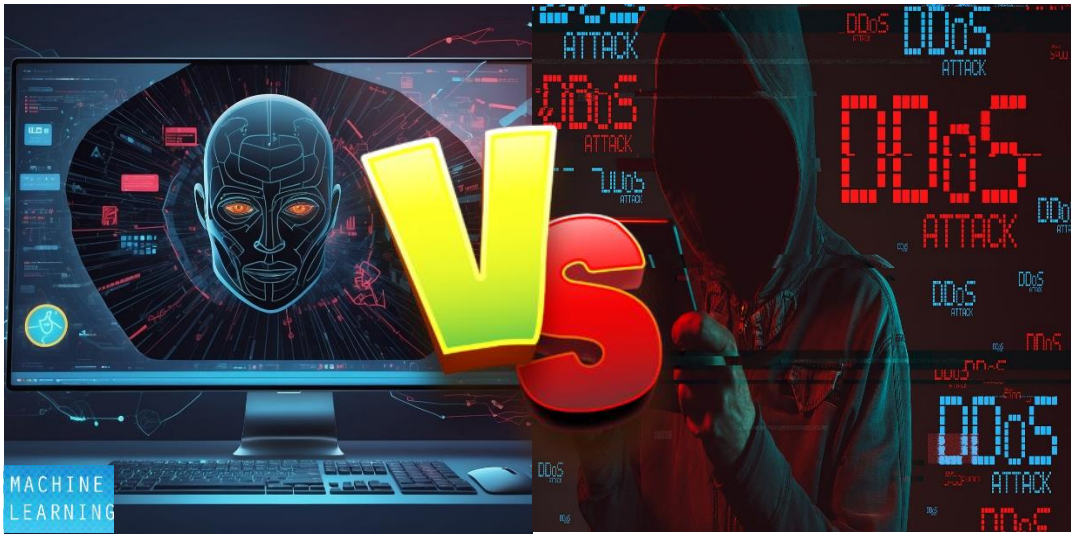
PREDICCIÓN DE ATAQUES DDOS CON ALGORITMOS DE APRENDIZAJE AUTOMÁTICO

Proyecto de Machine Learning



15 DE DICIEMBRE DE 2024

BOOTCAMP DATA SCIENCE - BILBAO
Por Jon Ameyugo Vázquez

Índice

Contenido

Introducción 2
Dataset 2
Archivo “analysis.py”.....2
Archivo EDA.py 3
Algoritmos de ML.....5
Consideraciones a tener en cuenta que se han introducido en el modelo final 7
Conclusiones.....8

Introducción

En la actualidad, los sistemas informáticos y las redes de comunicación son fundamentales para el funcionamiento de organizaciones y servicios críticos en todo el mundo. Sin embargo, este creciente nivel de dependencia también los hace vulnerables a diversas amenazas cibernéticas, siendo los ataques DDoS (Distribución de Denegación de Servicios) una de ellas.

Un ataque DDoS busca interrumpir la disponibilidad de un servicio o recurso en línea, generando una sobrecarga intencional de tráfico hacia un servidor, red o aplicación específica. Esto no solo afecta la experiencia del usuario final, sino que también genera pérdidas económicas significativas y compromete la confianza en la infraestructura tecnológica.

Detectar y mitigar ataques DDoS de manera efectiva requiere identificar patrones en el tráfico de red que diferencian un comportamiento normal de un malicioso. Es ahí donde entramos nosotros como futuros Data Scientists. Este documento tratará de seguir los pasos que se verán a continuación hasta llegar al objetivo en concreto partiendo solo de un dataset.

Dataset

Este dataset se puede localizar en Kaggle y es llamado “Friday-WorkingHours-Afternoon-DDos.pcap_ISCX.csv”. Este dataset, forma parte de un experimento controlado por parte del Instituto Canadiense de Ciberseguridad (CIC-IDS2017), el cual consistía en ir probando distintos tipos de ataques durante una semana. En parte, este dataset corresponde a una prueba de un viernes a la tarde, donde solamente se capturó tráfico malicioso DDoS y tráfico benigno, es decir, normal. El dataset, original en principio presenta 79 columnas con 225745 registros.

En principio, se puede decir con certeza, que disponemos de una barbaridad de datos y esto puede llegar a ser un problema a la hora de que los algoritmos procesen. Por tanto, en el archivo Python “analysis.py” lo que hemos realizado ha sido una disminución de registros, pasando de 225745 a tener solamente 50.000 registros. Esta reducción se ha llegado a realizar sin problema, manteniéndose patrones que hacen que los dos tipos de tráfico se acaben diferenciando.

Archivo “analysis.py”

Este es el primer archivo que se ha de ejecutar y es en él, donde realizamos esta reducción de variables. Este archivo además de ello, es donde se realiza la limpieza del dataset, ya que si nos damos cuenta, el dataset original presenta espacios en blanco en el nombre de cada variable. Se han solucionado aparte, problemas de valores nulos e infinitos además de datos redundantes.

Por último lo que hacemos en dicho archivo, es seleccionar las features que se llegaron a emplear en el anterior proyecto EDA. “Estas variables a mi modo de ver” eran las más eficientes a la hora de querer predecir dichos ataques DDoS, por lo tanto, lo que hacemos es generar dos documentos “.csv” al final del documento, tratando de hacer una copia limpia de nuestro dataset entero con tan solo 50000 registros y otra copia de 5000 registros pero solo con las features seleccionadas.

Archivo EDA.py

Tras tener nuestro dataset reducido y limpio con tan solamente las features seleccionadas, comenzamos a realizar nuestro EDA.

En nuestro EDA, comenzamos primeramente a realizar el análisis univariante para cada feature:

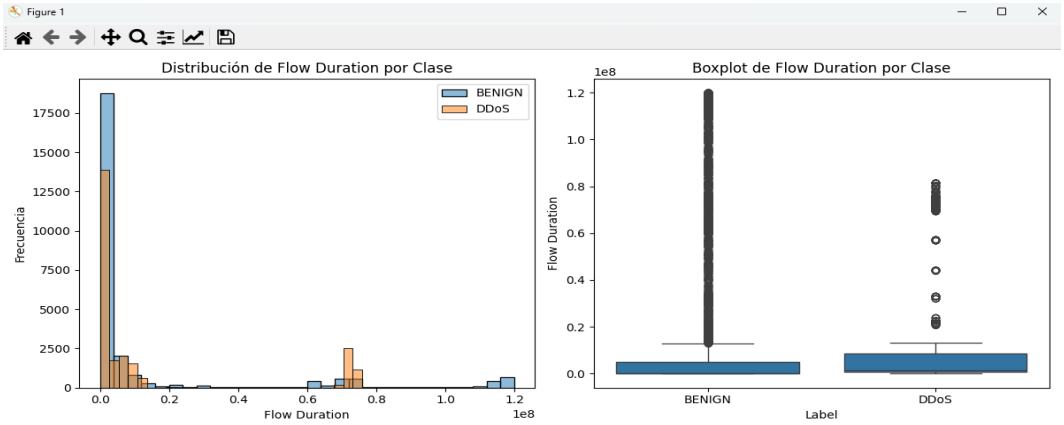


Figura 1: Distribución de la feature “Flow Duration” para el tráfico BENIGN y DDoS.

Hay que tener en cuenta, que este paso es crucial para comenzar a encontrar patrones claros de posibles outliers. Como nosotros ya sabemos, Flow Duration, mide lo que es el tiempo que un flujo de red está activo. Mirando entonces a la Figura 1 podemos encontrar diferencias entre los dos tipos de tráfico como por ejemplo: Flow Duration para los ataques DDoS muestran ser mucho mayores que los del tráfico BENIGN, esto nos dice y efectivamente es así, que el tráfico DDoS tiende a tener flujos mucho más prolongados. Efectivamente podemos saber esto con claridad realizando un “.describe()” y fijarnos en la media para saber cual de ellos, presenta más “Flow Duration”.

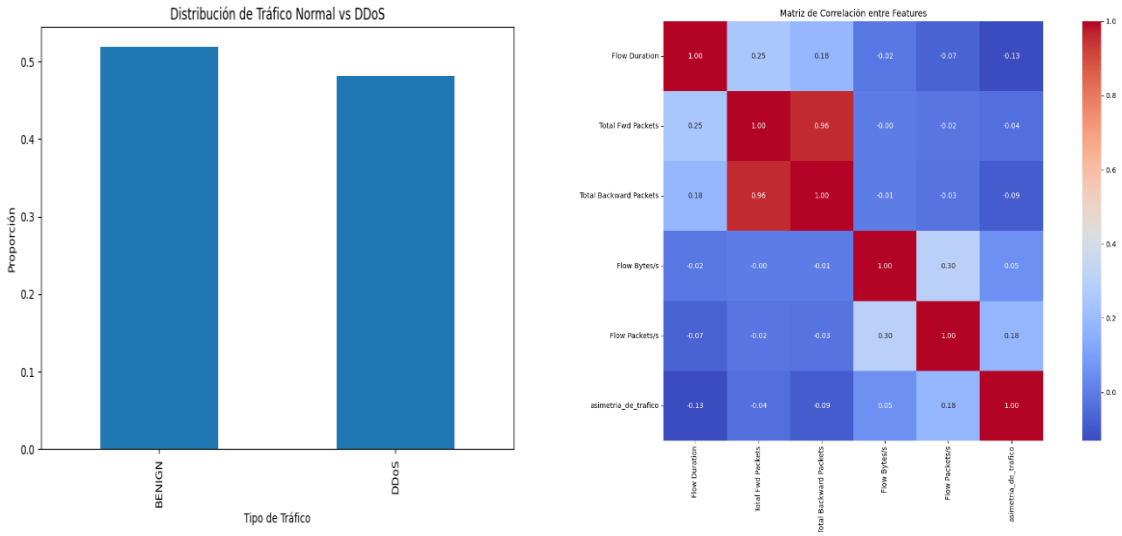
Estadísticas para Flow Duration:

Label	count	mean	std	min	25%	50%	75%	max
BENIGN	25696.0	1.274991e+07	3.010510e+07	1.0	235.00	58856.0	5168107.00	119998109.0
DDoS	23864.0	1.430251e+07	2.609519e+07	2.0	609343.75	1485373.0	8631689.25	81372696.0

Figura 2: Características de Flow Duration como ejemplo de Análisis Univariante.

Llegándose a observar que en verdad DDoS presenta mayor número de Flow Duration que el tráfico BENIGN. (Para ello, podemos fijarnos como hemos hecho en la media o mediana).

Tras ello comprobamos la distribución general del tipo de tráfico y la matriz de correlación de cada feature (exceptuando nuestro target “Label”).



Figuras 3 y 4: Muestran la distribución general del tráfico y la matriz de correlación.

Como puede apreciarse para la Figura 3 (izquierda), contamos con un 51.8% de tráfico BENIGN y un 48.15% para tráfico anómalo DDoS. Tras visualizar esto, notamos que se presenta un buen balance entre los dos tipos de tráfico.

Para la Figura 4 (derecha) en cambio, se presenta la matriz de correlación de nuestras features seleccionadas, notando gran correlación entre las variables “Total Fwd Packets” y “Total Backward Packets” con un índice de correlación de 0.96. A simple vista tiene sentido que haya una correlación fuerte entre estas variables ya que nos da una gran pista acerca de estos dos tipos de tráfico. “Lo normal es que un ataque DDoS envíe una gran cantidad de paquetes para saturar un servicio en concreto y no llegue a esperar una gran cantidad de paquetes de vuelta.” Para este caso DDoS en concreto, no se debería notar mucha correlación entre estas dos variables, en cambio para un tráfico BENIGN si tiene sentido.

Tras analizar la naturaleza de los datos realizando el EDA en cuestión, se planteó a continuación la métrica de clasificación adecuada para nuestro problema de predicción. De entre ellas la ganadora resultó ser: “**El recall**” debido a que se priorizó porque era fundamental minimizar el riesgo de que una anomalía DDoS no fuera detectada.

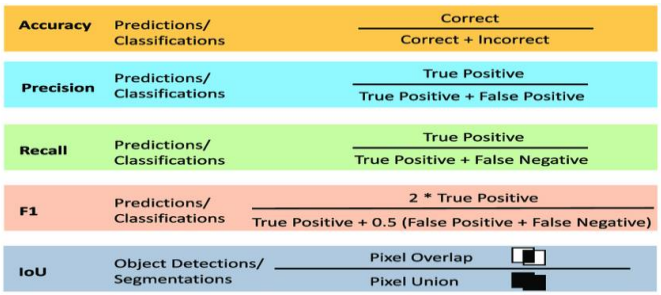


Figura 5: Métricas de clasificación

Algoritmos de ML

En este proyecto, se ha realizado esta predicción de ataques DDoS empleando los siguientes algoritmos de ML:

- Logistic Regression
- Support Vector Machine (SVM)
- Random Forest (RF)

De entre ellos 3, solo uno de ello, presenta las mejores características para predecir ataques DDoS a través de la métrica “recall”.

Los hiperparámetros que llegamos a emplear para estos 3 tipos de algoritmos fueron los siguientes:

```
# Definir parámetros para la búsqueda
param_grid = {
    'C': [0.1, 1, 10, 100],
    'penalty': ['l1', 'l2'],
    'solver': ['liblinear'],
    'max_iter': [1000]
}

print("\nIniciando búsqueda de hiperparám
n_combinaciones = len(param_grid['C']) *
print(f"Configuraciones a probar: {n_comb

# Búsqueda de mejores parámetros
grid_search = GridSearchCV(
    LogisticRegression(random_state=42),
    param_grid,
    cv=5,
    scoring='recall_macro',
    n_jobs=-1,
    verbose=2
)
```

```
# Definir parámetros para la búsqueda
param_grid = {
    'C': [0.1, 1, 10, 100],
    'gamma': ['scale', 'auto'],
    'kernel': ['rbf', 'linear']
}

print("\nIniciando búsqueda de hiperpa
n_combinaciones = len(param_grid['C'])
print(f"Configuraciones a probar: {n_c

# Búsqueda de mejores parámetros
grid_search = GridSearchCV(
    SVC(random_state=42),
    param_grid,
    cv=5,
    scoring='recall_macro',
    n_jobs=-1,
    verbose=2
)
```

```
# Definimos los parámetros para la búsqueda
param_grid = {
    "n_estimators": [50, 100, 200, 500],
    "max_features": ['sqrt', 'log2', None],
    "max_depth": [10, 20, 30, None],
    "min_samples_split": [2, 5, 10],
    "min_samples_leaf": [1, 2, 4],
    "bootstrap": [True, False]
}

print("\nIniciando búsqueda de hiperparámetros...")
n_combinaciones = len(param_grid['n_estimators']) * len(param_grid[
print(f"Configuraciones a probar: {n_combinaciones} combinaciones")

# Búsqueda de mejores parámetros
grid_search = GridSearchCV(RandomForestClassifier(random_state=42),
                           param_grid,
                           cv=5,
                           scoring='recall_macro',
                           verbose=2,
                           n_jobs=-1)
```

Figuras 6-7-8: Hiperparámetros que se llegaron a emplear.

Ya que el procesamiento tardaba demasiado debido a que había demasiadas combinaciones, me decanté por restringir mucho más los hiperparámetros además de emplear para “n_estimators” para cada algoritmo solamente 3 valores a probar. De entre ellos, los resultados para cada modelo fueron los siguientes: (Finales)

```
=== Resultados del modelo SVM ===

Mejores parámetros: {'C': 1, 'gamma': 'scale', 'kernel': 'rbf'}
Mejor puntuación CV: 0.932

Reporte de clasificación:
              precision    recall  f1-score   support

      BENIGN       0.98      0.87      0.92      5139
        DDoS       0.88      0.98      0.93      4773

    accuracy                           0.93      9912
   macro avg       0.93      0.93      0.93      9912
weighted avg       0.93      0.93      0.93      9912
```

Figura 9: Predicción final en test para el algoritmo SVM.

```
=== Resultados del modelo de Regresión Logística ===

Mejores parámetros: {'C': 100, 'max_iter': 1000, 'penalty': 'l1', 'solver': 'liblinear'}
Mejor puntuación CV: 0.808

Reporte de clasificación:
              precision    recall  f1-score   support

      BENIGN       0.85      0.75      0.80      5139
        DDoS       0.76      0.86      0.81      4773

    accuracy                           0.80      9912
   macro avg       0.81      0.81      0.80      9912
weighted avg       0.81      0.80      0.80      9912
```

Figura 10: Predicción final en test para el algoritmo Logistic Regression.

```
=== Resultados del modelo de Random Forest ===

Mejores parámetros: {'bootstrap': True, 'max_depth': 5, 'max_features': 'sqrt', 'min_samples_leaf': 10, 'min_samples_split': 20, 'n_estimators': 50}
Mejor puntuación CV: 0.932


Análisis de Overfitting:
Puntuación en entrenamiento: 0.942
Puntuación en prueba: 0.943
Diferencia (train - test): -0.001

Reporte de clasificación en prueba:
              precision    recall  f1-score   support

      BENIGN       0.99      0.89      0.94      5139
        DDoS       0.89      0.99      0.94      4773

    accuracy                           0.94      9912
   macro avg       0.94      0.94      0.94      9912
weighted avg       0.95      0.94      0.94      9912
```

Figura 11: Predicción final en test para el algoritmo Logistic Regression.

Llegándome a dar un valor buenísimo de predicción en recall para el algoritmo Random Forest sin llegar a presentarse “overfitting”.

Teniendo como ganador al algoritmo Random Forest (RF).

Consideraciones a tener en cuenta que se han introducido en el modelo final

La primera vez que llegué a emplear dichos algoritmos no me dieron valores tan altos hasta llegar a introducir en mis features seleccionadas, una nueva variable llamada “asimetria_de_trafico”. Esta nueva variable, llega de su mención del anterior proyecto EDA, que consiste en la siguiente métrica: abs(a-b) /(a+b) (Donde “a” representa el “Total Fwd Packets” y “b” representa “Total Backward Packets”.

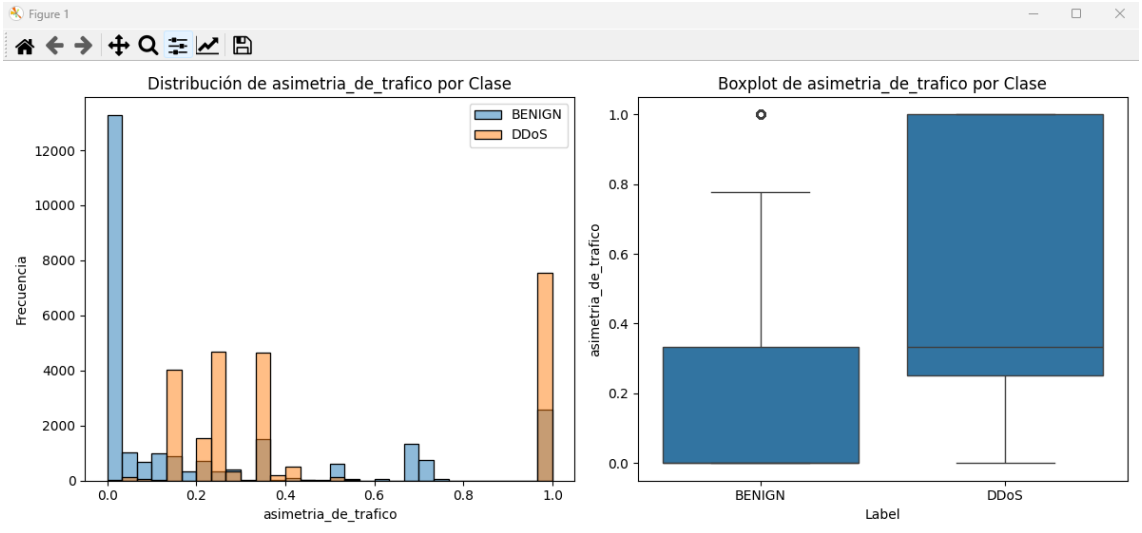


Figura 12: Análisis univariante de la nueva feature llamada “asimetria_de_trafico”.

La Figura 12, (“asimetria_de_trafico”) mide la diferencia en la cantidad de paquetes enviados en direcciones forward y backward. Realizando una comparativa, al igual que con Flow Duration, se puede comparar la asimetría entre BENIGN y DDoS. Si los valores de asimetría son significativamente más altos para DDoS, esto puede ser un indicador de que el tráfico DDoS tiene un patrón de tráfico más asimétrico. Realizando un “.describe()” una vez más:

```
Estadísticas para asimetria_de_trafico:
        count      mean       std  min   25%       50%       75%  max
Label
BENIGN  25696.0  0.223858  0.331746  0.0  0.00  0.000000  0.333333  1.0
DDoS    23864.0  0.488610  0.354758  0.0  0.25  0.333333  1.000000  1.0
```

Figura 13: Diferencias entre los dos tipos de tráfico en cuestión de la asimetría.

Llegamos a la conclusión entonces, que el tráfico más asimétrico es el DDoS frente al BENIGNO.

Además de ello, a la hora de entrenar nuestro modelo, hemos eliminado de nuestra matriz de correlación, una de las dos variables que altamente correlaban, siendo estas "Total Fwd Packets" y "Total Backward Packets". Por tanto, para evitar multicolinealidad, se ha eliminado "Total Backward Packets". Y así tener las siguientes features:

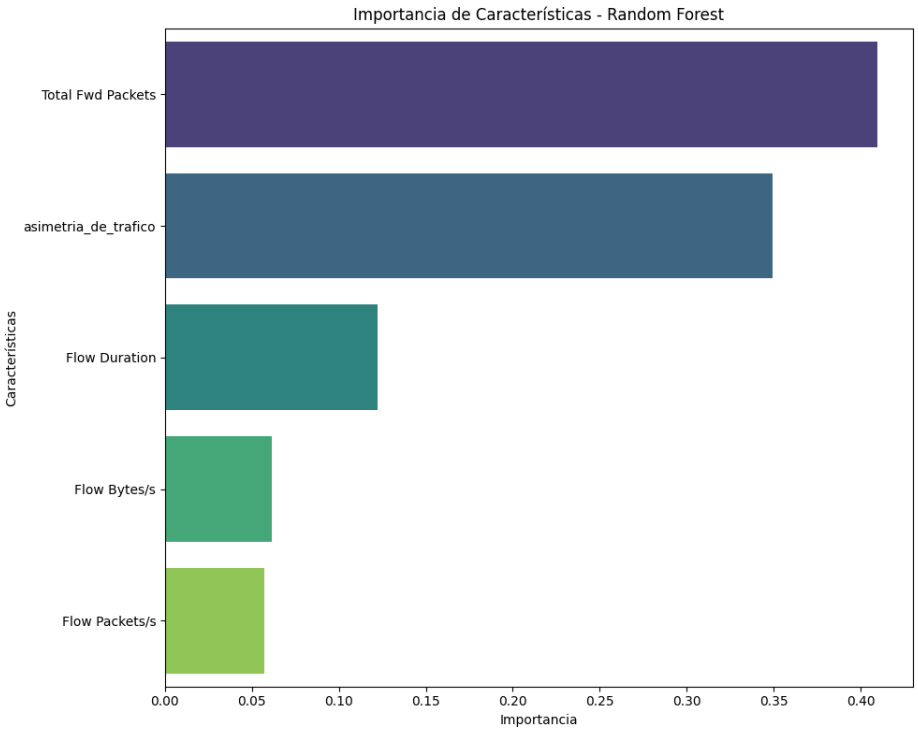


Figura 14: Feature importance de las variables con las que hemos entrenado el modelo.

Estas implementaciones se han tenido en cuenta a la hora de desarrollar el estudio final.

Conclusiones

Este trabajo de ML, presenta un estudio paso paso acerca de las tareas a realizar a la hora de predecir ataques DDoS frente a un tráfico normal. Útil además, para saber que algoritmo predice mejor los ataques.

A futuro, podríamos mejorar el rendimiento de dicho algoritmo Random Forest o hacer comparaciones frente a Redes Neuronales.