garantizando la entrega ordenada y completa de los datos, mientras que IP se encarga del direccionamiento y enrutamiento de paquetes a través de la red. [30]

Protocolo de Configuración Dinámica de Host (DHCP):

DHCP es un protocolo de red que permite a los dispositivos obtener automáticamente una dirección IP y otros parámetros de configuración de red necesarios para conectarse a la red. Esto simplifica la administración de redes al eliminar la necesidad de asignar manualmente direcciones IP. [31]

P*rotocolo de Resolución de Direcciones (ARP):*

ARP es un protocolo utilizado para mapear una dirección IP a una dirección MAC en una red de área local. Esto es esencial para que los dispositivos en una LAN se comuniquen entre sí utilizando direcciones MAC físicas. [32]

Protocolo Simple de Transferencia de Correo (SMTP):

SMTP es un protocolo utilizado para el envío de correos electrónicos a través de redes IP. Define cómo los mensajes de correo electrónico se envían desde un cliente a un servidor de correo y entre servidores de correo. [33]

Protocolo de Transferencia de Archivos (FTP):

FTP es un protocolo estándar utilizado para transferir archivos entre un cliente y un servidor en una red. Proporciona funciones para subir, descargar y gestionar archivos en servidores remotos. [34]

Protocolo de Información de Enrutamiento (RIP):

RIP es un protocolo de enrutamiento utilizado en redes de área local y extensa para determinar la mejor ruta para enviar datos. Utiliza un algoritmo de vector-distancia y actualiza periódicamente las tablas de enrutamiento en cada enrutador. [35]

Sus funciones principales son:

Direccionamiento: Proporcionan mecanismos para identificar dispositivos en la red de manera única mediante direcciones IP o MAC.

Enrutamiento: Determinan las rutas óptimas para el tránsito de datos entre nodos de la red.

Control de Flujo y Congestión: Gestionan el tráfico de datos para evitar la congestión de la red y asegurar una transmisión fluida.

Establecimiento de Conexiones: Facilitan la creación y finalización de conexiones entre dispositivos, asegurando una comunicación fiable.

Estado del arte

El uso de la inteligencia artificial (IA) para la detección de tráfico de redes maliciosas ha ganado prominencia debido a su eficacia en enfrentar desafíos derivados del tráfico encriptado, que dificulta la detección y clasificación de actividades maliciosas [36]. Esta tecnología se ha investigado extensamente, como lo demuestran estudios previos. Se revisó el uso de machine learning para analizar tráfico, destacando su aplicabilidad en la detección de registros de cuentas falsas y transacciones fraudulentas.[37] Se exploraron las posibilidades de la IA en ciberseguridad, utilizando el marco de ciberseguridad del NIST para evaluar su efectividad [37]. Por otro lado, se aplicó la IA para analizar el tráfico en entornos CLOUD, mejorando la predicción de ataques potenciales [38]. Se propuso un framework flexible para la detección de tráfico malicioso encriptado, compatible con diversos modelos y conjuntos de datos. [39]

En el ámbito de la detección de malware, se exploró el uso de redes neuronales profundas, como DNN y CNN, por su alto rendimiento y facilidad de aprendizaje[40]. El análisis de tráfico encriptado ha sido crucial, como enfatizó [41], al proponer la combinación de características numéricas independientes del protocolo con características específicas del tráfico encriptado para alimentar tanto algoritmos tradicionales como de aprendizaje profundo. [42] revisaron métodos de IA, incluyendo SVM y redes neuronales, para la detección de ataques en Internet de las Cosas (IoT).

Para mejorar la detección de intrusos en sistemas ciber-físicos, [43] evaluaron estrategias de detección de anomalías y sistemas híbridos, mientras que [44] introdujeron un modelo jerárquico de redes neuronales profundas para la detección eficiente de tráfico malicioso. Además, [45] utilizaron matrices de decisión de información para mejorar la selección de características en la detección de malware. En cuanto a la seguridad en dispositivos IoT. [47] ,[46] propusieron soluciones para la protección de redes masivas de IoT y sistemas de control industrial, respectivamente, utilizando métodos como KNN y RF. También se ha explorado la detección de ataques en sistemas de salud inteligentes mediante técnicas de IA y aprendizaje profundo [48].

En el campo de la detección de intrusiones basadas en redes, [49] desarrollaron un sistema híbrido que combina IA con técnicas de detección de intrusiones conocidas, mientras que [50] se enfocaron en la detección de ataques SlowDoS sobre tráfico encriptado usando IA en tiempo real. Además, [51] destacaron la importancia de la explicabilidad en la detección de intrusiones mediante modelos de IA basados en redes neuronales convolucionales.

Por último, [52] revisaron los avances en la detección de intrusiones en redes abiertas, resaltando desafíos como los altos falsos positivos y la falta de datasets compatibles. Estos estudios subrayan la continua evolución y la necesidad de modelos robustos de IA para enfrentar las complejidades de la ciberseguridad moderna.

Metodología

En esta investigación se busca desarrollar y proponer un algoritmo basado en IA para la detección de tráfico malicioso. La investigación de distintos métodos algoritmos usados para mejorar nuestra seguridad es un área que incluso hoy se sigue investigando, con los recientes avances en cuanto a inteligencia artificial, se busca que estos métodos algorítmicos puedan ser más eficaces al momento de detectar amenazas.

También se estudia como poder usar técnicas de ML para poder crear un algoritmo de fase inicial capaz de detectar amenazas a un nivel promedio; Los protocolos se analizarían a través de distintas pruebas, preferiblemente con datos reales para poder obtener datos variados para después compararlos y realizar un análisis y sacar conclusiones de que tan efectivo es el modelo.

Las distintas fases que el proyecto tendrá son:

Recolección de Datos:

En esta fase inicial, se busca recopilar una amplia variedad de datos de tráfico malicioso provenientes de diversas fuentes. Estos datos se obtendrán de fuentes virtuales y organizaciones que compartan los datasets de forma gratuita

Análisis de Datos:

Una vez recopilados los datos, se procederá a analizarlos en profundidad. Se examinarán los patrones estadísticos, tendencias y características que estos tengan. Este análisis permitirá identificar parámetros clave, como la frecuencia de los ataques, el nivel de amenaza para los dispositivos, las medidas de defensa tomadas y posibles correlaciones entre distintos tipos de ataques. Esta etapa incluirá:

-Análisis detallado de los datos de tráfico malicioso.

-Identificación de patrones y tendencias en los ataques.

-Evaluación de medidas de defensa utilizadas por los fabricantes.

Experimentación con distintas técnicas de ML:

Una vez comprendidas las amenazas y los datos recopilados, se procederá a realizar experimentos con diferentes algoritmos para comprobar su efectividad en cuanto a la detección de tráfico malicioso. Estos algoritmos se probarán en diferentes escenarios para evaluar su efectividad en la detección y respuesta a los ataques. Esta etapa incluirá:

-Pruebas y experimentos con varios algoritmos de ML para la detección de tráfico malicioso.

-Evaluación de la efectividad de distintos algoritmos de ML en escenarios reales.

Análisis de los algoritmos:

Una vez finalizadas las pruebas, se realizará un análisis exhaustivo de los algoritmos utilizados. Se evaluarán los beneficios y limitaciones de cada algoritmo, incluyendo su tiempo de detección de amenazas, susceptibilidad a ataques, compatibilidad con sistemas y actualización de protocolos de seguridad. Esta etapa incluirá:

-Análisis de los resultados de las pruebas de los algoritmos

-Evaluación de los beneficios y limitaciones de cada algoritmo.

-Identificación de algoritmos más efectivos y recomendaciones para su uso.

Desarrollo del Prototipo del modelo:

Finalmente, se utilizará toda la información recopilada y los resultados de las pruebas para proponer un modelo basado en los algoritmos evaluados para la detección de tráfico malicioso. Este prototipo se diseñará teniendo en cuenta los protocolos más efectivos identificados en las etapas anteriores. El objetivo es crear un modelo que sea capaz de detectar y responder eficazmente a distintas amenazas que pueden afectar.

Resultados

Se hizo uso de dos datasets para la prueba de los distintos modelos de aprendizaje, uno de ellos es kdd cup 54] 99], Este dataset simula intrusiones en un entorno de red militar, utilizando datos de volcado TCP/IP. Se creó un entorno para adquirir datos crudos de conexiones TCP/IP en una LAN típica de la Fuerza Aérea de los EE.UU. La LAN estaba configurada como un entorno real y fue expuesta a múltiples tipos de ataques. Una conexión es una secuencia de paquetes TCP que comienza y termina en un intervalo de tiempo determinado, durante el cual fluye datos desde y hacia una dirección IP origen hacia una dirección IP destino bajo un protocolo bien definido. Además, cada conexión está etiquetada como normal o como un tipo específico de ataque. Cada registro de conexión consta de aproximadamente 100 bytes.

Para cada conexión TCP/IP, se obtienen 41 características cuantitativas y cualitativas a partir de datos normales y de ataques (3 características cualitativas y 38 características cuantitativas). La variable de clase tiene dos categorías:

Normal

Anómalo

Preprocesamiento de datos

Uso de modelos.

Desde este punto se empiezan a aplicar los modelos para la validación de cada uno el objetivo de esto es evaluar cuál modelo es el que mejor evalúa el dataset proporcionado para así darnos una idea de cuál de estos modelos podemos aplicar en el proyecto.

Modelo de Regresión Lineal.

Primero se inicia evaluando el modelo de regresión lineal, el cual lo realizaremos con ayuda del lenguaje de python, por lo tanto, comenzaremos haciendo algunas importaciones de librerías que utilizaremos para el proceso. Más adelante lo que hacemos es asignar a una variable el nombre de data_train, la cual se encargará de almacenar el archivo csv de nuestro dataset. A la par de esto creamos otra variable llamada df_train la cual va a leer nuestro dataset que se encuentra en en la variable data_train y lo almacenará en forma de dataframe en nuestra variable df_train. Como se refleja en la imagen 1.1.1

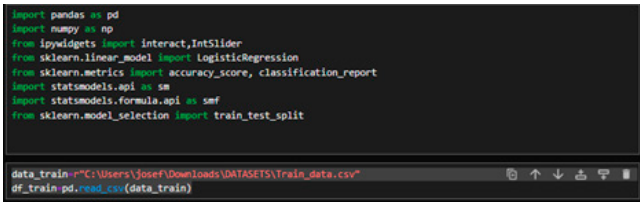
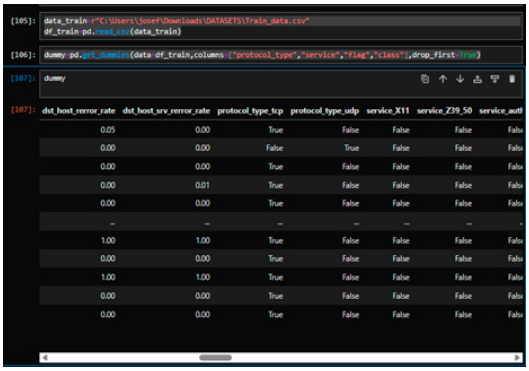


Figura 1.1.1. Importación y almacenamiento de dataset. Fuente: autores del proyecto

Continuando con lo relacionado al modelo como ya se explicó anteriormente en el apartado de pre procesamiento encontramos que algunos de los datos eran de tipo categórico, cosa que dificulta la ejecución del modelo de Regresión Lineal, por lo cual tuvimos que tomar manos a la obra y realizar algunos procesamientos sobre esto lo que realizamos ya se explica en el apartado de pre procesamiento de manera más concisa pero se dará una breve explicación en esta parte, haciendo uso de la codificación one-hot dummy la cual es de (1/0), vamos a crear dos nuevas columnas para cada una de columnas que poseen datos categóricos, esto se hace con el objetivo de crear nuevas columnas para cada uno de los datos que encontrábamos en cada una de estas columnas por ejemplo, en la columna protocol_type, teniamos datos de tipo tcp y upd, al aplicar la codificación one-hot, tendremos dos nuevas columnas llamadas protocol_type_tcp y protocol_type_udp, en caso dado que el dato al que estén asociados sea de tipo udp se colocará un 1 o un true en la columna udp y un 0 o un false en la columna tcp, de esta manera tendremos datos numericos en lugar de categoricos en nuestro dataframe, como podemos observar en la imagen 1.1.2.

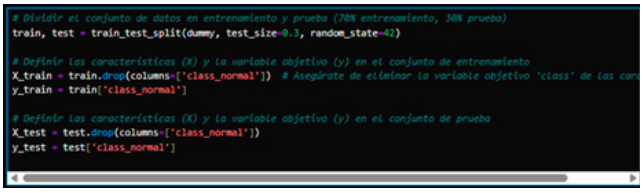


```
[105]: data_train=r"C:\Users\josef\Downloads\DATASETS\Train_data.csv"
       df_train=pd.read_csv(data_train)

[106]: dummy=pd.get_dummies(data=df_train,columns=["protocol_type","service","flag","class"],drop_first=True)

[107]: dummy

[107]:
```

dst_host_rerror_rate	dst_host_srv_rerror_rate	protocol_type_tcp	protocol_type_udp	service_X11	service_Z39_50	service_aut
0.05	0.00	True	False	False	False	Fals
0.00	0.00	False	True	False	False	Fals
0.00	0.00	True	False	False	False	Fals
0.00	0.01	True	False	False	False	Fals
0.00	0.00	True	False	False	False	Fals
...	...	...	...	...	...	...
1.00	1.00	True	False	False	False	Fals
0.00	0.00	True	False	False	False	Fals
1.00	1.00	True	False	False	False	Fals
0.00	0.00	True	False	False	False	Fals
0.00	0.00	True	False	False	False	Fals

Figura 1.1.2. Conversión de columnas categóricas a numéricas. Fuente: autores del proyecto

En la siguiente parte lo que realizaremos vendría siendo ya la primera parte de la aplicación del modelo.



```
# Dividir el conjunto de datos en entrenamiento y prueba (70% entrenamiento, 30% prueba)
train, test = train_test_split(dummy, test_size=0.3, random_state=42)

# Definir las características (X) y la variable objetivo (y) en el conjunto de entrenamiento
X_train = train.drop(columns=['class_normal'])  # Asegúrate de eliminar la variable objetivo 'class' de las car
y_train = train['class_normal']

# Definir las características (X) y la variable objetivo (y) en el conjunto de prueba
X_test = test.drop(columns=['class_normal'])
y_test = test['class_normal']
```

Figura 1.1.3. División del dataset y creación del test. Fuente: autores del proyecto

Lo primero que hacemos es tomar el conjunto de datos de test este lo tomamos del mismo data train, para eso hacemos uso de la librería train_test_split, la cual nos va ayudar a dividir los datos en 70/30, 70 para entrenamiento y 30 para prueba, además de eso indicamos con dummy que queremos usar la codificación one_hot para el dataframe que se ha generado. Siguiendo esto eliminamos la columna class normal en x_train ya que esa es la variable que se quiere predecir y en y_train lo que se hace es definir la variable objetivo como class_normal. El mismo proceso se hace con los datos de test.

En este paso ya empezamos a entrenar el modelo como podemos observar en la imagen 1.1.3.

```
y_train = y_train.astype(float)
X_train = X_train.astype(float)

y_test = y_test.astype(float)
X_test = X_test.astype(float)
```

Figura 1.1.4. Estandarización de los datos. Fuente: autores del proyecto

En la imagen 1.1.4 podemos observar cómo pasamos los datos que se encuentran en las variables x e y a tipo de dato float, esto se hace con el objetivo de estandarizar los datos en un único tipo de dato en el dataset, ya que además los datos de este tipo son más precisos que otros tipos de datos como los enteros.

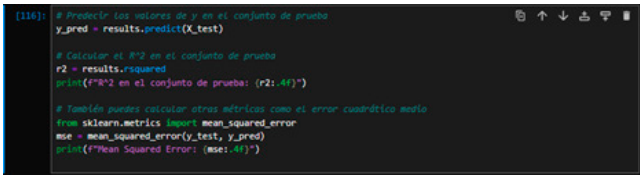
```python
# Agregar una constante a las características (intercepto)
X_train = sm.add_constant(X_train)
X_test = sm.add_constant(X_test)

# Crear el modelo de regresión lineal
model = sm.OLS(y_train, X_train)

# Ajustar el modelo
results = model.fit()

# Resumen del modelo
print(results.summary())
```

Figura 1.1.5. Creación del modelo de Regresión Lineal. Fuente: autores del proyecto

En la primera parte del código que vemos en la imagen 1.1.5 podemos observar cómo hacemos uso de la librería statsmodels para agregar una columna que va a representar el intercepto del modelo de regresión lineal, esto se hace para las x tanto de test como de train. El intercepto de estas x representará la variable y de ambas variables, al usar constantes podremos ajustar una línea de regresión que no podría pasar por el origen. Siguiendo con eso, hacemos uso de la función OLS de la librería statsmodels, especificando que haremos uso de las variables x_train e y_train, creamos una variable results que se encargará de almacenar el ajuste del modelo de datos de entrenamiento usando el método fit(). Este proceso lo que hace es calcular los coeficientes de regresión que minimizan la suma de cuadrados de los errores entre los valores predichos y los reales. Y por último se imprime el modelo de regresión ajustado, como se ve en la imagen 1.1.6.

```
                            OLS Regression Results
==============================================================================
Dep. Variable:           class_normal   R-squared:                       0.879
Model:                            OLS   Adj. R-squared:                  0.878
Method:                 Least Squares   F-statistic:                     1142.
Date:                Tue, 23 Apr 2024   Prob (F-statistic):               0.00
Time:                        14:44:09   Log-Likelihood:                 5830.7
No. Observations:               17634   AIC:                        -1.144e+04
Df Residuals:                   17522   BIC:                        -1.057e+04
Df Model:                         111
Covariance Type:            nonrobust
==============================================================================
                                coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const                         -0.0161      0.076     -0.212      0.832      -0.165       0.133
duration                    5.308e-06   6.01e-07      8.838      0.000    4.13e-06    6.49e-06
src_bytes                  -1.596e-09   4.82e-10     -3.312      0.001   -2.54e-09   -6.52e-10
dst_bytes                  -1.114e-07   1.89e-08     -5.898      0.000   -1.48e-07   -7.44e-08
land                           0.4991      0.125      3.990      0.000       0.254       0.744
wrong_fragment                -0.2029      0.006    -34.184      0.000      -0.215      -0.191
urgent                        -0.9910      0.182     -5.449      0.000      -1.347      -0.634
hot                           -0.0246      0.001    -21.058      0.000      -0.027      -0.022
num_failed_logins             -0.0163      0.036     -0.456      0.648      -0.086       0.054
logged_in                     -0.1949      0.010    -18.958      0.000      -0.215      -0.175
num_compromised               -0.0309      0.003    -10.747      0.000      -0.037      -0.025
root_shell                    -0.0478      0.048     -0.991      0.322      -0.142       0.047
su_attempted                   0.1992      0.049      4.085      0.000       0.104       0.295
num_root                       0.0276      0.003     10.567      0.000       0.022       0.033
num_file_creations             0.0046      0.002      2.036      0.042       0.000       0.009
num_shells                     0.1253      0.062      2.009      0.045       0.003       0.248
num_access_files               0.0056      0.018      0.315      0.752      -0.029       0.040
num_outbound_cmds          -6.707e-15   3.48e-16    -19.295      0.000   -7.39e-15   -6.03e-15
is_host_login              -5.83e-15   8.83e-16     -6.600      0.000   -7.56e-15   -4.1e-15
is_guest_login                 0.4184      0.034     12.479      0.000       0.353       0.484
count                         -0.0002   2.81e-05     -7.054      0.000      -0.000      -0.000
srv_count                     -0.0003   4.16e-05     -7.561      0.000      -0.000      -0.000
serror_rate                    0.0966      0.033      2.956      0.003       0.033       0.161
srv_serror_rate               -0.3416      0.034    -10.024      0.000      -0.408      -0.275
rerror_rate                   -0.0132      0.033     -0.398      0.691      -0.078       0.052
srv_rerror_rate               -0.6029      0.039    -15.654      0.000      -0.678      -0.527
same_srv_rate                  0.3353      0.012     28.822      0.000       0.313       0.358
diff_srv_rate                  0.0652      0.011      5.852      0.000       0.043       0.087
srv_diff_host_rate            -0.0349      0.007     -5.316      0.000      -0.048      -0.022
dst_host_count                -0.0004   2.08e-05    -16.875      0.000      -0.000      -0.000
dst_host_srv_count             0.0008   3.64e-05     23.059      0.000       0.001       0.001
dst_host_same_srv_rate        -0.1412      0.011    -12.736      0.000      -0.163      -0.119
dst_host_diff_srv_rate        -0.1834      0.013    -14.130      0.000      -0.209      -0.158
dst_host_same_src_port_rate   -0.1395      0.008    -16.782      0.000      -0.156      -0.123
dst_host_srv_diff_host_rate   -0.1443      0.019     -7.681      0.000      -0.181      -0.107
dst_host_serror_rate          -0.0734      0.019     -3.777      0.000      -0.111      -0.035
dst_host_srv_serror_rate       0.0224      0.024      0.926      0.354      -0.025       0.070
dst_host_rerror_rate          -0.1294      0.014     -9.114      0.000      -0.157      -0.102
```

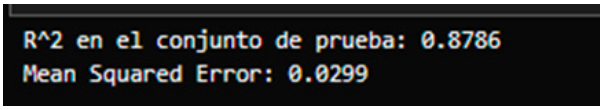
Figura 1.1.6. Modelo de regresión lineal ajustado. Fuente: autores del proyecto



```
[116]: # Predecir los valores de y en el conjunto de prueba
       y_pred = results.predict(X_test)

       # Calcular el R^2 en el conjunto de prueba
       r2 = results.rsquared
       print(f"R^2 en el conjunto de prueba: {r2:.4f}")

       # También puedes calcular otras métricas como el error cuadrático medio
       from sklearn.metrics import mean_squared_error
       mse = mean_squared_error(y_test, y_pred)
       print(f"Mean Squared Error: {mse:.4f}")
```

Figura 1.1.7. Evaluación del modelo de regresión lineal. Fuente: autores del proyecto

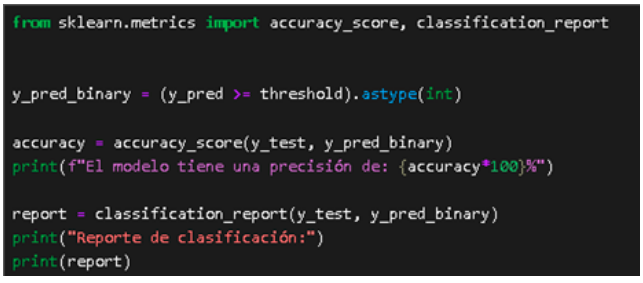
Como podemos observar en la imagen 1.1.7, utilizamos los coeficientes del modelo ajustado que se encuentran en almacenados en la variable y_pred para las observaciones en el conjunto de pruebas x_test. Después de eso lo que hacemos es calcular el r squared en el conjunto de pruebas con ayuda .rsquared. Y ya para obtener lo referente al error cuadrático hacemos uso de la librería skylar.metrics, para importar la función mean_squared_error, con la cual calculamos el error entre los valores de y test y los valores de y_pred que tiene los datos del modelo ajustado. Y al final imprimimos esos datos con un print. Estás métricas nos ayudarán a cuantificar que tan bien se está adaptando el modelo a los datos y que tan precisas son sus predicciones.



```
R^2 en el conjunto de prueba: 0.8786
Mean Squared Error: 0.0299
```

Figura 1.1.8. Regresión lineal resultados del uso del modelo. Fuente: autores del proyecto

Como podemos observar en la imagen 1.8 los resultados obtenidos de R squared son de cerca del 87%, lo que indica que el modelo tiene un buen ajuste a los datos y a las variables que se encuentran en el dataframe. Con respecto al MSE o Mean Squared Error, obtenemos un valor de 0.0299, este valor lo que nos indica es el promedio de los errores cuadráticos de los valores reales y los predichos por el modelo, un valor bajo como este indica que el modelo trabaja correctamente.



```
from sklearn.metrics import accuracy_score, classification_report


y_pred_binary = (y_pred >= threshold).astype(int)

accuracy = accuracy_score(y_test, y_pred_binary)
print(f"El modelo tiene una precisión de: {accuracy*100}%")

report = classification_report(y_test, y_pred_binary)
print("Reporte de clasificación:")
print(report)
```

Figura 1.1.9. Regresión lineal evaluación usando accuracy score y classification report. Fuente: autores del proyecto

Para usar estos modelos de evaluación del modelo para medir su precisión se usarán las librerías accuracy score y classification report, convertimos los datos de y_pred a binarios para poder evaluar de forma correcta el modelo y pasamos como parámetros en ambos casos "y_test", que es el dataset sin la clase y "y_pred_binary", donde están las predicciones hechas por el modelo.

```
El modelo tiene una precisión de: 96.9833289229955%
Reporte de clasificación:
              precision    recall  f1-score   support

         0.0       0.98      0.96      0.97      3516
         1.0       0.96      0.98      0.97      4042

    accuracy                           0.97      7558
   macro avg       0.97      0.97      0.97      7558
weighted avg       0.97      0.97      0.97      7558
```

Figura 1.1.10. Resultados del modelo usando accuracy score y classification report.
Fuente: autores del proyecto

Y como podemos observar en la figura 1.1.10, este tendría una precisión del 96.98%, lo cual indica que este tiene una un buen rendimiento en cuanto a la predicción de tráfico normal o benigno.

Modelo de regresión logística

El segundo modelo que se va a evaluar es el modelo de regresión logística, para esto lo primero que se va a realizar es importar las librerías necesarias, se normalizaron los datos para asegurarse de que cualquier dato categórico este en el formato correcto, se dividirá el dataset en uno de entrenamiento y otro de prueba, se escalará el dataset para reducir su tamaño y obtener una mejor precisión, se ajustará el modelo al dataset y por último se harán las predicciones para poder conocer su precisión. Como se había dicho anteriormente, en la figura 1.2.1, podemos observar todas las librerías necesarias para que todas las acciones que se realicen con el dataset se ejecuten de manera correcta.

```
import pandas as pd
from ipywidgets import interact,IntSlider
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

Figura 1.2.1. Importación de librerías. Fuente: autores del proyecto

```
data_train=r"C:\Users\asus\Documents\Machine learning\Data\Train_data.csv"
df_train=pd.read_csv(data_train)

dummy=pd.get_dummies(data=df_train,columns=["protocol_type","service","flag","class"],drop_first=True)
```

dummy

	duration	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	num_failed_logins	logged_in	num_c
0	0	491	0	0	0	0	0	0	0	
1	0	146	0	0	0	0	0	0	0	
2	0	0	0	0	0	0	0	0	0	
3	0	232	8153	0	0	0	0	0	1	
4	0	199	420	0	0	0	0	0	1	
...	...	...	...	...	...	...	...	...	...	
25187	0	0	0	0	0	0	0	0	0	
25188	0	334	0	0	0	0	0	0	1	
25189	0	0	0	0	0	0	0	0	0	
25190	0	0	0	0	0	0	0	0	0	
25191	0	0	0	0	0	0	0	0	0	

25192 rows × 116 columns

Figura 1.2.2. Conversión de datos categóricos. Fuente: autores del proyecto

El siguiente paso sería convertir los atributos categóricos del data frame "data_train" a datos que puedan ser manejables por el algoritmo como se puede observar en la figura 1.2.2, por lo que se va a usar el get_dummies, el cual toma todos los atributos categóricos y divide cada uno de los parámetros en columnas en las cuales se rellenan con 1 o 0 (o en verdadero y falso), donde 1 significa que el dato pertenece a este nuevo atributo y 0 significa que no pertenece, esto se realiza con todos los datos categóricos incluyendo las clases.

```
train, test = train_test_split(dummy, test_size=0.2, random_state=10)

X_train = train.drop(columns=['class_normal'])
y_train = train['class_normal']

X_test = test.drop(columns=['class_normal'])
y_test = test['class_normal']
```

Figura 1.2.3. División de dataframe en datos de entrenamiento y de prueba. Fuente: autores del proyecto

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

model = LogisticRegression(max_iter=3000, solver='lbfgs', random_state=42)

model.fit(X_train_scaled, y_train)
y_pred = model.predict(X_test_scaled)
```

Figura 1.2.4. Escalado, instanciación, ajuste y predicción del modelo. Fuente: autores del proyecto

Como podemos observar en la figura 1.2.3, dividimos los datos, donde se tomará el 20% para la prueba y el 80% restante serán los de entrenamiento. En entrenamiento tenemos dos partes, en X_train se usarán todos los datos del dataframe a excepción de la clase, mientras que en y_train se dejará la clase sola, lo mismo se hará para los datos de prueba.

En la figura 1.2.4 podemos ver varios procesos, lo primero que se realizará será escalar el modelo; se hace esto con la finalidad de reducir el tamaño del dataset, esto se debe a que el modelo de regresión logístico es un modelo que se beneficia de dataframes con un tamaño pequeño.

Para la instanciación del modelo, colocamos dentro de los parámetros de este un número fijo de iteraciones, el cual será igual a 3000, usamos el solucionador lbfgs, el cual es un código quasi-Newton de memoria limitada para optimización con restricciones de límites (Zhu, Byrd, Nocedal, & Morales), usado por defecto por el algoritmo de regresión logística y el estado al azar se deja en 42. Lo siguiente será ajustar el modelo pasando como parámetros "X_train_scaled" y "y_train". Y por último se harán las predicciones usando los datos "X_test_scaled" que como se había dicho antes, son los datos de prueba previamente escalados, sin clases y que hacen parte del 20% del dataset.

```
accuracy = accuracy_score(y_test, y_pred)
print(f"el modelo tiene una precision de: {accuracy*100}%")
```

```
el modelo tiene una precision de: 97.28120658860885%
```

```
# Mostrar un reporte de clasificación
report = classification_report(y_test, y_pred)
print("Reporte de clasificación:")
print(report)
```

```
Reporte de clasificación:
              precision    recall  f1-score   support

       False       0.98      0.96      0.97      2358
        True       0.97      0.98      0.97      2681

    accuracy                           0.97      5039
   macro avg       0.97      0.97      0.97      5039
weighted avg       0.97      0.97      0.97      5039
```

Figura 1.2.5. Precisión y reporte de clasificación del modelo. Fuente: autores del proyecto

En la figura 1.2.5 podemos observar como el modelo tiene una precisión del 97.2%, se puede observar mejoría sobre el modelo anterior, también podemos destacar algunos datos proporcionados por el reporte de clasificación son la precisión en el que el modelo clasifica ambas clases, y donde se puede observar una pequeña diferencia entre la forma en que este clasifica al tráfico malicioso y el no malicioso.

Modelo de árbol de decisión

Para este modelo los pasos a seguir serán importar las librerías, codificar los datos con la codificación one hot encoding usando la función get_dummies, dividir los datos en datos de entrenamiento y datos de prueba, se instancia el modelo, se harán las predicciones y por último se realizará la normalización de este.

Las librerías usadas para este modelo son bastante similares a las del modelo anterior, con la diferencia de que se usarán la librería seaborn y matplotlib.pyplot para poder realizar una representación gráfica de la matriz de confusión y determinar falsos positivos y negativos.

```
import pandas as pd
import numpy as np
from ipywidgets import interact,IntSlider
from sklearn.metrics import accuracy_score, classification_report
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sn
```

Figura 1.3.1. Importación de librerías. Fuente: autores del proyecto

```
data_train=r"C:\Users\asus\Documents\Machine learning\Data\Train_data.csv"
df_train=pd.read_csv(data_train)

dummy=pd.get_dummies(data=df_train,columns=["protocol_type","service","flag","class"],drop_first=True)

# Asegúrate de importar train_test_split de sklearn.model_selection si no lo has hecho
from sklearn.model_selection import train_test_split

# Dividir el conjunto de datos en entrenamiento y prueba (70% entrenamiento, 30% prueba)
train, test = train_test_split(dummy, test_size=0.3, random_state=42)

# Definir las características (X) y la variable objetivo (y) en el conjunto de entrenamiento
X_train = train.drop(columns=['class_normal'])  # Eliminar la variable objetivo "class" de las características
y_train = train['class_normal']

# Definir las características (X) y la variable objetivo (y) en el conjunto de prueba
X_test = test.drop(columns=['class_normal'])
y_test = test['class_normal']
```

Figura 1.3.2. Codificación y división de datos en datos de entrenamiento y datos de prueba. Fuente: autores del proyecto

Lo primero que se debe hacer es llamar al dataset de nuestra ruta del dispositivo e instanciar el data frame para poder realizar la codificación de este. Para la codificación, se usará One hot encoding, el cual toma los atributos categóricos y convierte cada una de los datos en columnas las cuales son asignadas valores de 1 o 0 dependiendo de si este es verdadero o falso en el contexto del dataset.

Y se divide tomando como datos de prueba al 30% del dataset.

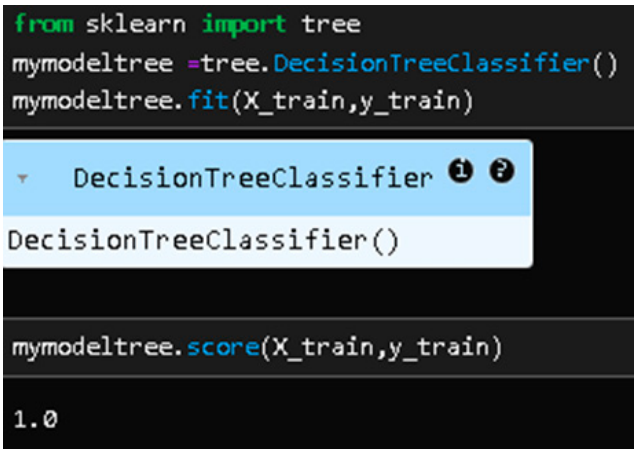


Figura 1.3.3. Instanciación del modelo. Fuente: autores del proyecto

Instanciamos el modelo de árbol de decisión y pasamos dentro de el ajuste los datos de entrenamiento "X" y "y", como se observa en la figura 1.3.3, los cuales son el 70% que no se usó para la prueba, siendo "X" donde estarán todos los datos a excepción de la clase y "y" la clase. Al evaluarlo, este debe tener un puntaje del 100%, esto es debido a que estos datos conocen la clase a la que pertenecen, por lo que sería anormal que a estos se les asignara un puntaje diferente. Se hace la predicción con los datos de entrenamiento que no contienen la clase y obtenemos estás predicciones en un array, esto se puede observar en la figura 1.3.4

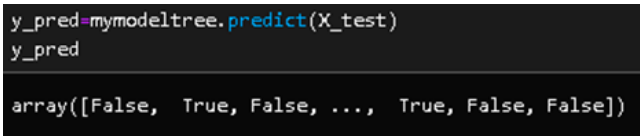


Figura 1.3.4. Predicción del modelo. Fuente: autores del proyecto

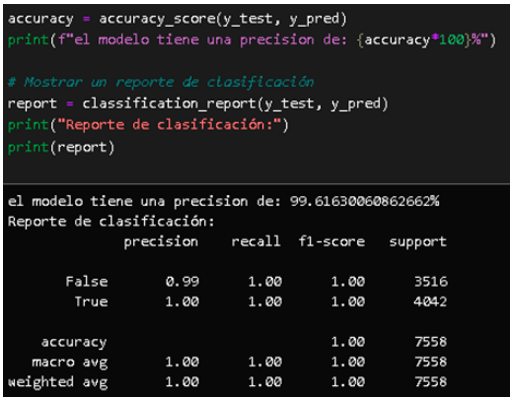


Figura 1.3.5. Predicción del modelo. Fuente: autores del proyecto

Se puede observar en la figura 1.3.5 como el modelo tiene una precisión del 99.6% y de acuerdo al reporte de clasificación, este clasifica en un 100% al tráfico normal.

Como se puede observar en la figura 1.3.6, el modelo es capaz de clasificar al tráfico malicioso en la mayor parte de los casos, teniendo estos pocos datos clasificados erróneamente siendo solo 11 datos falsos positivos y 18 falsos negativos. Como se puede observar en la figura 1.3.6, el modelo es capaz de clasificar al tráfico malicioso en la mayor parte de los casos, teniendo estos pocos datos clasificados erróneamente siendo solo 11 datos falsos positivos y 18 falsos negativos.

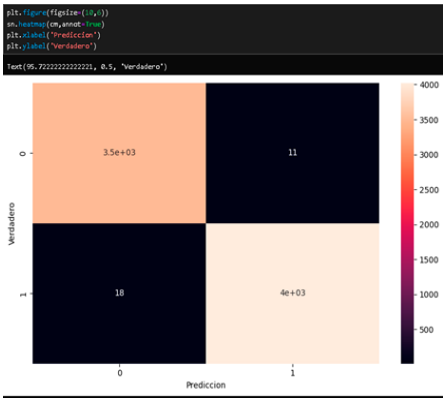


Figura 1.3.6. Mapa de calor y matriz de confusión . Fuente: autores del proyecto

Modelo Naive Bayes

Esta vez haremos uso del modelo de Naive Bayes, este modelo se usa principalmente para los problemas de clasificación. Este modelo asume que las características del conjunto de datos son independientes entre sí, lo que en la práctica no siempre es verdad, pero esto nos permite que el modelo pueda ser más eficiente.

Lo primero que hacemos según se ve en la imagen 1.4.1 es hacer las respectivas importaciones de las librerías de las cuales haremos uso para realizar el modelo en jupyter lab.

```
import pandas as pd
import numpy as np
from ipywidgets import interact,IntSlider
from sklearn.metrics import accuracy_score, classification_report
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sn
```

Figura 1.4.1. Importaciones de librerías. Fuente: autores del proyecto

Se define la variable data train la cual va a tener como valor la ruta donde se encuentra nuestro dataset, además de eso hacemos uso de otra variable llamada df_train, la cual hará uso de una función de la biblioteca Pandas la cual usaremos para poder leer los datos que se encuentran dentro de la variable data_train y cargar estos en un dataframe.

Figura 1.4.2. Almacenamiento de dataset. Fuente: autores del proyecto

Figura 1.4.3. Almacenamiento de dataset. Fuente: autores del proyecto

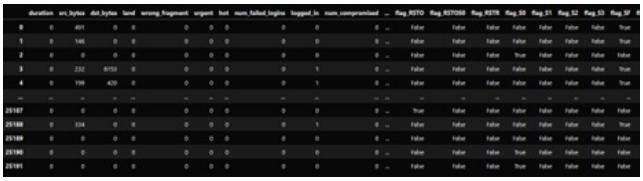


Figura 1.4.4. Dummy obtenido. Fuente: autores del proyecto

En la imagen 1.4.3 podemos observar cómo realizamos la transformación de los datos categóricos a numéricos, esto se hace con la finalidad de hacer más fácil el tratamiento del modelo y poder realizar un maquetado más accesible. Para la transformación de los datos hacemos uso de la librería dummy, está lo que hace es crear nuevas columnas con valores binarios, para indicar sí se encuentra la categoría indicada o no, si se encuentra es 1 y si no es 0. Al final lo que hacemos es mostrar el dummy obtenido, como se ve en la imagen 1.4.4.

Como podemos ver en la imagen 1.4.5 lo que realizamos es la división de los datos del dataset una parte para entrenamiento y otra parte para pruebas, esto, los x train shape, y x test shape, vienen siendo el tamaño de cada uno de estos datasets creados por la división, esto se hace con el objetivo de poder testear el modelo con estos datos, ya que no se cuenta con un dataset propio de pruebas.

Figura 1.4.5. Separación de data training y data target. Fuente: autores del proyecto

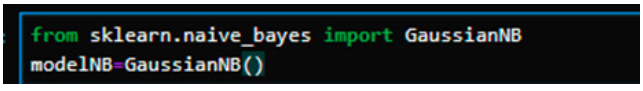


Figura 1.4.6. Modelo de clasificación utilizando algoritmo Naive Bayes. Fuente: autores del proyecto

En esta parte se empieza a hacer uso del modelo, para eso, hacemos uso de la librería sckit-learn, importando la función GaussianNB. Este modelo es bueno para problemas de clasificación donde las características se distribuyen de forma continua, como se ve en la imagen 1.4.6.

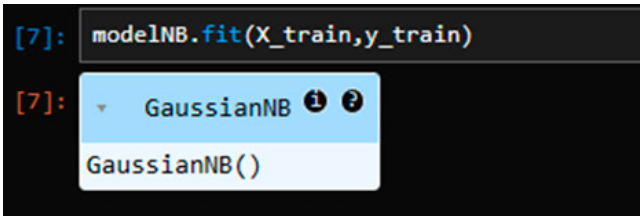


Figura 1.4.7. Entrenamiento de modelo. Fuente: autores del proyecto

Utilizamos el método fit, para entrenar el modelo con los datos de entrenamiento que poseemos, haciendo uso del conjunto de características x_train e y_train. El método fit ajusta el modelo a los datos de entrenamiento correspondientes. Durante el proceso de entrenamiento, el modelo estima la probabilidad a priori de cada clase. Estas estimaciones se basan en las distribuciones de las características en x_train las etiquetas_train. Una vez que ya entrenó almacena esos datos, para hacer predicciones.

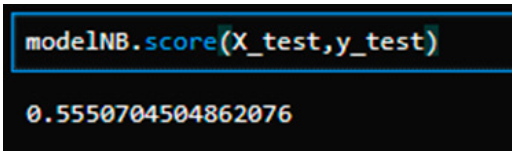


Figura 1.4.8. Evaluación de rendimiento. Fuente: autores del proyecto

En la imagen 1.4.8 hacemos uso del método score, este método de la librería scikit-learn se utiliza para evaluar el rendimiento del modelo en el conjunto de datos de prueba. Este hace uso de dos argumentos, estos dos argumentos podemos observar que son los dos dataframes que contienen los datos de prueba o de test. La precisión del modelo se calcula como la proporción de predicciones correctas que realiza el modelo en el conjunto de pruebas, como podemos observar en la imagen 1.4.8 fue de 0.55 o de 55% lo cual es muy poco.

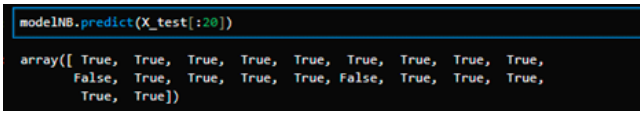


Figura 1.4.9. Predicción de los resultados. Fuente: autores del proyecto

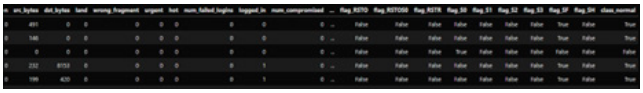


Figura 1.4.10. Datos del dataframe. Fuente: autores del proyecto

En esta parte hacemos uso del método predict, para realizar predicciones sobre nuestros datos utilizando el modelo entrenado, al hacer esto podemos observar según la imagen 1.4.9 que supuestamente los 20 primeros datos del dataframe son true, pero dirigiéndonos a el dataframe obtenido con dummy de la imagen 1.4.10, podemos observar que esto no es verdad ya que el tercer dato de estos primeros 20 es false, así como muchos otros, esto se podría deber a algún sesgo.También podemos obtener la probabilidad de que estos datos pertenecen a cada una de estas clases, en lugar de predecir la clase,

para eso hacemos uso del método predict prueba, para calcular las probabilidades de las 10 primeras muestras en el conjunto de características, como podemos observar en la imagen 1.4.11. Estos datos nos dicen que la confianza del modelo es buena, pero como podemos observar al ver el dataframe original esto no es para nada verdad, por lo que podemos decir que el modelo está sesgado. En la imagen 1.4.12 podemos observar la matriz de confusión del modelo en esta podemos interpretar lo siguiente se nos dice que tenemos 183 verdaderos negativos y 2181 falsos positivos, esto lo que nos quiere decir es que tenemos 183 que fueron adecuadamente clasificados y 2181 que no fueron clasificados de manera correcta, lo cual es un gran número de datos que fueron mal clasificados lo cual nos deja un sinsabor por este lado, además de esto tenemos 61 falsos negativos y 2614 verdaderos positivos, por lo que se puede decir que clasificó de manera correcta 2614 datos y clasificó mal a 61 datos positivos que no lo eran, esto nos muestra que el modelo tiene una inclinación a indicar la clase como true cuando no es el caso, lo cual nos explica lo que vimos en la imagen 1.4.9.

```
[10]: modelNB.predict_proba(X_test[:10])

[10]: array([[2.60132580e-01, 7.39867420e-01],
             [1.62315158e-04, 9.99837685e-01],
             [3.55997471e-01, 6.44002529e-01],
             [6.39716178e-04, 9.99360284e-01],
             [4.97681301e-03, 9.95023187e-01],
             [2.65463326e-02, 9.73453667e-01],
             [5.24930137e-02, 9.47506986e-01],
             [3.57106714e-01, 6.42893286e-01],
             [1.47189441e-04, 9.99852811e-01],
             [9.95755927e-01, 4.24407305e-03]])
```

Figura 1.4.11. Probabilidad de pertenencia de x_test. Fuente: autores del proyecto

```
cm= confusion_matrix(y_test,y_pred)
cm

array([[ 183, 2181],
       [  61, 2614]], dtype=int64)
```

Figura 1.4.12. Matriz de confusión. Fuente: autores del proyecto

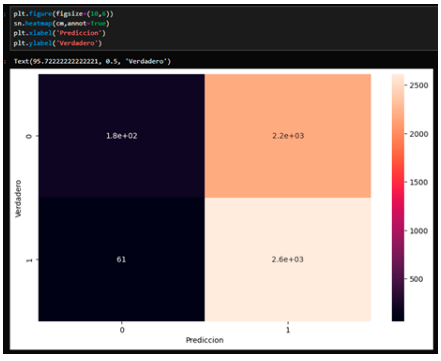


Figura 1.4.13. Mapa de calor de matriz de confusión. Fuente: autores del proyecto

En la imagen 1.4.13 podemos ver un mapa de calor, este mapa de calor nos ayuda a visualizar de mejor forma la matriz de confusión, aquí podemos ver como en la imagen pasada que este modelo es muy bueno al identificar verdaderos positivos, pero al también identificar como falsos positivos en la misma cantidad que lo anterior lo arruina todo, ya que es muy malo prediciendo datos que son falsos.

Random forest

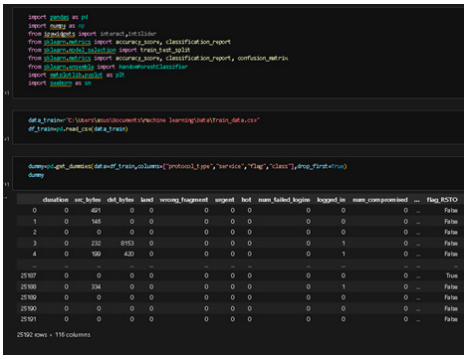


Figura 1.5.1. Importación de librerías y codificación de datos categóricos. Fuente: autores del proyecto

Se importan las librerías necesarias y se usa get dummies para convertir los datos categóricos en datos numéricos.

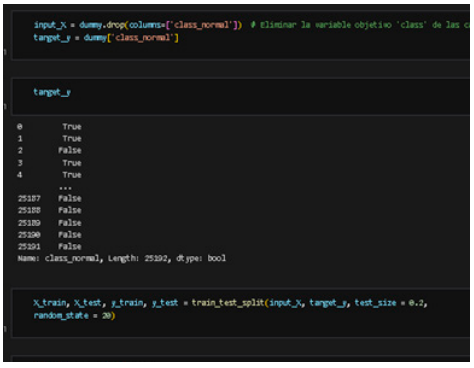


Figura 1.5.2. Separación en dataset de entrenamiento y de prueba. Fuente: autores del proyecto

Se separa el dataset en datos de entrenamiento y datos de prueba.

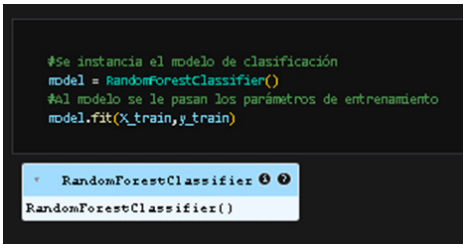


Figura 1.5.3. Implementación del modelo random forest. Fuente: autores del proyecto

Se implementa el modelo de random forest y se ajusta el dataset de entrenamiento.

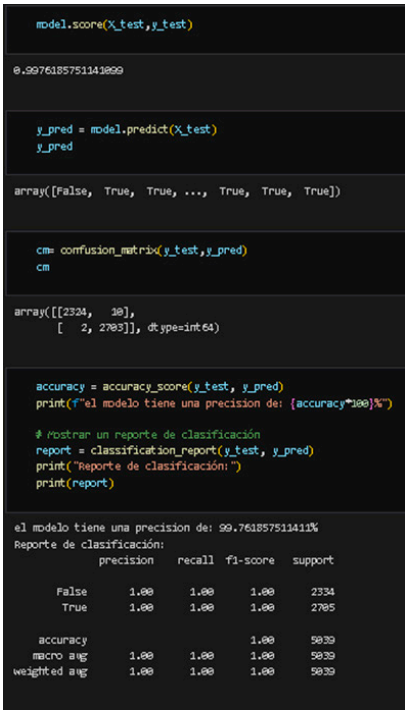


Figura 1.5.4. Resultados del modelo. Fuente: autores del proyecto

Se muestran los resultados obtenidos por el modelo, mostrando una precisión del 0.99 o el 99%.

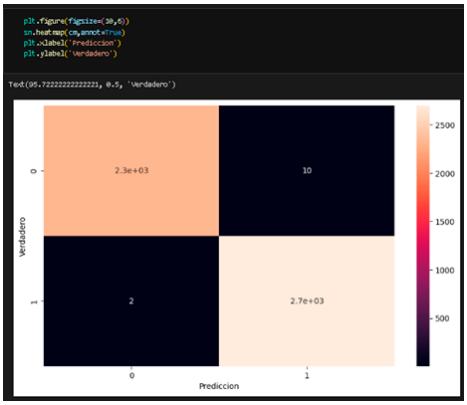


Figura 1.5.5. Mapa de Calor.
Fuente: autores del proyecto

Se logra observar que el modelo no comete tantos errores para clasificar a la clase del dataset en el dataset de prueba.

SMV

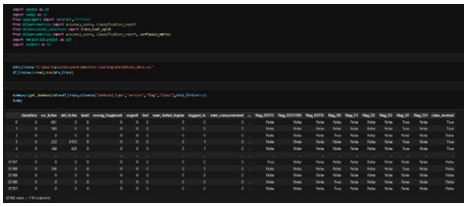


Figura 1.6.1. Importación de librerías y uso de la función get_dummies. Fuente: autores del proyecto

Se importan las librerías necesarias y se convierten los datos de tipo categórico a numéricos.

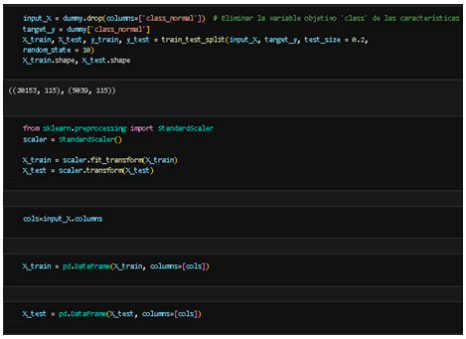


Figura 1.6.2. Separación de dataset de entreno y de prueba. Fuente: autores del proyecto

Se separa el dataset en datos de entrenamiento y de prueba para comprobar la precisión del modelo.

Figura 1.6.3. Puntuación de precisión.
Fuente: autores del proyecto

Se genera un puntaje de precisión que en este caso es de un 99% con parámetros predeterminados.

Figura 1.6.4. Mapa de calor de la matriz de confusión. Fuente: autores del proyecto

Se muestra el mapa de calor de la matriz de confusión para mostrar los datos que están clasificados de forma correcta y los que no están correctamente clasificados.

Como se pudo observar, los modelos con los mejores resultados fueron random forest, decision tree y SMV, dado a que son los que dan una precisión del 99%, la cual comparada a los otros modelos es la mejor, pero SMV se puede excluir debido a que esta toma demasiado tiempo en generar las predicciones, comparado con los otros algoritmos de ML.

El otro dataset que se usó es el dataset con el nombre: Network Traffic Data-Malicious Activity Detection [55], [56], [57]. Este conjunto de datos consiste en tráfico de red capturado desde una máquina Kali Linux, con el objetivo de ayudar al desarrollo y evaluación de modelos de aprendizaje automático para distinguir entre actividades normales y maliciosas (específicamente ataques de inundación) en redes. Incluye una variedad de características esenciales para identificar posibles amenazas de ciberseguridad, junto con etiquetas que indican si cada paquete forma parte del tráfico de inundación.

El conjunto de datos fue cuidadosamente compilado utilizando tráfico de red capturado desde una configuración dedicada de Kali Linux. El entorno de captura consistió en una máquina Kali Linux configurada para generar y capturar tanto tráfico de red normal como malicioso, y una máquina objetivo con sistema operativo Windows para simular un entorno de red real.

Preprocesamiento

Figura 1.7.1. Manejo de el dataset. Fuente: autores del proyecto

Como el dataset original tiene demasiados datos y la cantidad de datos identificados como maliciosos es menor a la cantidad de datos normales, se realiza una generación de datos sintéticos y se hace una reducción del dataset para que de esta manera se puedan manejar los modelos con mayor facilidad.

Regresión logística

Figura 1.8.1. Importación de librerías y codificación de datos categóricos. Fuente: autores del proyecto

Se hace la importación de todas las librerías relevantes, se inicializa el dataframe y se usa el método dummies para convertir los datos categóricos a numéricos.

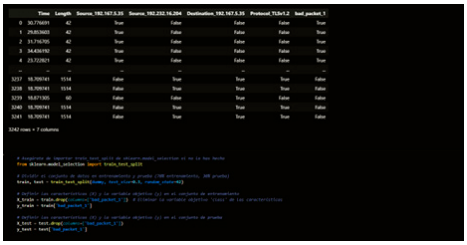


Figura 1.8.2. Implementación del modelo. Fuente: autores del proyecto

Después se hace la muestra de la tabla dummies y se realiza la implementación del modelo, donde se divide el conjunto de datos para obtener un dataset de entrenamiento y otro de prueba.

Figura 1.8.3. Resultados.
Fuente: autores del proyecto

Y por último se obtienen los resultados, donde como se puede observar, se obtuvo una precisión de 1.0.

Decisión tree

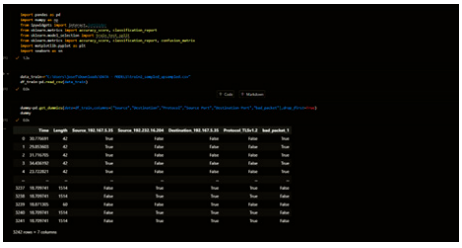


Figura 1.9.1. Resultados.
Fuente: autores del proyecto

Se importan las librerías necesarias y se convierten todos los datos categóricos en numéricos.

Figura 1.9.2. Implementación del modelo.
Fuente: autores del proyecto

Se hace la implementación del modelo de árbol de decisión.

Figura 1.9.3. Muestra de resultados.
Fuente: autores del proyecto

Mostramos los resultados de las predicciones.

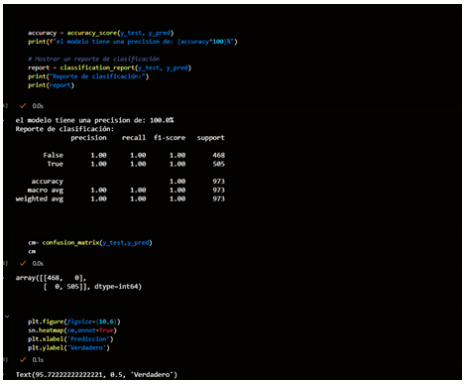


Figura 1.9.4. Reporte de clasificación.
Fuente: autores del proyecto

Se muestra la predicción del modelo, la cual viene siendo de 1.0

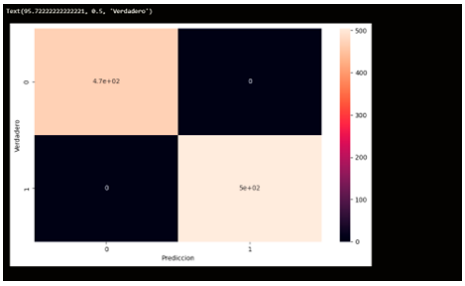


Figura 1.9.5. Mapa de calor.
Fuente: autores del proyecto

Y por último se muestra un mapa de calor con la matriz de confusión generada por los resultados del modelo.

Naive bayes

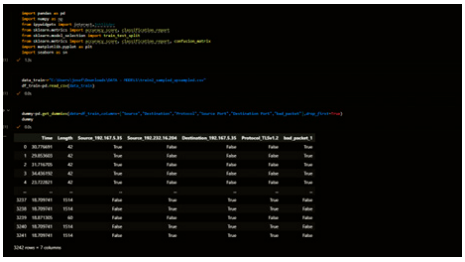


Figura 1.10.1. Importación de librerías y codificación de datos categóricos. Fuente: autores del proyecto

Se realiza el proceso de codificación para convertir todos los datos categóricos a numéricos.

Figura 1.10.2. Implementación del modelo.
Fuente: autores del proyecto

Se hace la implementación del modelo naive bayes.

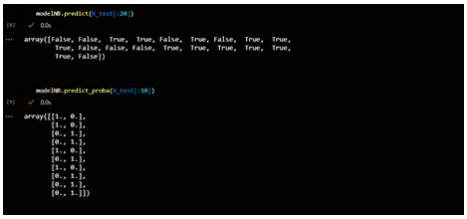


Figura 1.10.3. Muestra de resultados.
Fuente: autores del proyecto

Figura 1.10.4. Matriz de confusión.
Fuente: autores del proyecto

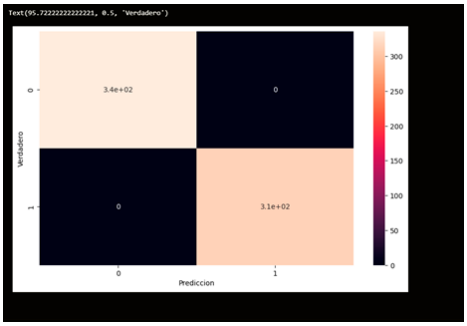


Figura 1.10.5. Mapa de calor.
Fuente: autores del proyecto

Y como podemos observar en los resultados otra vez obtenemos un resultado con una precisión perfecta.

Regresión lineal

Figura 1.11.1. Importación de librerías.
Fuente: autores del proyecto

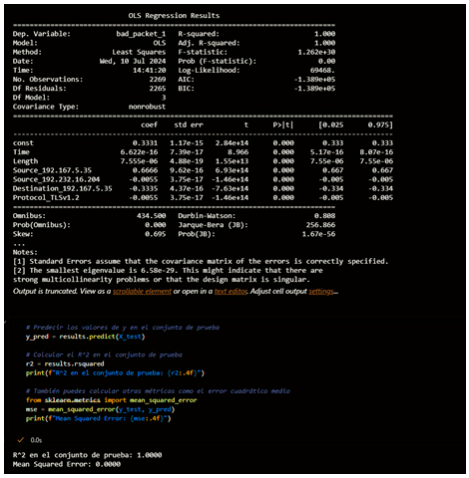
Figura 1.11.2. Separación del dataset.
Fuente: autores del proyecto

Se hacen todos los procesos pertinentes para poder trabajar con el modelo, procesos como separar el dataset en uno de entrenamiento y otro de prueba, y convertir estos en datos de tipo float para que no ocurra ningún problema al momento de implementar el modelo.

```
bool
Time                        float64
Length                        int64
Source_192.167.5.35            bool
Source_192.232.16.204          bool
Destination_192.167.5.35       bool
Protocol_TLSv1.2               bool
dtype: object

y_train = y_train.astype(float)
X_train = X_train.astype(float)

y_test = y_test.astype(float)
X_test = X_test.astype(float)

# Agregar una constante a las características (intercepto)
X_train = sm.add_constant(X_train)
X_test = sm.add_constant(X_test)

# Crear el modelo de regresión lineal
model = sm.OLS(y_train, X_train)

# Ajustar el modelo
results = model.fit()

# Resumen del modelo
print(results.summary())
```

Figura 1.11.3. Implementación del modelo.
Fuente: autores del proyecto

Se ajusta el dataset al modelo de la regresión lineal.

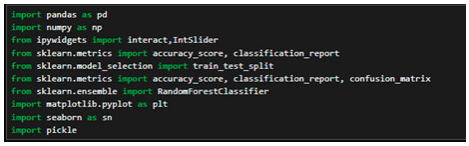


```
                            OLS Regression Results
==============================================================================
Dep. Variable:              bad_packet_1   R-squared:                       1.000
Model:                              OLS   Adj. R-squared:                  1.000
Method:                   Least Squares   F-statistic:                1.262e+30
Date:                  Wed, 10 Jul 2024   Prob (F-statistic):               0.00
Time:                          14:41:20   Log-Likelihood:                 69468.
No. Observations:                2269   AIC:                        -1.389e+05
Df Residuals:                    2265   BIC:                        -1.389e+05
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.3331   1.17e-15   2.84e+14      0.000       0.333       0.333
Time        6.622e-16   7.39e-17      8.966      0.000    5.17e-16    8.07e-16
Length      7.555e-05   4.88e-19   1.55e+11      0.000    7.55e-05    7.55e-05
Source_192.167.5.35        0.6666   9.62e-16   6.93e+14      0.000       0.667       0.667
Source_192.232.16.204     -0.0055   3.75e-17  -1.46e+14      0.000      -0.005      -0.005
Destination_192.167.5.35  -0.1335   4.37e-16  -7.63e+14      0.000      -0.134      -0.134
Protocol_TLSv1.2          -0.0055   3.75e-17  -1.46e+14      0.000      -0.005      -0.005
==============================================================================
Omnibus:                      434.500   Durbin-Watson:                   0.888
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              256.866
Skew:                           0.695   Prob(JB):                     1.67e-56
...
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 6.58e-29. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

# Predecir los valores de y en el conjunto de prueba
y_pred = results.predict(X_test)

# Calcular el R^2 en el conjunto de prueba
r2 = results.rsquared
print(f"R^2 en el conjunto de prueba: {r2:.4f}")

# También puedes calcular otras métricas como el error cuadrático medio
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse:.4f}")

✓ 0.0s
R^2 en el conjunto de prueba: 1.0000
Mean Squared Error: 0.0000
```

Figura 1.11.4. Resultados.
Fuente: autores del proyecto

Obtenemos los resultados, los cuales una vez, nos dan de forma perfecta.

Modelo Random Forest



```
import pandas as pd
import numpy as np
from ipywidgets import interact,IntSlider
from sklearn.metrics import accuracy_score, classification_report
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.ensemble import RandomForestClassifier
import matplotlib.pyplot as plt
import seaborn as sn
import pickle
```

Figura 1.12.1. Importaciones de librerías.
Fuente: autores del proyecto

Ahora procederemos con la implementación del modelo random forest. Este modelo se basa en la creación de varios árboles de tomas de decisión y la combinación de las predicciones para de esa manera obtener resultados más precisos y completos. Este modelo trabaja de muy buena manera con datasets de gran tamaño, por lo que esperamos obtener resultados de muy buena calidad en este modelo.

```
# Leer el archivo CSV completo
df = pd.read_csv("C:/Users/josef/Downloads/DATA - MODELS/train2.csv")
```

Figura 1.12.2. Almacenamiento de dataset. Fuente: autores del proyecto

El código carga los datos del archivo CSV train2.csv ubicado en la ruta específica C:/Users/josef/Downloads/DATA - MODELS/ en un DataFrame llamado df, lo que te permite manipular y analizar estos datos usando las funcionalidades de pandas.

```
# Rellenar valores nulos en Source Port y Destination Port con -1
df['Source Port'].fillna(-1, inplace=True)
df['Destination Port'].fillna(-1, inplace=True)
```

Figura 1.12.3. Rellenar valores nulos en Source Port y Destination Port con -1. Fuente: autores del proyecto

En la imagen 1.12.3 podemos ver un código que nos permite rellenar los valores nulos (NaN) en las columnas 'Source Port' y 'Destination Port' del DataFrame df con el valor -1. El parámetro inplace=True asegura que los cambios se realicen directamente en el DataFrame original df.

```
# Muestreo aleatorio del 30% de los datos (ajusta el porcentaje según tus necesidades)
sampled_df = df.sample(frac=0.0005, random_state=42)
```

Figura 1.12.4. Muestreo aleatorio del 30% de los datos. Fuente: autores del proyecto

En la imagen 1.12.4 hacemos uso de una nueva variable llamada sampled_df es un nuevo DataFrame que contiene una muestra aleatoria del 0.05% (0.0005) de las filas del DataFrame original df. El parámetro random_state=42 asegura que el muestreo sea reproducible, hacemos uso de una muestra aleatoria de 0.05% debido a la gran cantidad de datos que contiene el dataset de cerca de 3 millones y medio de datos.

```
# Separar las clases
df_majority = sampled_df[sampled_df.bad_packet == 1]
df_minority = sampled_df[sampled_df.bad_packet == 0]
```

Figura 1.12.5. Separar las clases. Fuente: autores del proyecto

Se separan las filas del DataFrame sampled_df en dos grupos basados en el valor de la columna 'bad_packet'. df_majority contiene las filas donde 'bad_packet' es igual a 1 (clase mayoritaria) y df_minority contiene las filas donde 'bad_packet' es igual a 0 (clase minoritaria)

```
# Sobremuestreo de la clase minoritaria
df_minority_upsampled = resample(df_minority,
                                 replace=True,     # muestra con reemplazo
                                 n_samples=len(df_majority),    # para igualar la clase mayoritaria
                                 random_state=42) # para reproducibilidad
```

Figura 1.12.6. Separación de características del dataframe. Fuente: autores del proyecto

En la imagen 1.12.6 realizamos la división del dummy en dos partes la primera parte el input_x contendrá todas las características del dataframe excepto la class normal; la segunda parte target_y contendrá solo la columna class_normal, con esto podremos comenzar la evaluación del modelo.

```
# Sobremuestreo de la clase minoritaria
df_minority_upsampled = resample(df_minority,
                                 replace=True,     # muestra con reemplazo
                                 n_samples=len(df_majority),    # para igualar la clase mayoritaria
                                 random_state=42) # para reproducibilidad
```

Figura 1.12.7. Sobremuestreo de la clase minoritaria. Fuente: autores del proyecto

En la imagen 1.12.7, se utiliza la función resample de sklearn.utils para realizar el sobremuestreo de la clase minoritaria (df_minority). replace=True indica que se deben seleccionar muestras con reemplazo, n_samples=len(df_majority) establece el número de muestras a igualar con la clase mayoritaria, y random_state=42 asegura la reproducibilidad.

```
# Combinar las clases mayoritaria y sobremuestreada
df_upsampled = pd.concat([df_majority, df_minority_upsampled])
```

Figura 1.12.8. Combinar las clases mayoritaria y sobre muestreada. Fuente: autores del proyecto

En la imagen 1.12.8 df_upsampled es un nuevo DataFrame que combina las filas de df_majority (clase mayoritaria) con las filas sobre muestreadas de df_minority_upsampled (clase minoritaria).

```
# Guardar el archivo CSV reducido
df_upsampled.to_csv("C:/Users/josef/Downloads/DATA - MODELS/train2_sampled_upsampled.csv", index=False)
```

Figura 1.12.9. Guardar el archivo CSV reducido. Fuente: autores del proyecto

Se guarda el DataFrame df_upsampled en un archivo CSV ubicado en la ruta especificada

```
dummy=pd.get_dummies(data=df_upsampled,columns=["Source","Destination","Protocol","Source Port","Destination Port","bad_packet"],drop_first=True)
dummy
```

Figura 1.12.10. Transformación de variables categóricas a numéricas. Fuente: autores del proyecto

Como se ve en la imagen 1.12.10, pd.get_dummies() de pandas transforma variables categóricas en variables dummy en un DataFrame. Cada columna categórica especificada en columns se convierte en múltiples columnas binarias, donde cada columna representa la presencia o ausencia de una categoría particular. El parámetro drop_first=True elimina la primera columna dummy de cada conjunto para evitar la multicolinealidad en modelos de regresión. Esto es útil para preparar datos categóricos para análisis estadístico o modelos de aprendizaje automático que requieren entradas numéricas en lugar de categorías.

```
dummy['bad_packet_1'].value_counts()

bad_packet_1
True     1621
False    1621
Name: count, dtype: int64
```

Figura 1.12.11. Conteo de datos del bad_packet. Fuente: autores del proyecto

En la imagen 1.12.11 podemos ver que gracias a dummy['bad_packet_1'].value_counts() obtenemos un conteo de cuántas filas tienen el valor 1 en la columna 'bad_packet_1' y cuántas tienen el valor 0, si 'bad_packet_1' es el nombre de una columna dummy que representa la variable categórica original 'bad_packet' con la categoría 1. Esto es útil para comprender la distribución de la variable dummy generada a partir de la categoría 'bad_packet' en tu conjunto de datos.

```
plt.figure(figsize=(10,6))
sn.heatmap(cm,annot=True)
plt.xlabel('Prediccion')
plt.ylabel('Verdadero')
```

```
Text(95.72222222222221, 0.5, 'Verdadero')
```

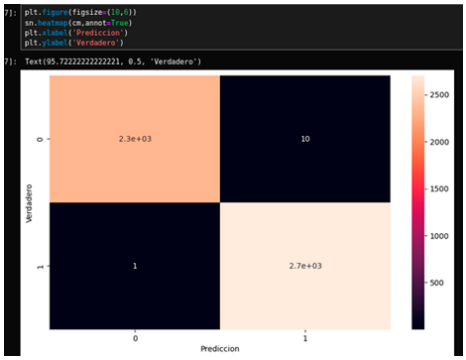


Figura 1.12.12. Mapa de calor de matriz de confusión. Fuente: autores del proyecto

Con el mapa de calor podemos observar de mejor manera la matriz de confusión y cómo son bastante pequeñas la cantidad de predicciones mal hechas.

SMV

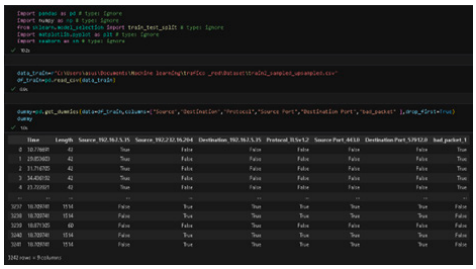


Figura 1.13.1. Importación de librerías y codificación de datos categóricos. Fuente: autores del proyecto

Se importan las librerías y se convierten los datos categóricos en numéricos.

Figura 1.13.2. Separación en dataset de entrenamiento y de prueba, y preparación del dataset. Fuente: autores del proyecto

Se separa el dataset en datos de entrenamiento y datos de prueba y se hace la respectiva preparación de los datos para no tener problemas con el modelo.

Figura 1.13.3. Resultados del modelo. Fuente: autores del proyecto

Se nos muestra el puntaje de precisión, el cual es de un 1.0

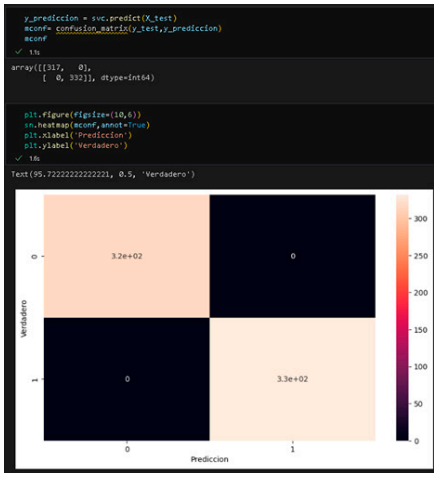


Figura 1.13.4. Mapa de calor. Fuente: autores del proyecto

Se genera un mapa de calor mostrando que el modelo no está clasificando ningún dato de forma errónea. De este dataset podemos observar que todos los modelos nos están dando una precisión de 1.0, lo cual puede significar varias opciones, tales como que el dataset está sesgado y no es capaz de mostrar de manera parcial los resultados reales; otra posibilidad es que este esté siendo sobreentrenado y por ende este es capaz de conocer la clase de los conjuntos de datos de prueba.

Dado este resultado se realizó un esquema de cómo funciona el algoritmo que se realizará a partir de estos modelos previamente analizados.

Conclusiones

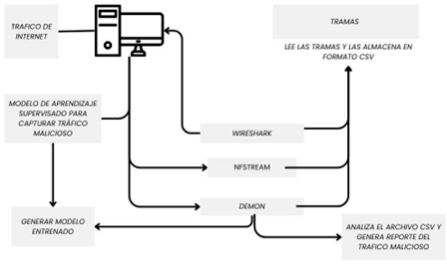


Figura 1.13.5. Esquema de algoritmo para la detección de tráfico malicioso. Fuente: autores del proyecto

El proyecto de Implementación de Modelos de Aprendizaje Supervisado para la Detección de Tráfico Malicioso se centra en utilizar técnicas de aprendizaje automatizado para fortalecer la seguridad cibernética. Mediante el análisis y modelado de datos, se pretende identificar y mitigar distintas amenazas que se pueden presentar al momento de navegar en la red. La aplicación de estos modelos no solo busca proteger sistemas críticos y datos sensibles, sino también proporcionar herramientas efectivas para enfrentar los desafíos constantes del ciberespacio. La integración de estas tecnologías representa un paso crucial hacia la prevención proactiva y la respuesta rápida frente a las amenazas digitales emergentes.

Y a pesar de que todavía faltan demasiados avances en el área es importante seguir desarrollando distintas técnicas y proponer modelos para poder crear un espacio más seguro y libre de amenazas cibernéticas.

Referencias

[1] Díaz Navarro, J. (2022). Inteligencia Artificial para la detección de binarios maliciosos.

[2] León, D. A., Martínezq, J. G., Ardila, I. A., & Mosquera, D. J. (2022). Inteligencia artificial para el control de tráfico en redes de datos: Una Revisión. Entre Ciencia e Ingeniería, 16(31), 17-24.

[3] The Bridge.Inteligencia artificial y ciberseguridad: detección de intrusiones. https://thebridge.tech/blog/inteligencia-artificial-y-ciberseguridad-deteccion-de-intrusiones

[4] Molina Abellán, I. (2024). Análisis y detección de Tráfico Malicioso mediante Machine Learning y Deep Learning.

[5] TIC Negocios. (2021). Malware e IA: una simbiosis perfecta con muchas luces y sombras. https://ticnegocios.camaravalencia.com/servicios/tendencias/ia-para-la-deteccion-y-la-creacion-de-malware-dos-cara-de-una-misma-moneda/

[6] Russell, S., & Norvig, P. (2016). Artificial Intelligence: A Modern Approach (3rd ed.). Pearson.

[7] Mitchell, T. M. (1997). Machine Learning. McGraw-Hill.

[8] Cortes, C., & Vapnik, V. (1995). Support-vector networks. Machine Learning, 20(3), 273-297.

[9] Murphy, K. P. (2012). Machine Learning: A Probabilistic Perspective. MIT Press.

[10] Seber, G. A. F., & Lee, A. J. (2012). Linear Regression Analysis (2nd ed.). Wiley.

[11] Hosmer, D. W., Lemeshow, S., & Sturdivant, R. X. (2013). Applied Logistic Regression (3rd ed.). Wiley.

[12] Breiman, L. (2001). Random forests. Machine Learning, 45(1), 5-32.

[13] Quinlan, J. R. (1986). Induction of decision trees. Machine Learning, 1(1), 81-106.

[14] Bishop, C. M. (2006). Pattern Recognition and Machine Learning. Springer.

[15] Hastie, T., Tibshirani, R., & Friedman, J. (2009). The Elements of Statistical Learning: Data Mining, Inference, and Prediction (2nd ed.). Springer.

[16] Scarfone, K., & Mell, P. (2007). Guide to Intrusion Detection and Prevention Systems (IDPS). NIST Special Publication 800-94. National Institute of Standards and Technology.

[17] Debar, H., Dacier, M., & Wespi, A. (1999). Towards a taxonomy of intrusion-detection systems. Computer Networks, 31(8), 805-822.

[18] Lippmann, R. P., Haines, J. W., Fried, D. J., Korba, J., & Das, K. (2000). The 1999 DARPA off-line intrusion detection evaluation. Computer Networks, 34(4), 579-595.

[19] Axelsson, S. (2000). The base-rate fallacy and the difficulty of intrusion detection. ACM Transactions on Information and System Security (TISSEC), 3(3), 186-205.

[20] Szor, P. (2005). The Art of Computer Virus Research and Defense. Addison-Wesley.

[21] Cohen, F. (1984). Computer viruses: Theory and experiments. Computers & Security, 6(1), 22-35.

[22] Spafford, E. H. (1989). The Internet worm program: An analysis. Computer Communications Review, 19(1), 17-57.

[23] Stallings, W. (2005). Cryptography and Network Security: Principles and Practices (4th ed.). Prentice Hall.

[24] Kharraz, A., Robertson, W. K., Balzarotti, D., Bilge, L., & Kirda, E. (2015). Cutting the Gordian Knot: A Look Under the Hood of Ransomware Attacks. In Proceedings of the 12th Conference on Detection of Intrusions and Malware & Vulnerability Assessment (pp. 3-24).

[25] Gritzalis, D. (2004). Enhancing Web Privacy and Anonymity in the Digital Era. Information Management & Computer Security, 12(3), 255-288.

[26] Edelman, B. (2004). Adverse Selection in Online 'Trust' Certifications and Search Results. In Proceedings of the 5th ACM Conference on Electronic Commerce (pp. 76-83).

[27] Symantec. (2019). Internet Security Threat Report. Symantec Corporation.

[28] Anderson, R., Böhme, R., Clayton, R., & Moore, T. (2008). Security Economics and the Internal Market. ENISA Research Report.

[29] Tanenbaum, A. S., & Wetherall, D. J. (2011). Computer Networks (5th ed.). Pearson.

[30] Stevens, W. R. (1994). TCP/IP Illustrated, Volume 1: The Protocols. Addison-Wesley.

[31] Droms, R. (1997). Dynamic Host Configuration Protocol. RFC 2131. Internet Engineering Task Force (IETF).

[32] Plummer, D. C. (1982). An Ethernet Address Resolution Protocol. RFC 826. Internet Engineering Task Force (IETF).

[33] Postel, J. (1982). Simple Mail Transfer Protocol. RFC 821. Internet Engineering Task Force (IETF).

[34] Postel, J., & Reynolds, J. (1985). File Transfer Protocol. RFC 959. Internet Engineering Task Force (IETF).

[35] Hedrick, C. (1988). Routing Information Protocol. RFC 1058. Internet Engineering Task Force (IETF).

[36] Borchani, H. (2020). Machine learning for malicious traffic analysis: A survey. Journal of Network and Computer Applications, 168, 102761. doi:10.1016/j.jnca.2020.102761

[37] Alqudah, A. M., & Yaseen, S. G. (2020). Machine learning in cybersecurity: A review. Procedia Computer Science, 167, 1234-1243. doi:10.1016/j.procs.2020.03.228

[38] Kaur, A., Gabrijelčič, D., & Klobučar, T. (2023). Artificial intelligence in cybersecurity: Evaluating the NIST cybersecurity framework. Computers & Security, 102, 102318.

[39] Wang, H., et al. (2020). AI-enabled cloud traffic analysis for predicting potential attacks. IEEE Transactions on Cloud Computing, 8(3), 842-855.

[40] K. W., & Thing, V. L. (2022). A flexible framework for encrypted malicious traffic detection using machine learning models. International Journal of Network Security, 24(2), 412-425.

[41] Djenna, A., et al. (2023). Deep neural networks for malware detection: DNN and CNN approaches. Journal of Information Security and Applications, 67, 102976.

[42] Wang, H., & Thing, V. L. (2023). Enhanced encrypted traffic analysis using hybrid feature sets for AI-driven models. Journal of Ambient Intelligence and Humanized Computing, 14(3),

[43] Abdullahi, M., et al. (2022). Machine learning methods for IoT attack detection: A review. Journal of Network and Computer Applications, 195, 105103.

[44] Jeffrey, S., Tan, C. W., & Villar, J. (2023). Hybrid anomaly detection strategies for cyber-physical systems. Future Generation Computer Systems, 136, 127-139.

[45] Liu, Y., & Liu, F. (2021). Hierarchical deep neural networks for efficient malicious traffic detection. IEEE Transactions on Information Forensics and Security, 16, 2690-2705.

[46] Lu, X., Cheng, X., & Yan, J. (2023). Information decision matrices for feature selection in malware detection. Future Generation Computer Systems, 125, 75-85.

[47] Alotaibi, M., & Barnawi, A. (2023). Protecting massive IoT networks using KNN and RF algorithms. Journal of Ambient Intelligence and Humanized Computing, 14(4), 1593-1605.

[48] Alzahrani, A., & Aldhyani, T. H. (2023). Industrial control systems security: A review of KNN and RF methods. Journal of Network and Computer Applications, 205, 105936.

[49] Zubair, M., et al. (2022). Deep learning approaches for secure healthcare systems: A survey. Computer Networks, 209, 107912.

[50] Ouiazzane, R., Addou, M., & Barramou, F. (2022). Hybrid AI-based intrusion detection system for network security. Journal of Ambient Intelligence and Humanized Computing, 13(11), 5203-5218.

[51] Shieh, S., et al. (2023). Real-time AI-based SlowDoS attack detection on encrypted traffic. IEEE Transactions on Dependable and Secure Computing, 20(3), 596-608.

[52] Younisse, R., Ahmad, I., & Abu Al-Haija, Q. (2022). Explainable AI for intrusion detection: CNN-based models. Journal of Network and Computer Applications, 212, 105979.

[53] Rimmer, V., et al. (2022). Advances in intrusion detection in open networks: A review. Computers & Security, 113, 102540.

[54] Sampada, B.. Network Intrusion Detection [Dataset].Kaggle. https://www.kaggle.com/datasets/sampadab17/network-intrusion-detection?select=Train_data.csv

[55] Advait Menon. (s.f.). Network Traffic Data for Malicious Activity Detection [Dataset].Kaggle. https://www.kaggle.com/datasets/advaitnmenon/network-traffic-data-malicious-activity-detection

[56] Gómez, J., Riaño, V. H., & Ramirez-Gonzalez, G. (2023). Traffic classification in IP networks through Machine Learning techniques in final systems. IEEE Access, 11, 44932-44940.

[57] Morelo, R. C., Flórez, J. A., & Gómez, J. E. G. (2023). Diseases and pests identification system in watermelon cultivation. Revista Colombiana de Tecnologías de Avanzada, 2(42), 93-104.