Facultade de Informática
UNIVERSIDADE DA CORUÑA

TRABALLO FIN DE GRAO
GRAO EN ENXEÑARÍA INFORMÁTICA
MENCIÓN EN TECNOLOXÍAS DA INFORMACIÓN

Técnicas de aprendizaje máquina para análisis de malware

Estudante: Samuel Mouro González

Dirección: Elena María Hernández Pereira

A Coruña, septiembre de 2022.

A mi abuela materna, mi ángel de la guarda.

Agradecimientos

En primer lugar me gustaría agradecerle a Elena María Hernández Pereira la paciencia que ha tenido conmigo, así como la ayuda que me ha brindado siendo la tutora de este trabajo.

También me gustaría agradecerle a mi familia el esfuerzo que han realizado durante estos años para que pudiese estudiar esta carrera que tanto me apasiona.

A mis amigos por siempre estar cuando les necesito.

A Lu por apoyarme y animarme cuando lo necesitaba.

Y por último pero no menos importante, me gustaría agradecer tanto a Pablo como al resto de mis compañeros en Sotelo por hacerme disfrutar tanto de esta profesión.

A todos, gracias de corazón.

Resumen

La detección de malware se torna cada vez más importante en el campo de las Tecnologías de la Información. Una de las principales herramientas que sirven para esta tarea es el uso del aprendizaje automático para el desarrollo de modelos de detección. En este trabajo se desarrollan diversos modelos a partir de múltiples algoritmos de aprendizaje supervisado para tres escenarios diferentes. Los resultados obtenidos son muy alentadores, lográndose valores de rendimiento similares e incluso superiores a trabajos previos.

Abstract

Malware detection is becoming increasingly important in the field of Information Technology. One of the main tools for this task is the use of machine learning for the development of detection models. In this dissertation, several models are developed from multiple supervised learning algorithms for three different scenarios. The results obtained are very encouraging, achieving similar or even higher performance values than previous studies.

Palabras clave:

- Detección de malware
- Aprendizaje Automático
- Aprendizaje Supervisado
- Ficheros PE
- Inteligencia Artificial

Keywords:

- Malware detection
- Machine Learning
- Supervised Learning
- PE Files
- Artificial Intelligence

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

Los ataques de malware representan, en la actualidad, una de las mayores problemáticas a las que se enfrenta el campo de las tecnologías de la información. Es por esto que la protección de los sistemas informáticos es una de las tareas más importantes tanto para empresas como para cualquier usuario común de internet.

Existen multitud de tipos de malware, desde virus o gusanos que pueden replicarse por sí mismos, hasta ransomware que se hacen con el control de tu sistema hasta que satisfagas las demandas del atacante. Debido a que Windows sigue siendo el sistema operativo más utilizado en todo el mundo, muchos de estos ataques se realizan sobre los ficheros PE (*Portable Executable*), ya que es el formato de archivos ejecutables que usa principalmente este sistema operativo.

Para atajar esta problemática es fundamental, primero la implementación de herramientas para la identificación del software malicioso y después para su clasificación. Para ello se puede hacer uso de algoritmos de aprendizaje automático, una de las herramientas más eficientes y empleadas para dicha tarea. Este tipo de aprendizaje se configura como un conjunto de técnicas relacionadas con la inteligencia artificial que permiten a los sistemas resolver de manera autónoma gran cantidad de cuestiones, sin necesidad alguna de intervención humana.

En el desarrollo del trabajo no realizaremos la clasificación del malware entre sus distintos tipos, sino que, se analizará la utilización de algoritmos de aprendizaje supervisado para la tarea de identificación de este en ficheros PE.

1.1 Objetivos

La finalidad del trabajo es construir modelos de aprendizaje supervisado que contribuyan a reducir el impacto de los ataques de malware mediante la detección temprana de estos. Para ello, utilizaremos varias técnicas de aprendizaje automático, como son la Regresión Logística, las Máquinas de Vectores de Soporte, los K-Vecinos Más Cercanos (KNN), el algoritmo Naive

Bayes, los Árboles de Decisión, los Bosques Aleatorios y los Modelos de Redes Neuronales. El proceso de desarrollo consistirá en:

1. Estudiar de forma detallada las técnicas de análisis estático y dinámico de malware para determinar el conjunto de características que identifican a una muestra, teniendo en cuenta los recursos computacionales.
2. Determinar el conjunto de datos de trabajo mediante la selección del tipo de muestras a tratar.
3. Análisis y tratamiento del conjunto de datos seleccionado.
4. Definir los modelos de Aprendizaje Supervisado para realizar la clasificación.
5. Analizar los resultados de los modelos generados.
6. Comparar los resultados de los modelos con otros estudios existentes.

1.2 Organización de la memoria

La memoria de este trabajo se ha estructurado de la siguiente forma:

1. En el capítulo 1 se describirán las motivaciones del proyecto, sus objetivos y la estructura de la memoria.
2. En el capítulo 2 se hace una introducción sobre el malware, el análisis de este y la estructura de los ficheros PE.
3. En el capítulo 3 se describe el aprendizaje supervisado, así como los algoritmos utilizados en el proyecto.
4. En el capítulo 4 se hace una breve mención a estudios previos en el ámbito de la detección y clasificación de malware mediante aprendizaje automático.
5. En el capítulo 5 se mencionan y describen las tecnologías y herramientas usadas en el proyecto, así como las fuentes de los ficheros PE.
6. En el capítulo 6 se describe la metodología y planificación del proyecto.
7. En el capítulo 7 se muestra el desarrollo del problema.
8. En el capítulo 8 se describen y comparan los resultados de los distintos modelos.
9. En el capítulo 9 se muestran las conclusiones, los conocimientos adquiridos y el trabajo futuro.

Capítulo 2

Descripción del dominio

La palabra Malware está formada por la unión de dos términos, ”mal-” del latín ”malus”, que hace referencia a aquello que es malo o maligno, y ”-ware” que hace referencia al software. Por lo tanto, Malware es el término con el que nos referimos al software malicioso, aquel software diseñado con el objetivo de dañar o infiltrarse en un sistema informático sin el consentimiento de su propietario. Existen multitud de tipos de malware entre los que se encuentran [1]:

- **Virus:** El objetivo de un virus es infectar sistemas y ficheros al replicarse a sí mismo (como un virus biológico). Este tipo no puede existir de forma independiente, por lo que va ligado a otros archivos, normalmente archivos ejecutables o aplicaciones. Gracias a su capacidad para replicarse, el virus es capaz de propagarse a través de archivos de un ordenador e incluso a través de la red, pudiendo así causar problemas de rendimiento en el sistema e incluso denegación de servicio (DoS) [2].

- **Gusanos:** Los gusanos, al igual que los virus, tienen la capacidad de replicarse por sí mismos, pudiendo propagarse a través de dispositivos de almacenamiento o correos electrónicos, consumiendo recursos de red y del sistema. La principal diferencia con respecto a los virus, es que los gusanos si pueden existir de forma independiente, aumentando de esta forma su capacidad para propagarse.

- **Troyanos:** Un troyano es un software malicioso que se presenta al usuario como un programa aparentemente legítimo, pero cuya ejecución permite al atacante acceso remoto al equipo infectado. Los troyanos no pueden replicarse por sí mismos, pero se pueden transferir a nuestros equipos a través de la interacción con internet, por ejemplo, al descargar un programa que creemos benigno. Los troyanos pueden robar información, observar la actividad de nuestro ordenador, o incluso alterar ficheros de nuestro sistema.

- **Rootkits:** El término rootkit proviene de la unión de las palabras root (nombre por defecto de la cuenta privilegiada de los sistemas UNIX) y kit (conjunto de herramientas). Por lo tanto, un rootkit es un software que permite al atacante obtener acceso privilegiado al sistema. Normalmente utilizan técnicas de ocultación para permanecer escondidos, así como para esconder otras muestras de malware y evadir los sistemas de detección de los antivirus.

- **Spyware:** Un spyware es un software malicioso que infecta el ordenador o dispositivo móvil y recopila información sobre el usuario, su uso habitual de Internet, así como otros datos, enviándola al atacante.

- **Adware:** Un adware es cualquier programa que automáticamente muestra u ofrece publicidad, ya sea incrustada en una página web mediante gráficos, carteles, ventanas flotantes, o durante la instalación de algún programa al usuario, con el fin de generar lucro a sus autores.

- **Sniffers:** Los sniffers o analizadores de paquetes, son software cuya función principal es observar y registrar el tráfico de red. Analizan multitud de campos de los paquetes de red y suelen usarse para recopilar información del sistema o la red con la finalidad de preparar un ataque.

- **Botnet:** Las botnets son redes de ordenadores infectados que son controladas por los atacantes con la finalidad de realizar actividades maliciosas en conjunto. Sus principales usos son: realizar ataques de denegación de servicio distribuido (DDoS) [2], enviar spam masivo, o robar información.

- **Keyloggers:** Es un tipo especial de spyware. Como su nombre indica, se usa para registrar la actividad del teclado, con el objetivo de robar cualquier tipo de información sensible, como pueden ser contraseñas, números de tarjeta de crédito, etc.

- **Ransomware:** El ransomware (del inglés ransom, 'rescate', y ware, acortamiento de software) es un software, que como su nombre indica, obtiene el control de un ordenador, restringiendo su uso o acceso hasta que el usuario satisfaga las demandas del atacante (rescate). Estas demandas normalmente se refieren al pago de ciertas cantidades de dinero. Probablemente este sea uno de los tipos de malware que más está de moda actualmente y que mayor amenaza presenta.

El malware es una de las preocupaciones en el mundo de las tecnologías de la información, y en todos aquellos ámbitos donde exista un dispositivo conectado a la red. Cuando hablamos de malware, hablamos de la pérdida, robo, alteración y modificación de la información. Con

millones de programas maliciosos que se encuentran todos los días, el análisis de estos programas es fundamental para responder ante los incidentes de seguridad informática en las organizaciones.

2.1 Análisis de malware

El análisis de malware es el proceso que nos permite determinar el propósito y funcionamiento de una muestra de código malicioso.

La protección de los sistemas informáticos ante ataques con malware es uno de los mayores retos de seguridad, tanto para organizaciones como para individuos. Por eso es tan importante su análisis, porque nos permite estudiar la estructura, el funcionamiento y la interacción de dicho software malicioso, con el propósito de evaluar el daño causado, diseñar técnicas para su defensa y valorar las intenciones y capacidad de un atacante.

Tradicionalmente, el análisis de malware se realizaba de forma manual, lo cual era tedioso y poco eficiente [3], pero, ante el incremento de los ataques a sistemas informáticos y la imposibilidad de realizar dicho análisis de forma manual, se han desarrollado métodos para poder llevar a cabo esta tarea de forma más rápida y eficaz. Estos métodos se engloban en dos tipos principales, el análisis estático que se encarga de examinar la muestra de malware sin ejecutarla, y el análisis dinámico que consiste en analizar el comportamiento de la muestra cuando es ejecutada.

2.1.1 Análisis estático

El objetivo de este método es realizar un análisis inicial con el fin de extraer información útil del código malicioso para tomar una decisión informada sobre cómo clasificarlo o analizarlo y donde enfocar los esfuerzos de análisis posteriores.

De esta manera, es como si tuviéramos que realizar una autopsia para conocer qué es lo que hace la muestra o cuáles son las consecuencias que tendría si llegase a infectar un sistema. Un primer acercamiento nos va a permitir conocer si el malware está empaquetado, en qué lenguaje de alto nivel fue desarrollado, que librerías (DLLs) importa, que funciones va a utilizar, y el tamaño de sus secciones, entre otras características.

Para realizar el análisis estático de malware existen varias alternativas, en función de la información que se quiera obtener. Estas alternativas incluyen:

- **Análisis con antivirus:** La primera acción para poder reconocer una muestra de malware es analizarlo con un antivirus. Sin embargo, este análisis solamente será útil si la firma de la muestra se encuentra en la base de datos del antivirus. Una de las herramientas más populares es VirusTotal [4] que fue creada por la empresa de seguridad española

Hispasec Sistemas e incluye 55 antivirus y 61 motores de detección en línea. VirusTotal forma parte hoy en día del proyecto Chronicle de Google [5].

- **Análisis de cadenas:** Mediante la conversión de la muestra a texto se pueden identificar cadenas que nos permitan determinar un comportamiento sospechoso. Para este análisis se puede utilizar la herramienta Strings de SysInternals [6] que busca una secuencia de tres o más caracteres ASCII y Unicode con el objetivo de identificar información sospechosa.

- **Análisis de ofuscamiento y empaquetado:** A menudo, los creadores de malware ofuscan o empaquetan sus códigos para que los archivos sean difíciles de analizar. Los códigos ofuscados son aquellos para los que el autor intenta ocultar su ejecución. Los códigos empaquetados son un subconjunto de los ofuscados en los que la muestra maliciosa se encuentra comprimida y no puede ser analizada. Ambas técnicas limitan de forma severa los intentos de analizar de forma estática el malware. El código legítimo suele incluir un número elevado de cadenas. Las muestras empaquetadas u ofuscadas suelen presentar un número reducido de estas. El número de cadenas es un indicador sobre la ofuscación o empaquetado de una muestra, que podría ser indicativo, a su vez, de una muestra sospechosa. Entre las herramientas más conocidas para identificar código ofuscado o empaquetado, se encuentra PEiD [7].

- **Ingeniería inversa:** Esta disciplina se encarga del estudio del código de una muestra maliciosa para conocer su comportamiento e identificar que vulnerabilidades explota dicha muestra. Para realizar ingeniería inversa sobre una muestra existen diversas herramientas que son conocidas como desensambladores o depuradores, los cuales se encargan de transformar el código máquina a lenguaje ensamblador, para así poder analizarlo con mayor facilidad. Entre ellas destacan IDA [8], OllyDbg [9] o Radare2 [10].

El análisis estático básico es sencillo y puede ser rápido, pero es en gran medida ineficaz contra malware sofisticado y puede omitir comportamientos importantes. Por ello, este tipo de análisis se complementa con el análisis dinámico.

2.1.2 Análisis dinámico

El análisis dinámico o de comportamiento, se encarga de ejecutar una muestra en un entorno aislado para monitorizar su actividad, interacción y su efecto. Este análisis permite conocer de una manera rápida y efectiva las acciones que realiza la muestra en el sistema, entre las que se encuentran la creación de archivos, establecimiento de conexiones de red o modificaciones en el registro. La ejecución en un entorno aislado (denominado sandbox) se realiza para evitar la infección del propio equipo o de terceros.

Para realizar el análisis dinámico existen también un conjunto de herramientas que se pueden dividir en tres categorías principales:

- **Herramientas basadas en hooks** [1]**:** Estas herramientas utilizan técnicas para interceptar llamadas a funciones o mensajes entre componentes del sistema con el objetivo de conocer sus parámetros y monitorizar a través de ellos lo que sucede en el sistema. Una de estas herramientas es el monitor de procesos (Process Monitor) de la suite Sysinternals [6], que a través de filtros permite detectar acciones en el registro, en el sistema de archivos, o las conexiones de red que se producen.

- **Herramientas basadas en diferencias:** Estas herramientas capturan el estado del sistema en términos de sistema de archivos y de registro, en dos momentos distintos para luego comparar los resultados y conocer cuáles han sido las modificaciones realizadas por la ejecución de la muestra. Una herramienta de este tipo es ESET SysInspector [11], que permite detectar cambios en procesos en ejecución, en las conexiones de red o en las entradas en el registro.

- **Herramientas basadas en notificaciones:** Esta herramientas registran las rutinas de notificación que el sistema invoca de manera automática cuando se producen determinados eventos. Entre estos eventos podemos destacar la creación de un directorio o la eliminación de un archivo. A diferencia de las herramientas basadas en hooks, estas no interfieren en la llamada a la función, sino que registran eventos provistos por el sistema. Dentro de estas herramientas, es necesario destacar aquellas que se encargan del análisis del tráfico de red. Un ejemplo es WireShark [12], que permite la creación de filtros para poder analizar en detalle a qué sitios o direcciones IP se conecta un código malicioso cuando se ejecuta.

2.2 Ficheros PE

Para un analista de malware, es importante comprender la estructura de estos ficheros ya que la gran mayoría de las muestras de malware se basan en esta estructura.

El formato de ficheros *Portable Executable* (PE), como su nombre indica, es un formato para ficheros ejecutables usado en las versiones de 32 y 64 bits de los sistemas operativos Windows, desde Windows NT 3.1. Este formato lo constituyen binarios o librerías como los archivos *.exe, .sys, .dll, .efi* o *.ocx*.

Como podemos apreciar en la figura 2.1, un fichero PE presenta la siguiente estructura [13]:

[1] gancho, anzuelo

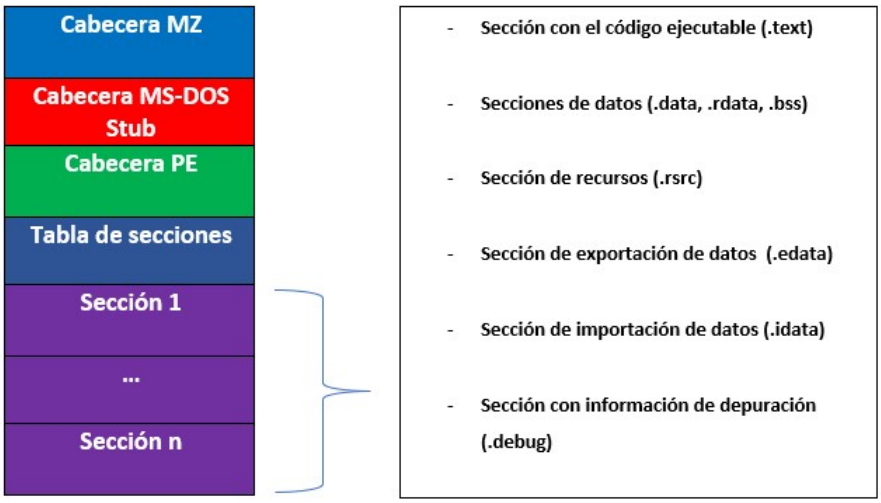

Figura 2.1: Estructura básica de un fichero PE.

- **Cabecera MZ**: esta cabecera contiene la combinación 0x4D5A, cuyo equivalente en ASCII es MZ (de ahí el nombre de la cabecera) por las siglas de Mark Zbikowski, uno de los desarrolladores de MS-DOS. Esta combinación permite identificar al fichero como ejecutable, por lo que si cambia su valor, no podrá ejecutarse en Windows.
- **Cabecera MS-DOS Stub**: esta cabecera indica si el archivo puede ser ejecutado desde la línea de comandos de MS-DOS.
- **Cabecera PE**: esta cabecera se divide en 3 partes, la firma (*Signature*), la cabecera (*FileHeader*) donde se incluye información como el número de secciones o la marca temporal; y una cabecera opcional (*OptionalHeader*) donde se puede incluir cierta información adicional.
- **Tabla de secciones**: Contiene una fila por cada una de las secciones del fichero PE que vendrán a continuación. Cada fila contiene información de cada una de las secciones entre la que se encuentra el nombre de la sección, su tamaño virtual o su tamaño real en disco.
- **Secciones**. El resto de los datos del archivo contiene las secciones ya propiamente dichas. Entre las secciones más destacables nos encontramos con:
  - **.text**: esta sección contiene el código ejecutable del fichero.
  - **.rdata**: esta sección contiene datos de sólo lectura. Algunas veces incluye información de exportación e importación.
  - **.idata**: esta sección contiene información sobre los datos que importa el fichero.

- **.edata**: esta sección contiene información sobre los datos exportados a los que otros archivos PE pueden acceder a través de la vinculación dinámica.
- **.data**: esta sección contiene las variables de lectura/escritura y las variables globales del fichero.
- **.rsrc**: esta sección contiene los recursos del fichero, como iconos, imágenes, cursores o sonidos.

A través de la información que ofrece la estructura de los ficheros PE, se pueden identificar muestras maliciosas y ello nos permitirá su detección.

Capítulo 3

Introducción teórica

Desde nuestra evolución, los seres humanos hemos utilizado diversos tipos de herramientas para realizar las tareas de una manera más sencilla. La creatividad del cerebro humano condujo, y sigue conduciendo cada día, a la invención de diferentes máquinas cuyo principal objetivo es facilitar el día a día de las personas, al permitirles satisfacer de una manera más sencilla sus necesidades y deseos. Prueba de estos avances son la creación de sistemas y tecnologías tales como los medios de transporte, las industrias e incluso la propia informática. El aprendizaje automático que estudiamos en este trabajo es, de hecho, una de estas herramientas.

Cada día, las organizaciones buscan formas con las que optimizar la gestión de datos para poder recabar la información verdaderamente valiosa y mejorar así su nivel de entendimiento sobre los datos recogidos. Para ello, cobran especial relevancia los modelos de aprendizaje automático, que permiten a las organizaciones contar con la capacidad de predicción de los cambios que van surgiendo en el negocio. No obstante, estos modelos se enfrentan a una dificultad: el flujo de datos es incesante. Por este motivo, es importante que se asegure que esta solución para predecir las variaciones se mantenga constantemente actualizada, para garantizar la capacidad de anticiparse a los cambios.

El uso de la inteligencia artificial y el aprendizaje automático no es algo nuevo, sino que se remonta a la década de los 50s. Es este el momento en que Arthur Lee Samuels, investigador de IBM, desarrolló uno de los primeros programas de aprendizaje automático: un programa de autoaprendizaje para jugar a las damas. De hecho, fue él quien acuñó el término de *aprendizaje automático* o *aprendizaje máquina*. Su enfoque del aprendizaje automático aparece explicado en un artículo publicado en IBM Journal of Research and Desarrollo en 1959 [14].

El aprendizaje automático (*Machine Learning* en inglés) es una rama de la inteligencia artificial, que permite a un sistema informático aprender a partir de información en lugar de mediante programación explícita [15]. Sin embargo, este sistema de aprendizaje no es un proceso simple. El aprendizaje automático utiliza una variedad de algoritmos para conseguir

que el sistema automáticamente aprenda de los datos, consiguiendo así que pueda mejorar de forma autónoma, identificar patrones y tomar decisiones sin apenas intervención humana. A medida que el algoritmo recibe más datos, este es capaz de producir modelos más precisos basados en dichos datos. Un modelo de aprendizaje automático es, por tanto, la salida generada de entrenar con datos al algoritmo. Por ejemplo, si tenemos un algoritmo de clasificación, este generará un modelo de clasificación, y por lo tanto, será capaz de crear de manera autónoma clasificaciones en función de los datos con los que el algoritmo fue entrenado.

El aprendizaje automático se divide en dos tipos principales: el aprendizaje supervisado, que se caracteriza por partir de un conjunto de datos previamente etiquetado, es decir, los resultados que deseamos obtener del modelo son conocidos previamente; y, por otro lado, el aprendizaje no supervisado, donde el resultado deseado no se utiliza durante el entrenamiento, llegando muchas veces incluso a no conocerse previamente. En este trabajo nos centraremos en el aprendizaje supervisado y en algunos de sus algoritmos.

3.1 Aprendizaje supervisado

Como mencionamos anteriormente, los algoritmos de aprendizaje supervisado basan su aprendizaje en un juego de datos de entrenamiento previamente etiquetados. Por etiquetado entendemos que para cada ocurrencia del conjunto de datos de entrenamiento conocemos el valor de su atributo objetivo. Esto le permitirá al algoritmo poder "aprender" una función capaz de predecir el atributo objetivo para un juego de datos nuevo (figura 3.1).

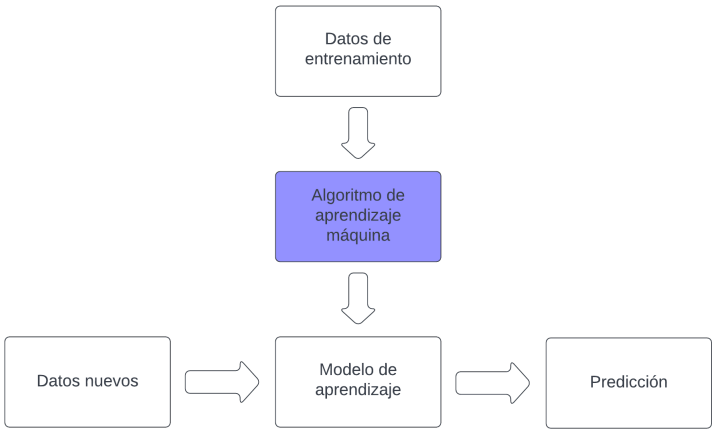

Figura 3.1: Proceso del aprendizaje supervisado.

Dependiendo de la etiqueta, existen dos categorías de algoritmos de aprendizaje supervisado: los algoritmos de regresión, que se utilizan para predecir un valor continuo, como por ejemplo, el precio de un inmueble a lo largo de los años; y los algoritmos de clasificación, que

se utilizan para predecir valores discretos, en nuestro caso, si un fichero ejecutable es una muestra de malware o no.

Como los algoritmos se entrenan con ejemplos preprocesados, existe la posibilidad de no realizar correctamente el entrenamiento y, por lo tanto, no obtener buenos resultados a la hora de hacer las predicciones. Existen dos posibles errores que podemos cometer cuando realizamos el entrenamiento: el sobre ajuste (*Overfitting*) y el sub ajuste (*Underfitting*) [16]. El *Overfitting* surge cuando "sobre-entrenamos" nuestro algoritmo con muestras de datos demasiado similares entre sí, por lo que no será capaz de realizar una buena predicción para datos que no se asemejen a los de entrenamiento. Por otro lado, el *Underfitting* surge cuando se carece de suficientes muestras de entrenamiento o existe un entrenamiento muy pobre y, a causa de ello, al modelo le faltará conocimiento para crear las predicciones. Para no cometer estos errores, debemos hallar un punto medio a la hora de realizar nuestro entrenamiento, es decir, deberemos contar con suficientes muestras de la mayor variedad posible.

A continuación describiremos los algoritmos de aprendizaje supervisado que se utilizarán en este trabajo.

3.2 Algoritmos de aprendizaje supervisado

3.2.1 Regresión logística

Aunque su nombre lleve a pensar que se trata de un algoritmo de regresión, la **Regresión Logística** (*Logistic Regression* en inglés) es en realidad un algoritmo de clasificación. La regresión logística es un método que se caracteriza por su simpleza y eficiencia para hacer frente a los eventuales problemas de clasificación binaria y lineal. Tiende a tener a su vez una ejecución sencilla y a generar un desempeño altamente satisfactorio.

Este algoritmo de aprendizaje hace uso de la función **sigmoide**,

$$y = \frac{1}{1 + e^{-}x} \tag{3.1}$$

que produce una curva en forma de S que es capaz de convertir y mapear cualquier cifra a un valor numérico englobado dentro del intervalo [0,1], sin llegar nunca a los límites de dicho intervalo (figura 3.2) [17].

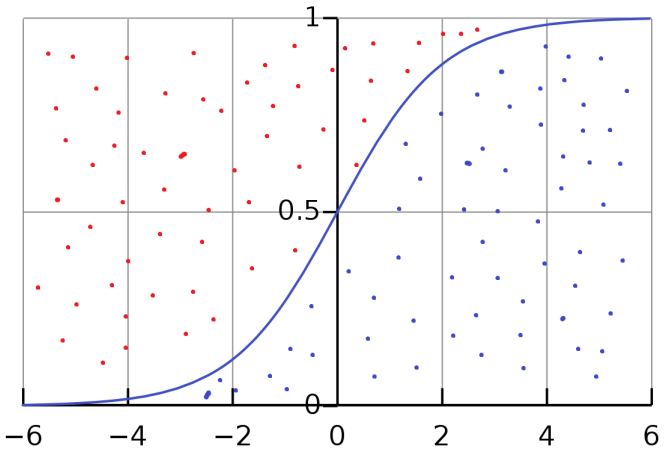

Figura 3.2: Función sigmoide.

3.2.2 Máquinas de vectores de soporte

Las **Máquinas de vectores de soporte** (*SVMs*) son un conjunto de algoritmos de aprendizaje supervisado basados en el uso de hiperplanos como separadores binarios de categorías.

Partiendo de un conjunto de datos divididos en dos clases distintas, el objetivo de estos algoritmos es encontrar un hiperplano que separe dichas clases de forma que el margen entre ambas clases sea el máximo posible.

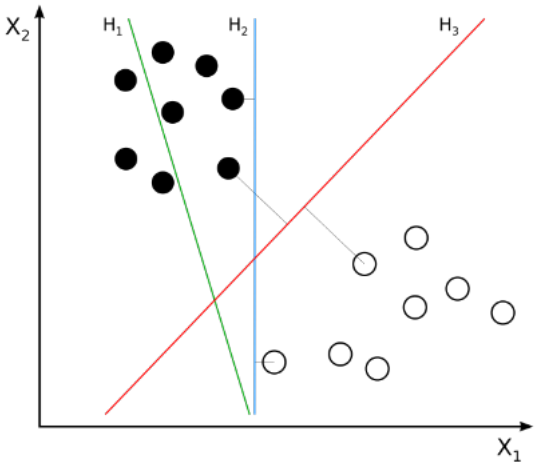

Figura 3.3: Posibles hiperplanos para 2 clases dadas.

Como vemos en la figura 3.3, para las dos clases de datos (puntos negros y puntos blancos, respectivamente), tenemos tres posibles hiperplanos. El hiperplano H1 no es un hiperplano válido ya que no separa correctamente las clases. El hiperplano H2 separa correctamente las clases pero el margen entre ambas es demasiado pequeño como para poder generalizar el

resultado. Por lo tanto, el hiperplano válido será H3, ya que separa correctamente las clases y el margen entre ambas es el máximo posible, dejando así la posibilidad de que aparezcan nuevos datos y se clasifiquen correctamente en dichas clases.

Muchas veces las clases no son directamente separables, por lo que tendremos que hacer uso del **Truco de Kernel** (*Kernel Trick* en inglés). El truco de kernel consiste en aplicar una función (*función kernel*), para asignar los datos a un espacio dimensional diferente (normalmente superior al original), con el objetivo de facilitar la separación de las clases. Existen distintos tipos de funciones kernel, entre las que se encuentran [18]:

- **Función de base radial (RBF) o gaussiana**

$$K(x_1, x_2) = e^{-\frac{||x_1 - x_2||^2}{2\sigma^2}} \tag{3.2}$$

- **Lineal**

$$K(x_1, x_2) = x_1^T x_2 \tag{3.3}$$

- **Polinómica**

$$K(x_1, x_2) = (x_1^T x^2 + 1)^\rho \tag{3.4}$$

- **Sigmoide**

$$K(x_1, x_2) = tanh(\beta_0 x_1^T x_2 + \beta_1) \tag{3.5}$$

Donde x_1 y x_2 representan las coordenadas del punto en el plano, σ representa la anchura del Kernel, x_1^T representa la matriz traspuesta de x_1, ρ representa el orden del polinomio, y β_0 y β_1 son dos parámetros ajustables del kernel sigmoide.

3.2.3 K-Vecinos Más Cercanos

El algoritmo **K-Vecinos Más Cercanos** (*KNN*) es un algoritmo de clasificación que se basa en el concepto de que proximidad equivale a similitud, es decir, los elementos más próximos entre sí, suelen pertenecer a la misma clase.

Para poder seleccionar los K vecinos más cercanos a un nuevo elemento, el algoritmo KNN calcula las distancias haciendo uso de la distancia euclídea [19]:

$$d(A, B) = \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2} \tag{3.6}$$

Donde A y B son los dos puntos de los que se quiere hallar su distancia y (xA,yA) y (xB,yB) son sus coordenadas.

Para la elección del valor de K no existe un método fijo, sino que probaremos con distintos valores y nos quedaremos con el valor que menos errores produzca. Cuanto mayor sea el valor

de K, menos errores producirá, siempre y cuando no se pierda el límite entre clases.

Una vez calculadas las distancias y seleccionado el valor de K, la clase perteneciente al nuevo elemento será la clase más frecuente entre esos K vecinos más cercanos. Por ello es importante que si nos encontramos con un problema de clasificación binaria, el valor de K sea impar para que siempre exista mayoría de una clase.

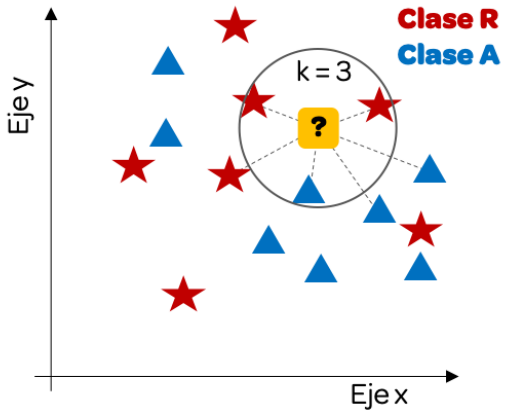

Figura 3.4: Ejemplo del algoritmo KNN.

3.2.4 Naive Bayes

El algoritmo Naive Bayes es un algoritmo de aprendizaje supervisado que pertenece al grupo de los algoritmos probabilísticos, ya que hace uso de la probabilidad de un objeto para obtener la predicción.

Este algoritmo se basa en el **Teorema de Bayes** [20] para resolver problemas de clasificación. Este teorema se utiliza para calcular la probabilidad de una hipótesis con cierto conocimiento previo. Su fórmula es la siguiente:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{3.7}$$

Donde:

- **A** y **B** son hipótesis
- **P(A)** es la probabilidad de la hipótesis A
- **P(B)** es la probabilidad de la hipótesis B
- **P(B|A)** es la probabilidad de la hipótesis B dada la hipótesis A
- **P(A|B)** es la probabilidad posterior de la hipótesis A dada la hipótesis B

El funcionamiento del algoritmo es muy sencillo. Primero debemos convertir nuestro conjunto de datos a una tabla de frecuencias y creamos otra tabla con las probabilidades de que ocurran los distintos eventos. Por último, aplicamos el Teorema de Bayes para calcular la probabilidad posterior de cada clase y seleccionamos como resultado la clase con la probabilidad más alta.

A este algoritmo también se le llama algoritmo *Inocente* (del término Naive en inglés), ya que asume que la presencia de una cierta característica en un conjunto de datos no está en absoluto relacionada con la presencia de cualquier otra característica [21]. Es un algoritmo sencillo que genera modelos con muy buen comportamiento.

3.2.5 Árboles de Decisión

Los **Árboles de Decisión** (*Decision Trees* en inglés) conforman el algoritmo más utilizado en aprendizaje automático, basado en una estructura de ramificaciones de la información, dividida en subconjuntos en función de sus características. Cada nodo del árbol representa una característica de nuestro conjunto de datos, y cada rama el valor que puede tomar dicha característica.

Existen tres tipos de nodos según su nivel jerárquico dentro del árbol:

- **Nodo raíz**: representa la característica que mejor divide nuestro conjunto de datos.
- **Nodos internos**: representan las características que continúan dividiendo el árbol.
- **Nodos hoja**: representan la clasificación final del algoritmo.

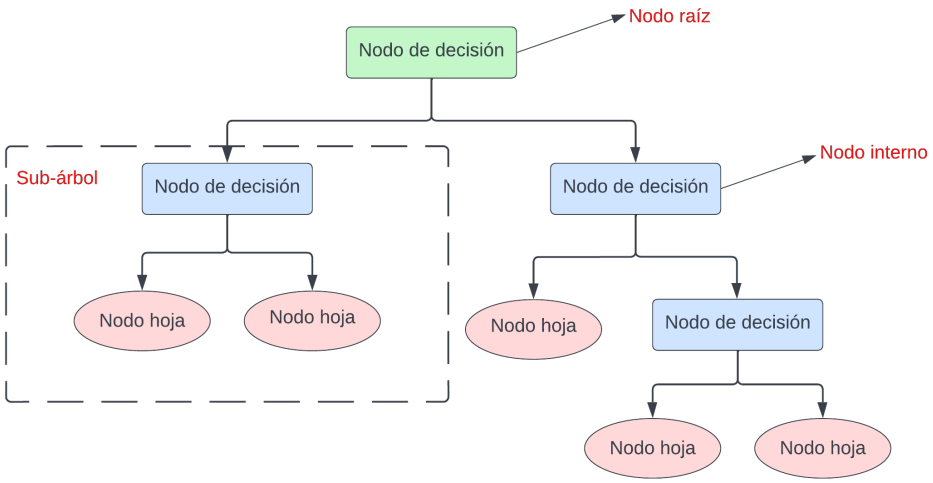

Figura 3.5: Estructura de un árbol de decisión.

Para la creación de un árbol de decisión se utiliza el **algoritmo de Hunt** [22], algoritmo basado en la división de subconjuntos. Si los elementos que componen un nodo pertenecen a la misma clase, este nodo será terminal (nodo hoja). Por el contrario, si los elementos que componen un nodo no pertenecen a una única clase, los elementos se dividirán en subconjuntos más pequeños en función de una condición, y se creará un nodo hijo para cada salida de dicha condición. Este proceso se repetirá para cada nodo hijo.

3.2.6 Bosques Aleatorios

El algoritmo de **Bosques Aleatorios** (*Random Forest*) es un algoritmo formado por la combinación de otros algoritmos (*ensemble* en inglés). En el caso de Random Forest, como su nombre indica, es un bosque o conjunto de distintos árboles de decisión que se puede utilizar tanto para problemas de clasificación como para problemas de regresión.

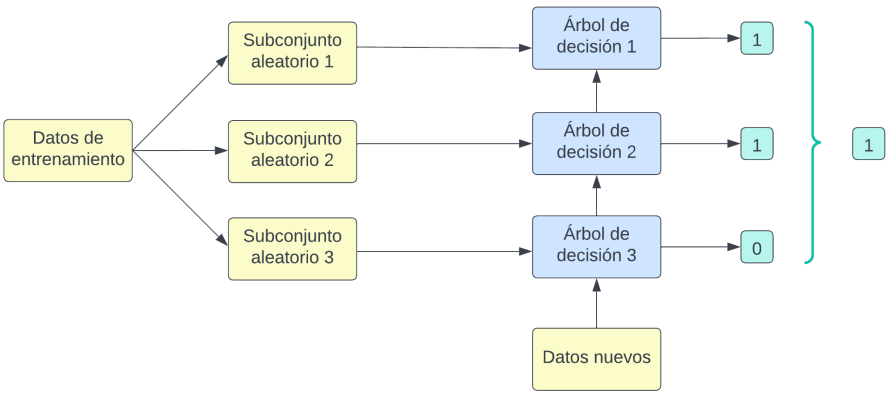

Figura 3.6: Método bagging en un Random Forest.

Cuando hablamos de algoritmos *ensemble*, nos encontramos distintos métodos para poder combinar los distintos algoritmos que conformarán el algoritmo principal [23]. Entre estos métodos destacan:

- **Boosting**: Este método consiste en crear una secuencia de clasificadores, donde cada clasificador trabaja sobre las muestras incorrectamente clasificadas del anterior clasificador en la secuencia. El algoritmo que mejor representa este método es Adaboost [24], aunque es propenso al sobre ajuste.
- **Stacking**: Este método consiste en apilar distintos algoritmos de aprendizaje automático, es decir, la salida de un algoritmo será la entrada de otro y así sucesivamente hasta obtener la predicción final.

- **Bagging**: Esta técnica implica entrenar a cada uno de los algoritmos con un subconjunto aleatorio del conjunto de datos, de manera que cada subconjunto sea distinto. La predicción final será la que obtenga la mayoría de los modelos generados. Este método es más robusto frente al problema del sobre ajuste y es la técnica que se utiliza en Random Forest.

3.2.7 Modelos de redes neuronales

Una red de neuronas artificiales (RNA) es una representación simplificada de la manera en la que el cerebro humano procesa la información. Su funcionamiento consiste en simular las neuronas del cerebro humano haciendo uso de una gran cantidad de unidades de procesamiento interconectadas entre si.

Estas unidades de procesamiento se agrupan en capas, existiendo, normalmente, tres tipos (figura 3.7):

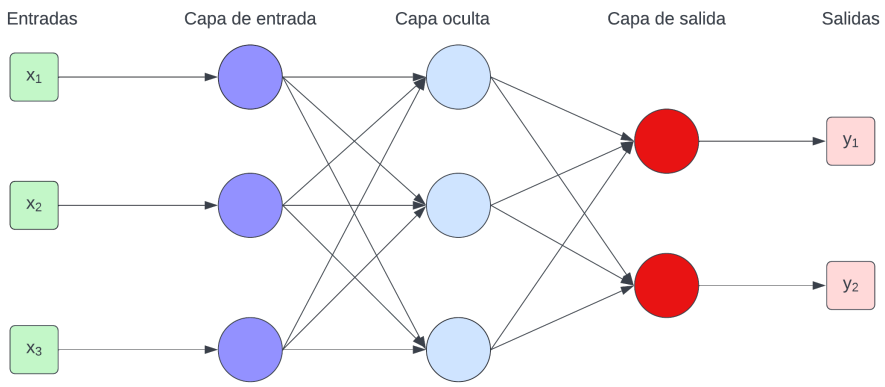

Figura 3.7: Ejemplo sencillo de una RNA.

- **Capa de entrada (*input layer*)**: representa los datos de entrada de la red.
- **Capa oculta (*hidden layer*)**: no tiene conexión directa con el entorno, es decir, se desconocen sus valores de entrada y sus valores de salida. Pueden existir varias capas ocultas.
- **Capa de salida (*output layer*)**: representa la salida de la red neuronal.

Las distintas unidades de procesamiento se interconectan mediante conexiones variables llamadas **ponderaciones** o **pesos**. Estos pesos son totalmente aleatorios al principio del entrenamiento, pero irán cambiando mediante el proceso de aprendizaje de la red. Este proceso consiste en tres fases [25]:

- **Propagación hacia delante** (*Forward propagation*): Esta fase consiste en pasar los datos de entrada por toda la red de forma que todas las neuronas apliquen su transformación a la información que reciben de las neuronas de la capa anterior, enviándola a las neuronas de la capa siguiente. Cuando los datos hayan cruzado todas las capas y todas sus neuronas hayan hecho sus cálculos, se llegará a la capa final con un resultado de predicción para esos ejemplos de entrada.
- **Función de pérdida o error** (*Loss function*): El siguiente paso es estimar el error (*Loss*) de nuestras predicciones en función del resultado esperado. Un error con valor cero quiere decir que no existe diferencia entre el resultado estimado y el resultado esperado.
- **Propagación hacia atrás** (*Back propagation*): Una vez calculado el error, esta información se propagará hacia atrás por las diferentes capas de la red. Cada neurona recibirá únicamente la fracción del error relativa a su contribución en la predicción y ajustará los pesos en base a la información recibida.

Existen multitud de modelos de RNAs en función de su estructura y las características de las unidades de procesamiento. Entre estos tipos se encuentra el modelo en el que nos centraremos en este trabajo, las **RNAs prealimentadas** (*Feedforward artificial neural networks*). Este tipo de RNAs son redes neuronales donde las conexiones entre las distintas unidades de procesamiento no forman un ciclo. Se les llama *feedforward* porque los datos solo avanzan hacia delante en la red, primero por la capa de entrada, a continuación por las capas ocultas y, por último, por la capa de salida [26].

Capítulo 4

Antecedentes

La detección y clasificación de archivos ejecutables maliciosos no es un problema nuevo en el campo de la ciberseguridad, sino que lleva intentando desde hace mucho tiempo mediante mediante multitud de técnicas distintas.

Anteriormente esta clasificación se realizaba mediante la detección de firmas de malware conocido. Estas firmas estaban compuestas por muchas propiedades diferentes, como por ejemplo: el nombre de archivo, las cadenas de texto del código binario o códigos de bytes. Este método requería que los expertos analizasen los programas sospechosos de forma manual. A pesar de que este método para realizar el análisis era preciso, también era lento y costoso. Si existe un pequeño conjunto de muestras malignas que analizar, este método funciona correctamente, pero a medida que se aumenta el número de muestras se vuelve inviable realizar el análisis de malware de esta forma. Desafortunadamente también existe la posibilidad de que una nueva muestra no contenga ninguna firma conocida, por lo que los métodos basados en firmas no detectarían este nuevo ejecutable malicioso. Es por esto que encontrar métodos automáticos para generar clasificadores de malware ha sido objeto de investigación durante los últimos años.

4.1 Estudios antecedentes

Con el objetivo de tener más claro el alcance de nuestro estudio, se ha revisado el trabajo previo de distintos autores en el ámbito de la detección y clasificación de malware. El objetivo es conocer la metodología de trabajo para atajar este problema.

En el año **2001 Schultz et al.** [27] realizan un estudio donde comparan el uso de tres tipos de algoritmos distintos: un algoritmo basado en reglas (RIPPER [28]), un algoritmo probabilístico (Naive Bayes) y un sistema multiclasificador que combina las salidas de varios clasificadores para generar la predicción (Multi-Naive Bayes). En este estudio no solo se han examinado ficheros PE, si no que se han examinado varios tipos de ficheros, de los cuales 3265

son malignos (38 son PE) y 1001 benignos (206 son PE). En este estudio se utilizan diferentes características en cada uno de los algoritmos mencionados. Para el algoritmo RIPPER se utilizan las librerías DLLs usadas por el binario así como las funciones a las que llaman dichas librerías; para Naive Bayes se usan las cadenas de texto presentes en los ficheros, y para el algoritmo Multi-Naive Bayes se utilizan las secuencias de bytes de los ficheros. De los tres algoritmos, los mejores resultados se consiguen con el algoritmo multiclasificador, en donde la tasa de detección alcanza el 97,76%.

Shafiq et al. (2009) en su trabajo **PE-Miner** [29] extraen 189 características de los ficheros PE, concretamente información sobre: DLLs usadas, la cabecera COFF, la cabecera opcional, las cabeceras de sección o la tabla de directorio de recursos, entre otras. Estas características son extraídas de un conjunto de archivos formado por 15925 muestras malignas y 1447 muestras benignas, a los que se les aplica un preprocesado para eliminar o combinar ciertas características similares para reducir la dimensionalidad del conjunto de datos. Las técnicas que se utilizan para dicho preprocesado son: Eliminación de Características Redundantes (RFR), Análisis de Componentes Principales (PCA) y Transformación de Ondícula de Haar (HWT). Al contrario que en otros estudios donde las muestras se clasifican entre benignas o malignas, este trabajo se centra en la clasificación de distintos tipos de malware como pueden ser: virus, troyanos, gusanos, sniffers, exploits, DoS o puertas traseras. La clasificación de estos tipos de malware se realiza mediante cinco modelos: un algoritmo de Aprendizaje Basado en Instancias (IBk), Árboles de Decisión, Naive Bayes, RIPPER y SVM, siendo el algoritmo de Árboles de Decisión el que mejores resultados con un 0,992 de *AUC* (área bajo la curva) en el mejor de los casos. A su vez es el algoritmo que menos carga computacional genera.

En el estudio de **Wang et al. (2009)** [30] se propone un método de detección de malware basado en el algoritmo SVM. Como conjunto de datos utiliza 1908 muestras benignas y 7863 malignas (913 virus, 808 gusanos, 2809 troyanos y 3333 puertas traseras), siendo todos ficheros de tipo PE. Para la extracción de características utilizan la herramienta de Microsoft, DUMPBIN [31], que permite volcar la información de las cabeceras PE. Para simplificar el proceso de entrenamiento en este estudio han decidido convertir todos los valores numéricos o nominales a valores binarios. Para los atributos numéricos primero ordenan todos sus valores y posteriormente realizan una división que generará dos intervalos de atributos distribuidos entre las clases. Para cada uno de los atributos nominales se transformarán sus valores a valores binarios, escogiendo uno de los valores como referencia (1 en binario) y los demás como la negación de este (0 en binario). Una vez realizado el entrenamiento obtienen una precisión del 98,19% y 93.96% para la detección de virus y gusanos respectivamente, y una precisión del 84,11% y 89.54% para la detección de troyanos y puertas traseras.

En el año **2012, Liao** realiza un estudio [32] donde propone su propio algoritmo basado en reglas para la detección de malware. Para la obtención de características extrae informa-

ción de la cabecera de fichero, la cabecera opcional y la cabecera de sección de 1237 ficheros PE benignos y 5598 malignos. Lo interesante de este estudio es que a través de tan solo cinco características (tamaño de datos inicializados, nombre de sección, número de características DLL, versión de imagen principal y el checksum) obtiene excelentes resultados. Concretamente, la tasa de verdaderos positivos es del 99,5% y la tasa de falsos positivos es del 0,16%.

En el estudio de **Kumar et al. (2020)** [33] proponen una comparación de los resultados tras aplicar distintos algoritmos de aprendizaje supervisado: Árboles de Decisión, Bosques Aleatorios, Potenciación del Gradiente, Máquinas de Vectores de Soporte, Regresión logística, XGBoost y AdaBoost. Utilizan un conjunto de datos compuesto por alrededor de 121000 ficheros PE de los que se extraen 57 características. Sobre este conjunto de datos realizan un preprocesado de los datos, para convertir los valores de las características categóricas a formato numérico con el objetivo de que puedan ser leídos por los algoritmos. Una vez realizado el preprocesado de los datos, seleccionan 15 características tras hacer uso de la clase *ExtraTreesClassifier* de *Scikit-learn*. Tras haber realizado el entrenamiento obtienen muy buenos resultados en todos los modelos, superando siempre el 94% de precisión y llegando a alcanzar el 99,7% para el algoritmo de Árboles de Decisión, siendo a su vez el que menor carga computacional genera.

Capítulo 5

Tecnologías y material usado

En este capítulo describiremos brevemente las tecnologías usadas, así como el entorno de desarrollo de nuestro trabajo y describiremos el conjunto de datos de trabajo.

5.1 Lenguaje de programación y entorno de desarrollo

El lenguaje de programación empleado en el desarrollo de este trabajo es Python [34]. Python es un lenguaje de alto nivel, interpretado y orientado a objetos. Fue creado por el informático holandés Guido Van Rossum y lanzado en 1991 con el número de versión 0.9.0. Actualmente es el lenguaje de programación más popular en todo el mundo [35] ya que se utiliza tanto para desarrollo web (*backend*), desarrollo software, matemáticas y *scripting* [1] de sistemas. Pero sobre todo destaca su uso en el campo de la inteligencia artificial.

Actualmente existen dos versiones principales del lenguaje, *Python 2.X* y *Python 3.X*. Aunque ambas versiones son bastante populares, la versión 2.X dejó de ser soportada oficialmente el 1 de Enero de 2020, por lo que su uso se está migrando hacia la versión 3.X. En este trabajo utilizaremos la última versión estable, la versión 3.10.5.

Como entorno para el desarrollo se ha utilizado VSCode [36], ya que nos ofrece una gran comodidad y versatilidad a la hora de gestionar nuestro código, así como una integración bastante sencilla con el depurador. Adicionalmente, se ha creado un repositorio en Github [37] para mantener el control de versiones de nuestro código.

5.1.1 Librerías auxiliares de Python

A continuación describiremos las librerías de Python utilizadas para realizar nuestro trabajo:

[1] El término **script** se utiliza para hacer referencia a aquellos programas informáticos simples que no requieren ser compilados para su ejecución, si no que son ejecutados por un intérprete que lee el código fuente al momento.

Pandas

Pandas [38] es una librería de Python especializada en el manejo y análisis de estructuras de datos de alto rendimiento. Utilizaremos esta librería en su versión 1.4.2 debido a la facilidad que nos ofrece para reordenar, dividir o combinar conjuntos de datos, así como para leer y escribir ficheros en formato CSV.

Scikit-learn

Scikit-learn o SKlearn [39] es la librería más utilizada dentro del campo de la inteligencia artificial y el aprendizaje automático. Nos brinda implementaciones de algoritmos de aprendizaje supervisado, tales como, Árboles de Decisión, Perceptrón Multicapa, Máquinas de Soporte Vectorial o Bosques Aleatorios.

Usaremos esta librería en su versión 1.1.1 para realizar el entrenamiento y test de los distintos algoritmos de aprendizaje supervisado propuestos, del mismo modo que para realizar la validación cruzada y para el cálculo de diversas métricas de evaluación de modelos.

Seaborn

Seaborn [40] es una librería orientada a la visualización gráfica de datos basada en Matplotlib [41] e integrada con las estructuras de datos de Pandas. Se ha decidido usar Seaborn en su versión 0.11.2 y no Matplotlib porque Seaborn es mucho más funcional y organizada que Matplotlib, además de resultar más cómoda a la hora de manejar el conjunto de datos.

Imbalanced-Learn

Como su nombre indica, Imbalanced-Learn [42], es una librería utilizada para ayudarnos con el balanceo de conjuntos de datos desbalanceados. Utilizaremos esta librería en su versión 0.9.1 debido al desbalanceo presente en nuestro conjunto de datos.

5.2 Descripción del conjunto de datos

En este trabajo se utilizarán ficheros PE como conjunto de datos. Este conjunto de ficheros PE se divide en dos subconjuntos: ficheros de malware y ficheros benignos.

Los ficheros de malware han sido extraídos del sitio web VirusShare [43]. VirusShare es un repositorio de muestras de malware donde podemos obtener innumerables muestras de código malicioso real, al que se accede por invitación. En total obtenemos 87061 muestras malignas.

Por otro lado, el conjunto de muestras benignas está formado por 4114 ficheros benignos extraídos del repositorio DikeDataset [44] y de nuestro propio equipo . Estos últimos, se

corresponden con ficheros PE extraídos de la carpeta System32 de nuestro sistema, carpeta donde residen la gran parte de archivos ejecutables de un sistema Windows.

Capítulo 6

Metodología y planificación

Ante el auge del uso de metodologías ágiles en el ámbito empresarial y nuestra necesidad de organizar las distintas fases de nuestro proyecto, hemos decidido utilizar **SCRUM** como metodología a seguir en nuestro trabajo.

SCRUM [45] (término procedente del rugby, que hace referencia a la forma en la que el equipo se esfuerza conjuntamente para hacer avanzar la pelota por el campo) es una metodología de trabajo que se basa en aplicar un conjunto de buenas prácticas para trabajar en equipo y así conseguir el mejor resultado posible de un proyecto. Dentro del equipo existen los siguientes roles:

- **Responsable del producto**: Esta persona es la responsable de tener la visión de lo se va a hacer y conseguir. Es el primer contacto con el cliente.
- **SCRUM Master**: persona que se encargará de que el equipo aplique la metodología SCRUM en el proyecto.
- **Equipo de desarrollo**: lo conforman las personas encargadas de realizar el trabajo. Deben tener las habilidades necesarias para convertir en realidad la visión del responsable del producto. El equipo de desarrollo debe ser de entre 3 y 9 personas.

Esta metodología tiene como base el desarrollo iterativo del producto, es decir, realizar entregas parciales y funcionales del producto final de manera regular. Para ello es necesario que existan una serie de eventos predefinidos con el fin de crear regularidad y cumplir con la planificación:

- **Sprint**: es el pilar principal de esta metodología ya que hace referencia al periodo de tiempo en el cual se lleva a cabo el trabajo correspondiente para cierta entrega. La duración no debe superar las 4 semanas.
- **Planificación de sprint**: Esta es la primera reunión del equipo, donde se deben definir las tareas a hacer y el objetivo de cada uno de los sprints.

- **Reunión diaria (*Daily*)**: en esta reunión debe participar tanto el equipo de desarrollo como el SCRUM Master. En ella cada persona hablará de lo que hizo el día anterior, de lo que hará hoy y de si existe algún punto bloqueante. Esta reunión debe hacerse de pie y su duración no debe superar los 15 minutos independientemente del número de participantes.
- **Demo del sprint**: esta es la reunión en la que el equipo muestra lo que ha conseguido en el sprint. En ella puede estar presente cualquiera, tanto el responsable del producto, el SCRUM Master y el equipo, como el cliente. En esta reunión el equipo le mostrará al cliente aquello que esté completamente terminado y sea entregable.
- **Retrospectiva del sprint**: esta es la última reunión del sprint. Después de la demo, el equipo se sentará a reflexionar sobre lo que ha ido bien, lo que podía haber ido mejor y lo que se podría perfeccionar en el siguiente sprint.

Aunque SCRUM surgió en empresas del ámbito tecnológico, es apropiada para cualquier tipo de proyecto donde se necesite obtener resultados pronto, donde los requisitos son cambiantes o poco definidos, y donde la flexibilidad y productividad son fundamentales.

6.1 Fases de desarrollo

En nuestro caso utilizaremos una versión simplificada de SCRUM manteniendo el concepto de sprint para poder llevar a cabo las siguientes fases del proyecto:

1. Adquisición de conocimientos teóricos sobre el análisis de malware.
2. Adquisición de conocimientos teóricos sobre el aprendizaje máquina.
3. Diseño de la estructura general del proyecto.
4. Selección y análisis del conjunto de datos de trabajo.
5. Preparación del conjunto de datos de trabajo.
6. Desarrollo y análisis de resultados de cada uno de los modelos de aprendizaje supervisado.
7. Evaluación de los resultados de los modelos y comparación con otros estudios.

El diseño de la estructura general del proyecto (fase 3) consiste en la división de las distintas funcionalidades del código en dos archivos con nuestro código Python:

- **extract_features.py**. En este script se encontrará el código usado para extraer cada una de las características de los ficheros PE y exportarlas a los archivos *dataset_malware.csv* y *dataset_benign.csv*.
- **models.ipynb**. Este archivo contendrá el código para gestionar el conjunto de datos y realizar el entrenamiento y test de los algoritmos de aprendizaje supervisado. Está dividido en celdas, de tal forma que podemos ejecutar el código de cada celda de manera independiente.

La fase de desarrollo de los modelos y análisis de los resultados (fase 6) se divide en las siguientes subfases:

1. Diseño e implementación de los modelos.
2. Validación y pruebas de los modelos.
3. Evaluación de resultados de cada modelo.

En la figura 6.1 se puede ver el diagrama de Gantt que detalla la planificación por semanas de las distintas fases del proyecto.

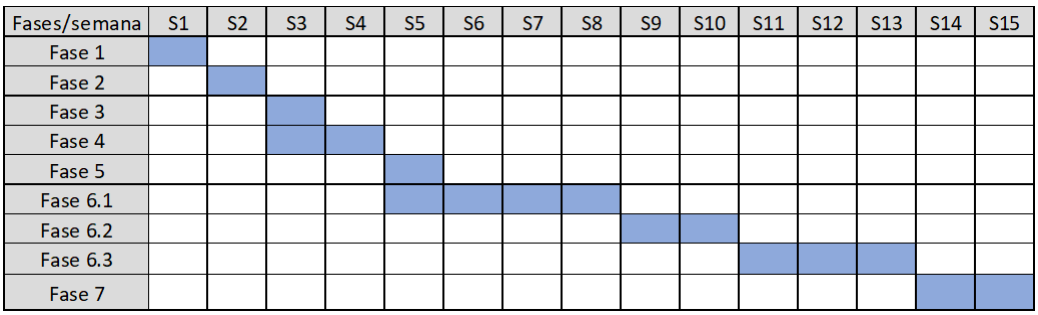

Figura 6.1: Diagrama de Gantt de la planificación del proyecto.

En cada una de las fases se realizará la redacción de la parte de la memoria que corresponda a dicha fase.

6.2 Estimación de costes

En todo proyecto real existe un coste, producto del salario de los recursos, del coste de materiales y de otros gastos. Para realizar la estimación de costes en nuestro proyecto, se ha decidido incluir la participación de dos perfiles junior para el desarrollo del proyecto, un ingeniero de computación y un programador.

La misión del ingeniero es diseñar la estructura general del proyecto, seleccionar y gestionar el conjunto de datos, así como definir los modelos de aprendizaje supervisado y evaluar sus resultados.

El papel del programador será ayudar al ingeniero en tareas de programación donde no se necesite de amplios conocimientos en el ámbito del aprendizaje automático.

Tanto el ingeniero como el programador tienen un salario distinto, por lo que el coste temporal se calcula suponiendo un precio de 20€/h para el ingeniero junior especializado en computación y de 15€/h para el programador junior (tabla 6.1):

Fase	**Perfil**	**Esfuerzo aprox. (horas)**	**Coste (€)**
1	Ingeniero	10	200
2	Ingeniero	15	300
3	Ingeniero	4	80
4	Ingeniero	15	300
	Programador	10	150
5	Programador	5	75
6	Ingeniero		
6.1	-	50	1000
6.2	-	30	600
6.3	-	40	800
7	Ingeniero	20	400
		Total: 199	3905

Tabla 6.1: Coste de los recursos.

Adicionalmente, para llevar a cabo un proyecto se necesita disponer del material de trabajo y los servicios necesarios, lo que implica unos gastos adicionales:

Elemento	Coste (€)
Internet	250
Electricidad	400
Equipo de desarrollo	2000
TOTAL:	2650

Tabla 6.2: Coste de materiales y servicios.

Todo esto implica un coste total del proyecto de 6555€.

6.3 Validación cruzada con K-fold

Es posible que al realizar la división de los conjuntos de entrenamiento y test se dé el caso de que todas las muestras de la clase *Benigna* estuviesen en el conjunto de entrenamiento, y las de la clase *Malware* en el conjunto de test (o viceversa). Esta posibilidad aumenta si el conjunto de datos está desbalanceado.

Para evitar esta posible problemática, se ha decidido realizar la validación del entrenamiento de nuestros modelos a través de la validación cruzada mediante el método *K-fold* [46].

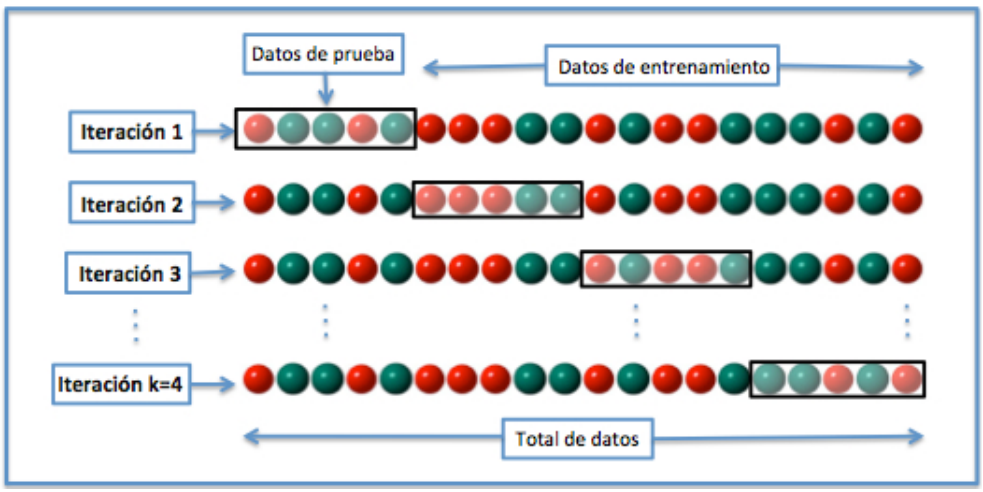

Figura 6.2: Validación cruzada con 4-fold.

Como podemos observar en la figura 6.2, el funcionamiento de este método consiste en dividir el conjunto de datos en K subconjuntos (*folds*). Uno de los subconjuntos se utilizará como datos de validación y el resto como datos de entrenamiento. Esto se repetirá para cada uno de los K posibles subconjuntos de validación. Por último se halla la media aritmética de

los resultados de las K iteraciones para obtener el resultado final. En nuestro caso utilizaremos *K-fold* con K=5.

6.4 Medidas de rendimiento

A continuación describiremos las medidas de rendimiento utilizadas para evaluar los resultados de los modelos desarrollados.

Exactitud

La exactitud (accuracy) representa el porcentaje de casos acertados sobre el total de predicciones.

$$accuracy = \frac{VP + VN}{VP + FP + VN + FN} \tag{6.1}$$

donde VP y VF son los verdaderos positivos y verdaderos negativos respectivamente, y FP y FN son los falsos positivos y falsos negativos respectivamente.

Precisión

La precisión (precision) se define como el porcentaje de casos positivos correctamente clasificados sobre el total de positivos clasificados (verdaderos positivos y falsos positivos).

$$precision = \frac{VP}{VP + FP} \tag{6.2}$$

Sensibilidad

La sensibilidad (recall) puede definirse como la proporción de casos positivos correctamente clasificados sobre el total de positivos reales (verdaderos positivos y falsos negativos).

$$recall = \frac{VP}{VP + FN} \tag{6.3}$$

Medida-F1

La medida-F1 (F1-score) es la media armónica entre la precisión y la sensibilidad.

$$F1 = 2 * \frac{precision * recall}{precision + recall} \tag{6.4}$$

Capítulo 7

Desarrollo y resultados

En este capítulo describiremos el proceso seguido para el desarrollo de los diferentes modelos de aprendizaje supervisado, así como los resultados que se obtienen con cada uno de ellos. Este desarrollo incluye las tareas de preparación de los datos para su consumo por los modelos de aprendizaje automático, y de selección de características motivado por los resultados obtenidos.

7.1 Preprocesado de datos

El contenido de nuestro conjunto de datos es el resultado de extraer 76 características de las 91175 muestras (4114 benignas y 87061 malignas) correspondientes a ficheros PE del sistema operativo Windows. En la tabla 7.1 se pueden ver algunos de estos datos para un conjunto reducido de dichas características, las cuales han sido extraídas de la cabecera *FileHeader*, de las cabeceras opcionales y de la tabla de secciones (información descrita en la sección 2.2). La descripción completa de todas estas características se puede encontrar en el apéndice A.

	e_magic	e_cblp	e_cp	e_crlc	e_cparhdr	...	ImageDirectoryEntryExport	ImageDirectoryEntryImport	ImageDirectoryEntryResource	ImageDirectoryEntryException	ImageDirectoryEntrySecurity
0	23117	144	3	0	4	...	4192	0	12288	0	3584
1	23117	144	3	0	4	...	4192	0	12288	0	3584
2	23117	144	3	0	4	...	438464	438888	475136	458752	0
3	23117	144	3	0	4	...	940080	940228	983040	958464	0
4	23117	144	3	0	4	...	89088	89168	106496	102400	0
...	...	...	...	...	...	...	...	...	...	...	...
91170	23117	144	3	0	4	...	0	15012	24576	0	0
91171	23117	144	3	0	4	...	0	246820	258048	0	0
91172	23117	144	3	0	4	...	0	21796	28672	0	0
91173	23117	144	3	0	4	...	0	87616	91008	0	0
91174	23117	144	3	0	4	...	0	104772	118784	0	0

91175 rows × 76 columns

Figura 7.1: Ejemplo de un fragmento del conjunto de datos.

Si realizamos un análisis estadístico sobre nuestro conjunto de datos, vemos que existen características cuya desviación típica tiene valor 0 (tabla 7.2), es decir, son características que presentan el mismo valor para todas las muestras del conjunto de datos.

(a) Muestras benignas

	e_magic	e_cblp	e_cp	e_crlc	e_cparhdr
count	4114.0	4114.0	4114.0	4114.0	4114.0
mean	23117.0	144.0	3.0	0.0	4.0
std	0.0	0.0	0.0	0.0	0.0
min	23117.0	144.0	3.0	0.0	4.0
25%	23117.0	144.0	3.0	0.0	4.0
50%	23117.0	144.0	3.0	0.0	4.0
75%	23117.0	144.0	3.0	0.0	4.0
max	23117.0	144.0	3.0	0.0	4.0

(b) Muestras maliciosas

	e_magic	e_cblp	e_cp	e_crlc	e_cparhdr
count	87061.0	87061.000000	87061.000000	87061.000000	87061.000000
mean	23117.0	410.199446	328.580432	259.449512	206.039409
std	0.0	2578.268582	2886.986401	2441.438010	2042.891389
min	23117.0	0.000000	0.000000	0.000000	0.000000
25%	23117.0	80.000000	2.000000	0.000000	4.000000
50%	23117.0	144.000000	3.000000	0.000000	4.000000
75%	23117.0	144.000000	3.000000	0.000000	4.000000
max	23117.0	65526.000000	65532.000000	65535.000000	64419.000000

Figura 7.2: Estadísticas de los valores de algunas características del conjunto de datos: (a) muestras benignas, (b) muestras maliciosas.

Tal y como se observa en la tabla 7.2, existen características que nos permitirían identificar una muestra como benigna o maliciosa. Se trata de aquellas características que presentan desviación típica cero en las muestras malignas pero no en las benignas (o al revés). Con esta información, cualquier modelo de aprendizaje automático sería capaz de realizar una clasificación perfecta. Para evitar este problema se ha decidido eliminar del conjunto de muestras total, aquellas características cuya desviación típica tenga valor 0, eliminando así 26 de las 76 características iniciales.

Se realiza un segundo análisis con el objetivo de determinar si existen características que estén relacionadas. Si así fuese, podría darse el caso de que una determinada combinación de características permitiese que los modelos de aprendizaje automático realizasen una clasifica-

ción perfecta. Es decir, tendríamos características que no estarían aportando ningún tipo de información a los modelos desarrollados. A partir de las 50 características seleccionadas en el análisis previo, se obtuvo la matriz de correlación entre ellas (figura 7.3):

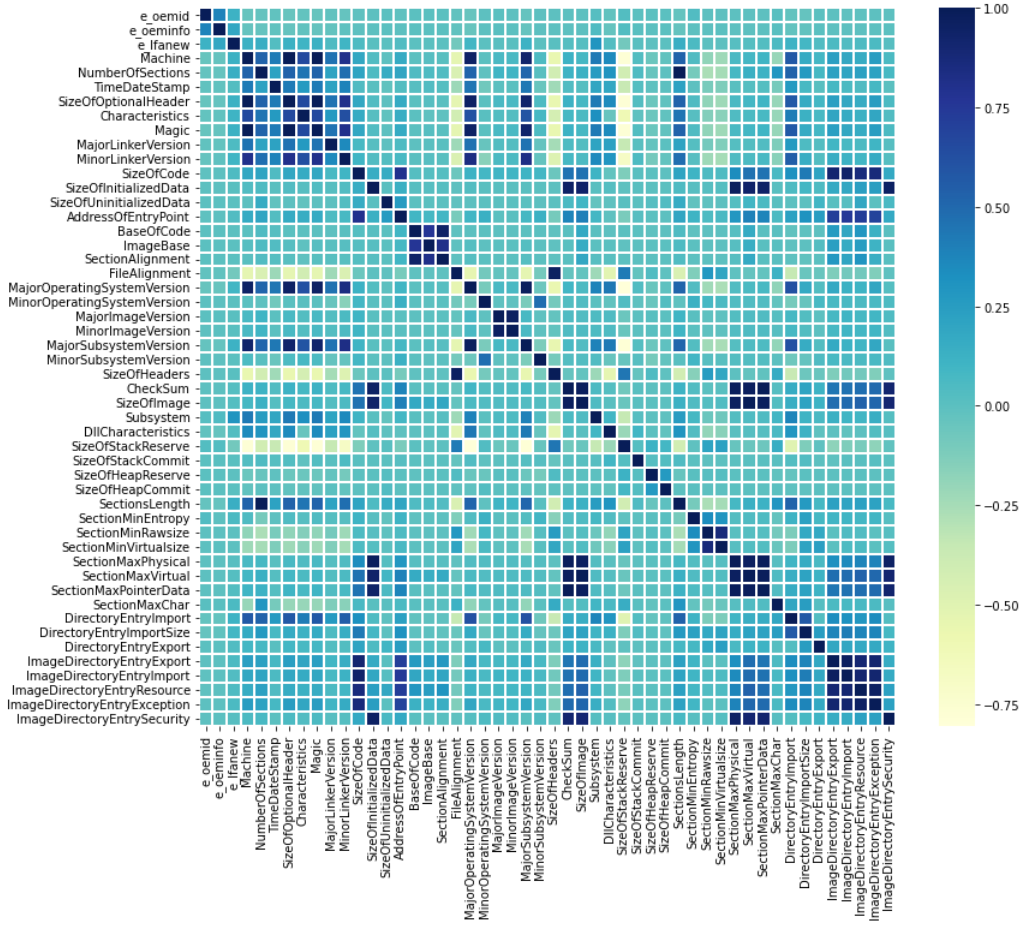

Figura 7.3: Matriz de correlación de las 50 características seleccionadas.

Como podemos observar, existen características correlacionadas, por lo que hemos decidido eliminar aquellas que tengan un índice de correlación igual o superior a 0.8. Como resultado de este segundo descarte, el conjunto final de características está formado por las 28 características siguientes (tabla 7.1):

Características finales	
e_oemid	e_oeminfo
e_lfanew	Machine
NumberOfSections	TimeDateStamp
Characteristics	MajorLinkerVersion
SizeOfCode	SizeOfInitializedData
SizeOfUninitializedData	BaseOfCode
ImageBase	FileAlignment
MinorOperatingSystemVersion	MajorImageVersion
MinorSubsystemVersion	Subsystem
DllCharacteristics	SizeOfStackCommit
SizeOfHeapReserve	SizeOfHeapCommit
SectionMinEntropy	SectionMinRawsize
SectionMaxChar	DirectoryEntryImport
DirectoryEntryImportSize	DirectoryEntryExport

Tabla 7.1: Conjunto de características final.

El último paso que se realiza en este preprocesado de las muestras es la estandarización. El proceso de estandarización es necesario para permitir que los distintos modelos de aprendizaje automático desarrollados trabajen con datos en la misma escala. Esta estandarización consiste en restarle la media a cada valor de la característica y dividir esta resta entre la desviación típica (fórmula 7.1). Esto lo haremos haciendo uso de la clase *StandardScaler* de la librería *Scikit-learn*.

$$X_* = \frac{X - mean(X)}{std(X)} \tag{7.1}$$

7.2 Balanceo del conjunto de datos

Como hemos mencionado en el apartado 5.1.1, nuestro conjunto de datos está desbalanceado, hay muchas más muestras de malware (87061 muestras) que muestras benignas (4114 muestras). Es por ello que se ha decidido aplicar técnicas de sobremuestreo y submuestreo sobre el conjunto de datos y evaluar si existen diferencias entre los resultados obtenidos por los

modelos de aprendizaje automático aplicando estas técnicas y los resultados sobre el conjunto de datos sin balancear. Para ello se han utilizado las clases *SMOTE* [47] (aplica sobremuestreo sobre la clase de datos minoritaria) y *NearMiss* [48] (aplica submuestreo sobre la clase de datos mayoritaria). Ambas clases pertenecen a la librería *Imbalanced-Learn*.

SMOTE (*Synthetic Minority Oversampling Technique*) es una técnica de sobremuestreo utilizada para tratar con conjuntos de datos desbalanceados. SMOTE sintetiza nuevas instancias de la clase minoritaria entre las instancias existentes de dicha clase. Las nuevas instancias no son copias de las instancias existentes, sino que el algoritmo toma muestras del espacio de características para cada instancia objetivo y sus vecinos más cercanos, generando nuevos ejemplos que combinan características de la instancia objetivo con características de sus vecinos.

NearMiss es una técnica de submuestreo que se utiliza para tratar con conjuntos de datos desbalanceados. Su objetivo es equilibrar la distribución de las clases eliminando de forma aleatoria muestras de la clase mayoritaria. Para ello también utiliza la técnica de los K-Vecinos Más Cercanos.

7.3 Modelado

Después de aplicar (o no) el balanceo del conjunto de datos, el conjunto resultante será dividido en dos subconjuntos, uno para entrenar los modelos y el otro para realizar las pruebas. La proporción de dicha división es del 80% para el conjunto de entrenamiento y 20% para el conjunto de test. Esta división la realizaremos a través de la función *train_test_split* de la librería *Scikit-learn*.

Como mencionamos en la sección 6.3 nosotros utilizaremos 5-fold para realizar la validación cruzada. Esta validación cruzada no será sobre el conjunto de datos total, si no que se realizará sobre el conjunto de entrenamiento (80% del total). De tal forma que entrenaremos nuestros modelos y aplicaremos la 5-fold sobre el conjunto de entrenamiento para validar el entrenamiento y seleccionar los mejores modelos, con los que realizaremos la predicción final sobre el conjunto de test (20% del total).

En cuanto a los modelos cabe decir que las pruebas iniciales las hacemos con los parámetros por defecto para cada uno de ellos, excepto para los K-Vecinos Más Cercanos, donde utilizaremos distintos valores de K (1, 3, 5, 7 y 9) para comparar sus resultados (tabla 7.2). Como los resultados de las pruebas con estos valores ya son extremadamente buenos, se ha decidido no probar más configuraciones para ningún modelo.

Modelo	Parámetros
Regresión logística	- **solver**: algoritmo a usar para la optimización del problema (por defecto = 'lbfgs') - **penalty**: especifica la norma de penalización (por defecto = 'l2')
KNN	- **n_neighbors (K)**: número de vecinos a usar (por defecto = 5) - **weights**: función de peso utilizada en la predicción (por defecto = 'uniform')
Bosques Aleatorios	- **n_estimators**: número de árboles en el bosque (por defecto = 100) - **max_depth**: profundidad máxima de cada árbol (por defecto = None) - **max_features**: número de características a tener en cuenta a la hora de buscar la mejor división (por defecto = 'sqrt')
Árbol de decisión	- **max_depth**: profundidad máxima del árbol (por defecto = None) - **max_features**: número de características a tener en cuenta a la hora de buscar la mejor división (por defecto = 'sqrt')
Naive Bayes	Sin parámetros destacables
SVM	- **kernel**: especifica el kernel usado en el algoritmo (por defecto = 'rbf')
MLP	- **hidden_layer_sizes**: número de neuronas por cada capa oculta (por defecto = 100) - **solver**: algoritmo a usar para la optimización del peso (por defecto = 'lbfgs')

Tabla 7.2: Parámetros destacables de los modelos utilizados.

7.4 Resultados

En esta sección presentaremos los resultados de los modelos sin haber aplicado balanceo, aplicando submuestreo y aplicando sobremuestreo.

7.4.1 Resultados sin balanceo

En la tabla 7.3 se presentan los resultados de los distintos modelos de aprendizaje automático desarrollados en términos de las métricas descritas en la sección 6.4, junto con el tiempo de entrenamiento de cada modelo y la precisión media del proceso de validación cruzada sobre el conjunto de entrenamiento:

Modelo	Exactitud	Precisión	Sensibilidad	Medida-F1	Media CV	Tiempo de entrenamiento (s)
Regresión logística	0.9995	**0.9999**	0.9996	0.9997	0.9996	0.2250
KNN con K=1	0.9996	0.9997	0.9998	0.9998	0.9997	0.0070
KNN con K=3	0.9995	0.9997	0.9998	0.9997	0.9996	0.0060
KNN con K=5	0.9996	0.9998	0.9998	0.9998	0.9997	0.0070
KNN con K=7	0.9996	0.9998	0.9998	0.9998	0.9997	0.0060
KNN con K=9	0.9994	0.9997	0.9997	0.9997	0.9997	0.0060
Árbol de decisión	**0.9998**	0.9998	**1.0000**	**0.9999**	0.9998	0.1250
Bosques aleatorios	**0.9998**	0.9998	**1.0000**	**0.9999**	**0.9999**	2.2515
Naive Bayes	0.9977	0.9998	0.9978	0.9988	0.9980	0.0280
SVM	0.9995	0.9997	0.9997	0.9997	0.9996	1.0956
MLP	0.9996	0.9998	0.9998	0.9998	0.9998	3.5804

Tabla 7.3: Resultados de test de los modelos de aprendizaje automático sin balanceo.

Como podemos observar, obtenemos muy buenos resultados con todos los modelos, rozando la perfección en todos ellos. Para poder identificar cual sería el mejor o los mejores modelos, decidimos visualizar los valores obtenidos para los VP, FP, VN, FN:

Modelo	VP	FP	VN	FN
Regresión logística	17394	2	832	7
KNN con K=1	17398	5	829	3
KNN con K=3	17397	5	829	4
KNN con K=5	17397	4	830	4
KNN con K=7	17397	4	830	4
KNN con K=9	17395	5	829	6
Árbol de decisión	17401	3	831	0
Bosques aleatorios	17400	3	831	1
Naive Bayes	17363	4	830	38
SVM	17396	5	829	5
MLP	17397	3	831	4

Tabla 7.4: Resultados en términos de VP, FP, VN y FN sin balanceo.

En nuestro estudio es importante que aquellas muestras benignas se clasifiquen como tal, pero lo realmente crítico es que las muestras de malware sean detectadas correctamente. Por lo tanto, si nos fijamos en la tabla 7.4, podemos observar que con todos los modelos obtenemos una tasa de falsos negativos muy baja salvo para el modelo de Naive Bayes, que presenta la tasa más alta con 38 ficheros potencialmente peligrosos clasificados como ficheros benignos.

7.4.2 Resultados con balanceo: Submuestreo

En la tabla 7.5 se presentan los resultados obtenidos por los distintos modelos tras aplicar la operación de submuestreo sobre la clase mayoritaria. De nuevo, se presentan las medias de la validación cruzada. Al igual que en el caso anterior, la tabla 7.6 presenta los valores obtenidos para los VP, FP, VN, FN:

Modelo	Exactitud	Precisión	Sensibilidad	Medida-F1	Media CV	Tiempo de entrenamiento (s)
Regresión logística	0.9982	0.9988	0.9976	0.9982	0.9992	0.0140
KNN con K=1	0.9982	0.9976	**0.9988**	0.9982	0.9989	0.0020
KNN con K=3	0.9970	0.9964	0.9976	0.9970	0.9982	0.0030
KNN con K=5	0.9970	0.9964	0.9976	0.9970	0.9980	0.0020
KNN con K=7	0.9970	0.9964	0.9976	0.9970	0.9974	0.0030
KNN con K=9	0.9970	0.9964	0.9976	0.9970	0.9971	0.0020
Árbol de decisión	**0.9988**	0.9988	**0.9988**	**0.9888**	**0.9998**	0.0060
Bosques aleatorios	**0.9988**	**1.0000**	0.9976	**0.9988**	0.9996	0.2170
Naive Bayes	0.9964	0.9976	0.9952	0.9964	0.9985	0.0020
SVM	0.9982	**1.0000**	0.9964	0.9982	0.9976	0.0540
MLP	0.9976	0.9976	0.9976	0.9976	0.9985	0.7943

Tabla 7.5: Resultados de test de los modelos de aprendizaje automático con submuestreo.

Modelo	VP	FP	VN	FN
Regresión logística	831	1	812	2
KNN con K=1	832	2	811	1
KNN con K=3	831	3	810	2
KNN con K=5	831	3	810	2
KNN con K=7	831	3	810	2
KNN con K=9	831	3	810	2
Árbol de decisión	832	1	812	1
Bosques aleatorios	831	0	813	2
Naive Bayes	829	2	811	4
SVM	830	0	813	3
MLP	831	2	811	3

Tabla 7.6: Resultados en términos de VP, FP, VN y FN con submuestreo.

De nuevo, si nos fijamos en la tabla 7.5, podemos observar que obtenemos unos resultados excelentes para cada uno de los modelos. Sin embargo en este caso, los valores de la tabla 7.6 no permiten identificar ningún modelo con peor rendimiento en cuanto a tasa de falsos negativos.

7.4.3 Resultados con balanceo: Sobremuestreo

En la tabla 7.7 se presentan los resultados obtenidos por los distintos modelos tras aplicar la operación de sobremuestreo sobre la clase minoritaria. De nuevo, se presentan las medias de la validación cruzada. Al igual que en los casos anteriores, la tabla 7.8 presenta los valores obtenidos para los VP, FP, VN, FN:

Modelo	Exactitud	Precisión	Sensibilidad	Medida-F1	Media CV	Tiempo de entrenamiento (s)
Regresión logística	0.9992	0.9991	0.9994	0.9992	0.9992	0.6570
KNN con K=1	**0.9999**	**1.0000**	**0.9999**	**0.9999**	0.9998	0.0140
KNN con K=3	0.9998	0.9999	0.9997	0.9998	0.9997	0.0140
KNN con K=5	0.9998	0.9999	0.9996	0.9998	0.9996	0.0140
KNN con K=7	0.9997	0.9999	0.9994	0.9997	0.9996	0.0140
KNN con K=9	0.9996	0.9999	0.9993	0.9996	0.9996	0.0150
Árbol de decisión	0.9998	0.9998	0.9998	0.9998	0.9999	0.2680
Bosques aleatorios	**0.9999**	**1.0000**	**0.9999**	**0.9999**	**1.0000**	5.7740
Naive Bayes	0.9969	0.9954	0.9985	0.9969	0.9968	0.0490
SVM	0.9996	0.9995	0.9997	0.9996	0.9996	6.6873
MLP	0.9999	0.9999	0.9998	0.9999	0.9998	7.5092

Tabla 7.7: Resultados de test de los modelos de aprendizaje automático con sobremuestreo.

Modelo	VP	FP	VN	FN
Regresión logística	17450	16	17348	11
KNN con K=1	17457	0	17364	4
KNN con K=3	17456	1	17363	5
KNN con K=5	17454	1	17363	7
KNN con K=7	17451	1	17363	10
KNN con K=9	17449	1	17363	12
Árbol de decisión	17458	3	17361	3
Bosques aleatorios	17459	0	17364	2
Naive Bayes	17435	81	17283	26
SVM	17456	8	17356	5
MLP	17459	0	17364	2

Tabla 7.8: Resultados en términos de VP, FP, VN y FN con sobremuestreo.

Como en los casos anteriores, si nos fijamos en la tabla 7.7 podemos ver que nuevamente obtenemos excelentes resultados. Por lo tanto, si nos fijamos en la tabla 7.8 podemos observar como el modelo Naive Bayes no solo es el que mayor tasa de falsos negativos presenta (26 ficheros peligrosos clasificados como benignos), sino que también es el modelo que presenta la mayor tasa de falsos positivos, con 81 ficheros benignos clasificados como maliciosos.

Capítulo 8

Evaluación

En este capítulo resumiremos y evaluaremos los resultados de los modelos desarrollados para los tres escenarios propuestos, y compararemos los resultados de los tres escenarios con los de alguno de los estudios mencionados el capítulo 4.

8.1 Evaluación de los resultados de los modelos desarrollados

Como pudimos observar en la sección anterior, se obtiene un rendimiento excelente en cualquiera de los modelos desarrollados en el primer escenario propuesto (tabla 7.3), es decir, sin balancear el conjunto de datos. Para poder descartar que sea este desbalanceo el causante de dichos resultados, se decidió probar a balancear el conjunto de datos. Tal y como se ha podido comprobar, los resultados mantienen un rendimiento óptimo en cualquiera de los modelos presentados (tablas 7.5 y 7.7).

Analizando los resultados en detalle, nos fijamos como el modelo Naive Bayes siempre es el que peores resultados obtiene en cada uno de los tres escenarios. Además, este modelo proporciona el valor más alto en cuanto a falsos negativos y falsos positivos, por lo que no se considera un modelo adecuado para la tarea que nos ocupa (tablas 7.4, 7.6 y 7.8).

Si comparamos los resultados entre los tres escenarios, podemos observar que son muy similares entre ellos, rozando la perfección en todos los modelos. No obstante, consideramos que el escenario donde se aplica el submuestreo es el más óptimo para nuestro estudio debido a los cortos tiempos de entrenamiento de todos los modelos y los buenos resultados que obtenemos sin que apenas sea necesario consumir recursos. Es necesario destacar que incluso se obtiene un 100% de precisión para el modelo de Bosques Aleatorios y para el modelo de Máquinas de Vectores de Soporte, siendo este último el que menor tiempo de entrenamiento requiere de los dos. Es cierto que los resultados que hemos obtenido para los escenarios donde aplicamos submuestreo son, en general, ligeramente peores que los de los demás escenarios. Esto se debe, sin embargo, a la poca cantidad de muestras con las que entrenamos los modelos.

Esto podría solucionarse simplemente aumentando el tamaño del conjunto de datos (siempre que sea posible).

8.2 Comparación con otros estudios

En esta sección intentamos comparar el mejor de nuestros resultados en cada unos de los tres escenarios propuestos, con los mejores resultados de alguno de los trabajos mencionados en el capítulo 4, aquellos donde ya obtengan la precisión de los modelos generados o bien aquellos donde podamos calcularla nosotros a partir de otros datos (recordemos la fórmula vista en la sección 6.4).

Debido al uso de distintos conjuntos de datos o de distintas características de los ficheros PE, la comparación no es del todo justa. A pesar de esto, en la tabla 8.1 podemos observar como los mejores modelos de nuestros escenarios superan a los modelos de los estudios citados.

Por lo tanto confirmamos que obtenemos unos resultados excelentes para los tres escenarios propuestos.

Trabajo	Número de muestras totales	Algoritmo	Precisión
Schultz et al. [27]	4266 (3265 malware y 1001 benignas)	Multi Naive Bayes	97,76%
Wang et al. [30]	9771 (7863 malware y 1908 benignas)	SVM	98,1% (Virus) 93,96% (Gusanos) 84,11% (Troyanos) 89,54%(Backdoors)
Liao [32]	6835 (5596 malware y 1237 benignas)	Algoritmo basado en reglas	99,96%
Kumar et al. [33]	± 121000	Árbol de Decisión Bosques Aleatorios	99,7%
Aproximación sin balanceo	91175 (87601 malware y 4114 benignas)	Regresión Logística	99,99%
Aproximación con submuestreo	8228 (4414 malware y 4114 benignas)	Bosques Aleatorios SVM	100%
Aproximación con sobremuestreo	174122 (87601 malware y 86521 benignas)	KNN K=1 Bosques Aleatorios	100%

Tabla 8.1: Comparación entre distintos estudios y las aproximaciones desarrolladas.

Capítulo 9
Conclusiones

9.1 Conclusiones

El gran auge de los ataques de malware en los últimos años, y la necesidad de atajar esta problemática a través de la óptima protección de los sistemas informáticos, ha ejercido de eje rector de la intencionalidad de este estudio. Este aseguramiento de los sistemas se realiza a través de algoritmos de aprendizaje automático, una de las herramientas más contrastadas en el campo de la inteligencia artificial.

En el desarrollo de este trabajo hemos procedido, en consecuencia, a realizar un análisis de la utilización de los algoritmos de aprendizaje supervisado para la tarea de identificación de malware en ficheros PE, con el objetivo de construir modelos de aprendizaje supervisado que contribuyan a la reducción del impacto de los ataques de malware mediante la detección temprana del mismo. Debido al desbalanceo inicial de nuestro conjunto de datos, se ha decidido comparar los resultados de tres escenarios distintos: sin aplicar balanceo al conjunto de datos, aplicando submuestreo y aplicando sobremuestreo. A partir de esto se han definido diversos modelos, utilizando técnicas de aprendizaje automático tales como Regresión Logística, Máquinas de Vectores de Soporte, K-Vecinos Más Cercanos (KNN), el algoritmo Naive Bayes, Árboles de Decisión, Bosques Aleatorios y Modelos de Redes Neuronales. A partir de estos algoritmos de aprendizaje automático, hemos podido desarrollar 11 modelos que han sido validados y evaluados usando diferentes métricas, obteniendo excelentes resultados en cada uno de los escenarios.

Concretamente, de estos resultados deriva un gran éxito de todos los modelos, superando en todos ellos un nivel de precisión del 99% y llegando incluso a alcanzar el 100% en los modelos de Bosques Aleatorios y Máquinas de Vectores de Soporte, para el escenario donde aplicamos submuestreo.

Podemos concluir confirmando la eficiencia de los modelos para la tarea de detección de malware en ficheros PE.

9.2 Conocimientos adquiridos

Los conocimientos adquiridos en la realización de este trabajo han sido:

- Conocimientos teóricos sobre el análisis de malware y los ficheros PE.
- Conocimiento teórico y desarrollo de modelos de Aprendizaje Supervisado.
- Manejo de librerías de alto nivel para aprendizaje automático en Python.
- Técnicas de selección de características y su aplicación al aprendizaje automático.

9.3 Trabajo futuro

En cuanto al trabajo futuro, se abren ciertas líneas de expansión en relación con los resultados obtenidos en el desarrollo:

- Minimizar el número de características del conjunto de datos en busca de minimizar los tiempos de entrenamiento y ejecución sin que afecte notablemente a los resultados.
- Realizar transferencia de aprendizaje utilizando los modelos generados sobre nuevos conjuntos de datos.
- Añadir al análisis de resultados algoritmos de *Deep Learning*

Por último, cabe decir que, pese a que en el anteproyecto de nuestro trabajo mencionamos la posibilidad de realizar también un análisis dinámico sobre los ficheros de nuestro conjunto de datos, esto ha sido descartado ya que, como hemos podido apreciar, obtenemos muy buenos resultados realizando simplemente análisis estático. Sin embargo, también podríamos considerar esta opción como una vía de trabajo futuro.

Apéndices

Apéndice A

Descripción de las características extraídas

En este apéndice se incluye una tabla con la descripción de las características extraídas para nuestro conjunto de datos (tabla A.1):

Característica	**Descripción**
e_magic	Número mágico
e_cblp	Bytes de la última página del fichero
e_cp	Páginas en el fichero
e_crlc	Redirecciones
e_cparhdr	Tamaño del encabezado en bloques
e_minalloc	Mínimo de bloques adicionales necesarios
e_maxalloc	Máximo de bloques adicionales necesarios
e_ss	Valor inicial SS
e_sp	Valor inicial SP
e_csum	Checksum de la cabecera MS-DOS
e_ip	Valor inicial IP
e_cs	Valor inicial CS

e_lfarlc	Dirección de archivo de la tabla de redirecciones
e_ovno	Número de superposición
e_oemid	Identificador OEM
e_oeminfo	Información OEM
e_lfanew	Dirección de archivo de la siguiente cabecera (normalmente la cabera PE)
Machine	Tipo de máquina de destino
NumberOfSections	Número de secciones en el fichero
TimeDateStamp	Fecha cuando el fichero fue creado
PointerToSymbolTable	Desplazamiento del archivo de la tabla de símbolos COFF (debe ser 0)
NumberOfSymbols	Especifica el número de entradas en la tabla de símbolos (debe ser 0)
SizeOfOptionalHeader	Tamaño de la cabecera opcional
Characteristics	Flag que indica las características del fichero
Magic	Entero sin signo que indica el tipo de archivo ejecutable
MajorLinkerVersion	Número de versión principal del linker
MinorLinkerVersion	Número de versión secundario del linker
SizeOfCode	Tamaño de la sección de código
SizeOfInitializedData	Tamaño de la sección de datos inicializados
SizeOfUninitializedData	Tamaño de la sección de datos no inicializados

AddressOfEntryPoint	Dirección del punto de entrada relativa a la imagen base cuando el ejecutable se carga en la memoria.
BaseOfCode	Dirección relativa a la imagen base de la sección de comienzo de código cuando se carga en la memoria
ImageBase	Dirección preferida del primer byte de la imagen cuando se carga en la memoria
SectionAlignment	Alineación (en bytes) de las secciones cuando se cargan en la memoria
FileAlignment	Factor (en bytes) que se utiliza para alinear los datos sin procesar de las secciones en el archivo de imagen
MajorOperatingSystemVersion	Número de versión principal del sistema operativo necesario
MinorOperatingSystemVersion	Número de versión secundario del sistema operativo necesario
MajorImageVersion	Número de versión principal de la imagen
MinorImageVersion	Número de versión secundario de la imagen
MajorSubsystemVersion	Número de versión principal del subsistema
MinorSubsystemVersion	Número de versión secundario del subsistema
SizeOfHeaders	Especifica el tamaño combinado de la cabecera MS-DOS, cabecera PE y cabeceras de sección redondeado a un múltiplo del FileAlignment
CheckSum	Checksum del fichero PE
SizeOfImage	Tamaño de la imagen

Subsystem	Subsistema necesario para ejecutar la imagen
DllCharacteristics	Flags que caracterizan al fichero PE
SizeOfStackReserve	Tamaño de la pila a reservar
SizeOfStackCommit	Tamaño de la pila a confirmar
SizeOfHeapReserve	Tamaño del montículo a reservar
SizeOfHeapCommit	Tamaño del montículo a confirmar
LoaderFlags	Reservado, debe ser cero.
NumberOfRvaAndSizes	Número de entradas del directorio de datos en el resto del encabezado opcional
SuspiciousNameSection	Número de nombres sospechosos de sección
SectionsLength	Longitud de las secciones
SectionMinEntropy	Valor mínimo de la entropía de las secciones
SectionMaxEntropy	Valor máximo de la entropía de las secciones
SectionMinRawsize	Valor mínimo del tamaño de las secciones
SectionMaxRawsize	Valor máximo del tamaño de las secciones
SectionMinVirtualsize	Valor mínimo del tamaño virtual de las secciones
SectionMaxVirtualsize	Valor máximo del tamaño virtual de las secciones
SectionMaxPhysical	Valor máximo de las direcciones físicas de las secciones
SectionMinPhysical	Valor mínimo de las direcciones físicas de las secciones
SectionMaxVirtual	Valor máximo de las direcciones virtuales de las secciones

SectionMinVirtual	Valor mínimo de las direcciones virtuales de las secciones
SectionMaxPointerData	Valor máximo del puntero a la primera página de las secciones
SectionMinPointerData	Valor mínimo del puntero a la primera página de las secciones
SectionMaxChar	Valor máximo de las características de las secciones
SectionMainChar	Valor mínimo de las características de las secciones
DirectoryEntryImport	Número de DLLs importadas
DirectoryEntryImportSize	Tamaño de las DLLs importadas
DirectoryEntryExport	Número de símbolos exportados
ImageDirectoryEntryExport	Dirección virtual del directorio de exportaciones
ImageDirectoryEntryImport	Dirección virtual del directorio de importaciones
ImageDirectoryEntryResource	Dirección virtual del directorio de recursos
ImageDirectoryEntryException	Dirección virtual del directorio de excepciones
ImageDirectoryEntrySecurity	Dirección virtual del directorio de seguridad

Tabla A.1: Descripción de las características extraídas.

Lista de acrónimos

ASCII American Standard Code for Information Interchange.

AUC Area Under the Curve.

COFF Common Objet File Format.

CSV Comma-Separated Values.

CV Cross Validation.

DDoS Distributed Denial of Service.

DLL Dynamic-link library.

DoS Denial of Service.

FN Falsos Negativos.

FP Falsos Positivos.

HWT Haar Wavelet Transformation.

IP Internet Protocol.

KNN K-Nearest Neighbors.

MLP Multi Layer Perceptron.

PCA Principal Component Analysis.

PE Portable Executable.

RFR Redundant Feature Removal.

RNA Red de Neuronas Artificiales.

SMOTE Synthetic Minority Oversampling Technique.

SVM Support Vector Machines.

VN Verdaderos Negativos.

VP Verdaderos Positivos.

Bibliografía

[1] R. Tahir, "A study on malware and malware detection techniques," 2018. [En línea]. Disponible en: https://www.mecs-press.org/ijeme/ijeme-v8-n2/IJEME-V8-N2-3.pdf

[2] "¿Qué son los ataques DoS y DDoS?" 2018. [En línea]. Disponible en: https://www.osi.es/es/actualidad/blog/2018/08/21/que-son-los-ataques-dos-y-ddos

[3] S. Gadhiya and K. Bhavsar, "Techniques for malware analysis," 2013. [En línea]. Disponible en: https://web.archive.org/web/20160418151823/http://www.ijarcsse.com/docs/papers/Volume_3/4_April2013/V3I4-0371.pdf

[4] "Virustotal." [En línea]. Disponible en: https://www.virustotal.com

[5] "Chronicle." [En línea]. Disponible en: https://chronicle.security/

[6] "Sysinternals." [En línea]. Disponible en: https://docs.microsoft.com/en-us/sysinternals/downloads/sysinternals-suite

[7] "PEiD." [En línea]. Disponible en: https://www.aldeid.com/wiki/PEiD

[8] "IDA Pro." [En línea]. Disponible en: https://hex-rays.com/ida-pro/

[9] "Ollydbg." [En línea]. Disponible en: https://www.ollydbg.de/

[10] "Radare2." [En línea]. Disponible en: https://rada.re/n/

[11] "Sysinspector." [En línea]. Disponible en: https://www.eset.com/es/soporte/sysinspector

[12] "Wireshark." [En línea]. Disponible en: https://www.wireshark.org/

[13] "El formato PE (Portable Executable) Parte I." [En línea]. Disponible en: http://www.redinskala.com/2013/10/09/formato-pe-parte-i/

[14] A. L. Samuel, "Some Studies in Machine Learning Using the Game of Checkers." [En línea]. Disponible en: https://people.csail.mit.edu/brooks/idocs/Samuel.pdf

[15] J. Hurwitz and D. Kirsch, *Machine Learning for dummies.* IBM, 2018.

[16] "Qué es overfitting y underfitting y cómo solucionarlo." [En línea]. Disponible en: https://www.aprendemachinelearning.com/que-es-overfitting-y-underfitting-y-como-solucionarlo/

[17] O. Theobald, *Machine Learning For Absolute Beginners.*

[18] "Support Vector Machine (SVM)." [En línea]. Disponible en: https://es.mathworks.com/discovery/support-vector-machine.html

[19] D. C. E. Saputra, A. Azharir, and A. Ma'arif, "K-Nearest Neighbor of Beta Signal Brainwave to Accelerate Detection of Concentration on Student Learning Outcomes," 2022. [En línea]. Disponible en: https://web.p.ebscohost.com/ehost/pdfviewer/pdfviewer?vid=1&sid=254a6549-b360-4644-9c8b-75f1eb75b9c9%40redis

[20] "Bayes' Theorem," 2003. [En línea]. Disponible en: http://seop.illc.uva.nl/entries/bayes-theorem/

[21] "Algoritmos Naive Bayes: Fundamentos e Implementación." [En línea]. Disponible en: https://medium.com/datos-y-ciencia/algoritmos-naive-bayes-fudamentos-e-implementaci%C3%B3n-4bcb24b307f

[22] B. Salehe, "CIT365: Data Mining & Data Warehousing," 2018. [En línea]. Disponible en: https://documen.site/download/hunts-algorithm-the-institute-of-finance-management_pdf

[23] J. M. Heras, "Ensembles: voting, bagging, boosting, stacking," 2019. [En línea]. Disponible en: https://www.iartificial.net/ensembles-voting-bagging-boosting-stacking/

[24] R. Wang, "AdaBoost for Feature Selection, Classification and Its Relation with SVM*, A Review," 2012. [En línea]. Disponible en: https://www.sciencedirect.com/science/article/pii/S1875389212005767

[25] J. Torres, "Learning Process of a Deep Neural Network," 2020. [En línea]. Disponible en: https://towardsdatascience.com/learning-process-of-a-deep-neural-network-5a9768d7a651

[26] "Neural Network Models Explained," 2022. [En línea]. Disponible en: https://www.seldon.io/neural-network-models-explained

[27] M. G. Schultz, E. Eskin, E. Zadok, and S. J. Stolfo, "Data mining methods for detection of new malicious executables," 2001. [En línea]. Disponible en: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=924286

[28] M. Britsch, N. Gagunashvili, and M. Schmelling, "Application of the rule-growing algorithm RIPPER to particle physics analysis," 2001. [En línea]. Disponible en: https://indico.cern.ch/event/34666/contributions/813578/attachments/683858/939357/talkBritschAcat2008.pdf.pdf

[29] M. Z. Shafiq, S. M. Tabish, F. Mirza, and M. Farooq, "PE-Miner: Mining Structural Information to Detect Malicious Executables in Realtime," 2009. [En línea]. Disponible en: https://www.researchgate.net/publication/221427459_PE-Miner_Mining_Structural_Information_to_Detect_Malicious_Executables_in_Realtime

[30] T.-Y. Wang, C.-H. Wu, and C.-C. Hsieh, "Detecting Unknown Malicious Executables Using Portable Executable Headers," 2009. [En línea]. Disponible en: https://ieeexplore.ieee.org/document/5331712

[31] "Referencia de DUMPBIN." [En línea]. Disponible en: https://docs.microsoft.com/es-es/cpp/build/reference/dumpbin-reference?view=msvc-170

[32] Y. Liao, "PE-Header-Based Malware Study and Detection," 2012. [En línea]. Disponible en: https://cobweb.cs.uga.edu/~liao/PE_Final_Report.pdf

[33] A. Kumar, K. Abhishek, K. Shah, D. Patel, Y. Jain, H. Chheda, and P. Nerurkar, "Malware Detection Using Machine Learning," 2020.

[34] "Python." [En línea]. Disponible en: https://www.python.org/

[35] "PYPL PopularitY of Programming Language." [En línea]. Disponible en: https://pypl.github.io/PYPL.html

[36] "Visual Studio Code." [En línea]. Disponible en: https://code.visualstudio.com/

[37] S. M. González, "Repositorio Github del proyecto." [En línea]. Disponible en: https://github.com/smourog/TFG

[38] "Pandas." [En línea]. Disponible en: https://pandas.pydata.org/

[39] "Scikit-learn." [En línea]. Disponible en: https://scikit-learn.org/stable

[40] "Seaborn." [En línea]. Disponible en: https://seaborn.pydata.org/

[41] "Matplotlib." [En línea]. Disponible en: https://matplotlib.org/

[42] "Imbalanced-Learn." [En línea]. Disponible en: https://imbalanced-learn.org/stable

[43] "Virusshare." [En línea]. Disponible en: https://virusshare.com/

[44] "Repositorio de muestras benignas DikeDataset." [En línea]. Disponible en: https://github.com/iosifache/DikeDataset/tree/main/files/benign

[45] J. Sutherland, *SCRUM: El revolucionario método para trabajar el doble en la mitad de tiempo*, 2014.

[46] "Cross-Validation : definición e importancia en machine learning." [En línea]. Disponible en: https://datascientest.com/es/cross-validation-definicion-e-importancia

[47] "SMOTE Imbalanced-learn." [En línea]. Disponible en: https://imbalanced-learn.org/stable/references/generated/imblearn.over_sampling.SMOTE.html

[48] "NearMiss Imbalanced-learn." [En línea]. Disponible en: https://imbalanced-learn.org/stable/references/generated/imblearn.under_sampling.NearMiss.html