

1. Diseño de caso de uso de ML en Ciberseguridad

Problema de Negocio:

En una institución financiera con una base de clientes de 1 millón de usuarios, se enfrentan a un problema de detección de fraude en transacciones con tarjeta de crédito y débito. Actualmente, están experimentando una tasa de fraude del 1.5%, lo que resulta en pérdidas económicas significativas debido a transacciones fraudulentas, así como en la pérdida de confianza de los clientes en la seguridad de sus servicios financieros.

Acciones Actuales:

La institución financiera está utilizando reglas de detección de fraude basadas en patrones conocidos y transacciones sospechosas. Sin embargo, estas reglas tienen una tasa de detección del 85% con un 15% de falsos positivos, lo que significa que algunas transacciones legítimas están siendo identificadas erróneamente como fraudulentas.

Acción Buscada:

La institución financiera busca implementar un sistema de detección de fraude basado en Machine Learning para mejorar la precisión y eficacia de la detección de transacciones fraudulentas. Este sistema utilizará algoritmos avanzados de ML para analizar el comportamiento de compra, patrones de gasto, ubicaciones de transacciones y otros datos relevantes para identificar transacciones sospechosas con mayor precisión.

KPIs - Indicadores de Negocio:

El KPI directo definido es la tasa de detección de fraudes verdaderos (TPR - True Positive Rate), que representa el porcentaje de transacciones fraudulentas detectadas correctamente por el sistema de detección. Un TPR objetivo del 95% se consideraría satisfactorio, lo que indica que el sistema es capaz de identificar la gran mayoría de las transacciones fraudulentas.

Mínimos Esperados:

Se espera un aumento del TPR del 10% en comparación con las métricas actuales, lo que significaría reducir la cantidad de fraudes no detectados y mejorar la capacidad del sistema para identificar transacciones fraudulentas con mayor precisión.

Validación:

Para determinar si la solución es aceptable, se establecerá un umbral máximo para la tasa de falsos positivos, asegurando que el sistema no esté generando una cantidad excesiva de alertas falsas que puedan afectar negativamente la experiencia del cliente y aumentar los costos operativos de la institución financiera.

Experimentación:

Se realizarán pruebas experimentales utilizando conjuntos de datos históricos para evaluar el rendimiento del sistema de detección de fraude basado en Machine Learning. La frecuencia de las

pruebas será mensual, y se esperará ver una mejora continua en la precisión y eficacia del sistema a lo largo del tiempo.

Productivización:

En caso de una experimentación satisfactoria, la solución se implementará en producción y se integrará en los sistemas de detección de fraude existentes de la institución financiera. Se proporcionará una interfaz de usuario para que los analistas de fraude revisen y gestionen las alertas generadas por el sistema de ML de manera recurrente.

Equipo de Trabajo:

El equipo de trabajo estará compuesto por científicos de datos, ingenieros de software y analistas de fraude, quienes trabajarán conjuntamente de manera continua y diaria para desarrollar, implementar y mantener la solución de detección de fraude basada en Machine Learning.

Detalle del Caso de Uso:

Se analizarán los patrones de comportamiento de compra, las ubicaciones de transacciones, la cantidad de transacciones y otros datos relevantes para identificar señales de actividad fraudulenta en las transacciones financieras de los clientes. Los datos transaccionales y los registros de actividad serán los principales orígenes de datos utilizados en el análisis y entrenamiento del modelo de ML.

Identificación de Orígenes de Datos:

Desde un punto de vista funcional, los principales orígenes de datos que se utilizarían para la detección de fraude en transacciones financieras incluirían:

1. Registros de Transacciones: Este conjunto de datos contendría información detallada sobre todas las transacciones realizadas por los clientes, incluyendo la fecha, hora, ubicación, monto de la transacción, tipo de tarjeta utilizada (crédito/débito), etc.
2. Historial de Actividad del Cliente: Este conjunto de datos incluiría información sobre el comportamiento histórico de compra de cada cliente, como la frecuencia de transacciones, los patrones de gasto, las categorías de compra preferidas, etc.
3. Datos de Ubicación: Los datos de ubicación podrían provenir de dispositivos móviles o sistemas de geolocalización asociados con las transacciones, lo que proporcionaría información sobre la ubicación física del cliente al realizar la transacción.
4. Datos Demográficos y de Perfil del Cliente: Estos datos incluirían información demográfica, como edad, género, ingresos, historial crediticio, etc., así como cualquier otra información relevante del perfil del cliente.

Desarrollo del Caso de Uso:

Para mejorar la detección de fraude en transacciones financieras, se realizarán los siguientes puntos intermedios o de seguimiento:

1. Análisis de Patrones de Comportamiento: Se investigará si ciertos patrones de comportamiento, como cambios inusuales en los hábitos de gasto o la aparición de nuevas categorías de compra,

podrían indicar actividad fraudulenta.

2. Validación de Alertas: Se realizarán pruebas para verificar la validez de las alertas generadas por el sistema de detección de fraude basado en Machine Learning, utilizando conjuntos de datos históricos y simulaciones de casos de fraude conocidos.

3. Optimización de Modelos de Machine Learning: Se explorarán diferentes algoritmos de ML y técnicas de preprocesamiento de datos para mejorar la precisión y eficacia del sistema de detección de fraude.

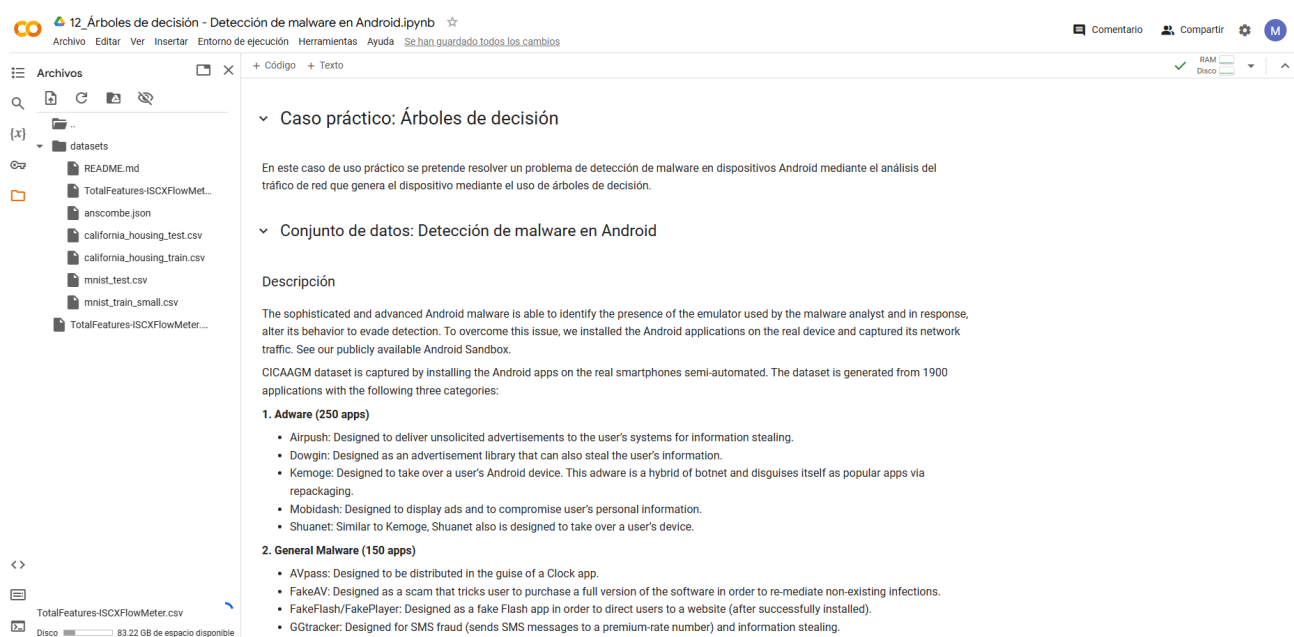
4. Evaluación de Rendimiento: Se evaluará el rendimiento del sistema en términos de tasa de detección de fraudes verdaderos, tasa de falsos positivos, etc., y se realizarán ajustes según sea necesario.

Aporte Esperado por Big Data:

El equipo de Big Data espera mejorar significativamente la capacidad de la institución financiera para detectar y prevenir fraudes en transacciones financieras mediante el uso de técnicas avanzadas de análisis de datos y Machine Learning. Se espera que esto reduzca las pérdidas económicas asociadas con transacciones fraudulentas, mejore la confianza de los clientes y proteja la reputación de la institución financiera en el mercado. Además, se espera que el uso de Big Data permita una detección más temprana de fraudes y una respuesta más rápida ante posibles amenazas, lo que minimizará el impacto negativo en la institución financiera y sus clientes.

2. Resolver usando python el siguiente problema de ML:

Detección de malware en Android



12_Árboles de decisión - Detección de malware en Android.ipynb

Archivos

- datasets
- README.md
- TotalFeatures-ISCXFlowMet...
- anscombe.json
- california_housing_test.csv
- california_housing_train.csv
- mnist_test.csv
- mnist_train_small.csv
- TotalFeatures-ISCXFlowMeter...

Código

Caso práctico: Árboles de decisión

En este caso de uso práctico se pretende resolver un problema de detección de malware en dispositivos Android mediante el análisis del tráfico de red que genera el dispositivo mediante el uso de árboles de decisión.

Conjunto de datos: Detección de malware en Android

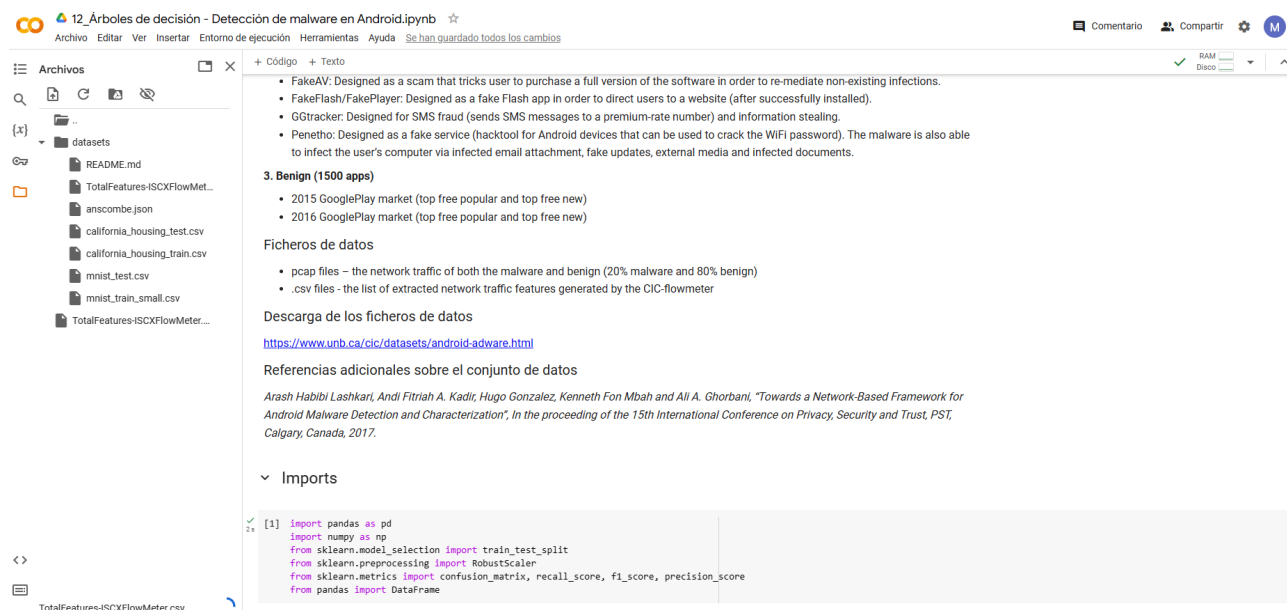
Descripción

The sophisticated and advanced Android malware is able to identify the presence of the emulator used by the malware analyst and in response, alter its behavior to evade detection. To overcome this issue, we installed the Android applications on the real device and captured its network traffic. See our publicly available Android Sandbox.

CICAAGM dataset is captured by installing the Android apps on the real smartphones semi-automated. The dataset is generated from 1900 applications with the following three categories:

- Adware (250 apps)**
 - Airpush: Designed to deliver unsolicited advertisements to the user's systems for information stealing.
 - Dowgin: Designed as an advertisement library that can also steal the user's information.
 - Kemoge: Designed to take over a user's Android device. This adware is a hybrid of botnet and disguises itself as popular apps via repackaging.
 - Mobidash: Designed to display ads and to compromise user's personal information.
 - Shuanet: Similar to Kemoge, Shuanet also is designed to take over a user's device.
- General Malware (150 apps)**
 - AVpass: Designed to be distributed in the guise of a Clock app.
 - FakeAV: Designed as a scam that tricks user to purchase a full version of the software in order to re-mediate non-existing infections.
 - FakeFlash/FakePlayer: Designed as a fake Flash app in order to direct users to a website (after successfully installed).
 - GGtracker: Designed for SMS fraud (sends SMS messages to a premium-rate number) and information stealing.

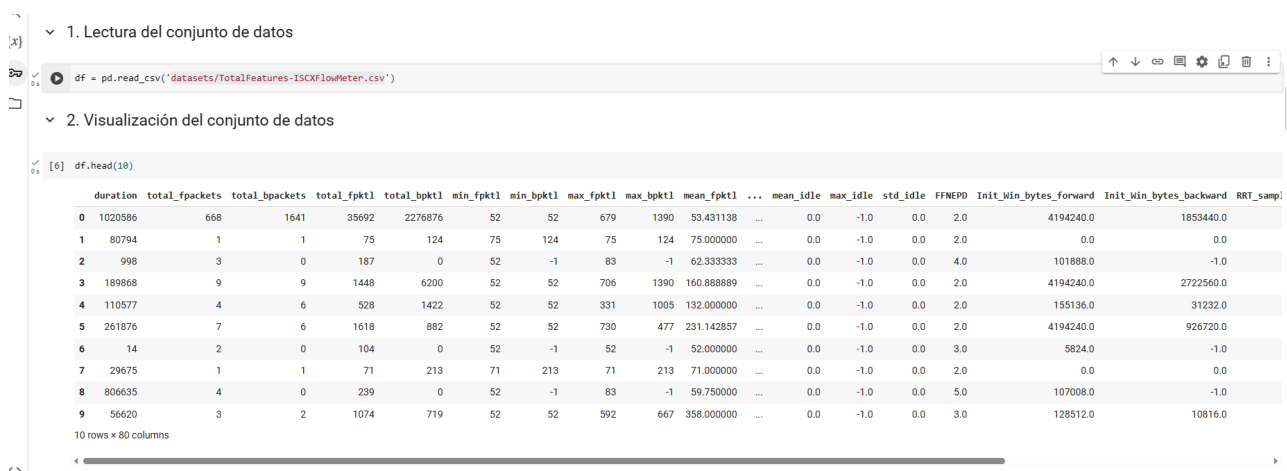
Cargamos las librerías



ejecuto la función `train_val_test_split` y divido la variable objetivo. También me dará la métrica con los datos preparados o sin preparar:



leemos el conjunto de datos del fichero `datasets/TotalFeatures-ISCXFlowMeter.csv`



la información de cada variable (nombre, no nulo y tipo de dato):

```
[X] df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 28129 entries, 0 to 28128
Data columns (total 80 columns):
#   Column                Non-Null Count  Dtype
---  ---                ---
0   duration              28129 non-null  int64
1   total_fpackets        28129 non-null  int64
2   total_bpackets        28129 non-null  int64
3   total_fpktl           28129 non-null  int64
4   total_bpktl           28129 non-null  int64
5   min_fpktl             28129 non-null  int64
6   min_bpktl             28129 non-null  int64
7   max_fpktl             28129 non-null  int64
8   max_bpktl             28129 non-null  int64
9   mean_fpktl            28129 non-null  float64
10  mean_bpktl            28129 non-null  float64
11  std_fpktl             28129 non-null  float64
12  std_bpktl            28129 non-null  float64
13  total_fiat            28129 non-null  int64
14  total_biat            28129 non-null  int64
15  min_fiat              28129 non-null  int64
16  min_biat              28129 non-null  int64
17  max_fiat              28129 non-null  int64
18  max_biat              28129 non-null  int64
19  mean_fiat             28129 non-null  float64
20  mean_biat             28129 non-null  float64
21  std_fiat              28129 non-null  float64
22  std_biat              28129 non-null  float64
23  fpush_cnt             28129 non-null  int64
24  bpush_cnt             28129 non-null  int64
25  furg_cnt              28129 non-null  int64
26  burg_cnt              28129 non-null  int64
27  total_fhlen           28129 non-null  int64
28  total_bhlen           28129 non-null  int64
29  fPktsPerSecond        28129 non-null  float64
30  bPktsPerSecond        28129 non-null  float64
31  flowPktsPerSecond     28129 non-null  float64
32  ...                   ...
```

buscamos correlaciones

Buscando correlaciones

```
[11] # Copiamos el conjunto de datos y transformamos la variable de salida a numérica para calcular correlaciones
X = df.copy()
X['calss'] = X['calss'].factorize()[0]
```

Calculamos correlaciones

```
corr_matrix = X.corr()
corr_matrix['calss'].sort_values(ascending=False)

calss      1.000000
max_flowpktl 0.005146
mean_bpktl  0.003492
min_bpktl   0.003474
max_bpktl   0.003123
Init_Win_bytes_forward ...
Init_Win_bytes_backward NaN
RRT_samples_clnt NaN
Act_data_pkt_forward NaN
min_seq_size_forward NaN
Name: calss, Length: 80, dtype: float64
```

X.corr()

	duration	total_fpackets	total_bpackets	total_fpktl	total_bpktl	min_fpktl	min_bpktl	max_fpktl	max_bpktl	mean_fpktl	...	mean_idle	max_idle	std_idle	FFNEPD	Init_Win_bytes_forward	Init_Win_byt
duration	1.000000	0.005019	0.005420	0.003147	0.004225	-0.027431	-0.009929	0.034428	0.047204	-0.001419	...	0.999707	0.999960	0.033629	0.022558		0.026891
total_fpackets	0.005019	1.000000	0.703932	0.935490	0.380521	-0.013346	-0.000883	0.056950	0.065512	0.052364	...	0.002403	0.003437	0.064060	-0.003188		0.038749
total_bpackets	0.005420	0.703932	1.000000	0.412576	0.923929	-0.019725	0.001336	0.056456	0.103982	0.007936	...	0.003472	0.003991	0.030805	-0.011905		0.061580
total_fpktl	0.003147	0.935490	0.412576	1.000000	0.036202	-0.004083	-0.000268	0.037836	0.023045	0.063586	...	0.000877	0.001870	0.063279	-0.002590		0.011604
total_bpktl	0.004225	0.380521	0.923929	0.036202	1.000000	-0.015332	0.000403	0.039384	0.095419	-0.016327	...	0.003108	0.003225	0.006396	-0.010054		0.053098
...
Init_Win_bytes_backward	0.021231	0.038637	0.057984	0.013207	0.048351	-0.266829	-0.003132	0.272844	0.398398	-0.110493	...	0.019056	0.020977	0.100903	-0.171238		0.889888
RRT_samples_clnt	0.004487	0.370799	0.920278	0.023385	0.998615	-0.018354	0.000039	0.044423	0.101878	-0.017851	...	0.003336	0.003476	0.007100	-0.011600		0.059915
Act_data_pkt_forward	0.005019	1.000000	0.703932	0.935490	0.380521	-0.013345	-0.000883	0.056952	0.065512	0.052369	...	0.002403	0.003437	0.064060	-0.003188		0.038750
min_seq_size_forward	0.031668	0.018656	0.025529	0.006191	0.020862	-0.706679	-0.256151	-0.092960	0.184831	-0.513385	...	0.031371	0.031777	0.023559	0.101462		0.388403
calss	0.000334	0.000164	0.000238	0.000050	0.000185	-0.012368	0.003474	-0.004060	0.003123	-0.010319	...	NaN	NaN	NaN	NaN		NaN

80 rows x 80 columns

```
# Se puede llegar a valorar quedarnos con aquellas que tienen mayor correlación
corr_matrix[corr_matrix["calss"] > 0.85]
```

	duration	total_fpackets	total_bpackets	total_fpktl	total_bpktl	min_fpktl	min_bpktl	max_fpktl	max_bpktl	mean_fpktl	...	mean_idle	max_idle	std_idle	FFNEPD	Init_Win_bytes_forward	Init_Win_bytes_backward	RRT_s
calss	0.000334	0.000164	0.000238	0.00005	0.000185	-0.012368	0.003474	-0.00406	0.003123	-0.010319	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN

1 rows x 80 columns

dividimos el conjunto de datos

3. División del conjunto de datos

```
[15] # Dividimos el conjunto de datos
train_set, val_set, test_set = train_val_test_split(X)
```

```
X_train, y_train = remove_labels(train_set, 'calss')
X_val, y_val = remove_labels(val_set, 'calss')
X_test, y_test = remove_labels(test_set, 'calss')
```

escalamos los datos

4. Escalando el conjunto de datos

Es importante comprender que los árboles de decisión son algoritmos que **no requieren demasiada preparación de los datos** concretamente, no requieren la realización de escalado o normalización. En este ejercicio se va a realizar escalado al conjunto de datos y se van a comparar los resultados con el conjunto de datos sin escalar. De esta forma se demuestra como aplicar preprocesamientos como el escalado puede incluso llegar a afectar al rendimiento del modelo.

```
05 scaler = RobustScaler()
X_train_scaled = scaler.fit_transform(X_train)

[18] scaler = RobustScaler()
X_test_scaled = scaler.fit_transform(X_test)

[19] scaler = RobustScaler()
X_val_scaled = scaler.fit_transform(X_val)

# Transformación a un DataFrame de Pandas
X_train_scaled = DataFrame(X_train_scaled, columns=X_train.columns, index=X_train.index)
X_train_scaled.head(10)
```

	duration	total_packets	total_bpkts	total_fpktl	total_bpktl	min_fpktl	min_bpktl	max_fpktl	max_bpktl	mean_fpktl	...	min_idle	mean_idle	max_idle	std_idle	FFNEPD	Init_Win_bytes_forward	Init_Win_bytes
449	-0.141791	0.0	-0.333333	-0.522346	-0.317073	0.000000	-1.0	-0.152174	-0.616279	-0.131171	...	0.0	0.0	0.0	0.0	0.0	-0.183911	
15744	-0.141791	0.0	-0.333333	0.511173	-0.317073	19.473684	-1.0	0.853261	-0.616279	1.434415	...	0.0	0.0	0.0	0.0	0.0	-0.197007	
17970	-0.141791	0.0	-0.333333	0.326816	-0.317073	16.000000	-1.0	0.673913	-0.616279	1.155148	...	0.0	0.0	0.0	0.0	0.0	-0.197007	
19514	-0.119612	0.5	-0.333333	-0.290503	-0.317073	0.000000	-1.0	-0.067935	-0.616279	-0.065585	...	0.0	0.0	0.0	0.0	1.0	0.037559	
16083	-0.141791	0.0	-0.333333	0.472067	-0.317073	18.736842	-1.0	0.815217	-0.616279	1.375176	...	0.0	0.0	0.0	0.0	0.0	-0.197007	
26495	250.867785	3.0	2.666667	3.851955	6.378049	0.000000	0.0	1.690217	4.883721	0.626839	...	59157872.0	59200000.0	59157872.0	0.0	0.0	9.233847	
19997	-0.141774	1.0	-0.333333	-0.145251	-0.317073	0.000000	-1.0	-0.067935	-0.616279	-0.087447	...	0.0	0.0	0.0	0.0	2.0	0.037559	

```
05 X_train_scaled.describe()
```

	duration	total_packets	total_bpkts	total_fpktl	total_bpktl	min_fpktl	min_bpktl	max_fpktl	max_bpktl	mean_fpktl	...	min_idle	mean_idle	max_idle	std_idle	FFNEPD	Init_Win_bytes_forward	Init_Win_bytes
count	16877.000000	16877.000000	16877.000000	16877.000000	16877.000000	16877.000000	16877.000000	16877.000000	16877.000000	16877.000000	...	1.687600e+04	1.687600e+04	1.687600e+04	1.687600e+04	16876.000000	16876.000000	16876.000000
mean	62.408948	2.541625	2.019533	8.299416	39.583684	3.729538	0.019233	0.360583	1.690014	0.327258	...	1.410876e+07	1.427730e+07	1.449960e+07	2.357698e+05	0.316900	0.316900	0.316900
std	1256.573685	103.637866	64.865012	759.545244	1459.301115	7.533917	1.758742	0.709434	4.393656	0.633235	...	2.972311e+08	2.974012e+08	2.973193e+08	5.124242e+06	0.939763	0.939763	0.939763
min	-0.141791	-0.500000	-0.333333	-0.667598	-0.317073	-2.789474	-1.000000	-0.296196	-0.616279	-0.351199	...	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000	0.000000	0.000000
25%	-0.141791	0.000000	-0.333333	-0.494413	-0.317073	0.000000	-1.000000	-0.152174	-0.616279	-0.131171	...	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000	0.000000	0.000000
75%	0.858209	1.000000	0.666667	0.505587	0.682927	1.000000	0.000000	0.847826	0.383721	0.868829	...	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000	0.000000	0.000000
max	143957.841224	12048.500000	4748.000000	92588.206704	120702.756098	70.421053	25.245283	3.483696	15.558140	5.530324	...	3.406373e+10	3.410000e+10	3.406373e+10	2.110000e+08	18.000000	18.000000	18.000000

8 rows x 19 columns

Ejecutamos/mostramos el árbol de decisión

5. Árbol de decisión

```
20 from sklearn.tree import DecisionTreeClassifier

MAX_DEPTH = 20

# Modelo entrenado con el conjunto de datos sin escalar
clf_tree = DecisionTreeClassifier(max_depth=MAX_DEPTH, random_state=42)
clf_tree.fit(X_train, y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=20, random_state=42)
```

```
25 # Modelo entrenado con el conjunto de datos escalado
clf_tree_scaled = DecisionTreeClassifier(max_depth=MAX_DEPTH, random_state=42)
clf_tree_scaled.fit(X_train_scaled, y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=20, random_state=42)
```

Comenzar prediciendo con el propio conjunto de datos con el que se ha entrenado el algoritmo (train set), suele ser interesante para comprobar si se esta produciendo overfitting.

```
✓ [22] # Predicimos con el conjunto de datos de entrenamiento
0s y_train_pred = clf_tree.predict(X_train)
    y_train_prep_pred = clf_tree_scaled.predict(X_train_scaled)

✓ [23] # Comparamos resultados entre escalado y sin escalar
0s evaluate_result(y_train_pred, y_train, y_train_prep_pred, y_train, f1_score)

f1_score WITHOUT preparation: 0.993112784026665
f1_score WITH preparation: 0.993112784026665
```

visualizamos el límite de decisión

A partir de este punto, y, en función de los resultados que hemos visto, nos quedamos con la opción de no escalar nuestros conjuntos de datos

6. Visualizando el limite de decision

```
✓ [26] # Reducimos el número de atributos del conjunto de datos para visualizarlo mejor
0s X_train_reduced = X_train[['min_flowpkt1', 'flow_fin']]

✓ [27] # Generamos un modelo con el conjunto de datos reducido
0s clf_tree_reduced = DecisionTreeClassifier(max_depth=2, random_state=42)
    clf_tree_reduced.fit(X_train_reduced, y_train)
```

DecisionTreeClassifier

DecisionTreeClassifier(max_depth=2, random_state=42)

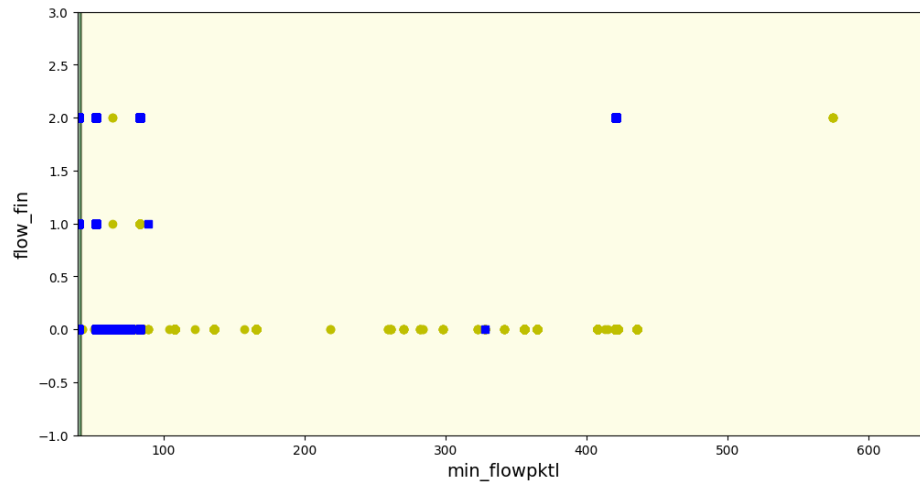
construimos el límite de decisión

```
✓ [28] # Representamos gráficamente el límite de decisión construido
1s from matplotlib.colors import ListedColormap
    import matplotlib.pyplot as plt
    %matplotlib inline

def plot_decision_boundary(clf, X, y, plot_training=True, resolution=1000):
    mins = X.min(axis=0) - 1
    maxs = X.max(axis=0) + 1
    x1, x2 = np.meshgrid(np.linspace(mins[0], maxs[0], resolution),
                        np.linspace(mins[1], maxs[1], resolution))
    X_new = np.c_[x1.ravel(), x2.ravel()]
    y_pred = clf.predict(X_new).reshape(x1.shape)
    custom_cmap = ListedColormap(['#fafab0', '#9898ff', '#a0faa0'])
    plt.contourf(x1, x2, y_pred, alpha=0.3, cmap=custom_cmap)
    custom_cmap2 = ListedColormap(['#7d7d58', '#4c4c7f', '#507d50'])
    plt.contour(x1, x2, y_pred, cmap=custom_cmap2, alpha=0.8)
    if plot_training:
        plt.plot(X[:, 0][y==0], X[:, 1][y==0], "yo", label="normal")
        plt.plot(X[:, 0][y==1], X[:, 1][y==1], "bs", label="adware")
        plt.plot(X[:, 0][y==2], X[:, 1][y==2], "g^", label="malware")
        plt.axis([mins[0], maxs[0], mins[1], maxs[1]])
    plt.xlabel('min_flowpkt1', fontsize=14)
    plt.ylabel('flow_fin', fontsize=14, rotation=90)

plt.figure(figsize=(12, 6))
plot_decision_boundary(clf_tree_reduced, X_train_reduced.values, y_train)
plt.show()
```

/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but DecisionTreeClassifier was fitted with feature names
warnings.warn(



y finalmente pintamos el árbol de decisión:

```
# Pintamos el árbol para compararlo con la representación gráfica anterior
from graphviz import Source
from sklearn.tree import export_graphviz
import os

export_graphviz(
    clf_tree_reduced,
    out_file="android_malware.dot",
    feature_names=X_train_reduced.columns,
    class_names=["benign", "adware", "malware"],
    rounded=True,
    filled=True
)

Source.from_file("android_malware.dot")
```

