Práctica 2

Representación de los datos

Diego Palacios

David Billhardt

Javier Rufo

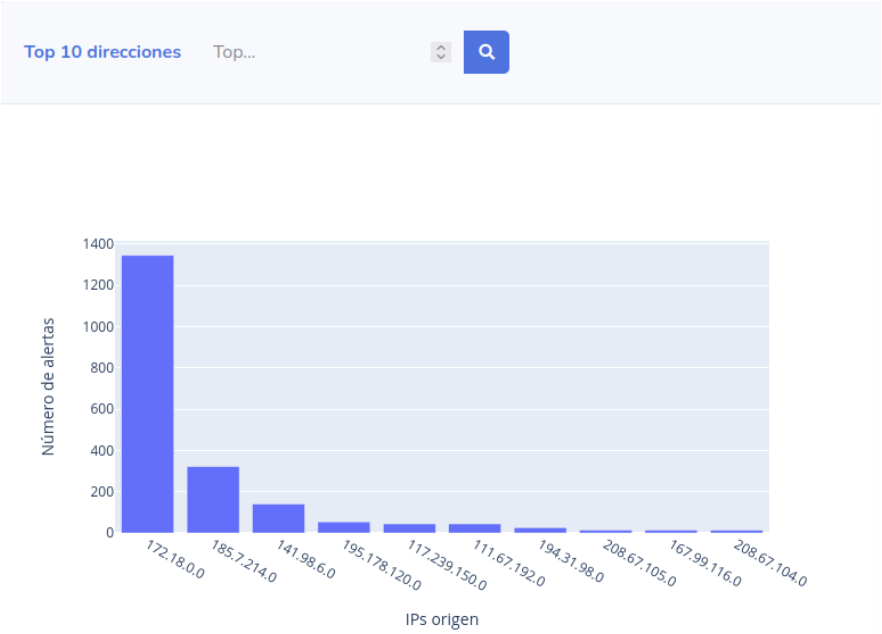
Introducción

Para realizar el CMI partimos del código de la práctica anterior (modificado parcialmente) que crea las tablas de la base de datos y las carga con los valores correspondientes. Este código (el archivo create_BBDD.py) solo debe ser ejecutado una vez y simula la idea de que se partiría de un almacén ya creado al desarrollar el CMI.

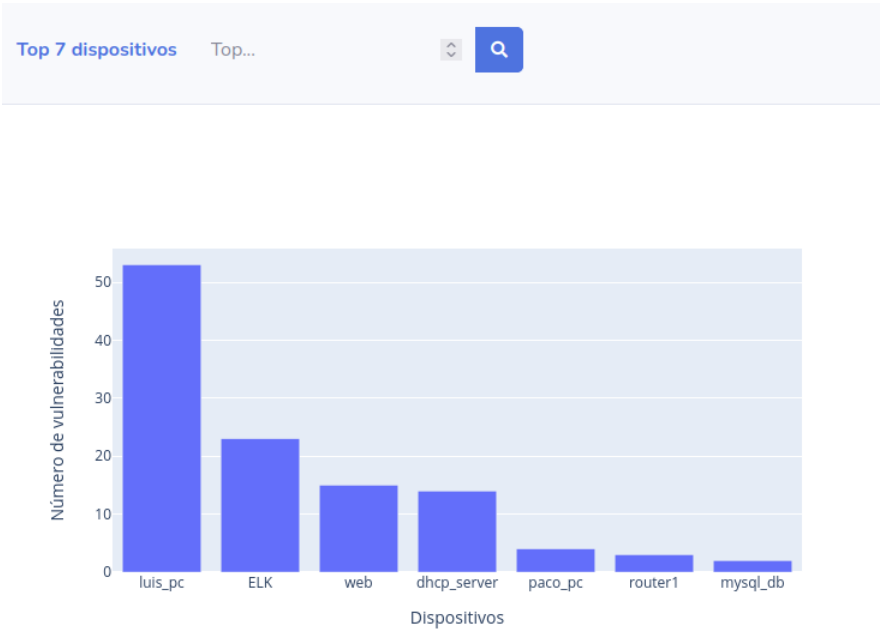
El CMI, por otro lado, es una página web cuyo back-end reside en un programa de Python basado en flask (archivo server.py) y cuyo front-end (parte esencial de un CMI) consiste en cinco apartados: dos reservados para el ejercicio 1, y uno para cada uno de los ejercicios 2, 3 y 4. Las gráficas han sido realizadas con Plotly (tanto Python como JavaScript), y las tablas con Tabulator de JavaScript. La mayoría de gráficos y tablas se cargan nada más se inicia la aplicación al llamarse a las funciones de cada endpoint con sus valores por defecto utilizando la sintaxis de plantillas de Jinja, a excepción de los CVEs y las noticias (ejercicio adicional) que requieren de pulsar el botón “Cargar datos” para que se realice la solicitud, con el fin de aligerar la carga inicial de la página y que no sea imprescindible que estas tablas se encuentren cargadas para poder entrar al dominio por primera vez.

Ejercicio 1

El top X (por defecto 10) de IPs de origen más problemáticas se representa mediante un gráfico de barras con las IPs más problemáticas y su respectivo número de alertas, en orden de mayor número de alertas (más problemática) a menor. El número de alertas se puede variar dinámicamente a elección del usuario utilizando la barra de búsqueda habilitada para poder visualizar el gráfico con más o menos IPs. Para hacer este gráfico hemos tenido que extraer la información de la base de datos mediante una query para generar la gráfica con Plotly, que se envía en formato JSON para ser de nuevo interpretada en el front-end. Como se ha explicado en la introducción, cuando se cargue la página se llamará a la función que realiza este mismo procedimiento con el valor por defecto, y se envía a la página para cargarlo con la plantilla de Jinja en primera instancia. Después, si se producen cambios en el número de IPs que se desean ver, se utiliza Plotly.react() para modificar los datos que se muestran.



Para mostrar el top X de dispositivos más vulnerables hemos tomado una perspectiva similar al apartado anterior, simplemente mostrarlos en un gráfico de barras. El nivel de vulnerabilidad de un dispositivo se mide, en nuestro caso, como el número de servicios inseguros sumado al número de vulnerabilidades detectadas. A efectos de front-end, la manera de proceder es exactamente la misma que en el caso anterior.

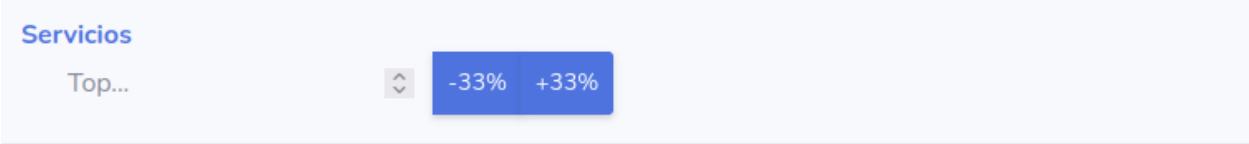


Ejercicio 2

En este ejercicio, para mostrar los servicios más o menos inseguros, recuperamos la información de la base de datos, calculamos un ratio de inseguridad (servicios inseguros entre servicios totales) y comparamos y ordenamos por ese ratio.

Está la opción de mostrar X número de dispositivos (por defecto, 3), y la query se construye teniendo en cuenta tanto el número de servicios que se desea mostrar en caso de que se especifiquen, como el hecho de que sean mayores o menores al 33%.

Para el front-end hemos utilizado Tabulator para construir tablas con los parámetros importantes, es decir ID, IP, localización y ratio.



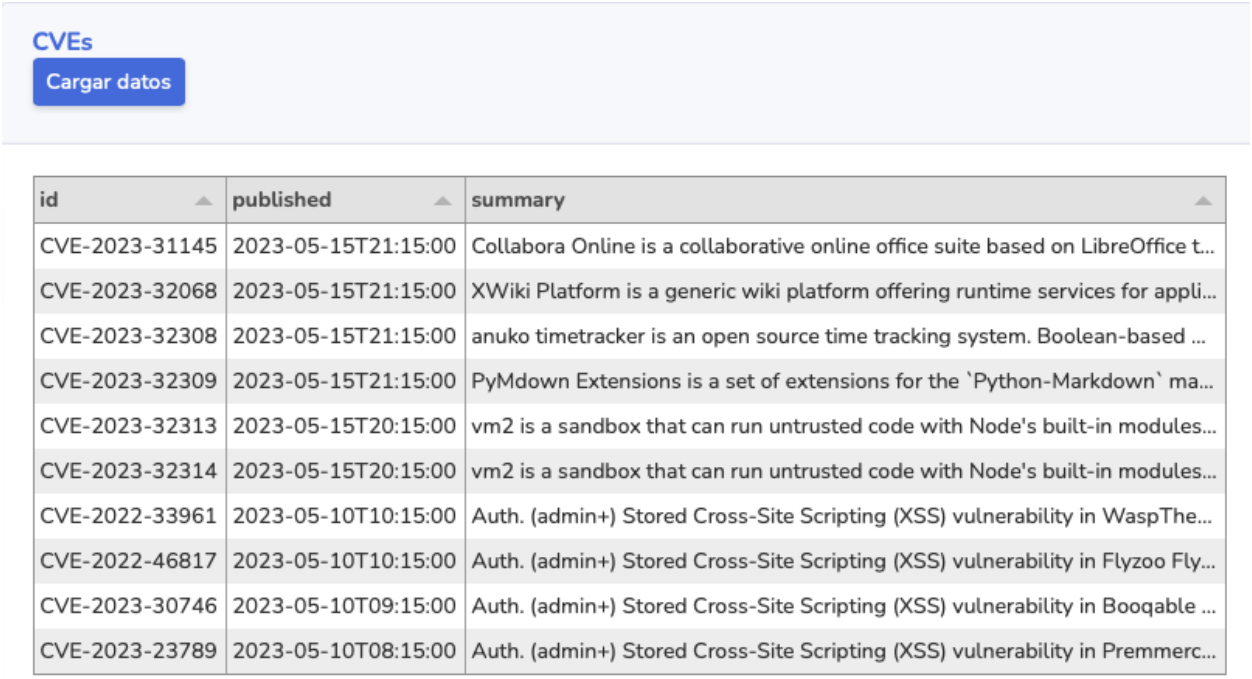
id	ip	localizacion	ratio
luis_pc	172.19.0.0	Madrid	1
dhcp_server	172.1.0.1	Madrid	1
ELK	172.18.0.2	None	0.667

Ejercicio 3

Para este ejercicio, vamos a utilizar la API de cve-search. Como nos interesa obtener las últimas vulnerabilidades, haremos peticiones a la siguiente ruta: https://cve.circl.lu/api/last

La API devuelve muchísima información, es por esto por lo que, en nuestro caso, hemos decidido quedarnos con los campos id, published y summary. Consideramos que son los más importantes, ya que dan información sobre el identificador de la vulnerabilidad, la fecha en la que se ha publicado y una descripción sobre esta vulnerabilidad. Toda esta información la vamos a ordenar por la fecha para que así mostremos las vulnerabilidades de más a menos reciente y solo nos vamos a quedar con las 10 primeras entradas. Otro punto relevante es que hemos reducido el número de caracteres que pueden aparecer en la descripción. Hemos tomado esta decisión, ya que, al mostrar la información en el CMI, si no acotábamos el número de caracteres, la información era muy dispar y eso provocaba que fuese molesto leer la descripción.

Finalmente, se muestra esta información en el CMI con Tabulator de nuevo:



id	published	summary
CVE-2023-31145	2023-05-15T21:15:00	Collabora Online is a collaborative online office suite based on LibreOffice t...
CVE-2023-32068	2023-05-15T21:15:00	XWiki Platform is a generic wiki platform offering runtime services for appli...
CVE-2023-32308	2023-05-15T21:15:00	anuko timetracker is an open source time tracking system. Boolean-based ...
CVE-2023-32309	2023-05-15T21:15:00	PyMdown Extensions is a set of extensions for the `Python-Markdown` ma...
CVE-2023-32313	2023-05-15T20:15:00	vm2 is a sandbox that can run untrusted code with Node's built-in modules...
CVE-2023-32314	2023-05-15T20:15:00	vm2 is a sandbox that can run untrusted code with Node's built-in modules...
CVE-2022-33961	2023-05-10T10:15:00	Auth. (admin+) Stored Cross-Site Scripting (XSS) vulnerability in WaspThe...
CVE-2022-46817	2023-05-10T10:15:00	Auth. (admin+) Stored Cross-Site Scripting (XSS) vulnerability in Flyzoo Fly...
CVE-2023-30746	2023-05-10T09:15:00	Auth. (admin+) Stored Cross-Site Scripting (XSS) vulnerability in Booqable ...
CVE-2023-23789	2023-05-10T08:15:00	Auth. (admin+) Stored Cross-Site Scripting (XSS) vulnerability in Premmerc...

La parte del código responsable de obtener los datos es:

```
@app.route('/cves', methods=['GET'])
def cves():
    cves_resultado = []
    res = requests.get("https://cve.circl.lu/api/last").text
    data = json.loads(res)
    data = sorted(data, key=lambda d:d['Published'], reverse=True)[:10]

    for i in range(10):
        cve = {'id': data[i]["id"], 'published': data[i]["Published"],
'summary': data[i]["summary"].strip()[:107] + "..."}
        cves_resultado.append(cve)

    return cves_resultado
```

Ejercicio 4

Para esta parte hemos decidido incorporar información adicional al CMI. En concreto, hemos incorporado información relativa a las últimas noticias de la página HackerNews. Para poder hacer esto, hemos usado la API que han desarrollado en colaboración con Firebase.

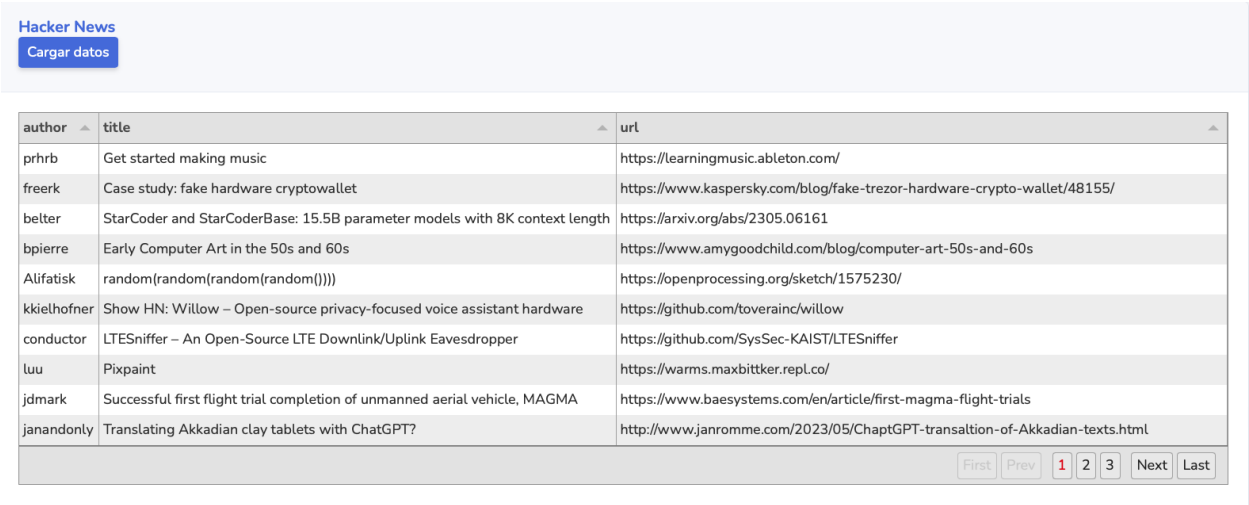
Haciendo una petición a: https://hacker-news.firebaseio.com/v0/topstories.json?print=pretty

Vamos a obtener los identificadores de las diferentes "stories". En nuestro caso, solo vamos a mostrar 30. Por tanto, nos guardamos los 30 primeros identificadores en una lista y vamos a ir haciendo peticiones a la siguiente ruta:

https://hacker-news.firebaseio.com/v0/item/id.json?print=pretty

Donde el campo id va a ir tomando los diferentes valores que hemos obtenido.

De toda la información que obtenemos de cada noticia, solo nos interesa mostrar el autor, la URL y el título. Es por esto por lo que tratamos el contenido. Finalmente, se muestra esta información en el CMI por medio de Tabulator, esta vez, con paginación para hacer 10 artículos por cada una de las tres páginas:



La parte del código responsable de obtener los datos es:

```
@app.route('/news', methods=['GET'])
def news():
    res =
requests.get("https://hacker-news.firebaseio.com/v0/topstories.json?print=p
```

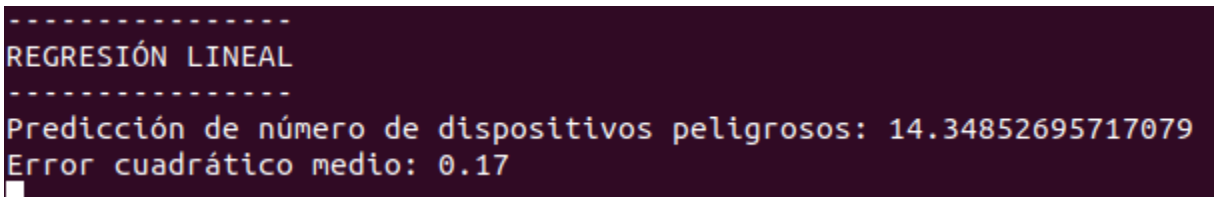
```
retty").text.strip().replace('[','').replace(']','').split(',')
    noticias = []
    for i in range(30):
        id = int(res[i])
        res2 =
requests.get(f"https://hacker-news.firebaseio.com/v0/item/{id}.json?print=p
retty").text
        data = json.loads(res2)
        noticia = {'title': data["title"], 'url': data["url"].strip(),
'author': data["by"]}
        noticias.append(noticia)

    return noticias
```

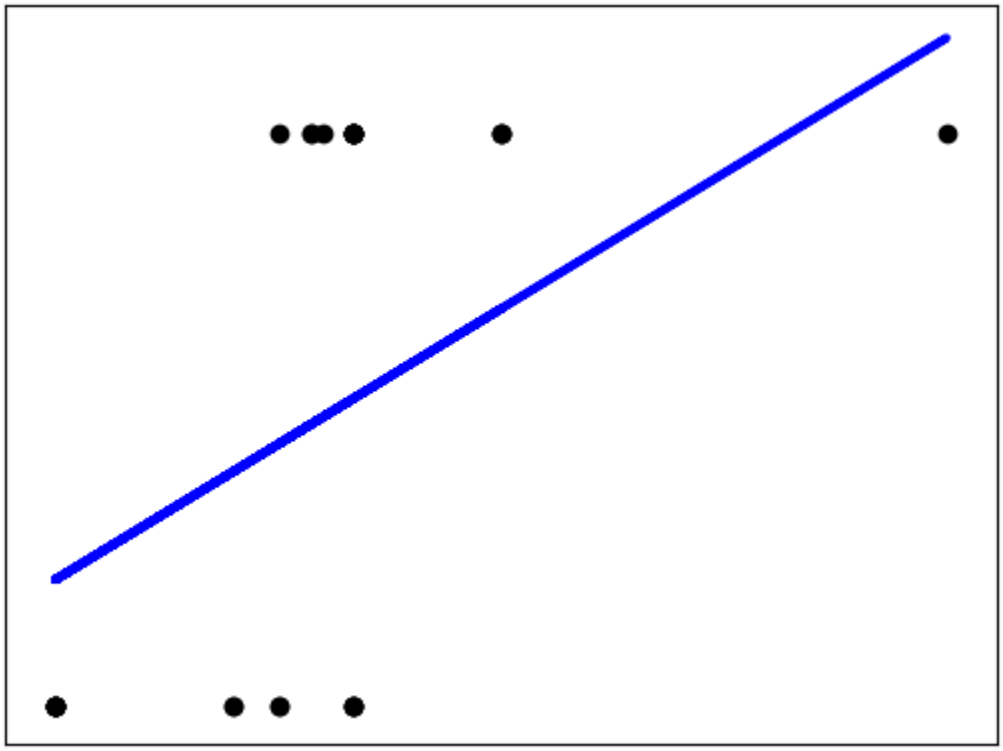
Ejercicio 5

En el código dedicado a los algoritmos de aprendizaje automático (archivo algoritmosIA.py en la carpeta ml) desarrollamos una función que lee y parsea la información dada en json y una función por cada algoritmo de aprendizaje. Además, hemos añadido otra función que saca un gráfico comparando las predicciones de los diferentes algoritmos con la predicción real.

Para realizar un método de Regresión Lineal tenemos que hacer uso del ratio entre servicios inseguros y servicios totales de cada dispositivo, ya que no se puede pasar como parámetro un array de más de una dimensión. Tras crear el modelo de sklearn y usarlo tenemos un resultado de 14,3 dispositivos predecidos como inseguros (donde 15 es el número real) y el error cuadrático medio es de 0,17:



Además, mostramos un gráfico que representa la regresión lineal:

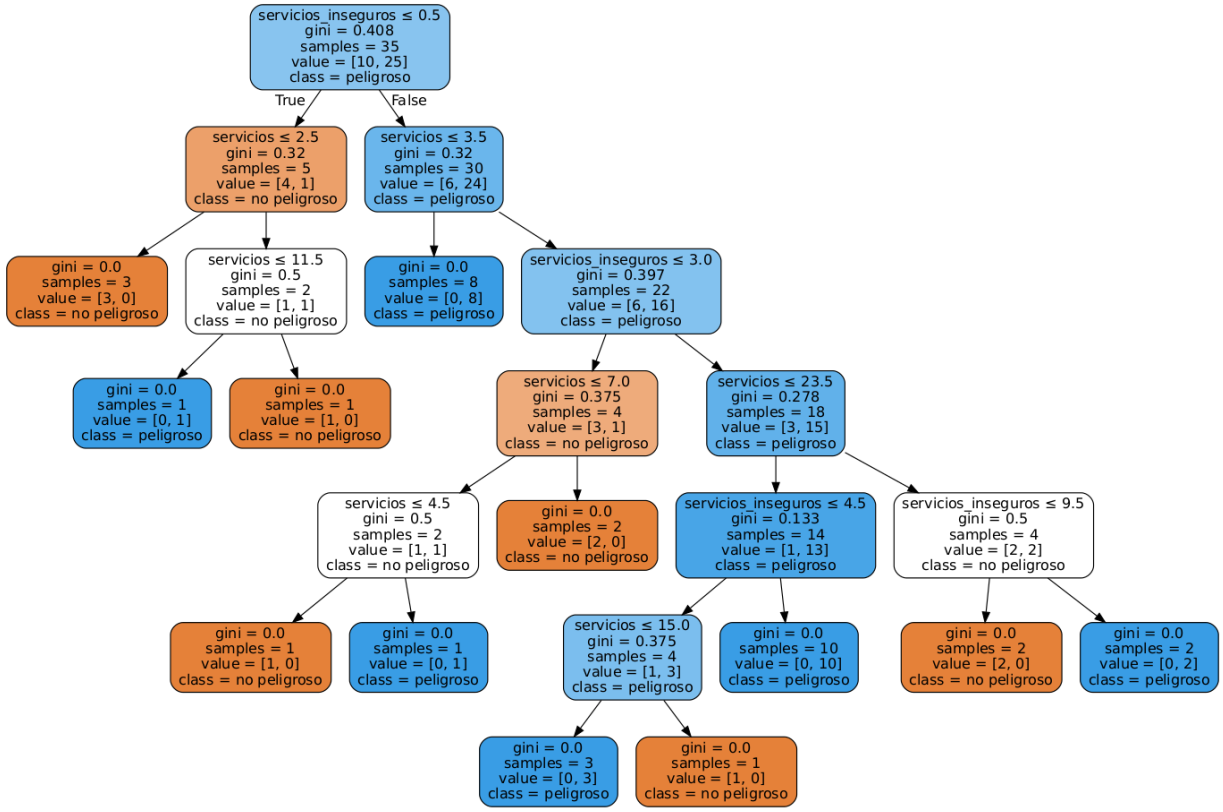


Como se puede ver, los valores se distribuyen por el plano que los separa, clasificándolos mediante una línea recta.

Sobre el método de árbol de decisión (decisión tree) lo que hacemos es simplemente pasarle el array con distintos atributos de los datos de entrenamiento, así como su etiqueta al modelo de sklearn de DecisionTreeClassifier que realiza un árbol de decisión sobre el que se prueban los datos de test. Como resultado da una predicción de 20 dispositivos inseguros, con una precisión del 57%:

```
-----------------
ÁRBOL DE DECISIÓN
-----------------
Predicción de número de dispositivos peligrosos: 20
Número de dispositivos peligrosos real: 15
Precisión de las predicciones: 0.5666666666666667
```

El árbol de decisión que queda, mostrado con graphviz, es el siguiente:



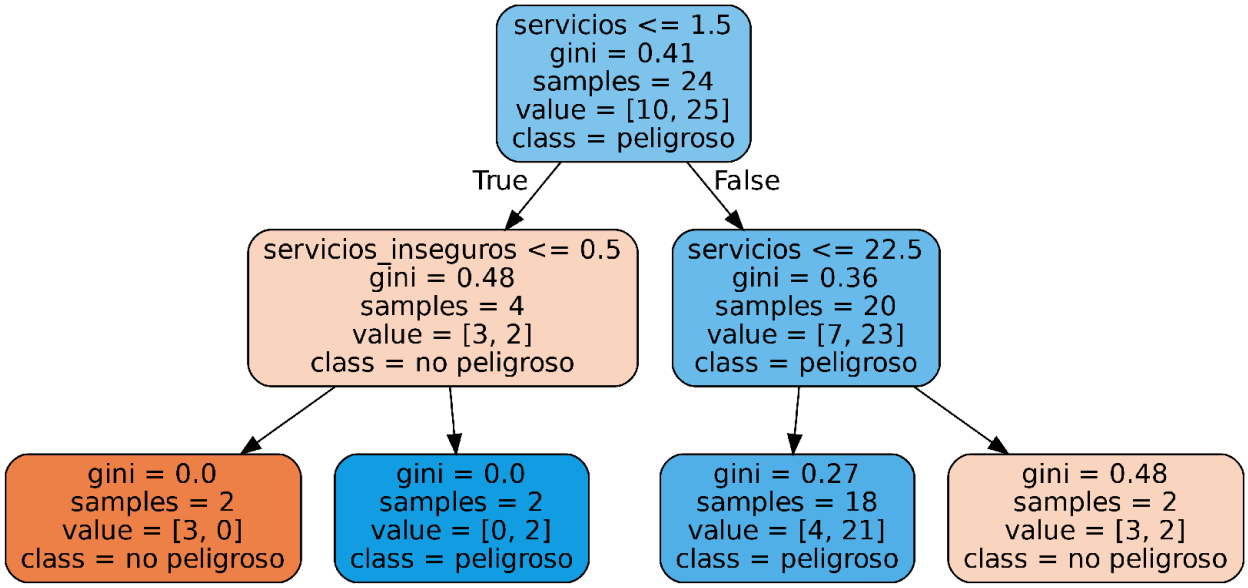
Por otro lado, en el método de random forest, que usa el modelo RandomForestClassifier, usamos un bucle while con una barra de carga, mientras se generan los archivos gráficos de los diferentes árboles de decisión del bosque. El número de dispositivos inseguros predecidos en este caso es de 23, con una precisión del 73%. Los gráficos generados no se muestran, sino que se guardan como archivos, por ser muchos.

```
-------------
RANDOM FOREST
-------------
Predicción de número de dispositivos peligrosos: 23
Precisión de las predicciones: 0.7333333333333333
Generando archivos gráficos...

|                                                  | 10/10 [100%] in 6.0s (1.57/s)

Archivos guardados como tree<numero>.png
```

Tras la ejecución, uno de los árboles sería el siguiente:



Por último, generamos un gráfico que compara los resultados de los diferentes métodos de clasificación y el resultado real (la etiqueta de los datos). En este gráfico, para cada dispositivo y cada método, se colorea en azul si se predice como inseguro (1) y no se colorea si se predice como seguro (0). Un ejemplo de un dispositivo donde todos los métodos aciertan sería 'pc_pepe' y uno donde todos fallan sería 'proxy1':