import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import MinMaxScaler

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification\_report, confusion\_matrix, accuracy\_score

# Cargar los datos

df = pd.read\_csv('C:/Users/Ramon/Downloads/cybersecurity\_attacks.csv')

print(df.head())

1. **Importaciones de librerías:**

* **pandas (pd):** Para manejar y analizar estructuras de datos.
* **numpy (np):** Para operaciones matemáticas y de arrays.
* **matplotlib.pyplot (plt):** Para crear gráficos estáticos, animados e interactivos.
* **seaborn (sns):** Para visualización de datos basada en matplotlib.
* **sklearn.model\_selection.train\_test\_split:** Para dividir el conjunto de datos en entrenamiento y prueba.
* **sklearn.preprocessing.MinMaxScaler:** Para normalizar los datos.
* **sklearn.ensemble.RandomForestClassifier:** Para crear el modelo de clasificación Random Forest.
* **sklearn.metrics.classification\_report, confusion\_matrix, accuracy\_score:**Para evaluar el rendimiento del modelo.

1. **Cargar los datos:**

* **pd.read\_csv:** Carga el archivo CSV ubicado en el camino especificado en un DataFrame llamado df.

1. **Visualización inicial de los datos:**

* **df.head():** Muestra las primeras cinco filas del DataFrame para dar una visión general de los datos cargados.

Timestamp Source IP Address Destination IP Address Source Port \

0 2023-05-30 06:33:58 103.216.15.12 84.9.164.252 31225

1 2020-08-26 07:08:30 78.199.217.198 66.191.137.154 17245

2 2022-11-13 08:23:25 63.79.210.48 198.219.82.17 16811

3 2023-07-02 10:38:46 163.42.196.10 101.228.192.255 20018

4 2023-07-16 13:11:07 71.166.185.76 189.243.174.238 6131

Destination Port Protocol Packet Length Packet Type Traffic Type \

0 17616 ICMP 503 Data HTTP

1 48166 ICMP 1174 Data HTTP

2 53600 UDP 306 Control HTTP

3 32534 UDP 385 Data HTTP

4 26646 TCP 1462 Data DNS

Payload Data ... Action Taken \

0 Qui natus odio asperiores nam. Optio nobis ius... ... Logged

1 Aperiam quos modi officiis veritatis rem. Omni... ... Blocked

2 Perferendis sapiente vitae soluta. Hic delectu... ... Ignored

3 Totam maxime beatae expedita explicabo porro l... ... Blocked

4 Odit nesciunt dolorem nisi iste iusto. Animi v... ... Blocked

Severity Level User Information \

0 Low Reyansh Dugal

1 Low Sumer Rana

2 Low Himmat Karpe

3 Medium Fateh Kibe

4 Low Dhanush Chad

Device Information Network Segment \

0 Mozilla/5.0 (compatible; MSIE 8.0; Windows NT ... Segment A

1 Mozilla/5.0 (compatible; MSIE 8.0; Windows NT ... Segment B

2 Mozilla/5.0 (compatible; MSIE 9.0; Windows NT ... Segment C

3 Mozilla/5.0 (Macintosh; PPC Mac OS X 10\_11\_5; ... Segment B

4 Mozilla/5.0 (compatible; MSIE 5.0; Windows NT ... Segment C

Geo-location Data Proxy Information Firewall Logs IDS/IPS Alerts \

0 Jamshedpur, Sikkim 150.9.97.135 Log Data NaN

1 Bilaspur, Nagaland NaN Log Data NaN

2 Bokaro, Rajasthan 114.133.48.179 Log Data Alert Data

3 Jaunpur, Rajasthan NaN NaN Alert Data

4 Anantapur, Tripura 149.6.110.119 NaN Alert Data

Log Source

0 Server

1 Firewall

2 Firewall

3 Firewall

4 Firewall

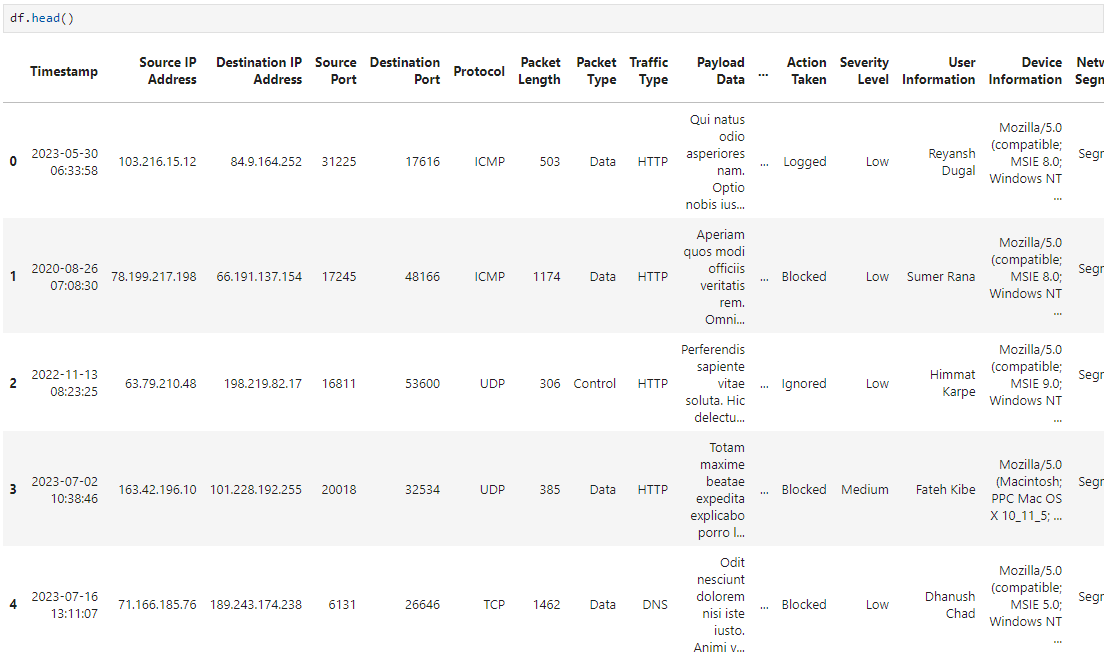
[5 rows x 25 columns]

**La salida muestra las primeras cinco filas del DataFrame cargado con los datos**.

* Aquí te describo los valores:
* **Timestamp:** La fecha y hora del ataque.
* **Source IP Address**: Dirección IP de origen del ataque.
* **Destination IP Address:** Dirección IP de destino del ataque.
* **Source Port:** Puerto de origen usado.
* **Destination Port:** Puerto de destino usado.
* **Protocol:** Protocolo de comunicación (ICMP, UDP, TCP).
* **Packet Length:** Longitud del paquete en bytes.
* **Packet Type:** Tipo de paquete (Data, Control).
* **Traffic Type:** Tipo de tráfico (HTTP, DNS).
* **Payload Data:** Datos del paquete (resumidos para muestra).
* **Action Taken:** Acción tomada (Logged, Blocked, Ignored).
* **Severity Level:** Nivel de severidad del ataque (Low, Medium).
* **User Information:** Información del usuario asociado.
* **Device Information:** Información del dispositivo (basado en agente de usuario).
* **Network Segment:** Segmento de red del ataque.
* **Geo-location Data:** Ubicación geográfica del origen del ataque.
* **Proxy Information:** Información de proxy si está disponible.
* **Firewall Logs:** Registros del firewall.
* **IDS/IPS Alerts:** Alertas del sistema de detección/prevention de intrusos.
* **Log Source:** Fuente del registro (Server, Firewall).

Eso es solo una primera pasada a las columnas. Cada columna tiene datos importantes para analizar los ciberataques y tomar acciones apropiadas.   
  
**El siguiente que uso es:**

**df.head()**: Similar al anterior, muestra las primeras cinco filas del DataFrame **df**, pero sin el uso de **print()**. Si lo estás usando en un entorno interactivo como Jupyter Notebook, se mostrarán directamente las primeras cinco filas sin necesidad de **print().**



print(df.describe())

print(df.dtypes)

**Descripción:**

* **print(df.describe())**

Muestra estadísticas descriptivas de las columnas numéricas en el DataFrame df. Explico cada fila de la salida:

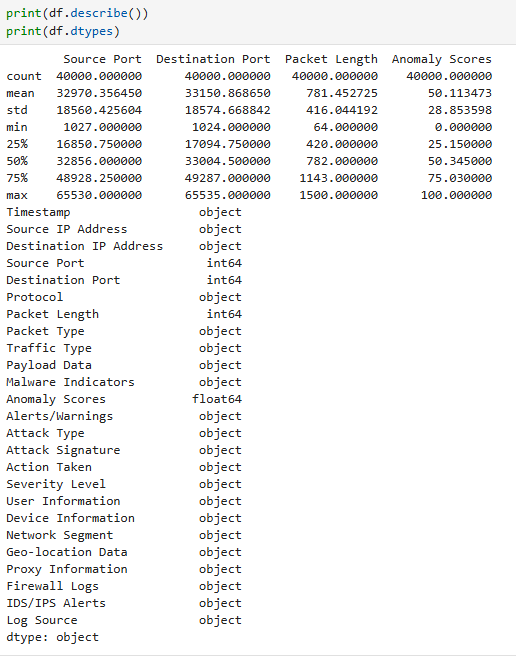
1. **count**: Número de valores no nulos en cada columna.
2. **mean**: Promedio de los valores en cada columna.
3. **std**: Desviación estándar de los valores en cada columna.
4. **min**: Valor mínimo en cada columna.
5. **25%**: Percentil 25 (primer cuartil) de los valores en cada columna.
6. **50%**: Percentil 50 (mediana) de los valores en cada columna.
7. **75%**: Percentil 75 (tercer cuartil) de los valores en cada columna.
8. **max**: Valor máximo en cada columna.

* print(df.dtypes)

muestra los tipos de datos de cada columna del DataFrame df. Es útil para identificar qué columnas son categóricas, numéricas, de texto, etc.

* + **Tipos de Datos Comunes:**

1. **object**: Cadenas de texto.
2. **int64**: Números enteros.
3. **float64**: Números decimales.
4. **bool**: Valores booleanos (True/False).
5. **datetime64**: Fechas y horas



sns.heatmap(df.isnull(), cbar=False, cmap='viridis')

plt.show()

Imagen que contiene Gráfico

Descripción generada automáticamente

Explicación:

* **sns.heatmap(df.isnull(), cbar=False, cmap='viridis'):**
* **sns.heatmap:** Esta función de Seaborn crea un gráfico de calor (heatmap).
* **df.isnull():** Esta parte del código devuelve un DataFrame con valores booleanos (**True** para valores **nulos**, False para valores no nulos).
* **cbar=False:** Esta opción desactiva la barra de color a la derecha del gráfico.
* **cmap='viridis'**: Esta opción establece la paleta de colores del gráfico a 'viridis', que es un gradiente de colores atractivos que va del morado al amarillo.

Esto se hace para Visualizar la distribución de valores nulos en el DataFrame df. Las celdas con valores True (nulos) serán resaltadas en el gráfico, lo que permitirá identificar fácilmente las columnas y filas con datos faltantes.

* + **plt.show(**): Esta función de Matplotlib muestra el gráfico generado en la pantalla. Es importante para visualizar cualquier gráfico creado con Seaborn o Matplotlib.

Este grafico nos puede valer para identificar patrones en los valores nulos dentro del **df**, se puede ver si hay columnas enteras filas con muchos valores faltantes, así se puede predecir como manejar eso datos en el procesamiento.

df.hist(bins=50, figsize=(20,15))

plt.show()

Gráfico, Histograma

Descripción generada automáticamente

1. **Explicación:**
   * **df.hist():** Esta función de Pandas crea histogramas para cada columna numérica en tu DataFrame df.
   * **bins=50:** Especifica el número de contenedores (bins) a utilizar para cada histograma. Más contenedores proporcionan más detalle, menos contenedores suavizan la distribución.
   * **figsize=(20,15):** Establece el tamaño de la figura para los histogramas. En este caso, 20 unidades de ancho y 15 unidades de alto, proporcionando una visualización más grande y clara.

Todo esto es para visualizar la distribución de valores en cada columna numérica, permitiéndome identificar patrones, tendencias y posibles outliers en los datos.

* + **plt.show():** Esta función de Matplotlib muestra el gráfico generado en la pantalla. Es para visualizar cualquier gráfico creado con Pandas o Matplotlib.

df\_sample = df.sample(frac=0.05, random\_state=42)

# 5% del dataset original

print(df\_sample.head())

1. **Tomo una muestra del dataset:**
   * **df.sample(frac=0.05, random\_state=42):**
     + **frac=0.05:** Indica que estoy tomando una muestra del 5% del DataFrame original (**df**).
     + **random\_state=42:** Establece una semilla para el generador de números aleatorios, asegura que cada vez que se ejecute el código, se obtenga la misma muestra, útil para la reproducibilidad.
   * **df\_sample:** Almacena el DataFrame con el 5% de las filas seleccionadas aleatoriamente.

Tomo una muestra del Dataframe porque para mi ordenador el conjunto de los datos son muy grandes, así permite realizar análisis y pruebas rápidas sin tener que procesar todo el conjunto de datos, además al usar la semilla asegura que los resultados sean consistentes cada vez que s ejecuta el código.

Timestamp Source IP Address Destination IP Address \

32823 2022-03-21 14:15:51 65.126.89.188 48.167.47.144

16298 2022-03-24 00:20:34 20.97.15.71 129.157.131.193

28505 2022-01-28 11:55:36 7.27.208.195 103.46.68.72

6689 2023-06-13 02:16:32 160.157.150.216 207.176.114.6

26893 2021-12-04 19:40:16 93.229.203.116 181.60.4.157

Source Port Destination Port Protocol Packet Length Packet Type \

32823 59855 56736 UDP 1402 Control

16298 47259 27663 ICMP 855 Control

28505 1659 19097 ICMP 297 Data

6689 10171 48022 ICMP 1090 Control

26893 61461 13536 TCP 330 Control

Traffic Type Payload Data ... \

32823 HTTP Explicabo voluptatem veniam. Qui sequi accusan... ...

16298 FTP Ipsa dignissimos possimus quisquam. Laudantium... ...

28505 DNS Ea voluptates temporibus iste sunt rem. Optio ... ...

6689 DNS Similique sint doloremque adipisci.\nUt expedi... ...

26893 FTP Provident quidem saepe ullam. Inventore quod s... ...

Action Taken Severity Level User Information \

32823 Logged Medium Ayesha Khurana

16298 Logged Medium Kimaya Thaker

28505 Logged High Rhea Deshmukh

6689 Logged High Kaira Choudhary

26893 Ignored High Neysa Bakshi

Device Information Network Segment \

32823 Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_6\_7 ... Segment B

16298 Opera/8.63.(X11; Linux i686; hne-IN) Presto/2.... Segment B

28505 Opera/9.32.(X11; Linux x86\_64; nn-NO) Presto/2... Segment A

6689 Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_9\_1)... Segment C

26893 Opera/9.48.(Windows NT 5.01; ja-JP) Presto/2.9... Segment C

Geo-location Data Proxy Information Firewall Logs \

32823 Begusarai, Tripura 38.89.175.22 NaN

16298 Jabalpur, Jharkhand NaN Log Data

28505 Asansol, Assam 3.113.185.207 Log Data

6689 Morena, Arunachal Pradesh NaN NaN

26893 Kirari Suleman Nagar, Manipur 146.14.175.94 NaN

IDS/IPS Alerts Log Source

32823 NaN Firewall

16298 NaN Server

28505 Alert Data Server

6689 NaN Firewall

26893 NaN Firewall

[5 rows x 25 columns]

Aunque podría parecer similar a **print(df.head())**, podemos ver que son similares en estructura, **print(df\_sample.head())** contienen los mismos nombres de columnas, los datos de las filas son diferentes en función de la muestra aleatoria tomada.

# Eliminación de columnas con alto porcentaje de valores nulos

threshold = 0.3

df = df[df.columns[df.isnull().mean() < threshold]]

**Explicación:**

1. **Definir el Umbral para los Valores Nulos**:
   * **threshold = 0.3**: Aquí defines que el umbral máximo permitido de valores nulos en una columna es del 30%. Esto significa que cualquier columna que tenga más del 30% de sus valores como nulos será eliminada.
2. **Eliminar Columnas con Alto Porcentaje de Valores Nulos**:
   * **df.isnull():** Genera un DataFrame del mismo tamaño con valores booleanos (True para valores nulos, False para valores no nulos).
   * **df.isnull().mean():** Calcula el porcentaje de valores nulos en cada columna.
   * **df.columns[df.isnull().mean() < threshold]:** Filtra las columnas que tienen un porcentaje de valores nulos menor que el umbral definido (30% en este caso).
   * **df = df[...]:** Actualiza el DataFrame df para que solo incluya las columnas que pasaron el filtro.

Este proceso es parte del preprocesamiento de datos, Eliminar las columnas con un alto porcentaje de valores nulos puede mejorar la calidad de los datos y la eficiencia del análisis, ya que las columnas con demasiados valores faltantes pueden no ser útiles para los modelos de machine learning.

print(df.isnull().sum())

Texto

Descripción generada automáticamente con confianza baja

Es para análisis exploratorio de datos, conocer la cantidad de valores nulos en cada columna te permite ver que columna necesita preprocesamiento adicional, como imputación de valores faltantes que es una técnica que se utiliza para rellenar o reemplazar los valores nulos (faltantes) eso es para que el algoritmo pueda trabajar correctamente ya que muchos de ellos no pueden manejar datos faltantes de manera eficiente. también esta la eliminación si tienen demasiados nulos.

**Explicación:**

* **df.isnull():**
  + Este comando genera un DataFrame del mismo tamaño que **df,** pero con valores booleanos: **True** si el valor es nulo, **False** si no lo es.
* **.sum():**
  + Al aplicar **.sum()** a este DataFrame de booleanos, se calcula la suma de valores **True** (nulos) para cada columna. El resultado es una Serie en Pandas donde los índices son los nombres de las columnas y los valores son el número de nulos en cada columna.
* **print(...):**
  + Imprime el resultado en la consola para que puedas ver cuántos valores nulos hay en cada columna de tu DataFrame **df.**

print(df\_sample.columns)

Salida:

Index(['Timestamp', 'Source IP Address', 'Destination IP Address',

'Source Port', 'Destination Port', 'Protocol', 'Packet Length',

'Packet Type', 'Traffic Type', 'Payload Data', 'Malware Indicators',

'Anomaly Scores', 'Alerts/Warnings', 'Attack Type', 'Attack Signature',

'Action Taken', 'Severity Level', 'User Information',

'Device Information', 'Network Segment', 'Geo-location Data',

'Proxy Information', 'Firewall Logs', 'IDS/IPS Alerts', 'Log Source'],

dtype='object')

# Separar características y etiquetas

X\_sample = df\_sample.drop(['Action Taken', 'Severity Level', 'Attack Type'], axis=1)

y\_sample = df\_sample['Action Taken']

# División en conjuntos de entrenamiento y prueba

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_sample, y\_sample, test\_size=0.3, random\_state=42)

Vamos a desglosarlo:

1. **Separar características y etiquetas:**

* **Características** (X\_sample):
  + df\_sample.drop(['Action Taken', 'Severity Level', 'Attack Type'], axis=1): Aquí estás eliminando las columnas 'Action Taken', 'Severity Level' y 'Attack Type' del DataFrame df\_sample. Lo que queda son todas las demás columnas que usas como características (atributos) para el modelo.
* **Etiquetas** (y\_sample):

**df\_sample['Action Taken']**: Aquí seleccionas únicamente la columna 'Action Taken' como tu etiqueta (variable objetivo) que quieres predecir.

1. **División en conjuntos de entrenamiento y prueba:**

* **train\_test\_split(X\_sample, y\_sample, test\_size=0.3, random\_state=42):**
  + Características de entrenamiento (**X\_train**) y prueba (**X\_test**)
  + Etiquetas de entrenamiento (**y\_train**) y prueba (**y\_test**)
* **test\_size=0.3**: Indica que el 30% de los datos se utilizarán para el conjunto de prueba y el 70% para el entrenamiento.
* **random\_state=42**: Establece una semilla para asegurar que la división sea reproducible cada vez que se ejecute el código.

Es para evaluar el rendimiento del modelo. El conjunto de entrenamiento se usa para entrenar el modelo, y el conjunto de prueba se utiliza para evaluar cómo se desempeña en datos que no ha visto antes.

# Identificar columnas no numéricas

non\_numeric\_columns = df\_sample.select\_dtypes(exclude=[np.number]).columns

print(non\_numeric\_columns)

* **Identificar columnas no numéricas**:
  + **df\_sample.select\_dtypes(exclude=[np.number])**: Esto selecciona todas las columnas del DataFrame **df\_sample** que no sean de tipo numérico (excluyendo **np.number**).
  + **.columns:** Obtiene los nombres de las columnas seleccionadas.
  + **non\_numeric\_columns:** Almacena los nombres de las columnas no numéricas.
* **Imprimir los nombres de las columnas no numéricas**
* **print(non\_numeric\_columns)**
  + - Muestra los nombres de las columnas que no son de tipo numérico. Esto es útil para identificar las columnas que pueden requerir un tratamiento especial, como codificación a variables dummy, antes de realizar operaciones numéricas o de escalado.

Index(['Timestamp', 'Source IP Address', 'Destination IP Address', 'Protocol',

'Packet Type', 'Traffic Type', 'Payload Data', 'Malware Indicators',

'Alerts/Warnings', 'Attack Type', 'Attack Signature', 'Action Taken',

'Severity Level', 'User Information', 'Device Information',

'Network Segment', 'Geo-location Data', 'Proxy Information',

'Firewall Logs', 'IDS/IPS Alerts', 'Log Source'],

dtype='object')

Esto ayuda a diferenciar entre columnas numéricas y no numéricas en el DataFrame. Es un paso importante para el preprocesamiento de datos, asegurando que sabemos qué tipo de tratamiento aplicar a cada tipo de columna.

# Codificación de variables categóricas

df = pd.get\_dummies(df, drop\_first=True)

Explicación:

Toma las columnas categóricas del df y las transforma en una serie de columnas binarias (0s y 1s). Cada columna representará una categoría de la variable original.

Al establecer **drop\_first=True**, elimino la primera categoría de cada variable categórica. Esto ayuda a evitar la **trampa de la variable ficticia** (dummy variable trap), una situación en la que las variables dummy son perfectas multicolineales, lo que puede causar problemas en modelos de regresión.

print(df.head())

El comando **print(df.head())** en pandas muestra las primeras 5 filas del DataFrame df. Es una forma rápida de ver una muestra de los datos para entender su estructura y contenido.

Salida:

Source Port Destination Port Packet Length Anomaly Scores \

0 31225 17616 503 28.67

1 17245 48166 1174 51.50

2 16811 53600 306 87.42

3 20018 32534 385 15.79

4 6131 26646 1462 0.52

Timestamp\_2020-01-01 01:14:07 Timestamp\_2020-01-01 03:02:42 \

0 False False

1 False False

2 False False

3 False False

4 False False

Timestamp\_2020-01-01 03:11:48 Timestamp\_2020-01-01 03:29:20 \

0 False False

1 False False

2 False False

3 False False

4 False False

Timestamp\_2020-01-01 04:19:53 Timestamp\_2020-01-01 04:30:19 ... \

0 False False ...

1 False False ...

2 False False ...

3 False False ...

4 False False ...

Geo-location Data\_Yamunanagar, Punjab \

0 False

1 False

2 False

3 False

4 False

Geo-location Data\_Yamunanagar, Rajasthan \

0 False

1 False

2 False

3 False

4 False

Geo-location Data\_Yamunanagar, Sikkim \

0 False

1 False

2 False

3 False

4 False

Geo-location Data\_Yamunanagar, Tamil Nadu \

0 False

1 False

2 False

3 False

4 False

Geo-location Data\_Yamunanagar, Telangana \

0 False

1 False

2 False

3 False

4 False

Geo-location Data\_Yamunanagar, Tripura \

0 False

1 False

2 False

3 False

4 False

Geo-location Data\_Yamunanagar, Uttar Pradesh \

0 False

1 False

2 False

3 False

4 False

Geo-location Data\_Yamunanagar, Uttarakhand \

0 False

1 False

2 False

3 False

4 False

Geo-location Data\_Yamunanagar, West Bengal Log Source\_Server

0 False True

1 False False

2 False False

3 False False

4 False False

[5 rows x 233225 columns]

# Separar características y etiquetas

X = df.drop(['Log Source\_Server'], axis=1)

y = df['Log Source\_Server']

# Verificar X e y

print(X.head())

print(y.head())

* **Separar las características (X) y las etiquetas (y)**:
  + **X**: Contendrá todas las columnas del DataFrame **df** excepto la columna 'Log Source\_Server'. Esta línea utiliza **drop** para eliminar la columna especificada.
  + **y**: Contendrá únicamente la columna 'Log Source\_Server'. Aquí simplemente se selecciona esta columna del DataFrame **df.**
  + **print(X.head())**
* Esto muestra las primeras 5 filas de X, es decir, todas las columnas del DataFrame original excepto 'Log Source\_Server'.
* La salida de X muestra columnas como **Source Port**, **Destination Port**, **Packet Length**, **Anomaly Scores**, múltiples timestamps y datos de geolocalización. Esto confirma que la columna 'Log Source\_Server' ha sido eliminada correctamente.
  + **print(y.head())**
* Esto muestra las primeras 5 filas de y, que contienen los valores de la columna 'Log Source\_Server'.
* La salida de y muestra los valores booleanos (True o False) que indican si el registro proviene o no del servidor.

En la salida del comando vemos la separación entre la X y la Y es importante para el entrenamiento de los modelos de machine learning, “X” representa las características que utilizara para hacer predicciones y “Y” es la etiqueta que intenta predecir.

Salida:  
  
Source Port Destination Port Packet Length Anomaly Scores \

0 31225 17616 503 28.67

1 17245 48166 1174 51.50

2 16811 53600 306 87.42

3 20018 32534 385 15.79

4 6131 26646 1462 0.52

Timestamp\_2020-01-01 01:14:07 Timestamp\_2020-01-01 03:02:42 \

0 False False

1 False False

2 False False

3 False False

4 False False

Timestamp\_2020-01-01 03:11:48 Timestamp\_2020-01-01 03:29:20 \

0 False False

1 False False

2 False False

3 False False

4 False False

Timestamp\_2020-01-01 04:19:53 Timestamp\_2020-01-01 04:30:19 ... \

0 False False ...

1 False False ...

2 False False ...

3 False False ...

4 False False ...

Geo-location Data\_Yamunanagar, Odisha \

0 False

1 False

2 False

3 False

4 False

Geo-location Data\_Yamunanagar, Punjab \

0 False

1 False

2 False

3 False

4 False

Geo-location Data\_Yamunanagar, Rajasthan \

0 False

1 False

2 False

3 False

4 False

Geo-location Data\_Yamunanagar, Sikkim \

0 False

1 False

2 False

3 False

4 False

Geo-location Data\_Yamunanagar, Tamil Nadu \

0 False

1 False

2 False

3 False

4 False

Geo-location Data\_Yamunanagar, Telangana \

0 False

1 False

2 False

3 False

4 False

Geo-location Data\_Yamunanagar, Tripura \

0 False

1 False

2 False

3 False

4 False

Geo-location Data\_Yamunanagar, Uttar Pradesh \

0 False

1 False

2 False

3 False

4 False

Geo-location Data\_Yamunanagar, Uttarakhand \

0 False

1 False

2 False

3 False

4 False

Geo-location Data\_Yamunanagar, West Bengal

0 False

1 False

2 False

3 False

4 False

[5 rows x 233224 columns]

0 True

1 False

2 False

3 False

4 False

Name: Log Source\_Server, dtype: bool

# Eliminar columnas no numéricas

X\_sample = df\_sample.drop(non\_numeric\_columns, axis=1)

y\_sample = df\_sample['Action Taken']

# División en conjuntos de entrenamiento y prueba

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_sample, y\_sample, test\_size=0.3, random\_state=42)

**Explicación:**

1. **Eliminar Columnas No Numéricas**

**X\_sample = df\_sample.drop(non\_numeric\_columns, axis=1)**

* **Objetivo**: Remover columnas que no contienen datos numéricos del DataFrame **df\_sample.**
* **Método drop:**
  + **non\_numeric\_columns**: Se espera que sea una lista o un array de nombres de columnas que contienen datos no numéricos.
  + **axis=1:** Especifica que se están eliminando columnas (si fuera **axis=0**, se eliminarían filas).

1. **Asignación de Características y Etiquetas**

**y\_sample = df\_sample['Action Taken']**

* **Objetivo**: Asignar la columna 'Action Taken' como la etiqueta (variable dependiente) que el modelo intentará predecir.
* **Resultado**: y\_sample contendrá solo los valores de la columna 'Action Taken'.

1. **División en Conjuntos de Entrenamiento y Prueba**

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_sample, y\_sample, test\_size=0.3, random\_state=42)**

* **Objetivo**: Dividir los datos en conjuntos de entrenamiento y prueba.
* **Detalles del Método**train\_test\_split:
  + **X\_sample**: Características (variables independientes) después de eliminar las columnas no numéricas.
  + **y\_sample**: Etiquetas (variable dependiente) que se desea predecir.
  + **test\_size=0.3**: El 30% de los datos se utilizará como conjunto de prueba, mientras que el 70% se utilizará para entrenar el modelo.
  + **random\_state=42**: Semilla aleatoria para asegurar que la división sea reproducible.

1. **Resultado Final:**

* **X\_train**: Conjunto de características de entrenamiento.
* **X\_test**: Conjunto de características de prueba.
* **y\_train**: Conjunto de etiquetas de entrenamiento.
* **y\_test**: Conjunto de etiquetas de prueba.

Estas operaciones son para preparar los datos para el entrenamiento y la evaluación de un modelo de machine learning, asegurando que el modelo se entrene adecuadamente y que su rendimiento se pueda medir de manera efectiva.

print(df\_sample.columns)

**print(df\_sample.columns)** imprime una lista de todas las columnas en el DataFrame **df\_sample**. La salida que ves es un índice de estas columnas.

Index(['Timestamp', 'Source IP Address', 'Destination IP Address',

'Source Port', 'Destination Port', 'Protocol', 'Packet Length',

'Packet Type', 'Traffic Type', 'Payload Data', 'Malware Indicators',

'Anomaly Scores', 'Alerts/Warnings', 'Attack Type', 'Attack Signature',

'Action Taken', 'Severity Level', 'User Information',

'Device Information', 'Network Segment', 'Geo-location Data',

'Proxy Information', 'Firewall Logs', 'IDS/IPS Alerts', 'Log Source'],

dtype='object')

df\_sample = df.sample(frac=0.05, random\_state=42) # 5% del dataset original

print(df\_sample.head())

* + **frac=0.05:** Indica que estamos tomando una muestra del 5% del DataFrame original df.
  + **random\_state=42:** Establece una semilla para el generador de números aleatorios, asegurando que cada vez que se ejecute el código, se obtenga la misma muestra. Esto es útil para la reproducibilidad de los resultados.
  + **df\_sample:** Almacena el DataFrame con el 5% de las filas seleccionadas aleatoriamente del DataFrame original.
  + df\_sample.head(): Muestra las primeras cinco filas del DataFrame df\_sample.

Tomo una muestra del Dataframe porque para mi ordenador el conjunto de los datos son muy grandes, así permite realizar análisis y pruebas rápidas sin tener que procesar todo el conjunto de datos, además al usar la semilla asegura que los resultados sean consistentes cada vez que s ejecuta el código.   
  
Salida:

Timestamp Source IP Address Destination IP Address \

32823 2022-03-21 14:15:51 65.126.89.188 48.167.47.144

16298 2022-03-24 00:20:34 20.97.15.71 129.157.131.193

28505 2022-01-28 11:55:36 7.27.208.195 103.46.68.72

6689 2023-06-13 02:16:32 160.157.150.216 207.176.114.6

26893 2021-12-04 19:40:16 93.229.203.116 181.60.4.157

Source Port Destination Port Protocol Packet Length Packet Type \

32823 59855 56736 UDP 1402 Control

16298 47259 27663 ICMP 855 Control

28505 1659 19097 ICMP 297 Data

6689 10171 48022 ICMP 1090 Control

26893 61461 13536 TCP 330 Control

Traffic Type Payload Data ... \

32823 HTTP Explicabo voluptatem veniam. Qui sequi accusan... ...

16298 FTP Ipsa dignissimos possimus quisquam. Laudantium... ...

28505 DNS Ea voluptates temporibus iste sunt rem. Optio ... ...

6689 DNS Similique sint doloremque adipisci.\nUt expedi... ...

26893 FTP Provident quidem saepe ullam. Inventore quod s... ...

Action Taken Severity Level User Information \

32823 Logged Medium Ayesha Khurana

16298 Logged Medium Kimaya Thaker

28505 Logged High Rhea Deshmukh

6689 Logged High Kaira Choudhary

26893 Ignored High Neysa Bakshi

Device Information Network Segment \

32823 Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_6\_7 ... Segment B

16298 Opera/8.63.(X11; Linux i686; hne-IN) Presto/2.... Segment B

28505 Opera/9.32.(X11; Linux x86\_64; nn-NO) Presto/2... Segment A

6689 Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_9\_1)... Segment C

26893 Opera/9.48.(Windows NT 5.01; ja-JP) Presto/2.9... Segment C

Geo-location Data Proxy Information Firewall Logs \

32823 Begusarai, Tripura 38.89.175.22 NaN

16298 Jabalpur, Jharkhand NaN Log Data

28505 Asansol, Assam 3.113.185.207 Log Data

6689 Morena, Arunachal Pradesh NaN NaN

26893 Kirari Suleman Nagar, Manipur 146.14.175.94 NaN

IDS/IPS Alerts Log Source

32823 NaN Firewall

16298 NaN Server

28505 Alert Data Server

6689 NaN Firewall

26893 NaN Firewall

[5 rows x 25 columns]

# Eliminar columnas no numéricas

X\_sample = df\_sample.drop(non\_numeric\_columns, axis=1)

y\_sample = df\_sample['Action Taken']

# División en conjuntos de entrenamiento y prueba

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_sample, y\_sample, test\_size=0.3, random\_state=42)

1. **Eliminar columnas no numéricas:**
   * **X\_sample = df\_sample.drop(non\_numeric\_columns, axis=1)**
     + **df\_sample.drop(non\_numeric\_columns, axis=1)**: Elimina las columnas listadas en **non\_numeric\_columns**del DataFrame **df\_sample**. Esto deja solo las columnas numéricas en **X\_sample**.
     + **axis=1:** Indica que estamos eliminando columnas (y no filas).
2. **Separar características y etiquetas:**
   * **y\_sample = df\_sample['Action Taken']**
     + **df\_sample['Action Taken']**: Selecciona la columna 'Action Taken' como la variable objetivo o etiqueta que quieres predecir.
     + **X\_sample:** Contiene solo las características numéricas.
     + **y\_sample**: Contiene las etiquetas.
3. **División en conjuntos de entrenamiento y prueba:**
   * X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_sample, y\_sample, test\_size=0.3, random\_state=42)
     + train\_test\_split(X\_sample, y\_sample, test\_size=0.3, random\_state=42): Divide los datos en conjuntos de entrenamiento y prueba.
       1. test\_size=0.3: El 30% de los datos se utilizan para el conjunto de prueba, y el 70% restante para el entrenamiento.
       2. random\_state=42: Asegura que la división sea reproducible cada vez que se ejecute el código.

En resumen, este bloque de código prepara tus datos numéricos para el modelado, asegurando que solo las características relevantes (numéricas) se utilicen para el entrenamiento y evaluación del modelo.

# Configuración del tamaño del lote

chunk\_size = 500 # Ajusta este tamaño según la capacidad de tu memoria

# Inicializar scaler

scaler = MinMaxScaler()

# Para almacenar datos escalados

X\_train\_scaled = []

X\_test\_scaled = []

# Procesar datos en lotes

for i in range(0, len(X\_train), chunk\_size):

chunk = X\_train[i:i+chunk\_size]

X\_train\_scaled.append(scaler.fit\_transform(chunk))

for i in range(0, len(X\_test), chunk\_size):

chunk = X\_test[i:i+chunk\_size]

X\_test\_scaled.append(scaler.transform(chunk))

# Convertir listas a arrays numpy

X\_train = np.vstack(X\_train\_scaled)

X\_test = np.vstack(X\_test\_scaled)

# Verificar los primeros datos escalados

print(X\_train[:5])

print(X\_test[:5])

1. **Configuración del Tamaño del Lote**

**chunk\_size = 500** # Ajusta este tamaño según la capacidad de tu memoria

* **chunk\_size:** Define el tamaño de cada lote de datos que se procesará a la vez. Esto es útil para manejar grandes cantidades de datos sin sobrecargar la memoria.

1. **Inicializar el Scaler**

**scaler = MinMaxScaler()**

* **MinMaxScaler:** Escalador que normaliza los datos para que estén dentro de un rango específico (por defecto, entre 0 y 1).

1. **Almacenar Datos Escalados**

**X\_train\_scaled = []**

**X\_test\_scaled = []**

* Listas vacías para almacenar los datos escalados de entrenamiento y prueba.

1. **Procesar Datos en Lotes**

**for i in range(0, len(X\_train), chunk\_size):**

**chunk = X\_train[i:i+chunk\_size]**

**X\_train\_scaled.append(scaler.fit\_transform(chunk))**

**for i in range(0, len(X\_test), chunk\_size):**

**chunk = X\_test[i:i+chunk\_size]**

**X\_test\_scaled.append(scaler.transform(chunk))**

* **Bucle para Entrenamiento (**X\_train**)**:
  + Itera sobre X\_train en segmentos de tamaño chunk\_size.
  + chunk: Un segmento de X\_train del tamaño definido.
  + scaler.fit\_transform(chunk): Escala el segmento y ajusta el escalador.
  + X\_train\_scaled.append(...): Añade el segmento escalado a la lista X\_train\_scaled.
* **Bucle para Prueba (**X\_test**)**:
  + Similar al bucle de entrenamiento, pero usa scaler.transform(chunk) en lugar de fit\_transform para aplicar la misma escala ajustada a los datos de prueba.

1. **Convertir Listas a Arrays NumPy**

X\_train = np.vstack(X\_train\_scaled)

X\_test = np.vstack(X\_test\_scaled)

* np.vstack: Combina todos los segmentos de datos escalados en un  solo array NumPy.

1. **Verificar los Datos Escalados**

**print(X\_train[:5])**

**print(X\_test[:5])**

* Muestra las primeras 5 filas de los datos de entrenamiento y prueba escalados.

Este código es útil para manejar y escalar grandes conjuntos de datos en lotes, lo que mejora la eficiencia y evita problemas de memoria.

[[0.80104284 0.46719543 0.43675751 0.47974607]

[0.30924111 0.31534915 0.3661775 0.45838372]

[0.5566497 0.88417899 0.66037736 0.84995969]

[0.18358148 0.62180452 0.52690426 0.64933495]

[0.29317439 0.72314402 0.63661775 0.08958081]]

[[0.93705967 0.02893576 0.71179344 0.13224547]

[0.61246676 0.00178616 0.92951849 0.39613575]

[0.40524736 0.90739935 0.24633636 0.52337571]

[0.16053127 0.76206822 0.96441033 0.18440284]

[0.6745517 0.82997329 0.56106071 0.25558114]]

1. **Interpretación de los Valores**

* **Valores Normalizados**: Todos los valores están escalados entre 0 y 1. Esto facilita que los algoritmos de machine learning funcionen de manera más eficiente.
* **Columnas Representadas**: Cada número en la matriz corresponde a una característica específica del conjunto de datos original. Por ejemplo, la primera columna puede representar Source Port, la segunda Destination Port, y así sucesivamente.

1. **Importancia del Escalado**

* **Homogeneidad**: Asegura que ninguna característica domine sobre las otras debido a una escala mayor.
* **Mejora del Rendimiento del Modelo**: Los algoritmos de machine learning, como las redes neuronales y los modelos de regresión, tienden a converger más rápido y a obtener mejores resultados cuando las características están en la misma escala.

He escalado los datos en lotes para evitar sobrecargar la memoria y los he convertido a un formato que los modelos de machine learning pueden interpretar fácilmente.

# Modelo de Random Forest

rf\_model = RandomForestClassifier(n\_estimators=100, random\_state=42)

rf\_model.fit(X\_train, y\_train)

# Predicciones

y\_pred = rf\_model.predict(X\_test)

# Evaluación

print(confusion\_matrix(y\_test, y\_pred))

print(classification\_report(y\_test, y\_pred))

print(f'Accuracy: {accuracy\_score(y\_test, y\_pred)}')

**Explicación:**

**Configuración y Entrenamiento del Modelo Random Forest**

**rf\_model = RandomForestClassifier(n\_estimators=100, random\_state=42)**

**rf\_model.fit(X\_train, y\_train)**

1. **Instanciar el Modelo**:
   * RandomForestClassifier(n\_estimators=100, random\_state=42): Crea un modelo de Random Forest con 100 árboles y una semilla aleatoria fija para reproducibilidad.
2. **Entrenamiento del Modelo**:
   * rf\_model.fit(X\_train, y\_train): Entrena el modelo con los datos de entrenamiento X\_train y y\_train.

**Predicciones**

**y\_pred = rf\_model.predict(X\_test)**

1. **Hacer Predicciones**:
   * rf\_model.predict(X\_test): Usa el modelo entrenado para hacer predicciones sobre los datos de prueba X\_test.

**Evaluación del Modelo**

**print(confusion\_matrix(y\_test, y\_pred))**

**print(classification\_report(y\_test, y\_pred))**

**print(f'Accuracy: {accuracy\_score(y\_test, y\_pred)}')**

1. **Evaluación del Modelo**:
   * confusion\_matrix(y\_test, y\_pred): Muestra la matriz de confusión, que resume el rendimiento de un algoritmo de clasificación.
   * classification\_report(y\_test, y\_pred): Proporciona un informe detallado de los principales indicadores de rendimiento como la precisión, el recall y el F1-score para cada clase.
   * accuracy\_score(y\_test, y\_pred): Calcula la precisión global del modelo.

**Salida:**

**Matriz de Confusión:**

**[[66 61 86]**

**[64 51 68]**

**[64 68 72]]**

* **Filas**: Representan las verdaderas clases (Blocked, Ignored, Logged).
* **Columnas**: Representan las clases predichas.
* Ejemplo: La primera fila muestra que de 213 casos reales de "Blocked", el modelo predijo 66 correctamente como "Blocked", 61 incorrectamente como "Ignored", y 86 incorrectamente como "Logged".

**Informe de Clasificación:**

plaintext

precision recall f1-score support

Blocked 0.34 0.31 0.32 213

Ignored 0.28 0.28 0.28 183

Logged 0.32 0.35 0.33 204

accuracy 0.32 600

macro avg 0.31 0.31 0.31 600

weighted avg 0.32 0.32 0.31 600

* **Precision**: La proporción de verdaderos positivos entre los positivos predichos.
* **Recall**: La proporción de verdaderos positivos entre los positivos reales.
* **F1-score**: La media armónica de la precisión y el recall.
* **Support**: El número de ocurrencias reales de cada clase.

**Precisión del Modelo:**

plaintext

Accuracy: 0.315

* **Accuracy**: Proporción de predicciones correctas respecto al total de casos. En este caso, el modelo tiene una precisión del 31.5%.

El modelo de Random Forest ha sido configurado, entrenado y evaluado. Aunque la precisión del modelo no es muy alta (31.5%), con

esta información es esencial para realizar mejoras en el modelo, como ajustar hiperparámetros, balancear las clases, o utilizar diferentes técnicas de preprocesamiento de datos.

[[66 61 86]

[64 51 68]

[64 68 72]]

precision recall f1-score support

Blocked 0.34 0.31 0.32 213

Ignored 0.28 0.28 0.28 183

Logged 0.32 0.35 0.33 204

accuracy 0.32 600

macro avg 0.31 0.31 0.31 600

weighted avg 0.32 0.32 0.31 600

Accuracy: 0.315

las métricas clave:

1. **Matriz de Confusión:**

plaintext

Copiar

[[66 61 86]

[64 51 68]

[64 68 72]]

* + Filas: Predicciones
  + Columnas: Valores reales
  + Celdas: Número de observaciones

1. **Métricas de Evaluación para cada Clase (Blocked, Ignored, Logged):**
   * **Precisión:** Proporción de predicciones correctas de una clase respecto al total de predicciones hechas para esa clase.
     + Blocked: 0.34
     + Ignored: 0.28
     + Logged: 0.32
   * **Recall:** Proporción de casos verdaderos de una clase que han sido correctamente identificados por el modelo.
     + Blocked: 0.31
     + Ignored: 0.28
     + Logged: 0.35
   * **F1-Score:** Promedio armónico entre precisión y recall, ideal para medir el balance.
     + Blocked: 0.32
     + Ignored: 0.28
     + Logged: 0.33
   * **Soporte:** Número de instancias reales en cada clase.
     + Blocked: 213
     + Ignored: 183
     + Logged: 204
2. **Métricas Globales:**
   * **Accuracy (Precisión Global):** 0.32 (32% de las predicciones fueron correctas).
   * **Macro Avg:** Promedio simple de precisión, recall y F1-Score.
   * **Weighted Avg:** Promedio ponderado de precisión, recall y F1-Score según el soporte de cada clase.

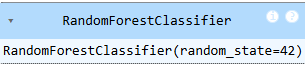
El modelo tiene un **32% de precisión** en sus predicciones globalmente. No es el mejor resultado, pero me da una idea clara de dónde mejorar.

rf\_model = RandomForestClassifier(n\_estimators=100, random\_state=42)

rf\_model.fit(X\_train, y\_train)

* RandomForestClassifier(n\_estimators=100, random\_state=42): Crea un modelo de clasificación llamado Random Forest con 100 árboles (estimators) y un estado aleatorio fijo para obtener resultados reproducibles.
* rf\_model.fit(X\_train, y\_train): Entrena el modelo con mis datos de entrenamiento (X\_train para las características y y\_train para las etiquetas).

Salida



# Importa las métricas ROC AUC y las funciones para calcular la curva ROC

**from sklearn.metrics import roc\_auc\_score, roc\_curve, auc**

**import matplotlib.pyplot as plt** # Para la visualización de datos

**from sklearn.preprocessing import label\_binarize** # Para convertir etiquetas en una forma binarizada

# Identifica todas las clases únicas en tus datos de entrenamiento y prueba

**all\_classes = np.unique(np.concatenate((y\_train, y\_test)))**

# Binariza las etiquetas de entrenamiento y prueba, es decir, las convierte en una matriz de valores binarios

**y\_train\_binarized = label\_binarize(y\_train, classes=all\_classes)**

**y\_test\_binarized = label\_binarize(y\_test, classes=all\_classes)**

# Predice las probabilidades de las clases para los datos de prueba

**y\_pred\_prob = rf\_model.predict\_proba(X\_test)**

# Calcula el área bajo la curva ROC (ROC AUC) para las predicciones

**roc\_auc = roc\_auc\_score(y\_test, y\_pred\_prob, multi\_class='ovr')**

# Imprime el valor de ROC AUC

**print(f'ROC-AUC: {roc\_auc}')**

salida:

ROC-AUC: 0.4963540251320652

El valor de **ROC-AUC de 0.4963540251320652** indica que el modelo de Random Forest no está rindiendo mucho mejor que una predicción aleatoria, generalmente:

Un valor de ROC-AUC de 0.5 sugiere que el modelo tiene la misma capacidad de discriminación que un sistema aleatorio.

* Valores por encima de 0.5 indican que el modelo tiene cierta capacidad predictiva.
* Valores cercanos a 1.0 indican un excelente desempeño del modelo.

El resultado de ~0.496 sugiere que puede haber problemas en el modelo o en la preparación de datos.

all\_classes = np.unique(np.concatenate((y\_train, y\_test)))

y\_train\_binarized = label\_binarize(y\_train, classes=all\_classes)

y\_test\_binarized = label\_binarize(y\_test, classes=all\_classes)

* **all\_classes = np.unique(np.concatenate((y\_train, y\_test))):** Este paso encuentra todas las clases únicas presentes en los datos de entrenamiento (**y\_train**) y prueba (**y\_test**). **np.concatenate** junta ambos conjuntos de etiquetas, y np.unique identifica las clases únicas.
* **y\_train\_binarized = label\_binarize(y\_train, classes=all\_classes)**: Este comando convierte las etiquetas de entrenamiento en una forma binarizada, es decir, en una matriz donde cada columna representa una clase y contiene 1 o 0 dependiendo de si la clase está presente o no.
* **y\_test\_binarized = label\_binarize(y\_test, classes=all\_classes)**: Hace lo mismo que el paso anterior, pero con las etiquetas de prueba.

La binarización es útil para la evaluación de modelos en problemas multiclase, permitiendo calcular métricas como la curva ROC de manera más efectiva.

from sklearn.metrics import auc

importa la función auc de la librería scikit-learn, que es una métrica utilizada para calcular el área bajo la curva (AUC) para una curva ROC.

y\_pred\_prob = rf\_model.predict\_proba(X\_test)

# Inicializar diccionarios

fpr = dict()

tpr = dict()

roc\_auc = dict()

# Calcular AUC para cada clase

for i in range(y\_test\_binarized.shape[1]):

if np.sum(y\_test\_binarized[:, i]) == 0:

continue # Omite clases sin muestras positivas

fpr[i], tpr[i], \_ = roc\_curve(y\_test\_binarized[:, i], y\_pred\_prob[:, i])

roc\_auc[i] = auc(fpr[i], tpr[i])

# Verificar que los diccionarios tienen datos válidos

print(fpr)

print(tpr)

print(roc\_auc)

**Explicación:**

1. **y\_pred\_prob = rf\_model.predict\_proba(X\_test)**: Se predicen las probabilidades de clase para los datos de prueba.
2. **fpr, tpr, roc\_auc**: Se inicializan diccionarios para almacenar las tasas de falsos positivos (FPR), tasas de verdaderos positivos (TPR) y los valores de Área Bajo la Curva (AUC) para cada clase.
3. **for i in range(y\_test\_binarized.shape[1]):** Se recorre cada clase binarizada.
4. **if np.sum(y\_test\_binarized[:, i]) == 0: continue**: Si no hay muestras positivas para una clase específica, se omite esa clase.
5. **fpr[i], tpr[i], \_ = roc\_curve(y\_test\_binarized[:, i], y\_pred\_prob[:, i])**: Se calculan las tasas de falsos positivos y verdaderos positivos para la clase i.
6. **roc\_auc[i] = auc(fpr[i], tpr[i])**: Se calcula el área bajo la curva ROC para la clase i.
7. **print(fpr), print(tpr), print(roc\_auc)**: Se imprimen los diccionarios con las tasas y los valores de AUC.

Este código ayuda a evaluar el rendimiento del modelo en términos de su capacidad para distinguir entre diferentes clases.

plt.figure()

colors = ['blue', 'red', 'green']

for i, color in zip(roc\_auc.keys(), colors):

plt.plot(fpr[i], tpr[i], color=color, lw=2, label=f'ROC curve of class {i} (area = {roc\_auc[i]:.2f})')

plt.plot([0, 1], [0, 1], 'k--', lw=2, label='Random chance')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic (ROC) for Multi-class')

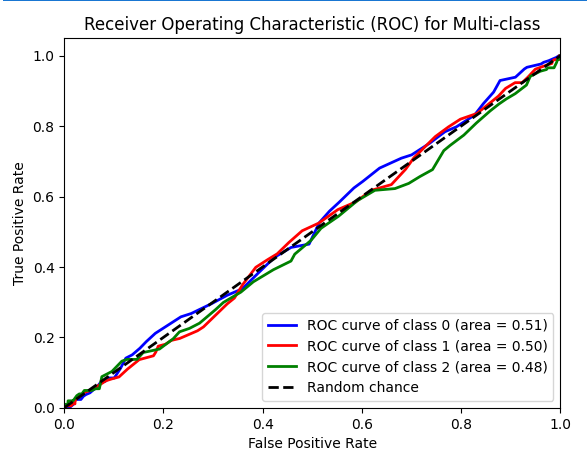
plt.legend(loc='lower right')

plt.show()

1. **plt.figure()**: Crea una nueva figura para la gráfica.
2. **colors = ['blue', 'red', 'green']**: Define una lista de colores para diferenciar las clases en la gráfica.
3. **for i, color in zip(roc\_auc.keys(), colors)**: Recorre cada clase junto con su color correspondiente. **roc\_auc.keys()** devuelve las clases, y **colors** los colores.
4. **plt.plot(fpr[i], tpr[i], color=color, lw=2, label=f'ROC curve of class {i} (area = {roc\_auc[i]:.2f})')**: Dibuja la curva ROC para la clase **i** con su color específico, línea de ancho **2** (**lw=2**), y una etiqueta que incluye el área bajo la curva (AUC).
5. **plt.plot([0, 1], [0, 1], 'k--', lw=2, label='Random chance')**: Dibuja una línea discontinua (k--) que representa una predicción aleatoria, con ancho de línea 2.
6. **plt.xlim([0.0, 1.0])** y **plt.ylim([0.0, 1.05]):** Establecen los límites de los ejes X e Y, respectivamente.
7. **plt.xlabel('False Positive Rate')** y **plt.ylabel('True Positive Rate'):** Añaden etiquetas descriptivas a los ejes X e Y.
8. **plt.title('Receiver Operating Characteristic (ROC) for Multi-class')**: Añade un título a la gráfica.
9. **plt.legend(loc='lower right')**: Coloca la leyenda en la esquina inferior derecha.
10. **plt.show()**: Muestra la gráfica en pantalla.

Esta visualización permite ver cómo de bien el modelo discrimina entre las clases en términos de sus tasas de falsos positivos y verdaderos positivos. Cada curva ROC representa una clase diferente con su área bajo la curva (AUC).

Salida:



Evaluación del Modelo

Curvas ROC y AUC

Tus resultados del clasificación multiclase no están funcionando de manera efectiva.

1. **Clase 0 (AUC = 0.51)**: Este resultado indica que el modelo tiene un desempeño apenas mejor que el azar para la clase 0.
2. **Clase 1 (AUC = 0.50)**: Un AUC de 0.50 implica que el modelo no está diferenciando correctamente entre las clases; su rendimiento es equivalente a una clasificación aleatoria.
3. **Clase 2 (AUC = 0.48)**: Con un AUC menor a 0.50, el modelo está rindiendo peor que un clasificador aleatorio para esta clase específica.

**Conclusión**

Durante esta práctica, aborde un problema de clasificación utilizando un conjunto de datos realista relacionado con ciberataques. Nuestro objetivo principal era predecir la columna "Action Taken". Para ello, seguimos los siguientes pasos:

1. Preparación de datos: Dividimos los datos en conjuntos de entrenamiento y prueba.

2. Análisis exploratorio: Realizamos una descripción básica de los datos y visualizamos los valores nulos.

3. Preprocesamiento: Eliminamos columnas con altos valores nulos, imputamos valores faltantes y codificamos variables categóricas.

4. Modelado: Entrenamos un modelo de Random Forest y evaluamos su rendimiento.

El modelo obtuvo una precisión de aproximadamente 32%, lo cual indica que hay margen de mejora. el proceso seguido demostró un enfoque metodológico adecuado en la preparación y modelado de datos, aunque los resultados indican la necesidad de optimizaciones adicionales. La baja precisión sugiere que el modelo podría beneficiarse de la optimización de hiperparámetros, la selección de características más relevantes y la ingeniería de nuevas características. La evaluación de otros algoritmos también podría proporcionar mejoras significativas en el rendimiento.  
  
Aunque el desempeño actual del modelo es limitado, el enfoque y las técnicas aplicadas son correctas y constituyen una base sólida para futuros trabajos y mejoras en el análisis de ciberataques mediante Machine Learning. Es importante la atención a los detalles en el preprocesamiento y la evaluación del modelo para avanzar en este tipo de problemas complejos.

He implementado una evaluación detallada del modelo de Random Forest utilizando la métrica ROC-AUC para cada clase en un problema de clasificación multiclase. El valor global de ROC-AUC obtenido fue 0.496, lo que indica un rendimiento inferior al esperado. La curva ROC y el área bajo la curva para cada clase son las siguientes: Clase 0 (AUC = 0.51) indica que el modelo apenas supera la predicción aleatoria, Clase 1 (AUC = 0.50) muestra que el modelo tiene un rendimiento equivalente a la predicción aleatoria, y Clase 2 (AUC = 0.48) revela que el modelo tiene un rendimiento inferior a la predicción aleatoria.

Visualizo las curvas ROC para cada clase, proporcionando una representación gráfica del rendimiento del modelo en términos de las tasas de verdaderos positivos y falsos positivos. Los valores de AUC para las tres clases indican un rendimiento subóptimo del modelo. Un AUC cercano a 0.50 sugiere que el modelo no es capaz de diferenciar eficazmente entre las clases y su rendimiento es comparable al de una clasificación aleatoria.

El bajo rendimiento del modelo puede deberse a varios factores. Es posible que el conjunto de datos no tenga suficientes muestras representativas para cada clase, o que las clases estén desequilibradas, lo cual afecta negativamente el rendimiento del modelo. También es probable que las características utilizadas no sean las más relevantes para la tarea de clasificación. La configuración actual del modelo de Random Forest podría no ser la óptima, por lo que la optimización de hiperparámetros podría mejorar los resultados.

Aunque los resultados actuales del modelo son limitados, he seguido un enfoque sólido y metodológico en la evaluación y visualización del rendimiento del modelo. Las mejoras adicionales pueden contribuir a obtener resultados más precisos y útiles en la detección de ciberataques mediante Machine Learning.