Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Grado en Ingeniería Informática

Aprendizaje Automático para Detección de Tráfico IoT Anómalo, Spam y Malware

Machine Learning for Anomalous IoT Traffic, Spam and Malware Detection

Bárbara Valentina García Deus

La Laguna, *10* de *junio* de 2021

D. **Pino Caballero Gil**, con N.I.F. 45.534.310-Z Catedrática de Universidad adscrita al Departamento de Ingeniería Informática y Sistemas de la Universidad de La Laguna, como tutora

D. **Carlos Rosa Remedios**, con N.I.F. 43.786.084-H responsable de la Unidad de Tecnologías de la Información y la Comunicación de Gestión de Servicios para la Salud y Seguridad en Canarias, como cotutor

C E R T I F I C A (N)

Que la presente memoria titulada:

“Aprendizaje Automático para Detección de Tráfico IoT Anómalo, Spam y Malware”

ha sido realizada bajo su dirección por D. **Bárbara Valentina García Deus**,

con N.I.F. 43.834.712-R.

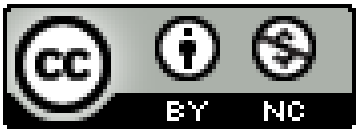
Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 10 de junio de 2021

Agradecimientos

En primer lugar, agradecer a mi tutora, Pino Caballero Gil, y a mi cotutor, Carlos Rosa Remedios, por ayudarme y guiarme en todos los pasos de la elaboración de este trabajo.

También me gustaría dar las gracias a toda mi familia que me ha apoyado, incluso desde la distancia. En especial, agradecer a mi hermano, por estar siempre a mi lado; y, a mi madre, ya que ha sido y sigue siendo un pilar fundamental. Es la persona que me ha ayudado en todas las etapas de mi vida, me ha apoyado incondicionalmente, y ha hecho mi camino siempre más fácil. Es gracias a ella y solo a ella que he llegado hasta aquí.

Licencia



Resumen

El objetivo del presente trabajo es analizar diferentes aplicaciones del aprendizaje automático en el campo de la ciberseguridad.

Como punto de partida se realiza una breve introducción a los conceptos básicos del aprendizaje automático, las herramientas más sencillas para comenzar este tipo de análisis y la construcción de modelos.

Dada la amplia variedad de contextos con los que se relaciona la ciberseguridad, se han elegido tres de especial impacto en la seguridad de las organizaciones: los dispositivos IoT, el spam y el malware.

La eclosión exponencial de los dispositivos IoT los ha convertido en origen de los ataques más graves en los últimos años. Por ello, la detección temprana de anomalías en este contexto resulta fundamental. Para este caso se ha empleado un algoritmo de aprendizaje no supervisado.

En el caso del spam, vía de entrada de grandes estafas y de infección masiva de equipos informáticos, se ha optado por el análisis del contenido de los correos empleando varios modelos diferentes.

Por último, se ha trabajado sobre un repositorio de ficheros de diferentes tipos de malware para tratar de discernir en qué casos podemos identificar no solo si se trata de elementos maliciosos, sino también para tratar de identificar de qué tipo se trata en cada caso.

Palabras clave: Aprendizaje automático, spam, malware, IoT.

Abstract

The objective of this work is to analyze different applications of machine learning in the field of cybersecurity.

As a starting point, there is a brief introduction to the basic concepts of machine learning, the simplest tools to start this type of analysis and the construction of models.

Given the wide variety of contexts with which cybersecurity is related, three have been chosen with a special impact on the security of organizations: IoT devices, spam and malware.

The exponential emergence of IoT devices has made them the source of the most serious attacks in recent years. Therefore, early detection of anomalies in this context is essential. For this case, an unsupervised learning algorithm has been used.

In the case of spam, the entry point for large scams and massive infection of computer equipment, the analysis of the content of emails has been chosen using several different models.

Finally, we have worked on a repository of files of different types of malware to try to discern in which cases we can identify not only if they are malicious elements, but also to try to identify what type it is in each case.

Keywords: Machine learning, spam, malware, IoT.

Índice general

Índice de Figuras

Índice de tablas

Capítulo 1 Introducción

1.1 Motivación

Existen muchos problemas por resolver en el ámbito de la ciberseguridad. Algunos de ellos son: anomalías en tráfico de red que pueden suponer una amenaza para un sistema informático, el envío de emails no deseados para lanzar ataques de phishing a cualquier tipo de usuario, y el malware que puede afectar a cualquier ordenador, habitualmente a aquellos que trabajan y guardan información importante, como por ejemplo, directivos de empresas. La solución que se plantea en el presente trabajo es aplicar el aprendizaje automático, una rama dentro de la inteligencia artificial.

La inteligencia artificial se utiliza hoy en día en muchos ámbitos: sanitarios, educativos, medioambientales, etc. La ciberseguridad es uno más de ellos. La IA es una de las principales herramientas que hoy en día utilizan las empresas y organizaciones para protegerse ante todo tipo de ataques informáticos. Concretamente, con el aprendizaje automático se puede, entre muchas más cosas, detectar anomalías en tráfico de red, detectar malware en ficheros, y detectar spam.

Actualmente las empresas y organizaciones están invirtiendo grandes cantidades de dinero y recursos en aplicar inteligencia artificial para la seguridad de sus infraestructuras y servicios. Igualmente hacen los atacantes, mejorando sus estrategias. Es por esto que aplicar la inteligencia artificial en la ciberseguridad se ha vuelto fundamental.

Este trabajo se centra en el uso del aprendizaje automático para la clasificación de tráfico de dispositivos IoT, detección de spam y de malware.

1.2 Objetivos

Los objetivos principales de este trabajo son:

- Comprender los conceptos básicos del aprendizaje automático.
- Construir datasets con la información que se necesita en cada uno de los casos que se estudian en este trabajo.
- Aplicar los diferentes algoritmos de aprendizaje automático en cada uno de los casos relacionados con la ciberseguridad, y comprender el resultado obtenido.

1.3 Estado del arte

El aprendizaje automático, también conocido con el término en inglés, machine learning, permite que el usuario alimente un algoritmo informático con una gran cantidad de datos, a partir de los cuales el ordenador será capaz de tomar decisiones basándose únicamente en los datos introducidos (Figura 1.1).

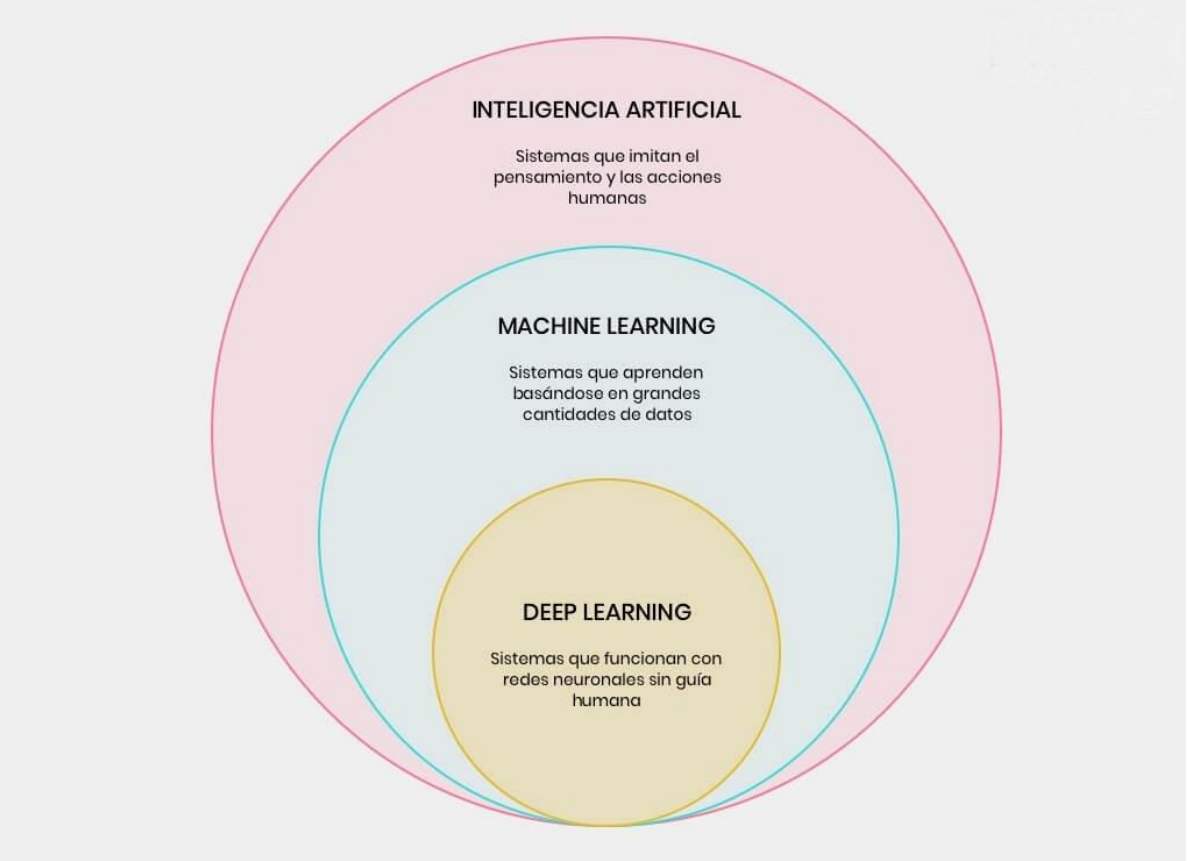


Figura 1.1: Diferencias entre la inteligencia artificial, machine learning y deep learning.

En el campo de la ciberseguridad se utiliza en técnicas variadas como antispam o para detectar emails anómalos. En este caso se basa en que estos emails cumplen ciertos criterios definidos: contienen faltas de ortografía, usan mayúsculas en palabras clave, usan los símbolos del dólar o el euro, y características similares.

También se aplica en la detección de anomalías en redes. Lo que se hace en este caso es monitorizar el tráfico de red, es decir, el modelo de machine learning estudia el comportamiento habitual del tráfico y establece un patrón. Lo que capte que se salga de dicho patrón, lo detecta como anómalo. Es importante que un humano estudie estos resultados, porque no todas las anomalías que se puedan producir tienen porque ser un ataque al sistema informático. Otra aplicación similar es la detección de patrones anómalos en el comportamiento humano (UEBA, *User and Entity Behavior Analytics*). Consiste en monitorizar los hábitos de personas pertenecientes a una red; es decir, controlar a qué servidores accede, en qué horarios, desde qué dispositivos, la ubicación, etc. Si se halla algo extraño, se detecta como anomalía.

La resolución de captchas se puede romper con inteligencia artificial. Se pueden utilizar redes neuronales de reconocimiento de caracteres e identificar el captcha de texto. Los de audio también se pueden romper porque existen librerías de reconocimiento de voz. Y aunque la entrada de texto para introducir las soluciones esté controlada para detectar si lo está escribiendo un humano, una máquina es capaz de introducir retardos entre los caracteres que introduce para imitar la escritura humana.

1.4 Fases

El primer paso del presente trabajo consistió en buscar las actuales aplicaciones de la inteligencia artificial y del aprendizaje automático en el ámbito de la ciberseguridad. Este paso fue importante para comenzar a hacer un esquema de en qué basar y continuar el trabajo.

Para la detección de anomalías de tráfico IoT, se buscó un dataset que contuviera dicho tráfico. Se prepararon y limpiaron los datos para que el algoritmo fuera capaz de entenderlos. Con el algoritmo se crea el modelo que identifica que tráfico considera anómalo y cuál no.

Para la detección de spam se hicieron dos cosas. Primero, buscar un dataset con el contenido del email y la etiqueta de si es spam legítimo o no. Y segundo, buscar un repositorio con emails clasificados como legítimos o no, y crear un dataset a partir de ellos. En ambos casos, se construyen y se entrenan modelos utilizando diferentes algoritmos, obteniendo así distintos resultados.

El siguiente paso es la detección de malware. Se partió de un repositorio con diferentes tipos de malware. Y en este caso, se crearon diferentes datasets teniendo en cuenta diferentes características de los ficheros, tratando de averiguar cuál era la mejor estrategia. En todos los casos, se construyen y se entrenan diferentes modelos.

1.5 Estructura de la memoria

En el capítulo 2 se realiza una escueta introducción a la solución propuesta, tratando por separado y brevemente los tres casos de estudio en los que se aplicó el aprendizaje automático. También se detallan los conceptos del aprendizaje automático que son necesarios para comprender el resto del trabajo y las herramientas y tecnologías empleadas y su utilidad. Desde el capítulo 3 al 5, se especifican las soluciones para cada uno de los casos de estudio previamente mencionados. Los capítulos 6 y 7 corresponden a las conclusiones y líneas futuras de trabajo, tanto en inglés como en español. El capítulo 8 es el presupuesto. Y, por último, el capítulo 9 contiene la bibliografía.

Capítulo 2 Introducción a la solución

En el capítulo anterior se ha introducido la importancia de la Inteligencia Artificial en la ciberseguridad, y también se ha comentado que este trabajo se centra en el uso del aprendizaje automático, que es una rama de la inteligencia artificial, y sus aplicaciones en el mismo ámbito. Para ello, se ha aplicado dicho aprendizaje en los casos prácticos que a continuación se detallan. En este mismo capítulo se describen los conceptos básicos del aprendizaje automático necesarios para la comprensión del trabajo.

2.1 Casos estudiados

2.1.1 Tráfico IoT

Los dispositivos IoT se utilizan cada vez más en la vida cotidiana y al estar conectados a Internet, son un nuevo blanco de ataques de los ciberdelincuentes, ya que no disponen de fuertes medidas de seguridad debido a sus recursos limitados. Es importante entonces centrarse en su seguridad.

Estos dispositivos son vulnerables, principalmente, porque sus capacidades de computación son limitadas, ya que tienen funciones específicas que requieren de recursos limitados, dejando poco espacio para añadir un sistema de seguridad. Es por este motivo que una mejor solución consistiría en controlar y monitorizar la red en la que estos dispositivos se encuentran conectados, en lugar de intentar proteger cada uno independientemente. Hacer esto último tiene el principal inconveniente de que cada dispositivo es diferente, con características y funcionalidades heterogéneas; por lo que habría que desarrollar un software capaz de suplir las necesidades de cada tipo distinto de dispositivo IoT.

2.1.2 Detección de Spam

El spam está formado por todos los emails que se envían a muchísimas personas y se reciben aunque no se hayan solicitado. La mayoría de las veces, el spam es de naturaleza comercial y, aunque es preocupante, no es necesariamente malicioso o fraudulento.

Sin embargo, a pesar de esto, el spam es la principal vía de distribución de malware y de ataques de phishing. Teniendo en cuenta que, según los datos de Cisco [1], el spam supone el 85.39% de los e-mails enviados, detectar y desechar estos e-mails no deseados es importante y necesario.

Todos los e-mails no deseados tienen algunas similitudes entre ellos que facilitan su detección. Son enviados a varias personas, por lo que el número de remitentes es elevado. Suelen contener los símbolos del dólar ($) o del euro (€), y del porcentaje (%), ofreciendo algún tipo de oferta o descuento llamativo; y también incluyen enlaces URL a otros sitios web, entre otras características.

2.1.3 Detección de Malware

El malware es cualquier tipo de software que intencionalmente realiza operaciones dañinas en un sistema informático. Los ejemplos típicos de estas actividades maliciosas son el robo de información, el daño o mal funcionamiento del sistema informático, acceder a datos bancarios, chantajear al propietario de los datos con la publicación o divulgación de los mismos, permitir que usuarios no autorizados accedan al sistema informático, causar molestias o una combinación de varias de estas acciones.

En este trabajo se hará una clasificación de cuatro tipos diferentes de malware.

APT1

APT es un acrónimo, en inglés, de *Advanced Persistent Threat* (Amenaza Avanzada Persistente). Este concepto se hizo famoso después de los ataques de la unidad militar china (llamada APT1) a redes de diferentes medios. Una característica que distingue a las APT, es que no atacan de forma aleatoria, sino que tienen un objetivo concreto. Partiendo de que no es tan fácil atacar un ordenador de un alto directivo de una empresa importante, estos ataques se centran en otros objetivos más sencillos: empleados de menor rango, los cuales no tienen acceso a información valiosa, pero que utilizan la misma red.

Crypto

El malware crypto cifra los archivos de un equipo, para que el dueño de dichos archivos no sea capaz de abrirlos sin conocer la clave de descifrado. Para obtener esta clave y acceder a sus datos, se le pedirá un rescate económico.

Locker

El ransomware locker lo que hace es bloquear el acceso a un dispositivo, es decir, impide que un usuario pueda entrar en su ordenador. Este problema es aún mayor que el anterior, ya que no solo no se podría acceder a los archivos, si no que no se puede acceder al equipo: la interfaz de usuario se bloquea. Como en el caso anterior, a cambio de permitir que la víctima recupere su dispositivo, se le pedirá un rescate económico.

Zeus

Entre los virus que se pueden encontrar en Internet, se puede determinar que los troyanos son los más peligrosos porque pueden hacerse pasar por aplicaciones aparentemente legítimas mientras roban cualquier tipo de información o destruyen el sistema operativo. Zeus es un troyano. Una vez dentro de un ordenador, el virus Zeus tiene la capacidad de extraer casi cualquier información del sistema, como contraseñas, datos bancarios e información personal de redes sociales, y transmitirla a terceros.

2.2 Tipos de aprendizaje

Los algoritmos de aprendizaje automático pueden ser de aprendizaje supervisado, no supervisado o aprendizaje por refuerzo. Este último tipo de aprendizaje no se ha aplicado en ninguno de los casos prácticos, por lo que no se ha entrado en su explicación.

2.2.1 Aprendizaje supervisado.

El aprendizaje supervisado utiliza datos etiquetados. Es decir, el algoritmo se entrena dándole unas preguntas (llamadas características) y una respuesta (llamada etiqueta). Esto se hace para que el algoritmo los combine y pueda hacer predicciones.

Por ejemplo, en un ámbito sanitario, las características pueden ser los síntomas, y la etiqueta es la enfermedad correspondiente a esos síntomas.

2.2.2 Aprendizaje no supervisado.

A diferencia del aprendizaje supervisado, el aprendizaje no supervisado trabaja con datos no etiquetados. Recibe las características, pero no la etiqueta. Su función es catalogar y hacer asociaciones por analogía de los datos.

Dicho de otro modo, estos algoritmos identifican patrones en los datos, pero no los clasifican. Por lo que tiene el inconveniente de que se necesita una persona que sea capaz de interpretar los resultados, y se pierde la independencia del ordenador de no necesitar trabajo humano.

2.3 Tipos de algoritmos

Existen muchos algoritmos tanto en el aprendizaje supervisado como en el no supervisado. A continuación sólo se explican los algoritmos que se han utilizado en los tres casos de estudio. Los algoritmos Random Forest, Naive Bayes y el de regresión logística, son algoritmos de aprendizaje supervisado. El algoritmo Isolation Forest es un algoritmo de aprendizaje no supervisado.

2.3.1 Random Forest

Antes de explicar cómo funciona el algoritmo random forest, se explica cómo funciona un árbol de decisión (en inglés, Decision Tree) mediante un ejemplo sencillo y comprensible. En la Figura 1.1 hay un conjunto de datos: Figuras geométricas de diferentes colores, y formas. Si se quiere clasificar las Figuras para agrupar todas las que son iguales, el árbol de decisión podría comenzar como se muestra en la Figura 1.2. Después de esta división, la clasificación de la izquierda no se puede subdividir. Sin embargo, la de la izquierda sí (Figura 1.3). Y finalmente se obtienen los datos clasificados.

Figura 1.1: Random Tree. Conjunto de datos que se quieren clasificar.

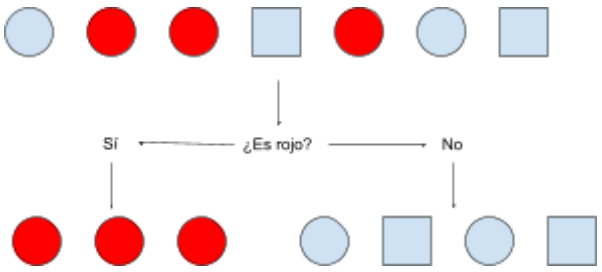


Figura 1.2: Random Tree. Primera iteración.

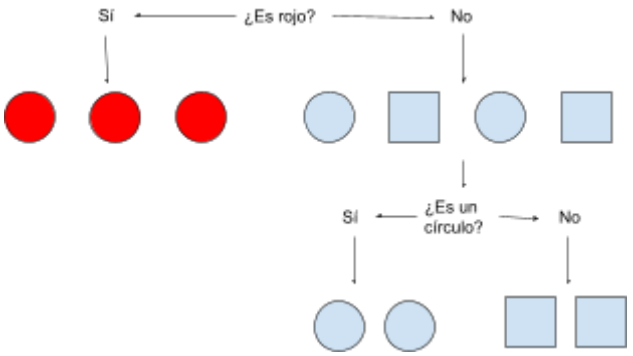


Figura 1.3: Random Tree. Segunda iteración.

Una vez comprendido el concepto de árbol de decisión, se explica a continuación cómo funciona el algoritmo de Random Forest. Este algoritmo crea no uno, sino varios árboles de decisión para clasificar un nuevo dato, basado en atributos. Cada árbol da una clasificación, independiente de los demás árboles, y finalmente el "bosque" elige la clasificación de ese nuevo dato según el resultado que más se haya obtenido en los diferentes árboles.

Esto se ve claramente en el siguiente ejemplo en el que se quiere clasificar si la Figura es un cuadrilátero, un triángulo o un círculo y para ello se construyen tres árboles de decisión (Figura 1.4).

Partiendo de los clasificaciones obtenidas, se puede observar lo siguiente:

- El cuadrado ha sido las tres veces clasificado correctamente como cuadrilátero.
- El triángulo con el ángulo recto, ha sido clasificado dos veces como triángulo y solo una como cuadrilátero, por lo que el algoritmo determina que es un triángulo.
- El triángulo equilátero, ha sido clasificado dos veces como triángulo y solo una vez como cuadrilátero por lo que se determina que es un triángulo también.
- El círculo ha sido clasificado las tres veces como círculo.
- Por último, el rectángulo ha sido clasificado dos veces como cuadrilátero, y solo una como círculo, por lo que se determina que es un cuadrilátero.

El bosque logrado ha conseguido clasificar correctamente las cinco Figuras.

El éxito del Random Forest se halla en la baja correlación entre los diferentes árboles que se generan. Los árboles se protegen unos a otros de sus propios errores (siempre que no sigan cometiendo errores en la misma dirección). Aunque algunos árboles pueden estar equivocados, muchos otros serán correctos, por lo que el bosque en su conjunto tomará la decisión acertada [2].

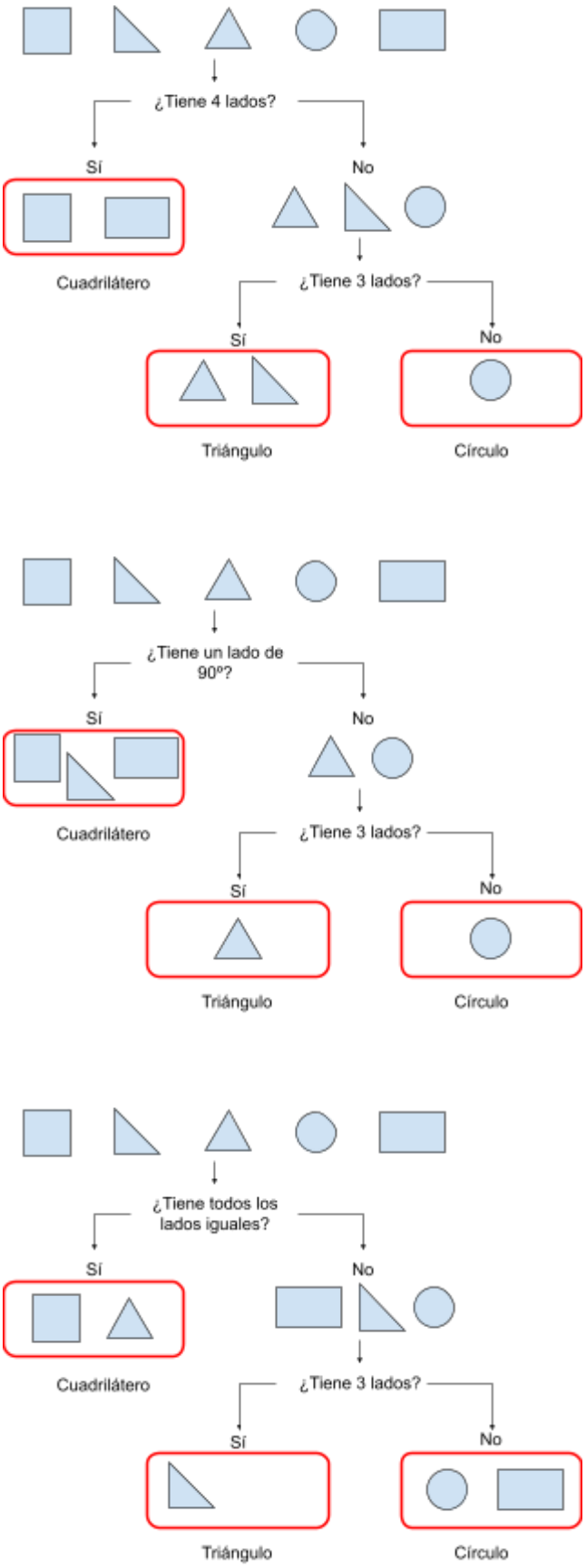


Figura 1.4: Random Forest. Ejemplo.

2.3.2 Naive Bayes

El algoritmo de Naive de Bayes asume que la existencia de una característica en particular no tiene nada que ver con la existencia de otras características. Por ejemplo, si un animal tiene 4 patas, pelo y es vertebrado, puede considerarse un mamífero. El algoritmo de Naive de Bayes cree que, independientemente de la presencia o ausencia de otras características, cada una de estas características afecta independientemente a la probabilidad de que el animal sea un mamífero.

De nuevo, se explicará cómo funciona este algoritmo mediante un ejemplo.

A dos sujetos, Alice y Bob, les gusta montar en bicicleta. Alice lo hace cinco veces a la semana, y Bob tres veces. Esta es nuestra información anterior.

Hoy, nosotros hemos visto un ciclista, pero no hemos sido capaces de distinguir si era Alice o era Bob. La probabilidad de que sea uno u otro es la siguiente:

P(A) = 5 días / 7 días de la semana = 0.71

P(B) = 3 días / 7 días de la semana = 0.43

El ciclista llevaba una camiseta naranja. Nosotros sabemos que Alice lleva una camiseta de ese color una vez a la semana, y Bob, 2 veces. Con esta información, obtenemos la probabilidad de que Alice lleve una camiseta naranja, y la probabilidad de que Bob lleve una camiseta naranja. Esta es nuestra información posterior.

P(Naranja | Alice) = 1 / 7 = 0.14

P(Naranja | Bob) = 2 / 7 = 0.29

De esta forma obtenemos las probabilidades reales:

$$P(Alice \mid Naranja) = \frac{P(A)\ x\ P(Naranja \mid Alice)}{P(A)\ x\ P(Naranja \mid Alice)\ + P(B)\ x\ P(Naranja \mid Bob)} = \frac{0.71\ x\ 0.14}{(0.71\ x\ 0.14) + (0.43\ x\ 0.29)} = 0.44$$

$$P(Bob \mid Naranja) = \frac{P(A)\ x\ P(Naranja \mid Bob)}{P(A)\ x\ P(Naranja \mid Alice)\ + P(B)\ x\ P(Naranja \mid Bob)} = \frac{0.43\ x\ 0.29}{(0.71\ x\ 0.14) + (0.43\ x\ 0.29)} = 0.57$$

El resultado muestra que lo más probable es que hayamos visto a Bob.

Las principales desventajas de utilizar y aplicar este algoritmo es que, por ejemplo, si una característica es irrelevante, se estarán realizando cálculos que no aporten al resultado. En este caso, una característica irrelevante podría ser que supiéramos que Alice cocina cinco veces por semana y Bob, ocho. Son datos que no aportan. La otra desventaja es que si el conjunto de datos de prueba tiene una característica que no ha sido observada entre los datos de prueba, el modelo le asignará una probabilidad de cero, y tampoco aportará nada al resultado, a pesar de que la característica pueda o no ser relevante [3].

2.3.3 Regresión logística

La regresión logística se basa en la función logística, de la que se obtiene un valor binario entre 0 y 1.

Con la regresión logística se mide la relación entre la variable dependiente (la afirmación que se desea predecir) con una o más variables independientes (el conjunto de características). Para ello, utiliza la función logística para determinar la probabilidad de la variable dependiente. Esta probabilidad se ha de traducir en valores binarios, y es por esto por lo que se utiliza un valor umbral. Los valores por encima del umbral indican que la afirmación es cierta y por debajo que es falsa. Generalmente este valor es 0,5, aunque se puede aumentar o reducir, según interese o sea necesario.

A la función que relaciona la variable dependiente con las independientes también se le conoce como función sigmoidea. Esta función es una curva con forma de S que puede tomar cualquier valor entre 0 y 1, pero nunca valores fuera de estos límites. Tiene la siguiente forma:

$$f(x) = \frac{1}{1 + e^{-x}}$$

Por ejemplo, en el campo de la ciberseguridad, en la detección de spam, las variables independientes pueden ser el número de veces que aparecen determinadas palabras en cada uno de los emails de la muestra, y el resultado, determinaría si este email es spam o no. Teniendo en cuenta que el 0 es ham y el 1 es spam, y que a partir del 0.5 se considera spam, si el resultado de la función es 0.26, se determina que el email no es spam.

Las principales ventajas de utilizar la regresión logística son que los resultados son fácilmente interpretables, y que no es necesario disponer de grandes recursos computacionales, tanto en entrenamiento como en ejecución, además, esta técnica es eficaz y simple.

Una desventaja que afecta a uno de los casos estudiados, concretamente, el de detección de malware, es que al ser un algoritmo binario, es decir, solo identifica entre dos clasificaciones, no es aplicable en el caso de detectar y diferenciar cuatro tipos diferentes de malware [4].

2.3.4 Isolation Forest

Isolation Forest es un algoritmo utilizado para identificar anomalías cuando los datos no están etiquetados, es decir, cuando no se conoce la clasificación real (anomalía o no anomalía) de los datos proporcionados. Se basa en el principio de que los datos anómalos son más fáciles de aislar mediante el particionado recursivo del dataset [5].

Antes de explicar el funcionamiento del Isolation Forest, es necesario comprender el algoritmo isolation tree:

1. Se crea un nodo raíz que contiene las **N** observaciones de entrenamiento.
2. Se selecciona aleatoriamente un atributo ***i*** y un valor aleatorio ***a*** dentro del rango observado de ***i***.
3. Se crean dos nuevos nodos, separando las observaciones según si $x_i \leq a$ o $x_i > a$.
4. Se repiten los pasos 2 y 3 hasta que todas las observaciones quedan aisladas de forma individual en nodos terminales.

El modelo Isolation Forest se obtiene al combinar múltiples isolation tree. El valor

predicho para cada observación es el número de divisiones promedio que se han necesitado para aislar dicha observación en el conjunto de árboles. Cuanto menor es este valor, mayor es la probabilidad de que se trate de una anomalía [6].

Esta explicación es más fácil de entender a través de un ejemplo muy sencillo.

Los datos del ejemplo son los del gráfico de la Figura 2.1.

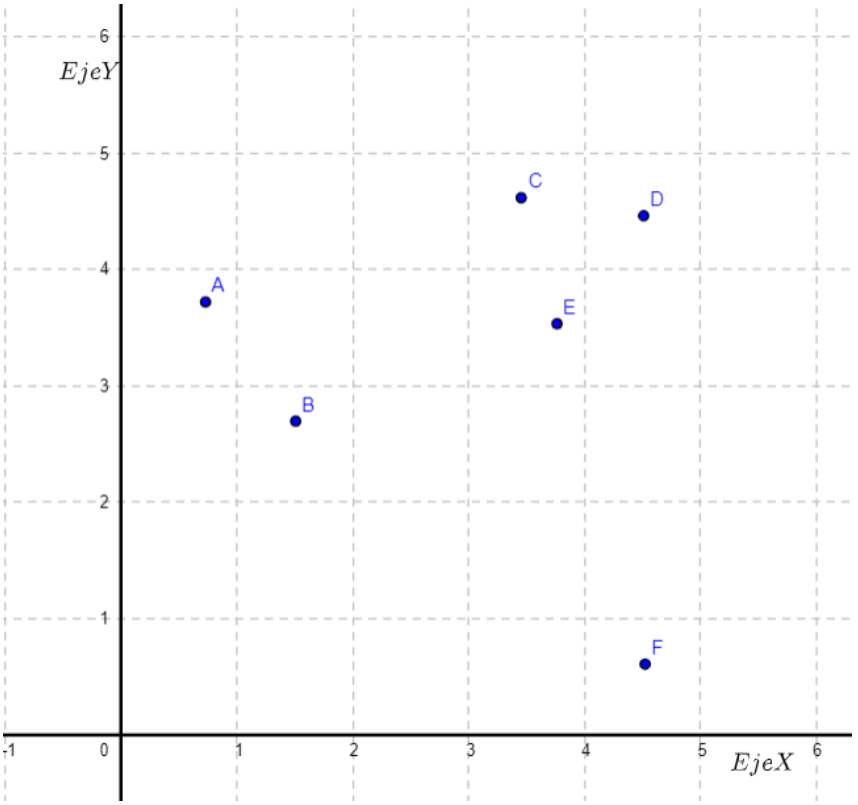


Figura 2.1: Conjunto de observaciones.

Se crea el primer isolation tree. El nodo raíz contiene todas las observaciones, se selecciona el atributo que corresponde con el eje de las X y el valor 1 y se crean dos nuevos nodos atendiendo a si el valor de la observación es mayor que 1, es decir, si x > 1 (Figura 2.2). De esta forma se ha aislado una de las observaciones. El proceso continúa hasta que todas las observaciones queden aisladas (Figura 2.3).

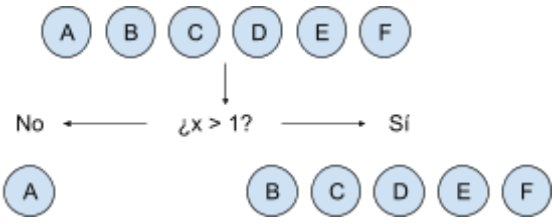


Figura 2.2: Primera división de las observaciones.

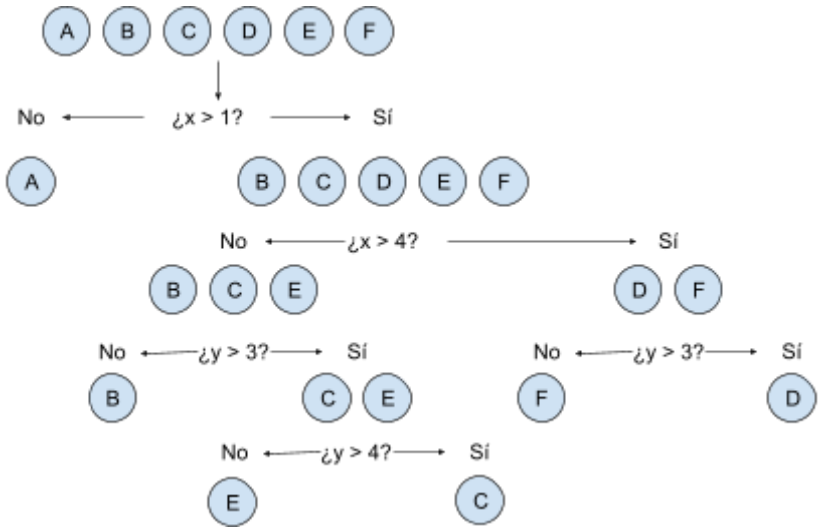


Figura 2.3: Primer isolation tree.

Otro isolation tree sería el de la Figura 2.4.

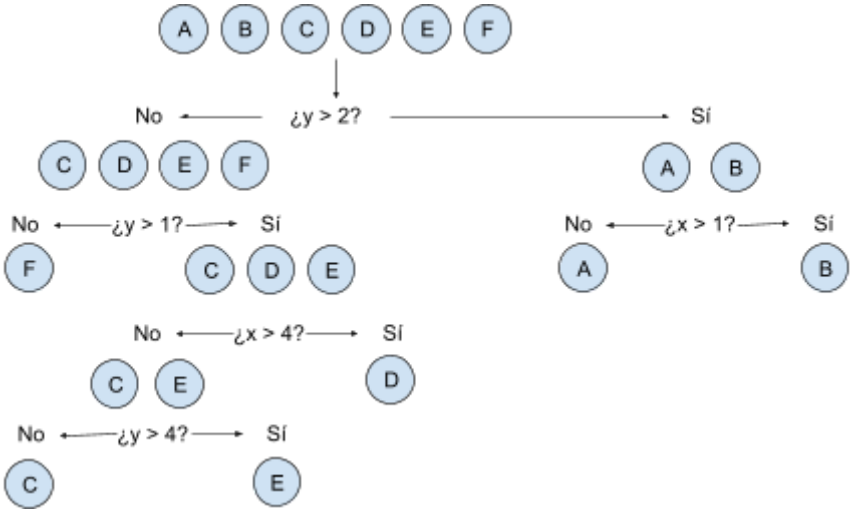


Figura 2.4: Segundo isolation tree.

Con estos dos isolation trees se observa que:

- La observación A se aísla en el primer árbol en la primera división, y en el segundo, en la segunda división.
- La observación B se aísla primero en la tercera división y luego en la segunda.
- La observación C se aísla en la cuarta división las dos veces.
- La observación D se aísla también en la tercera división las dos veces.
- La observación E se aísla también en la cuarta división las dos veces.
- La observación F se aísla en la tercera y luego en la segunda división.

Por lo tanto, se puede concluir que la observación A es más probable que sea anómala, y que las observaciones C y E no lo sean.

2.4 Interpretación de los resultados

Los resultados obtenidos de los algoritmos del aprendizaje supervisado se pueden mostrar de una forma clara y entendible mediante una matriz de confusión. Esta matriz relaciona los resultados que el proceso ha obtenido, con el resultado correcto.

En la Figura 3.1, se observa a la izquierda, que cuando el resultado debería haber sido “0”, se obtuvo 215 aciertos, y 8 errores; y que cuando el resultado debería haber sido “1”, se obtuvo 74 aciertos, y 22 errores.

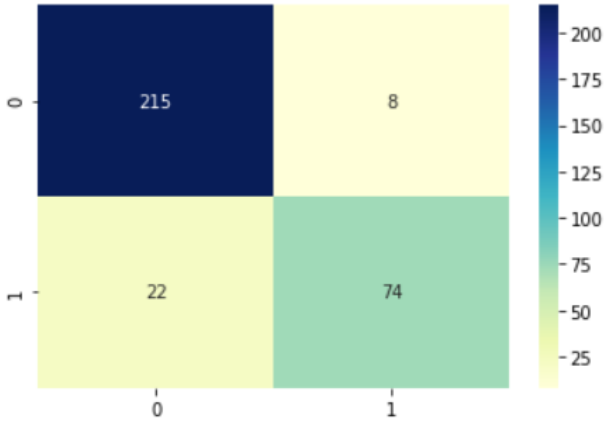


Figura 3.1: Matriz de confusión.

En otras palabras, la diagonal principal son las predicciones que se han realizado correctamente. Los otros dos cuadrantes son predicciones incorrectas.

Para entender mejor los resultados de la matriz de confusión, es necesario previamente definir cuál es la clase positiva, es decir, aquello que se quiere detectar. Por ejemplo, siguiendo con la Figura 3.1, dicha matriz representa la clasificación de spam y ham de uno de los casos que se han estudiado, y que se comentará más adelante. El “0” representa ham (correo legítimo) y el “1” el spam. Por lo tanto, la clase positiva, lo que se quiere detectar, es el spam. Por consiguiente, los verdaderos positivos y los verdaderos negativos son las clasificaciones que se han hecho correctamente (la diagonal principal). Los falsos positivos, son los correos de ham que se han clasificado como spam, es decir, el cuadrante superior derecho; y luego los falsos negativos son los correos spam que se han clasificado como ham, el cuadrante inferior izquierdo (Figura 3.2).

Verdadero negativo	Falso positivo
Falso negativo	Verdadero positivo

Figura 3.2: Esquema de la matriz de confusión.

Se comprende fácilmente que los “falsos negativos” son peores resultados que los “falsos positivos”. Dicho de otro modo, aunque ambas clasificaciones son erróneas, es peor clasificar como ham un correo de spam, que clasificar como spam un correo ham.

Existen métricas que permiten valorar si el modelo construido es bueno o no. La principal es la precisión (en inglés, *Accuracy Score*), que indica las muestras correctamente clasificadas sobre el total. Esta métrica es la que se utiliza para dar por válido un modelo.

$$precisión = \frac{verdaderos\ positivos + verdaderos\ negativos}{verdaderos\ positivos + verdaderos\ negativos + falsos\ positivos + falsos\ negativos}$$

Otra de las métricas es la exactitud (en inglés, *Precision*), que indica qué porcentaje de las muestras clasificadas como pertenecientes a la clase que se le ha indicado, lo era realmente:

$$exactitud = \frac{verdaderos\ positivos}{verdaderos\ positivos + falsos\ positivos}$$

2.5 Importancia de los datos

La obtención de unos buenos resultados depende en su gran mayoría de los datos con los que se trabajen. Según que caso se esté estudiando, puede resultar mejor un algoritmo que otro, pero la diferencia se verá en los datos que se tengan de entrada.

Los datos con los que se trabaje deben de ser voluminosos, es decir, mientras más, mejor. No es lo mismo construir un modelo partiendo de 100 muestras que de 1000.

También, estos datos deben de haber sido preprocesados. Hay que limpiarlos y obtener aquellas características que se crea que puedan ser de mayor utilidad. Mientras con menos características trabaje el algoritmo, más rápido será.

Si la información que se maneja está en diferentes formatos, habría que normalizarla y estandarizarla.

Si el dataset contiene nulos, ya sea porque ese valor no se obtuvo por causa de un error, o porque se ha eliminado, hay que determinar qué hacer con ellos. Se pueden llevar a cabo diferentes técnicas:

1. Eliminar todas las filas del dataset que contengan un valor nulo. Esto tiene el inconveniente de que pueden ser varias las filas que contengan un valor nulo, y por consiguiente, se puede estar eliminando gran parte del dataset.
2. Sustitución ficticia. Consiste en rellenar los nulos, para campos numéricos, con el valor cero; para cadenas string, con “null”, y para variables categóricas, con una nueva clase “desconocido”. En el caso de los campos numéricos, se puede estar alterando la información de forma inadecuada.
3. Sustitución media. Se aplica a campos numéricos solamente, ya que consiste en rellenar un valor nulo por el valor medio de varios de los otros registros, o por un valor similar a un registro cercano.
4. Sustitución frecuente. Consiste en rellenar el valor nulo por el valor más frecuente.
5. Sustitución de regresión. Se pueden predecir los datos nulos a partir de las demás variables, aplicando el aprendizaje automático.

2.6 Tecnologías y herramientas utilizadas

Scikit-learn

Se ha elegido Python como lenguaje de programación ya que incluye la biblioteca Scikit-learn [7]. Scikit-learn es una librería de software libre que incluye varios algoritmos de aprendizaje supervisado y no supervisado [8].

Kaggle

Kaggle es una comunidad de científicos de datos y profesionales del aprendizaje automático, que permite a los usuarios compartir conjuntos de datos [9].

Google Colab y Drive

Para el entorno de programación se ha utilizado Google Colab [10]. Esta herramienta es un servicio en la nube, basado en los Notebooks de Jupyter [11], que permite el uso gratuito de las GPUs. Google Colab no almacena los ficheros y/o datasets que se utilizan en el notebook, pero sincronizándose con Google Drive, podemos guardar estos ficheros en esta herramienta y acceder a ellos de forma fácil.

Figura 4.1: Logos de las herramientas utilizadas.

Capítulo 3 Tráfico IoT

3.1 Introducción

El objetivo de este capítulo es aplicar un algoritmo de aprendizaje no supervisado, concretamente, el algoritmo Isolation Forest, a los datos del tráfico de dispositivos IoT [12] para identificar si este es anómalo o no.

3.2 Procesamiento de los datos

El contenido del dataset es el que se muestra en la Figura 5.1 (sólo se muestran las cinco primeras filas). Las características de los datos son: el número, el momento en el que se captura, y la longitud de cada frame; la dirección MAC e IP tanto de origen como de destino; el número de protocolo IP, la longitud del paquete IP, la longitud de la cabecera TCP, el puerto TCP origen y destino, y el contenido del frame.

	frame.number	frame.time	frame.len	eth.src	eth.dst	ip.src	ip.dst	ip.proto	ip.len	tcp.len	tcp.srcport	tcp.dstport	_ws.col.Info
0	1	Feb 16, 2020 12:37:22.736684743 IST	54	50:02:91:69:66:71	98:22:ef:d5:cc:2f	192.168.0.35	192.168.0.121	6.0	40.0	0.0	49279.0	80.0	49279 → 80 [FIN, ACK] Seq=1 Ack=1 Win=1910 Len=0
1	2	Feb 16, 2020 12:37:22.736773147 IST	62	50:02:91:69:66:71	98:22:ef:d5:cc:2f	192.168.0.35	192.168.0.121	6.0	48.0	0.0	56521.0	80.0	56521 → 80 [SYN] Seq=0 Win=2144 Len=0 MSS=536 ...
2	3	Feb 16, 2020 12:37:22.736824792 IST	62	98:22:ef:d5:cc:2f	50:02:91:69:66:71	192.168.0.121	192.168.0.35	6.0	48.0	0.0	80.0	56521.0	80 → 56521 [SYN, ACK] Seq=0 Ack=1 Win=64240 Le...
3	4	Feb 16, 2020 12:37:22.736836228 IST	54	98:22:ef:d5:cc:2f	50:02:91:69:66:71	192.168.0.121	192.168.0.35	6.0	40.0	0.0	80.0	49279.0	80 → 49279 [FIN, ACK] Seq=1 Ack=2 Win=64025 Len=0
4	5	Feb 16, 2020 12:37:22.749684991 IST	54	50:02:91:69:66:71	98:22:ef:d5:cc:2f	192.168.0.35	192.168.0.121	6.0	40.0	0.0	56521.0	80.0	56521 → 80 [ACK] Seq=1 Ack=1 Win=2144 Len=0

Figura 5.1: Dataset de tráfico de dispositivos IoT.

Para que el algoritmo Isolation Forest sea capaz de procesar los datos es necesario que estos sean números, no cadenas de caracteres. Por este motivo se ha decidido descartar las columnas del contenido del frame, así cómo el momento exacto de su captura.

Por consiguiente, los valores de las direcciones MAC e IP hay que preprocesarlos. Para ello se hace uso de las funciones que se muestran en la Figura 5.2, que convierte dichas direcciones en un número entero [13].

Las demás columnas también hay que pasarlas a formato entero. En el caso de que algún valor de alguna de las columnas sea nulo, hay que tratar con ellos como se explicó en el apartado 2.6: no es suficiente con cambiar los valores nulos por un cero.

En el caso de la columna “ip.proto”, el número de protocolo 0 hace referencia al protocolo *IPv6 Hop-by-Hop Option*. En su lugar se sustituye por el valor 256 que no indica ninguno [14].

```python
def mac_to_int(mac):
    res = re.match('^((?:(?:[0-9a-f]{2}):){5}[0-9a-f]{2})$', mac)
    if res is None:
        raise ValueError('invalid mac address')
    return int(res.group(0).replace(':', ''), 16)

def ip_to_int(ip):
  try:
    addr = ipaddress.ip_address(ip)
    return int(addr)
  except:
    return 0
```

Figura 5.2: Funciones en Python para convertir a número entero, direcciones MAC e IP.

Con respecto a los valores nulos correspondientes al protocolo TCP, el campo que hace referencia a la longitud de la cabecera de dicho protocolo, sí se puede sustituir por un cero. En cambio, los valores de los puertos no. Teniendo en cuenta que los números de los puertos se asignan según tres rangos: puertos del sistema (0-1023), puertos de usuario (1024-49151) y puertos dinámicos y / o privados (49152-65535) [15], se sustituyen los nulos por el valor 65536, que no hace referencia a ninguno de los puertos mencionados.

El dataset resultante es el de la Figura 5.3 (de nuevo, solo se muestran las cinco primeras filas).

	frame.number	frame.len	eth.src	eth.dst	ip.src	ip.dst	ip.proto	ip.len	tcp.len	tcp.srcport	tcp.dstport
0	1	54	87971959760497	167275820076079	3232235555	3232235641	6	40.0	0.0	49279.0	80.0
1	2	62	87971959760497	167275820076079	3232235555	3232235641	6	48.0	0.0	56521.0	80.0
2	3	62	167275820076079	87971959760497	3232235641	3232235555	6	48.0	0.0	80.0	56521.0
3	4	54	167275820076079	87971959760497	3232235641	3232235555	6	40.0	0.0	80.0	49279.0
4	5	54	87971959760497	167275820076079	3232235555	3232235641	6	40.0	0.0	56521.0	80.0

Figura 5.3: Dataset resultante de tráfico de dispositivos IoT.

3.3 Aplicación del algoritmo

Para aplicar el algoritmo de Isolation Forest, en primer lugar, se ha seleccionado en una variable, todas las columnas del dataset con las que se crea el modelo (Figura 6.1).

```python
data = dataset[['frame.number', 'frame.len', 'eth.src', 'eth.dst', 'ip.src', 'ip.dst', 'ip.proto', 'tcp.len', 'tcp.srcport', 'tcp.dstport']]
```

Figura 6.1: Variables a tener en cuenta para la construcción del modelo.

A continuación, se ha entrenado el modelo (Figura 6.2).

```
isoForest = IsolationForest()
isoForest.fit(data)
```

Figura 6.2: Entrenamiento del modelo.

Finalmente, con el método *predict* (Figura 6.3), se ha obtenido la identificación de anomalía, que se representa mediante “-1”, o no anomalía, mediante “1” (Figura 6.4).

```
anomaly = isoForest.predict(data)
```

Figura 6.3: Aplicación del método predict.

	frame.number	frame.len	eth.src	eth.dst	ip.src	ip.dst	ip.proto	ip.len	tcp.len	tcp.srcport	tcp.dstport	anomaly
0	1	54	87971959760497	167275820076079	3232235555	3232235641	6	40.0	0.0	49279.0	80.0	1
1	2	62	87971959760497	167275820076079	3232235555	3232235641	6	48.0	0.0	56521.0	80.0	1
2	3	62	167275820076079	87971959760497	3232235641	3232235555	6	48.0	0.0	80.0	56521.0	1
3	4	54	167275820076079	87971959760497	3232235641	3232235555	6	40.0	0.0	80.0	49279.0	-1
4	5	54	87971959760497	167275820076079	3232235555	3232235641	6	40.0	0.0	56521.0	80.0	1
...	...	...	...	...	...	...	...	...	...	...	...	...
125153	125154	98	167275820076079	110425385261001	3232235641	3232235703	1	84.0	0	65536	65536	-1
125154	125155	98	110425385261001	167275820076079	3232235703	3232235641	1	84.0	0	65536	65536	-1
125155	125156	98	167275820076079	110425385261001	3232235641	3232235703	1	84.0	0	65536	65536	-1
125156	125157	98	110425385261001	167275820076079	3232235703	3232235641	1	84.0	0	65536	65536	-1
125157	125158	98	167275820076079	110425385261001	3232235641	3232235703	1	84.0	0	65536	65536	-1

Figura 6.4: Dataset con la columna que identifica si el tráfico es anómalo (-1) o no (1).

Capítulo 4 Detección de spam

4.1 Introducción

En este capítulo se ha trabajado con un dataset obtenido de la plataforma Kaggle [16], y luego, se ha creado un dataset a partir de un conjunto de emails clasificados como spam o ham [17]. El objetivo es aprender a trabajar tanto con datos ya estructurados y normalizados, como crear un dataset con los datos que se necesitan.

4.2 Datos clasificados previamente

El dataset está formado por dos columnas: el contenido del email y la etiqueta que identifica si es spam o no. El contenido del email viene modificado de manera que las URLs han sido sustituidas por la palabra "URL" y los números por la palabra "NUMBER".

Como los algoritmos de machine learning trabajan con datos numéricos, es necesario transformar el contenido del email a formato numérico de alguna forma. En primer lugar, lo que se ha hecho es procesar el lenguaje natural. En la Figura 7.1, los pasos que se siguen son los siguientes:

```
corpus = []
for row in range(0, len(dataset)):
  text = re.sub('[^a-zA-Z]', ' ',str(dataset['email'][row]))
  text = text.lower()
  text = text.split()
  text = [ps.stem(word) for word in text if not word in stopwords.words('english')]
  text = ' '.join(text)
  corpus.append(text)
```

Figura 7.1: Transformación del contenido del email a formato numérico.

1. Se crea un array *corpus*, que contiene una cadena de caracteres con todas las palabras que forman cada uno de los emails. Estas palabras se modifican siguiendo los pasos que se mencionan a continuación.
2. Se reemplazan con un espacio todos aquello que no sean caracteres.
3. Todo se convierte en minúscula.
4. El texto se transforma en una lista de palabras.
5. Se obtienen sólo las palabras raíces de todas las palabras, excepto de aquellas incluidas en *stopwords.* Las *stopwords* (palabras vacías) son palabras sin significado como artículos, pronombres, preposiciones, etc. que son filtradas en el procesamiento de datos de lenguaje natural.
6. Se unen todas las palabras separadas por un espacio.
7. Se añaden al vector corpus.

A continuación, se identifican las variables predictora y objetivo (Figura 7.2).

```
predictor = corpus
target = dataset.label
```

Figura 7.2: Variable objetivo y variable predictora.

Se divide entre datos para el entrenamiento y datos para testear (Figura 7.3). El tamaño de la muestra para hacer la prueba es del 20%.

```
pred_train, pred_test, tar_train, tar_test = train_test_split(predictor, target, test_size = 0.20)
```

Figura 7.3: Datos de training y de test.

Se construye el modelo con el algoritmo Naive Bayes (Figura7.4). Se hace uso de un *Pipeline*. Un *Pipeline* es, básicamente, una lista secuencial de transformadas y un estimador final. Las transformadas son manipulaciones directas sobre los datos; y el estimador es el algoritmo que se quiere utilizar. La ventaja de utilizar pipelines es que permite tener el código agrupado, compacto y a la vista, haciéndolo más legible, facilitando su comprensión.

La primera transformada que se utiliza es el método CountVectorizer, que sirve para transformar un texto en un vector con la frecuencia de cada palabra que aparece en dicho texto. La segunda transformada es el método TFidtTransformer, que mide con qué frecuencia aparece un término dentro de un fichero determinado, y lo compara con el número de ficheros que mencionan ese término dentro de una colección entera de ficheros. Por otro lado, el algoritmo que se utiliza es el Isolation Forest.

Con estas dos transformadas es con las que el contenido de texto pasa a formato numérico para que el algoritmo sea capaz de trabajar con los datos.

```
naive_bayes = Pipeline([('cv', CountVectorizer()),
                        ('tfidf', TfidfTransformer()),
                        ('nb', MultinomialNB())])
```

Figura 7.4: Modelo con el algoritmo Naive Bayes.

Para aplicar este modelo, se llama al método *fit* para entrenar el modelo y luego al método *predict* para realizar las predicciones (Figura 7.5).

```
naive_bayes.fit(pred_train, tar_train)
predictions = naive_bayes.predict(pred_test)
```

Figura 7.5: Entrenamiento del modelo y realización de las predicciones.

La matriz obtenida es la de la Figura 7.6. El correo legítimo (ham) se clasifica correctamente, sin embargo, el spam lo clasifica la mayoría de las veces como ham. A pesar de esta clasificación errónea, el índice de precisión es de casi el 90%.

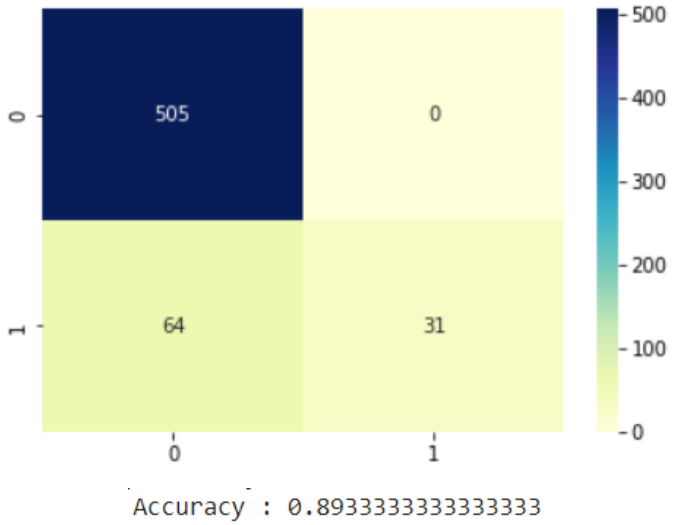


Figura 7.6: Detección de spam. Matriz de confusión con algoritmo de Naive Bayes y el índice de precisión. El “0” indica “ham” y el “1”, spam.

Si al modelo que se ve en la Figura 7.4 le suprimimos la segunda transformada (Figura 7.7), la matriz de confusión resultante es la de la Figura 7.8. Esta vez la clasificación de ham tiene 3 clasificaciones incorrectas, y la clasificación de spam mejora con respecto a la anterior, obteniendo solo 5 incorrectas. Por consiguiente, el valor de la precisión mejora.

```
naive_bayes = Pipeline([('cv', CountVectorizer()),
                        ('nb', MultinomialNB())])
```

Figura 7.7: Modelo con el algoritmo Naive Bayes, sin segunda transformada.

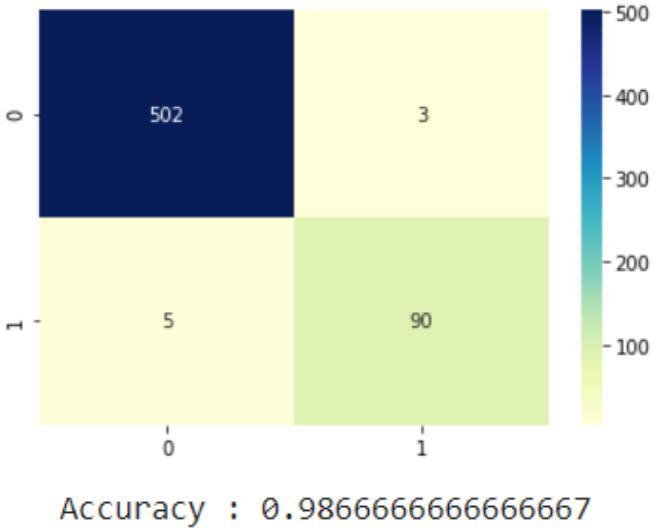


Figura 7.8: Detección de spam. Segunda matriz de confusión con Naive Bayes.

En este caso, se ve cómo aplicar el método *TFidtTransformer*, que mide la frecuencia con la que aparece un término, no resulta beneficioso.

A continuación, se construyen los dos modelos (tanto con el método *TFidtTransformer*, como sin él) aplicando el algoritmo de Random Forest. Los resultados se ven en las Figuras 7.9 y 7.10. Se observa cómo aplicar este método no trae malas consecuencias como en el caso anterior. El valor de la precisión es mínimamente superior.

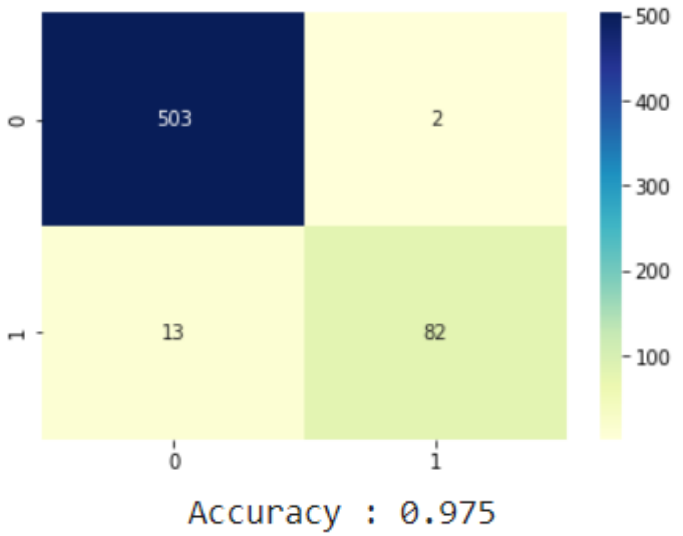


Figura 7.9: Detección de spam. Matriz de confusión, con Random Forest y aplicando la transformada *TFidtTransformer*.

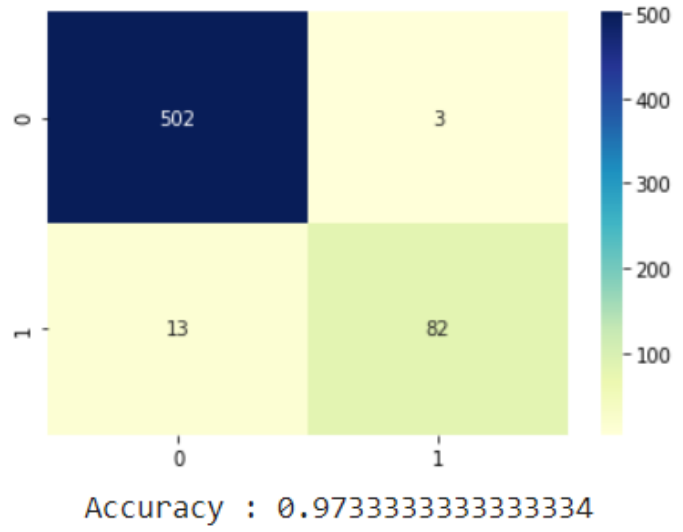


Figura 7.10: Detección de spam. Segunda matriz de confusión, con Random Forest y sin aplicar la transformada *TFidtTransformer*.

Por último, se aplica el algoritmo de regresión logística, siguiendo los mismos pasos. Los resultados se aprecian en las Figuras 7.11 y 7.12. Ocurre lo mismo que en el caso ya visto de aplicar Naive Bayes: se obtienen resultados peores utilizando el método *TFidtTransformer*.

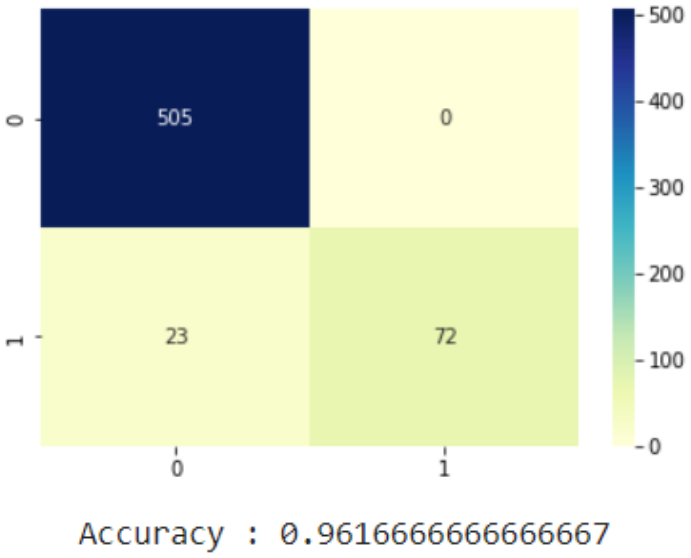


Figura 7.11: Detección de spam. Matriz de confusión con regresión logística, aplicando el método *TFidtTransformer.*

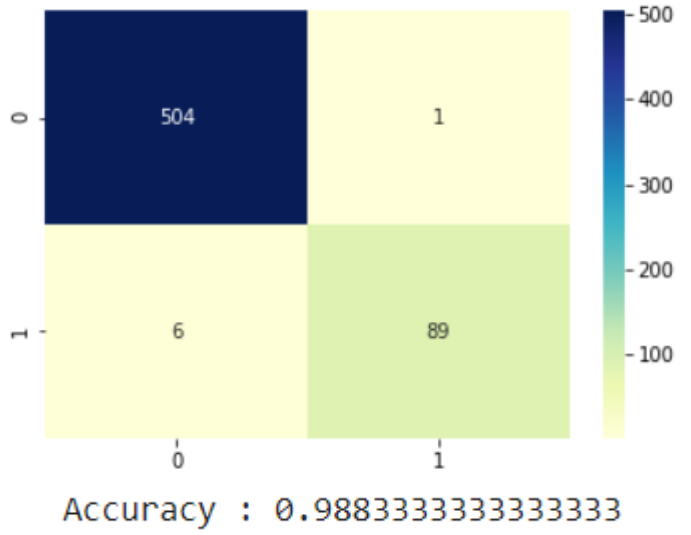


Figura 7.12: Detección de spam. Segunda matriz de confusión con regresión logística, no aplicando el método *TFidtTransformer.*

4.3 Construcción de nuestro propio dataset

Para la creación del dataset que se ha utilizado en este caso contamos con un repositorio de emails clasificados como spam o ham (Figura 8.1).

- /data
  - /ham
    - fichero_ham1
    - fichero_ham2
    - ...
  - /spam
    - fichero_spam1
    - fichero_spam2
    - ...

Figura 8.1: Esquema del directorio.

El primer paso consiste en abrir y leer cada uno de los ficheros. Una vez leídos, hay que convertir a texto plano, el contenido de los emails. Para ello, se hace uso de las funciones obtenidas de la plataforma kaggle [18]. Además, se crea otra función, basada en las anteriores, para conseguir, como texto plano, un fichero con contenido html (Figura 8.2).

```
def email_to_plain(email):
    struct = get_email_structure(email)
    for part in email.walk():
        partContentType = part.get_content_type()
        if partContentType not in ['text/plain','text/html']:
            continue
        try:
            partContent = part.get_content()
        except:
            partContent = str(part.get_payload())
        return partContent
```

Figura 8.2: Función para obtener el contenido html del email en texto plano.

A continuación, se procede a la construcción del dataset. Se definen cuáles van a ser las características (preguntas) del conjunto de datos. En este caso se ha optado por definir las siguientes:

1. Longitud del texto del asunto. Algunos ficheros spam suelen ni siquiera tener texto en el asunto.
2. Número de destinatarios. Un correo que se envía a varias personas es más probable que sea spam.
3. La hora, en formato 24 horas, del envío del email. Se selecciona solo la hora, no los minutos. El motivo es que en horas de la madrugada (desde las 00:00 hasta las 06:00), si se recibe un correo electrónico, es posible que se trate de spam. No es habitual mandar un correo a esas horas.
4. Longitud del contenido del email.
5. Número de veces que aparecen los símbolos del dólar ($), euro (€) y porcentaje (%), ya que normalmente puede tratarse de una oferta engañosa y/o fraudulenta.
6. También la frecuencia con la que aparecen enlaces url.
7. Número de veces que aparecen los símbolos de exclamación (!).

8. Número de veces que se encuentran las 10 palabras más comunes. Para conseguir esto, primero hay que descartar las palabras contenidas en las *stopwords*.
9. Y, por último, indicar con un “1” si se encuentra contenido html. Si no, añadir un “0”.

Por otro lado, la etiqueta de cada dato es indicar si es spam o no lo es.

La función definida para construir este dataset es la que se aprecia en las Figuras 8.3 y 8.4. La llamada a esta función se encuentra en la Figura 8.5. La función tiene como parámetros el email en cuestión y un “1” si es spam o un “0” si no lo es. La variable *dataset*, es la que va almacenando el dataset construido. El contenido de esta variable se puede escribir en un fichero .csv y en la Figura 8.6 se muestra el contenido de ese fichero, el dataset creado.

```
def read_data(email, target):
  data = []

  # 1. Longitud del asunto:
  data.append(len(email['Subject']))

  # 2. Número de destinatarios
  if not email['To'] :
    data.append(0)
  else :
    data.append(len(email['To'].split(',')))

  # 3. Hora (formato 24h)
  data.append(int(email['Date'].split(' ')[4].split(':')[0]))

  email = email_to_plain(email)

  # 4. Longitud del texto
  data.append(len(email))

  # 5. Número de símbolos del $
  dollars = re.findall("\$", email)
  if not dollars :
    data.append(0)
  else :
    data.append(len(dollars))

  # 6. Número de símbolos del €
  euros = re.findall("\€", email)
  if not euros :
    data.append(0)
  else :
    data.append(len(euros))
```

Figura 8.3: Construcción del dataset.

```
# 7. Número de símbolos del %
percent = re.findall("\%", email)
if not percent :
  data.append(0)
else :
  data.append(len(percent))

# 8. Número de URLs
data.append(len(re.findall('(http?://\S+)', email)))

# 9. Número de símbolos de exclamación
exclamation = re.findall("!", email)
if not exclamation :
  data.append(0)
else :
  data.append(len(exclamation))

# Las 10 palabras más comunes.
email = re.sub('[^a-zA-Z].<.>', ' ', email)
text = email.split()
word_counts = Counter(text)
for i in range(10):
  data.append(word_counts[common_words[i]])

# Si tiene contenido html
if word_counts['<!DOCTYPE'] or word_counts['<html>']
  data.append(1)
else:
  data.append(0)

# etiqueta
data.append(target)

return data
```

Figura 8.4: Construcción del dataset.

```
for spam in spam_emails:
  data = read_data(spam, 1)
  dataset.append(data)

for ham in ham_emails:
  data = read_data(ham, 0)
  dataset.append(data)
```

Figura 8.5: Llamada a la función para construir el dataset.

	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	F15	F16	F17	F18	F19	F20	F21
0	30	1	20	3668	0	0	8	1	9	0	1	2	0	0	0	0	0	1	0	1	1
1	53	1	6	781	0	0	2	7	2	0	0	0	0	0	0	1	0	0	0	0	1
2	71	1	7	625	0	0	2	6	2	0	0	0	0	0	0	1	0	0	0	0	1
3	76	1	8	2904	0	0	0	6	16	0	1	2	0	3	0	0	0	2	0	0	1
4	64	1	9	659	0	0	2	5	1	0	0	0	0	0	0	1	0	0	0	0	1
5	34	1	10	2411	10	0	0	0	26	0	3	1	1	0	0	1	0	1	0	0	1
6	41	1	13	692	0	0	0	2	0	0	0	0	0	1	0	0	0	0	0	0	1
7	25	1	10	29663	0	0	41	13	0	0	1	0	0	0	0	0	0	0	0	1	1

Figura 8.6: Contenido del dataset.

A continuación, se siguen los mismos pasos que ya se han comentado anteriormente: indicar cuáles son las variables predictoras y cuál la objetivo y dividir en datos de entrenamiento y en datos de prueba.

El primer modelo que se ha construido es aplicando el algoritmo de Random Forest. Los resultados se observan en la matriz de confusión de la Figura 8.7.

Esta vez, se ha extraído la importancia que ha tenido cada una de las características en la determinación de si el email es spam o no lo es (Figura 8.8). Se observa que la característica número seis (frecuencia con la que aparece el símbolo del euro), no es absolutamente nada relevante. Los emails están en inglés, de origen estadounidense, por lo que se trabaja con el dólar.

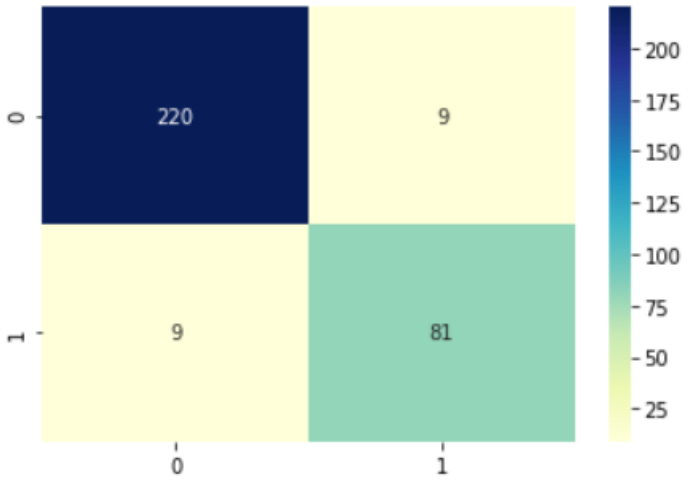


Figura 8.7: Detección de spam. Matriz de confusión con algoritmo Random Forest.

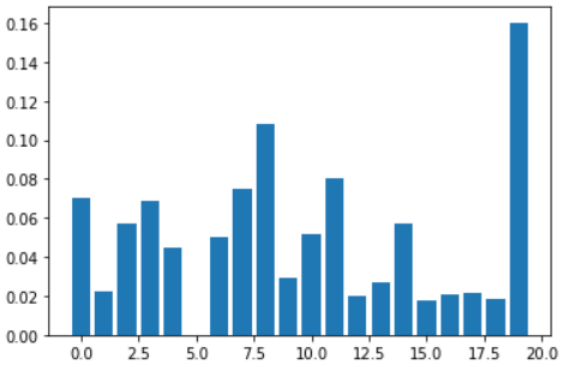


Figura 8.8: Relevancia de cada una de las características.

Si se seleccionan como variables predictoras las principales características más importantes (Figura 8.9), el resultado de la matriz de confusión es el de la Figura 8.10. Se pueden comparar ambos resultados, es decir, las dos matrices de confusión que se han obtenido, mediante el índice de precisión (Figura 8.11). Se observa que, aunque ambos sean buenos resultados porque superan el 90%, al dejar de tener en cuenta 6 características, este valor disminuye menos de un 0.04%.

```
predictors = df[['F1', 'F3', 'F4', 'F5', 'F7', 'F8', 'F9', 'F11', 'F12', 'F15', 'F20']]
```

Figura 8.9: Variables predictoras más relevantes.

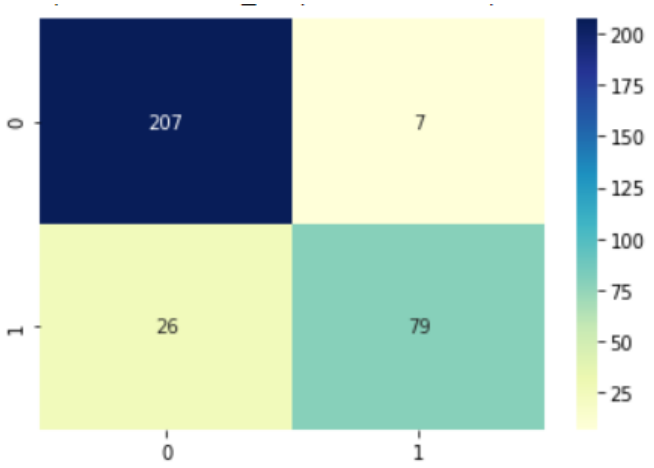


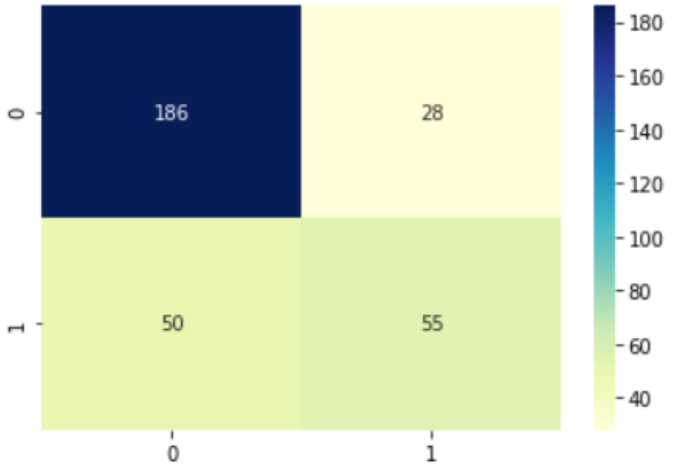
Figura 8.10: Detección de spam. Matriz de confusión con algoritmo Random Forest.

```
Random Forest (1). Accuracy Score: 0.9435736677115988

Random Forest (2). Accuracy Score:  0.896551724137931
```

Figura 8.11: Índices *Accuracy Score* (precisión) de ambas matrices.

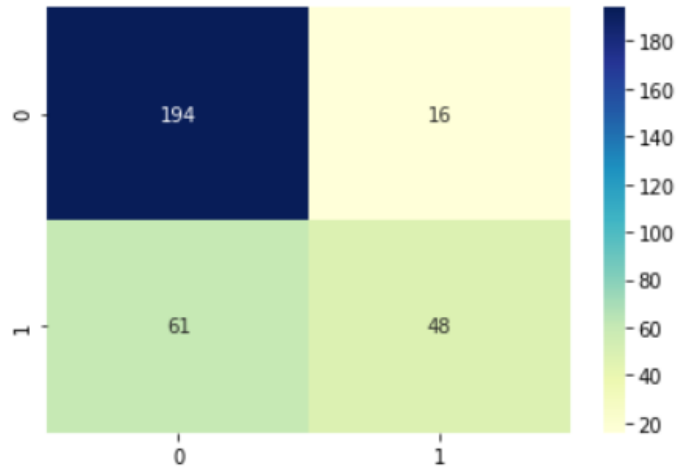
A continuación, se siguen los mismos pasos, esta vez aplicando el algoritmo de Naive Bayes y el de regresión logística.

Aplicando Naive Bayes, la primera matriz de confusión que se obtiene es la de la Figura 8.12, con el índice de precisión que se indica. Si a continuación se tienen en cuenta las mismas características de la Figura 8.9, el resultado es el de la Figura 8.13.



Naive Bayes (1). Accuracy Score: 0.7554858934169278

Figura 8.12: Matriz de confusión aplicando el algoritmo de Naive Bayes.



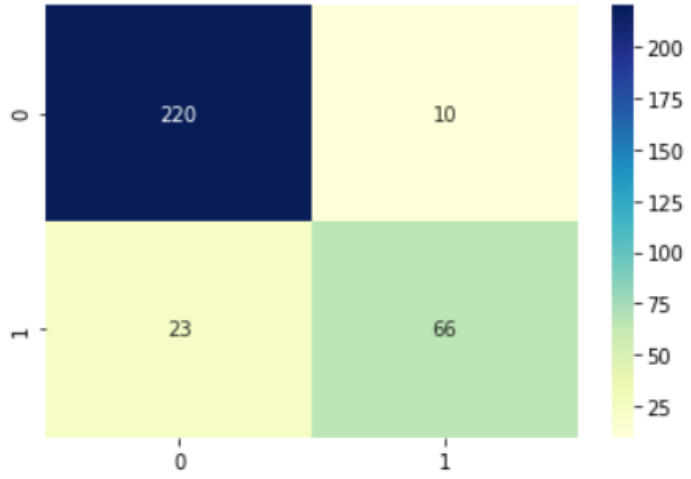
Naive Bayes (2). Accuracy Score: 0.7586206896551724

Figura 8.13: Matriz de confusión aplicando el algoritmo de Naive Bayes, teniendo en cuenta las características más relevantes.

Los valores de precisión obtenidos indican que la segunda matriz ofrece mejores resultados (un 0.003%). Esto se debe a que es capaz de detectar mejor el correo ham.

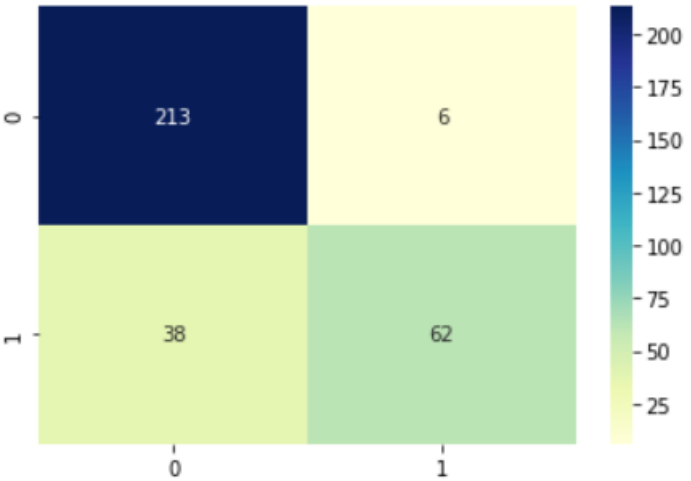
Por último, aplicando el algoritmo de regresión logística, la matriz de confusión que se obtiene, teniendo en cuenta todas las características es la mostrada en la Figura 8.14, junto al valor de precisión. Luego, la matriz obtenida teniendo en cuenta, de nuevo, las

características de la Figura 8.9, y su índice de precisión, se observa en la Figura 8.15.



Regresión logística (1). Accuracy Score: 0.896551724137931

Figura 8.14: Matriz de confusión aplicando el algoritmo de regresión logística y su índice de precisión.



Regresión logística (2). Accuracy Score: 0.8620689655172413

Figura 8.15: Matriz de confusión aplicando regresión logística según características relevantes.

Se aprecia que la clasificación del spam no consigue ser mejor, e incluso el índice de precisión disminuye un 0.03%. En la Figura 8.14, a pesar de la mala clasificación del spam, la clasificación del ham sí es satisfactoria, de ahí a que el índice mencionado sea casi del 90%.

Capítulo 5 Detección de malware

5.1 Introducción

Para este caso de estudio, que consiste en la detección de malware, se hace uso de un repositorio que contiene diferentes ficheros malware clasificados en directorios [19, 20]. Estos ficheros son del tipo APT1, Crypto, Locker y Zeus, como se mencionó en el apartado 2.1.3.

En este capítulo se quiere conseguir detectar cada tipo de malware entre el conjunto de ficheros. El problema reside en saber seleccionar las propiedades adecuadas de cada malware para conseguir distinguirlo de los demás.

Cada fichero posee una estructura similar a la que se muestra en la Figura 9.1.Todos tienen las mismas *keys* de *entityId, entityType, event, eventTime* y *properties*. Sin embargo, las *keys* que aparecen en *properties*, no son siempre las mismas para todos los ficheros, ni siquiera para los que corresponden al mismo tipo de malware.


```
{
  "entityId": 1471,
  "entityType": "content",
  "event": "malware",
  "eventTime": "2016-12-15T09:02:15.971+0000",
  "properties": {
    "pe_sec_entropy": "121e23c4 2b5970e7 b14399cb 54721433 eaa95980",
    "pe_sec_name": "916e7571 e8997399 b7b66a05 1f9f1073 94fe2c55",
    "reg_access": "51697019 54b53072 fc5364bf c14535e8",
    "net_dns": "4eae35f1 fd456406 8b177187 4d236d9a",
    "sig_fraudguard_threat_intel_api": "",
    "sig_static_pe_anomaly": "a9f2c818",
    "label": "Crypto",
    "pe_sec_character":
    "9271b28c 2f0acaf7 ad8225f3 2f0acaf7 f64ccb6c",
    "net_con": "ad921d60 d511915c 0839019c",
    "str": "916e7571 ec73657d e4787bb1 1f9f1073 7211bc06 a34fea09 0e2a1
    "sig_network_http": "8c47705f",
    "pe_imports": "cec36c4f 3ccc11b0 0459daaa 48a43d94 22d48376 1d96227
    "sig_injection_rwx": "",
    "sig_antivirus_virustotal": "b8c9478b a4acb20e ab825f73 b8c9478b f7
    "net_http": "9bd330a9",
    "api_resolv": "090a162f 8c70d8b3 70d48670 3d6c3869 38c96c1d 4b03b39
    "sig_stealth_network": ""
    }
}
```


Figura 9.1: Estructura fichero malware (formato JSON).

5.2 Implementación

5.2.1 Primera versión

En el primer intento, se ha construido un dataset con dos columnas: la característica y la etiqueta. La primera está formada por el contenido de todas las propiedades, menos la propiedad *label*, que es la etiqueta del dato, la segunda columna (Figura 9.2). Esta información la guardamos en variables (Figura 9.3). La columna *properties* es la variable

predictora, y la columna *label*, la variable objetivo.

	properties	label
0	d5197d93 c3581516 45b96339 45b96339 0976bdd4 ...	0
1	d5197d93 c3581516 45b96339 45b96339 0976bdd4 ...	0
2	4a4212d1 05bc38e9 b7b66a05 af5cc73b 4bd26db0 ...	0
3	3d801aa5 385273e4 98e83379 2e5d8aa3 385273e4 ...	0
4	3d801aa5 350fd223 98e83379 bda9f42e 249a6812 ...	0

Figura 9.2: Primer dataset para la detección de malware.

```
predictor = df.properties
target = df.label
```

Figura 9.3: Variable predictora y objetivo.

A continuación, se dividen los datos en datos de entrenamiento y en datos de prueba (Figura 9.4). Los datos de prueba corresponden al 20% de los datos totales.

```
pred_train, pred_test, tar_train, tar_test = train_test_split(predictor, target, test_size=.2)
```

Figura 9.4: División de los datos de entrenamiento y prueba.

El siguiente paso es crear el modelo (Figura 9.5). Se utiliza, de nuevo, un *Pipeline*, como se explicó en el capítulo anterior.

```
naivebayes = Pipeline([('cv', CountVectorizer()),
                       ('tfidf', TfidfTransformer()),
                       ('nb', MultinomialNB()),
])

naivebayes.fit(pred_train, tar_train)

predictions = naivebayes.predict(pred_test)
```

Figura 9.5: Construcción del modelo, entrenamiento y predicción.

Los resultados se muestran en la matriz de confusión (Figura 9.6). El 0, 1, 2 y 3 corresponden a APT1, Crypto, Locker y Zeus, respectivamente. Se observa que únicamente tiene gran cantidad de acierto al identificar el malware Zeus. Por contrapartida, cuando el resultado esperado es Crypto, en ningún caso consigue detectarlo correctamente.

Se prueba con otro de los algoritmos, el Random Forest, siguiendo el mismo procedimiento. La matriz de confusión resultante es la de la Figura 9.7. En este caso, el modelo es capaz de diferenciar y clasificar correctamente la mayoría de los ficheros APT1 y Locker, sin embargo, los ficheros Crypto y Zeus, parece confundirlos.

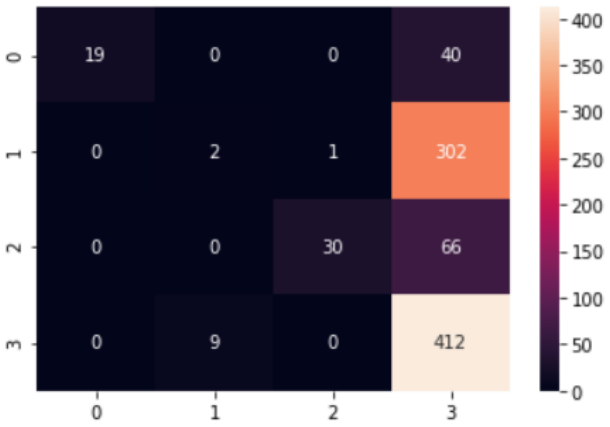


Figura 9.6: Matriz de confusión (algoritmo Naive Bayes).

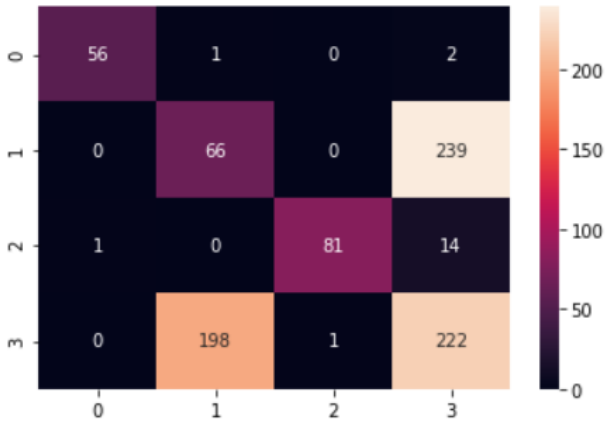


Figura 9.7: Matriz de confusión (algoritmo Random Forest).

Se puede ver qué algoritmo ha obtenido mejores resultados, extrayendo el índice de precisión, que resume la Matriz de Confusión y la cantidad de aciertos (Figura 9.8). Para que un modelo se considere válido, este valor tiene que ser cercano al 90% o superior, por lo que, los dos modelos que se han construido en este primer intento, no son válidos.

```
Accuracy score:  0.5255391600454029

Accuracy score:  0.48240635641316687
```

Figura 9.8: Accuracy Score de los dos algoritmos (Naive Bayes y Random Forest).

5.2.2 Segunda versión

En este segundo intento y en los posteriores, los modelos que se han construido siguen las mismas normas vistas en el intento anterior, lo que cambia es la construcción del dataset, ya que se han tenido en cuenta diferentes propiedades de los ficheros.

Se obtienen todas las propiedades que son comunes en todos los ficheros de un mismo tipo de malware (Figura 10.1). Se observa que la propiedad común a los 4 malware es “label”, que corresponde con el tipo de malware que es, dicho de otro modo,

es la variable objetivo (la etiqueta). Sin contar el malware locker y crypto, las propiedades comunes son 'pe_sec_character', 'pe_sec_entropy', 'pe_sec_name', y 'str'. Se procede a trabajar con estas propiedades, y añadir un 0 cuando se trate de los ficheros locker o crypto. Se añaden los valores de estas propiedades a un string y este string corresponde a la columna “properties” del dataset (Figura 10.2).

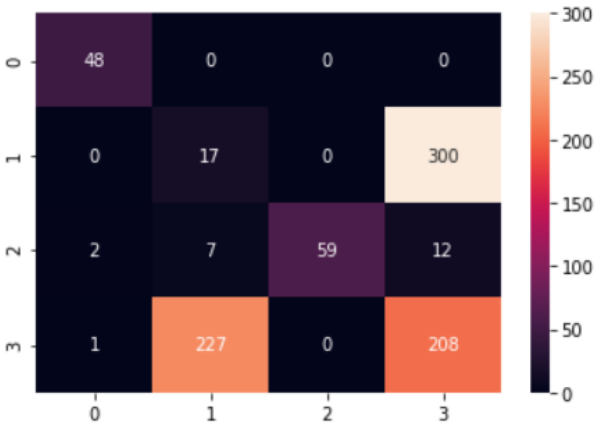
```
Number of APT1 files properties:  5
APT1 files properties:  ['label', 'pe_sec_character', 'pe_sec_entropy', 'pe_sec_name', 'str']

Number of Crypto files properties:  1
Crypto files properties:  ['label']

Number of Locker files properties:  2
Locker files properties:  ['label', 'str']

Number of Zeus files properties:  5
Zeus files properties:  ['pe_sec_entropy', 'label', 'pe_sec_character', 'str', 'pe_sec_name']

Number of common properties:  1
Common properties:  ['label']
```

Figura 10.1: Lista de las propiedades comunes en los ficheros de un tipo de malware.

	properties	label
0	09271b28c 2f0acaf7 ad8225f3cd69931f 5affe341 0...	0
1	09271b28c 2f0acaf7 ad8225f3 2f0acaf7cd69931f a...	0
2	0ad8225f3 ad8225f34a4212d1 05bc38e9b7b66a05 af...	0
3	09271b28c 2f0acaf7 ad8225f3 2f0acaf75f3e0d9b 8...	0
4	09271b28c 2f0acaf7 ad8225f383f6790f 0d480777 a...	0

Figura 10.2: Dataset construido.

De nuevo, se crea un Pipeline, se dividen los datos (entrenamiento y prueba), y los resultados obtenidos son los de la Figura 10.3 (con Random Forest), y la Figura 10.4 (con Naive Bayes). Como en el caso anterior, los resultados dejan bastante que desear.




```
Accuracy score:  0.376844494892168
```

Figura 10.3: Matriz de confusión aplicando Random Forest y el índice de precisión.

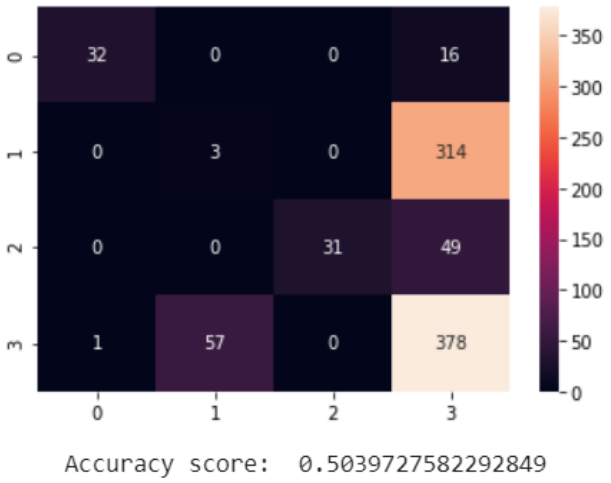


Figura 10.4: Matriz de confusión aplicando Naive Bayes y el índice de precisión.

5.2.3 Tercera versión

En este tercer intento se ha creado el dataset con las cinco propiedades que aparecen con mayor frecuencia en los ficheros de un mismo tipo de malware, descartando las que se han tenido en cuenta en la segunda implementación. Además, en lugar de almacenar el contenido de cada una de las propiedades seleccionadas, se ha almacenado su longitud.

Las propiedades más comunes resultan ser ocho (Figura 11.1). El dataset construido se muestra en la Figura 11.2.

```
Number of common properties:  8
Common properties:  ['pe_imports', 'reg_access', 'reg_read', 'file_access',
 'sig_antivirus_virustotal', 'api_resolv', 'file_read', 'sig_injection_rwx']
```

Figura 11.1: Propiedades más frecuentes.

	pe_imports	reg_access	reg_read	file_access	sig_antivirus_virustotal	api_resolv	file_read	sig_injection_rwx	label
0	917	4913	2699	107	440	332	80	0	0
1	926	4931	2717	107	449	350	80	0	0
2	620	0	0	0	422	0	0	0	0
3	710	359	161	107	0	125	53	0	0
4	737	359	125	215	422	152	80	0	0

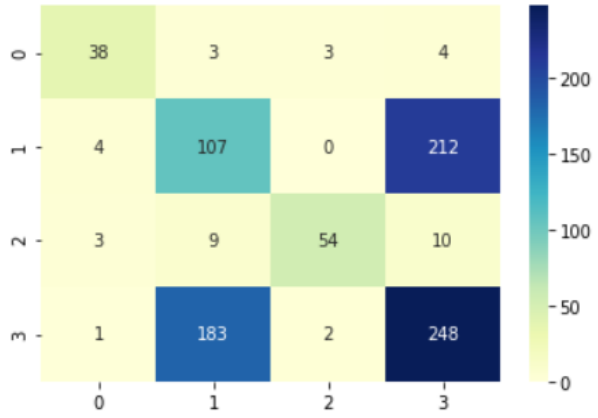
Figura 11.2: Dataset construido con las propiedades seleccionadas.

En esta implementación no se ha creado un *pipeline*. En su lugar, se aplica directamente el algoritmo de aprendizaje supervisado (Figura 11.3). Lo que sí sigue siendo igual es la selección de las variables predictoras y objetivo, y la división de datos de prueba y de entrenamiento.

```
classifier = RandomForestClassifier()
classifier = classifier.fit(pred_train, tar_train)
```

Figura 11.3: Creación del modelo con el algoritmo de Random Forest.

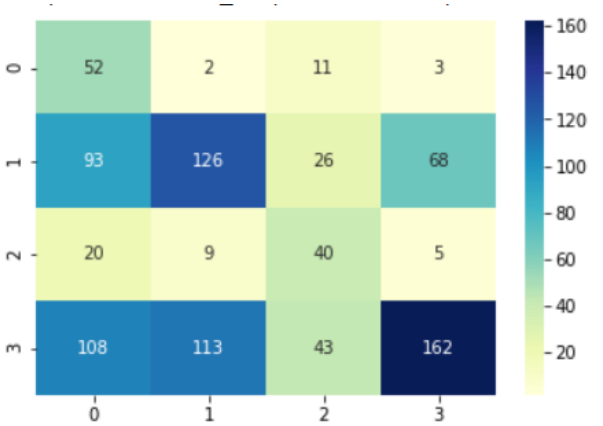
La matriz de confusión obtenida tras aplicar el modelo construido es la de la Figura 11.4. En los resultados se aprecia la confusión entre el malware crypto y zeus. El valor de la precisión es de prácticamente el 50%.




```
Accuracy score:  0.5073779795686719
```

Figura 11.4: Matriz de confusión (Random Forest) e índice de precisión.

A continuación, se aplica el algoritmo de Naive Bayes. La matriz de confusión es la de la Figura 11.5. Estos resultados son más desastrosos que los anteriores. En este caso, el malware APT1 y el Locker los logró clasificar con mayor exactitud, pero Crypto y Zeus ni siquiera los clasifica en su mayoría como un tipo de malware, si no como varios, es decir, no se equivoca en el mismo sentido.




```
Accuracy score:  0.43132803632236094
```

Figura 11.5: Matriz de confusión (Naive Bayes) e índice de precisión.

5.2.4 Cuarta versión

Esta cuarta y última versión se basa en la anterior, pero en lugar de almacenar en el dataset la longitud del contenido de las características, se ha almacenado el propio contenido (Se puede observar este dataset en la Figura 12.1). De nuevo, se dividen los datos en datos de entrenamiento y en datos de prueba, y se construye, esta vez sí, el Pipeline, como en la Figura 9.5.

	properties	label
0	09e49750e 9d47863c dd50ea4c f659f996 a2d78c4b ...	0
1	09e49750e 9d47863c dd50ea4c f659f996 a2d78c4b ...	0
2	0971d2c25 a405bc49 35b3913c 9492b793 e9b9792a ...	0
3	0d77083a7 57b32fd5 268d42b1 93288686 dc11a7db ...	0
4	0839ca61c 2b78df95 7f8fa70c 69fe6141 d77083a7 ...	0

Figura 12.1: Dataset.

Los resultados obtenidos se muestran en las Figuras 12.2 y 12.3; que corresponden a los resultados obtenidos construyendo el modelo con el algoritmo de random forest, y con naive bayes, respectivamente.

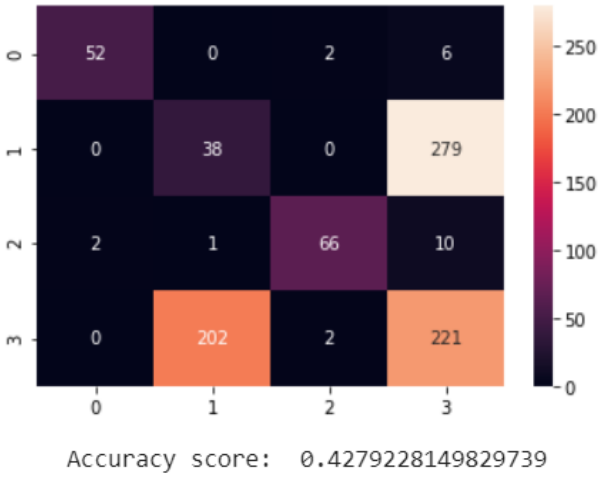


Figura 12.2: Matriz de confusión con el algoritmo Random Forest.

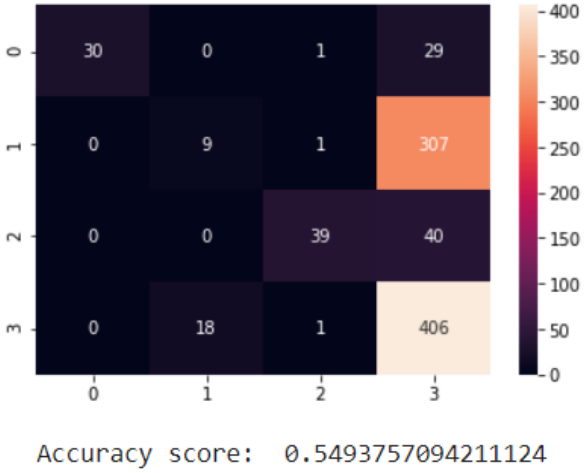


Figura 12.3: Matriz de confusión con el algoritmo Naive Bayes.

5.2.5 Análisis de los resultados

A pesar de los numerosos intentos, los resultados obtenidos en todas las implementaciones y aplicando diferentes algoritmos, no son buenos. Para que un modelo se considere válido, como se ha comentado en varias ocasiones, es necesario que el índice de precisión sea superior al 90%, y en este caso, ninguno lo es.

En líneas generales, los ficheros APT1 sí los logra identificar de entre los demás ficheros. Sin embargo, los otros tres tipos de malware, los confunde; en especial los crypto y zeus.

Capítulo 6 Conclusiones

El aprendizaje automático es una rama de la inteligencia artificial formada por numerosas herramientas poderosas de las que se pueden obtener resultados que una persona por sí sola no podría conseguir de forma rápida y eficiente. Por otra parte, la ciberseguridad, en un mundo cada vez más digitalizado y conectado, es fundamental. Ya no solo las empresas invierten en ella para protegerse ante todo tipo de ataques, sino que los delincuentes informáticos mejoran sus métodos de ataque. Por lo tanto, utilizar el aprendizaje automático para la ciberseguridad solo puede traer ventajas.

Poniendo el foco en el aprendizaje automático, es un campo valioso y de consideración puesto que se basa en el trabajo y manejo de grandes cantidades de datos. Estos datos implican una información, y saber tratar con ellos y hacer predicciones, clasificaciones, agrupaciones, etc, implica tener un conocimiento sobre el tema que se está tratando con los datos proporcionados. Poseer conocimiento significa tener valor.

En este trabajo se han creado datasets para la construcción de los modelos. Esto no es tarea fácil, pues es fundamental saber escoger qué características se tomarán en cuenta para luego fabricar el modelo. Esta es la labor que más tiempo lleva, y también la más importante, pues el modelo en sí, básicamente es la aplicación de un algoritmo. Aunque es cierto que se ha visto que a partir de los mismos datos, los resultados varían dependiendo de qué algoritmo se haya utilizado.

En cuanto a los resultados obtenidos en los tres casos de estudio, dedicados a la detección de tráfico IoT anómalo, spam y malware, los mejores se han conseguido en la detección de spam, ya que el índice de precisión obtenido en la mayoría de los casos gira en torno al 90%. Con esta información se puede concluir que los modelos construidos son válidos, y que el trabajo es satisfactorio. Por contrapartida, en los resultados de la detección de malware rara vez se alcanza el 50% de precisión, es decir, que ninguno de los modelos, y ninguno de los conjuntos de datos construidos son válidos. En el caso práctico de la detección de tráfico anómalo en dispositivos IoT, sería necesario que un humano evaluase los resultados obtenidos.

El siguiente paso consistiría en mejorar el preprocesamiento de los datos del caso práctico de la detección de malware, para conseguir tener mejores resultados de los obtenidos. El principal inconveniente es que los diferentes modelos confunden los ficheros crypto y zeus. Una solución podría ser conseguir una característica frecuente en los ficheros crypto, que no esté en los ficheros zeus, o viceversa. Otra solución podría ser obtener o fabricar un nuevo conjunto de ficheros malware. Por otro lado, se podría aplicar la técnica de validación cruzada (en inglés, *cross validation*), que consiste en evaluar los resultados de un análisis estadístico y garantizar que son independientes de la partición entre datos de entrenamiento y prueba. En cuanto a los algoritmos utilizados, se empleó el Random Forest, que es un algoritmo de bagging. Como esto no produjo buenos resultados, podrían aplicarse entonces algoritmos de boosting.

Capítulo 7 Conclusions

Machine learning is a branch of artificial intelligence made up of numerous powerful tools from which results can be obtained that one person alone could not achieve quickly and efficiently. On the other hand, cybersecurity, in an increasingly digitized and connected world, is essential. Not only are companies investing in it to protect themselves against all kinds of attacks, but cybercriminals are improving their attack methods. Therefore, using machine learning for cybersecurity can only bring advantages.

Putting the focus on machine learning, it is a valuable and considered field since it is based on the work and handling of large amounts of data. These data imply information, and knowing how to deal with them and make predictions, classifications, groupings, etc., implies having knowledge about the subject that is being dealt with with the data provided. Having knowledge means having value.

In this work, datasets have been created for the construction of the models. This is not an easy task, since it is essential to know how to choose which characteristics will be taken into account and then manufacture the model. This is the task that takes the longest, and also the most important, since the model itself is basically the application of an algorithm. Although it is true that it has been seen that from the same data, the results vary depending on which algorithm has been used.

Regarding the results obtained in the three case studies, dedicated to the detection of anomalous IoT traffic, spam and malware, the best ones have been obtained in the detection of spam, since the precision index obtained in most cases revolves around 90%. With this information it can be concluded that the built models are valid, and that the work is satisfactory. On the other hand, the results of malware detection rarely reach 50% accuracy, that is, none of the models and none of the data sets constructed are valid. In the practical case of detecting anomalous traffic in IoT devices, it would be necessary for a human to evaluate the results obtained.

The next step would be to improve the preprocessing of the data from the malware detection case study, in order to obtain better results than those obtained. The main drawback is that the different models confuse the crypto and zeus files. A solution could be to get a common feature in crypto files, which is not in zeus files, or vice versa. Another solution could be to obtain or manufacture a new set of malware files. On the other hand, the cross validation technique could be applied, which consists of evaluating the results of a statistical analysis and ensuring that they are independent of the partition between training and test data. Regarding the algorithms used, the Random Forest was used, which is a bagging algorithm. Since this did not produce good results, boosting algorithms could then be applied.

Capítulo 8 Presupuesto

Este capítulo consta de una propuesta del presupuesto que estima el coste del trabajo según el tiempo empleado en el desarrollo de las diferentes tareas que lo conforman.

Tarea	Tiempo	Coste
Investigación de los conocimientos relacionados con el machine learning, los algoritmos, construcción de modelos, preprocesamiento de los datos.	60 horas	8€ / hora
Adquirir experiencia utilizando Python en aplicaciones de aprendizaje automático.	10 horas	10€ / hora
Búsqueda de conjuntos de datos adecuados.	5 horas	15€ / hora
Construcción de datasets.	10 horas	20€ / hora
Construcción de los modelos.	30 horas	15€ / hora
Análisis de los resultados y redacción de la memoria	80 horas	10€ / hora

Tabla 1.1: Presupuesto

Capítulo 9 Código e instalación

En este capítulo se explica el despliegue del código desarrollado [21].

El código está formado por notebooks de Google Colab como ya se había mencionado. El notebook del enlace 22 corresponde al código de detección de tráfico anómalo de dispositivos IoT del capítulo 3. Los notebooks 23 y 24 son los dos códigos de la detección de spam (capítulo 4). Y por último, el notebook 25 es el del malware (capítulo 5).

Para utilizar los notebooks hay que descargarlos y abrirlos en Google Colab. Cuando se abre esta aplicación, se muestra la ventana de la Figura 13.1.

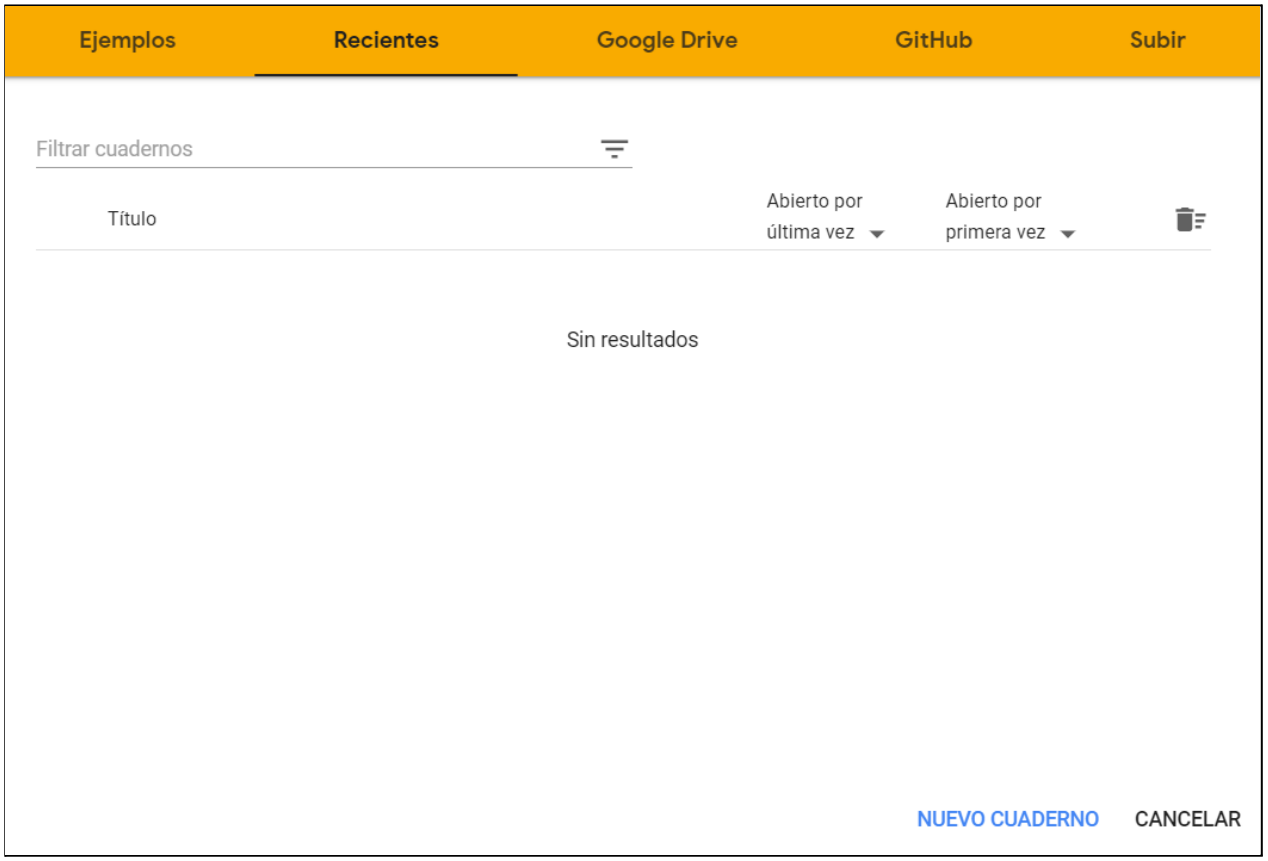


Figura 13.1: Ventana que se muestra al abrir Google Colab.

A continuación, se selecciona la pestaña “subir” y se sube o arrastra el notebook deseado. Una vez abierto el fichero, el siguiente paso consistiría en ejecutar las celdas en el orden que aparecen para el correcto funcionamiento. Si es la primera vez que se abre el fichero y no se le ha aplicado ningún cambio, mantiene los resultados obtenidos de

última ejecución.

Hay que tener en cuenta que las rutas que se le indican a las funciones para leer los ficheros son las rutas de acceso que se han utilizado para el despliegue y desarrollo del trabajo. Cada usuario diferente debe cambiarlas con la dirección en la que se encuentren los ficheros correspondientes. En el caso del tráfico IoT no es necesario. En los otros dos sí. Por ejemplo, en el caso de Detección de Spam con el dataset obtenido de Kaggle (Figura 13.2).

```
dataset = pd.read_csv("/content/drive/MyDrive/4to año/2do Cuatrimestre/TFG/Materiales/detección de spam/data/spam_or_not_spam.csv")
```

Figura 13.2: Ejemplo de ruta de acceso para leer el dataset en el caso de estudio de detección de Spam.

Bibliografía

[1] “Vulnerability Information,” *Email and Spam Data || Cisco Talos Intelligence Group - Comprehensive Threat Intelligence*. [Online]. Available: https://talosintelligence.com/reputation_center/email_rep. [Accessed: 12-May-2021].

[2] T. Yiu, “Understanding Random Forest”, *Towards Data Science*, 12-Jun-2019. [Online]. Available: https://towardsdatascience.com/understanding-random-forest-58381e0602d2

[3] V. Román, “Algoritmos Naive Bayes: Fundamentos e Implementación”, *Medium*, 25-Apr-2019. [Online]. Available: https://medium.com/datos-y-ciencia/algoritmos-naive-bayes-fudamentos-e-implementaci%C3%B3n-4bcb24b307f

[4] D. Rodríguez, “La regresión logística,” *Analytics Lane*, 01-Jul-2018. [Online]. Available: https://www.analyticslane.com/2018/07/23/la-regresion-logistica/. [Accessed: 28-May-2021].

[5] Carmen Torrano, “Jugando a policías y ladrones para detectar anomalías en red con ML”, *CyberCamp17,* 2017. Available: https://cybercamp.es/sites/default/files/contenidos/videos/adjuntos/cybercamp2017-jugando_a_policias_y_ladrones_para_detectar_anomalias_en_red_con_ml_carmen_torrano.pdf

[6] Joaquín Amat Rodrigo, “Detección de anomalías con Isolation Forest y python”, *Ciencia de datos*, Diciembre, 2020. Available: https://www.cienciadedatos.net/documentos/py22-deteccion-anomalias-isolation-forest-python.html

[7] Python. (2021). [Online]. Available: https://www.python.org/.

[8] Scikit Learn. (2021). [Online]. Available: https://scikit-learn.org/

[9] “Your Machine Learning and Data Science Community,” Kaggle. [Online]. Available: https://www.kaggle.com/. [Accessed: 12-May-2021].

[10] Google Colaboratory (2021). [Online]. Available: https://colab.research.google.com/

[11] Project Jupyter. [Online]. Available: https://jupyter.org/ . [Accessed: 21-May-2021].

[12] Sahil Dixit, 2020, "Iot Device Network Logs" Kaggle, doi: https://www.kaggle.com/speedwall10/iot-device-network-logs.

[13] Nicolas Limage, 2018, “mac address <=> integer conversions in python” Github Gist, doi: https://gist.github.com/nlm/9ec20c78c4881cf23ed132ae59570340

[14] Protocol Numbers. [Online]. Available: https://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml. [Accessed: 12-May-2021].

[15] J. Touch, J. Iyengar, B. Trammell, A. Zimmerman, W. Eddy, A. Melnikov, L. Eggert, M. Stiemerling, K. Ono, M. Kojo, A. Mankin, and E. Lear, “Service Name and Transport Protocol Port Number Registry,” Protocol Numbers, 06-May-2021. [Online]. Available: https://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml. [Accessed: 12-May-2021].

[16] Hakan Ozler, 2019, “Spam or Not Spam Dataset” kaggle, doi: https://www.kaggle.com/ozlerhakan/spam-or-not-spam-dataset

[17] Wessel van Lit , 2019, "Email Spam" Kaggle, doi: https://www.kaggle.com/veleon/ham-and-spam-dataset

[18] Wessel van Lit , 2019, "Email Spam. Turning-Emails-into-plaintext" Kaggle, doi: https://www.kaggle.com/veleon/spam-classification#Turning-Emails-into-plaintext

[19] M. Ramilli, “Malware Training Sets: A machine learning dataset for everyone,” Marco Ramilli Web Corner, 13-May-2019. [Online]. Available: https://marcoramilli.com/2016/12/16/malware-training-sets-a-machine-learning-dataset-for-everyone/. [Accessed: 16-May-2021].

[20] Marco Ramilli, 2019, “MalwareTrainingSets” GitHub, doi: https://github.com/marcoramilli/MalwareTrainingSets

[21] B. García Deus, 2021, “Notebooks de Google Colab con el código del TFG: Aprendizaje Automático para Detección de Tráfico IoT Anómalo, Spam y Malware” Github, doi: https://github.com/barbaragd/TFG

[22] B. García Deus, 2021, “Iot Traffic Notebook” GitHub, doi: https://github.com/barbaragd/TFG/blob/main/iot-traffic.ipynb

[23] B. García Deus, 2021, “Detección de spam con datos de Kaggle” GitHub, doi: https://github.com/barbaragd/TFG/blob/main/detecci%C3%B3n%20de%20spam%20con%20datos%20de%20kaggle.ipynb

[24] B. García Deus, 2021, “Detección de spam con dataset creado” GitHub, doi: https://github.com/barbaragd/TFG/blob/main/detecci%C3%B3n%20de%20spam%20con%20dataset%20creado.ipynb

[25] B. García Deus, 2021, “Malware Detection” GitHub, doi: https://github.com/barbaragd/TFG/blob/main/malware-detection.ipynb