Universitat Rovira i Virgili (URV) y Universitat Oberta de Catalunya (UOC)

Máster Universitario en Ingeniería Computacional y Matemática

TRABAJO FINAL DE MÁSTER

Área: Ciberseguridad

Métodos de aprendizaje automático aplicados a la ciberseguridad: Clasificación binaria de ejecutables del Sistema Operativo Windows.

Autor: Luis Alberto Glaría Silva

Tutor: Ángel Elbaz Sanz

Profesor: Josep Prieto Blázquez

---

7 de junio de 2023

El Dr. **Ángel Elbaz Sanz**, certifica que el estudiante **Luis Alberto Glaría Silva** ha elaborado el trabajo bajo su tutoría y autoriza la presentación de esta memoria para su evaluación.

Firma del tutor:

FICHA DEL TRABAJO FINAL

Título del trabajo:	Métodos de aprendizaje automático aplicados a la ciberseguridad: Clasificación binaria de ejecutables del Sistema Operativo Windows.
Nombre del autor:	Luis Alberto Glaría Silva
Nombre del del consultor/a:	Ángel Elbaz Sanz
Nombre del PRA:	Josep Prieto Blázquez
Fecha de entrega (mm/aaaa):	06/2023
Titulación o programa:	Máster Universitario en Ingeniería Computacional y Matemática
Área del Trabajo Final:	Ciberseguridad
Idioma del trabajo:	Español
Palabras clave	Ciberseguridad, malware, aprendizaje automático, formato PE, Python, Streamlit
Keywords	Cybersecurity, malware, machine learning, PE format, Python, Streamlit

Resumen

La digitalización de nuestra sociedad ha aumentado el impacto de las amenazas a la seguridad informática, haciendo que las soluciones clásicas en materia de detección de malware se vuelvan insuficientes. El aprendizaje automático aporta a la ciberseguridad la flexibilidad necesaria para enfrentar este entorno cambiante y cada vez más complejo.

En este trabajo se ha hecho un estudio del estado del arte de la intersección entre estas dos disciplinas, evaluando diferentes métodos de aprendizaje automático y sus aplicaciones.

Para poner en práctica estas ideas, se ha construido un conjunto de datos que ha servido para entrenar diversos algoritmos en la tarea de la clasificación binaria de archivos ejecutables de Windows. Utilizando las técnicas con el mejor desempeño se ha creado un modelo optimizado con el que se han alcanzado valores de precisión, exactitud y sensibilidad superiores al 97 % en el conjunto de prueba.

Este modelo se ha desplegado en una aplicación web, lo que permite tanto la clasificación de archivos ejecutables individuales como de conjuntos de datos formados por características ya extraídas.

Palabras clave: Ciberseguridad, malware, aprendizaje automático, formato PE, Python, Streamlit

Abstract

The digitalization of our society has increased the impact of threats to computer security, making traditional solutions for malware detection insufficient. Machine learning provides cybersecurity with the necessary flexibility to confront this changing and increasingly complex environment.

In this work, a study of the state of the art of the intersection between these two disciplines has been conducted, evaluating different machine learning methods and their applications.

To put these ideas into practice, we built a dataset that has been used to train multiple algorithms for the task of binary classification of Windows executable files. Using the algorithms with the best performance, an optimized model was created, achieving precision, accuracy, and recall values greater than 97 % in the test set.

This model has been deployed on a web application which allows both the classification of individual executable files and datasets composed of already extracted features.

Keywords: Cybersecurity, malware, machine learning, PE format, Python, Streamlit

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

1.1. Relevancia del problema y descripción general

La continua y creciente digitalización de todos los ámbitos de la sociedad actual, con el desarrollo de la Internet de las Cosas (*Internet-of-Things*) y la migración de archivos y sistemas analógicos a formatos digitales, ha hecho no solo que el número de incidentes vinculados a la seguridad informática aumente de forma exponencial sino que el impacto de tales incidentes se ha visto igualmente incrementado [61].

Según la AV-TEST, una institución alemana que se ocupa de evaluar software antivirus, se estima que en en el año 2022 el número de aplicaciones maliciosas (malware) superó los 1000 millones de los cuales casi 100 millones fueron creados durante el propio 2022 [44].

Las amenazas a la seguridad informática tienen igualmente un impacto económico elevado: IBM estima en 4,35 millones de dólares el coste medio a nivel mundial de una filtración de datos, elevándose este valor a los 9,44 millones en el caso de los Estados Unidos [56]. Pero no hablamos solo de daños económicos, los ciberataques pueden llegar a afectar o paralizar infraestructuras críticas, como en el caso de los ataques de ransomware al Servicio Público de Salud de Reino Unido (NHS) en 2017 [47], a los servicios digitales de la ciudad belga de Amberes en diciembre de 2022 [75] o el más reciente ataque al Hospital Clínic de Barcelona en marzo de 2023 [55], en todos estos casos el correcto funcionamiento de hospitales u otros servicios públicos se vio alterado o llegó a quedar paralizado de forma temporal.

Se hace necesario por tanto, implementar medidas, protocolos o procesos que se encarguen de proteger redes, dispositivos o datos del acceso o uso no autorizados, así como evitar la interrupción de servicios informáticos. De todo esto se encarga la ciberseguridad, es decir, es la responsable de enfrentar cualquier amenaza a la seguridad informática.

De entre las amenazas a los que se enfrenta la ciberseguridad destaca el malware, término donde se engloban una gran variedad de diferentes tipos de software malicioso como ransomwa-

re, gusanos, troyanos o spyware. La naturaleza intrusiva y potencialmente destructiva de estos programas, hace que no se limiten a afectar la disponibilidad o integridad de los sistemas, sino que también puedan llegar a comprometer datos confidenciales. Por todo esto, en el presente trabajo nos centraremos en el malware, contra el cual se han aplicado históricamente distintas técnicas de detección y prevención. Sin embargo, aunque estas técnicas clásicas, que analizaremos con más detalle, han demostrado ser efectivas en ciertos contextos, presentan limitaciones importantes en un entorno de amenazas cada vez más complejas y en constante evolución. Por ello, se hace necesario introducir nuevas técnicas, y es en este punto donde entran en juego los avances en el área del aprendizaje automático.

El aprendizaje automático constituye una rama dentro del campo de la inteligencia artificial que se centra en el desarrollo de algoritmos y modelos estadísticos que aprenden de los datos sin haber sido programados explícitamente. Estos métodos pueden ser utilizados tanto para identificar patrones, predecir nuevos valores o clasificar nuevos elementos.

Este trabajo se centrará en el punto de contacto entre estas dos tecnologías, es decir, tratará de como el aprendizaje automático puede utilizarse en la detección de malware, dentro del ámbito de la ciberseguridad.

1.2. Objetivos

A continuación se listan los objetivos generales de este Trabajo de Fin de Máster, así como los específicos, que constituyen apartados de los generales. Se mencionan igualmente ciertos objetivos opcionales que se encuentran al mismo nivel que los específicos y aparecen debidamente señalados. Aunque no son imprescindibles para el cumplimiento de los generales, los objetivos opcionales constituyen áreas en las que resultaría interesante investigar y trabajar y que serán desarrollados si se dispone del tiempo suficiente.

Todos estos objetivos, establecidos al redactar el Plan de Trabajo, han sido actualizados para tener en cuenta los cambios realizados en etapas posteriores.

1. Análisis de técnicas de aprendizaje automático

    a) Listado de algoritmos de aprendizaje automático (supervisado y no supervisado) con aplicaciones a la ciberseguridad.

    b) Desarrollo de los fundamentos de los algoritmos descritos, así como disntintas formas de clasificarlos.

    c) *(Opcional)* Introducir técnicas de *deep learning*.

2. Aplicaciones del aprendizaje automático a la ciberseguridad.

   a) Explicar la relevancia y características de la ciberseguridad. Listar las principales categorías en las que se pueden agrupar las amenazas a las que se enfrenta la ciberseguridad

   b) Introducir y describir técnicas clásicas de detección de malware y sus limitaciones

   c) Fundamentar como los algoritmos de aprendizaje automático mejoran y complementan las técnicas anteriores. Describir ejemplos concretos.

3. Implementación de algoritmos de aprendizaje automático aplicados a la detección de malware.

   a) Identificar bases de datos disponibles y crear un conjunto de datos para el entrenamiento de los algoritmos

   b) Elegir implementaciones existentes de algoritmos de aprendizaje automático (por ejemplo en librerías de Python) y adaptarlas al contexto de la clasificación de malware.

   c) Realizar un estudio comparativo del desempeño de los disntintos algoritmos analizados

   d) Integrar tanto la extracción de datos como el entrenamiento y validación del modelo en un proyecto único cuyo código pueda ser exportado.

   e) *(Opcional)* Desarrollar una aplicación donde se implemente el modelo exportado y permita clasificar archivos como maliciosos o benignos.

1.3. Enfoque y método seguido

Por las características de este Trabajo, donde se han combinado tanto los conceptos teóricos de ciertos algoritmos de aprendizaje automático con cuestiones prácticas de la detección de malware, se hizo necesario aplicar un enfoque capaz de unir ambos aspectos (el teórico y el práctico) de forma efectiva.

De entre las diferentes metodologías existentes en la industria la que mejor se adaptaba a este enfoque era la CRISP-DM (*Cross-Industry Standard Process for Data Mining*). Donde en sus dos primeras fases se requiere de un proceso de análisis del negocio (o del problema en cuestión en nuestro caso) y de entender debidamente los datos con los que se trabaja. Estas dos primeras fases (sobre todo la primera) estuvieron estrechamente relacionadas con la parte teórica de este trabajo. Las siguientes etapas del método CRISP-DM: Preparación de los datos, Desarrollo del Modelo y Evaluación se correspondieron con el apartado práctico de este TFM, donde implementamos, entrenamos y validamos algoritmos de aprendizaje automático aplicados

a la ciberseguridad. Por último, la fase de puesta en producción, última etapa de la metodología CRISP-DM, se correspondió con la creación de la aplicación web donde desplegamos el modelo entrenado.

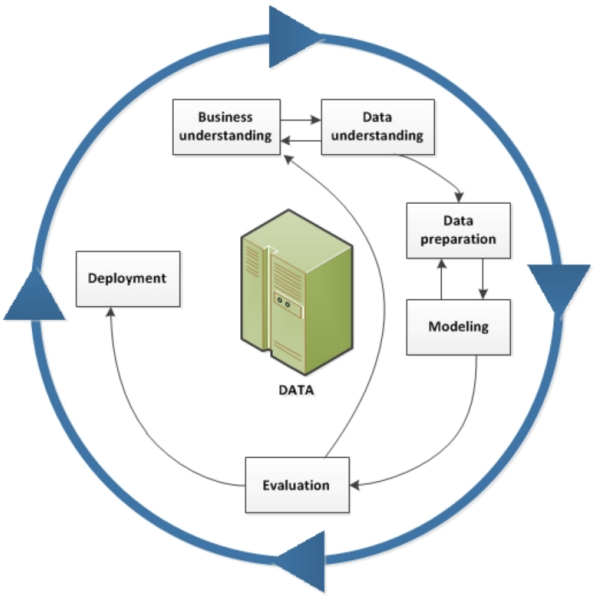


Figura 1.1: Diagrama con las diferentes etapas de la metodología CRISP-DM [13]

Estas seis etapas en su conjunto, cuya relación se puede apreciar en la figura 1.1, permitieron estructurar los procedimientos seguidos en este trabajo de forma que en cada momento fue completamente clara la tarea a realizar, lo que contribuyó a optimizar el tiempo. Igualmente, al tratarse de una metodología flexibe, fue posible volver a puntos anteriores del esquema, en caso que los objetivos obtenidos no fueran los esperados.

1.4. Planificación y cronograma

Se listan a continuación las tareas realizadas durante este trabajo. Cada tarea incluyó implícitamente la redacción de la parte de la memoria del trabajo correspondiente, por lo que la propia redacción de la memoria no se encuentra señalada como una tarea. El cronograma se puede consultar en la figura 1.2

1. Estado del arte: búsqueda y estudio de trabajos similares y artículos de investigación.
2. Estudio de algoritmos de aprendizaje automático y sus fundamentos teóricos.
3. Estudio del estado actual de la ciberseguridad. Principales amenazas.

4. Investigación y desarrollo de las aplicaciones de la inteligencia articial y el aprendizaje automático a la detección de malware.
5. Búsqueda de bases de datos disponibles con software malicioso y benigno, debidamente etiquetado.
6. Desarrollo de un programa para la extracción de características de archivos ejecutables de Windows.
7. Limpieza, preparación y análisis de los datos. Creación del dataset a utilizar en el trabajo.
8. Investigación y análisis de diferentes implementaciones de algoritmos de aprendizaje automático con aplicaciones a la detección de malware.
9. Elección de un algoritmos y desarrollo de las implementaciones.
10. Evaluación del desempeño de las implementaciones.
11. Creación de un modelo final usando los mejores algoritmos.
12. Realización de un estudio comparativo y redacción de conclusiones a partir del mismo.
13. Creación y despliegue de la aplicación web con el modelo final y los métodos de extracción de características.
14. Revisión de la Memoria del trabajo.
15. Creación de la presentación.

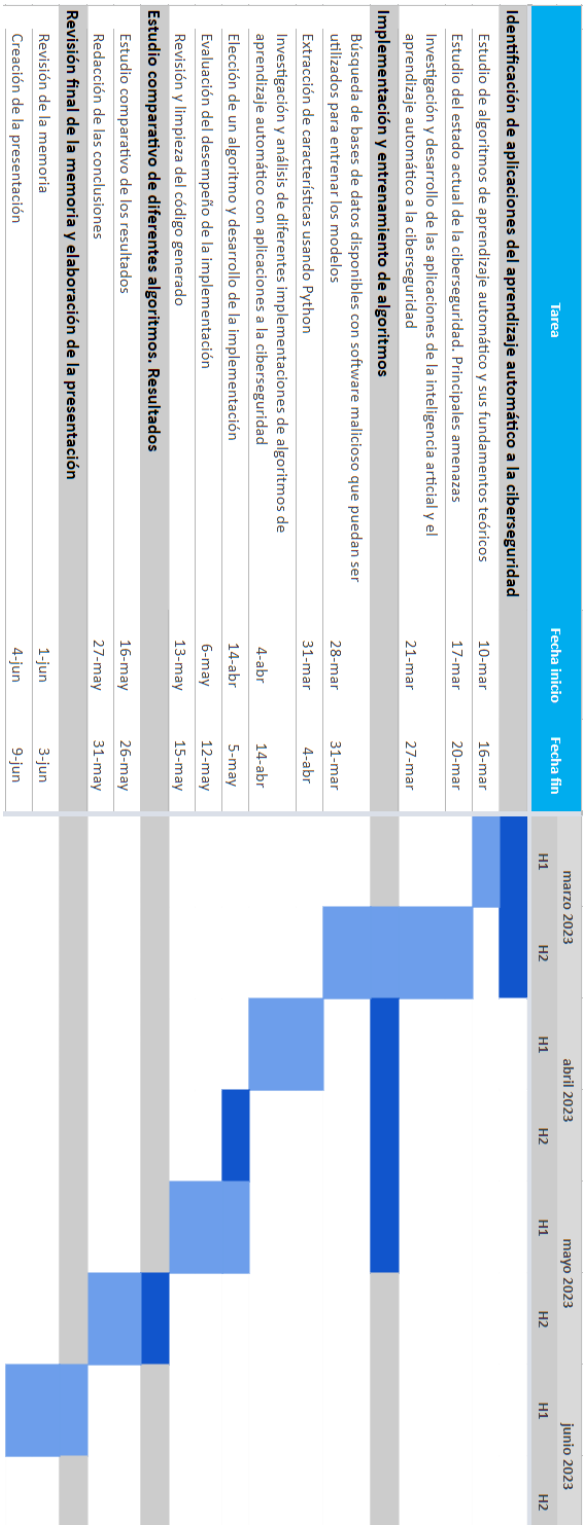


Figura 1.2: Diagrama de Gantt con el calendario del TFM

1.5. Resumen de productos obtenidos

Todos los productos obtenidos como resultado de este trabajo se encuentran disponibles en el repositorio del TFM: https://github.com/glaria/TFM/. Su ubicación exacta dentro del repositorio, así como una breve descripción pueden ser consultadas en la lista a continuación:

1. Cuadernos de Jupyter. Estos seis cuadernos, desarrollados en Google Colab, contienen las distintas pruebas y experimentos realizados durante este trabajo. Los cuadernos están disponibles en Colab Notebooks/

2. Modelo de aprendizaje automático entrenado en la detección de malware. Los dos algoritmos de los que consta este modelo pueden ser obtenidos a partir de los cuadernos de Jupyter o como objetos de Python exportados a archivos en: models/

3. Conjunto de datos etiquetado y balanceado con características de archivos benignos y maliciosos. Este dataset, llamado *balanced_df.csv* y disponible en: datasets/ puede resultar útil en el entrenamiento de nuevos modelos. Un aspecto importante es que dispone del mismo número de clases malware y benignas, al contrario que la mayoría de conjuntos de datos disponibles en la red, que se encuentran desbalanceados en favor de la clase malware.

4. Aplicación web. Esta aplicación, desarrollada y desplegada usando Streamlit, permite hacer pruebas sobre modelos entrenados de forma sencilla. La aplicación está disponible en Streamlit: https://glaria-tfm-malware-classifier-app-7qnhnt.streamlit.app/ y su código fuente se puede consultar en el archivo *malware_classifier_app.py* de la carpeta principal del repositorio.

Capítulo 2

Marco teórico

2.1. Ciberseguridad

De acuerdo con la *Agencia de Ciberseguridad y Seguridad de las Infraestructuras* (CISA por sus siglas en inglés: *Cybersecurity and Infrastructure Security Agency*), la ciberseguridad es el arte de proteger redes, dispositivos y datos de accesos no autorizados o usos delictivos y la práctica de garantizar la confidencialidad, integridad y disponibilidad de la información [20]. De esta definición se desprenden tres puntos u objetivos clave que buscamos garantizar en cualquier sistema informático y que definen su fiabilidad, es decir, la probabilidad de que se comporte tal y como se espera de él. Estos objetivos, denominados triada CIA (Confidenciality. Integrity and Availability), se definen a continuación [72]:

- **Confidencialidad**: Proteger la información sensible de accesos no autorizados y garantizar que sólo sea accesible para aquellos que están autorizados a verla.
- **Integridad**: Mantener la exactitud y coherencia de los datos a lo largo del tiempo y garantizar que no han sido alterados ni dañados de ninguna manera.
- **Disponibilidad**: Garantizar que los usuarios autorizados tengan acceso a la información y a los sistemas que necesitan en todo momento.

Con el fin de garantizar los tres estándar anteriores pueden tomarse diversas medidas como clasificar y etiquetar los datos, aplicar políticas de control de acceso, cifrar la información y utilizar sistemas de autenticación multifactor (MFA).También es necesario implementar métodos de no-repudio como firmas digitales que permitan verificar la integridad de la información e impidan el repudio o la negación de determinada acción. Por último, pueden utilizarse redes, servidores y aplicaciones redundantes que aseguren la continuidad de las operaciones en caso de interrupción o fallo del sistema principal, así como actualizaciones periódicas del software y

de los sistemas de seguridad que permiten mejorar la disponibilidad al reducir la probabilidad de mal funcionamiento de las aplicaciones o de infiltración de nuevas amenazas. Los planes de recuperación en caso de catástrofe y las copias de seguridad periódicas también pueden ayudar en la recuperación de la disponibilidad rápidamente tras un acontecimiento negativo.

Estas medidas buscan anular o mitigar el impacto de las amenazas a las ciberseguridad que veremos en la siguiente sección.

2.1.1. Principales amenazas a la ciberseguridad y métodos clásicos de protección

Las amenazas a la ciberseguridad hacen referencia a cualquier actividad malintencionada que busque dañar, interrumpir o acceder ilegalmente a sistemas informáticos, redes o datos. En el contexto de la triada CIA introducida anteriormente estas amenazas pueden comprometer la seguridad de la información en diversas formas.

La confidencialidad, por ejemplo, puede verse amenazada cuando actores malintencionados acceden de forma no autorizada a datos privados. La integridad se ve comprometida cuando se alteran información, ya sea mediante la manipulación, introducción de datos falsos o eliminación de información relevante. Y finalmente, la disponibilidad se ve afectada cuando se impide el acceso a servicios o datos.

De acuerdo con el reporte [27] elaborado en 2022 por la Agencia Europea de Ciberseguridad (ENISA por sus siglas en inglés) [26] podemos clasificar las principales amenazas a la seguridad informática en 5 grupos fundamentales:

1. **Amenazas de Ingeniería Social**: Esta categoría de amenazas cibernéticas se basa en explotar las vulnerabilidades inherentes al ser humano. En lugar de atacar directamente la infraestructura de un sistema informático, los ciber-delincuentes que usan estas técnicas apuntan a los usuarios de estos sistemas, explotando errores humanos para obtener acceso no autorizado a información confidencial o servicios importantes.

   El término *ingeniería social* hace referencia a una serie de estrategias y tácticas que manipulan la confianza de las personas y consiguen persuadir a los individuos para que realicen determinadas acciones que en situaciones normales no harían, como revelar información sensible o crear un acceso no seguro. La manipulación se lleva a cabo a menudo a través de la suplantación de identidad, presentándose los atacantes como figuras de confianza o de autoridad.

   Entre las técnicas más comunes de ingeniería social, encontramos el phishing y el smishing. El phishing generalmente se realiza a través de correo electrónico, donde los atacantes

envían un mensaje que parece ser de una entidad conocida, como un banco o una empresa de confianza. Este mensaje suele incuir un enlace que lleva a las víctimas a un sitio web fraudulento, donde se les pide que ingresen información confidencial (nombres de usuario, contraseñas, datos bancarios,..).

El smishing, por otro lado, es una forma de phishing que se realiza mediante mensajes de texto (SMS). Al igual que con el phishing, los atacantes envían un mensaje que parece ser de una fuente confiable. Al acceder al enlace proporcionado en el mensaje se puede instalar software malicioso en el dispositivo móvil de la víctima o dirigir al usuario a un sitio web fraudulento.

2. **Amenazas contra los datos** Este grupo engloba un conjunto de amenazas cibernéticas que apuntan directamente a las fuentes de datos con el objetivo de obtener acceso no autorizado o manipular la información para interferir en el comportamiento de los sistemas.

   Una brecha en la seguridad datos (data breach) es un ataque intencional llevado a cabo por un ciberdelincuente con el objetivo de obtener acceso no autorizado y la divulgación de datos sensibles, confidenciales o protegidos. Se trata pues de un incidente en el que la información se obtiene de manera ilegítima, con el fin de causar daño, extraer beneficio económico o incluso comprometer la seguridad de una entidad.

   Por otro lado, una fuga de datos (data leak) es un evento que puede causar la fuga no intencional de información sensible, confidencial o protegida. A diferencia de las brechas de datos, las fugas de datos suelen ser accidentales y son causadas por una variedad de razones entre las que se incluyen configuraciones incorrectas de los sistemas, vulnerabilidades no corregidas en el software o hardware o errores humanos. Por ejemplo, un empleado puede enviar accidentalmente un correo electrónico que contiene información confidencial a la persona equivocada, o un administrador de sistema puede configurar incorrectamente un servidor, permitiendo el acceso no autorizado a los datos almacenados.

3. **Ataques de denegación de servicio (Denial-of-Service (DoS) attacks and DDos)** Los ataques de denegación de servicio (DoS) y de denegación de servicio distribuidos (DDoS) tienen como objetivo interrumpir o degradar el funcionamiento de un servicio en línea, como un sitio web o una aplicación, haciéndolo inaccesible para los usuarios legítimos. En un ataque de denegación de servicio (DoS), un atacante utiliza una única fuente (por ejemplo, un ordenador o un servidor) para inundar el objetivo con una gran cantidad de tráfico falso o solicitudes maliciosas. Esto puede agotar los recursos como ancho de banda, memoria o capacidad de procesamiento, y provocar que el servicio se ralentice o colapse, impidiendo el acceso al mismo

Un ataque de denegación de servicio distribuido es similar a un ataque DoS, pero se lleva a cabo utilizando múltiples fuentes de tráfico distribuidas en diferentes ubicaciones, a menudo mediante el control de una red de dispositivos comprometidos llamada botnet. Al distribuir el ataque a través de varias fuentes, un ataque DDoS es más difícil de detener y rastrear, y puede generar una cantidad mucho mayor de tráfico malicioso hacia el objetivo.

Para protegerse de los ataques DoS y DDoS, las empresas pueden implementar medidas como el uso de servicios de mitigación de DDoS, la configuración de firewalls para bloquear tráfico sospechoso, la aplicación de sistemas de detección de intrusiones y la adopción de una arquitectura de red resiliente que pueda distribuir el tráfico de forma eficiente.

4. **Amenazas contra la disponibilidad de internet** Este tipo de amenazas están dirigidas a comprometer la disponibilidad y accesibilidad de internet, un recurso esencial en la vida moderna. Tales ataques se pueden realizar de diferentes formas, pero todas buscan limitar o interrumpir el acceso a la información y servicios en línea.

   Una forma de amenaza es la toma de control físico y destrucción de la infraestructura de internet. Esto puede implicar ataques a servidores, routers, nodos de internet, cables submarinos u otros equipos físicos que forman la columna vertebral de la red. En el peor de los casos, estos ataques pueden resultar en cortes generalizados de internet, afectando a usuarios, empresas y servicios esenciales. El sabotaje de la infraestructura física de internet puede ser llevado a cabo por ciberdelincuentes o grupos terroristas, pero también puede ser resultado de conflictos políticos o bélicos.

   Otro tipo de amenaza contra la disponibilidad del internet es la censura activa. En este contexto, la censura puede venir en forma de bloqueo de ciertos sitios web, como páginas de noticias o redes sociales. Esta forma de ataque suele ser común en regiones donde los gobiernos buscan controlar o restringir el flujo de información, limitando la libertad de expresión y el acceso a la información.

5. **Malware** El malware, abreviatura de "software malicioso", es un tipo de software diseñado para infiltrarse, dañar o realizar acciones no autorizadas en sistemas informáticos, dispositivos electrónicos o redes. El objetivo de los ciberdelincuentes al crear y distribuir malware puede variar e incluye el robo de información confidencial, espionaje, sabotaje, fraude o simplemente causar daño y molestias a los usuarios. El malware puede poner en riesgo tanto la confidencialidad, como la integridad y la disponibilidad de un sistema, se trata por tanto de una las pocas amenazas a la seguridad que compromete potencialmente todos los puntos de la triada CIA.

Entre los principales tipos de malware caben destacar:

- Virus: infectan archivos y programas, y pueden propagarse a través de dispositivos y redes. El término virus suele usarse coloquialmente como sinónimo de malware, aunque en la práctica existan ejemplos de software malicioso que no entran en esta categoría.
- Troyanos: se disfrazan de software legítimo y, una vez instalados, permiten a los atacantes acceder y controlar el sistema comprometido.
- Ransomware: se trata de un tipo de malware cada vez más complejo, que por su impacto y la relevancia de los incidentes que provoca se suele clasificar en una categoría propia separa del resto de malware. Los atacantes toman el control de los datos de un sistema u organización cifrándolos y exigiendo un rescate para liberarlos, lo que suele comprometer el correcto funcionamiento de empresas o instituciones.
- Spyware: recopila información del sistema o del usuario sin su consentimiento.
- Adware: muestra anuncios no deseados y puede recopilar datos del usuario para orientar anuncios específicos.
- Gusanos: se propagan automáticamente a través de redes, aprovechando vulnerabilidades en sistemas y software

Dada la relevancia del malware, y que se trata del tema principal de este trabajo, en la siguiente sección veremos los métodos clásicos usados para su detección.

2.1.2. Métodos clásicos de detección de malware

La detección de malware es un elemento crucial dentro de la ciberseguridad. Su objetivo es identificar y bloquear software malicioso antes que llegue a infectar una red o sistema, o identificar, neutralizar y eliminar cualquier código malitencionado que haya conseguido infiltrarse. Los métodos clásicos de detección de malware se basan en un conjunto de técnicas y reglas que detallaremos y analizaremos a continuación

1. **Detección basada en firmas** Se trata de un método ampliamente utilizado en la detección de malware [7]. Una firma es en esencia una secuencia de bytes que se utiliza para identificar un software malicioso específico. De esta forma, mediante patrones de coincidencia se analiza si un archivo es o no malicioso.

   Las soluciones de antivirus que se basan en la detección de firmas deben mantener un repositorio de firmas de software malicioso que debe ser continuamente actualizado debido a que se crea e identifica malware continuamente.

Una de las ventajas de la detección basada en firmas es su simplicidad y rápidez, además de cierta eficacia contra los tipos de malware más comunes. Sin embargo, este enfoque tiene ciertas limitaciones como el requerir de una base de datos que debe ser actualizada de forma continua, pues cualquier malware cuya firma no esté presente no será detectado. Además, la detección basada en firmas puede ser evadida a través de ciertas técnicas de ofuscación [9]. Por ejemplo, un ciberdelincuente puede cambiar mínimamente el código de un malware sin alterar su funcionalidad, creando una *variante* que tiene una firma diferente y que por lo tanto no será reconocida como malware por el antivirus.

2. **Análisis estático** A diferencia del enfoque de detección basado en firmas, el análisis estático examina el código fuente o el archivo binario de un programa sin necesidad de ejecutarlo, buscando patrones de código que podrían indicar actividad maliciosa.

   Este método de análisis ofrece ciertas ventajas como el no requerir de la ejecución del programa, por lo que se trata de una forma segura de examinar el software potencialmente dañino sin riesgo de infectar el sistema donde se realiza el análisis. Además, al proporcionar una visión de lo que un programa está diseñado para hacer, permite identificar características de malware como conexiones a redes sospechosa o el acceso a sectores del sistema operativo que el programa no debería usar.

   Sin embargo, el análisis estático también tiene limitaciones como el hecho de que es difícil de aplicar en archivos cuyo código ha sido ofuscado o empaquetado para ocultar su funcionalidad real. Este problema, que es común al análisis basado en firmas, puede hacer que el análisis del código sea más complicado o incluso imposible y que consuma más tiempo.

   Por otro lado, el análisis estático puede no ser capaz de detectar ciertos comportamientos maliciosos que solo se revelan cuando el programa es ejecutado [3]. Esto significa que algunos tipos de malware pueden pasar desapercibidos si se analizan exclusivamente con métodos estáticos [62].

3. **Análisis dinámico** Este es un enfoque complementario al análisis estático. Mientras que en el análisis estático se examina el contenido de archivos y programas en busca de información que delate un carácter potencialmente malicioso, el análisis dinámico implica la ejecución del código para observar su comportamiento.

   Este análisis se lleva a cabo en un entorno de pruebas aislado, conocido como *sandbox*, lo que permite examinar las posibles amenazas sin poner en riesgo de infección al sistema principal. Esto ofrece una ventaja importante ya que permite observar el comportamiento de un programa en un entorno seguro y controlado.

El análisis dinámico de malware es especialmente útil para descubrir amenazas no documentadas previamente que no suelen encontrarse mediante el análisis de malware estático.

No obstante, este método también presenta ciertas limitaciones como el hecho que algunos programas maliciosos están diseñados para detectar cuando están siendo ejecutados en un entorno de *sandbox* y pueden alterar su comportamiento en consecuencia para eludir la detección [34]. Además, el análisis dinámico puede ser más lento y requerir más recursos que el análisis estático, ya que implica la ejecución del código y la monitorización de su comportamiento.

2.1.3. El problema de la detección de virus

El problema de detectar malware ha sido ampliamente discutido en el campo de la ciberseguridad y la teoría de la computación. Fred Cohen, en su disertación de 1986 *Computer Viruses - Theory and Experiments* [18], introdujo la idea de la *indecidibilidad* en la detección de virus informáticos.

Este problema, según demostró Cohen guarda relación con otro ya conocido en la teoría de la computación: el *problema de la parada*. El problema de la parada, demostrado por Alan Turing en 1936 [74], establece que no existe un algoritmo que pueda determinar en todos los casos posibles si un programa arbitrario se detendrá o continuará ejecutándose para siempre dada una entrada específica.

Aplicado al software malicioso, este *problema de detección de virus* establece que no hay un algoritmo capaz de identificar con total precisión todos los virus posibles en todos los escenarios. Esto se debe a la capacidad del malware de modificar su propio código mediante técnicas de ofuscación y polimorfismo, lo que desafía la identificación a través de firmas de virus estáticas. Además, un virus puede diseñarse para presentar un comportamiento similar al problema de la parada, donde su ejecución puede detenerse o continuar dependiendo de ciertas condiciones, por ejemplo, podría pasar desapercibido durante el análisis dinámico si está diseñado para mostrar compartamientos maliciosos pasado un tiempo significativo desde su instalación. De esta manera, el problema de detección de virus, al igual que el problema de la parada, es indecidible: no hay un algoritmo que pueda abordarlo de manera precisa en todos los contextos posibles.

Es importante señalar que aunque la detección de virus es un problema indecidible y por tanto nunca será posible construir un algoritmo capaz de identificar todos los virus (salvo quizás un algoritmo trivial que clasifique como malicioso cualquier entrada que reciba), en la práctica los antivirus pueden y de hecho detectan y neutralizan una gran cantidad de virus informáticos

conocidos. Para conseguir esto se utilizan las técnicas vistas anteriormente, en combinación con los algoritmos de aprendizaje automático que estudiaremos en la siguiente sección.

2.2. Aprendizaje automático e Inteligencia Artificial

La inteligencia artificial (IA) y el aprendizaje automático (AM) son dos disciplinas que están transformando rápidamente nuestra sociedad, con un uso prácticamente omnipresente en múltiples secotres. A pesar de esta presencia generalizada, sigue habiendo cierta ambigüedad en torno a la definición exacta de IA. La Inteligencia Artificial se refiere a la ciencia y la ingeniería cuyo objetivo es crear máquinas inteligentes que puedan percibir, razonar y actuar como seres humanos [42]. La IA no es un concepto nuevo, sus orígenes se remontan a la década de 1940, con los trabajos teóricos de Alan Turing, y el surgimiento de las primeras máquinas que intentaron imitar el cerebro humano.

Los primeros éxitos en el campo de la inteligencia artificial incluyen la creación de Eliza, el primer programa informático capaz de procesar lenguaje natural [76], y el desarrollo de programas informáticos de resolución de problemas capaces de resolver automáticamente juegos como las Torres de Hanoi. En las siguientes décadas continuó el estudio de este campo con un aumento del interés de la industria, y por ello de la financiación.

El siguiente paso para la IA era tener la capacidad de procesar datos externos, aprender de ellos y adaptarse a entornos cambiantes. Esto llevó al desarrollo de redes neuronales y el Deep Learning.. Pero en este punto la frontera entre la inteligencia artificial y el aprendizaje automático se torna más difusa En las últimas décadas, la IA ha experimentado avances significativos con la creación de AlphaGo de Google, que ha conseguido derrotar a los mejores jugadores de Go del mundo [16]. Quizás el avance más relevante de los últimos años lo encontramos en los chats basados en modelos de lenguaje como ChatGPT-3 de OpenAI: en este punto la IA vuelve a encontrarse con Eliza.

El aprendizaje automático, por su parte, es un campo dentro de la inteligencia artificial que se centra en el desarrollo de algoritmos y modelos estadísticos que permiten a los ordenadores aprender de los datos, sin ser programados explícitamente para ello. Estos algoritmos y modelos pueden utilizarse para hacer predicciones, clasificar datos y realizar otras tareas basadas en patrones y relaciones identificadas en los datos. Cuatro áreas de las matemáticas se combinan y fundamentan los algoritmos de aprendizaje automático: Cálculo, Álgebra Lineal, Probabilidad y Estadística. El campo del aprendizaje automático tiene sus raíces en los años 50 y 60 del siglo XX [29] con el desarrollo del “perceptron” por parte de Frank Rosenblatt [57]. Resulta importante señalar que se trataba de una máquina y no del algoritmo del mismo nombre que se analizará en este trabajo y que constituye la base de las redes neuronales. Aunque estos sean los

orígenes del aprendizaje automático como disciplina, algunos de los algoritmos que estudiaremos son anteriores, como es el caso del Discriminante Linear de Fisher (DLF) presentado en 1936 [25] . Sin embargo, estos algoritmos más antiguos pueden encuadrarse completamente en el área de la Estadística. En los últimos años, el aprendizaje automático ha experimentado un crecimiento y desarrollo significativos debido a los avances en el campo de la computación, con la construcción de ordenadores con una capacidad de cálculo cada vez más grande y a la disponibilidad de grandes conjuntos de datos.

En la actualidad, el aprendizaje automático se utiliza ampliamente en diversos campos, como la informática, la ciencia de datos, las finanzas, la sanidad o la publicidad, y como veremos, en la seguridad informática.

2.2.1. Algunas clasificaciones de algoritmos de aprendizaje automático

Antes de nombrar y describir algunos de los algoritmos de aprendizaje automático existentes conviene mencionar al menos una parte de las diveras formas en las que los podemos clasificar. Esto resulta útil pues suele ser la forma más sencilla de decidir que método utilizar para resolver determinado problema a partir de las características del mismo

Los algoritmos de ML pueden clasificarse por ejemplo atendiendo a como se realiza el proceso de entrenamiento, es decir, de acuerdo al proceso con el cual el algoritmo *aprende*. De esta forma, tenemos las clasificaciones de:

- Aprendizaje supervisado: En el aprendizaje supervisado, el algoritmo se entrena en un conjunto de datos etiquetados, que incluye tanto los datos de entrada como las etiquetas de salida correctas correspondientes. El algoritmo utiliza estos datos de entrenamiento para aprender a predecir las etiquetas de salida de datos nuevos que no se han visto. Algunos ejemplos de algoritmos de aprendizaje supervisado son la regresión lineal, la regresión logística y las máquinas de vectores soporte.

- Aprendizaje no supervisado: En el aprendizaje no supervisado, el algoritmo no es entrenado con datos etiquetados. Sino que debe descubrir la estructura subyacente de los datos mediante técnicas como la agrupación o la reducción de la dimensionalidad. Algunos ejemplos de algoritmos de aprendizaje no supervisado son la agrupación k-means y el análisis de componentes principales.

- Aprendizaje semisupervisado: En el aprendizaje semisupervisado, el algoritmo se entrena en un conjunto de datos que incluye tanto datos etiquetados como no etiquetados. Este

tipo de aprendizaje puede ser útil cuando es difícil o costoso etiquetar completamente un gran conjunto de datos y esta tarea se hace parcialmente

- Aprendizaje por refuerzo (reinforcement learning): En el aprendizaje por refuerzo, el algoritmo aprende por ensayo y error, recibiendo recompensas o castigos por sus acciones en un entorno determinado. Este tipo de aprendizaje se utiliza a menudo en robótica y sistemas de control.

También pueden clasificarse de acuerdo con el tipo de predicción que realizan. Por ejemplo, los algoritmos que predicen un valor categórico (como decidir si un correo electrónico es spam o no) se denominan algoritmos de clasificación, mientras que los algoritmos que predicen un valor numérico continuo (como la probabilidad de que determinada consulta sea un ataque de denegación de servicio) se denominan algoritmos de regresión.

Los algoritmos de aprendizaje automático también pueden clasificarse en función de la separabilidad lineal del conjunto de datos de entrada. Se dice que dos conjuntos de puntos son linearmente separables cuando su envoltura convexa es disjunta. Definiéndose la envoltura convexa de un conjunto de puntos como :

$$Conv(X) = \left\{ \sum_{i \in I} \lambda_i x_i : \lambda_i > 0, \sum_{i \in I} \lambda_i = 1 \text{ y } \lambda_i = 0 \text{ solo para un número finito de índices} \right\}$$

Tenemos por tanto algoritmos que solo pueden resolver correctamente problemas linearmente separables, y que llamaremos algoritmos lineales, y otros métodos de aprendizaje automático que pueden resolver tanto problemas lineales como no lineales. La ventaja de los primeros viene dada por su mayor sencillez y menor coste computacional.

2.2.2. Ejemplos de algoritmos de aprendizaje automático

2.2.2.1. Algoritmos lineales

Podemos iniciar el listado de algoritmos con aquellos considerados más sencillos: los algoritmos lineales.. Entre los algoritmos de aprendizaje automático lineales más conocidos y utilizados tenemos el Discriminante lineal de Fisher, el Análisis Discriminante Lineal, la regresión lineal y el perceptrón.

El discriminante lineal de Fisher y el Análisis Discriminante Lineal (LDA) son algoritmos supervisados que buscan encontrar una proyección lineal que maximice la separabilidad entre clases en un conjunto de datos etiquetados. La única diferencia entre ambos es que el discriminante lineal de Fisher resuelve el problema para dos clases (o grupos) mientras que el LDA

es la extensión de este método a más clases. El LDA aplicado a dos clases es justamente el Discriminante de Fisher.

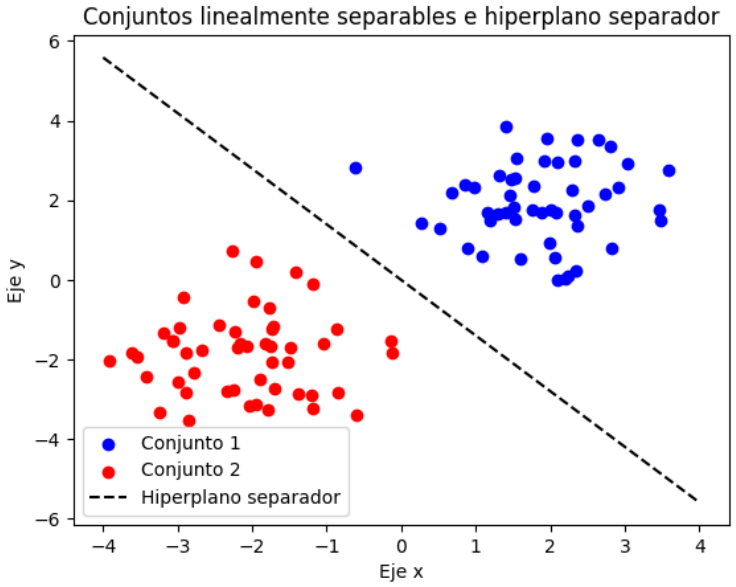


Figura 2.1: Conjuntos linearmente separables

El objetivo de estos algoritmos es encontrar una proyección lineal de los puntos de forma que se intente maximizar la varianza total de las medias de los datos proyectados al tiempo que se minimiza la varianza de los datos proyectados dentro de la misma clase. En el caso de dos clases esto equevale a encontrar una recta de proyección de tal forma que los datos de clases distintas queden separados por un punto $c \in \mathbb{R}$

Otra característica de estos métodos es que se tratan de técnicas de reducción de dimensiones: el modelo que generan estos algoritmos posee una dimensión inferior a la de los datos de entrada, lo que facilita la visualización de los resultados (por ejemplo datos de dimensión superior a 3 que no pueden ser visualizados fácilmente pueden ser proyectados a un plano o a una recta lo que facilita su representación)

La regresión lineal (o recta de regresión) es un algoritmo utilizado para predecir valores numéricos continuos basándose en un conjunto de puntos o datos de entrada. La idea básica es encontrar la mejor línea que se ajuste a los puntos de los datos dados. Esta línea se denomina línea de regresión y representa la relación entre las características de entrada y la variable de salida. Esto es, dada una variable dependiente (llamada respuesta) y una o varias variables independientes (predictores) encontrar los coeficientes β que minimicen la función de error cuadrático dada por:

$$\min_{\beta} \sum_{i=1}^{n} (y_i - (\beta_0 + \beta_1 x_{i1} + \ldots + \beta_d x_{id}))^2 \tag{2.1}$$

donde los y_i son las respuestas y $x_{i1}, \ldots, x_{id}$ son las características de entrada para la observación *i-ésima*.

Otro de los algoritmos lineales es el perceptrón, cuyo interés principal viene dado por el hecho de que simula una neurona biológica y se trata del caso más sencillo de red neural (aquella formada por una única neurona). Se trata de un algoritmo utilizado fundamentalmente en tareas de clasificación binaria y que realiza predicciones basándose en combinaciones lineales de los datos de entrada, asignándole a cada característica un peso determinado w. El perceptrón calcula una suma ponderada de las características de entrada y emite una salida binaria basada en si la suma ponderada es mayor o menor que cierto umbral de activación b:

$$\hat{y} = \begin{cases} 1 & \text{si } w \cdot x - b > 0 \\ 0 & \text{en caso contrario} \end{cases} \tag{2.2}$$

El algoritmo de entrenamiento del perceptrón actualiza los pesos y el umbral basándose en los errores cometidos en las predicciones.

Otro aspecto interesante del perceptrón es el algoritmo utilizado en su entrenamiento: el descenso de gradiente estocástico que es ampliamente utilizado en otros métodos de aprendizaje automático. El objetivo del descenso de gradiente es minimizar el error definido como función de coste. Se puede demostrar que siempre que los datos de entrada sean linearmente separables el algoritmo de descenso de gradiente aplicado al perceptrón converge a la solución [52].

2.2.2.2. Árboles de decisión y métodos ensemble

Los árboles de decisión son algoritmos de aprendizaje supervisado utilizados tanto para tareas de clasificación como de regresión. Funcionan dividiendo iterativamente el espacio de características en subespacios homogéneos, tomando decisiones basadas en reglas simples derivadas de los datos de entrenamiento. Pueden clasificar datos de clases no linearmente separables, por lo que pueden resultar muy útiles ante problemas que los algoritmos antes vistos no pueden resolver.

La construcción de un árbol de decisión implica la selección de atributos y la división de los datos en subconjuntos basados en el valor de esos atributos. Para decidir qué atributo se debe seleccionar en cada nodo, se utilizan medidas de impureza como el índice de Gini, la entropía o el error de clasificación. El objetivo es minimizar la impureza en cada subconjunto resultante.

$H(S) = -\sum_{i=1}^{n} p_i \log_2 p_i$, donde S es un conjunto de datos y p_i es la probabilidad de la clase i en S.

Los árboles de decisión ofrecen varias ventajas, como su fácil interpretabilidad, ya que permiten visualizar y comprender las decisiones tomadas por el algoritmo, se trata por tanto

de de un algoritmo *caja blanca*. Además, pueden manejar tanto variables categóricas como numéricas sin necesidad de preprocesamiento adicional (aunque esto es cierto desde un punto de vista teórico, varias implementaciones existentes, por ejemplo en ciertas librerías de Python, solo permiten trabajar con datos númericos) Igualmente, no se ven afectados por la escala de los datos, lo que elimina la necesidad preprocesar la información de entrada para normalizarla.

Sin embargo, también presentan algunas desventajas. Por un lado, tienden a sufrir de sobreajuste (overfitting), lo que puede resultar en un bajo rendimiento al usar datos que no han sido vistos durante el entrenamiento. Además, son sensibles a pequeñas variaciones en los datos de entrada, lo que puede provocar cambios significativos en la estructura del árbol resultante. Los árboles de decisión también pueden estar sesgados si algunas clases están sobrerrepresentadas en los datos de entrenamiento, por lo que es importante equilibrar el conjunto de datos durante el preprocesado.

El algoritmo de Random Forest es una extensión del concepto de árboles de decisión, donde en lugar de construir un único árbol de decisión, se genera un conjunto de árboles durante el entrenamiento del modelo. Posteriormente las predicciones de todos los árboles se promedian (en el caso de regresión) o se selecciona la clase que recibe más votos (en el caso de clasificación) para hacer la predicción final. Este procedimiento pertenece a un conjunto de técnicas llamadas métodos *ensemble* donde se combinan diferentes algoritmos para obtener un modelo final más estable y menos susceptible al sobreajuste. Además, Random Forest introduce más aleatoriedad en la construcción de los árboles, eligiendo al azar subconjuntos de las características en cada división.

Por otro lado, eXtreme gradient boosting tree (o árbol de potenciación de gradiente extremo) es otro algoritmo que pertenece a la familia de los métodos ensemble y que también se basa en árboles de decisión. En este caso se utiliza una técnica diferente llamada boosting, donde en vez de construir los árboles de forma independiente como en Random Forest, estos se crean de forma secuencial: cada nuevo árbol trata de corregir los errores cometidos por el conjunto de árboles anteriores. Esto se logra utilizando el algoritmo gradient boosting para minimizar una función de pérdida que cuantifica la diferencia entre las predicciones y los verdaderos valores de salida de los árboles anteriores.

Sobre el nombre de este algoritmo debemos señalar que es usual en la literatura denominarlo XGBoost, que es también el nombre de una librería de código abierto, disponible en varios lenguajes de programación, con implementaciones de este método. En lo adelante en este trabajo utilizaremos el nombre de *XGBoost* para referirnos tanto al algoritmo como a la librería.

Volviendo al algoritmo, este presenta una serie de ventajas como su eficiencia computacional y la capacidad de manejar conjuntos de datos de gran tamaño. También es flexible, ya que permite personalizar la función de pérdida y ofrece varias opciones para regularizar el modelo

y evitar el sobreajuste.

Tanto Random Forest como XGBoost representan un avance importante respecto a los árboles de decisión, proporcionando maneras de combinar múltiples árboles para mejorar la precisión y la estabilidad de las predicciones. De hecho, estos algoritmos superan a métodos mucho más costosos computacionalmente, como deep learning, si los datos de entrada están en formato tabular [33].

Un punto negativo de ambos algoritmos es que se pierde algo de la interpretabilidad de un único árbol de decisión, ya que la predicción final es el resultado de múltiples árboles en lugar de una serie de divisiones claras de las características.

2.2.2.3. Redes neuronales

Las redes neuronales artificiales (ANN) son un algoritmo de aprendizaje automático inspirado en el funcionamiento del cerebro humano entendido como un conjunto de neuronas interconectadas. Anteriormente se explicó el funcionamiento del perceptrón como una unidad básica de procesamiento. Las redes neuronales se construyen conectando múltiples perceptrones, formando capas (layers) o estructuras más complejas. De esta forma se construye un modelo que puede clasificar datos etiquetados (pues se trata de aprendizaje supervisado) tanto lineales como no lineales. Al contrario que el perceptrón una red neural puede resolver problemas de clasificación que no son linealmente separables [80].

Una red neuronal artificial típica consiste en una capa de entrada, una o más capas ocultas y una capa de salida. Cada una de estas capas contiene un conjunto de neuronas, también llamadas nodos, que están conectadas con algunas o todas las neuronas de la siguiente capa. Estas conexiones tienen ciertos pesos que se ajustan durante la fase de entrenamiento.

El proceso de entrenamiento de una red neuronal implica ajustar los pesos de las conexiones mediante un algoritmo de optimización, como el descenso de gradiente estocástico, para minimizar cierta función de pérdida que mide la discrepancia entre las predicciones de la red y los valores reales. Uno de los algoritmos más comunes para entrenar redes neuronales es el de retropropagación, que consiste en aplicar la regla de la cadena del cálculo de derivadas para ajustar los pesos de las conexiones en función de los errores propagados desde la capa de salida hacia la capa de entrada.

El Deep Learning es una área dentro del aprendizaje automático que se centra en el uso de redes neuronales con múltiples capas ocultas, conocidas como redes neuronales profundas (DNN). Estas redes son capaces de aprender representaciones jerárquicas de los datos, permitiendo la detección y extracción automática de características a diferentes niveles de abstracción. Esto las hace particularmente adecuadas para tareas de aprendizaje automático que involucren datos de alta dimensión y complejidad, como el reconocimiento de imágenes, la traducción automática

y el análisis de voz.

2.2.2.4. Máquinas de vectores soporte

Las Máquinas de Vectores Soporte (SVM, por sus siglas en inglés) son algoritmos de aprendizaje supervisado utilizados principalmente para tareas de clasificación y, en menor medida, de regresión. El objetivo de las SVM es encontrar el hiperplano que mejor separa dos clases en el espacio de características, considerando como *mejor* aquel que maximice el margen entre ellas. Es decir, buscamos aquel hiperplano que guarde la mayor distancia posible a los puntos más cercanos de ambas clases, llamados vectores de soporte, quedando definido el margen justamente como la distancia de los hiperplanos a estos vectores soporte [8].

Esto se plantea como un problema de maximización dentro de la investigación operativa.

Las Máquinas de Vectores soporte hasta ahora explicadas solo son capaces de resolver problemas de clasificación linealmente separables, si queremos extenderlas a otra clase de problemas hay que hacer uso de los métodos kernel. Estos métodos (llamados también trucos kernel) consisten en proyectar los datos a un espacio de mayor dimensión (espacio de características transformado) donde los datos puedan ser separados linealmente, y luego aplicar el algoritmo en este nuevo espacio.

La clave del método kernel es que no es necesario calcular explícitamente las transformaciones de las características en el espacio de mayor dimensión. En lugar de ello, se utiliza una función kernel que calcula el producto escalar de los datos transformados, sin necesidad de realizar la transformación en sí. Esto resulta en un cálculo computacionalmente eficiente y permite manejar espacios de características de dimensiones muy altas.

2.2.2.5. Análisis de Componentes Principales (PCA)

El Análisis de Componentes Principales (PCA por sus siglas en inglés) es el único ejemplo de algoritmo no supervisado que veremos en este trabajo. Se trata de una técnica de reducción de dimensiones lineal que busca encontrar las direcciones de mayor varianza en los datos. La idea es proyectar los datos originales en un espacio de menor dimensión (de forma similar al discriminante lineal de Fisher) de tal forma que se maximice la varianza y se minimice la pérdida de información.

Esto se consigue calculando la matriz de covarianza del conjunto de datos, y extrayendo de ella los autovectores y autovalores (esta matriz siempre es cuadrada). Los autovectores representan las direcciones de los componentes principales, mientras que los autovalores indican la cantidad de información (varianza) aportada por su autovector correspondiente. Al elegir los k primeros autovectores ordenados según sus autovalores, construimos una matriz de transformación que permite proyectar los datos auna dimensión $k < n$ de forma que se conserve la mayor

variabilidad posible [31].

2.3. Aprendizaje automático aplicado a la ciberseguridad

En esta sección veremos aplicaciones de algoritmos vistos en el apartado anterior ante las diferentes amenazas a la ciberseguridad que han sido descritas en este trabajo. Recopilaciones similares de trabajos sobre este tema han sido hechas en algunos artículos de investiación [21], [71].

2.3.1. Algoritmos lineales

Las limitaciones de los algoritmos lineales, fundamentalmente en lo referente a su incapacidad de modelar datos con relaciones no lineales, hacen que no sean adecuados como método exclusivo en aplicaciones de ciberseguridad. Sin embargo, puede ser utilizados en combinación con otros algoritmos (lineales o no lineales) para mejorar el rendimiento global del proceso. Por ejemplo, se han desarrollado plataformas de seguridad para detectar ciberataques en redes de comunicación recurriendo a una combinación de algoritmos lineales y no lineales [50] . De entre los algoritmos utilizados cabe destacar el Perceptron Multicapa, que aunque se trate de un algoritmo no lineal se fundamenta en el Perceptrón que como ya se vió sí que es lineal.

2.3.2. Árboles de decisión y métodos ensemble

Utilizando árboles de decisión se han conseguido superar a otras técnicas de clasificación como redes neuronales o máquinas de vectores soporte en la detección de ataques DoS utilizando el conjunto de datos KDD Cup 1999 [54] . Los autores propusieron técnicas de podado para reducir el sobreajuste del algoritmo (que como vimos es un problema común de los árboles de decisión) y aumentar la precisión.

Los árboles decisión también han sido utilizados de forma exitosa en la detección de malware. Por ejemplo, se han llegado a obtener mejores resultados con árboles de decisión que con Máquinas de Vectores Soporte o con el Perceptrón multicapa en la tarea de clasificar software en base a su comportamiento con el fin de identificar malware [24].

Siguiendo con la detección de malware, se ha analizado el uso del algoritmo Random Forest para clasificar aplicaciones de Android como maliciosas o benignas, obteniendo resultados muy buenos (precisión superior al 99 %) [2].

Por su parte, usando Random Forest en la tarea de detección de Ransomware, se han obtenido resultados óptimos con 100 árboles (precisión 97.7 %) [41].

Por otro lado, el algoritmo XGBoost ha sido usado también en la tarea de extracción de características, elemento clave en el proceso de entrenamiento de algoritmos en la detección de malware [43]. La importancia de este trabajo viene en el hecho que gracias a esta correcta extracción de características se puede construir un clasificador optimizado hacia la eficiencia que permite el entrenamiento de modelos con recursos de computación limitados

2.3.3. Redes neuronales y Deep Learning

Debido al avance de las tecnologías de hardware y a la existencia de conjuntos de datos públicos, las Redes Neuronales Artificiales (ANNs) se utilizan en diversas aplicaciones de ciberseguridad.

Un Sistema de Detección de Intrusiones (IDS) que utiliza un clasificador basado eb redes neuronales para identificar patrones en datos de red fue propuesto, mejorando significativamente el rendimiento del método de detección de intrusiones basado en firmas [63].

En cuanto a la detección de malware, se ha propuesto usar una red neuronal en el contexto de aprendizaje semisupervisado, donde también fue utilizado el algoritmo PCA [69]. Con un conjunto de datos privado, los autores evaluaron el rendimiento de su algoritmo y afirmaron que su enfoque puede obtener una tasa de falsos positivos más baja en comparación con la tecnología de detección de anomalías convencional.

Si nos movemos al área del Deep Learning, en particular a las redes neuronales convolucionales, tenemos diversas aplicaciones en el proceso de reconocimiento de patrones. El Internet de las Cosas (IoT) ha generado grandes volúmenes de datos y ha abierto muchas vulnerabilidades que pueden ser aprovechadas por atacantes. La falta de soluciones de seguridad y métodos de detección para entornos emergentes de IoT ha provocado recientemente muchos ataques DDoS, como Mirai y Brickerbot botnets, entre otros [21]. Para detectar estos ataques de malware DDoS en grandes conjuntos de datos de usuarios, se están empleando algoritmos basados en CNN. Por ejemplo, en 2018, algunos autores [68] propusieron un enfoque novedoso para la detección de malware, donde primero se convierte el binario de malware en una imagen en escala de grises y luego se utiliza una Red neuronal convolucional para identificar el malware. Sus resultados experimentales mostraron que durante la clasificación de dos clases, su enfoque proporcionó una precisión del 94,0 %, mientras que para la clasificación de tres clases ofreció una precisión del 81,8 %.

2.3.4. Máquinas de vectores soporte

Respecto a las máquinas de vectores soporte, se ha desarrollado un modelo de clasificación para la detección de anomalías en sistemas de control industrial [70]. Los autores utilizaron

parámetros de las comunicaciones como los intervalos y longitud de paquetes de los sistemas de control industrial (ICS) como características bases para entrenar su modelo. Además, propusieron integrar su modelo discriminante con el Sistema de Detección de Intrusiones (IDS) existente para complementar dicho IDS. Evaluaron su método en un banco de pruebas de ciberseguridad mediante pruebas de penetración. Sus resultados experimentales demostraron que el IDS existente con la integración del algoritmo SVM proporciona una tasa de falsas alarmas un 5 % menor en comparación con el sistema de detección de intrusiones (sin SVM)

En otro caso de uso de máquinas de vectores soporte se presentó una propuesta innovadora para identificar a usuarios auténticos mediante el desarrollo de un mecanismo de autenticación de dos factores utilizando patrones de presión y frecuencia de pulsación de teclas [79]. El algoritmo SVM fue utilizado en la tarea de detectar a usuarios reales, planteado como un problema de clasificación binaria. Este enfoque novedoso puede contribuir a mejorar el rendimiento de las tecnologías de autenticación para evitar accesos no autorizados.

2.3.5. Soluciones comerciales y de código abierto

Hasta el momento nos hemos centrado en artículos acádemicos donde se estudian las aplicaciones del aprendizaje automático a la ciberseguridad. Pero estas ideas, como se ya discutió, se aplican en numerosas soluciones de la industria. Se hace por tanto necesario hacer un análisis con algo más de detalles de las soluciones existentes, tanto comerciales como gratuitas o de código abierto y ver como aplican los diferentes métodos vistos hasta ahora.

Sin embargo, debemos tener en cuenta que el mundo de la ciberseguridad es muy competitivo, por lo que las empresas suelen ver sus tecnologías y algoritmos como valiosos activos comerciales. Dada la naturaleza sensible de esta información, suelen ser cautelosas a la hora de publicar detalles precisos sobre cómo funcionan sus sistemas de detección de malware. Este nivel de confidencialidad se extiende tanto a las tecnologías tradicionales de identificación de malware, como la detección mediante firma o el análisis heurístico, como a los algoritmos de Machine Learning.

El motivo de este hermetismo no es solo mantener una ventaja competitiva, sino también minimizar la posibilidad de que ciberdelincuentes obtengan información que podría ser utilizada para eludir los sistemas de detección. Al mantener los detalles técnicos en secreto, las empresas de ciberseguridad intentan hacer más difícil el diseño de malware capaz de burlar sus productos.

Por otra parte, existen numerosos sitios web y empresas que realizan comparativas y análisis de software antivirus, proporcionando informes detallados y clasificaciones basadas en pruebas y análisis. Si bien estos informes pueden ser útiles para obtener una visión general de las capacidades de diferentes soluciones antivirus, es importante tratar esta información con cierto grado de escepticismo. En muchos casos, no hay forma de garantizar completamente la veracidad

e imparcialidad de estas pruebas. Las metodologías utilizadas pueden variar significativamente o incluso llegar a estar influenciadas por intereses comerciales.

Hay que señalar que la falta de transparencia no es un problema exclusivo de las empresas de software antivirus, sino que en general dentro del campo de la ciberseguridad es más una norma que una excepción. Algunos problemas derivados de esta falta de transparencia han sido analizados, así como posibles soluciones entre las que se pueden señalar los Centros de Compartición y Análisis de Información (ISAC) [51]. También se han tratado algunos de los riesgos relativos a una excesiva transparencia en materia de seguridad informática, que, como ya discutimos, podría beneficiar a los ciberdelincuentes [35].

Entre las aplicaciones comerciales de seguridad informática que usan aprendizaje automático o inteligencia artificial caben destacar:

- **Kaspersky Anti-Virus** Es un producto de seguridad que proporciona protección en tiempo real contra toda clase de malware y otras amenazas en línea. Utiliza métodos clásicos como detección por firmas y heurística, combinados con tecnologías de inteligencia artificial y aprendizaje automático para mejorar la detección y la respuesta a amenazas. Entre los algoritmos de aprendizaje automático usados por Kaspersky se encuentran métodos ensemble basados en árboles de decisión, algoritmos de clustering como K-medias y redes neuronales multicapa, tal como se desprende de un artículo [28] de la propia empresa.

  La versión empresarial de este software ofrece un entorno de *sandbox* donde es posible hacer análisis dinámico de malware [59]. Por lo que los analistas de seguridad que usen esta solución tienen un espacio seguro para estudiar el comportamiento de archivos potencialmente peligrosos.

  Igualmente, Kaspersky ha creado centros de transparencia que permiten a agencias gubernamentales y grupos de ciberseguridad analizar con diferentes niveles de detalle los algoritmos usados y desarrollados por Kaspersky [40]

- **Microsoft Defender y Microsoft 365 Defender** Se trata de dos soluciones ofrecidas por Microsoft. La primera de ellas es la herramienta antes conocida como Windows Defender que viene integrada en los sistemas operativos Windows, por lo que es gratuita para sus usuarios. Se trata de una herramienta antimalware que brinda protección en tiempo real.

  De acuerdo con un artículo [45] publicado por el equipo de ciberseguridad de Microsoft, Microsoft Defender combina diversos algoritmos de aprendizaje automático, algunos muy ligeros que permiten analizar rápidamente gran cantidad de archivos. Entre ellos destacaba

LightGBM [46], que se basa en Gradient Boosting para árboles de decisión. De hecho la librería pública del mismo nombre fue desarrollada por Microsoft.

Microsoft 365 Defender por su parte, está más enfocada al sector empresarial y consiste en un conjunto de aplicaciones de ciberseguridad integradas en la suite de Microsoft 365. Una característica de Microsoft Security es que ofrece un entorno *sandbox* [60] donde se pueden ejecutar y probar aplicaciones de forma segura. De forma similar a otros entornos de este tipo, puede ser utilizado para hacer análisis dinámico de software.

- **Cylance** (perteneciente BlackBerry), fue una empresa conocida por su enfoque pionero en el uso del Machine Learning aplicado a la seguridad informática.

  Una característica especial de Cylance era que utilizaba exclusivamente aprendizaje automático [6], por lo que no recurría a ninguna de las técnicas clásicas que hemos visto anteriormente. Sin embargo, esto entrañaba ciertos riesgos. En 2019 un grupo de investigadores israelíes encontró una debilidad en el algoritmo de aprendizaje automático de Cylance, que permitía eludir la detección de malware simplemente agregando ciertas cadenas de texto a un archivo malicioso. Según explicaron en un artículo [65], en el algoritmo usado por Cylance existía un método de clustering que podía ignorar las clasificaciones del modelo principal y cuya función era reducir el número de falsos positivos y falsos negativos. De este artículo se desprende también que Cylance funcionaba extrayendo las características de los archivos que analizaba y posteriormente entrenaba un conjunto de redes neuronales en función de clasificación.

  Cylance fue descontinuado en 2022 y ya solo brinda soporte a antiguos clientes [5].

- **Trend Micro Antivirus+ Security** Es una herramienta de seguridad que protege contra virus, ransomware y otras amenazas utilizando diferentes tecnologías. Por ejemplo, Trend Micro usa aprendizaje automático para identificar posibles estafas a phishing a través de correo empresarial. [1]

  Trend Micro afirma ser el primer sistema autónomo de prevención de intrusiones de nueva generación (NGIPS) en usar aprendizaje automático en detección de intrusiones y malware [11] . En línea con lo anterior, patentaron un modelo (*TrendX Hybrid Model*) que combina métodos clásicos como análisis estático y dinámico con técnicas de Machine Learning, sin embargo, no se han encontrado detalles de los algoritmos en concreto que se utilizan

  Trend Micro, al igual que otras soluciones comerciales, ofrece un entorno *sandbox* para que los analistas de seguridad puedan hacer análisis dinámico [12]

Por otra parte, tenemos soluciones gratuitas o de código abierto entre las que podemos señalar:

Herramienta	Uso de Machine Learning	Transparencia de Algoritmos	Tipo de código	Uso Comercial
Kaspersky Anti-Virus	Sí	Centros de transparencia	Propietario	Sí
Microsoft Security	Sí	Publicaciones de Microsoft	Propietario	Sí
Cylance	Sí	Publicaciones de Cylance	Propietario	Sí
Trend Micro	Sí	No se especifican algoritmos	Propietario	Sí
Cuckoo Sandbox	No	Código Abierto	Código Abierto	No
Ember	Sí	Código Abierto	Código Abierto	No

Cuadro 2.1: Comparación de diferentes herramientas de detección de malware

- **Cuckoo Sandbox** Es un sistema de análisis dinámico de malware automatizado. Se trata de una herramienta de análisis de malware de código abierto que permite a los usuarios ejecutar y analizar archivos sospechosos en un entorno aislado y seguro. Es decir, que se trata de un entorno de *sandbox* gratuito y alternativo a los que ofrecen las soluciones comerciales antes mencionadas.

  Aunque no utiliza aprendizaje automático ni inteligencia artificial, Cuckoo Sandbox [58] ofrece información detallada sobre el comportamiento del malware y ayuda en la identificación de amenazas, además de proporcionar datos que sirven para entrenare modelos [39].

  Aprovechando el hecho que es código abierto se han desarrollado otras herramientas y APIs que mejoran esta solución. Por ejemplo CuckooML [19] buscaba ofrecer integración con librerías de Machine Learning de Python como scikit-learn, sin embargo, este proyecto parece haber sido abandonado y no recibe actualizaciones desde hace 7 años, por lo que es posible que presente problemas de compatibilidad.

- **Ember (Endgame Malware BEnchmark for Research)** se trata de un conjunto de datos etiquetados que incluye características extraídas de más de un millón de archivos binarios de Windows. Aunque no se trata de una herramienta de análisis de malware, sino de una base de datos para entrenar modelos, este proyecto incluye código para implementar fácilmente algoritmos como gradient boosting trees (LightGBM) [4].

Todas las soluciones comerciales anteriores tienen en común que no explican con detalle que algoritmos usan o como los aplican. Esto limita cualquier análisis que se quiera hacer sobre los modelos que utilizan.

Además, estas soluciones solo permiten analizar archivos binarios, lo que hace que no puedan ser usadas para verificar si las características extraídas de un archivo corresponden con un software malicioso o benigno.

Este punto tiene cierta relevancia porque buena parte de las bases de datos de malware disponibles en internet no incluyen los binarios originales, por lo que es imposible verificar si las etiquetas de los datos son correctas o no.

Poder evaluar la calidad de bases de datos de malware de forma sencilla, es algo clave a la hora de usar esta información para entrenar nuestros algoritmos. Si disponemos de un modelo ya entrenado que reciba características extraídas como datos de entrada podemos hacer una verificación rápida sobre las listas etiquetadas disponibles en la red y solo si los resultados entran en un margen razonable de error, usar esta información para mejorar nuestro modelo o entrenar otros algoritmos.

Todo lo anterior, unido al coste de usar estas soluciones nos hacen descartarlas en este trabajo.

Por otro lado, de las soluciones de código abierto, Cuckoo Sandbox no ofrece ninguna implementación de algoritmos de aprendizaje automático y está pensado para hacer exclusivamente análisis dinámico.

EMBER, por su parte, presenta diversos problemas de compatibilidad que imposibilitan su uso en versiones recientes de Python [38].

Por todo lo anterior se hace necesario tener una implementación que permita explorar aquellas opciones que no tenemos en los software existentes, a la vez que se pongan en práctica los diferentes algoritmos vistos en secciones anteriores, esa será la tarea a llevar a cabo en el siguiente capítulo.

Capítulo 3

Detección de malware mediante aprendizaje automático: Implementación

3.1. Temática elegida y descripción del problema

Como hemos visto en secciones anteriores el malware como amenaza a la ciberseguridad tiene el potencial de impactar los tres componentes de la tríada CIA.

El 83 % del nuevo malware producido en el primer trimestre de 2020 tenía como objetivo sistemas Windows [53]. Este dato no es de extrañar si tenemos en cuenta que la cuota de mercado de este sistema operativo ha sido superior al 75 % durante toda la década de 2010 [23].

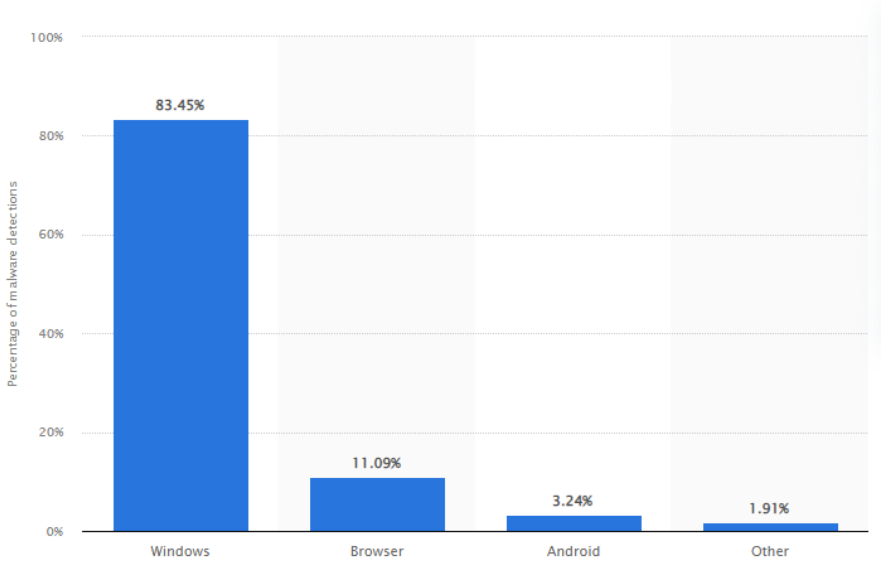


Figura 3.1: Distribución de malware según sistema operativo. Fuente: [53]

Por otro lado, tenemos que dentro de Windows el malware se localiza fundamentalmente

en los archivos ejecutables [62]. Estos motivos: la relevancia del malware como amenaza a la ciberseguridad y que el principal origen de nuevo malware se tenga en archivos ejecutables de Windows, justifican que nos centremos en la detección de malware en el sistema Windows en archivos ejecutables.

Por tanto, el problema central que trataremos en la sección práctica del Trabajo de Fin de Máster es el de la clasificación binaria de archivos ejecutables de Windows. Es decir, construiremos un modelo, usando algunos de los algoritmos de aprendizaje automático vistos en la sección 2.2.2, capaz de clasificar archivos ejecutables del sistema operativo Windows como benignos o maliciosos.

Como se mencionó en la sección 2.3.5, al discutir por qué se debía hacer una implementación ad-hoc, el objetivo no será solo construir este modelo, sino implementar una aplicación con una interfaz de usuario que permita tanto clasificar un archivo ejecutable como maligno o benigno, como también procesar y clasificar archivos con características ya extraídas. Esta última funcionalidad se trata de un elemento que no está presente en la mayoría de las soluciones existentes.

En la realización de esta implementación deben seguirse una serie de pasos o fases que serán dependientes unos de otros y permitirán obtener la aplicación implementada como producto final. A muy alto nivel podemos mencionar fases como la elección de un lenguaje de programación, que será un paso que condicionará el resto de la implementación, o la identificación de diferentes fuentes de información y la extracción de los datos obtenidos de las mismas. En una siguiente fase, más relacionada con la aplicación en sí, será necesario implementar funciones o métodos en el lenguaje de programación elegido que permitan extraer características de archivos ejecutables de Windows. Estas características deberán ser analizadas y se deberán hacer tareas de procesamiento y limpieza de datos si procede.

Posteriormente, ya estaremos en condiciones de centrarnos en el entrenamiento de distintos algoritmos de aprendizaje automático simples, y en base a los resultados obtenidos pasar a métodos más complejos como ensemble.

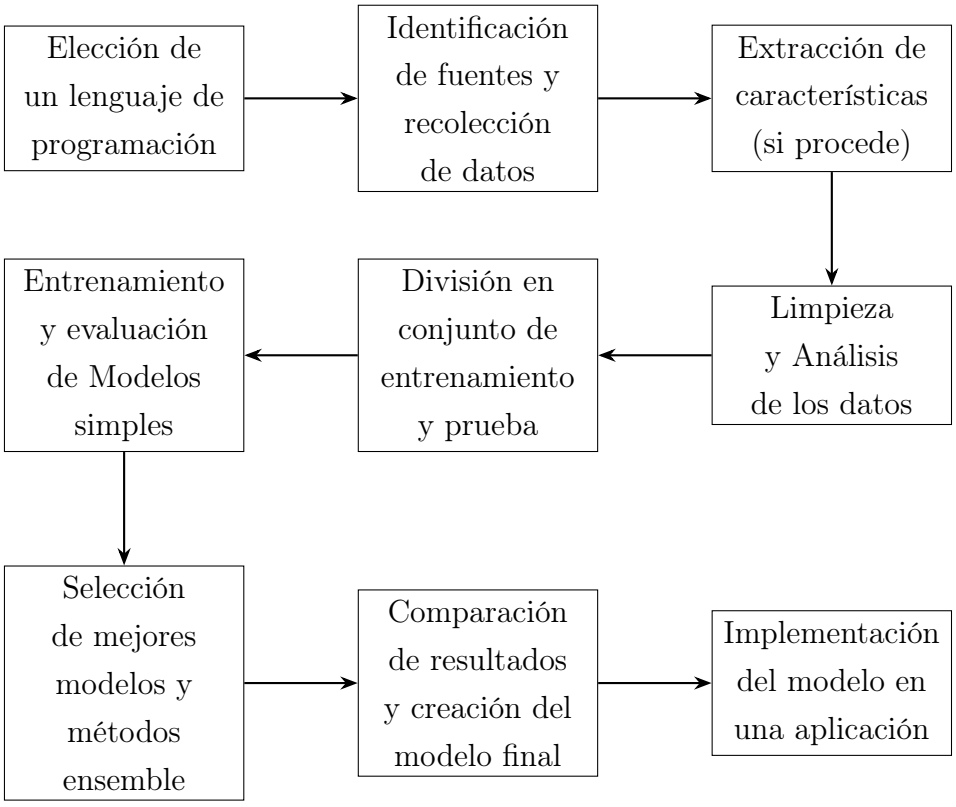
Por último, solo quedaría construir el modelo final con los mejores algoritmos (luego de un análisis comparativo de los resultados) y desarrollar la aplicación donde este modelo será usado.

Debemos también tener en cuenta que dado que estaremos construyendo una suerte de aplicación antivirus no debemos considerar exclusivamente la minimización de los falsos negativos (aquellos archivos maliciosos que son clasificados erroneamente como benignos), sino también controlar que el número de falsos positivos no sea muy alto, de otra forma la disponibilidad del sistema, uno de los pilares de la ciberseguridad, estaría siendo comprometida. Esto es así porque un sistema de detección de malware debe analizar mayoritariamente elementos benignos en su día a día, por lo que al clasificar un porcentaje elevado de tales archivos como maliciosos

las medidas que se tomen para eliminarlos o bloquearlos estarían comprometiendo el correcto funcionamiento del sistema en cuestión. Esto será de particular relevancia en la elección de las métricas y de los algoritmos que integrarán nuestro modelo final.

Esta descripción a muy alto nivel será analizada en detalle en las siguientes secciones.

Los distintos pasos a seguir en la parte práctica se listan a continuación:



3.2. Elección del lenguaje de programación

La primera tarea, antes de buscar fuentes de datos o elegir algoritmos, es la selección del lenguaje de programación o plataforma que será utilizado el resto del trabajo. Durante la toma de esta decisión, es crucial considerar diversos factores como la disponibilidad de librerías o métodos que faciliten las distintas tareas que vamos a realizar, disponer de implementaciones de los distintos algoritmos de aprendizaje automático que vamos a probar, así como disponer de un lenguaje debidamente documentado y con una comunidad amplia que permita consultar y resolver las distintas incidencias que puedan surgirs.

En este punto tenemos tres opciones principales:

- **Weka (Waikato Environment for Knowledge Analysis)** se trata de una colección de algoritmos de aprendizaje automático enfocados principalmente a tareas de minería de

datos. Se encuentra implementada sobre Java y al disponer de una interfaz gráfica puede ser usado por usuarios sin conocimientos de este lenguaje de programación. Aunque ha sido usada en análisis de malware por algunos investigadores [17] resulta menos flexible que lenguaje de programación como Python o R, ofreciendo peor rendimiento [48], y en general una menor variedad de opciones.

- **R** es un lenguaje de programación y entorno de software normalmente usado en combinación con RStudio. Es una solución de código abierto muy popular en el campo de la estadística o en entornos académicos al disponer de una vasta colección de paquetes con implementaciones de un gran número de funciones de densidad y distribución o tests de hipótesis. R dispone también de Shiny, un módulo para el desarrollo de aplicaciones web, que permite crear interfaces de usuario sin necesidad de recurrir a otros lenguajes o plataformas. Sin embargo, R tiene una sintaxis menos intuitiva que Python y una curva de aprendizaje más pronunciado, y ha ido siendo desplazado progresivamente de áreas como el análisis y la ciencia de datos [66].

- **Python** es un lenguaje de programación de propósito general conocido por su simplicidad y legibilidad del código. Dispone de un variado ecosistema de librerías mantenido por una comunidad muy activa. Por ejemplo, para la extracción de características de archivos ejecutables de Windows, Python dispone de la biblioteca *pefile*, que permite realizar este proceso de forma muy sencilla. Este lenguaje también destaca por sus potentes capacidades para la visualización y limpieza de datos.

  Pero es en el área del aprendizaje automático donde Python sobresale sobre el resto de opciones, con librerías como scikit-learn que ofrece implementaciones de un gran número de algoritmos de aprendizaje automático, o Tensorflow enfocada al deep learning y optimizada para ser usada sobre tarjetas gráficas. La integración de Python con Google Colab es otro aspecto a destacar. Esta plataforma ofrece recursos de computación gratuitos, incluyendo la computación en GPU, lo que agiliza significativamente el entrenamiento de modelos de aprendizaje automático. Python también ofrece en Streamlit una alternativa a Shiny de R, por lo que no es necesario recurrir a otros lenguajes para desarrollar aplicaciones web simples. De esta forma, podemos crear interfaces de usuario interactivas que permitan aplicar los modelos construidos a casos reales.

Por todo lo descrito anteriormente, Python se convierte en la opción escogida para realizar la implementación que será presentada en este trabajo. Por tanto, en las siguientes secciones se presentarán las librerías de este lenguaje usadas. Pero antes de ello es necesario introducir el sistema de archivos ejecutables de Windows que en combinación con una librería de Python serán claves en la creación del conjunto de datos con el que entrenaremos nuestros algoritmos.

3.3. Sistema PE Windows y librería pefile de Python

El formato de archivo Portable Executable (PE) es utilizado en los sistemas operativos Windows por archivos ejecutables (.EXE) y librerías de enlace dinámico (.DLL).

El formato PE, crucial para la carga y ejecución de aplicaciones y módulos del sistema en Windows, contiene información sobre la estructura del programa, las secciones de código y datos, la asignación de memoria, las funciones exportadas e importadas, y la información necesaria para la inicialización y terminación del programa entre otras [62].

Un archivo PE comienza por una estructura llamada DOS Header, seguida de un bloque de código conocido como DOS Stub. El DOS Header y el DOS Stub están presentes por razones de retrocompatibilidads.

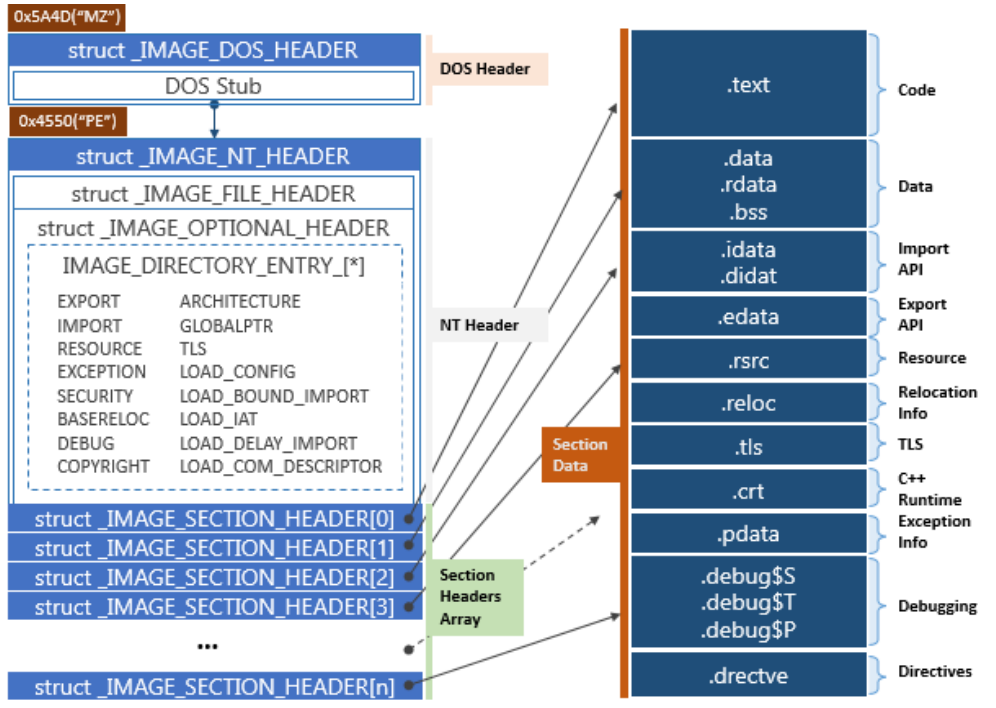


Figura 3.2: Formato de archivos ejecutables de Windows. Fuente [67]

Después del DOS Stub, se encuentra el PE Header, que incluye información clave sobre el archivo, como el tipo de máquina objetivo, el número de secciones, marcas temporales y punteros a otras estructuras dentro del propio archivo. Por ejemplo, aquí se indica la dirección de memoria virtual preferida para cargar el archivo, o el tamaño total en memoria del archivo una vez cargado. Dentro del PE Header tenemos la firma del archivo, el IMAGE_FILE_HEADER y el IMAGE_OPTIONAL_HEADER, esta última sección, a pesar de su nombre, es esencial para la ejecución de archivos pues es leída por el sistema operativo para poder hacer la carga en memoria ya que contiene información sobre la dirección de memoria por donde Windows

comenzará ejecutando el programa (*AddressOfEntryPoint*), o el tamaño en memoria ocupado por el archivo (*SizeOfImage*). Aquí también se definen las funciones importadas o exportadas por lo que puede dar información sobre la presencia de malware.

Las siguientes secciones se engloban dentro del *Section Headers Array* pudiendo ser de tipo *.text* que se trata de código ejecutable, de tipo *.data* que define datos y variables o *.rsrc* que contiene recursos como iconos usados por el archivo.

El formato PE es esencial para el correcto funcionamiento del sistema operativo Windows y alo largo de los años ha sido objeto de numerosas extensiones y mejoras para adaptarse a nuevas necesidades, aunque siempre intentando garantizar la retrocompatibilidad.

La librería pefile es un módulo de Python [14] que proporciona una forma sencilla de acceder y analizar archivos ejecutables portátiles (Portable Executable files o PE) y que será utilizada en este trabajo en la tarea de extraer características de archivos PE de Windows.

Esta librería permite extraer de forma rápida información de los archivos PE con el fin de generar las características que serán usadas posteriormente en los algoritmos de aprendizaje automático o en análisis estático.

3.4. Plataformas y librerías utilizadas en la implementación

El entrenamiento de los algoritmos y la mayor parte de la manipulación de datos se realizaron utilizando la plataforma Google Colab. Esta plataforma proporciona un entorno de ejecución en la nube que permite el uso de GPU de forma gratuita, facilitando así el proceso de entrenamiento de modelos de aprendizaje automático. Además, al trabajar en una máquina virtual, se evita cualquier riesgo derivado de manipular software malicioso.

Respecto a las librerías de Python usadas, tenemos la siguiente lista:

- pefile: permite analizar y manipular archivos ejecutables de Windows Portable Executable (PE) para extraer información relevante.
- sklearn (scikit-learn): proporciona una amplia variedad de algoritmos de aprendizaje automático, herramientas de preprocesamiento y métricas de evaluación.
- pandas: facilita la manipulación y análisis de datos estructurados, especialmente a través de DataFrames.
- matplotlib: librería de gráficos 2D para la generación de gráficos y visualizaciones en Python.

- numpy: proporciona soporte para trabajar con vectores y matrices, así como funciones matemáticas de alto nivel.
- tensorflow: librería de código abierto para aprendizaje automático que soporta paralelismo tanto en CPU's como GPU's. Es particularmente útil en el entrenamiento de redes neuronales y Deep Learning.
- keras: se trata de una librería de alto nivel enfocada a la creación de redes neuronales. Está enfocada al usuario, por lo que requiere de menos código. En nuestro caso, estaremos usando keras sobre tensorflow.
- XGBoost: es una librería con implementaciones de algoritmos de refuerzo de gradiente (Gradient Boosting), diseñada para ser altamente eficiente y flexible. Posee soporte para computación paralela
- pickle: se trata de una librería de Python que implementa protocolos para serializar y deserializar la estructura de un objeto en Python. Es útil por ejemplo para almacenar modelos entrenados, para que puedan ser usados en otra aplicación sin tener que volver a ser entrenados.
- Streamlit: es una herramienta para crear aplicaciones web en Python de forma rápida y sencilla. Permite el despliegue de modelos de aprendizaje automático o de análisis de datos tanto para compartir resultados como para ofrecer servicios.

3.5. Fuentes de datos utilizadas

A pesar de existir numerosas plataformas que se ocupan de la clasificación del malware, no abundan las bases de datos públicas que puedan ser utilizadas en el entrenamiento del algoritmos de aprendizaje automático.

En este trabajo se han utilizado básicamente 4 fuentes:

- Dataset de Kaggle: este dataset, disponible en https://www.kaggle.com/datasets/amauricio/pe-files-malwares, contiene un conjunto características extraídas usando la librería pefile a partir de archivos ejecutables de Windows y sus etiquetas
- Repositorio de GitHub: el repositorio, disponible en https://github.com/Kiinitix/malware-Detection-using-Machine-learning , proporciona otro conjunto de datos de archivos ejecutables etiquetados como malware o benignos

- Repositorio de Github: este repositorio descrito en [81] y disponible en https://github.com/da-proj/pe-malware-dataset1 Contiene muestras extraídas de VirusTotal y clasificadas según su familia de malware. Puede ser usado para clasificación multiclase, aunque en este trabajo nos limitaremos al caso binario.

- Fuente propia: consiste en archivos ejecutables (.exe y .dll) extraídos de un ordenador personal con Windows 10. Esta fuente contiene exclusivamente software benigno y se utiliza para equilibrar los otros conjuntos de datos que están desbalanceados en favor de la clase malware.

  En la recolección de los datos disponibles online se utilizó el cuaderno de Jupyter llamado *2_TFM_data_preparation_analysis.ipynb* que se puede encontrar en el repositorio Github del TFM. Por otro lado, para reunir los archivos de la fuente propia y extraer sus características, se desarrollaron dos scripts de Python, basados en la documentación de la librería pefile [14] y en [62]. El primero de estos scripts explora carpetas de Windows y reune los archivos ejecutables, mientas que el otro, utilizando la librería pefile, extrae características de los archivos y genera un archivo CSV. Este segundo script puede ser utilizado también para extraer características de archivos maliciosos (la única condición es que se trate de ejecutables de Windows). Ambos archivos se pueden encontrar en la carpeta other_scripts/ del repositorio del TFM.

  Los datos de las fuentes anteriores fueron guardados en formato CSV en una carpeta de Google Drive. Almacenar los archivos en Google Drive facilita su acceso y manipulación dentro de Google Colab, sin necesidad de descargarlos y cargarlos manualmente cada vez que se inicie una nueva sesión. Así evitamos el uso de la API de Kaggle, que obliga a regenerar la clave de acceso diariamente.

3.6. Preprocesamiento y análisis de los datos extraídos

Como primer paso en la preparación de los datos, hay que analizar el número de registros y columnas en cada conjunto de datos:

- Primer conjunto de datos: 19,611 filas y 79 columnas.
- Segundo conjunto de datos: 138047 filas y 57 columnas.
- Tercer conjunto de datos: 18551 filas y 54 columnas
- Cuarto conjunto de datos (solo software benigno): 6,209 filas y 34 columnas.

	Column	Dataset1_dtype	Dataset2_dtype	Dataset3_dtype	Dataset4_dtype
0	MinorOperatingSystemVersion	int64	int64	int64	int64
1	BaseOfCode	int64	int64	int64	int64
2	SizeOfOptionalHeader	int64	int64	int64	int64
3	SizeOfHeapCommit	int64	int64	int64	int64
4	CheckSum	int64	int64	int64	int64
5	MinorImageVersion	int64	int64	int64	int64
6	MinorLinkerVersion	int64	int64	int64	int64
7	MajorSubsystemVersion	int64	int64	int64	int64
8	NumberOfRvaAndSizes	int64	int64	int64	int64
9	MajorLinkerVersion	int64	int64	int64	int64
10	SizeOfUninitializedData	int64	int64	int64	int64
11	MinorSubsystemVersion	int64	int64	int64	int64
12	SizeOfStackReserve	int64	int64	int64	int64
13	MajorOperatingSystemVersion	int64	int64	int64	int64
14	SizeOfHeapReserve	int64	int64	int64	int64
15	SizeOfCode	int64	int64	int64	int64
16	AddressOfEntryPoint	int64	int64	int64	int64
17	LoaderFlags	int64	int64	int64	int64
18	ImageBase	int64	float64	float64	int64
19	SizeOfInitializedData	int64	int64	int64	int64
20	MajorImageVersion	int64	int64	int64	int64
21	SectionAlignment	int64	int64	int64	int64
22	SizeOfHeaders	int64	int64	int64	int64
23	Subsystem	int64	int64	int64	int64
24	Characteristics	int64	int64	int64	int64
25	Malware	int64	int64	int64	int64
26	Machine	int64	int64	int64	int64
27	SizeOfImage	int64	int64	int64	int64
28	FileAlignment	int64	int64	int64	int64
29	DllCharacteristics	int64	int64	int64	int64

Figura 3.3: Categorías comunes y tipos de datos

Dado que estos conjuntos de datos tienen diferente número de columnas se hace necesario identificar las características (columnas) comunes a los cuatro datasets antes de combinarlos, ya que solo estas características podrán ser utilizadas para entrenar los algoritmos. Tenemos que 30 columnas son comunes a los cuatro conjuntos de datos. Al explorar los tipos de datos, se aprecia que son iguales (figura: 3.3) o compatibles para todas las columnas, lo que facilitará la unión de los datasets.

Una vez unidos los cuatro conjuntos de datos y eliminados los registros duplicados (basándonos en la combinación de campos *combined_key*, *SizeOfCode*, *AddressOfEntryPoint* y *CheckSum*) , obtenemos un total de 133796 registros únicos. Si analizamos la distribución de las etiquetas tenemos que el 38.4 % de los datos corresponden a archivos benignos, mientras que el 61.6 % restante se trata de malware (figura: 3.4). También tenemos que el Dataset 2 representa más del 70 % del total de datos. Esto debe ser solucionado antes de entrenar modelos, pues de otra forma estarían sesgados hacia este conjunto que está sobrerrepresentado. Lo que haremos será quedarnos con 10000 elementos tomados al azar de cada clase del Dataset 2.

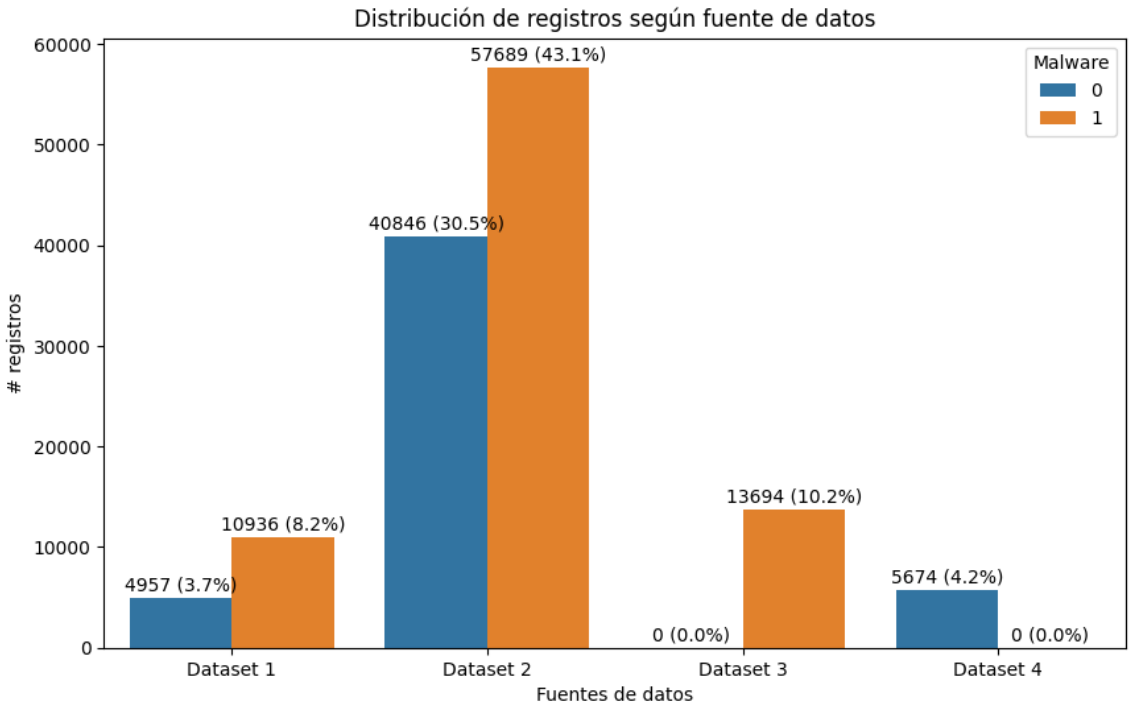


Figura 3.4: Distribución por fuente y etiqueta

La distribución de clases vista anteriormente (un 38 % vs 62 %) debe ser tenida durante el proceso de entrenamiento y validación de los modelos de aprendizaje automático, ya que tal y como hemos visto algunos modelos son sensibles a desequilibrioss entre las clases (imbalance). Para el conjunto de datos 2 ya hemos solucionado este problema, pero debemos hacer lo mismo para el resto de datasets. Se podrían aplicar técnicas como el sobremuestreo de la clase minoritaria que consiste en duplicar algunos valores de la clase minoritaria o generar nuevos ejemplos sintéticos a partir de los existentes, para de esta forma equilibrar la distribución de las clases antes de entrenar los modelos.

En lugar de tratar el desequilibrio de las clases con alguna de las técnicas antes mencionadas, lo que haremos será podar la clase mayoritaria (malware) hasta tener el mismo número de registro que en la otra (software benigno) Aunque perdamos información seguimos contando con una cantidad de registros superior a la utilizada en trabajos similares. Además, esta estrategia acelerará la ejecución de los algoritmos de aprendizaje automático usados, dado que muchas implementaciones no aprovechan las GPU's disponibles en Google Colab y por tanto el entrenamiento con un conjunto de datos grande resulta costoso en términos de tiempo y recursos computacionales.

Eligiendo 20600 registros al azar para cada una de las clases obtenemos un conjunto de datos balanceado que será utilizado en las siguientes fases del trabajo. La distribución de datos según fuente, como se puede ver en 3.5 es más equitativa que en el conjunto de datos combinado

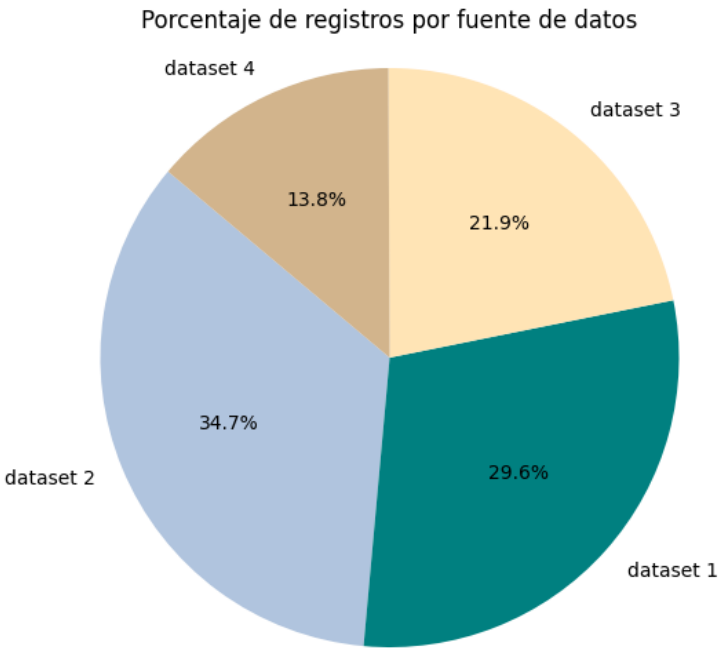


Figura 3.5: Distribución por fuente actualizada

obtenido originalmente, lo que ayudará a que las contribuciones de cada una de estas fuentes en los modelos sea similar.

El nuevo conjunto de datos se guarda en un archivo csv en Google Drive para poder ser utilizado fácilmente en otros cuadernos de Colab. Todo el preprocesamiento y análisis de datos, así como la creación y exportación del archivo csv resultante, se ha hecho usando el cuaderno de Jupyter, *2_ TFM_ data_ preparation_ analysis.ipynb* disponible en el repositorio del TFM.

3.7. Entrenamiento de modelos

3.7.1. Métricas de evaluación

Para evaluar el rendimiento de los modelos que implementaremos, se han seleccionado cuatro métricas que consideramos relevantes en la tarea de detección de malware: precisión, exactitud, F_1 y sensibilidad (recall). A continuación, se describen brevemente estas métricas así como su definición:

- Precisión: Indica la proporción de verdaderos positivos entre todos los ejemplos clasificados como positivos. Es decir, que su valor será máximo cuando el número de falsos positivos sea mínimo, por lo que en nuestro caso es útil para construir un modelo que minimice el número de software benigno que es incorrectamente clasificado como malware

- Exactitud (accuracy): Mide la proporción de predicciones correctas (verdaderos positivos y verdaderos negativos) en relación con el total de predicciones.
- Sensibilidad (recall): Representa la proporción de verdaderos positivos entre la suma de verdaderos y falsos positivos. Por tanto es importante cuando buscamos minimizar los falsos negativos que equivalen a que nuestros modelos clasifiquen malware como software benigno.
- F1 esta métrica se define como la media armónica entre la precisión y la sensibilidad. Dado que en nuestro caso buscamos reducir falsos y verdaderos positivos al mismo tiempo este será el indicador principal a la hora de evaluar nuestros algoritmos.

3.7.2. Algoritmos lineales

Comenzamos trabajando con dos algoritmos lineales: Análisis Discriminante Lineal (LDA) y Perceptrón, en ambos casos se han evaluado tanto sobre los datos originales como sobre datos proyectados mediante PCA (mantenemos un 95 % de los componentes principales). Dado que son algoritmos con un coste computacional bajo pueden servirnos para entender fácilmente si nuestros datos son o no linealmente separables. En estos algoritmos, así como en el PCA, se han utilizando las implementaciones disponibles en la librería scikit-learn de Python.

Aunque se analizarán los resultados con mayor detalle en la comparativa de los distintos algoritmos, hemos observado que el Perceptrón con PCA ha sido ligeramente superior a los otros métodos en todas las métricas.

3.7.3. Árboles de decisión y Máquinas de Vectores Soporte

Pasando a algoritmos más complejos se han implementado Árboles de Decisión y Máquinas de Vectores Soporte (SVM) sobre el conjunto de datos. Ambos algoritmos cuentan con múltiples parámetros que se deben optimizar para obtener un mejor rendimiento. Esto puede hacerse mediante búsqueda en cuadrícula (GridSearch), pero debido gran número de registros en el conjunto de datos, y a la cantidad de combinaciones posibles de los parámetros esto no es aconsejable pues entraña un alto coste computacional.. Por lo tanto, se ha optado por utilizar métodos heurísticos, específicamente, la búsqueda aleatoria sobre combinaciones de parámetros utilizando el método RandomizedSearchCV de scikit-learn. De esta manera, se puede encontrar una aproximación a la mejor combinación de parámetros sin pagar el coste computacional de hacer una búsqueda exhaustiva.

Se han obtenido resultados significativamente mejores con el Árbol de Decisión, lo cual es consistente con lo discutido en algunos artículos del estado del arte. Por otro lado, las

Máquinas de Vectores Soporte han requerido tiempos de entrenamiento más largos a la vez que los resultados obtenidos son comparables a los de los algoritmos lineales, esta combinación de factores hacen que sean menos apropiadas en la tarea de clasificación binaria de malware (al menos con el conjunto de datos utilizado).

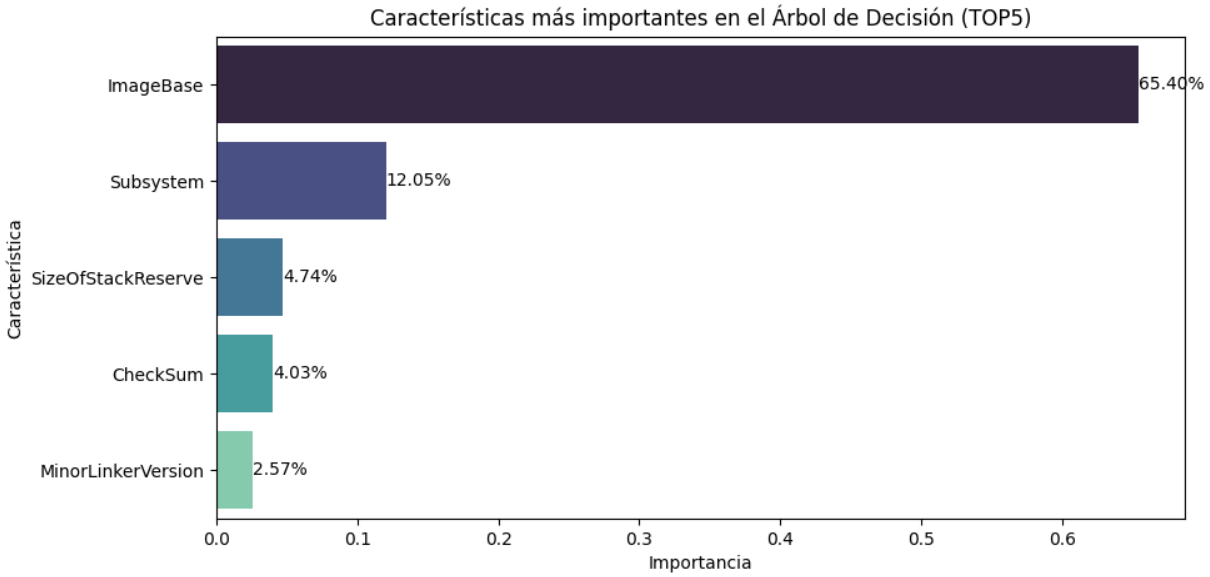


Figura 3.6: Características más importantes del árbol de decisión

Una ventaja adicional de los Árboles de Decisión es que permiten visualizar fácilmente su estructura, lo que facilita la interpretación y comprensión del modelo. Además, los Árboles de Decisión proporcionan una medida de la importancia de cada característica en el modelo, lo que puede ser útil para la selección de características y la mejora del rendimiento del modelo. Las cinco características más importantes de nuestro algoritmo pueden consultarse en 3.6 y como se puede apreciar solo las tres más importantes bastan para explicar más de un 80 % del árbol.

Tanto la implementación de los algoritmos lineales, como la de la máquina de vectores soporte y el árbol de decisión, se ha hecho en el tercero de los cuadernos de Jupyter que se pueden encontrar en el repositorio del trabajo: *3_ TFM_ models_ basic.ipynb*.

3.7.4. Métodos ensemble: Random forest

Existe la posibilidad de combinar la clasificación de distintos modelos mediante una votación. Es decir, cada modelo clasifica el mismo dato por separado y luego la opinión mayoritaria (o ponderada) se usa como valor final. Esta sería una forma básica de método ensemble definido a partir de modelos simples.

En nuestro caso, hemos usado los algoritmos Perceptrón, Máquina de Vectores Soporte y Árbol de Decisión, dándole más peso al árbol (1.5 frente 1 para el resto) de forma que solo un

voto unánime del Perceptrón y la Máquina de Vectores soporte puede anular una decisión del Árbol. Se ha utilizado la clase *VotingClassifier* de la librería sklearn.ensemble. Los resultados obtenidos mediante el método de votación son inferiores a los del Árbol de Decisión, salvo para la métrica precisión. Por tanto, podemos descartar esta aproximación y será necesario probar con métodos más complejos.

Random Forest es un método ensemble más avanzado que combina múltiples árboles, a diferencia de nuestro ejemplo de clasificador ensemble simple en Random Forest se pueden combinar cientos de árboles de decisión independientes para obtener un mejor resultado Un primera prueba usando los parámetros por defecto de la implementación disponible en sklearn.ensemble (*RandomForestClassifier*) mejoró los resultados del Árbol de Decisión en todas las métricas, lo que muestra la potencia de este método.

Parámetro	**Valor**
criterion	gini
min_samples_leaf	1
min_samples_split	2
n_estimators	300

Cuadro 3.1: Mejores parámetros para Random Forest

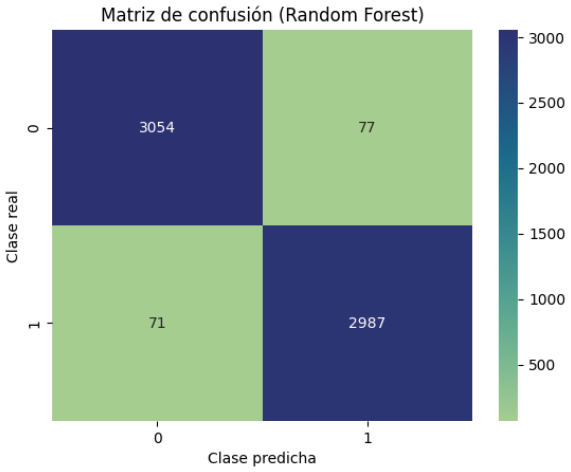


Figura 3.7: Matriz de confusión del Random Forest obtenido por optimización de hiperparámetros (conjunto de prueba)

Al usar hiperparámetros optimizados (tabla 3.1) mediante búsqueda aleatoria, de forma similar a como se hizo con el árbol de decisión, los resultados mejoran nuevamente respecto a los parámetros por defecto. La matriz de confusión obtenida con los mejores hiperparámetros se puede consultar en la figura 3.7. El único punto negativo a tener en cuenta es que la optimización de hiperparámetros requiere de mucho tiempo de computación, no así el proceso de

entrenamiento del algoritmo con los mejores parámetros.

3.7.5. Métodos ensemble: XGBoost

En 2.2.2 vimos otro método ensemble también basado en árboles de decisión, el Gradient Boosting. En este caso, a diferencia de Random Forest, los árboles se van generando secuencialmente y cada nuevo árbol intenta mejorar los resultados de sus predecesores. En la implementación se ha utilizado la librería xgboost.

Al igual que con Random Forest hacemos una primera prueba con los parámetros por defecto, obteniendo resultados ligeramente inferiores a los del Random Forest con parámetros no optimizados que entrenamos anteriormente. Repitiendo la metodología aplicada para los otros modelos no lineales, usamos búsqueda aleatoria para obtener mejores hiperparámetros (tabla 3.2), obteniéndose resultados similares a los del Random Forest optimizado.

Parámetro	**Valor**
subsample	1.0
n_estimators	1000
min_child_weight	1
max_depth	9
learning_rate	0.05
gamma	0
colsample_bytree	0.7

Cuadro 3.2: Mejores parámetros para XGBoost

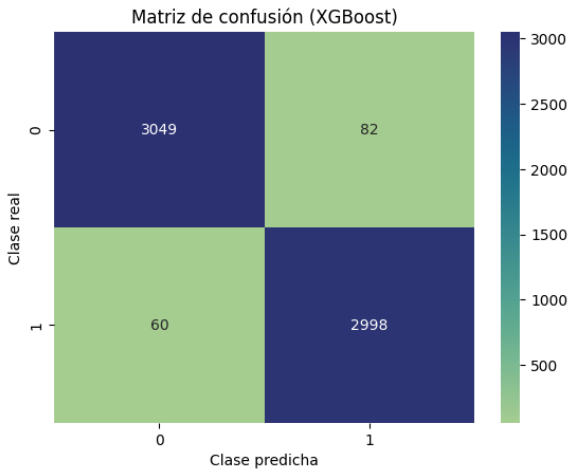


Figura 3.8: Matriz de confusión del algoritmo XGBoost obtenido por optimización de hiperparámetros (conjunto de prueba)

3.7.6. Redes neuronales

Siguiendo con la implementación de los distintos algoritmos vistos en la sección 2.2.2 nos queda por probar las redes neuronales en el conjunto de datos. En este caso tenemos disponibles varias librerías de Python con distintas implementaciones, por ejemplo en sklearn existe una implementación del perceptrón multicapa (*MLPClassifier*), o se podría construir una red neuronal ad-hoc combinando varios perceptrones como el que fue entrenado en esta sección. Sin embargo, dado que disponemos de GPU's en Google Colab, usaremos la librería keras (que se ejecuta sobre tensorflow) y que está especialmente diseñada para trabajar con redes neuronales usando computación paralela. Esta parte del código se encuentra en el quinto de los cuadernos de Google Colab: 5_TFM_NN.ipynb

La red neuronal construida consta de cinco capas: una primera capa oculta con 64 neuronas que admite datos con la dimensión del número de características de nuestro datraset, otras dos capas ocultas con 32 neuronas cada una, y una capa oculta adicional de 16 neuronas. Todas las capas ocultas usan la función de activación *relu*, que es de uso común ya que permite a la red aprender patrones complejos. Tenemos también, una capa de salida con una única neurona y función de activación *sigmoide* que permite que nuestra red devuelva un valor entre 0 y 1 que equivale a la probabilidad de que el dato a clasificar sea benigno o malicioso (0 o 1 respectivamente).

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_30 (Dense)            (None, 64)                1920

 dense_31 (Dense)            (None, 32)                2080

 dense_32 (Dense)            (None, 32)                1056

 dense_33 (Dense)            (None, 16)                528

 dense_34 (Dense)            (None, 1)                 17

=================================================================
Total params: 5,601
Trainable params: 5,601
Non-trainable params: 0
```

Figura 3.9: Estructura de la red neuronal

Sin embargo, los resultados obtenidos con la red neuronal son muy discretos, con valores en las métricas comparables a los del árbol de decisión simple implementado anteriormente. Este resultado confirma lo discutido en el marco teórico cuando se mencionó que para datos tabulares los mejores algoritmos o modelos eran aquellos basados en árboles.

3.8. Creación del Modelo final

En base a los resultados se ha optado por combinar el mejor Random Forest, con el mejor XGBoost, ambos obtenidos mediante optimización de hiperparámentros. Teniendo en cuenta nuestro objetivo de reducir tanto falsos positivos y falsos negativos, vamos a aplicar un enfoque algo distinto a lo hecho hasta ahora en la construcción del modelo final, que se puede consultar en el cuaderno: *6_ TFM_ MODELO_ FINAL.ipynb* del repositorio.

Al algoritmo XGBoost le cambiaremos su límite de decisión (*treshold*) de forma que obtengamos dos nuevos algoritmos *xgb_ fp_ pred* y *xgb_ fn_ pred*. El primero de estos algoritmos se centrará en reducir falsos positivos, por lo que solo clasificará un dato como malware si la probabilidad que devuelve es mayor a 0.75 (0.5 es el valor por defecto). El segundo algoritmo, que se centra en reducir falsos negativos, clasifica como malware todo valor con una probabilidad mayor a 0.1. Estos valores se han establecido teniendo en cuenta la proporción de falsos y verdaderos positivos en el XGBoost original.

Nuestro Modelo final devuelve el valor de XGBoost si un dato recibe la misma clasificación por parte de *xgb_ fp_ pred* y *xgb_ fn_ pred*, en otro caso, se usa la clasificación del Random Forest. De esta forma, los casos extremos para los que la decisión de XGBoost está clara, son clasificados por este algoritmo y los restantes por Random Forest.

Este modelo mejora los resultados obtenidos tanto con el mejor XGBoost como con el mejor Random Forest, para todas las métricas.

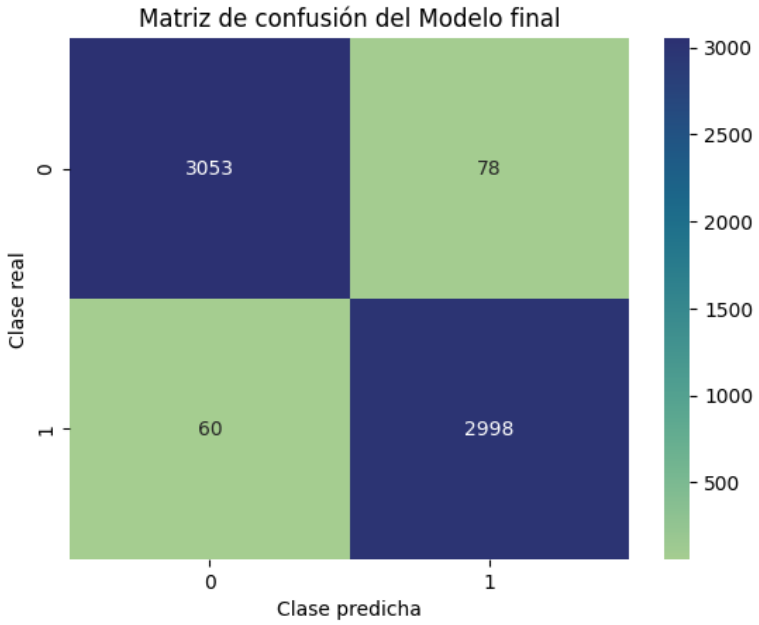


Figura 3.10: Matriz de confusión del Modelo final

3.9. Despliegue del Modelo: aplicación web

El paso siguiente a la implementación del modelo final es la creación de una aplicación donde se pueda utilizar este modelo para clasificar archivos como maliciosos o benignos. Para ello, primero debemos exportar nuestro modelo (en este caso los algoritmos XGBoost y Random Forest ya entrenados) y definir nuestro modelo como una función de Python. En la exportación de objetos de Python resulta útil la librería pickle, que permite exportar en nuestro caso los algoritmos entrenados como archivos. Los modelos exportados se guardan en la carpeta models/.

La aplicación web se ha implementado usando la librería Streamlit. Las aplicaciones en esta librería se definen como un script de Python estándar. Cada vez que el usuario realiza una acción, o si la página web es refrescada, el código del script es ejecutado desde el principio, esto se debe tener en cuenta, porque cualquier objeto que no haya sido previamente guardado en otra estructura de datos se pierde con cada acción del usuario (como cargar un archivo o presionar un botón).

Esta aplicación ha sido desplegada usando los servicios gratuitos que ofrece Streamlit y se encuentra disponible en https://glaria-tfm-malware-classifier-app-7qnhnt.streamlit.app/. Cuenta con una funcionalidad de carga de archivo que admite tanto archivos ejecutables de Windows como CSV con características. Para los CSV existe la opción de definir el separador de columnas (por defecto es la coma).

Para la extracción de las características de los archivos ejecutables se ha usado el script creado durante la extracción de datos para crear el dataset 4 (datos benignos extraídos de un ordenador con Windows. El script original se puede encontrar en la carpeta other_scripts/ mientras que el código adaptado a la aplicación está disponible en feature_extraction.py

Esta funcionalidad de carga se encuentra limitada, debido a restricciones impuestas por Streamlit, a archivos de un máximo de 200 MB. Aunque es posible eliminar esta restricción, durante las pruebas de la aplicación no ha sido necesario hacerlo, así que se ha preferido mantener este límite para evitar posibles errores derivados de procesar archivos grandes.

Figura 3.11: Pantalla de inicio de la aplicación

Una vez cargado los archivos estos son procesados y se muestran por pantalla las caracterís-

ticas extraídas, que pueden ser consultadas por el usuario usando la barra de desplazamiento horizontal.

Figura 3.12: Carga de CSV

El usuario puede hacer click en el botón clasificar (si hace click sin haber cargado ningún archivo recibe un aviso por pantalla) Dependiendo de si se trata de un ejecutable o un CSV el resultado es diferente. En el caso de los CSV, si falta alguna de las columnas usadas por los modelos, se muestra un aviso por pantalla junto con esas columnas (figuras 3.12 y 3.13). A los campos faltantes se les asigna el valor cero por defecto.

Figura 3.13: Lista de columnas no disponibles

Una vez procesado el archivo CSV por el modelo, se muestran por pantalla las primeras filas del archivo original con una columna extra: malware_fl que vale 1 si el dato de esa fila se ha clasificado como malware y 0 en otro caso. También se incluye un enlace que permite descargar el nuevo CSV (con la columna añadida) en el terminal del usuario (figura 3.14)

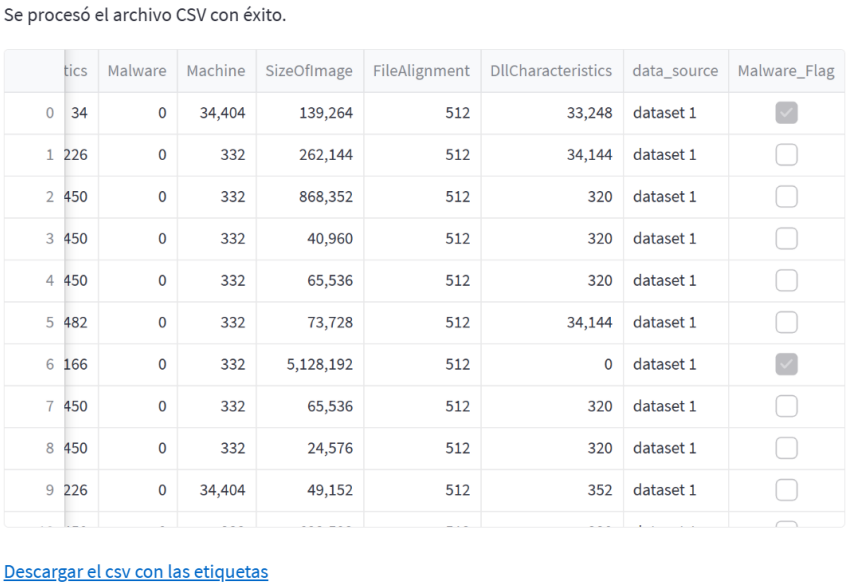


Figura 3.14: CSV clasificado

En el caso de los ejecutables, una vez clasificados simplemente se muestra si el archivo es malicioso o benigno con un mensaje en pantalla (figura 3.15)

Figura 3.15: Archivo ejecutable benigno

Esta aplicación, aunque simple en cuanto a funcionalidades, permite interactuar y probar nuestro modelo de forma sencilla y rápida. Al permitir trabajar con características pre-extraídas, viene solucionar el problema de la ausencia de clasificadores que vayan más allá del procesamiento de archivos ejecutables.

3.10. Resultados

Los resultados de los diferentes modelos para las métricas usadas se pueden consultar en la tabla 3.3.

Tal y como se discutió en las secciones anteriores se observa como la optimización de hiperparámetros es una tarea siempre deseable, aunque el alto coste computacional que implica debe ser tenido en cuenta.

Otro punto importante es la relevancia del PCA para mejorar los algoritmos lineales, y en general el buen desempeño que estos muestran, lo que puede ser un indicativo de la calidad (o simplicidad) de nuestro conjunto de datos.

También hay que destacar el hecho que algoritmos complejos y con un elevado coste computacional como las Máquinas de Vectores Soporte o Redes Neuronales muestren un desempeño pobre en comparación con todos aquellos basados en árboles de decisión.

Estos resultados validan la decisión de utilizar XGBoost combinado con Random Forest en el modelo final, e igualmente confirman que la aproximación usada en este modelo es positiva.

Modelo	**F1 Score**	**Accuracy**	**Precision**	**Recall**
Perceptron sin PCA	0.8305	0.8240	0.7926	0.8721
LDA sin PCA	0.8449	0.8347	0.7877	0.9111
Perceptron con PCA	0.8678	0.8601	0.8139	0.9294
LDA con PCA	0.8472	0.8366	0.7877	0.9163
Árbol de decisión	0.9660	0.9664	0.9663	0.9657
Máquina de vectores soporte	0.8940	0.8929	0.8746	0.9143
Método ensemble básico (votación)	0.9500	0.9520	0.9660	0.9350
Random forest (parámetros por defecto)	0.9752	0.9754	0.9745	0.9758
Random forest (mejores hiperparámetros)	0.9758	0.9761	0.9749	0.9768
XGBoost (parámetros por defecto)	0.9587	0.9591	0.9573	0.9601
XGBoost (mejores hiperparámetros)	0.9769	0.9771	0.9734	0.9804
Red neuronal	0.9137	0.9180	0.9588	0.8727
Modelo final (XGBoost + Random Forest)	0.9775	0.9777	0.9746	0.9804

Cuadro 3.3: Tabla de F1 Score, Exactitud (Accuracy), Precisión y Recall de los modelos (conjunto de prueba).

Queda un último punto y es comprobar como escala el modelo final, respecto a los tiempos de ejecución con diferentes tamaños muestrales, tanto en la fase de entrenamiento como durante las predicciones. Esto es relevante pues un modelo con un comportamiento exponencial, puede no ser el más adecuado dado que obligará a trabajar con conjuntos de datos acotados. Como puede verse en las figuras 3.16 y 3.17 la tendencia del modelo es lineal, por lo que no tenemos elementos que indiquen posibles problemas de escalabilidad.

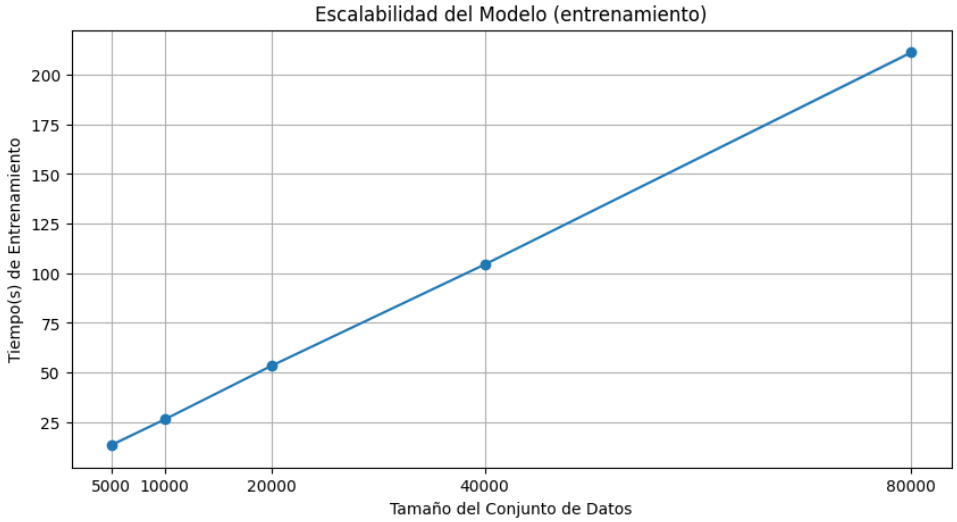


Figura 3.16: Escalabilidad del Modelo para datos durante el entrenamiento

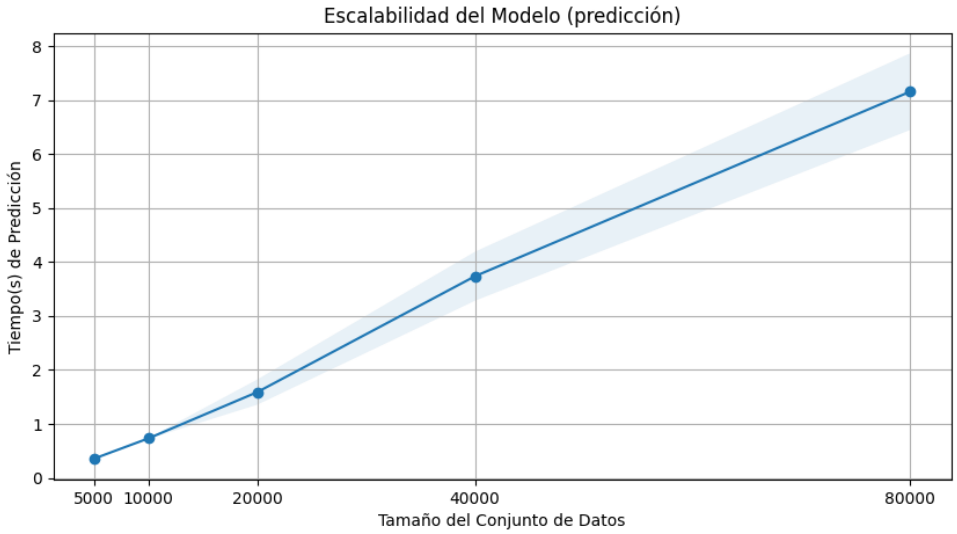


Figura 3.17: Escalabilidad del Modelo en producción

Capítulo 4

Conclusiones y Trabajo futuro

4.1. Conclusiones

Los diferentes objetivos establecidos al inicio del trabajo han sido conseguidos:

- Se ha hecho un estudio sobre la relevancia cada vez mayor de la ciberseguridad y de como el malware, en sus distintas formas y categorías, constituye una de las amenazas más importantes que enfrenta la ciberseguridad. En este sentido, se hizo una revisión de diferentes métodos clásicos de detección de malware, apreciándose sus ventajas y limitaciones, y la necesidad de recurrir a técnicas más sofisticadas como aquellas proporcionadas por el aprendizaje automático

- Dentro del campo del aprendizaje automático, se han analizado diferentes algoritmos, distintas formas de clasificarlos y sus aplicaciones dentro de la ciberseguridad. Estas aplicaciones han sido estudiadas no solo desde un punto de vista académico, con la revisión de diversas publicaciones, sino también desde un enfoque más práctico a través de la elaboración de una lista de múltiples aplicaciones comerciales que hacen uso de tales técnicas.

- Han sido identificadas diferentes fuentes de datos etiquetadas con las características ya extraídas. Igualmente se ha desarrollado código para extraer características de archivos ejecutables de Windows que ha sido utilizada para enriquecer el conjunto de datos creado.

- Los diferentes algoritmos analizados han sido entrenados en la tarea de la clasificación binaria de archivos ejecutables usando el conjunto de datos construido. A partir de estas implementaciones ha sido posible hacer un estudio comparativo en base a varias métricas como *f1 score* o *exactitud*, entre otras, evaluando el rendimiento con diferentes hiperparámetros.

- Se ha concluido que para el conjunto de datos utilizado los algoritmos XGBoost y Random Forest, construidos a partir de la optimización de hiperparámetros, son los que muestran un mejor desempeño. Por este motivo han sido utilizados en la construcción de un modelo adhoc diseñado con el objetivo de minimizar el número de falsos positivos y negativos.
- Este modelo ha sido implementado y desplegado en una aplicación web lo que permite clasificar tanto archivos ejecutables como tablas con características ya extraídas.

4.2. Trabajo futuro

Pueden establecerse varias líneas de trabajo en base a lo desarrollado en este TFM. Entre ellas caben señalar:

- Sería conveniente probar el modelo con otros conjuntos de datos extraídos de diferentes fuentes. También resultaría interesante medir otras métricas como *logloss* o *ROC-AUC* que no han sido consideradas en este trabajo
- En el modelo final se han usado dos algoritmos basados en árboles de decisión, podría ser relevante probar algoritmos de otras familias y comprobar si mejoran el desempeño.

  Una línea de investigación interesante es la desarrollada en [68] donde primero transformaron las muestras de archivos ejecutables a imágenes en escala de grises y luego usaron redes neuronales como clasificador.
- En el entrenamiento de los distintos algoritmos, y del modelo final, se ha usado un número limitado de características, sería conveniente reentrenar este modelo incluyendo otras características extraídas de archivos ejecutables.
- Por otro lado, existen diversas mejoras que podrían hacerse en la aplicación web. La primera de ellas es usar alguna plataforma o lenguaje de programación específico del desarrollo de aplicaciones web, que le otorgue más flexibilidad y permita añadir nuevas opciones o simplemente mejorar el entorno gráfico.

  Otro punto en el que se podría trabajar en la aplicación sería la posibilidad de crear conexiones a bases de datos, de forma que se puedan almacenar los datos introducidos. De esta forma, también se podría entrenar continuamente el modelo en base a nuevos datos de entrada.

  Por último, se podría mejorar el apartado de clasificación de archivos CSV con características, de forma que si faltan campos el comportamiento del clasificador sea diferente y no asigne valores por defecto fijos para todas las columnas faltantes.

Bibliografía

[1] Trend Micro Business AI and machine learning. https://www.trendmicro.com/vinfo/ph/security/news/cybercrime-and-digital-threats/curbing-the-bec-problem-using-ai-and-machine-learning, 2018. [Online; accessed 21-May-2023].

[2] Mohammed S Alam and Son T Vuong. Random forest classification for detecting android malware. In *2013 IEEE international conference on green computing and communications and IEEE Internet of Things and IEEE cyber, physical and social computing*, pages 663–669. IEEE, 2013.

[3] Bitdefender What Is Dynamic Malware Analysis? https://www.bitdefender.com/blog/businessinsights/what-is-dynamic-malware-analysis/, 2023. [Online; accessed 7-May-2023].

[4] Hyrum S Anderson and Phil Roth. Ember: an open dataset for training static pe malware machine learning models. *arXiv preprint arXiv:1804.04637*, 2018.

[5] Cylance Smart Antivirus. https://www.blackberry.com/us/en/support/smartantivirus/, 2022. [Online; accessed 7-May-2023].

[6] Safety Detectives Cylance Smart Antivirus. https://es.safetydetectives.com/best-antivirus/cylance-smart-antivirus/, 2022. [Online; accessed 7-May-2023].

[7] John Aycock. *Computer Viruses and Malware*, volume 22. 01 2006.

[8] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, 1 edition, 2007.

[9] Jean-Marie Borello and Ludovic Mé. Code obfuscation techniques for metamorphic viruses. *Journal in Computer Virology*, 4(3):211–220, 2008.

[10] Trend Micro Business. https://www.trendmicro.com/vinfo/us/security/news/security-technology/faster-and-more-accurate-malware-detection-through-predictive 2019. [Online; accessed 21-May-2023].

[11] Trend Micro Business. https://www.trendmicro.com/vinfo/us/security/definition/machine-learning, 2022. [Online; accessed 21-May-2023].

[12] Trend Micro Business. https://www.trendmicro.com/es_es/business/products/network/advanced-threat-protection/analyzer.html, 2022. [Online; accessed 21-May-2023].

[13] Conceptos básicos de ayuda de CRISP-DM. https://www.ibm.com/docs/es/spss-modeler/saas?topic=dm-crisp-help-overview, 2021. [Online; accessed 10-April-2023].

[14] Ero Carrera. https://github.com/erocarrera/pefile/, 2023. [Online; accessed 28-March-2023].

[15] Carlos Cepeda, Dan Lo Chia Tien, and Pablo Ordóñez. Feature selection and improving classification performance for malware detection. In *2016 IEEE International Conferences on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom) (BDCloud-SocialCom-SustainCom)*, pages 560–566, 2016.

[16] Jim X. Chen. The evolution of computing: Alphago. *Computing in Science & Engineering*, 18(4):4–7, 2016.

[17] Mozammel Chowdhury, Azizur Rahman, and Rafiqul Islam. Malware analysis and detection using data mining and machine learning classification. In *International conference on applications and techniques in cyber security and intelligence: applications and techniques in cyber security and intelligence*, pages 266–274. Springer, 2018.

[18] Fred Cohen. Computer viruses: theory and experiments. *Computers & security*, 6(1):22–35, 1987.

[19] CuckooML. https://honeynet.github.io/cuckooml/, 2016. [Online; accessed 21-May-2023].

[20] Cybersecurity and Infrastructure Security Agency What is Cybersecurity? https://www.cisa.gov/news-events/news/what-cybersecurity/, 2023. [Online; accessed 28-March-2023].

[21] Dipankar Dasgupta, Zahid Akhtar, and Sajib Sen. Machine learning in cybersecurity: a comprehensive survey. *The Journal of Defense Modeling & Simulation*, 19:2020, 09 2020.

[22] Dipankar Dasgupta, Zahid Akhtar, and Sajib Sen. Machine learning in cybersecurity: a comprehensive survey. *The Journal of Defense Modeling and Simulation*, 19(1):57–106, 2022.

[23] Rosa Fernández. https://es.statista.com/estadisticas/634540/sistemas-operativos-para-pc-cuota-de-mercado-mundial/, 2023. [Online; accessed 25-May-2023].

[24] Ivan Firdausi, Charles Lim, Alva Erwin, and Anto Nugroho. Analysis of machine learning techniques used in behavior-based malware detection. *Advances in Computing, Control, and Telecommunication Technologies, International Conference on*, 0:201–203, 12 2010.

[25] R. A. FISHER. THE USE OF MULTIPLE MEASUREMENTS IN TAXONOMIC PROBLEMS. *Annals of Eugenics*, 7(2):179–188, September 1936.

[26] European Union Agency for Cybersecurity. https://www.enisa.europa.eu/, 2022. [Online; accessed 7-May-2023].

[27] European Union Agency for Cybersecurity-Threat Landscape. https://www.enisa.europa.eu/publications/enisa-threat-landscape-2022, 2022. [Online; accessed 7-May-2023].

[28] Machine Learning for Malware Detection. https://media.kaspersky.com/en/enterprise-security/Kaspersky-Lab-Whitepaper-Machine-Learning.pdf, 2022. [Online; accessed 7-May-2023].

[29] Alexander L. Fradkov. Early history of machine learning. *IFAC-PapersOnLine*, 53(2):1385–1390, 2020. 21st IFAC World Congress.

[30] P Ganeshkumar and N Pandeeswari. Adaptive neuro-fuzzy-based anomaly detection system in cloud. *International journal of fuzzy systems*, 18:367–378, 2016.

[31] Aurélien Géron. *Hands-on machine learning with Scikit-Learn and TensorFlow : concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, Sebastopol, CA, 2017.

[32] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[33] Leo Grinsztajn, Edouard Oyallon, and Gael Varoquaux. Why do tree-based models still outperform deep learning on typical tabular data? In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022.

[34] Francis Guibernau and Ayelen Torello. 'catch me if you can!—detecting sandbox evasion techniques. *Proc. USENIX Assoc*, 2020.

[35] Eldar Haber and Tal Zarsky. Cybersecurity for infrastructure: a critical analysis. *Fla. St. UL Rev.*, 44:515, 2016.

[36] Anand Handa, Ashu Sharma, and Sandeep Shukla. Machine learning in cybersecurity: A review. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 9:e1306, 07 2019.

[37] Why Python is the King for Machine Learning. https://www.machinelearningmike.com/post/why-python-is-the-king-for-machine-learning, 2020. [Online; accessed 21-May-2023].

[38] EMBER Open issue. https://github.com/elastic/ember/issues/103, 2023. [Online; accessed 21-May-2023].

[39] Sainadh Jamalpur, Yamini Sai Navya, Perla Raja, Gampala Tagore, and G. Rama Koteswara Rao. Dynamic malware analysis using cuckoo sandbox. In *2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT)*, pages 1056–1060, 2018.

[40] CENTRO DE TRANSPARENCIA KASPERSKY. https://www.kaspersky.es/transparency-center-offices, 2023. [Online; accessed 7-May-2023].

[41] Ban Mohammed Khammas. Ransomware detection using random forest technique. *ICT Express*, 6(4):325–331, 2020.

[42] Joost N Kok, Egbert J Boers, Walter A Kosters, Peter Van der Putten, and Mannes Poel. Artificial intelligence: definition, trends, techniques, and cases. *Artificial intelligence*, 1:270–299, 2009.

[43] Rajesh Kumar and S Geetha. Malware classification using xgboost-gradient boosted decision tree. *Adv. Sci. Technol. Eng. Syst*, 5:536–549, 2020.

[44] AV-TEST Institute: Malware. https://www.av-test.org/en/statistics/malware/, 2022. [Online; accessed 15-December-2022].

[45] Microsoft Defender Security Research Team. https://www.microsoft.com/en-us/security/blog/2018/02/14/how-artificial-intelligence-stopped-an-emotet-outbreak/, 2018. [Online; accessed 17-May-2023].

[46] microsoft/LightGBM. https://github.com/Microsoft/LightGBM, 2023. [Online; accessed 17-May-2023].

[47] Dan Milmo. Nhs ransomware attack: what happened and how bad is it? https://www.theguardian.com/technology/2022/aug/11/nhs-ransomware-attack-what-happened-and-how-bad-is-it, 2022. [Online; accessed 15-December-2022].

[48] Jarernsri Mitrpanont, Wudhichart Sawangphol, Thanita Vithantirawat, Sinattaya Paengkaew, Prameyuda Suwannasing, Atthapan Daramas, and Yi-Cheng Chen. A study on using python vs weka on dialysis data analysis. In *2017 2nd International Conference on Information Technology (INCIT)*, pages 1–6, 2017.

[49] Sara Mohammadi, Hamid Mirvaziri, Mostafa Ghazizadeh-Ahsaee, and Hadis Karimipour. Cyber intrusion detection by combined feature selection algorithm. *Journal of information security and applications*, 44:80–88, 2019.

[50] Mehrnoosh Monshizadeh, Vikramajeet Khatri, Buse Atli, and Raimo Kantola. An intelligent defense and filtration platform for network traffic. In Kaushik Roy Chowdhury, Marco Di Felice, Ibrahim Matta, and Bo Sheng, editors, *Wired/Wireless Internet Communications*, pages 107–118, Cham, 2018. Springer International Publishing.

[51] Tyler Moore. The economics of cybersecurity: Principles and policy options. *International Journal of Critical Infrastructure Protection*, 3(3):103–117, 2010.

[52] Charlie Murphy, Patrick Gray, and Gordon Stewart. Verified perceptron convergence theorem. In *Proceedings of the 1st ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, pages 43–50, 2017.

[53] Ani Petrosyan. https://www.statista.com/statistics/680943/malware-os-distribution/, 2020. [Online; accessed 25-March-2023].

[54] Naveen Mohan Prajapati, Atish Mishra, and Praveen Kumar Bhanodia. Literature survey - ids for ddos attacks. *2014 Conference on IT in Business, Industry and Government (CSIBIG)*, pages 1–3, 2014.

[55] Ciberataque ransomware paraliza actividad del Hospital Clínic de Barcelona. https://www.ciberseguridad.eus/ultima-hora/ciberataque-ransomware-paraliza-actividad-del-hospital-clinic-de-barcelona, 2023. [Online; accessed 20-May-2023].

[56] IBM reports. Cost of a data breach 2022 A million-dollar race to detect and respond. https://www.ibm.com/reports/data-breach/, 2022. [Online; accessed 15-December-2022].

[57] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[58] Cuckoo Sandbox. https://cuckoosandbox.org/, 2019. [Online; accessed 21-May-2023].

[59] Kaspersky Sandbox. https://content.kaspersky-labs.com/se/media/es/business-security/enterprise/Kaspersky-Sandbox-product-brief-es.pdf, 2019. [Online; accessed 7-May-2023].

[60] Microsoft 365 Windows Sandbox. https://learn.microsoft.com/es-es/windows/security/threat-protection/windows-sandbox/windows-sandbox-overview, 2023. [Online; accessed 21-May-2023].

[61] Iqbal H. Sarker, A. S. M. Kayes, Shahriar Badsha, Hamed Alqahtani, Paul Watters, and Alex Ng. Cybersecurity data science: an overview from machine learning perspective. *Journal of Big Data*, 7(1):41, Jul 2020.

[62] J. Saxe and H. Sanders. *Malware Data Science: Attack Detection and Attribution*. No Starch Press, 2018.

[63] Alex Shenfield, David Day, and Aladdin Ayesh. Intelligent intrusion detection systems using artificial neural networks. *Ict Express*, 4(2):95–99, 2018.

[64] Christoffer Sjöblom. Artificial intelligence in cybersecurity and network security. 2021.

[65] I Kill You!"Skylight Cyber Çylance. https://skylightcyber.com/2019/07/18/cylance-i-kill-you/, 2019. [Online; accessed 7-May-2023].

[66] I. Stančin and A. Jović. An overview and comparison of free python libraries for data mining and big data analysis. In *2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 977–982, 2019.

[67] A Comprehensive Guide To PE Structure. https://tech-zealots.com/malware-analysis/pe-portable-executable-structure-malware-analysis-part-2/, 2022. [Online; accessed 22-May-2023].

[68] Jiawei Su, Danilo Vargas Vasconcellos, Sanjiva Prasad, Daniele Sgandurra, Yaokai Feng, and Kouichi Sakurai. Lightweight classification of iot malware based on image recognition. In *2018 IEEE 42Nd annual computer software and applications conference (COMPSAC)*, volume 2, pages 664–669. IEEE, 2018.

[69] TT Teoh, Yue Zhang, YY Nguwi, Yuval Elovici, and WL Ng. Analyst intuition inspired high velocity big data analysis using pca ranked fuzzy k-means clustering with multi-layer perceptron (mlp) to obviate cyber security risk. In *2017 13th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)*, pages 1790–1793. IEEE, 2017.

[70] Asuka Terai, Shingo Abe, Shoya Kojima, Yuta Takano, and Ichiro Koshijima. Cyber-attack detection for industrial control system monitoring with support vector machine based on communication profile. In *2017 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 132–138, 2017.

[71] Javier Martínez Torres, Carla Iglesias Comesaña, and Paulino J. García-Nieto. Review: machine learning techniques applied to cybersecurity. *International Journal of Machine Learning and Cybernetics*, 10(10):2823–2836, January 2019.

[72] FORTINET-CIA TRIAD. https://www.fortinet.com/resources/cyberglossary/cia-triad/, 2022. [Online; accessed 7-May-2023].

[73] E. Tsukerman. *Machine Learning for Cybersecurity Cookbook: Over 80 recipes on how to implement machine learning algorithms for building security systems using Python*. Packt Publishing, 2019.

[74] AM Turing. On computable numbers, with an application to the entscheidungsproblem. a correction. *Proceedings of the London*, 1938.

[75] Ransomware Attack Disrupts Antwerp City Services via a Digital Partner. https://www.cpomagazine.com/cyber-security/ransomware-attack-disrupts-antwerp-city-services-via-a-digital-partner/, 2022. [Online; accessed 20-May-2023].

[76] Joseph Weizenbaum. ELIZA—a computer program for the study of natural language communication between man and machine (1966). In *Ideas That Created the Future*, pages 271–278. The MIT Press, February 2021.

[77] Isaac Wiafe, Felix Nti Koranteng, Emmanuel Nyarko Obeng, Nana Assyne, Abigail Wiafe, and Stephen R. Gulliver. Artificial intelligence for cybersecurity: A systematic mapping of literature. *IEEE Access*, 8:146598–146612, 2020.

[78] Nadine Wirkuttis and Hadas Klein. Artificial intelligence in cybersecurity. *Cyber, Intelligence, and Security*, 1(1):103–119, 2017.

[79] Changsheng Wu, Wenbo Ding, Ruiyuan Liu, Jiyu Wang, Aurelia Wang, Jie Wang, Shengming Li, Yunlong Zi, and Zhong Wang. Keystroke dynamics enabled authentication and identification using triboelectric nanogenerator array. *Materials Today*, 02 2018.

[80] Zhao Yanling, Deng Bimin, and Wang Zhanrong. Analysis and study of perceptron to solve xor problem. In *The 2nd International Workshop on Autonomous Decentralized System, 2002.*, pages 168–173, 2002.

[81] Muhammad Irfan Yousuf, Izza Anwer, Tanzeela Shakir, Minahil Siddiqui, and Maysoon Shahid. Multi-feature dataset for windows pe malware classification. *arXiv preprint arXiv:2210.16285*, 2022.