Aprendizaje automático aplicado a la detección de malware y de ciberataques

Universitat Oberta de Catalunya

TRABAJO FINAL DE GRADO

Autor: **Jorge Pablo Trías Posa**

Área: Seguridad Informática

Grado de Ingeniería Informática

Nombre del director de TFG:
Gerard Farràs Ballabriga

Nombre del PRA:
Pau Perea Paños

11 de junio de 2024

Esta obra esta sujeta a una licencia de Reconocimiento-NoComercial-SinObraDerivada
https://creativecommons.org/licenses/by-nc-nd/4.0/deed.es

Con humildad y gratitud, me gustaría dedicar este trabajo final primeramente a Dios, por haberme acompañado a lo largo de mi carrera y por haberme ayudado a obtener uno de mis anhelos más deseados. A ti, Señor, dedico este logro con todo mi corazón.

A Stephanie, mi compañera de vida.
Gracias por tu paciencia y comprensión infinitas durante este viaje. Tu amor y aliento me han sostenido en los momentos más difíciles. Agradezco tu apoyo cada vez que querías ver un capítulo nuevo de una serie juntos y yo, en muchas ocasiones, tenía que decirte que no porque debía estudiar. Gracias por entenderlo siempre y por nunca soltar mi mano.
Este logro es tan tuyo como mío.

A mi madre, cuya infinita paciencia, amor y apoyo incondicional han sido la base de todo lo que he logrado. Gracias por enseñarme a soñar en grande, a no rendirme nunca y a creer en mí mismo.

A mi hermano, por su constante aliento y apoyo que han sido fundamentales para mí en cada paso de este camino académico. Desde los desafíos más difíciles hasta las pequeñas victorias, siempre has estado a mi lado. Gracias.

Ficha del Trabajo Final de Grado

Título del trabajo:	Aprendizaje automático aplicado a la detección de malware y de ciberataques
Nombre del autor:	Jorge Pablo Trías Posa
Nombre del director de TFG:	Gerard Farràs Ballabriga
Nombre del PRA:	Pau Perea Paños
Fecha de entrega:	11 de junio de 2024
Titulación o programa:	Grado de Ingeniería Informática
Área del trabajo final:	Área: Seguridad Informática
Idioma del trabajo:	Castellano
Palabras clave:	Aprendizaje Automático, Inteligencia Artificial, Ciberseguridad

Resumen del trabajo

Desde el 21 de noviembre de 1969, fecha en la que se conectó el ordenador SDS Sigma 7 al primer nodo de la primera red de computadoras llamada ARPANET, correspondiente a la Universidad de California en Los Ángeles (UCLA, EE. UU.), se han observado los riesgos que representan los ataques informáticos y la propagación de malware a través de la red.

Esta interconexión computacional ha avanzado a pasos agigantados desde sus inicios, y con ello, la cantidad de vulnerabilidades de seguridad que ha llevado asociadas. Por esta razón, a partir de aquel año 1969, se ha producido una batalla interminable entre actores atacantes y defensores que se remonta hasta el presente.

Al mismo tiempo que se producían estos avances tecnológicos, se iban generando en esta red enormes volúmenes de datos, lo que hoy en día se conoce como *"Big Data"*. Con el paso de los años, se ha descubierto el verdadero potencial que se esconde tras estos datos, y cómo a través de diferentes algoritmos, se puede dotar a una máquina computacional de la capacidad de recibir estas enormes cantidades de datos y aprender de ellos (denominado *"aprendizaje automático"* o *"machine learning"*).

Por tanto, si se fusionan todos estos avances, se obtiene la combinación perfecta para explotar todo el potencial existente en las técnicas de aprendizaje automático y poder aplicarlo al campo de la Seguridad Informática con el fin de contribuir en la mejora de los sistemas de defensa existentes.

En este Trabajo Final de Grado se abordará la creación de dos modelos de aprendizaje automático, a través del entrenamiento de diferentes algoritmos con un conjunto de datos seleccionado para cada modelo, con el objetivo de que estos modelos entrenados sean capaces de predecir malware y ataques en la red.

Abstract

Since November 21, 1969, the date the SDS Sigma 7 computer was connected to the first node of the first computer network called ARPANET, corresponding to the University of California in Los Angeles (UCLA, USA), the risks posed by computer attacks and the spread of malware through the network have been observed.

This computational interconnection has advanced by leaps and bounds since its inception and with it, the number of security vulnerabilities that have been associated with it. For this reason, since that year 1969, there has been an endless battle between attacking and defending actors that dates back to the present.

At the same time that these technological advances were taking place, enormous volumes of data were being generated on this network, what today is known as *"Big Data"*. Over the years, the true potential hidden behind this data has been discovered, and how, through different algorithms, a computing machine could be given the ability to receive these enormous amounts of data and learn from them (called *"machine learning"*).

Therefore, if all these advances are merged, the perfect combination is obtained to exploit all the potential that exists in machine learning techniques and to be able to apply it to the field of Computer Security in order to contribute to the improvement of existing defense systems.

This Final Degree Project will cover the creation of two machine learning models, through the training of different algorithms with a dataset selected for each model, with the aim that these trained models are capable of predicting network attacks and malware.

Índice general

1. Introducción **1**
1.1. Contexto y justificación del trabajo 2
1.2. Objetivos del trabajo 4
1.3. Impacto en sostenibilidad, ético-social y de diversidad 4
1.4. Enfoque y método seguido 6
1.5. Planificación del trabajo 8
1.6. Breve sumario de productos obtenidos 12
1.7. Breve descripción de los otros capítulos de la memoria 12

2. Estado del arte **13**
2.1. Avances en el análisis de malware 13
2.2. Avances en el análisis de anomalías de red 14

3. Aprendizaje automático **16**
3.1. Tipos de aprendizaje 16
3.1.1. Aprendizaje supervisado 17
3.1.2. Aprendizaje no supervisado 17
3.1.3. Aprendizaje por refuerzo 17
3.1.4. Aprendizaje profundo 18
3.2. Algoritmos de aprendizaje automático 18
3.3. Algoritmos empleados en el desarrollo de este TFG 19
3.3.1. Algoritmos basados en modelos lineales 19
3.3.2. Algoritmos de agrupamiento (Clustering) 20
3.3.3. Algoritmos basados en árboles de decisión 22
3.3.4. Algoritmos probabilísticos 25
3.3.5. Algoritmo SVM (Máquina de vectores de soporte) 28
3.4. ¿Qué es un modelo de aprendizaje automático? 30
3.5. Ruta seguida para el desarrollo de este TFG 30

4. Malware **32**
4.1. Tipos de malware 32
4.2. ¿Qué se esconde detrás de un malware? 34

4.3. Malware para la plataforma Microsoft Windows 35
4.3.1. Identificación de un archivo PE 35
4.3.2. El formato de archivo PE (Portable Executable) 36
4.4. Métodos de detección de malware 40
4.4.1. Métodos basados en firmas 40
4.4.2. Métodos basados en el comportamiento 41
4.4.3. Métodos heurísticos: Aprendizaje automático 41
4.5. ¿Como puede mejorar el aprendizaje automático las técnicas actuales de detección de malware? 42

5. Ataques informáticos en la red **43**
5.1. Clasificación de los ataques en la red 43
5.2. Fases de un ataque informático 46
5.3. Métodos de detección de intrusiones en la red 47

6. Caso de uso 1 **49**
6.1. Justificación de la elección del formato PE 50
6.2. Creación del dataset 50
6.2.1. Creación del subconjunto de datos de archivos legítimos 51
6.2.2. Creación del subconjunto de datos de malware 51
6.3. Creación de diferentes modelos de aprendizaje 52
6.3.1. Comprensión de los datos 52
6.3.2. Preprocesado de los datos 52
6.3.3. División del dataset en un conjunto de entrenamiento y en un conjunto de test 53
6.3.4. Balanceo del conjunto de datos 53
6.3.5. Validación cruzada 54
6.3.6. Selección de características 55
6.3.7. Desarrollo de diferentes modelos de aprendizaje automático 57

7. Caso de uso 2 **59**
7.1. Búsqueda de un conjunto de datos adecuado 59
7.1.1. Comparativa de diferentes conjuntos de datos existentes 60
7.1.2. Justificación de la elección del conjunto de datos NSL-KDD 61
7.2. Descripción del conjunto de datos NSL-KDD 62
7.3. Creación de diferentes modelos de aprendizaje 64
7.3.1. Comprensión de los datos 64
7.3.2. Exploración de los diferentes tipos de ataques existentes en ambos datasets 66
7.3.3. Planteamiento de la estrategia a seguir 67
7.3.4. Preprocesamiento de los datos 68
7.3.5. Selección de características 70
7.3.6. Desarrollo de diferentes modelos de aprendizaje automático 72

8. Evaluación de los modelos **74**
8.1. Evaluación de los modelos creados para el caso de uso 1 75
8.2. Evaluación de los modelos creados para el caso de uso 2 75

9. Conclusiones **77**

10. Trabajo futuro **78**

A. CU1. Detección de malware **81**
A.1. Código en Python para crear el conjunto de datos a partir de ficheros PE 81
A.2. Bibliotecas necesarias para la creación de los modelos 83
A.3. Exploración del conjunto de datos generado 84
A.4. Creación de funciones que se utilizaran para la creacion de los modelos de aprendizaje 84
A.5. División del dataset en un conjunto de entrenamiento y en un conjunto de test . 86
A.6. Balanceo del conjunto de datos 86
A.7. Análisis de las características del conjunto de datos 87
A.8. Construcción del dataset final 87
A.9. Desarrollo de diferentes modelos 87

B. CU2. Detección de ciberataques **89**
B.1. Etiquetas de grupo de ataque de los diferentes tipos de ataques existentes 89
B.2. Exploración del conjunto de datos generado 90
B.3. Adición del nombre de las características al dataset 90
B.4. Clasificación del tráfico de red en el dataset de entrenamiento 91
B.5. Exploración de los diferentes tipos de ataques existentes 91
B.6. Etiquetas de grupo de ataque de los diferentes tipos de ataques existentes 92
B.7. División entre las características y el objetivo de ambos datasets 92
B.8. Tratamiento de las variables simbólicas y continuas 93
B.9. Estandarización de los datos 93
B.10. Análisis de la relación entre las características del conjunto de datos 94
B.11. Construcción de diferentes versiones del dataset 94
B.12. Creación de un modelo de aprendizaje automático con el algoritmo Árboles de decisión 94
B.13. Automatización en la creación de varios modelos de aprendizaje aplicando diferentes algoritmos 95

Índice de figuras

1.1. Diagrama de *Gantt* del TFG 9
1.2. Tablero de *Kanban* - Fase I 10
1.3. Tablero de *Kanban* - Fase II 10
1.4. Tablero de *Kanban* - Fase III 11
1.5. Tablero de *Kanban* - Fase IV, V y VI 11

3.1. Algoritmos de aprendizaje automático. Fuente: [12] 18
3.2. Ejemplo de una regresión lineal. Fuente: Wikipedia 19
3.3. Función sigmoide. Fuente: Wikipedia 20
3.4. Ejemplo de algoritmo K-Means. Fuente: [16] 21
3.5. Funcionamiento del algoritmo K-nn. Fuente: [18] 22
3.6. Ejemplo de árbol de decisión. Fuente: [19] 23
3.7. Ilustración del funcionamiento de los bosques aleatorios. Fuente: Wikipedia [21] 25
3.8. Algoritmo Naïve Bayes. Fuente: [22] 26
3.9. Separación de clases en algoritmo SVM. 28
3.10. Existencia de diferentes hiperplanos en algoritmo SVM. 29
3.11. Truco del núcleo (Kernel Trick). 30
3.12. Proceso de creación de modelos seguido en este TFG. 31

4.1. Versión hexadecimal del código ejecutable (calc.exe) 36

5.1. Fases de un ciberataque 46

6.1. Ejemplo de división de un dataset. Fuente: [24] 53
6.2. Distribución de archivos PE 54
6.3. Esquema de validación cruzada de 4 iteraciones. Fuente: [25] 55
6.4. Relación entre la variable 'MajorSubsystemVersion' del dataset de malware vs archivos legítimos 56
6.5. Mapa de calor de correlaciones entre características 57

7.1. Lista de características del dataset NSL-KDD. 62
7.2. Tipos de ciberataques existentes en el conjunto de datos NSL-KDD. 63
7.3. Clasificación del tráfico de red en el dataset de entrenamiento. 65

7.4. Clasificación del tráfico de red en el dataset de test. 65
7.5. Tipos de ataque por cantidad que contiene el dataset de entrenamiento. 66
7.6. Tipos de ataque por cantidad que contiene el dataset de test. 66
7.7. Grupos de ataque por cantidad que contiene el dataset de entrenamiento. 67
7.8. Grupos de ataque por cantidad que contiene el dataset de test. 68
7.9. One Hot Encoding. 70
7.10. Mapa de calor de correlaciones entre características 71
7.11. Árbol de decisión generado. 72

8.1. Resultados del entreamiento de diferentes algoritmos de aprendizaje CU1 75
8.2. Resultados del entreamiento de diferentes algoritmos de aprendizaje CU2 75

Índice de cuadros

3.1. Ejemplo de funcionamiento algoritmo Naïve Bayes 26
3.2. Dataset de ejemplo algoritmo Naïve Bayes 27
3.3. Ejemplo de tabla de frecuencias algoritmo Naïve Bayes 27
3.4. Ejemplo de tabla de probabilidad algoritmo Naïve Bayes 27

4.1. Ejemplos de números mágicos de diferentes tipos de archivos (magic bytes) . . . 36

Capítulo 1

Introducción

Con el nacimiento de la mayor red de comunicaciones en torno a aquellos años sesenta, comúnmente llamada Internet, muchas personas se dieron cuenta del gran potencial que este desarrollo tecnológico representaba para la humanidad. Sin embargo, mientras parte de esas personas creían que esta red proporcionaría un gran valor para la sociedad por la posibilidad de intercambiar información y conocimientos, la otra parte descubrió un enorme medio de transmisión donde poder perpetuar diferentes ataques informáticos con el objetivo de obtener un beneficio.

El campo de la seguridad informática se encuentra íntimamente relacionado con estas redes de comunicaciones, ya que, a través de ellas, se propaga una cantidad ingente de paquetes de datos cada minuto en el mundo y, averiguar que se esconde detrás de estos paquetes, puede marcar un antes y un después en las organizaciones. Pensando en los comienzos de esta computación interconectada, si alguien tenía los conocimientos técnicos necesarios podía crear malware, si alguien disponía de conexión a esta red de comunicaciones podía obtener paquetes de ella o difundirlos, pero, si alguien contaba con estos dos factores y con la intención de infectar a uno o varios equipos, ahí es cuando se veía comprometida cualquier entidad que fuera objetivo de dicho ataque.

Para un atacante, crear un determinado malware sin poder distribuirlo no significa absolutamente nada. Este es el motivo por el que la seguridad informática, el malware y las redes de comunicaciones van de la mano. Esta afirmación la podemos verificar si nos remontamos a 1971, donde nació *Creeper*, el primer virus informático creado por Bob Thomas con el único objetivo de demostrar la capacidad que tiene un software de autoreplicarse de forma autónoma a través de esta red de comunicaciones.

En las últimas décadas, la seguridad informática ha cobrado especial relevancia debido a la aparición de una enorme gama de amenazas, dentro de las cuales se encuentran: *malware, phishing, spam, ataques de intrusión, ataques de disponibilidad, ataques de aplicaciones web*, entre otras.

Al mismo tiempo y siguiendo con los avances tecnológicos, se produce un gran impulso del desarrollo de la Inteligencia Artificial, nacida en aquel 1943 con la publicación del artículo *"A Logical Calculus of Ideas Immanent in Nervous Activity"* de Warren McCullough y Walter Pitts. En los últimos años se han obtenido unos avances sin precedentes en esta disciplina que busca imitar las habilidades cognitivas propias del ser humano, empleando para ello las capacidades de la computación.

La Inteligencia Artificial (IA) persigue imitar aquellos comportamientos inteligentes propios del ser humano, como por ejemplo: reconocer voces, conducir, analizar patrones en una radiografía, reconocer el correo no deseado, entre otros. Sin embargo, la capacidad que nos define a los humanos como seres inteligentes es la capacidad de aprender, y aquí es cuando entra en juego el Aprendizaje Automático (AA) o Machine Learning (ML), una rama de la IA que estudia cómo enseñar a una máquina a razonar y tomar decisiones. En otras palabras, pasamos de que una máquina imite un determinado comportamiento inteligente porque alguien le codificó cómo hacerlo (IA) a que imite este mismo comportamiento pero porque el propio sistema ha aprendido a realizarlo (AA).

El aprendizaje automático es un conjunto de técnicas (algoritmos y procesos que ayudan a una máquina a aprender) implementadas en un sistema computacional, que permiten a una máquina extraer información de un conjunto de datos (*dataset*) para posteriormente descubrir patrones en estos (aprender de ellos) y poder predecir resultados futuros con la entrada de nuevos datos. Esto se realiza a través del entrenamiento de modelos pero eso se verá en capítulos posteriores.

Dentro de las disciplinas en las que el aprendizaje automático puede ser aplicado con el objetivo de aportar mejoras, se encuentra la seguridad informática. Si juntamos todo lo mencionado anteriormente: redes a través de las que pasan tráficos de millones de paquetes cada segundo, conocimientos en seguridad informática y técnicas de aprendizaje automático, tenemos una excelente combinación de variables en las que trabajar. En este TFG nos sumergiremos en el mundo de la detección automática de intrusiones y de malware, exponiendo dos aplicaciones o casos de uso para los cuales el aprendizaje automático ha mostrado grandes resultados: la detección de ciberataques y el análisis de malware.

1.1. Contexto y justificación del trabajo

En la sección anterior se ha introducido una breve historia de los comienzos de las redes de comunicaciones, del software malicioso y de la Inteligencia Artificial. Sin embargo, ese panorama inicial de los años sesenta ha cambiado drásticamente. Hoy en día, los avances agigantados en el desarrollo y construcción de *hardware*, van de la mano con los avances de desarrollo de *software*. En este momento, es cuando, recordando aquella *ley de Moore* que expresa que la

cantidad de transistores en un microprocesador se vería duplicada cada dos años (hasta llegar a un límite donde esta cantidad sea menor), nos damos cuenta que la computación actual avanza a un ritmo vertiginoso.

Los pasos en el campo de la seguridad informática también son agigantados, de tal manera que mientras se escriben estas líneas, están surgiendo miles de vulnerabilidades nuevas alrededor del mundo. Llegados a este punto, debemos percatarnos de que este avance tecnológico debe estar acompañado por aquellas técnicas y mecanismos de protección, propios de la seguridad informática, quienes nos ayudarán a mantener la confidencialidad, la integridad y la disponibilidad de estos nuevos sistemas desarrollados. Sin embargo, es importante destacar que todo debe de ir acorde; no podemos proteger estos sistemas con las mismas técnicas que se usaban en tiempos anteriores. Estas técnicas empleadas también deben evolucionar y adaptarse a los nuevos entornos, beneficiándose de los avances descubiertos en materia de seguridad.

Actualmente, gran parte de las organizaciones trabajan en sus Centros de Operaciones de Seguridad *(SOC - Security Operations Center)* en desventaja respecto a sus atacantes, debido a que no juegan con las mismas cartas. Mientras unos siguen con un modelo de defensa tradicional, otros se preparan y se forman con las técnicas más avanzadas de ataques y de creación de malware. La motivación de desarrollar este TFG viene debido a que las amenazas de seguridad aumentan cada día en complejidad y número; gran parte debido a que los actores atacantes, también están empleando diversas técnicas de aprendizaje automático pero en este caso para evadir los sistemas de prevención y detección de intrusiones o para la generación automatizada de malware.

En este punto es cuando aparece la necesidad de investigar sobre las nuevas técnicas empleadas en el campo de la seguridad informática para mantener a los sistemas computacionales protegidos. En una investigación previa a la redacción de este TFG, se ha comprobado como los principales avances en materia de seguridad, pasan por el campo de la Inteligencia Artificial y más concretamente, por su rama de Aprendizaje Automático, escondiéndose dentro de ella las técnicas más sofisticadas. Esto se debe a raíz de que los investigadores y profesionales del sector, se han dado cuenta del enorme potencial que tienen los datos, de todo el conocimiento que se puede extraer de ellos y de la posibilidad de dotar a una máquina de la capacidad de aprendizaje a partir de modelos entrenados con los datos anteriormente mencionados.

Esta necesidad de investigación en profundidad será en parte cubierta en los contenidos descritos en este Trabajo Final de Grado, donde se recorrerán diferentes algoritmos de aprendizaje automático a través de dos casos de uso, con el objetivo de detectar ciberataques y malware en tiempo real, y con ello, poder tomar medidas proactivas para proteger a estos sistemas y minimizar los riesgos. Con la redacción de este TFG se pretende investigar las técnicas pioneras en este campo y llevarlas a la práctica a través de en dos casos de uso donde se pueda demostrar realmente cómo el aprendizaje automático puede ayudar a mejorar los sistemas de defensa existentes en las organizaciones.

1.2. Objetivos del trabajo

Los objetivos que se pretenden alcanzar tras la finalización de este Trabajo Final de Grado son los siguientes:

1. Estudiar de qué forma se puede mejorar la seguridad de los sistemas de computación actuales, haciendo uso para ello, de los avances que se han producido en el campo de la Inteligencia Artificial en los últimos años.

2. Investigar sobre la rama de la IA que es capaz de dotar a los ordenadores de la capacidad de que aprendan por sí mismos, llamada Aprendizaje Automático.

3. Estudiar las características propias de los ataques informáticos en una red de comunicaciones y las características propias del malware.

4. Analizar los diferentes algoritmos y técnicas de aprendizaje automático existentes, que puedan aplicarse al campo de la seguridad informática con el objetivo de fortificar los sistemas actuales.

5. Conocer los diferentes métodos de preparación de los datos, selección de características y extracción de conocimiento.

6. Aprender las técnicas de diseño y construcción de modelos predictivos basados en aprendizaje automático, así como el entrenamiento posterior de estos modelos con diferentes conjuntos de datos.

7. Desarrollar un modelo predictivo para la detección de anomalías en la red y otro para la detección de malware, con el objetivo de que estos sean capaces de analizar los datos futuros sin supervisión humana.

8. Identificar los fortalezas y las debilidades que puede suponer implementar en una organización un sistema basado en técnicas de aprendizaje automático.

1.3. Impacto en sostenibilidad, ético-social y de diversidad

El desarrollo de la Inteligencia Artificial no sólo ha transformado muchos aspectos de nuestra vida, generando nuevas facilidades en tan diversos campos como la medicina, la robótica o la seguridad informática, sino que también ha traído consigo profundos dilemas éticos. Entre estos dilemas se encuentran, por ejemplo, la incertidumbre de si algún día, estas máquinas pensantes creadas por el hombre podrían volverse en su contra, si los avances de la inteligencia artificial terminarían por disminuir al mínimo la plantilla de personal de una empresa, o, si los algoritmos de aprendizaje profundo (*deep learning*) tomaran decisiones incorrectas quién sería

la persona responsable.

Respecto a estos dilemas ético-sociales existe en el mundo una gran preocupación, y por ende, una gran necesidad de regular el comportamiento moral de las máquinas. Es por ello que la Unión Europea llevaba un tiempo redactando, junto con expertos en la materia, una Ley cuyo principal objetivo era garantizar que exista un control profesional y ético en el desarrollo y uso de sistemas de IA, que ofrezca garantías de seguridad, de privacidad y de respeto a los derechos de los ciudadanos. La Ley de IA finalmente ha sido aprobada por el el Parlamento Europeo este pasado 13 de marzo de 2024.

En cuanto a esta Ley, el Parlamento Europeo ha asegurado lo siguiente: *"Tenemos la primera regulación del mundo que marca un camino claro para un desarrollo seguro y centrado en el ser humano de la IA. Ahora tenemos un texto que refleja las prioridades del Parlamento. El punto principal en este momento será la aplicación y el cumplimiento por parte de empresas e instituciones".*

Por otro lado, tal y como se menciona en la web oficial de la Organización de las Naciones Unidas, el 25 de septiembre de 2015, los líderes mundiales adoptaron un conjunto de objetivos globales para erradicar la pobreza, proteger el planeta y asegurar la prosperidad para todos como parte de una nueva agenda de desarrollo sostenible. Cada objetivo tiene metas específicas que deben alcanzarse en los próximos 15 años.

Llegados a este punto, únicamente nos queda por tratar el apecto de diversidad, del cual podemos asegurar que trabajar en materia de equidad, inclusión social y diversidad, es un tema de vital importancia para el contribuir a mejorar el rendimiento de las organizaciones. En la actualidad, existen informes, como el elaborado por Aspen Institute, donde se propone una lista de prioridades que deberían seguir aquellas organizaciones de la industria de la seguridad informática que deseen mejorar en este aspecto. Asimismo, organizaciones como ISC2, ofrecen a través de sus páginas web, diferentes repositorios con gran cantidad de recursos para que las entidades puedan incorporar estos aspectos a su modelo de negocio.

Una vez obtenido el marco actual en cuanto a políticas ético-sociales, de sostenibilidad y de diversidad, únicamente nos queda fusionar todos estos conceptos para poder evaluar el impacto que este Trabajo Final de Grado puede generar sobre ellos.

En cuanto al posible impacto en sostenibilidad, este TFG se encuentra íntimamente relacionado con el objetivo de desarrollo sostenible número 9: *"Industria, Innovación e Infraestructuras"*, que tal como se menciona en la web oficial de la ONU: *"El Objetivo 9 pretende construir infraestructuras resilientes, promover la industrialización sostenible y fomentar la innovación"*. Al mismo tiempo, sabemos que el número y el impacto de los ataques informáticos en las organizaciones continuará aumentando en los próximos años, según el último informe del Centro Criptológico Nacional (CCN-CERT).

Por tanto, este TFG se plantea como una solución para contribuir a la fortificación de los sistemas computacionales existentes, empleando para ello técnicas modernas de aprendizaje automático. Con esto se pretende conseguir infraestructuras más seguras, evitando posibles ataques contra las organizaciones, ya que, el objetivo es que los modelos de aprendizaje automático generados en este TFG sean capaces de reconocer ataques y malware en la red. Todo ello tendrá igualmente repercusiones para la sociedad, debido a que, con sistemas más seguros se podrían evitar, por ejemplo, posibles fugas de información, caídas de servicios o ataques a infraestructuras críticas.

Protegiendo de ataques informáticos a las infraestructuras críticas, como la energía o el agua, también estamos promoviendo la sostenibilidad ambsiental, ya que, estos podrían causar daños ambientales si llegasen a perpetuarse.

Asimismo, protegiendo la privacidad y la seguridad de la información de las personas, también estamos colaborando con muchos ODS como puede ser el número 3: *"Salud y bienestar"* que versa sobre *"Garantizar una vida sana y promover el bienestar para todos en todas las edades"*, y que lleva asociado tanto el almacenamiento de datos médicos de carácter personal como la existencia de sistemas informáticos en los hospitales. Protegiendo estos sistemas informáticos críticos y estos datos del acceso no autorizado, es otra forma de colaborar con este ODS.

1.4. Enfoque y método seguido

La estrategia elegida para afrontar este proyecto se sustenta en las siguientes fases:

a. Fase de investigación sobre los diferentes algoritmos y técnicas de aprendizaje automático que pueden ser aplicados con garantías de éxito al campo de la ciberseguridad, concretamente a la detección de malware y de ataques informáticos a través de la red.

b. Fase de análisis de la información recopilada, a través de numerosas fuentes, con el objetivo de seleccionar las técnicas y los algoritmos de aprendizaje automático que se consideren mas adecuados para los dos casos de uso propuestos.

c. Fase de desarrollo de dos modelos de aprendizaje automático; uno para cada caso de uso. Estos modelos se crearán a través del entrenamiento de los algoritmos elegidos con el conjunto de datos etiquetado de cada caso. Aquí es donde se llevará a la práctica toda la teoría investigada y detallada en los apartados teóricos de este TFG.

   Los dos casos de uso escogidos son los siguientes:

   a) **Caso de uso 1**: Detección de malware empleando técnicas de Aprendizaje Automático.
   En este caso de uso se presentará el análisis de diferentes archivos binarios con diferentes formatos, en base a la información extraída de su interior. A continuación, se

entrenará a un modelo de aprendizaje automático con muestras de ejecutables benignos y de ejecutables maliciosos, con el objetivo de que el modelo final entrenado pueda predecir la tipología de un binario en base a sus características.

Es muy importante que un analista de malware conozca la estructura de los ficheros donde se encuentra el código malicioso, ya que, dependiendo de la plataforma o del tipo de archivo, esta estructura variará. Para desarrollar este caso de uso, se empleará el formato de archivos PE (Portable Executable); un formato de archivo para ficheros ejecutables usados en versiones de 32 y 64 bits del sistema operativo Microsoft Windows. Dentro de este formato de archivo se encuentran ficheros con extensión *.exe*, *.dll*, *.sys* o *.efi*.

El motivo de esta elección es debido a que según un informe de AV-Test para el primer trimestre de 2023, se descubrió que la industria del malware continúa apuntando principalmente a los sistemas Windows, ya que, el informe refleja que el 83,45 de todo el malware desarrollado recientemente se concentra en el sistema operativo Windows. Últimamente, también se encuentra en aumento el porcentaje de malware en los sistemas operativos móviles como andoid o iOS, debido a que este porcentaje se encuentra estrechamente relacionado con la participación de mercado que va alcanzando cada uno de estos.

Para crear y entrenar a nuestro modelo de aprendizaje automático, se utilizará un *dataset* generado a partir de muestras de malware actuales, proporcionadas por las plataformas VirusTotal [1] y VirusShare [2]. Para ello, se ha solicitado previamente el acceso académico, justificado por este TFG, a los repositorios de malware de estas plataformas de análisis de archivos online, donde se encuentran los últimos hallazgos encontrados. En base a estos directorios de malware descargados, se extraerán los valores del encabezado PE de cada código malicioso, empleando para ello, el módulo *pefile* [3] de *Python*. Nos centraremos en los campos del encabezado PE de los ficheros ejecutables, ya que, los analistas de malware los han considerado desde hace años como características muy eficaces en la detección de código malicioso.

Finalmente, se diseñará un script en *Python* donde se aprovechará la extracción de multitud de valores del encabezado PE con la librería *pefile* [3], para posteriormente automatizar el almacenamiento de ellas en un fichero con extensión *.csv* que será finalmente nuestro *dataset*.

b) **Caso de uso 2**: Detección de ciberataques empleando técnicas de Aprendizaje Automático.

Para llevar a cabo este caso, se partirá de la captura de un determinado tráfico de red para posteriormente analizarlo, con la intención de encontrar patrones en estos paquetes que nos ayuden a detectar posibles anomalías. El objetivo de esta fase es crear un modelo de aprendizaje automático, a través del entrenamiento de diferentes algoritmos con este tráfico de red capturado, para encontrar cuál de ellos realiza una mejor clasificación de los ataques existentes en él (DDos, fuerza bruta, escalada de privilegios, etc). Esto nos ayudará a obtener el modelo más adecuado, que luego

podrá ser capaz de recibir otro tráfico cualquiera y en base a su entrenamiento anterior, poder clasificarlo sin supervisión.

La captura de este tráfico de red se podría realizar a través de programas como *Wireshark* o *Tshark* y posteriormente exportarlo en un fichero con extensión *.csv* para poder analizarlo. Sin embargo, para focalizar este TFG en la explotación de las técnicas de aprendizaje automático, y no tanto en la preparación de un conjunto de datos donde existan diferentes tipos de ataques, se utilizará el conjunto de datos (*dataset*) NSL-KDD [4]. Este conjunto de datos, empleado por miles de profesionales dedicados a la detección de intrusiones dentro de la ciencia de datos, se detallará en un capítulo posterior.

Todo el código desarrollado y la documentación escrita para este TFG se encuentra alojado en el siguiente repositorio de GitHub:

`https://github.com/ptposa/TFG`

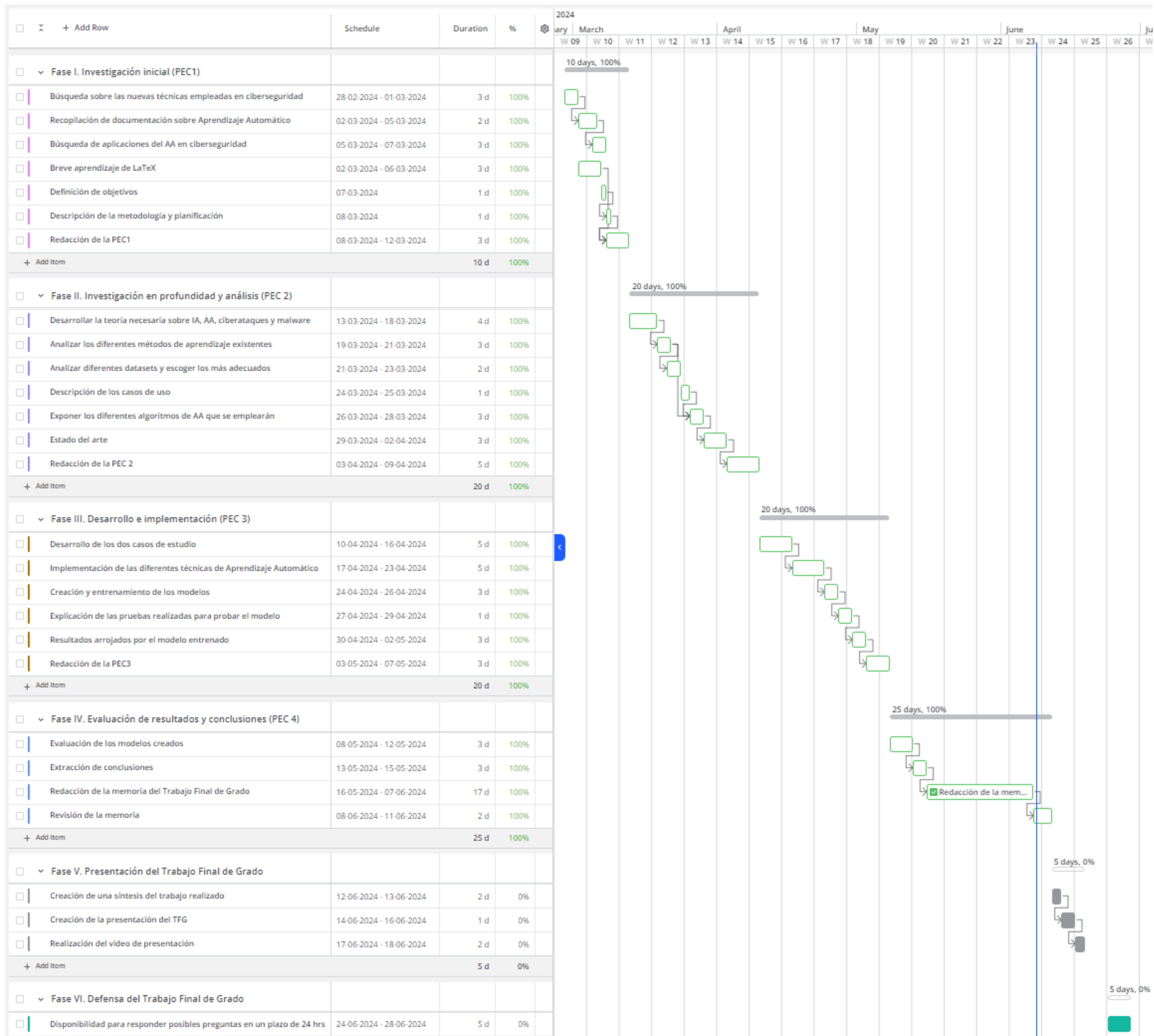
d. Fase de evaluación de los resultados obtenidos. Se evaluarán los modelos creados y se extraerán conclusiones. Asimismo, se valorará si la aplicación de técnicas de aprendizaje automático a los casos de uso ha conseguido mejorar los sistemas existentes.

1.5. Planificación del trabajo

Este Trabajo Final de Grado seguirá la metodología de desarrollo de proyectos *Kanban*. El término *kanban* proviene de la cultura japonesa y es una palabra formada por dos términos: *kan* que significa visual y *ban*, tarjeta. Se empleará esta metodología porque a través de estas tarjetas visuales añadidas a un tablero, se visualizarán todos los procesos con sus diferentes flujos de trabajo. Esta metodología de proceso continuo, ayudará a conocer lo que está ocurriendo de un solo vistazo, ya que, estas tarjetas permitirán una lectura fácil y rápida de las tareas a realizar, de las prioridades que tienen asignadas, de sus fechas límite, etc.

Sin embargo, también aprovecharemos las ventajas del diagrama de *Gantt*, propio de la metodología en cascada (*waterfall*), de forma que podamos contar con una metodología híbrida, aprovechando lo mejor de ambos mundos. Cuando hablamos de *Kanban* y de diagramas de *Gantt*, muchas personas suelen pensar que se debe escoger entre una representación u otra. Si bien es cierto que aquellos equipos que emplean a diario los diagramas de *Gantt*, practican enfoques de gestión de proyectos tradicionales, y, aquellos que emplean *Kanban* suelen practicar enfoques más ágiles, existen proyectos donde se han unido las fortalezas de ambas herramientas y el resultado ha sido muy satisfactorio.

Por tanto, emplearemos *Kanban* para monitorear los procesos diarios y, los diagramas de *Gantt* para planificar el proyecto y crear objetivos a largo plazo. Los diagramas de *Gantt* nos permitirán ver las dependencias entre tareas de una forma más gráfica (a través de flechas).



Add Row	Schedule	Duration	%
Fase I. Investigación inicial (PEC1)			
Búsqueda sobre las nuevas técnicas empleadas en ciberseguridad	28-02-2024 - 01-03-2024	3 d	100%
Recopilación de documentación sobre Aprendizaje Automático	02-03-2024 - 05-03-2024	2 d	100%
Búsqueda de aplicaciones del AA en ciberseguridad	05-03-2024 - 07-03-2024	3 d	100%
Breve aprendizaje de LaTeX	02-03-2024 - 06-03-2024	3 d	100%
Definición de objetivos	07-03-2024	1 d	100%
Descripción de la metodología y planificación	08-03-2024	1 d	100%
Redacción de la PEC1	08-03-2024 - 12-03-2024	3 d	100%
Add Item		10 d	100%
Fase II. Investigación en profundidad y análisis (PEC 2)			
Desarrollar la teoría necesaria sobre IA, AA, ciberataques y malware	13-03-2024 - 18-03-2024	4 d	100%
Analizar los diferentes métodos de aprendizaje existentes	19-03-2024 - 21-03-2024	3 d	100%
Analizar diferentes datasets y escoger los más adecuados	21-03-2024 - 23-03-2024	2 d	100%
Descripción de los casos de uso	24-03-2024 - 25-03-2024	1 d	100%
Exponer los diferentes algoritmos de AA que se emplearán	26-03-2024 - 28-03-2024	3 d	100%
Estado del arte	29-03-2024 - 02-04-2024	3 d	100%
Redacción de la PEC 2	03-04-2024 - 09-04-2024	5 d	100%
Add Item		20 d	100%
Fase III. Desarrollo e implementación (PEC 3)			
Desarrollo de los dos casos de estudio	10-04-2024 - 16-04-2024	5 d	100%
Implementación de las diferentes técnicas de Aprendizaje Automático	17-04-2024 - 23-04-2024	5 d	100%
Creación y entrenamiento de los modelos	24-04-2024 - 26-04-2024	3 d	100%
Explicación de las pruebas realizadas para probar el modelo	27-04-2024 - 29-04-2024	1 d	100%
Resultados arrojados por el modelo entrenado	30-04-2024 - 02-05-2024	3 d	100%
Redacción de la PEC3	03-05-2024 - 07-05-2024	3 d	100%
Add Item		20 d	100%
Fase IV. Evaluación de resultados y conclusiones (PEC 4)			
Evaluación de los modelos creados	08-05-2024 - 12-05-2024	3 d	100%
Extracción de conclusiones	13-05-2024 - 15-05-2024	3 d	100%
Redacción de la memoria del Trabajo Final de Grado	16-05-2024 - 07-06-2024	17 d	100%
Revisión de la memoria	08-06-2024 - 11-06-2024	2 d	100%
Add Item		25 d	100%
Fase V. Presentación del Trabajo Final de Grado			
Creación de una síntesis del trabajo realizado	12-06-2024 - 13-06-2024	2 d	0%
Creación de la presentación del TFG	14-06-2024 - 16-06-2024	1 d	0%
Realización del video de presentación	17-06-2024 - 18-06-2024	2 d	0%
Add Item		5 d	0%
Fase VI. Defensa del Trabajo Final de Grado			
Disponibilidad para responder posibles preguntas en un plazo de 24 hrs	24-06-2024 - 28-06-2024	5 d	0%

Figura 1.1: Diagrama de *Gantt* del TFG

En la anterior ilustración, se puede observar el diagrama de *Gantt* y, en las siguientes, se observa el tablero de *Kanban* para cada una de las Fases de este TFG, pudiéndose apreciar como se encuentran relacionados entre ellos.

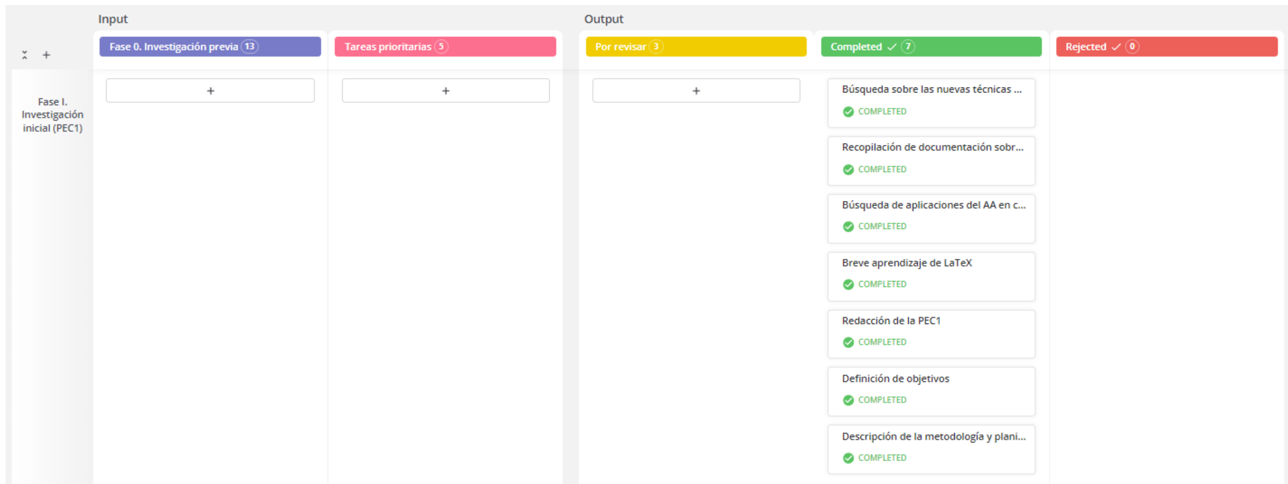


Figura 1.2: Tablero de *Kanban* - Fase I

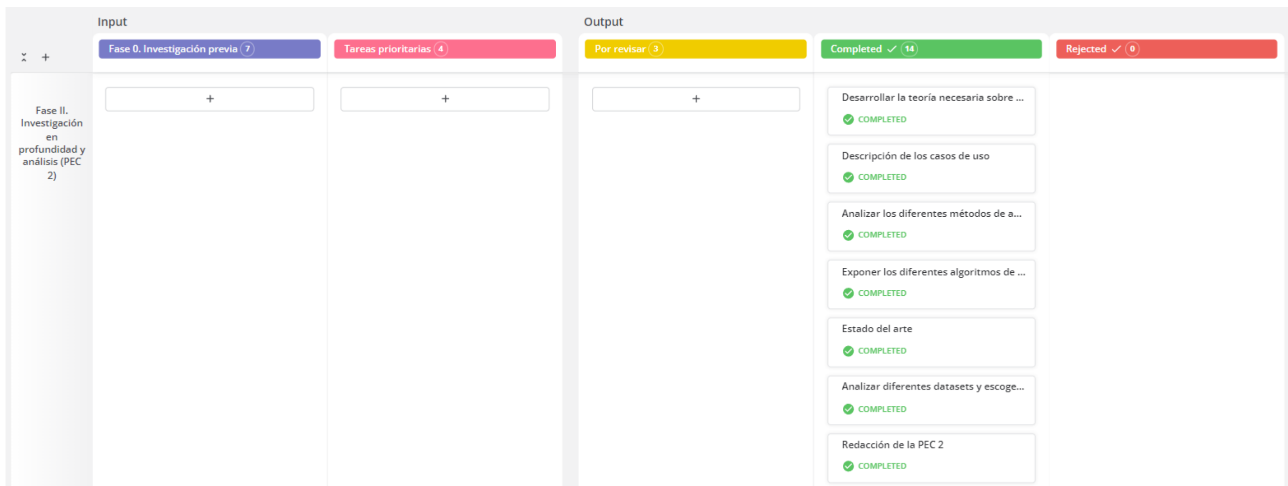


Figura 1.3: Tablero de *Kanban* - Fase II

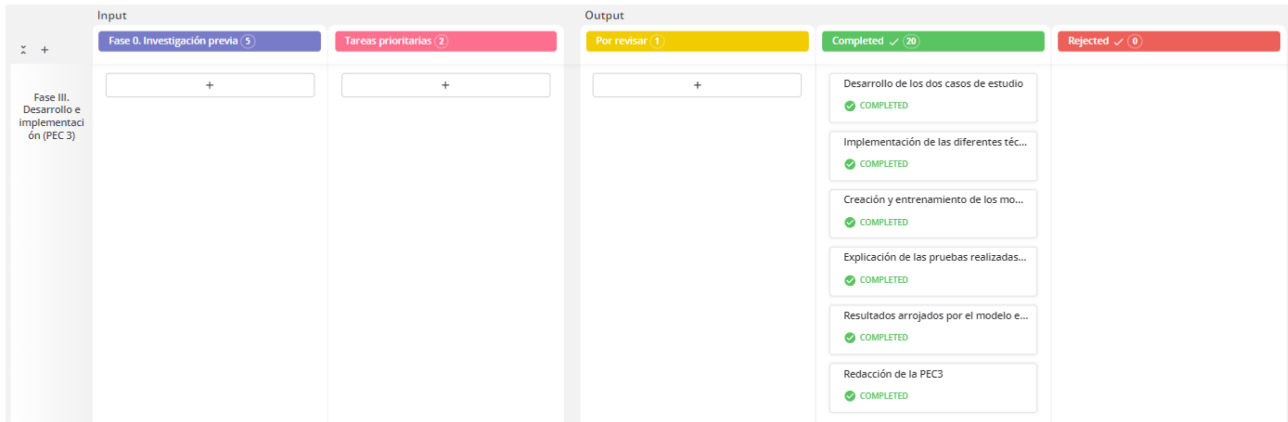


Figura 1.4: Tablero de *Kanban* - Fase III

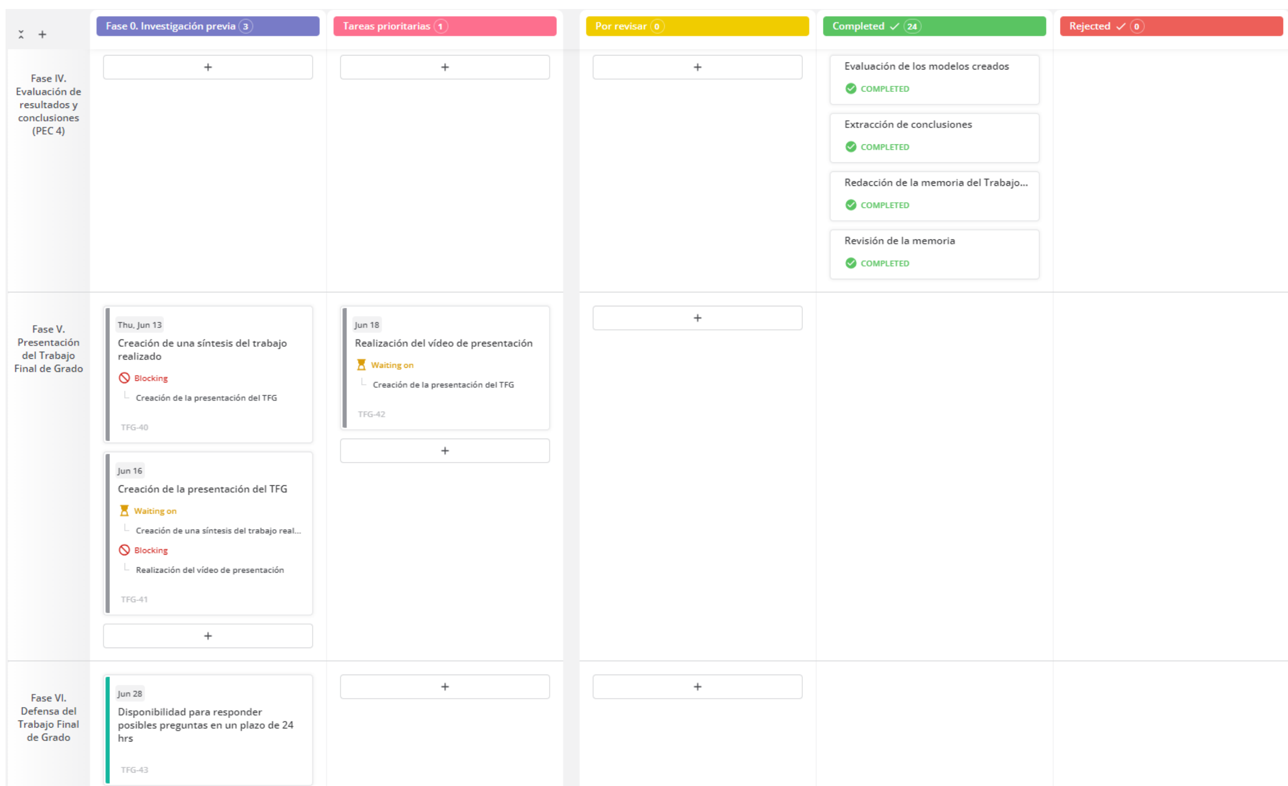


Figura 1.5: Tablero de *Kanban* - Fase IV, V y VI

1.6. Breve sumario de productos obtenidos

Tras la finalización de este TFG, se habrán creado los siguientes modelos:

1. Modelo de aprendizaje automático supervisado capaz de predecir anomalías de red, con el objetivo de contribuir a la detección temprana de ciberataques.

2. Modelo de aprendizaje automático supervisado capaz de detectar malware, con el objetivo de reducir el impacto que pueda tener la ejecución este tipo de código malicioso, al efectuarse una detección temprana de este.

1.7. Breve descripción de los otros capítulos de la memoria

Este TFG se compone de un primer capítulo donde se ha expuesto la introducción al proyecto, el contexto y la justificación del mismo, sus objetivos, su impacto en materias éticas, de sostenibilidad y de diversidad, el enfoque y método seguido, y su planificación.

A continuación, en el segundo capítulo se tratará el estado del arte, donde se expondrá un resumen de la situación actual sobre la incorporación de diversas técnicas de aprendizaje automático al campo de la Seguridad Informática, desde una doble vertiente: por un lado, se verán los últimos avances encontrados referente al empleo del aprendizaje automático para la detección de anomalías de red y, por otro lado, los avances referentes a la detección de malware.

En el tercer capítulo se introducirá la teoría de aprendizaje automático necesaria para el posterior desarrollo del proyecto; donde se verán temas como las diferentes técnicas de análisis de datos, los diferentes algoritmos existentes, o las fases del proceso en sí del aprendizaje automático.

Se continuará con el cuarto capítulo, que versa sobre la teoría de malware necesaria para afrontar la fase de desarrollo, continuando con la teoría de ataques informáticos en la red en el capítulo 5. Posteriormente, se pasará a los capítulos propios de la fase de desarrollo, donde encontraremos la implementación del primer y segundo caso de uso en los capítulos 6 y 7 respectivamente.

Tras la creación y el entrenamiento de los modelos, tan sólo queda pendiente la evaluación de los mismos, que se verá en el octavo capítulo, finalizando con las conclusiones de este TFG en el noveno, y el trabajo futuro en el décimo capítulo.

Capítulo 2

Estado del arte

El Aprendizaje Automático o *Machine Learning* es una rama de la Inteligencia Artificial que lleva ya mucho tiempo acompañando al campo de la Seguridad Informática. Sin embargo, es en los últimos años cuando se produjo un mayor desarrollo en las técnicas de aprendizaje automático centradas en hacer frente a las amenazas cibernéticas cada vez más sofisticadas. En la actualidad existen multitud de proyectos e investigaciones que fusionan estas dos áreas de conocimiento.

Como ya se ha adelantado en el apartado anterior, en las siguientes líneas se expondrá la investigación realizada acerca de los diferentes avances existentes, a día de escribir este TFG, sobre la incorporación de diversas técnicas de aprendizaje automático al campo de la Seguridad Informática.

Esta investigación se detallará desde una doble vertiente: por un lado, los avances encontrados referente al empleo de técnicas de aprendizaje automático para la detección de malware y, por otro lado, los avances referentes a la detección de anomalías de red.

2.1. Avances en el análisis de malware

La detección de código ejecutable malicioso se lleva realizando hace mucho tiempo mediante la aplicación de diferentes técnicas. Sin embargo, respecto a los avances investigados sobre la detección de malware, en este apartado únicamente se detallarán las soluciones existentes basadas en las técnicas de aprendizaje automático, ya que, es en el Capítulo 5 donde se explica la teoría sobre malware y las diversas formas de detección.

Cuando hablamos de malware lo primero que se nos viene a la cabeza como medida preventiva/correctora es un software antivirus. Desde hace varios años, las empresas propietarias de software antivirus referentes en el sector, han visto el enorme potencial que la incorporación de

algoritmos de aprendizaje computacional podía aportar a su negocio.

A continuación, se expondrán los principales antivirus que hacen uso del AA en la detección de malware:

1. **Kaspersky**: Conjuntos de árboles de decisión, hash sensible a la localidad, modelos de comportamiento o agrupación de flujos entrantes. Según sostienen en su web, todos sus métodos de aprendizaje automático (ML) están diseñados para cumplir con los requisitos de seguridad del mundo real: baja tasa de falsos positivos, interpretabilidad y solidez ante un adversario potencial [5].

2. **Microsoft Defender**: Microsoft Defender Advanced Threat Protection (ATP) es la plataforma de seguridad unificada de Microsoft para protección preventiva inteligente, detección posterior a infracciones, investigación y respuesta automatizadas; protege los puntos finales de las amenazas cibernéticas, detecta ataques avanzados y violaciones de datos, automatiza los incidentes de seguridad y mejora la postura de seguridad utilizando una combinación del poder de la nube, análisis de comportamiento y aprendizaje automático [6].

3. **Avast**: Según sostienen en su web, el sistema avanzado de inteligencia artificial (IA) que han desarrollado utiliza el aprendizaje automático para recopilar y extraer automáticamente datos de toda su base de usuarios y luego entrena cada módulo de seguridad. Después de encontrar una nueva muestra de malware, sus productos se actualizan automáticamente con nuevos modelos, lo que brinda una protección crucial y actualizada [7].

2.2. Avances en el análisis de anomalías de red

Respecto al análisis de redes, el campo de la Seguridad Informática cuenta con herramientas como los Sistemas de Detección de Intrusiones (IDS, *Intrusion Detection System*) en sus diferentes variantes: HIDS (IDS de Host), NIDS (IDS de Red), IDS basados en firmas, IDS basados en anomalías, entre otros. Estos sistemas consisten en mecanismos de seguridad cuya finalidad es la detección de accesos no autorizados a una red o a un equipo, a través del análisis de todo el tráfico existente y de todos los dispositivos de la red, buscando actividades maliciosas conocidas.

Como se ha mencionado anteriormente, en el mercado existen diferentes tipos de IDS dependiendo del tipo de técnicas que emplean para la detección y del tipo de dato que analizan. Sin embargo, en este apartado únicamente se detallarán las soluciones de análisis de redes existentes basadas en las técnicas de aprendizaje automático, ya que, es en el Capítulo 4 donde se explica la teoría sobre los ataques informáticos y los fundamentos teóricos de la detección de intrusiones.

A continuación, se expondrán las principales soluciones IDS que hacen uso del AA en la detección de intrusiones y anomalías de red:

1. **OSSEC+**: se trata de un HIDS de código abierto, que tiene como misión principal, llevar un seguimiento detallado sobre las actividades realizadas en un determinado servidor. Una de las novedades respecto a sus versiones pasadas es que actualmente ha incorporado de un sistema de aprendizaje automático para generar alertas sobre posibles amenazas una vez han sido detectadas [8].

2. **RETN**: esta solución posee un modo inteligente que utiliza algoritmos de aprendizaje automático para aprender de los datos históricos del tráfico de red y detectar si la red está bajo algún ataque [9].

3. **Hogzilla IDS**: es una solución de IDS basado en anomalías, de código abierto, que se presenta con la capacidad de detectar ataques *zero-day*, ya que, su motor de detección de anomalías se basa en varios modelos entrenados usando para ello diferentes algoritmos de aprendizaje automático [10].

Aparte de estas soluciones empresariales expuestas en ambos avances, existen numerosas investigaciones y estudios (reflejadas en diferentes publicaciones) sobre el empleo de técnicas de aprendizaje automático tanto para la detección de malware como para la detección de anomalías de red, pero no se exponen aquí para que esta sección no quede demasiado extensa.

Como se puede observar, el aprendizaje automático y la seguridad informática llevan de la mano muchos años.

Capítulo 3

Aprendizaje automático

Se denomina Aprendizaje Automático o *Machine Learning (ML)* a la rama de la Inteligencia Artificial (AI) que "estudia métodos para conseguir incrementar el rendimiento de los sistemas a partir de su experiencia. Los métodos de aprendizaje están fuertemente relacionados con los formalismos de representación del conocimiento, existiendo en la actualidad distintos tipos de métodos de aprendizaje, que corresponden a las distintas alternativas que hay para la representación de este: reglas difusas, lógica de primer orden, redes bayesianas, etc"[11].

Esta rama de la IA intenta imitar las capacidades de aprendizaje inherentes de un ser humano y llevarlas a las máquinas, para que puedan ser capaces de aprender a partir de su propia experiencia. Esta definición nos lleva a la pregunta: ¿cómo hacemos que una máquina aprenda por sí sola?. La respuesta a esta pregunta se encuentra en los datos, de aquí radica la importancia del *Big Data*, mencionado en diversos capítulos de esta memoria.

Hace ya muchos años que expertos en la ciencia de datos, se dieron cuenta que la mejor forma de que una máquina sea capaz de aprender para predecir un determinado resultado, es entrenarla con diferentes ejemplos de predicciones previas realizadas por especialistas en la materia, para que posteriormente, el sistema pueda identificar patrones de comportamiento en estos datos, que le den la capacidad de predecir la respuesta más adecuada a una nueva pregunta.

3.1. Tipos de aprendizaje

Las técnicas de aprendizaje automático consisten en automatizar la identificación de patrones ocultos tras los datos, mediante la aplicación de diferentes algoritmos. Estas técnicas comienzan aprendiendo de los datos, luego extraen conocimiento de ellos y van generando un modelo que sea capaz de resolver un problema determinado sin haber tenido que programar una solución específica para ello.

En la creación de estos modelos, la elección del tipo de algoritmo a utilizar no es una tarea

fácil sino que dependerá del problema que se quiera resolver. Según cuál sea el problema al que nos enfrentemos, los tipos de aprendizaje se pueden clasificar en: aprendizaje supervisado, no supervisado, por refuerzo y aprendizaje profundo.

3.1.1. Aprendizaje supervisado

Este tipo de aprendizaje se llama supervisado porque utiliza como datos de entrenamiento un dataset que contiene datos etiquetados, es decir, contiene la clasificación de cada uno de los datos mediante una etiqueta. En este método de aprendizaje es importante conocer cómo están clasificados los datos, ya que, a partir de las etiquetas, se buscará predecir el valor que tendrá esa propiedad para un nuevo conjunto de datos sin etiquetar.

El aprendizaje supervisado clasifica un dato y después obtiene la medida del error de dicha clasificación, de tal manera que el propio modelo se va autoajustando con el objetivo de reducir este error. Es por ello, que después de cada iteración del modelo, aumenta la precisión con la que este es capaz de clasificar el dataset.

3.1.2. Aprendizaje no supervisado

En este tipo de aprendizaje, por el contrario, se utiliza como datos de entrenamiento un dataset sin ninguna clasificación (datos no etiquetados), donde el objetivo es descubrir relaciones implícitas en estos datos, desconocidas en un principio.

En el aprendizaje no supervisado, el uso de los diferentes algoritmos permite clasificar varios elementos bajo un mismo grupo atendiendo al estudio de sus características. Este tipo de aprendizaje analiza los datos de manera iterativa y los ordena en grupos según los patrones detectados sin partir de un conocimiento previo de los datos.

3.1.3. Aprendizaje por refuerzo

En el Aprendizaje por refuerzo, para entrenar el modelo se utiliza la retroalimentación de las acciones llevadas a cabo por el modelo en un determinado entorno. En otras palabras, este aprendizaje examina y aprende del entorno en el que se encuentra mediante la ejecución de diferentes acciones que le darán la posibilidad de alcanzar el objetivo final. Este método de aprendizaje se basa en el modelo de prueba y error, donde un agente recibe una serie de recompensas o penalizaciones dependiendo de las acciones tomadas. El objetivo por parte del agente será obtener la máxima puntuación posible y esto se consigue cuando es capaz de solucionar el problema.

Por tanto, este tipo de aprendizaje no aprende directamente de los datos sino de los resultados obtenidos en las distintas pruebas realizadas sobre estos.

3.1.4. Aprendizaje profundo

Finalmente, este tipo de aprendizaje se basa en las redes neuronales, quienes fueron diseñadas con el objetivo de imitar el funcionamiento de las neuronas humanas. Se denomina Aprendizaje Profundo o *Deep Learning (DL)* porque este se produce de manera iterativa a través de numerosos niveles; partiendo de un modelo jerarquizado, se crea el aprendizaje a través de un gran número de niveles, donde la red neuronal realiza una tarea iterativa de ajustes en cada nivel que ayuda a mejorar de manera gradual el resultado, permitiendo con esto el aprendizaje progresivo.

El aprendizaje profundo suele emplearse cuando se dispone de un dataset desestructurado sin etiquetar. El número de nivele de la red neuronal será proporcional a la complejidad del problema a resolver.

3.2. Algoritmos de aprendizaje automático

En la siguiente ilustración (Figura 3.1), podemos observar una clasificación de los algoritmos existentes según su funcionamiento, elaborada por una de las bibliotecas de aprendizaje automático más importantes en el sector (Scikit-learn).

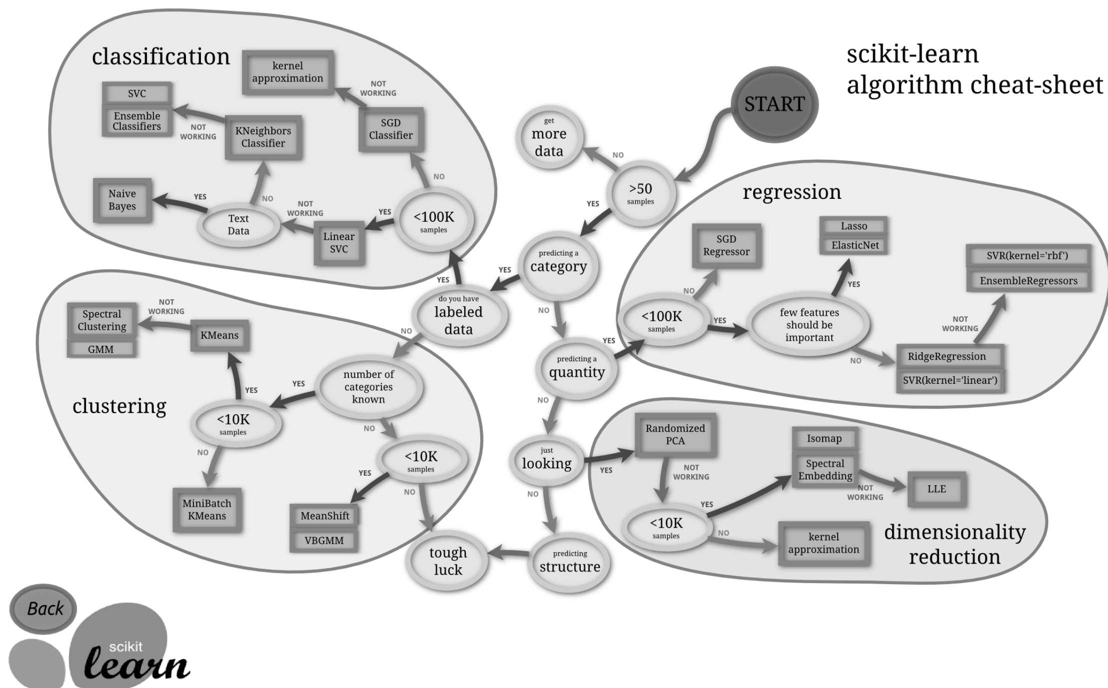


Figura 3.1: Algoritmos de aprendizaje automático. Fuente: [12]

3.3. Algoritmos empleados en el desarrollo de este TFG

A continuación, se explicarán los algoritmos que se utilizarán para la fase de desarrollo de este Trabajo Final de Grado, alojada en el repositorio: `https://github.com/ptposa/TFG`.

3.3.1. Algoritmos basados en modelos lineales

1. **Algoritmo de regresión lineal**

   Este algoritmo busca la ecuación lineal que describe mejor la correlación de las variables explicativas con la variable dependiente. Se logra ajustando una línea a los datos mediante mínimos cuadrados. La línea intenta minimizar la suma de los cuadrados de los residuales. El residual es la distancia entre la línea y el valor real de la variable explicativa. Encontrar la línea de mejor ajuste es un proceso iterativo. [13]

   Este modelo puede ser expresado como:

$$Y = \beta_O + \beta_1 X_1 + \cdots + \beta_m X_m + \epsilon \tag{3.1}$$

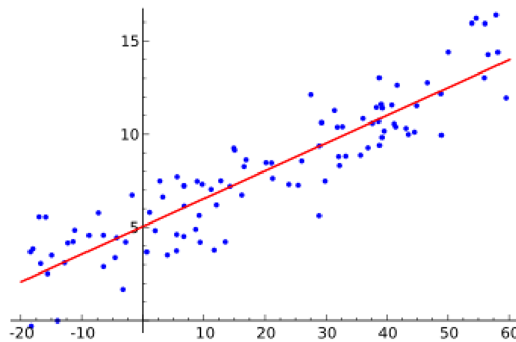


Figura 3.2: Ejemplo de una regresión lineal. Fuente: Wikipedia

2. **Algoritmo de regresión logística**

   El método de regresión logística es un método estadístico que se usa para resolver problemas de clasificación binaria, donde el resultado solo puede ser de naturaleza dicotómica, o sea, solo puede tomar dos valores posibles. Por ejemplo, se puede utilizar para detectar la probabilidad que ocurra un evento. [14]

La función logística también es llamada función Sigmoide. Que se describe con la siguiente fórmula:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{3.2}$$

Esta función produce una curva en forma de S, capaz de convertir y mapear cualquier cifra a un valor numérico encuadrado dentro del intervalo [0,1], sin llegar nunca a los límites de dicho intervalo (Figura 3.3).

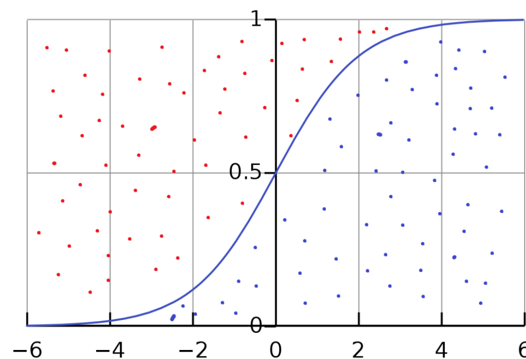


Figura 3.3: Función sigmoide. Fuente: Wikipedia

3.3.2. Algoritmos de agrupamiento (Clustering)

1. **Algoritmo K-Means**

K-means es un método de agrupamiento basado en centroides, que tiene como objetivo la partición de un conjunto de n observaciones en k grupos, en el que cada observación pertenece al grupo cuyo valor medio es más cercano. [15]

El algoritmo k-means resuelve un problema de optimización, siendo la función a optimizar (minimizar) la suma de las distancias cuadráticas de cada objeto al centroide de su cluster. [16]

Los objetos se representan con vectores reales de $\boldsymbol{d}$ dimensiones $(\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n)$ y el algoritmo k-means construye $\boldsymbol{k}$ grupos, donde se minimiza la suma de distancias de los objetos, dentro de cada grupo $\mathbf{S} = \{S_1, S_2, \dots, S_k\}$, a su centroide.

Este modelo se puede formular como:

$$\min_{\mathbf{S}} E\left(\boldsymbol{\mu_i}\right) = \min_{\mathbf{S}} \sum_{i=1}^{k} \sum_{\mathbf{x}_j \in S_i} \left\|\mathbf{x}_j - \boldsymbol{\mu}_i\right\|^2 \tag{3.3}$$

donde $\boldsymbol{S}$ es el conjunto de datos cuyos elementos son los objetos $\mathbf{x}_j$ representados por vectores, donde cada uno de sus elementos representa una característica o atributo. Tendremos $\boldsymbol{k}$ grupos o clusters con su correspondiente centroide $\boldsymbol{\mu_i}$.

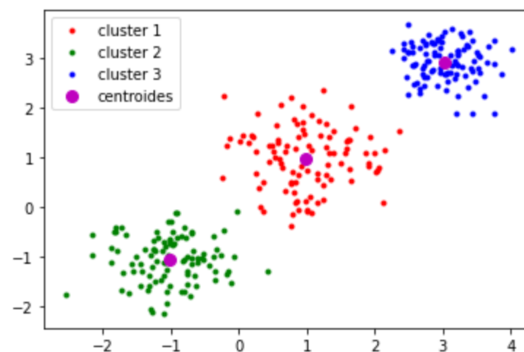


Figura 3.4: Ejemplo de algoritmo K-Means. Fuente: [16]

2. **Algoritmo K-nn (k-nearest neighbors)**

El método de los k vecinos más cercanos es un método de agrupamiento basado en densidad, que busca agrupar los putos por cercanía a los puntos de su alrededor. Este algoritmo se emplea para agrupar datos existentes de los que se desconocen las características que pueden tener en común, ya que, se basa en el concepto de que proximidad equivale a similitud, es decir, los elementos más próximos entre sí, pertenecen a la misma clase. [17]

Supongamos que tenemos un conjunto de datos donde X es una matriz de características de una observación y, Y es una etiqueta de clase. K-nn es un método de clasificación que estima la distribución condicional de Y dado X y clasifica una observación en la clase con mayor probabilidad. Dado un entero positivo k, K-nn busca las k observaciones más cercanas a una observación de prueba x_0 y estima la probabilidad condicional de que pertenezca a la clase j, usando la siguiente fórmula:

$$Pr(Y = j|X = x_0) = \frac{1}{k} \sum_{i \in \mathcal{N}_0} I(y_i = j) \tag{3.4}$$

Para poder seleccionar los K vecinos más cercanos a un nuevo elemento, el algoritmo K-nn calcula las distancias usando para ello la distancia euclídea:

$$d(x, y) = \sqrt{\sum_{i=1}^{p} (x_i - y_i)^2} \tag{3.5}$$

Para la elección del valor de K no existe un método fijo, sino que debemos ir probando con distintos valores y elegiremos el valor que menos errores produzca. Cuanto mayor sea el valor de K, menos errores producirá, siempre y cuando no se pierda el límite entre clases.

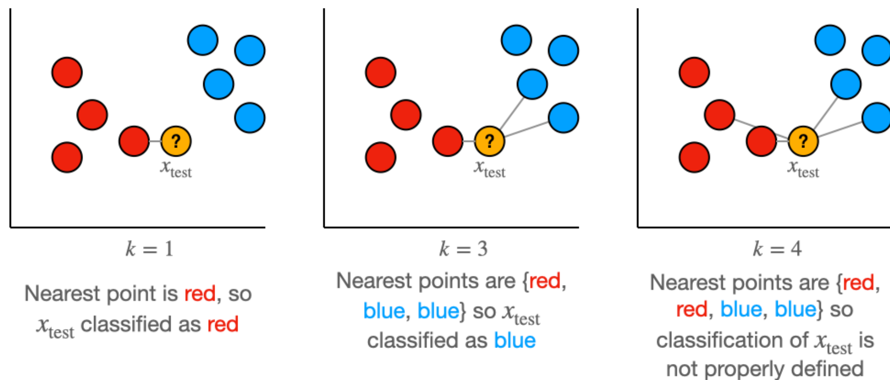


Figura 3.5: Funcionamiento del algoritmo K-nn. Fuente: [18]

3.3.3. Algoritmos basados en árboles de decisión

1. **Árboles de decisión** (*Decision Tree*)

   Un árbol de decisión es un algoritmo de aprendizaje automático que se utiliza tanto para tareas de clasificación como de regresión. Tiene una estructura de árbol jerárquica, que consta de un nodo raíz, ramas, nodos internos y nodos hoja.

   a) El nodo raíz representa donde comienza el árbol de decisión, ya que no tiene ramas entrantes.

   b) Las ramas representan la decisión en función de una determinada condición.

   c) Los nodos de decisión representan cada una de las características o propiedades a considerar para tomar una decisión.

d) Los nodos hoja representan todos los resultados posibles dentro del conjunto de datos.

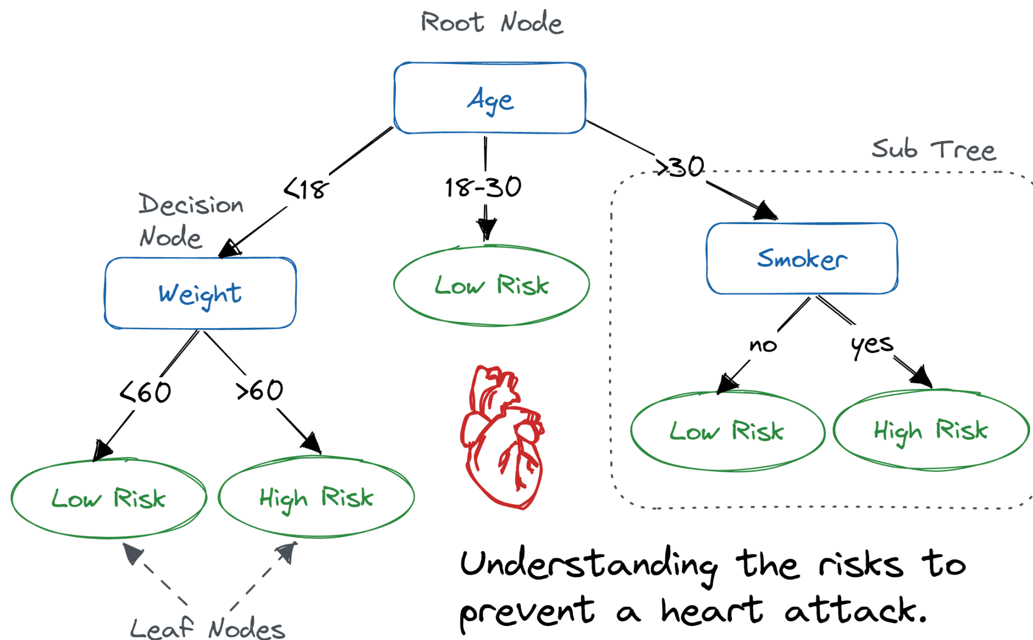


Figura 3.6: Ejemplo de árbol de decisión. Fuente: [19]

Tras entender de qué partes se compone un árbol de decisión, debemos pasar a uno de los aspectos más importantes a tener en cuenta en el empleo de este algoritmo: cómo podemos elegir el mejor atributo en cada uno de los nodos.

Si bien hay varias formas de seleccionar el mejor atributo en cada nodo, la *ganancia de información* y la *impureza de Gini* son dos de los métodos más usados por los profesionales del sector hoy en día. Estos métodos ayudan a evaluar la calidad de cada condición de prueba y qué tan bien podrá clasificar las muestras en una clase.

Para explicar la ganancia de información primero debemos explicar la entropía, ya que es un concepto que se encuentra íntimamente ligado a este método. La entropía es un concepto que mide la impureza de los valores de la muestra. Se define con la siguiente fórmula:

$$E(S) = \sum_{i=1}^{c} -p_i \log_2 p_i \tag{3.6}$$

donde: S representa el conjunto de datos en el que se calcula la entropía, i representa las clases en conjunto S y, p_i representa la probabilidad de los posibles valores.

Los valores de entropía pueden estar entre 0 y 1. Si todas las muestras en el conjunto de datos, S, pertenecen a una clase, entonces la entropía será igual a cero. Si la mitad de las muestras se clasifican en una clase y la otra mitad en otra clase, la entropía estará en su punto más alto en 1. Para seleccionar la mejor característica para dividir y encontrar el árbol de decisión óptimo, se debe usar el atributo con la menor cantidad de entropía. La ganancia de información representa la diferencia de entropía antes y después de una división en un atributo determinado. El atributo con la ganancia de información más alta producirá la mejor división, ya que hace el mejor trabajo al clasificar los datos de entrenamiento de acuerdo con su clasificación de destino.

La ganancia de información se representa con la siguiente fórmula:

$$\text{Information Gain } (S, A) = E(S) - \sum_{v \in V(A)} \frac{|Sv|}{|S|} E(Sv) \tag{3.7}$$

donde: $E(S)$ representa la entropía del conjunto de datos (S), S representa los objetos del conjunto de datos, A representa los atributos de los objetos y $V(A)$ representa el conjunto de valores que puede tomar A.

Llegados a este punto, sólo nos queda hablar de la impureza de Gini. La *impureza de Gini* es la probabilidad de clasificar incorrectamente un punto de datos aleatorio en el conjunto de datos si se etiquetara en función de la distribución de clases del conjunto de datos. Similar a la entropía, si el conjunto S es puro (es decir, pertenece a una clase), entonces su impureza es cero. Esto se denota mediante la siguiente fórmula:

$$\text{Gini Impurity } = 1 - \sum_{i=1}^{C} (p_i)^2 \tag{3.8}$$

Fuente de consulta: IBM [20]

2. **Bosques aleatorios** (*Random Forest*)

Los bosques aleatorios son un método de aprendizaje conjunto, principalmente empleado para tareas de clasificación y regresión, que opera mediante la construcción de una

multitud de árboles de decisión en el momento del entrenamiento. Para las tareas de clasificación, la salida del bosque aleatorio es la clase seleccionada por la mayoría de los árboles. Para tareas de regresión, se devuelve la predicción media o promedio de los árboles individuales. Los bosques de decisiones aleatorias corrigen el hábito de los árboles de decisión de sobreadaptarse a su conjunto de entrenamiento.

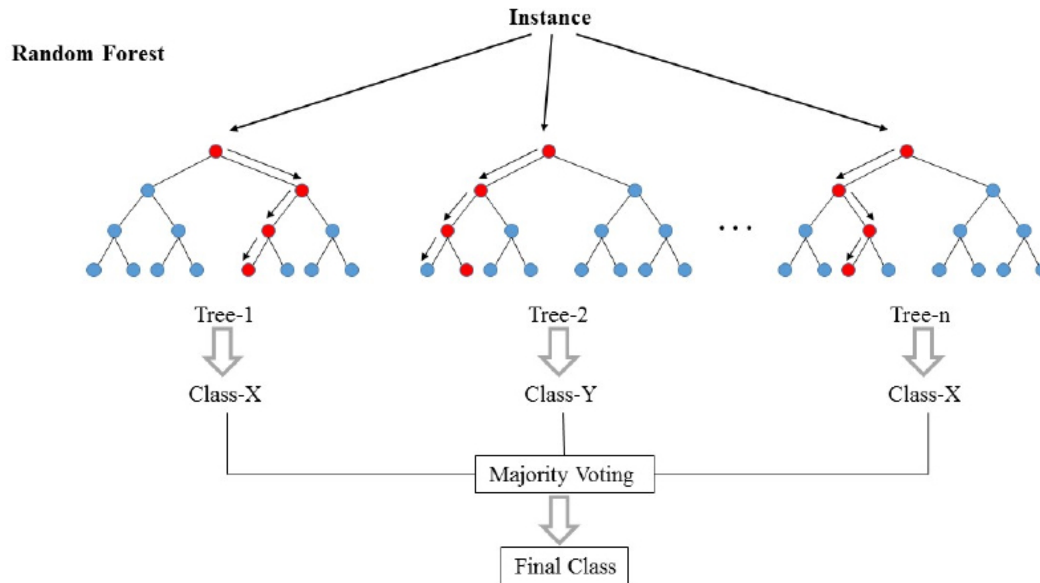


Figura 3.7: Ilustración del funcionamiento de los bosques aleatorios. Fuente: Wikipedia [21]

3.3.4. Algoritmos probabilísticos

Son algoritmos empleados para tareas de clasificación y de regresión que utilizan el Teorema probabilístico de Naive Bayes. Los algoritmos de esta clasificación más utilizados son: algoritmo Naïve Bayes, algoritmo Naïve Bayes Gaussiano y algoritmo Naïve Bayes Multinomial. Estos modelos son llamados algoritmos *Naïve*, o *Inocentes* en castellano. En ellos, se asume que las variables predictoras son independientes entre sí, es decir, que la presencia de una cierta característica en un conjunto de datos no está en absoluto relacionada con la presencia de cualquier otra característica.

Para el desarrollo de este TFG, únicamente emplearemos el algoritmo Naïve Bayes, que es el que detallaremos a continuación.

1. **Algoritmo Naïve Bayes**

   El término *naïve (ingenuo)* hace referencia a la forma en que el algoritmo analiza las características de un determinado conjunto de datos, ya que este ignora la correlación

entre las características del dataset (features).

Supongamos, para explicar este algoritmo, que tenemos un dataset de tipos de pelotas de deporte y dentro de él aparece un registro que corresponde a una pelota que está etiquetada como `Color: amarillo fluorescente`, `Diámetro: pequeño` y `Tipo de pelota: tenis`. Tal y como hemos mencionado anteriormente, si borramos la columna objetivo (tipo de pelota), el algoritmo no tendrá en cuenta la correlación entre las otras características (atributos) del dataset para clasificar las pelotas, ya que tratará a cada característica de manera independiente.

Color	Diámetro	Tipo de pelota
Amarillo fluorescente	Pequeño	?

Cuadro 3.1: Ejemplo de funcionamiento algoritmo Naïve Bayes

Es importante destacar que este modelo también asume que todas las características del dataset tienen la misma importancia para el resultado.

Este teorema se basa en calcular la probabilidad de que ocurra un cierto **evento A**, dado que ya ha ocurrido otro **evento B** anterior, lo que se denomina probabilidad condicional.

Este modelo se formula matemáticamente de la siguiente manera:

$$P(A \mid B) = \frac{P(B \mid A) * P(A)}{P(B)} \tag{3.9}$$

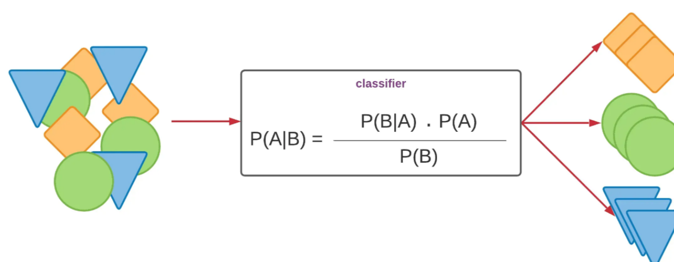


Figura 3.8: Algoritmo Naïve Bayes. Fuente: [22]

Para entender cómo funciona el algoritmo de Naïve Bayes, volvamos al ejemplo del tenis. En esta ocasión, tenemos un dataset con diferentes partidos de tenis jugados y suspendi-

dos, atendiendo a factores climáticos, y debemos clasificar si los jugadores van a jugar o no en base a la previsión meteorológica.

Clima	Partido disputado
Soleado	No
Nubleado	Sí
Lluvioso	Sí
Nubleado	No
Soleado	Sí
Soleado	Sí
Nubleado	Sí
Lluvioso	No
Nubleado	Sí
Nubleado	Sí

Cuadro 3.2: Dataset de ejemplo algoritmo Naïve Bayes

Para aplicar el algoritmo Naïve Bayes seguiremos los siguientes pasos:

Paso 1. Convertimos el dataset en una tabla de frecuencias

Clima	Sí	No
Soleado	2	1
Nubleado	4	0
Lluvioso	1	2
Total	7	3

Cuadro 3.3: Ejemplo de tabla de frecuencias algoritmo Naïve Bayes

Paso 2. Creamos una tabla de probabilidad

Clima	Sí	No	Probabilidad
Soleado	2	1	$3/10 = 0.3$
Nubleado	4	0	$4/10 = 0.4$
Lluvioso	1	2	$3/10 = 0.3$
Total	7	3	-
Probabilidad	$7/10 = 0.7$	$3/10 = 0.3$	-

Cuadro 3.4: Ejemplo de tabla de probabilidad algoritmo Naïve Bayes

Paso 3. Calculamos la probabilidad posterior de cada clase, usando la ecuación de Naïve Bayes anteriormente expuesta. Para este ejemplo, únicamente calcularemos la probabilidad de que un jugador dispute un partido de tenis si hace un día soleado.

Por tanto, tenemos:

$$P(A \mid Soleado) = \frac{P(Soleado \mid A) * P(A)}{P(Soleado)} = \frac{\frac{2}{7} * \frac{7}{10}}{\frac{3}{10}} = 0{,}67 \tag{3.10}$$

Con este resultado concluímos que hay un 67 por ciento de posibilidades de que un jugador juegue al tenis con tiempo soleado.

3.3.5. Algoritmo SVM (Máquina de vectores de soporte)

Las máquinas de vectores de soporte (SVM) son un conjunto de métodos de aprendizaje que se utilizan para la clasificación, la regresión y para la detección de valores atípicos. En este TFG se empleará el algoritmo SVM únicamente para tareas de clasificación, por tanto, nos centraremos en explicar cómo funciona SVM como clasificador.

Supongamos que tenemos un conjunto de datos en el que los registros pertenecen a dos clases, por ejemplo, a una clase azul y a una clase roja, que podrían representar dentro del caso de uso 1, a los archivos legítimos y al malware respectivamente. Para clasificar todos los registros del dataset, el algoritmo SVM comenzará dibujando una línea de separación entre las dos clases de modo que los registros que pertenezcan a la clase roja (malware) estén en un lado de la línea y los registros que pertenezcan a la clase azul (archivos legítimos) estén en el otro lado de la línea. Esta línea que separa las dos clases se conoce como *Límite de Decisión* o *Hiperplano*.

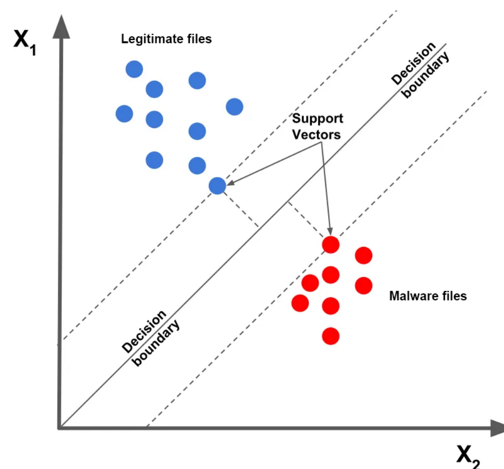


Figura 3.9: Separación de clases en algoritmo SVM.

Sin embargo, para separar dos clases, pueden existir diferentes hiperplanos posibles, tal y como se refleja en la siguiente ilustración, donde los hiperplanos aparecen representados por H1, H2 y H3.

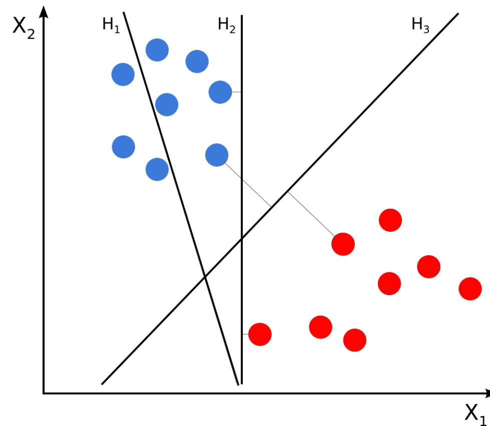


Figura 3.10: Existencia de diferentes hiperplanos en algoritmo SVM.

El objetivo principal del clasificador de vectores de soporte es encontrar el hiperplano de separación óptimo (límite de decisión). Este hiperplano es aquel que separa perfectamente todos los registros del dataset en dos clases distintas, de forma que el margen entre ambas clases sea el máximo posible. En la ilustración anterior, es hiperplano más adecuado es el H3, ya que posee un mayor margen que el H2, siendo el H1 descartado por una mala clasificación de las clases.

Es importante destacar que el algoritmo SVM no es únicamente un clasificador binario, sino que para clasificar los datos solo separa los registros en dos clases a la vez.

Terminología básica

1. *Vectores*: Son los registros del dataset.
2. *Vectores soporte (support vectors)*: Subconjunto de datos más cercano al hiperplano.

Otro aspecto a tener en cuenta es que muchas veces, las clases no son linealmente separables. Para resolver este problema SVM utiliza el truco del núcleo (Kernel Trick). El **truco del núcleo** es un método utilizado por el algoritmo SVM para convertir datos que no son separables linealmente en un espacio de características de mayor dimensión donde se pueden separar linealmente.

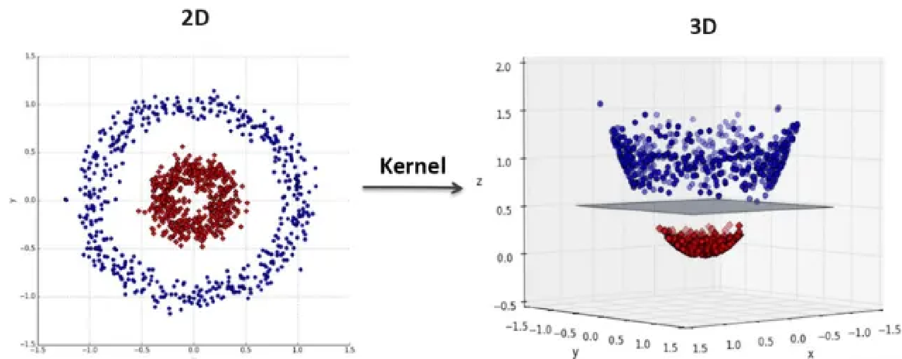


Figura 3.11: Truco del núcleo (Kernel Trick).

Cuando se utiliza el truco del kernel, es importante elegir una función del kernel apropiada que asigne los datos de entrada a un espacio de dimensiones superiores donde se puedan separar más fácilmente. Existen distintos tipos de funciones kernel.

3.4. ¿Qué es un modelo de aprendizaje automático?

Un modelo de aprendizaje automático o machine learning es un objeto (almacenado localmente en un archivo), que es el resultado de entrenar a un determinado algoritmo de aprendizaje con un conjunto de datos específico, con el objetivo de que sea capaz de reconocer ciertos tipos de patrones. El algoritmo elegido le ayudará al modelo a razonar por sí mismo y a aprender de los datos introducidos.

Una vez que el modelo se encuentre entrenado, este será capaz de razonar sobre datos que no ha visto antes y de hacer predicciones sobre esos datos.

3.5. Ruta seguida para el desarrollo de este TFG

Para la fase de desarrollo de este TFG se empleará el tipo de aprendizaje automático **supervisado**, tanto para el primer caso de uso como para el segundo. Los algoritmos que se emplearán se han detallado en el apartado 3.3, y con esto, sólo nos falta una variable de la ecuación: el conjunto de datos de cada caso de uso. Estos datasets han sido mencionados en el apartado 1.4, donde se puede observar que para el caso de uso 1 se creará un dataset a partir de muestras de malware actuales, proporcionadas por las plataformas VirusTotal [1] y VirusShare [2], y, para el caso de uso 2 se empleará el NSL-KDD dataset [4].

Por tanto, el proceso que seguiremos para la creación de los modelos es el siguiente:

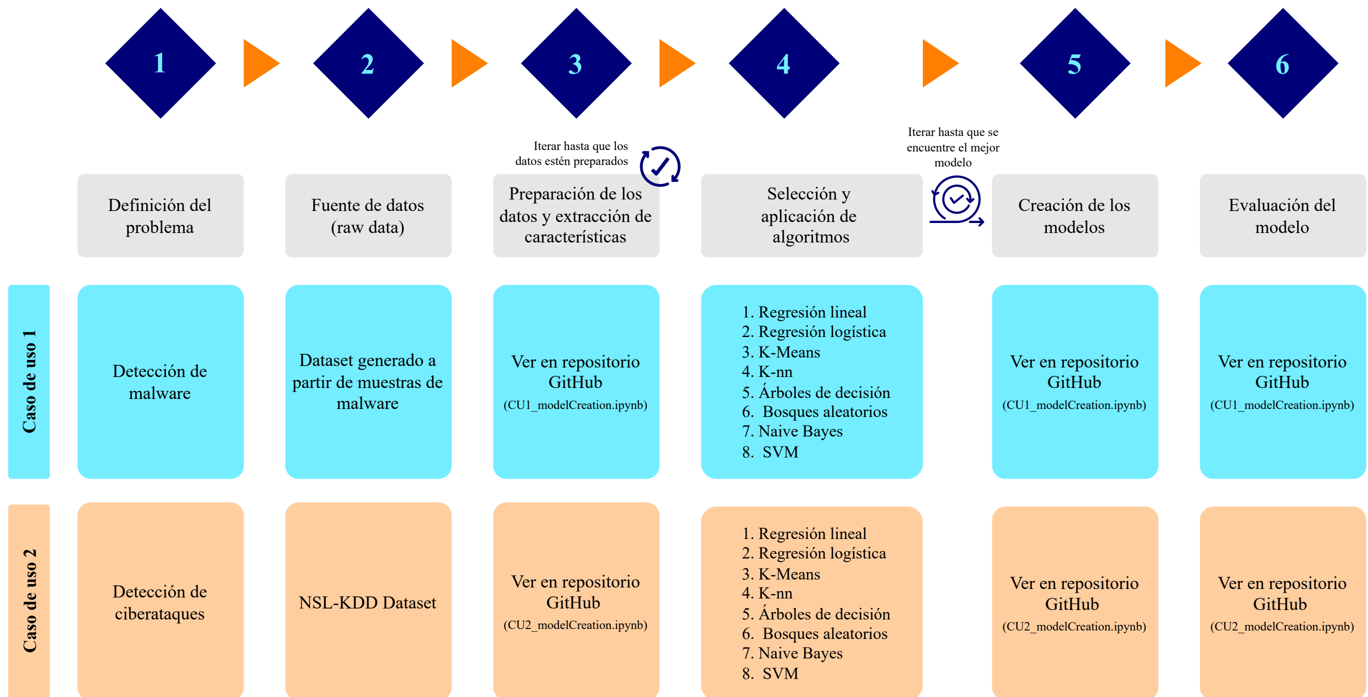


Figura 3.12: Proceso de creación de modelos seguido en este TFG.

La creación de los modelos se encuentra alojada en el siguiente repositorio de GitHub:

`https://github.com/ptposa/TFG`

Capítulo 4

Malware

Cuando las centrifugadoras nucleares en la instalación de enriquecimiento de uranio de Natanz (Irán) dejaron inexplicablemente de funcionar en enero de 2010, nadie sabía con seguridad lo que estaba sucediendo. Tras largos meses de investigación, en junio de 2010, una compañía de seguridad informática llamada *VirusBlokAda* descubrió que un sofisticado gusano informático, llamado *Stuxnet*, había sido el responsable de la primer ciberarma del mundo que había afectado a una infraestructura física.

El objetivo de Stuxnet eran las centrifugadoras nucleares iraníes, y dañó y destruyó capacidades militares claves, y causó importantes trastornos en el programa nuclear de Irán. Fue diseñado específicamente para dirigirse a sistemas de control industrial que ejecutaban software de Siemens en entornos Microsoft Windows. A día de hoy, se cree que Stuxnet había estado activo al menos desde 2007, lo que significa que este malware pasó desapercibido dentro de una infraestructura crítica, por alrededor de tres años.

Este Trabajo Final de Grado pretende realizar un acercamiento a las grandes ventajas que puede ofrecer la incorporación de técnicas de aprendizaje automático supervisado a la detección temprana de malware y de ciberataques. Recordemos que el poder del malware, no sólo radica en su carga útil (payload) sino también en su propagación; y cuando un malware se propaga se produce un ciberataque. Por lo tanto, podemos ver cómo se encuentran íntimamente relacionados los dos casos de uso propuestos, y cómo el gusano Stuxnet es un claro ejemplo de lo que este TFG busca ayudar a mitigar.

4.1. Tipos de malware

El malware, también llamado código malicioso, se refiere a cualquier código de software o programa informático escrito de manera intencionada para dañar un sistema informático o a sus usuarios. Hoy en día, casi todos los ciberataques modernos implican algún tipo de malware embebido. Estos códigos maliciosos pueden adoptar muchas formas, desde un gusano hasta

adware, dependiendo de lo que los ciberdelincuentes pretendan conseguir.

A continuación, realizaremos una breve clasificación de los principales tipos de malware existentes:

1. **Troyanos**. Son un tipo de malware que se presenta como software legítimo o útil para engañar a los usuarios y lograr que lo descarguen e instalen en sus sistemas.
2. **Gusanos**. Los gusanos tienen la capacidad de replicarse y propagarse automáticamente a través de redes y sistemas informáticos sin la intervención del usuario.
3. **Virus**. Son programas maliciosos diseñados para infectar archivos y sistemas informáticos, propagándose de una computadora a otra cuando los usuarios comparten archivos o ejecutan programas infectados.
4. **Keyloggers**. También conocidos como registradores de pulsaciones de teclas, son programas maliciosos diseñados para registrar y almacenar las pulsaciones de teclas realizadas por un usuario en un teclado de computadora.
5. **Bots y botnets**. Los bots son programas informáticos diseñados para realizar automáticamente tareas específicas en internet. Por otro lado, una botnet es una red de computadoras infectadas con bots que están bajo el control de un atacante o un operador malintencionado.
6. **Bombas lógicas**. Es un tipo de código malicioso que se activa en respuesta a un evento específico o después de que se cumpla cierta condición.
7. **Ransomware y malware de cifrado**. El ransomware y el malware de cifrado son tipos específicos de software malicioso diseñados para cifrar archivos en un sistema informático y exigir un rescate a cambio de proporcionar la clave de descifrado.
8. **Spyware**. Es un tipo de software malicioso diseñado para recopilar información sobre las actividades de un usuario en su dispositivo sin su conocimiento ni consentimiento.
9. **Adware**. Es un tipo de software que muestra anuncios publicitarios en un dispositivo, a menudo de manera intrusiva y no deseada.
10. **Rootkit**. Es un tipo de software malicioso diseñado para ocultar la presencia de otras amenazas cibernéticas en un sistema, permitiendo así que los atacantes mantengan el acceso no autorizado y el control sobre la computadora infectada.
11. **Backdoor *(Remote Administration Tools)***. Es una puerta trasera o una entrada oculta en un sistema o software que permite el acceso no autorizado al sistema sin pasar por los métodos de autenticación normales.

4.2. ¿Qué se esconde detrás de un malware?

Al margen de la clasificación expuesta en el apartado anterior, hoy en día existen distintos tipos de malware creados para diferentes tipos de software. Cada uno de estos, puede estar escrito en diversos lenguajes de programación, dirigido a diferentes entornos y con diferentes requisitos de ejecución.

Teniendo acceso al código fuente de alto nivel (en lenguajes como C, C++, Java, JavaScript o Python), resulta bastante sencillo determinar las acciones del programa y analizar su comportamiento. Sin embargo, lo más probable es que no se tenga acceso al código de alto nivel utilizado para producir dicho malware.

La mayoría del código malicioso se detecta y recopila en la red. En su forma empaquetada, la mayor parte del malware se presenta como archivos binarios, los cuales generalmente no son legibles por humanos y están diseñados para ejecutarse directamente en una máquina. Estos archivos binarios son por naturaleza confusos, lo que representa grandes desafíos para los analistas de malware que intentan extraer información de ellos. Si no se conoce el propósito del programa (contexto), el lenguaje de codificación que se ha empleado o el algoritmo de decodificación, los datos binarios carecen de significado por sí mismos.

Analizar las características y el comportamiento de dicho código se convierte entonces en un proceso de ingeniería inversa, cuyo objetivo es descubrir qué está haciendo el programa. Para identificar y extraer características útiles para un análisis de seguridad posterior de esos archivos binarios, es necesaria una comprensión profunda de los aspectos internos del fichero que se está manejando. A través de la ingeniería inversa de un binario, podemos entender su funcionalidad, propósito e incluso, en ocasiones, su origen.

El malware puede integrarse en una variedad de formatos binarios distintos, los cuales funcionan de maneras muy diferentes entre sí. Por ejemplo, los archivos Microsoft Windows PE (Portable Executable, con extensiones .exe, .dll, .efi), los archivos ELF de Unix (formato ejecutable y vinculable) y, los archivos APK de Android (Android Package Kit, con extensión .apk) tienen estructuras de archivos y contextos de ejecución muy diferentes. Por supuesto, los conocimientos previos necesarios para analizar cada tipo de ejecutables también son distintos.

En este TFG nos centraremos en el malware alojado en el formato de archivos Microsoft Windows PE (Portable Executable). Por esta razón, en las siguientes líneas se dará una breve introducción teórica a este formato de archivos, con el objetivo de poder fusionar todos estos conceptos teóricos con la parte de desarrollo alojada en el repositorio de GitHub, y alcanzar una compresión integral del funcionamiento del modelo de aprendizaje automático creado.

4.3. Malware para la plataforma Microsoft Windows

Para la fase de desarrollo del primer caso de uso, se empleará el formato de archivos PE (Portable Executable); un formato de archivo para ficheros ejecutables usados en versiones de 32 y 64 bits del sistema operativo Microsoft Windows. Por lo tanto, centraremos la explicación teórica en este tipo de archivo.

4.3.1. Identificación de un archivo PE

En esta sección analizaremos cómo identificar unívocamente a un fichero portable ejecutable de la plataforma sobre la que trabajaremos: Microsoft Windows. Actualmente, sabemos que cualquier máquina computacional únicamente entiende ceros y unos, por tanto, cada archivo en nuestro sistema operativo será un archivo binario. Sin embargo, en muchas ocasiones hemos escuchado la idea errónea de que todos los archivos binarios son ejecutables.

Todos los ficheros de nuestro ordenador (archivos ejecutables, documentos de Microsoft Word, documentos de texto, documentos PDF, imágenes, etc) se manifiestan en el sistema operativo en forma de un archivo binario. Sin embargo, cuando se abren, cada archivo se presenta al usuario de forma diferente, dependiendo principalmente (en entornos Windows) de la extensión del fichero.

En los sistemas operativos Windows, la extensión de un archivo es un sufijo del nombre del archivo, que suele ser un punto seguido de tres letras que identifican el tipo de archivo. Los ficheros PE tienen como extensión: .exe, .dll, .sys, .cpl, .ocx, .scr y .drv.

Sabemos que actualmente existen muchas formas de ocultación de la extensión real de un archivo, o incluso sabemos que los archivos de malware se sitúan en el sistema operativo objetivo sin nombres legibles ni extensión alguna. Por todo ello, a continuación se presentará la forma confiable de identificar el tipo de archivo inequívocamente.

Para la siguiente explicación, vamos a abrir el ejecutable de la calculadora de Windows (*calc.exe*) con el editor hexadecimal de Notepadd++. Con esto estaremos accediendo a la versión hexadecimal del código fuente del archivo ejecutable de la calculadora.

En la siguiente ilustración, que representa la versión hexadecimal del ejecutable de la calculadora de Windows, podemos observar la presencia de unos bytes (recuadros rojos), llamados bytes o números mágicos (magic bytes), que son muy importantes a la hora de conocer el tipo de archivo. El número mágico es una secuencia específica de bytes al comienzo de un archivo que sirve para identificar su tipo de manera inequívoca. Estos bytes permiten a los diferentes sistemas operativos y a las aplicaciones reconocer y manejar diferentes tipos de archivos correctamente, incluso si su extensión es incorrecta o ha sido cambiada.

```
calc.exe
Address   0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f  Dump
00000000 4d 5a 90 00 03 00 00 00 04 00 00 00 ff ff 00 00 MZ..........ÿÿ..
00000010 b8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 ¸.......@.......
00000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000030 00 00 00 00 00 00 00 00 00 00 00 00 f0 00 00 00 ............ð...
00000040 0e 1f ba 0e 00 b4 09 cd 21 b8 01 4c cd 21 54 68 ..º..´.Í!¸.LÍ!Th
00000050 69 73 20 70 72 6f 67 72 61 6d 20 63 61 6e 6e 6f is program canno
00000060 74 20 62 65 20 72 75 6e 20 69 6e 20 44 4f 53 20 t be run in DOS 
00000070 6d 6f 64 65 2e 0d 0d 0a 24 00 00 00 00 00 00 00 mode....$.......
00000080 b4 30 dc 14 f0 51 b2 47 f0 51 b2 47 f0 51 b2 47 ´0Ü.ðQ²GðQ²GðQ²G
00000090 f9 29 21 47 f6 51 b2 47 bb 29 b1 46 f1 51 b2 47 ù)!GöQ²G»)±FñQ²G
```

Figura 4.1: Versión hexadecimal del código ejecutable (calc.exe)

Estos números mágicos son idénticos para todos los archivos del mismo tipo, es decir, todos los archivos ejecutables de Windows comienzan por MZ (cuya representación hexadecimal es 4d 5a), como se puede observar en la ilustración anterior.

En la siguiente tabla, aparecen unos pocos ejemplos de los números mágicos de diferentes tipos de archivos:

Tipo de archivo	Extensión típica	Magic Bytes (hexadecimal)	Magic Bytes (ascii)
Ejecutables MS Windows	.exe, .dll, .sys, ..	4d 5a	MZ
Unix elf		7f 45 4c 46	.ELF
PDF	.pdf	25 50 44 46	PDF
DOCX	.docx	50 4B 03 04	PK
PNG	.png	89 50 4E 47	PNG

Cuadro 4.1: Ejemplos de números mágicos de diferentes tipos de archivos (magic bytes)

En la fase de desarrollo de este TFG nos centraremos únicamente en los Ficheros PE, es decir, en los ficheros ejecutables de la plataforma Microsoft Windows, cuyo número mágico es MZ (4d5a en hexadecimal).

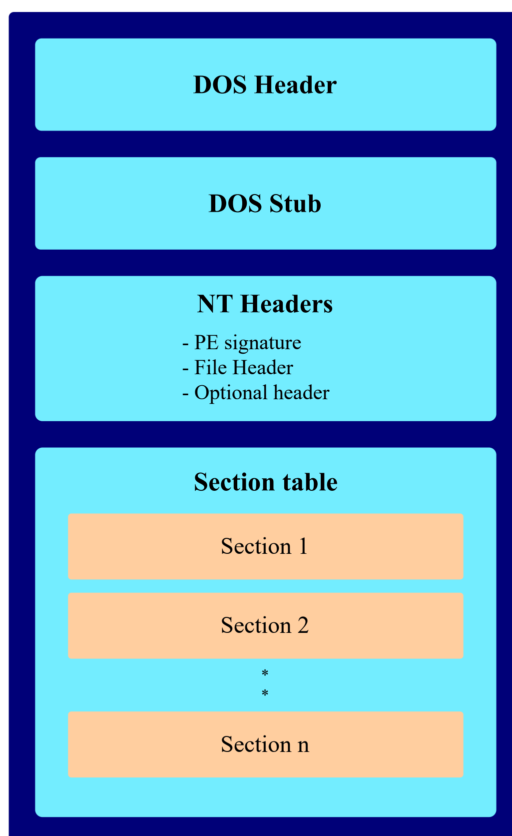
4.3.2. El formato de archivo PE (Portable Executable)

Un fichero PE (Portable Executable) es un formato de archivo utilizado en los sistemas operativos de Microsoft Windows para almacenar archivos ejecutables, archivos de código objeto, bibliotecas de enlace dinámico (DLL), y otros tipos de archivos que pueden ser ejecutados o cargados en memoria. Las diferentes extensiones que pueden tener los ficheros PE se han comentado anteriormente. Este formato de archivos contiene una estructura de datos que proporciona al cargador del sistema operativo Windows la información necesaria para gestionar el

código ejecutable que contiene. Esto abarca referencias a bibliotecas dinámicas para vincularlas, importaciones de tablas, datos de gestión de recursos, entre otros.

Descripción general del formato

Las estructura de datos PE está formada principalmente por el encabezado DOS, el código auxiliar de DOS (DOS Stub), los encabezados NT y la tabla de secciones.



El encabezado DOS ocupa los primeros 64 bytes del archivo y aquí es donde se encuentra el número mágico que representa a los ficheros ejecutables de Windows (MZ). A continuación, se encuentra el código auxiliar (DOS Stub), que simplemente es un pequeño programa que se ejecuta de forma predeterminada cuando se inicia el archivo y que, en caso de incompatibilidad (cuando el programa no es compatible con la plataforma Microsoft Windows), imprime el mensaje: "*Este programa no se puede ejecutar en modo DOS*".

Posteriormente, encontramos la sección de encabezados NT, que se compone de tres partes principales:

1. **Firma PE**: una firma de 4 bytes que identifica el archivo como un archivo PE.

2. **Encabezado de archivo**: este es un encabezado estándar COFF (Common Object File Format). Contiene información general sobre el archivo PE, como: tipo de máquina para la que está destinado el archivo, número de secciones, fecha y hora de creación del archivo PE, puntero al símbolo de tabla, número de símbolos de la tabla, tamaño de la tabla del encabezado opcional, etc.

3. **Encabezado opcional**: es el encabezado más importante de los encabezados NT, ya que proporciona información crucial al cargador del sistema operativo, como, por ejemplo, la dirección de memoria donde comienza el código y los datos. Recibe el nombre de opcional porque no todos los archivos lo tienen, como es el caso de los archivos objeto. Sin embargo, es obligatorio para los archivos de imagen (como los archivos .exe).

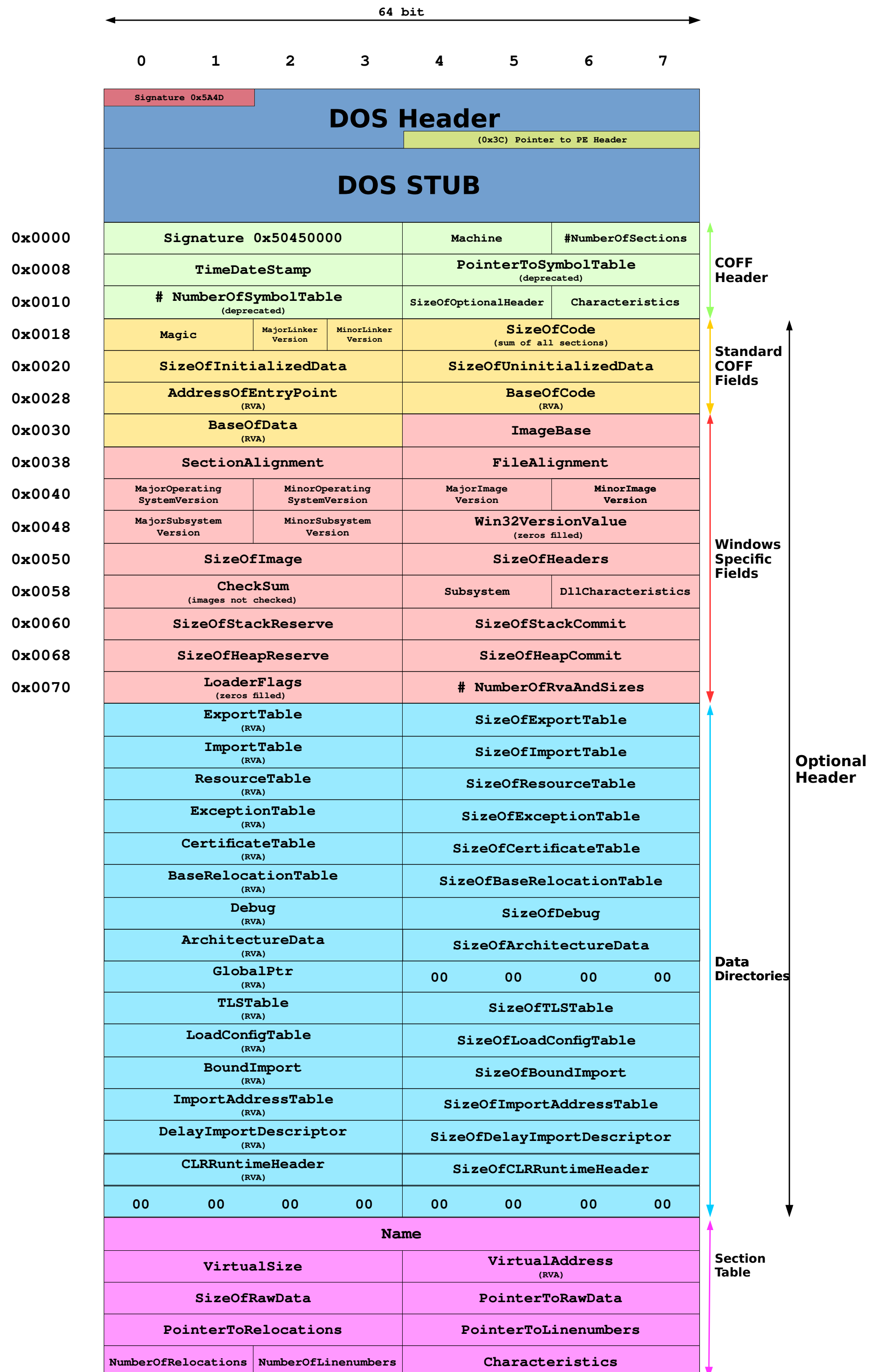
Finalmente, encontramos la tabla de secciones, que es una matriz de encabezados de secciones de imágenes. Contiene información detallada sobre cada sección del archivo PE, lo que permite al sistema operativo y a otros programas, entender cómo están organizados los datos y el código dentro del fichero PE.

Cada entrada en la tabla de secciones describe una sección específica del archivo PE, incluyendo su nombre, ubicación en el archivo, tamaño, características de protección y otros atributos importantes. Estos datos son esenciales para el cargador del sistema operativo al cargar y ejecutar el fichero, así como para otros programas que puedan tener relación con el archivo PE.

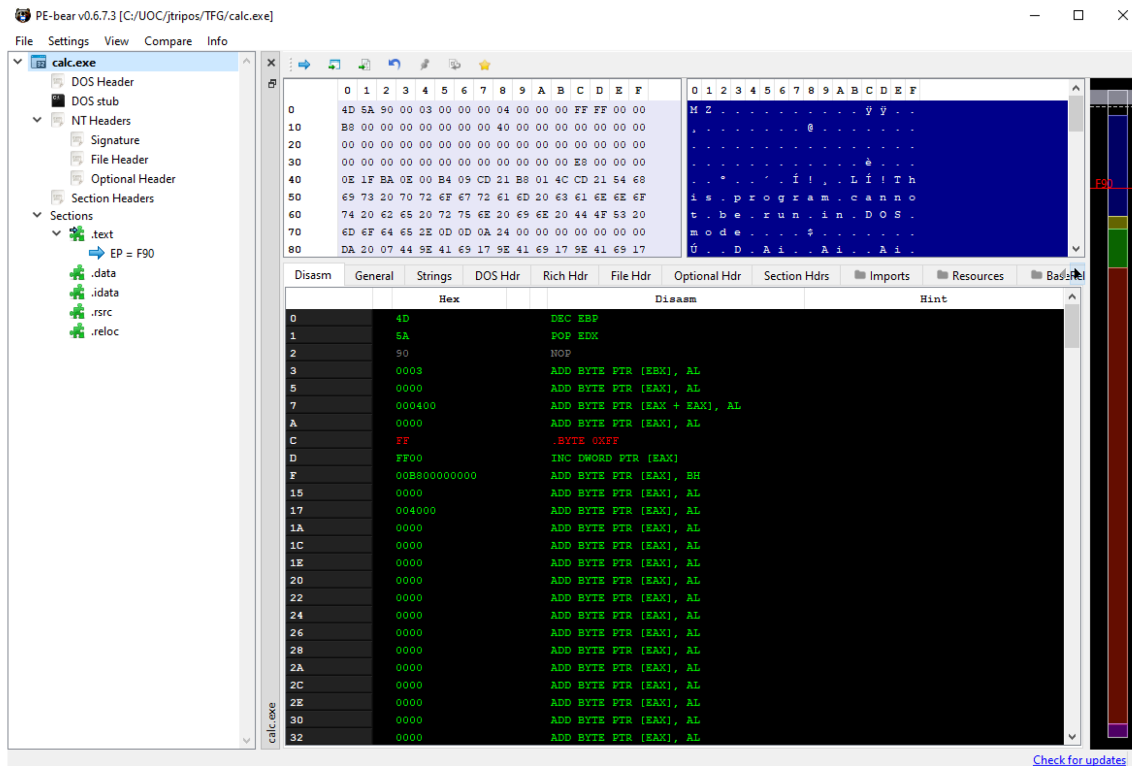
El malware puede ocultarse dentro del código ejecutable legítimo del archivo PE, modificando estas secciones o agregando secciones adicionales en esta tabla, con código malicioso. Esto puede incluir la modificación de la sección de código existente para insertar saltos a código malicioso, la adición de secciones de datos no utilizadas para almacenar el código malicioso, entre otros métodos.

En la siguiente ilustración, creada por Wikipedia [23], se puede observar de una manera más detallada cada uno de los campos de la estructura de fichero portable ejecutable (PE file structure). Esto es importante porque en la **fase de desarrollo** trabajaremos con cada uno de estos campos, extrayendo su contenido mediante un script escrito en *Python*, para posteriormente crear un dataset con estos valores y poder entrenar a diferentes modelos de aprendizaje para que sean capaces de detectar malware embebido en estos ficheros en tiempo real.

Estructura del formato PE



Siguiendo con el ejemplo del ejecutable de la calculadora de Windows (*calc.exe*), al abrir el ejecutable con el programa PE-bear, podemos observar el contenido de cada una de las secciones descritas anteriormente.



4.4. Métodos de detección de malware

Los métodos de detección de malware se pueden dividir en tres categorías principales: métodos basados en firmas, métodos basados en el comportamiento y métodos heurísticos. En esta sección, exploraremos brevemente estos enfoques.

4.4.1. Métodos basados en firmas

Estos métodos buscan detectar malware mediante la comparación de los archivos con una base de datos de muestras de malware conocidas. Cada archivo se compara con esta base de datos con el objetivo de identificar si son maliciosos. Para ello, primero se calcula la función hash del archivo PE objeto de análisis y luego, se compara con la base de datos de hashes de malware conocidos.

En el ámbito de la seguridad informática, el hash de un archivo actúa como una especie de huella digital única. Este enfoque es altamente efectivo para identificar malware ya conocido debido a la singularidad de los hashes. Sin embargo, su simplicidad también lo hace vulnerable

porque cualquier cambio en el archivo PE, como la adición de una sección con código malicioso, resultará en un hash completamente diferente, y esto hará que el malware pase desapercibido por este método de detección.

4.4.2. Métodos basados en el comportamiento

Estos métodos buscan identificar malware observando las acciones sospechosas o maliciosas que una aplicación realiza. Mientras que los métodos basados en firmas se enfocan en comparaciones con una base de datos conocida, los métodos de comportamiento se centran en lo que realmente hace la aplicación una vez que se ejecuta. Estos métodos analizan una variedad de acciones, como la cantidad de solicitudes de red que realiza la aplicación, las conexiones a URLs sospechosas, el acceso al almacenamiento de archivos, entre otras; con el objetivo de determinar si la aplicación puede ser considerada como maliciosa.

Extrayendo estos patrones, se pueden crear reglas simples para detectar comportamientos maliciosos. Además, se pueden analizar diferentes casos anteriores de malware conocido para identificar estrategias que puedan ser efectivas. Los métodos basados en comportamiento son más resistentes a los intentos de evasión respecto a los métodos anteriores, ya que, un atacante tendría que modificar el comportamiento de una aplicación para eludir su detección.

4.4.3. Métodos heurísticos: Aprendizaje automático

Los métodos heurísticos representan una de las herramientas más potentes en la detección de malware hasta la fecha. En lugar de enfocarse en patrones de comportamiento específicos, como los métodos basados en el comportamiento, estos métodos emplean técnicas de minería de datos y modelos de aprendizaje automático para comprender cómo son los archivos maliciosos y reconocer las características distintivas de este tipo de ficheros.

Estos métodos analizan una amplia gama de características de ejecución de los archivos, como las llamadas a diferentes APIs, las secuencias de instrucciones de código (OpCode), los gráficos de llamadas (muestran cómo se comunican diferentes componentes de un software entre sí y con recursos externos), entre otros muchos atributos, para finalmente entrenar modelos de clasificación.

Los métodos heurísticos son incluso más robustos que los enfoques basados en comportamiento, ya que, si un atacante realiza pequeños cambios en los parámetros de un malware conocido, no necesariamente conseguirá engañar al modelo. Esto los convierte en modelos altamente efectivos ya que pueden detectar incluso variantes desconocidas de malware.

Este es el método que emplearemos en la **fase de desarrollo** de este Trabajo Final de Grado.

4.5. ¿Como puede mejorar el aprendizaje automático las técnicas actuales de detección de malware?

El aprendizaje automático puede ayudar con algunos de los desafíos presentados por el malware moderno, principalmente debido a las siguientes características que lo distinguen de las técnicas tradicionales (como por ejemplo, las basadas en la identificación de firmas estáticas):

1. **Capacidad para detectar malware desconocido**
   Mientras que los métodos tradicionales dependen de la identificación de patrones conocidos de malware, el aprendizaje automático puede analizar grandes cantidades de datos para identificar características comunes entre diferentes muestras de malware y así detectar nuevas variantes o amenazas desconocidas.

2. **Adaptación a la evolución del malware**
   El malware moderno es dinámico y puede cambiar rápidamente con el objetivo de evadir la detección. Sin embargo, el aprendizaje automático puede adaptarse y aprender de nuevas muestras de malware, ajustando continuamente sus modelos y algoritmos para mantenerse al día con las nuevas tácticas de los atacantes.

3. **Análisis de comportamiento**
   En lugar de depender únicamente de características estáticas, como las firmas de virus, el aprendizaje automático puede analizar el comportamiento del programa y de los sistemas en tiempo real. Esto permite identificar comportamientos anómalos que pueden indicar la presencia de malware, incluso si el código en sí no coincide con ninguna firma conocida.

Capítulo 5

Ataques informáticos en la red

En el mundo interconectado y digitalizado en el que vivimos, los ciberataques representan una amenaza cada vez más frecuente y sofisticada para personas, empresas, organizaciones e incluso para naciones enteras. Estos ataques pueden tener consecuencias devastadoras, desde la pérdida de datos confidenciales hasta la interrupción de servicios críticos (aquellos que son esenciales para el funcionamiento y la estabilidad de un país), vulnerando con ello la seguridad nacional.

Uno de los incidentes más significativos ocurrió en 2017, cuando el ransomware WannaCry infectó cientos de miles de computadoras en más de 150 países en un corto período de tiempo. Este ataque paralizó diferentes sistemas de salud, bancos, empresas y hasta servicios gubernamentales, causando enormes pérdidas económicas y mostrando lo vulnerable que son las infraestructuras digitales de diferentes empresas y naciones ante ciberataques.

En este contexto, la detección temprana de las amenazas de red se vuelve de vital importancia para prevenir o mitigar sus efectos nocivos. La incorporación de técnicas avanzadas de aprendizaje automático supervisado en los sistemas de detección actuales puede proporcionar una herramienta poderosa para identificar y neutralizar los ciberataques de manera proactiva.

Este Trabajo Final de Grado pretende realizar un acercamiento a las grandes ventajas que puede ofrecer la incorporación de técnicas de aprendizaje automático supervisado a la detección temprana de ciberataques, siendo el ransomware WannaCry otro claro ejemplo de lo que este TFG busca ayudar a mitigar.

5.1. Clasificación de los ataques en la red

Las amenazas de red abarcan desde las más evidentes, como un correo electrónico fraudulento que promete riquezas a cambio de información bancaria, hasta las más sofisticadas y encubiertas, como un fragmento de código malicioso que se infiltra en una organización y permanece latente en su red durante un largo período de tiempo antes de eliminar su rastro,

logrando con ello una enorme y costosa fuga de información. El conocimiento de los distintos tipos de ciberataques que existen en la actualidad es fundamental para prevenir, prepararse y responder de manera efectiva ante dichas amenazas.

En la red existen numerosas clasificaciones de ciberataques y cada una de ellas es muy diferente a la anterior. Sin embargo, en las siguientes líneas se expondrá una clasificación propia de dichas amenazas, en base a todas las clasificaciones estudiadas.

1. **Ataques de denegación de servicio** (DoS y DDoS). Los ataques de denegación de servicio (DoS) y de denegación de servicio distribuido (DDoS) son tipos de ciberataques que tienen como objetivo hacer que un servicio, red o sitio web sea inaccesible para sus usuarios legítimos. Esto se consigue inundando al servidor, red o sitio web con una cantidad inmensa de solicitudes o tráfico.

2. **Ataques Man In The Middle** (MITM). Son un tipo de ciberataque en el que el atacante se sitúa en medio de la comunicación y puede espiar, interceptar o modificar los mensajes enviados entre las dos partes sin que estas se den cuenta.

3. **Ataques de contraseña**. Los ataques de contraseña son intentos malintencionados de obtener, adivinar o forzar la contraseña de un usuario para acceder a un sistema o cuenta. Los tipos más comunes de ataques de contraseña son:

    a) Ataque de fuerza bruta

    b) Ataque de diccionario

    c) Ataque de credenciales filtradas

4. **Ataques a aplicaciones web**. Son intentos maliciosos de explotar diferentes vulnerabilidades en las aplicaciones web para robar datos, alterar el funcionamiento de la aplicación o tomar el control de ella. Los tipos más comunes son:

    a) Inyección SQL (SQL Injection)

    b) Secuencias de comandos entre sitios (Cross-Site Scripting - XSS)

    c) Falsificación de solicitudes entre sitios (Cross-Site Request Forgery - CSRF)

    d) Inyección de comandos (Command Injection)

5. **Ataques de suplantación de identidad** (Spoofing). Son técnicas que los atacantes utilizan para engañar a sistemas, redes o usuarios haciéndose pasar por una entidad confiable mediante la falsificación de datos.

    a) DNS Spoofing

    b) IP Spoofing

c) ARP Spoofing
d) Email Spoofing
e) GPS Spoofing

6. **Ataques de escalada de privilegios** (Privilege Escalation). En este tipo de ataques, un atacante intenta obtener niveles más altos de acceso o permisos dentro de un sistema de los que originalmente tenía. Esto puede permitirle realizar acciones que no debería poder hacer, como modificar o acceder a datos sensibles, instalar software malicioso, o alterar configuraciones críticas del sistema. La escalada de privilegios se divide en dos categorías principales: escalada de privilegios vertical y horizontal.

7. **Ataques de sondeo** (Probing). Son actividades de exploración realizadas por un atacante para obtener información sobre una red, sus dispositivos y servicios. Estos sondeos pueden tener diversos propósitos, como identificar vulnerabilidades, mapear la infraestructura de red, o recopilar información para futuros ataques. Los sondeos más comunes son:

   a) Escaneo de Puertos (Port Scanning)
   b) Escaneo de Subredes (Subnet Scanning)
   c) Escaneo de Host (Host Scanning)
   d) Vigilancia de Red (Network Surveillance)

8. **Ataques de Cryptojacking**. Son un tipo de ciberataque en el que los atacantes usan el poder de procesamiento de computadoras, dispositivos móviles o servidores sin el consentimiento del propietario para minar criptomonedas.

9. **Ataques de secuestro de sesiones (Session Hijacking)**. Es un tipo de ciberataque en el que un atacante roba o usa de manera fraudulenta una sesión válida de usuario para obtener acceso no autorizado a información y servicios en un sistema.

En la **fase de desarrollo** nos sumergiremos por el conjunto de datos NSL-KDD, que contiene los ataques descritos en los puntos 1, 3, 6 y 7; es decir, los ciberataques existentes en el dataset se dividen en cuatro categorías principales:

1. Ataques de denegación de servicio. Estos ataques tendrán la etiqueta **DOS**.

2. Ataques de contraseña. Estos ataques tendrán la etiqueta **R2L** (Remote to Local) y representarán el acceso no autorizado desde una máquina remota.

3. Ataques de escalada de privilegios. Estos ataques tendrán la etiqueta **U2R** (User to Root) y representarán el acceso no autorizado a privilegios de superusuario local (root).

4. Ataques de sondeo. Estos ataques tendrán la etiqueta **Probe** y representarán las actividades de vigilancia y otros sondeos que realice un atacante.

5.2. Fases de un ataque informático

Para estudiar los ciberataques es esencial entender las acciones y los métodos de intrusión que éstos llevan a cabo. Ya hemos visto que los distintos tipos de ataques se llevan a cabo de diferentes maneras y que pueden tener objetivos muy diferentes. A pesar de estas diferencias, se puede identificar un patrón común en el flujo ofensivo que siguen los diferentes ataques, y que se puede observar en la siguiente figura:

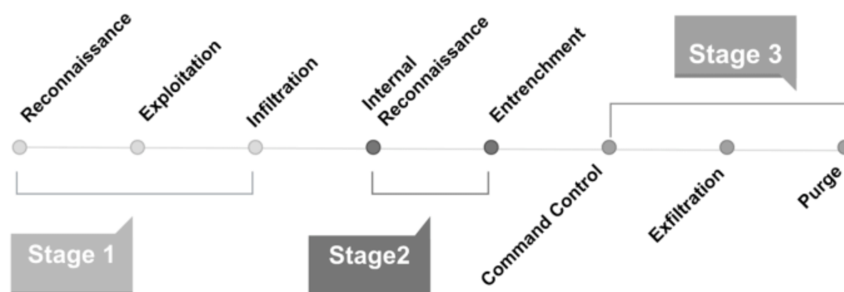


Figura 5.1: Fases de un ciberataque

Durante la **etapa 1**, se realizan acciones de reconocimiento activo, como el escaneo de puertos, con el fin de obtener información detallada y actualizada sobre el objetivo. Esto incluye la identificación de posibles vulnerabilidades que podrían ser aprovechadas para infiltrarse en el sistema objetivo.

En la **etapa 2**, una vez que el malware se encuentra en el sistema de la víctima, se embarca en un proceso de reconocimiento interno y movimiento lateral a través de la red. Una vez identificados los sistemas de alto valor, el malware se establece de forma persistente en el entorno mediante la instalación de puertas traseras para un acceso futuro o la configuración como un proceso demonio en segundo plano.

Finalmente, en la **etapa 3**, una vez que el atacante ha asegurado el control del sistema objetivo, cuando finaliza la tarea y culmina su propósito, el malware suele optar por purgarse y eliminar todas las huellas de su presencia en el entorno para evitar ser detectado.

Un ataque informático sigue una ruta bien definida desde el reconocimiento del sistema objetivo hasta la toma de control. Interrumpir su progreso en cualquier punto de esta ruta podría revelar información valiosa sobre sus intenciones y capacidades.

5.3. Métodos de detección de intrusiones en la red

Entre los métodos más comunes de detección de ciberataques se encuentran:

1. **Sistemas de Detección de Intrusiones** (IDS). Son herramientas de seguridad diseñadas para monitorear y analizar el tráfico de red o la actividad en sistemas informáticos, con el objetivo de encontrar comportamientos anómalos. Los IDS se clasifican según su ubicación en el sistema y el tipo de actividad que supervisan, siendo los más conocidos:

    a) **Sistemas de detección de intrusiones en la red** (NIDS). Monitorean el tráfico entrante y saliente a los dispositivos a través de la red. Estos sistemas se instalan en ubicaciones estratégicas de la infraestructura de red, a menudo justo detrás de los firewalls, en el perímetro de la red. Su propósito principal es identificar y alertar sobre cualquier actividad sospechosa o maliciosa que pueda intentar infiltrarse en la red.

    b) **Sistemas de detección de intrusiones basado en host** (HIDS). Este tipo de sistemas se instala en un punto terminal específico, como un portátil, un router o un servidor. Estos sistemas monitorean exclusivamente la actividad en el dispositivo donde están instalados, supervisando el tráfico de entrada y salida. Un HIDS típicamente opera capturando instantáneas periódicas de los archivos críticos del sistema operativo y comparándolas a lo largo del tiempo. Si el HIDS detecta algún cambio, como la modificación de archivos de registro o la alteración de configuraciones, genera una alerta para el equipo de seguridad.

2. **Sistema de Prevención de Intrusiones** (IPS). Estos sistemas, al igual que los IDS, monitorean el tráfico de red en busca de amenazas potenciales pero, a diferencia de estos, los IPS toman medidas para bloquearlas de manera automática. Estas acciones incluyen alertar al equipo de seguridad, finalizar conexiones peligrosas, eliminar contenido malicioso o activar otros dispositivos de seguridad.

3. **Honeypots y Honeynets**. Son técnicas de seguridad utilizadas para detectar, desviar y estudiar ciberataques.

    a) Un **honeypot** es un mecanismo de ciberseguridad que replica un objetivo de ataque, creado con el propósito de desviar a los atacantes de los verdaderos objetivos y recolectar datos sobre sus identidades y técnicas, con el objetivo de mejorar la seguridad de la organización.

    b) Una **honeynet** es una red completa de honeypots interconectados y desplegados en una red simulada, que se asemeja a una red real. Mientras que un honeypot individual puede ser utilizado para atraer y estudiar a un atacante específico, una honeynet es capaz de recopilar información sobre ataques más amplios y sofisticados.

4. **Sistemas de detección/prevención de instrusiones empleando aprendizaje automático** (IDS/IPS Machine Learning Systems). Son sistemas formados por un conjunto de técnicas avanzadas de detección y prevención de ciberataques, que utilizan diferentes algoritmos y modelos de aprendizaje automático para analizar grandes cantidades de datos y detectar patrones o comportamientos anómalos que podrían indicar la presencia de un ciberataque. Lo más importante de este tipo de sistemas es que son capaces de adaptarse y aprender de manera continua a medida que enfrentan nuevas amenazas.

En la **fase de desarrollo** de este TFG, emplearemos el cuarto método para la detección y prevención de ciberataques en la red, donde crearemos diferentes modelos de aprendizaje automático supervisado que nos ayuden a fortificar cualquier tipo de red.

Capítulo 6

Caso de uso 1

Detección de malware empleando técnicas de Aprendizaje Automático

En un mundo cada vez más pegado a las pantallas, la amenaza del malware representa un desafío constante para las personas, empresas y organizaciones. El malware, que se encuentra presente de diferentes formas (virus, gusanos, troyanos, ransomware, etc.), persigue la capacidad de ocultación en el sistema informático objetivo, con el fin de dañar este sistema, robar información de él, obtener acceso no autorizado a la red, extorsionar a las víctimas, entre otros.

Ante esta creciente amenaza, las técnicas tradicionales de detección de malware, como las basadas en firmas y patrones conocidos, han demostrado ser cada vez menos efectivas frente a la evolución constante de las técnicas de creación de malware y a su mayor sofisticación. En este contexto, el aprendizaje automático emerge como una poderosa herramienta para combatir el malware, permitiendo detectar diferentes códigos maliciosos de una manera más temprana y eficiente.

En este primer caso de uso, exploraremos cómo las diferentes técnicas de Aprendizaje Automático pueden aplicarse a la detección de malware en un determinado sistema o red. A lo largo de estas páginas, veremos cómo el Aprendizaje Automático permite analizar grandes volúmenes de datos, identificar patrones y características distintivas del malware etiquetado, y entrenar modelos predictivos capaces de detectar nuevas amenazas con alta precisión y velocidad. Para ello, emplearemos diferentes algoritmos de Aprendizaje Automático, los cuales entrenaremos con el dataset creado expresamente para este caso de uso, y tras este entrenamiento, generaremos diferentes modelos que luego decidiremos cuál es el que mejor realiza nuevas predicciones.

Así como el malware se encuentra presente de diferentes formas también se encuentra presente en diferentes sistemas operativos. Es muy importante que un analista de malware conozca el sistema operativo y la estructura de los ficheros donde se encuentra el código malicioso, ya que, dependiendo de la plataforma o del tipo de archivo, esta estructura variará. Para desarrollar este caso de uso, se empleará el formato de archivos PE (Portable Executable); un formato

de archivo para ficheros ejecutables usados en versiones de 32 y 64 bits del sistema operativo Microsoft Windows. Dentro de este formato de archivo se encuentran ficheros con extensión *.exe*, *.dll*, *.sys* o *.efi*.

6.1. Justificación de la elección del formato PE

El formato Portable Executable (PE) proporciona un entorno propicio para ocultar malware gracias a su complejidad y flexibilidad. Una de las formas más comunes en las que se oculta el malware dentro de archivos PE es a través de técnicas de ofuscación y camuflaje. Estas técnicas pueden incluir la modificación de secciones del archivo, la inserción de código malicioso en secciones aparentemente benignas, el uso de técnicas anti-análisis para evitar la detección por parte de software antivirus, entre otros métodos.

Además, el malware alojado en este formato de archivos, puede utilizar funciones del sistema operativo de manera legítima para evitar levantar sospechas, aprovechando la amplia variedad de funciones disponibles en el entorno Microsoft Windows. Estas estrategias hacen que resulte muy difícil, para los programas antivirus y para los analistas de malware, identificar y eliminar con eficacia el malware oculto en archivos PE.

Por esta razón, este TFG busca colaborar en la creación de un modelo de detección temprana de malware, sin supervisión humana, donde el propio modelo sea capaz de aprender y de adaptarse a un código malicioso en continuo cambio, con el objetivo de poder detectarlo a tiempo y evitar sus consecuencias.

Por otro lado, se ha elegido el formato de archivos PE debido a que es el formato de archivo ejecutable utilizado por el sistema operativo Microsoft Windows, quien a día de hoy, sigue siendo uno de los sistemas operativos más utilizados en el mundo, tanto en entornos domésticos como en entornos empresariales. Todo esto hace que sea vital para los analistas de malware comprender cómo funciona este formato de archivos.

6.2. Creación del dataset

Como se ha comentado anteriormente, en este caso de uso se trabajará sobre un dataset generado a partir de diferentes archivos PE (Portable Executable) propios de la plataforma Microsoft Windows. Este dataset consta de dos subconjuntos: el subconjunto de archivos legítimos (no infectados) y el subconjunto de malware.

6.2.1. Creación del subconjunto de datos de archivos legítimos

a. Se crea un directorio que albergará 3.457 muestras de diferentes archivos legítimos (no maliciosos) procedentes de nuestro propio sistema operativo Windows 11. Para ello, se ha accedido a la carpeta C:\Windows de nuestro Windows 11 y se han extraído cantidades aleatorias de archivos con extensión .exe, .dll y .sys. Estos ficheros se pueden encontrar en un archivo comprimido llamado legitimate.7z en la siguiente URL:

   `https://drive.google.com/drive/folders/1KgSOpMyK11GkDyfCVsMenV3bhV1rEXYJ?usp=drive_link`

b. Se diseña un script en Python llamado *datasetsCreationFromPEFiles.py*, que se puede encontrar dentro del repositorio de GitHub, en el que se emplea el módulo Pefile de Python para extraer diferentes características de la estructura de los archivos PE obtenidos en el apartado anterior, para posteriormente almacenarlas de forma estructurada en un archivo con etensión .csv que será el dataset de archivos legítimos (legitimateDataset.csv). Este fichero .csv tiene una cabecera donde figura el nombre del fichero PE objeto de iteración y la descripción de las características de su estructura que se han escogido previamente (por considerarse representativas para la detección de malware). A continuación, se encontrará en cada fila, cada uno de los archvivos PE con sus respectivas características.

6.2.2. Creación del subconjunto de datos de malware

a. Se crea un directorio que albergará 13.971 muestras de diferentes archivos de código malicioso procedentes de los repositorios de VirusTotal y VirusShare. Para ello, se ha solicitado el acceso académico a cada una de estas plataformas de análisis de malware online.

b. Este apartado es el mismo que para el dataset de archivos legítimos, ya que, el script datasetsCreationFromPEFiles.py crea los dos subconjutos de datos al ejecutarse. La única diferencia es que este nuevo dataset, llamado malwareDataset.csv, cogerá los datos del directorio de malware y no del directorio de archivos legítimos. Como el proceso de extracción de características PE es similar, se ha optado por unificar todo el código en un script. Todos estos ficheros de malware se han descargado y agrupado en un único directorio, que se encuentra en un archivo comprimido llamado malware.7z en la siguiente URL:

   `https://drive.google.com/drive/folders/1KgSOpMyK11GkDyfCVsMenV3bhV1rEXYJ?usp=drive_link`

   Como se trata de código potencialmente peligroso, el fichero se encuentra encriptado. Para desencriptarlo, la contraseña es: infected.

6.3. Creación de diferentes modelos de aprendizaje

En este apartado nos sumergiremos en el desarrollo de diferentes modelos de aprendizaje, lo que involucra una serie de pasos fundamentales, como la comprensión del problema que se está abordando o la recopilación y preparación de datos relevantes. Una vez que dispongamos de los datos adecuados, procederemos a la selección de características y al entrenamiento de diferentes algoritmos de aprendizaje.

6.3.1. Comprensión de los datos

Tanto el dataset de archivos legítimos como el de malware han sido el resultado de la extracción de 53 características de las 17428 muestras de archivos PE examinados. Estas características han sido extraídas tanto de las diferentes cabeceras como de las diferentes secciones existentes en los ficheros PE.

6.3.2. Preprocesado de los datos

Cuando hablamos de preparación de los datos, una de las primeras cosas que debemos tener en cuenta es la desviación estándar, ya que esta medida nos indicará cuánto se desvían los valores individuales de la media del conjunto de datos. Esto es importante porque si tenemos desviación estándar cero en un conjunto de datos significa que todos los valores en ese conjunto son idénticos.

Llevado esto a nuestro caso de uso, si una determinada característica del dataset tiene desviación estándar 0, significa que ese campo no será muy relevante a la hora de realizar una correcta clasificación de nuestros archivos.

Por tanto, lo primero que crearemos es una función que nos devuelva aquellas características del dataset cuya desviación estándar es 0, con la intención de posteriormente eliminarlas del dataset. Tras la ejecución de la función, observamos que existen un total de 15 características que poseen una desviación estándar de cero. Las borramos.

El siguiente paso en este preprocesamiento de nuestro dataset es la estandarización. El método de estandarización más habitual consiste en convertir cada valor en su puntuación típica, es decir, restarle el valor medio y dividirlo por su desviación estándar. Para realizar este tipo de estandarización, emplearemos la clase sklearn.preprocessing.StandardScaler de la librería scikit-learn.

6.3.3. División del dataset en un conjunto de entrenamiento y en un conjunto de test

Los algoritmos de Machine Learning aprenden de los datos con los que los entrenamos. A partir de ellos, intentan encontrar o inferir el patrón que les permita predecir el resultado para un nuevo caso. Pero, para poder calibrar si un modelo funciona, necesitaremos probarlo con un conjunto de datos diferente. Por ello, en todo proceso de aprendizaje automático, los datos de trabajo se dividen en dos partes: datos de entrenamiento y datos de prueba o test.

Los datos de entrenamiento o training data son los datos que usamos para entrenar un modelo y los datos de prueba, validación o testing data son los datos que nos reservamos para comprobar si el modelo que hemos generado a partir de los datos de entrenamiento funciona.

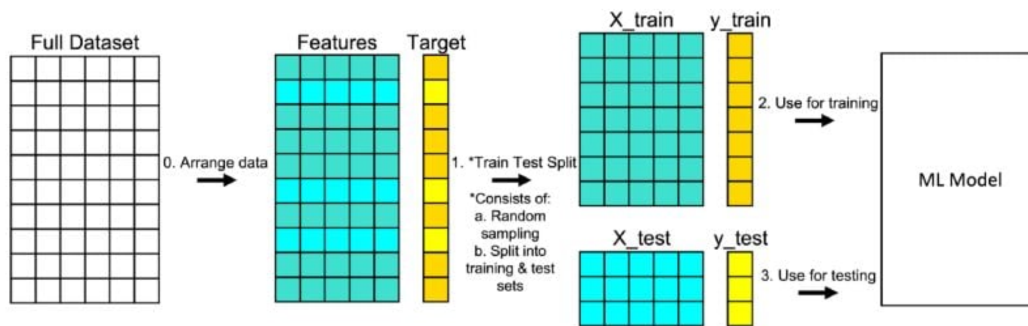


Figura 6.1: Ejemplo de división de un dataset. Fuente: [24]

Normalmente el conjunto de datos se suele repartir en un 80 % de datos de entrenamiento y un 20 % de datos de test, pero se puede variar la proporción según el caso. Lo importante es ser conscientes de que hay que evitar el sobreajuste u .ºverfittingz el subajuste o ünderfitting”, pero eso son aspectos de los que hablaremos más adelante.

6.3.4. Balanceo del conjunto de datos

Los conjuntos de datos desequilibrados son aquellos en los que una clase tiene significativamente menos instancias que las otras, como es el caso de nuestras muestras de malware vs las legítimas. Como se puede observar en los siguientes gráficos, la cantidad de muestras de malware es muy superior a las muestras de archivos legítimos.

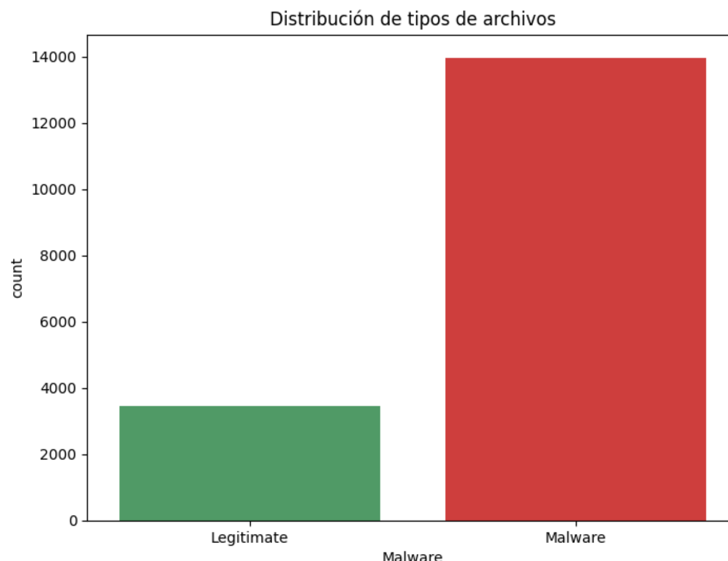


Figura 6.2: Distribución de archivos PE

Los algoritmos de clasificación pueden funcionar mal cuando se entrenan en conjuntos de datos desequilibrados. El sobremuestreo (oversampling) y el submuestreo (undersampling) pueden resultar útiles a la hora de superar el desequilibrio de clases y, por tanto, mejorar el rendimiento del modelo.

El sobremuestreo aumenta la cantidad de datos que pertenecen a la clase minoritaria al duplicar los existentes o generar otros nuevos mediante la creación de datos sintéticos, mientras que el submuestreo reduce el número de muestras de la clase mayoritaria para adaptar la muestra a la cantidad de datos minoritaria.

Al decidir entre estos dos enfoques para equilibrar un conjunto de datos desequilibrado, se deben considerar sus ventajas y limitaciones. En este caso de uso se practicará con ambos enfoques de cara a encontrar el modelo que realice la predicción más exacta.

6.3.5. Validación cruzada

La validación cruzada o cross-validation es una técnica para evaluar el rendimiento de un modelo de aprendizaje automático y estimar su capacidad de generalización con nuevos datos desconocidos. En lugar de dividir el conjunto de datos en únicamente dos subconjuntos (entrenamiento y test), como hemos visto en un apartado anterior, la validación cruzada divide el conjunto de datos en múltiples subconjuntos llamados *folds*.

Veamos cómo funciona...

En la validación cruzada de K iteraciones o *K-fold cross-validation*, el cojunto de datos se divide en K subconjuntos. Uno de los subconjuntos se utiliza como datos de test y el resto (K-1) como datos de entrenamiento. El proceso de validación cruzada es repetido durante k iteraciones, con cada uno de los posibles subconjuntos de datos de prueba. Finalmente, se realiza la media aritmética de los resultados de cada iteración para obtener un único resultado.

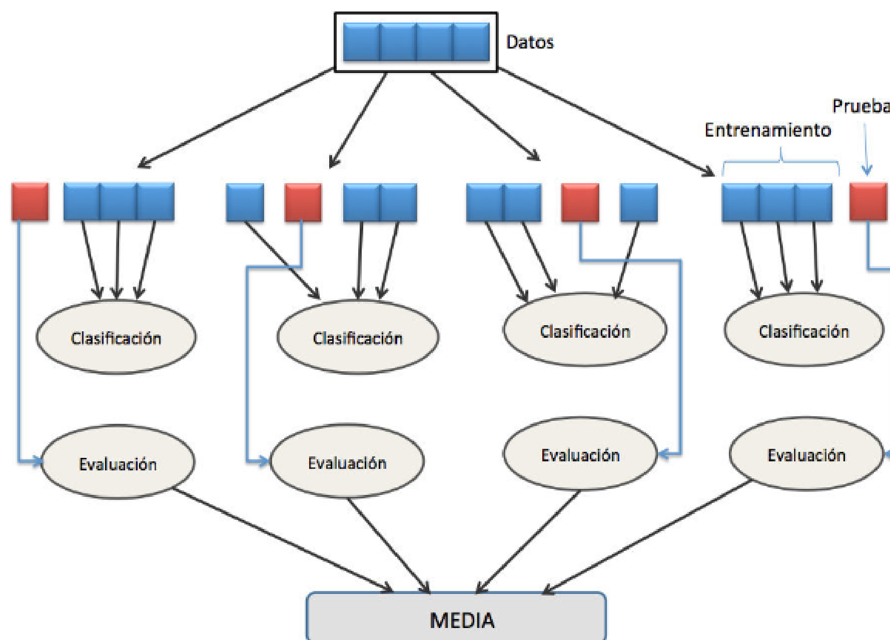


Figura 6.3: Esquema de validación cruzada de 4 iteraciones. Fuente: [25]

En este caso de uso, emplearemos la técnica de validación cruzada con 7 iteraciones. Para ello, emplearemos la función *cross_val_score* de la biblioteca scikit-learn, concretamente del módulo sklearn.model_selection.

6.3.6. Selección de características

Uno de los componentes fundamentales de cualquier análisis de datos es explorar las potenciales relaciones que puedan existir entre las variables que están siendo analizadas. Para esto, se utiliza una medida estadística llamada correlación, que nos indica tanto la magnitud como la dirección de la relación que podría haber entre dos variables.

Comenzaremos por explorar la relación existente entre cada variable independiente y la variable dependiente (nuestro objetivo, "Malware") y, a continuación, graficaremos y analizaremos la matriz de correlaciones.

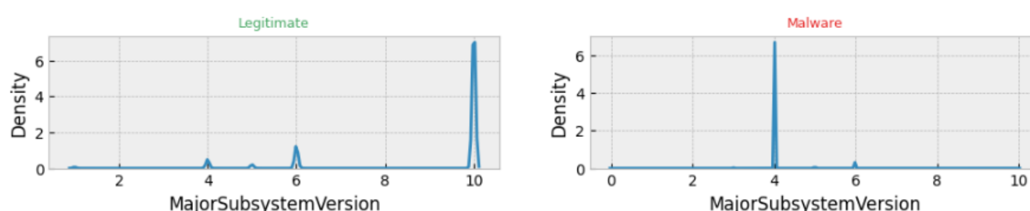


Figura 6.4: Relación entre la variable 'MajorSubsystemVersion' del dataset de malware vs archivos legítimos

Si procedemos a la lectura de la ilustración anterior, obtenemos que los valores de la característica 'MajorSubsystemVersion' oscilan en torno a 6 y 10 para los archivos legítimos mientras que para el malware, los valores se encuentran centrados mayormente en el valor 4. Utilizaremos todos estos patrones extraídos del análisis de las características junto con el mapa de calor de correlaciones entre características, con el objetivo de seleccionar las características más adecuadas para el posterior entreno de los modelos.

En un mapa de calor de correlación, cada variable está representada por una fila y una columna, y las celdas muestran la correlación entre ellas. El color de cada celda representa la fuerza y dirección de la correlación, y los colores mas oscuros indican correlaciones más fuertes (en nuestro caso, las correlaciones más fuertes se reflejan con colores azulados y rojizos intensos).

Los mapas de calor de correlación son importantes porque ayudan a identificar qué variables pueden dar lugar a multicolinealidad, lo que comprometería la integridad del modelo. La multicolinealidad ocurre cuando las variables independientes (posibles predictoras) están correlacionadas. Las variables independientes deberían ser eso, independientes. Esto se debe a que si el grado de correlación entre las variables independientes es alto, no podremos aislar la relación entre cada variable independiente y la variable dependiente (nuestro objetivo, "Malware").

Por tanto, por esta razón, en este caso de uso se ha considerado eliminar aquellas características que tengan un índice de correlación superior a +0.8 o inferior a -0.8.

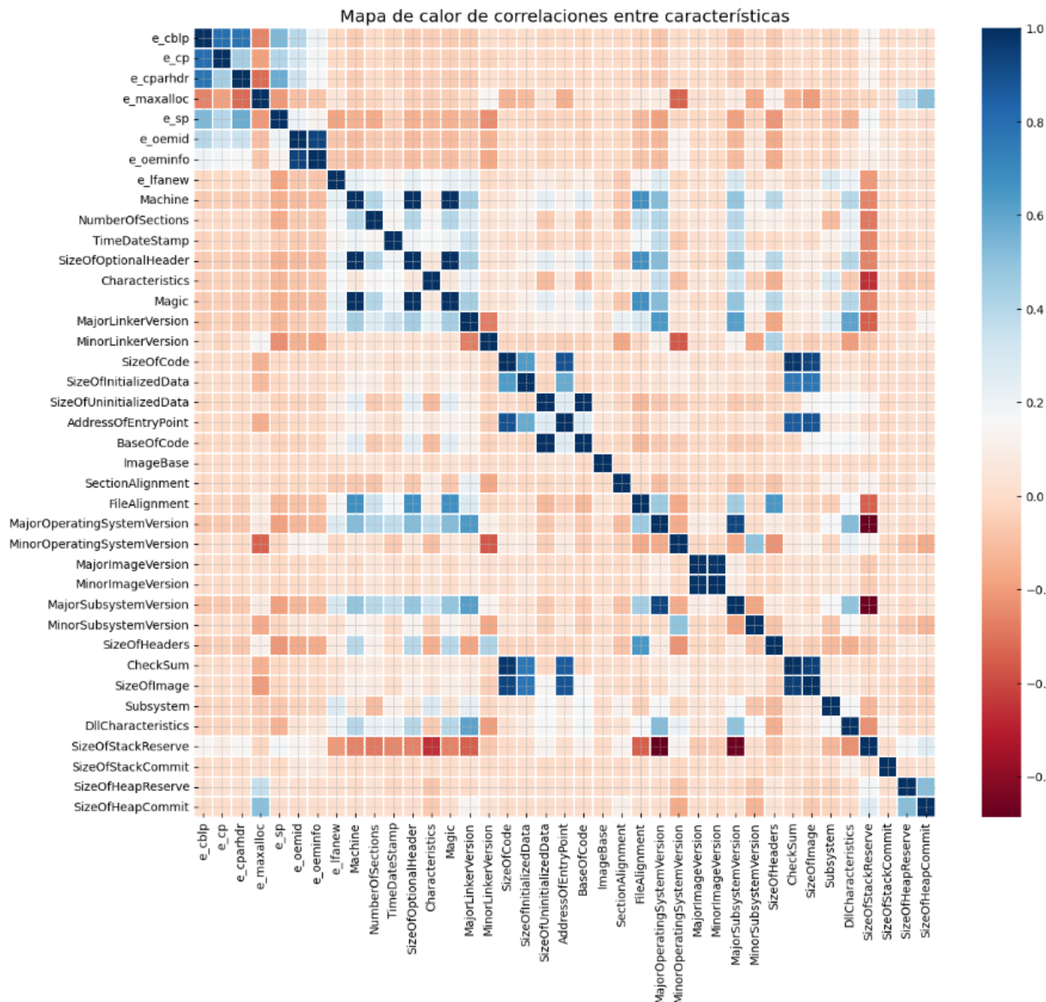


Figura 6.5: Mapa de calor de correlaciones entre características

Finalmente, tras todos estos pasos, ya estamos en condiciones de seleccionar únicamente aquellas características que consideramos que aportarán gran valor al entrenamiento de nuestros modelos. En el *jupyter notebook* del repositorio de GitHub aparece detallado el paso a paso práctico de toda esta teoría.

6.3.7. Desarrollo de diferentes modelos de aprendizaje automático

Con el objetivo de automatizar el entrenamiento de varios algoritmos de aprendizaje, se ha codificado un script en *Python*, donde hacemos lo siguiente:

1. Añadimos en un vector todos los algoritmos que deseamos aplicar para crear los diferentes modelos.

2. Recorremos cada uno de los algoritmos del vector y los entrenamos con nuestros datos para crear un modelo por cada algoritmo.

3. Realizamos validación cruzada del modelo entrenado y almacenamos tanto su media como su desviación estándar.

4. Imprimimos los resultados.

El código referente a la creación de estos modelos se encuentra tanto en el anexo A.9 de esta memoria como en el repositorio de GitHub.

Capítulo 7

Caso de uso 2

Detección de ciberataques empleando técnicas de Aprendizaje Automático

En este segundo caso de uso, exploraremos cómo las diferentes técnicas de Aprendizaje Automático pueden aplicarse a la detección de ciberataques en la red. A lo largo de estas páginas, veremos cómo el Aprendizaje Automático permite analizar grandes volúmenes de datos, identificar patrones y características distintivas de tráfico etiquetado, y entrenar modelos predictivos capaces de detectar nuevas amenazas de red con alta precisión y velocidad. Para ello, emplearemos diferentes algoritmos de Aprendizaje Automático, los cuales entrenaremos con el dataset elegido para este caso de uso, y tras este entrenamiento, generaremos diferentes modelos que luego decidiremos cuál es el que mejor realiza nuevas predicciones.

7.1. Búsqueda de un conjunto de datos adecuado

Los modelos de aprendizaje automático aprenden de los datos proporcionados y aplican este conocimiento en la toma de decisiones. En el contexto de este caso de uso, es decir, en la detección de anomalías de red, los datos con los que se entrenen los modelos deben, al menos, tener una cantidad adecuada de ejemplos de diferentes tipos de ataque, contar con un número de patrones representativo y poseer una dimensionalidad adecuada. Todo esto es debido a que el gran valor de un modelo no sólo depende del algoritmo empleado, sino sobre todo de la calidad de los datos introducidos.

Adicionalmente, para una correcta categorización de todos los paquetes que viajen a través de la red, es necesario que el conjunto de datos elegido posea un tráfico completo (funcional y no funcional), múltiples protocolos de red, conexiones internas y externas y, lo más importante, que tenga los datos etiquetados. Se requiere que el dataset tenga los datos etiquetados con los diferentes nombres de ataques a los que pertenece cada registro ya que, al tratarse de modelos

de aprendizaje automático supervisado, es necesario que primero el sistema aprenda de una correcta clasificación para que posteriormente pueda extraer patrones y realizar sus propias predicciones.

La creación de un conjunto de datos propio, como sí hemos realizado en el primer caso de uso, excede el alcance de este proyecto, debido a que es bastante complejo generar tráfico de red donde se produzcan diferentes tipos de ataques. Es por ello, por lo que se optó por utilizar un dataset público.

7.1.1. Comparativa de diferentes conjuntos de datos existentes

Para evaluar los diferentes conjuntos de datos públicos utilizados en la detección de ciberataques, hemos considerado las características necesarias mencionadas en el apartado anterior.

Uno de los datasets más conocidos en el mundo de la detección de intrusiones es el **KDD Cup 1999**, que ofrece una amplia variedad de ataques, aunque algunos están desactualizados. Este dataset contiene una gran cantidad de registros de tráfico de red simulado, generados mediante una variedad de herramientas y técnicas de simulación. Incluye tanto datos de tráfico normal como datos de varios tipos de ataques en un entorno de red simulado. Cada registro en el dataset tiene atributos que describen características de la conexión de red, como la duración de la conexión, los protocolos utilizados, la cantidad de bytes transferidos, entre otros. Además, cada registro está etiquetado con la categoría de ataque correspondiente o como tráfico normal.

Otro dataset relevante es el **NSL-KDD**, que mejora algunos aspectos del KDD Cup 99, al ofrecer una mayor diversidad de ataques, con una distribución más equilibrada entre las clases de tráfico normal y las de ciberataques. Al igual que el KDD Cup 99, se centra en protocolos TCP/IP y ofrece una cobertura de tráfico basada en simulaciones. Otra característica a destacar de este dataset es que, a diferencia de su predecesor, incorpora datos de múltiples fuentes y escenarios. Los datos siguen estando completamente etiquetados para facilitar la clasificación.

El conjunto de datos **UGR'16**, desarrollado por el Grupo de Investigación en Telemática de la Universidad de Granada (UGR), es otro recurso muy valioso en el campo de la detección de intrusiones en redes. Este conjunto de datos contiene registros de tráfico de red capturados en un entorno de laboratorio controlado durante un período de tiempo específico. Tal y como se menciona en la web de su universidad, estos datos proceden de varios recolectores netflow v9 ubicados estratégicamente en la red de un ISP español.

Como cuarta alternativa, presentamos el conjunto de datos **CICIDS2017**, otro recurso de gran valor para la detección de intrusiones en redes informáticas. Creado por el Centro de Investigación en Seguridad Cibernética de Canadá (CIC), este dataset contiene registros de

tráfico de red normal y de ciberataques. Se compone de nueve tipos de tráfico diferentes, incluyendo tráfico HTTP, FTP, SSH, y otros protocolos comunes, así como una variedad de ataques simulados y de tráfico malicioso. Posee ataques modernos y variados, lo que lo convierte en una herramienta valiosa para evaluar la capacidad de los sistemas de detección de intrusiones para identificar amenazas actuales. Además, este dataset también está etiquetado con información detallada sobre cada conexión de red.

Existen otros dataset incluso más modernos que estos mencionados anteriormente, como por ejemplo: CSE-CIC-IDS2018, CIC-DDoS2019, CICEV2023 o CICIoV2024. Sin embargo, detallamos esos cuatro por ser los conjuntos de datos sobre los que consideramos que es más adecuado aprender a crear un modelo de aprendizaje por poseer menor complejidad en sus ataques.

7.1.2. Justificación de la elección del conjunto de datos NSL-KDD

En este segundo caso de uso se trabajará con el conjunto de datos NSL-KDD que, como hemos comentado antes, es una mejora del conjunto de datos original de detección de intrusiones en la red, llamado KDD Cup 99.

Hemos seleccionado el conjunto de datos NSL-KDD porque ha sido y sigue siendo uno de los dataset más utilizados para la detección de intrusiones hoy en día. Ya hemos visto que existen otros datasets más modernos, pero nos hemos decantado por este conjunto de datos por los siguientes motivos:

1. En el análisis de malware del primer caso de uso se ha empleado las muestras de malware más modernas y se quería hacer justo lo contrario en este caso de uso, es decir, sumergirse en los fundamentos de un dataset relativamente antiguo para aprender las bases de las técnicas de detección de intrusiones antes de pasar a datasets más complejos que implementen nuevas características como el anonimato o la encriptación.

2. Como se ha mencionado, es una mejora del dataset KDDCup99, que fue uno de los primeros conjuntos de datos en el campo de la detección de intrusiones en las redes.

3. Considero que es un dataset suficientemente maduro, ampliamente utilizado por los profesionales de la ciencia de datos de seguridad, siendo objeto de numerosos estudios y comparaciones con otros datasets que también ayudan a crear modelos de detección de intrusiones. Considero que esto nos proporciona una base sólida para poder evaluar y comparar el rendimiento de diferentes algoritmos de aprendizaje.

4. Aunque no es tan grande como algunos conjuntos de datos más modernos mencionados anteriormente, el NSL-KDD es lo suficientemente grande como para entrenar modelos de machine learning de manera efectiva y también lo suficientemente manejable para experimentación y desarrollo.

5. NSL-KDD abarca una amplia variedad de escenarios de red y tipos de ataques, lo que lo hace adecuado para entrenar y evaluar modelos de aprendizaje automático de detección de intrusiones en diferentes entornos y situaciones.

7.2. Descripción del conjunto de datos NSL-KDD

El conjunto de datos original (KDD Cup 99) fue creado y administrado por el Laboratorio Lincoln del MIT para el Programa de Evaluación de Detección de Intrusiones de DARPA. Este dataset incluye una amplia variedad de intrusiones simuladas en un entorno de red militar.

El Laboratorio Lincoln del MIT creó un entorno de prueba para adquirir durante nueve semanas, una cantidad ingente datos procedentes del volcado de todo el tráfico de red producido en una red de área local (LAN) que simulaba a una LAN típica de la Fuerza Aérea de EE. UU. Durante este tiempo, operaron esta LAN como si fuera un verdadero entorno de la Fuerza Aérea, pero la acribillaron con múltiples ataques.

Por su parte, el conjunto de datos NSL-KDD, una mejora del anterior, fue presentado en el Segundo Simposio IEEE sobre inteligencia computacional para aplicaciones de seguridad y defensa (CISDA) en 2009 por M. Tavallaee, E. Bagheri, W. Lu y A. Ghorbani.

A continuación, se presenta la lista de características (atributos) del dataset NSL-KDD:

Sr. No.	Feature	Sr. No.	Feature	Sr. No.	Feature
1	Duration	15	Su attempted	29	Same srv rate
2	Protocol type	16	Num root	30	Diff srv rate
3	Service	17	Num file creations	31	Srv diff host rate
4	Flag	18	Num shells	32	Dst host count
5	Source bytes	19	Num access files	33	Dst host srv count
6	Destination bytes	20	Num outbound cmds	34	Dst host same srv rate
7	Land	21	Is host login	35	Dst host diff srv rate
8	Wrong fragment	22	Is guest login	36	Dst host same src port rate
9	Urgent	23	count	37	Dst host srv diff host rate
10	Hot	24	Srv count	38	Dst host serror rate
11	Number failed logins	25	Serror rate	39	Dst host srvserror rate
12	Logged in	26	Srvserror rate	40	Dst host rerror rate
13	Num compromised	27	Rerror rate	41	Dst host srvrerror rate
14	Root shell	28	Srvrerror rate	42	Class label

Figura 7.1: Lista de características del dataset NSL-KDD.

Para finalizar, en la siguiente tabla, se presenta el detalle de los diferentes ataques existentes en el conjunto de datos y su clasificación en cuatro categorías.

Attack category	Attack type	KDD Cup 99		NSL-KDD	
		Training set kddcup.data10percent	Testing set corrected.gz	Training set KDDTrain+20Percent	Testing set KDDTest+
Denial of Service (DoS)	Neptune	107201	58001	8282	4657
	Smurf	164091	280790	529	665
	Pod	264	87	38	41
	Teardrop	979	12	188	12
	Land	21	9	1	7
	Back	2203	1098	196	359
	Apache2	-	794	-	737
	Udpstorm	-	2	-	2
	Process-table	-	759	-	685
	Mail-bomb	-	5000	-	293
User to Root (U2R)	Buffer-overflow	30	22	6	20
	Load-module	9	2	1	2
	Perl	3	2	0	2
	Rootkit	10	13	4	13
	Spy	2	-	1	-
	Xterm	-	13	-	13
	Ps	-	16	-	17
	Http-tunnel	-	158	-	133
	Sql-attack	-	2	-	2
	Worm	-	2	-	2
	Snmp-guess	-	2406	-	331
Remote to Local (R2L)	Guess-password	53	4367	10	1231
	Ftp-write	8	3	1	3
	Imap	12	1	5	1
	Phf	4	2	2	2
	Multihop	7	18	2	18
	Warezmaster	20	1602	7	944
	Warezclient	1020	-	181	-
	Snmpgetattack	-	7741	-	178
	Named	-	17	-	17
	Xlock	-	9	-	9
	Xsnoop	-	4	-	4
	Send-mail	-	17	-	14
Probe	Port-sweep	1040	354	587	157
	IP-sweep	1247	306	710	141
	Nmap	231	84	301	73
	Satan	1589	1633	691	735
	Saint	-	736	-	319
	Mscan	-	1053	-	996

Figura 7.2: Tipos de ciberataques existentes en el conjunto de datos NSL-KDD.

Según la documentación oficial del dataset, como podemos observar en la tabla anterior, todos estos ataques se encuentran agrupados en cuatro categorías:

1. Ataques de denegación de servicio: etiqueta **DoS**. Un ataque de denegación de servicio

(DoS) es un ataque destinado a un sistema de computadoras o red que causa que un servicio o recurso sea inaccesible para los usuarios legítimos. Los ataques DoS logran esto inundando al objetivo con tráfico o enviándole información que desencadene una falla.

2. Ataques remoto a local: etiqueta **R2L**. Con este tipo de ataques, el atacante busca obtener acceso no autorizado a una máquina víctima de la red.

3. Ataque de usuario root (escalada de privilegios): etiqueta **U2R**. Este ataque se lanza para obtener ilegalmente privilegios de root al acceder legalmente a una máquina local.

4. Ataques de sondeo de fuerza bruta: etiqueta **Probe**. El atacante escanea una máquina o un dispositivo de red para determinar las posibles vulnerabilidades existentes en la red que luego puedan ser explotadas con el objetivo de comprometer el sistema.

7.3. Creación de diferentes modelos de aprendizaje

En este apartado nos sumergiremos en el desarrollo de diferentes modelos de aprendizaje, lo que involucra una serie de pasos fundamentales, como la comprensión del problema que se está abordando o la recopilación y preparación de datos relevantes. Una vez que dispongamos de los datos adecuados, procederemos a la selección de características y al entrenamiento de diferentes algoritmos de aprendizaje.

7.3.1. Comprensión de los datos

Trabajaremos con un dataset de entrenamiento y otro de test. El conjunto de datos NSL-KDD ya proporciona varias configuraciones de conjuntos de datos que pueden emplearse para diferentes tareas. La descripción de todas las configuraciones, así como los ficheros correspondientes, se encuentran alojados en el respositorio de GitHub creado para la fase de desarrollo de este TFG, que se aloja en:

`https://github.com/ptposa/TFG`

Por tanto, emplearemos el KDDTrain+.txt como conjunto de entrenamiento y el KDD-Test+.txt como conjunto de test. De esta manera, no será necesario realizar la división del dataset en un conjunto de entrenamiento y en un conjunto de test, tal y como hemos realizado para el primer caso de uso.

Ambos dataset contienen un total de 42 características (43 con el índice), pero se diferencian en el número de registros. Mientras el dataset KDDTrain+.txt posee un total de 125.972

registros, el KDDTest+.txt contiene 22.543.

En el siguiente gráfico, se observa la proporción de tráfico legítimo y de ciberataques que posee el dataset de entrenamiento (KDDTrain+.txt):

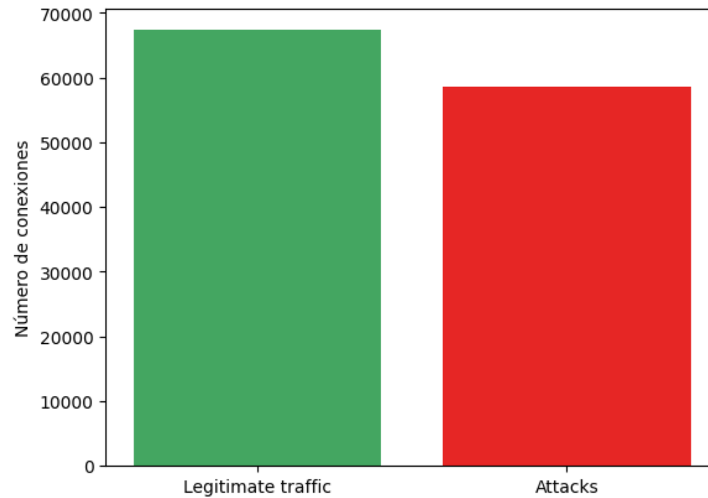


Figura 7.3: Clasificación del tráfico de red en el dataset de entrenamiento.

Por otro lado, en este otro gráfico, podemos observar la proporción de tráfico legítimo y de ciberataques que posee el dataset de test (KDDTest+.txt):

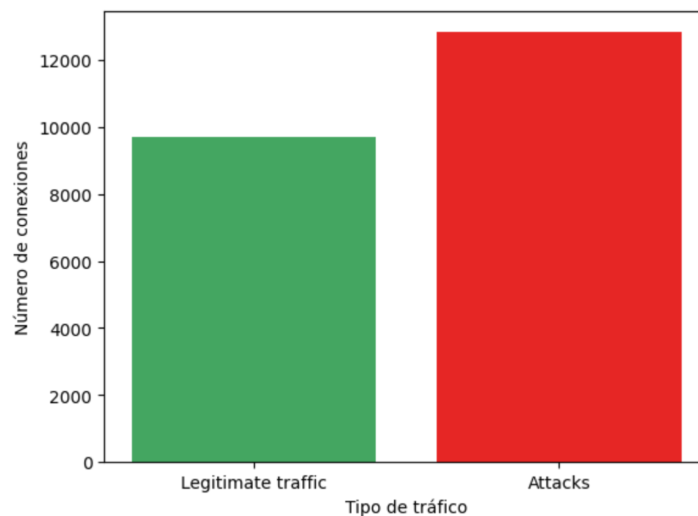


Figura 7.4: Clasificación del tráfico de red en el dataset de test.

7.3.2. Exploración de los diferentes tipos de ataques existentes en ambos datasets

Para ello, emplearemos la característica 'attack' de nuestro dataset, y mostraremos los diferentes tipos de ataques y que cantidad existe de cada uno de ellos en cada dataset (entrenamiento y test).

Diferentes tipos de ataque por cantidad que contiene el dataset de entrenamiento (KDDTrain+.txt):

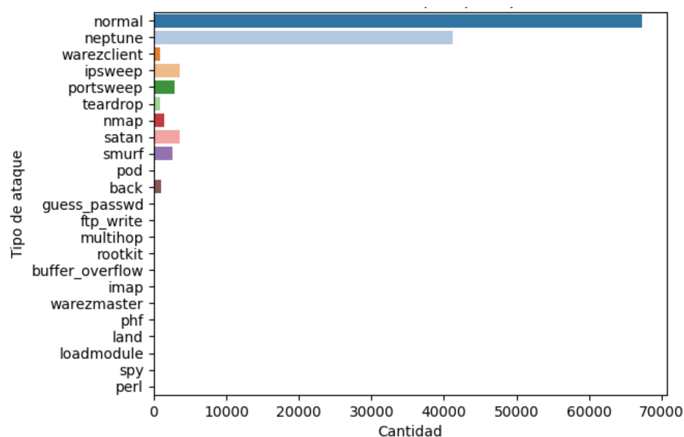


Figura 7.5: Tipos de ataque por cantidad que contiene el dataset de entrenamiento.

Diferentes tipos de ataque por cantidad que contiene el dataset de test (KDDTest+.txt):

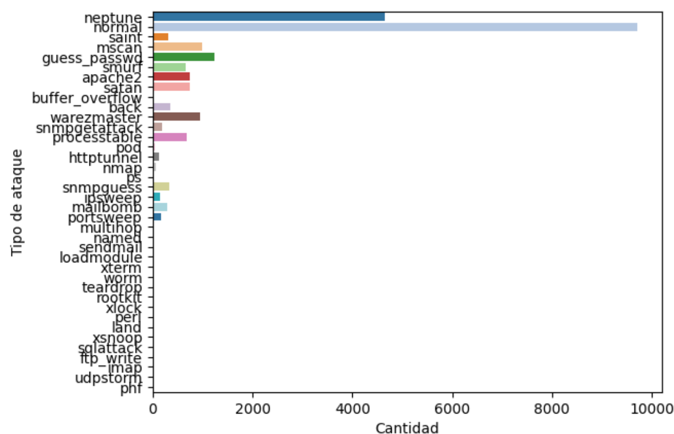


Figura 7.6: Tipos de ataque por cantidad que contiene el dataset de test.

Llegados a este punto es importante darnos cuenta que el dataset de entrenamiento (KDD-Train+.txt) contiene 22 tipos diferentes de ataques mientras que el dataset de test (KDD-Test+.txt) contiene 37. Esto se traduce en que el conjunto de datos de test posee 15 tipos de ataques adicionales, no presentes en el conjunto de datos con el que entrenaremos a nuestros modelos. Pero todo esto tiene sentido, ya que, estos conjuntos de datos fueron minuciosamente pensados y diseñados justamente para probar qué tan bien generaliza los datos de entrenamiento el modelo entrenado. Si el modelo resultante es capaz de realizar predicciones correctas sobre el tipo de tráfico que le entre, significa que este modelo aprendió con éxito algunas propiedades generalizables de cada grupo de ataques que le permitieron clasificar correctamente otros tipos de ataques nunca antes vistos.

7.3.3. Planteamiento de la estrategia a seguir

En este caso de uso se diseñará un modelo que sea capaz de clasificar el tráfico de red en cinco categorías: los cuatro grupos mencionados anteriormente (dos, r2l, u2r y probe) y la categoría de tráfico legítimo (legitimate).

En la clasificación de los ataques, hemos visto que existen cuatro grupos principales pero, sin embargo, aún no hemos visto que tipos de ataques pertenecen a cada grupo. Para ello, nos serviremos de la documentación oficial del dataset, donde podemos observar que la asignación de etiquetas de grupo de ataque a los diferentes tipos de ataques se especifica en el archivo training_attack_types.txt (puede encontrarse dentro del repositorio del TFG y en el anexo B.1).

Si fusionamos los tipos de ataques con sus grupos, obtenemos lo siguiente:

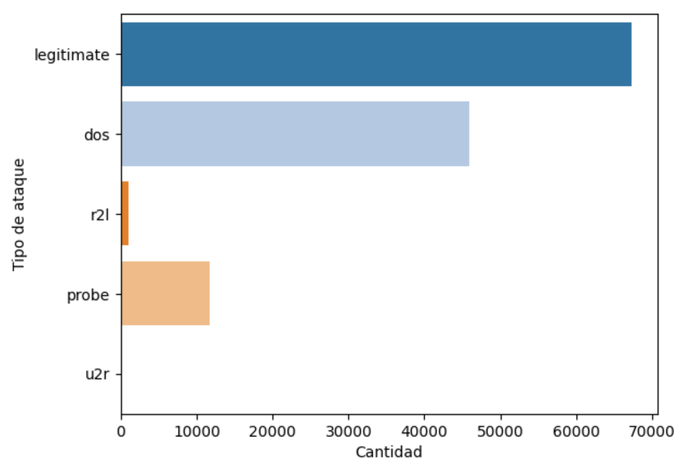


Figura 7.7: Grupos de ataque por cantidad que contiene el dataset de entrenamiento.

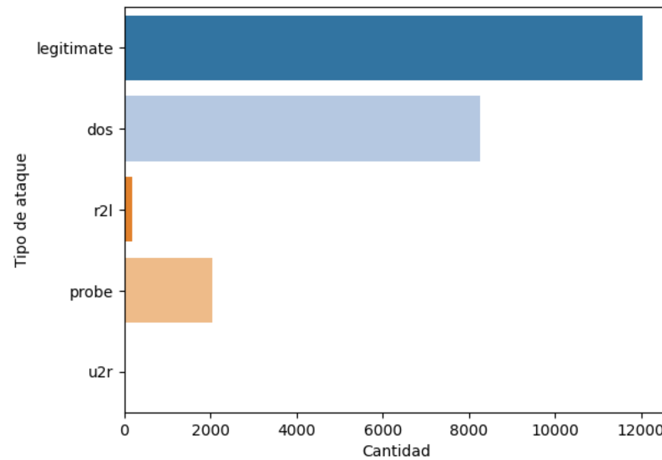


Figura 7.8: Grupos de ataque por cantidad que contiene el dataset de test.

7.3.4. Preprocesamiento de los datos

Variables simbólicas vs variables continuas

En el conjunto de datos NSL-KDD las variables pueden clasificarse como simbólicas (también conocidas como categóricas) o continuas (también conocidas como numéricas).

Variables simbólicas (categóricas): Estas son variables que representan categorías o clases discretas que no tienen un orden inherente. En nuestro dataset, estas variables podrían incluir, por ejemplo, el tipo de protocolo de red utilizado (TCP, UDP), el tipo de servicio (HTTP, FTP, etc.), entre otros. Estas variables no pueden ser tratadas como números y generalmente se codifican utilizando técnicas como One-Hot Encoding. El One-Hot Encoding transforma cada valor categórico en un vector binario que representa la presencia o ausencia de esa categoría.

Aplicado a nuestro dataset, un ejemplo de utilización de la técnica One-Hot Encoding podría ser para representar el tipo de protocolo de red utilizado. Con One-Hot Encoding, convertiremos cada valor posible del tipo de protocolo de red utilizado en un vector binario como el siguiente:

- TCP:[1,0,0]
- UDP:[0,1,0]
- ICMP:[0,0,1]

Variables continuas (numéricas): Estas son variables que representan cantidades que pueden tomar un rango infinito de valores dentro de un intervalo continuo. En nuestro dataset,

por ejemplo, estas variables podrían incluir el tiempo de duración de una conexión, el número de bytes transferidos, la tasa de transferencia de datos, entre otros. Estas variables se pueden manipular matemáticamente y esto es importante porque son las que se utilizan para realizar análisis estadísticos y para entrenar a modelos de predicción.

Es importante comprender esta distinción al analizar y modelar datos, ya que el tratamiento de cada tipo de variable requiere diferentes enfoques y técnicas.

Variables simbólicas puras vs variables simbólicas binarias

Otra aspecto importante que podemos observar dentro de las variables simbólicas es que existen variables simbólicas que tienen varias categorías internas (por ejemplo, la variable **service** puede tomar varios valores: http, private, domain_u, smtp, tftp_u, entre otros), y variables simbólicas que solo toman dos valores (variables binarias) como, por ejemplo, la variable **logged_in** que solo toma los valores 0 y 1. Es por ello que subdividiremos, a su vez, las variables simbólicas en variables simbólicas nominales (nominal) y variables simbólicas binarias (binary), ya que, las preprocesaremos de forma diferente.

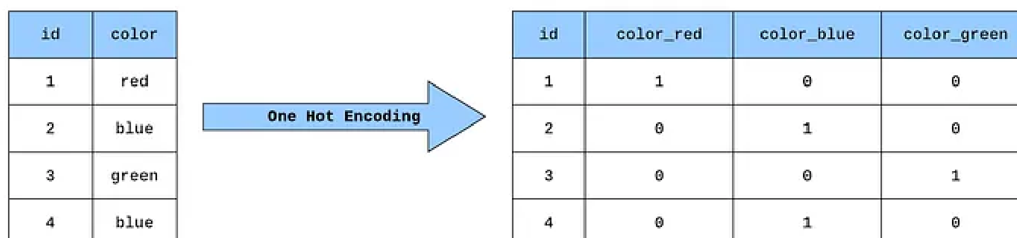
Llegados a este punto, ya hemos observado cuáles son las variables simbólicas de nuestro dataset y cuáles son las continuas. Dentro de las continuas, hemos diferenciado las variables nominales de las binarias. El siguiente paso es convertir las variables nominales en binarias, con el objetivo de permitir que los algoritmos de aprendizaje automático que emplearemos más adelante, trabajen con estas variables simbólicas a través de un formato que puedan entender y procesar de manera efectiva.

Para lograr esto, lo primero que haremos será concatenar los dataset de características de entrenamiento y de test, ya que, pueden existir algunos valores de variables simbólicas que aparezcan en un conjunto de datos y no en el otro, generarando de esta manera variables binarias independientes para cada uno de ellos, lo que daría como resultado inconsistencias en las columnas de ambos conjuntos de datos.

One Hot Encoding

One Hot Encoding es una forma común de preprocesar características categóricas para modelos de aprendizaje automático. Este tipo de codificación crea una nueva característica binaria para cada categoría posible y asigna un valor de 1 a la característica de cada muestra que corresponde a su categoría original. Las características o variables categóricas (también llamadas simbólicas) son variables que representan categorías o clases discretas que no tienen un orden inherente. En nuestro dataset, estas variables incluyen, por ejemplo, el tipo de protocolo de red utilizado (TCP, UDP), el tipo de servicio (HTTP, FTP, etc.), entre otros. Estas variables

no pueden ser tratadas como números y generalmente se codifican utilizando técnicas como One-Hot Encoding.



id	color
1	red
2	blue
3	green
4	blue

id	color_red	color_blue	color_green
1	1	0	0
2	0	1	0
3	0	0	1
4	0	1	0

Figura 7.9: One Hot Encoding.

Estandarización de los datos

Durante la exploración de los datos, observamos que algunas características del dataset tienen distribuciones muy diferentes, lo que puede influir en nuestros resultados si aplicamos, por ejemplo, un método de clasificación que se basa en la distancia. Para solucionar esto emplearemos la técnica de estandarización, que, como ya hemos visto en el caso de uso 1, se trata de un proceso que cambia la escala de una serie de datos para que tenga una media de 0 y una desviación estándar de 1.

7.3.5. Selección de características

Tal y como hemos visto en el primer caso de uso, uno de los componentes fundamentales de cualquier análisis de datos es explorar las potenciales relaciones que puedan existir entre las variables que están siendo analizadas. Para esto, se utiliza una medida estadística llamada correlación, que nos indica tanto la magnitud como la dirección de la relación que podría haber entre dos variables.

En un mapa de calor de correlación, cada variable está representada por una fila y una columna, y las celdas muestran la correlación entre ellas. El color de cada celda representa la fuerza y dirección de la correlación, y los colores mas oscuros indican correlaciones más fuertes (en nuestro caso, las correlaciones más fuertes se reflejan con colores azulados y rojizos intensos).

Los mapas de calor de correlación son importantes porque ayudan a identificar qué variables pueden dar lugar a multicolinealidad, lo que comprometería la integridad del modelo. La

multicolinealidad ocurre cuando las variables independientes (posibles predictoras) están correlacionadas. Las variables independientes deberían ser eso, independientes. Esto se debe a que si el grado de correlación entre las variables independientes es alto, no podremos aislar la relación entre cada variable independiente y la variable dependiente.

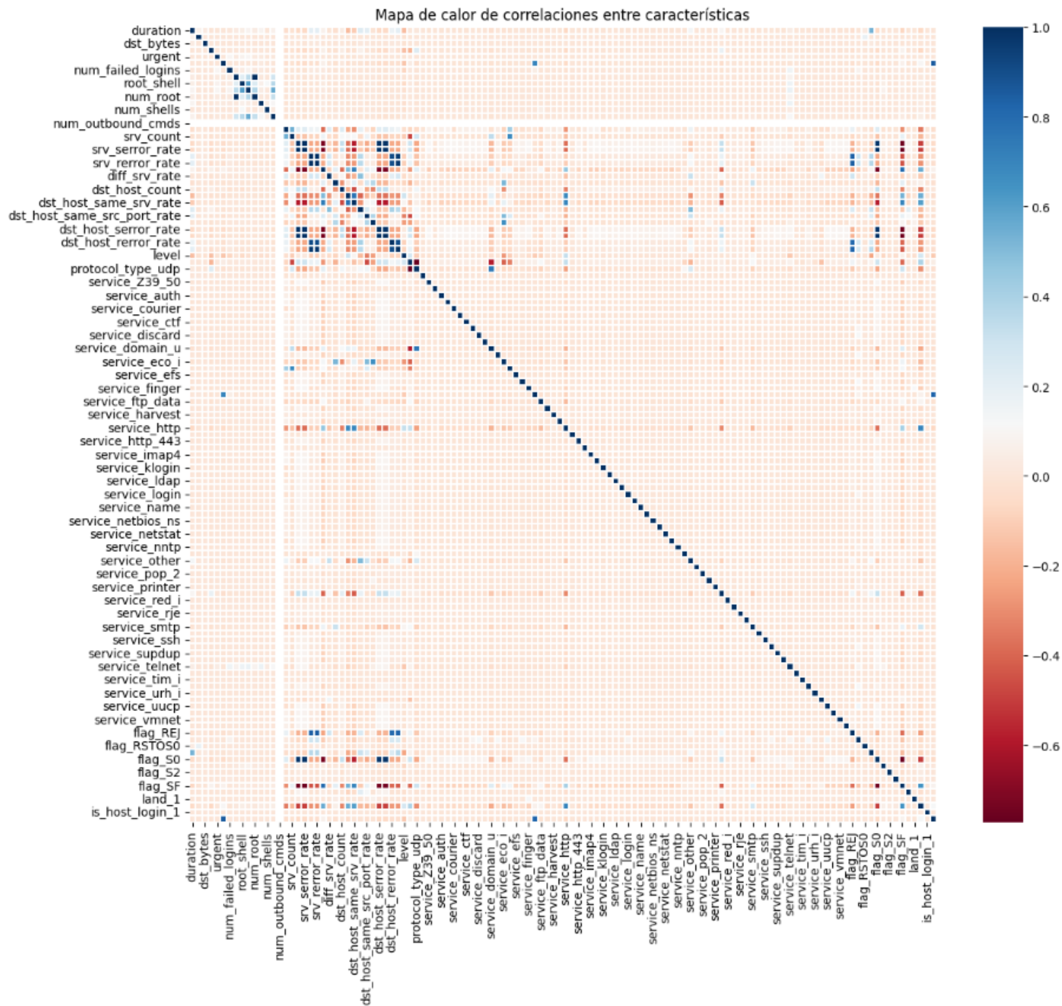


Figura 7.10: Mapa de calor de correlaciones entre características

Llegados a este punto, ya estamos en condiciones de seleccionar únicamente aquellas características que consideramos que aportarán gran valor al entrenamiento de nuestros modelos. En el *jupyter notebook* del repositorio de GitHub aparece detallado el paso a paso práctico de toda esta teoría.

7.3.6. Desarrollo de diferentes modelos de aprendizaje automático

En esta fase, abordaremos la creación de diferentes modelos de aprendizaje automático supervisado, empleando para ello diferentes algoritmos, con el objetivo de poder decidir posteriormente, cuál es el modelo que es capaz de predecir un ciberataque con mayor precisión.

Comenzaremos por crear un modelo de aprendizaje utilizando el algoritmo "Árboles de decisión"(Decision Tree) para observar cómo crear el primer modelo desde cero y, a continuación, crearemos un vector que contenga varios algoritmos para proceder a su entrenamiento de forma automatizada.

Modelo de aprendizaje automático aplicando el algoritmo "Árboles de decisión"sobre dataset sin balancear

El código referente a la creación de este modelo se encuentra tanto en el anexo B.12 de esta memoria como en el repositorio de GitHub.

Si graficamos el modelo resultante, obtenemos el siguiente árbol:

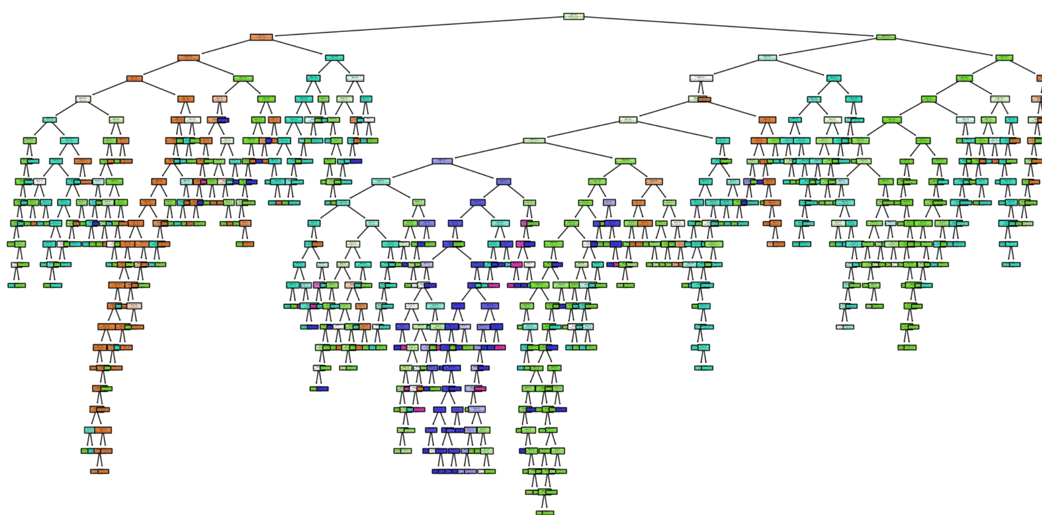


Figura 7.11: Árbol de decisión generado.

Desarrollo de diferentes modelos de aprendizaje automático

Con el objetivo de automatizar el entrenamiento de varios algoritmos de aprendizaje, se ha codificado un script en *Python*, donde hacemos lo siguiente:

1. Añadimos en un vector todos los algoritmos que deseamos aplicar para crear los diferentes modelos.
2. Recorremos cada uno de los algoritmos del vector y los entrenamos con nuestros datos para crear un modelo por cada algoritmo.
3. Realizamos validación cruzada del modelo entrenado y almacenamos tanto su media como su desviación estándar.
4. Imprimimos los resultados.

El código referente a la creación de estos modelos se encuentra tanto en el anexo B.13 de esta memoria como en el repositorio de GitHub.

Capítulo 8

Evaluación de los modelos

En el campo del aprendizaje automático, la evaluación de los modelos es una etapa crucial que determina su eficacia y capacidad para predecir datos no vistos anteriormente. Este capítulo se centra en los métodos y métricas empleados para valorar el rendimiento de los modelos desarrollados en este Trabajo Final de Grado para los dos casos de uso propuestos.

A continuación, se presenta las puntuaciones de diversos modelos de clasificación y regresión entrenados con los conjunto de datos de cada caso de uso. Para la creación de estos modelos, se han utilizado múltiples algoritmos, incluyendo regresiones, vecinos más cercanos, máquinas de soporte vectorial, árboles de decisión y bosques aleatorios, cada uno con sus propias características y capacidades.

Para asegurar una evaluación rigurosa y precisa, se han empleado técnicas de validación cruzada y se han calculado métricas clave como la exactitud y la precisión. La validación cruzada permite utilizar múltiples particiones del conjunto de datos para obtener una medida más robusta del rendimiento del modelo, evitando el sobreajuste y asegurando que los resultados sean generalizables a nuevos datos.

Las métricas de exactitud y precisión proporcionan una visión integral del rendimiento de los modelos. La exactitud mide la proporción de predicciones correctas realizadas por el modelo, mientras que la precisión evalúa la capacidad del modelo para minimizar los falsos positivos, especialmente relevante en conjuntos de datos desbalanceados.

En las siguientes líneas se compararán las puntuaciones de todos los modelos entrenados en base a estas métricas, separados por caso de uso, y se justificará la selección del mejor modelo considerando tanto su rendimiento cuantitativo como la complejidad y el tiempo de entrenamiento. La combinación de estas evaluaciones no solo nos permite identificar el modelo más preciso, sino también el más eficiente y adecuado para el problema específico abordado en cada caso de uso.

8.1. Evaluación de los modelos creados para el caso de uso 1

Los resultados arrojados del entrenamiento de los siguientes algoritmos, para el primer caso de uso, son:

Algoritmo	Puntuación media	Desviación estándar	Tiempo de entrenamiento
LogisticRegression	0.9721704	0.0023218	0.0279062
K-Nearest-Neighbors_k=2	0.9969877	0.0010039	0.0009968
K-Nearest-Neighbors_k=3	0.9971311	0.0009935	0.0009966
K-Nearest-Neighbors_k=4	0.9967008	0.0011014	0.0009966
K-Nearest-Neighbors_k=5	0.9969159	0.0005642	0.0009966
K-Nearest-Neighbors_k=7	0.9961987	0.0010682	0.0009968
SupportVectorMachine	0.9943336	0.0018735	0.2053127
NaiveBayes	0.4145683	0.0398695	0.0039868
K-Means	0.1623105	0.2067356	0.0129564
DecisionTree	0.9986372	0.0007453	0.0179398
RandomForest	0.9993545	0.0004421	0.3069735

Figura 8.1: Resultados del entreamiento de diferentes algoritmos de aprendizaje CU1

Como podemos observar, la mayoría de los modelos (excepto los creados con los algoritmos Naive Bayes y K-Means), han conseguido una exactitud (accuracy) de más del 99 por ciento. Por tanto, cualquiera de ellos sería lo suficientemente confiable para desplegar en un entorno real, y obtener de él muy buenas predicciones sobre la presencia de malware en datos desconocidos.

8.2. Evaluación de los modelos creados para el caso de uso 2

Los resultados arrojados del entrenamiento de los siguientes algoritmos, para el segundo caso de uso, son:

Algoritmo	Puntuación media	Desviación estándar	Tiempo de entrenamiento
LogisticRegression	0.9887038	0.0005099	12.3850789
K-Nearest-Neighbors_k=2	0.9972692	0.0002886	0.1245842
K-Nearest-Neighbors_k=3	0.9975074	0.0002171	0.1225905
K-Nearest-Neighbors_k=4	0.9973724	0.0002237	0.1235859
K-Nearest-Neighbors_k=5	0.9971184	0.0002421	0.1285701
K-Nearest-Neighbors_k=7	0.9969120	0.0003779	0.1176064
SupportVectorMachine	0.9900851	0.0006680	226.1421571
NaiveBayes	0.6094688	0.0945449	0.2541537
K-Means	0.0000000	0.0000000	0.3832715
DecisionTree	0.9978567	0.0003265	0.4116232
RandomForest	0.9984282	0.0002444	4.1754935

Figura 8.2: Resultados del entreamiento de diferentes algoritmos de aprendizaje CU2

En este segundo caso de uso se repite el mismo patrón que en el anterior, es decir, la mayoría de los modelos (excepto los creados con los algoritmos Naive Bayes y K-Means), han conseguido una exactitud (accuracy) de más del 99 por ciento. Por tanto, cualquiera de ellos sería lo suficientemente confiable para desplegar en un entorno real y obtener de él muy buenas predicciones sobre la presencia de anomalías de red en datos desconocidos.

Capítulo 9

Conclusiones

El aumento significativo de la generación de malware y de los ciberataques en los últimos años, y la necesidad de proteger eficazmente los sistemas informáticos, han sido las motivaciones que han guiado este Trabajo Final de Grado. Para contribuir a la mejora de los sistemas de defensa existentes, se ha investigado sobre los diferentes algoritmos y técnicas de aprendizaje automático, y cómo estas herramientas, destacadas en el ámbito de la inteligencia artificial, podían ayudar en esta labor.

En este TFG se ha analizado el uso de diferentes algoritmos y técnicas de aprendizaje automático supervisado para identificar, por un lado, malware en archivos PE y, por otro, amenazas de red, con el objetivo de crear modelos que contribuyan a la detección temprana de malware y de ciberataques, y con esto, a la mitigación de su impacto.

Para conseguir el mejor modelo para cada caso de uso y, con el objetivo de tener en cuenta el balanceo de los datos, se ha optado por comparar los modelos en tres enfoques de nuestros datasets: sin balanceo, con submuestreo y con sobremuestreo.

A lo largo de la fase de desarrollo de este TFG, desarrollamos diversos modelos, utilizando algoritmos de aprendizaje automático como: Regresión Logística, Máquinas de Vectores de Soporte, K-Vecinos más cercanos (KNN) con diferentes valores de K, Naive Bayes, Árboles de Decisión o Bosques Aleatorios. Se crearon un total de 11 modelos para cada caso de uso, que fueron validados y evaluados mediante diferentes métricas, mostrando muy buenos resultados de predicción con nuevos datos desconocidos.

Para finalizar, podemos confirmar que la creación de estos modelos ha sido de gran valor para el aprendizaje personal y consideramos que también ha sido un pequeño aporte para este campo de la informática cada vez más en auge: la Seguridad Informática.

Capítulo 10

Trabajo futuro

Considerando que los objetivos del TFG, descritos en la sección 1.2, han sido en su mayoría conseguidos, existen varias direcciones en las que este trabajo puede ser ampliado y mejorado en el futuro:

1. Aplicación de técnicas de aprendizaje profundo. Explorar el uso de redes neuronales más complejas, como redes convolucionales (CNN) o redes recurrentes (RNN), podría mejorar el rendimiento del sistema en la detección de malware y de ciberataques.

2. Evaluación en entornos reales. Implementar y evaluar los modelos en entornos de producción reales para validar su efectividad y eficiencia en situaciones del mundo real.

3. Análisis de rendimiento en tiempo real. Examinar la capacidad de los modelos para detectar malware y ciberataques en tiempo real y optimizar su desempeño para minimizar la latencia y el consumo de recursos.

4. Ampliación del estudio a otros tipos de archivos o sistemas. Por ejemplo, extender el enfoque utilizado para archivos PE a otros formatos de archivos y sistemas operativos, ampliando el alcance del sistema de detección de malware.

Bibliografía

[1]VirusTotal. *Análisis de archivos, dominios, IP y URL sospechosos para detectar malware.* URL: https://www.virustotal.com. (acceso: 01.04.2024).
[2]VirusShare. *Because Sharing is Caringt.* URL: https://virusshare.com. (acceso: 01.04.2024).
[3]Pefile. *Work with Portable Executable files.* URL: https://github.com/erocarrera/pefile. (acceso: 01.04.2024).
[4]M. Tavallaee E. Bagheri. *A Detailed Analysis of the KDD CUP 99 Data Set.* URL: https://www.unb.ca/cic/datasets/nsl.html. (acceso: 29.04.2024).
[5]Kaspersky. *Machine Learning in Cybersecurity.* URL: https://www.kaspersky.com/enterprise-security/wiki-section/products/machine-learning-in-cybersecurity. (acceso: 01.04.2024).
[6]Windows Defender Security Center. *Microsoft Defender uses ML.NET to stop malware.* URL: https://dotnet.microsoft.com/en-us/platform/customers/microsoft-defender. (acceso: 01.04.2024).
[7]Avast. *AI and machine learning.* URL: https://www.avast.com/technology/ai-and-machine-learning#pc. (acceso: 01.04.2024).
[8]OSSEC+. *Server Intrusion Detection for Every Platform.* URL: https://www.ossec.net/. (acceso: 01.04.2024).
[9]RETN. *DDoS Protection.* URL: https://retn.net/products/ddos-protection. (acceso: 01.04.2024).
[10]HOGZILLA IDS. *Big data technologies empowering your detection capabilities.* URL: https://ids-hogzilla.org/. (acceso: 01.04.2024).
[11]David Masip i Rodó Vicenç Torra i Reventós. *Aprendizaje Computacional.* PID00200713. FUOC, 2013.
[12]Scikit-learn. *Choosing the right estimator.* URL: https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html. (acceso: 03.04.2024).
[13]ArcGIS Pro. *Cómo funciona el algoritmo Regresión lineal.* URL: https://pro.arcgis.com/es/pro-app/latest/tool-reference/geoai/how-linear-regression-works.htm. (acceso: 07.04.2024).
[14]Brayan Buitrago. *Regresión Logística.* URL: https://medium.com/iwannabedatadriven/regresi%C3%B3n-log%C3%ADstica-i-machine-learning-84ffe9d6be15. (acceso: 07.04.2024).
[15]Wikipedia. *K-medias.* URL: https://es.wikipedia.org/wiki/K-medias. (acceso: 07.04.2024).

[16]Escuela de Ingeniería Informática. Universidad de Oviedo. *El algoritmo k-means*. URL: https://www.unioviedo.es/compnum/laboratorios_py/new/kmeans.html. (acceso: 07.04.2024).
[17]LibreTexts. *K-Nearest Neighbors*. URL: https://stats.libretexts.org/Bookshelves/Computing_and_Modeling/RTG%3A_Classification_Methods/3%3A_K-Nearest_Neighbors_(KNN). (acceso: 07.04.2024).
[18]Yousef Nami. *Increasing k Decrease Variance in kNN*. URL: https://towardsdatascience.com. (acceso: 08.04.2024).
[19]Avinash Navlani. *Decision Tree Classification*. URL: https://www.datacamp.com/tutorial/decision-tree-classification-python. (acceso: 05.05.2024).
[20]IBM. *Árboles de decisión*. URL: https://www.ibm.com/es-es/topics/decision-trees. (acceso: 05.05.2024).
[21]Wikipedia. *Random forest*. URL: https://en.wikipedia.org/wiki/Random_forest. (acceso: 05.05.2024).
[22]Meet Ajay. *Demystifying Naïve Bayes*. URL: https://medium.com/@dancerworld60/demystifying-na%C3%AFve-bayes-simple-yet-powerful-for-text-classification-ad92b14a5c7. (acceso: 07.05.2024).
[23]Wikipedia. *Portable Executable*. URL: https://en.wikipedia.org/wiki/Portable_Executable. (acceso: 04.05.2024).
[24]Builtin. *Split Dataset*. URL: https://builtin.com. (acceso: 05.05.2024).
[25]Wikipedia. *Validación cruzada*. URL: https://es.wikipedia.org/wiki/Validaci%C3%B3n_cruzada. (acceso: 05.05.2024).

Apéndice A

CU1. Detección de malware

A.1. Código en Python para crear el conjunto de datos a partir de ficheros PE

```
# Bibliotecas necesarias
import pefile
import csv
import os
import sys

# Localizacion de las muestras
malware = 'E:/jtripos/TFG/malware/'
legitimate = 'E:/jtripos/TFG/legitimate/'

# Nombre de los datasets que se crearan
malwareDatasetName = 'malwareDataset.csv'
legitimateDatasetName = 'legitimateDataset.csv'


# Definicion de la estructura de un fichero PE por orden de aparicion
def PEFileStructure(name, pe, malware):
    features = {'Name': name,
        'e_magic': pe.DOS_HEADER.e_magic,
        'e_cblp': pe.DOS_HEADER.e_cblp,
        'e_cp': pe.DOS_HEADER.e_cp,
        'e_crlc': pe.DOS_HEADER.e_crlc,
        'e_cparhdr': pe.DOS_HEADER.e_cparhdr,
        'e_minalloc': pe.DOS_HEADER.e_minalloc,
        'e_maxalloc': pe.DOS_HEADER.e_maxalloc,
        'e_ss': pe.DOS_HEADER.e_ss,
        'e_sp': pe.DOS_HEADER.e_sp,
        'e_csum': pe.DOS_HEADER.e_csum,
        'e_ip': pe.DOS_HEADER.e_ip,
        'e_cs': pe.DOS_HEADER.e_cs,
        'e_lfarlc': pe.DOS_HEADER.e_lfarlc,
        'e_ovno': pe.DOS_HEADER.e_ovno,
        'e_oemid': pe.DOS_HEADER.e_oemid,
        'e_oeminfo': pe.DOS_HEADER.e_oeminfo,
        'e_lfanew': pe.DOS_HEADER.e_lfanew,
        'Machine': pe.FILE_HEADER.Machine,
        'NumberOfSections': pe.FILE_HEADER.NumberOfSections,
        'TimeDateStamp': pe.FILE_HEADER.TimeDateStamp,
```

```
39          'PointerToSymbolTable': pe.FILE_HEADER.PointerToSymbolTable,
40          'NumberOfSymbols': pe.FILE_HEADER.NumberOfSymbols,
41          'SizeOfOptionalHeader': pe.FILE_HEADER.SizeOfOptionalHeader,
42          'Characteristics': pe.FILE_HEADER.Characteristics,
43          'Magic': pe.OPTIONAL_HEADER.Magic,
44          'MajorLinkerVersion': pe.OPTIONAL_HEADER.MajorLinkerVersion,
45          'MinorLinkerVersion': pe.OPTIONAL_HEADER.MinorLinkerVersion,
46          'SizeOfCode': pe.OPTIONAL_HEADER.SizeOfCode,
47          'SizeOfInitializedData': pe.OPTIONAL_HEADER.SizeOfInitializedData,
48          'SizeOfUninitializedData': pe.OPTIONAL_HEADER.SizeOfUninitializedData,
49          'AddressOfEntryPoint': pe.OPTIONAL_HEADER.AddressOfEntryPoint,
50          'BaseOfCode': pe.OPTIONAL_HEADER.BaseOfCode,
51          'ImageBase': pe.OPTIONAL_HEADER.ImageBase,
52          'SectionAlignment': pe.OPTIONAL_HEADER.SectionAlignment,
53          'FileAlignment': pe.OPTIONAL_HEADER.FileAlignment,
54          'MajorOperatingSystemVersion': pe.OPTIONAL_HEADER.MajorOperatingSystemVersion,
55          'MinorOperatingSystemVersion': pe.OPTIONAL_HEADER.MinorOperatingSystemVersion,
56          'MajorImageVersion': pe.OPTIONAL_HEADER.MajorImageVersion,
57          'MinorImageVersion': pe.OPTIONAL_HEADER.MinorImageVersion,
58          'MajorSubsystemVersion': pe.OPTIONAL_HEADER.MajorSubsystemVersion,
59          'MinorSubsystemVersion': pe.OPTIONAL_HEADER.MinorSubsystemVersion,
60          'SizeOfHeaders': pe.OPTIONAL_HEADER.SizeOfHeaders,
61          'CheckSum': pe.OPTIONAL_HEADER.CheckSum,
62          'SizeOfImage': pe.OPTIONAL_HEADER.SizeOfImage,
63          'Subsystem': pe.OPTIONAL_HEADER.Subsystem,
64          'DllCharacteristics': pe.OPTIONAL_HEADER.DllCharacteristics,
65          'SizeOfStackReserve': pe.OPTIONAL_HEADER.SizeOfStackReserve,
66          'SizeOfStackCommit': pe.OPTIONAL_HEADER.SizeOfStackCommit,
67          'SizeOfHeapReserve': pe.OPTIONAL_HEADER.SizeOfHeapReserve,
68          'SizeOfHeapCommit': pe.OPTIONAL_HEADER.SizeOfHeapCommit,
69          'LoaderFlags': pe.OPTIONAL_HEADER.LoaderFlags,
70          'NumberOfRvaAndSizes': pe.OPTIONAL_HEADER.NumberOfRvaAndSizes,
71          'Malware': malware
72          }
73
74      return features
75
76
77  # Creacion del dataset de malware
78  with open(malwareDatasetName, 'w+', newline='') as csvfile:
79      try:
80          PEFiles = os.listdir(legitimate)
81          fields = PEFileStructure('test', pefile.PE(legitimate + PEFiles[0], fast_load=True),
                0)
82          fieldNames = fields.keys()
83          writer = csv.DictWriter(csvfile, fieldnames=fieldNames)
84          writer.writeheader()
85
86          for fileName in os.listdir(malware):
87              try:
88                  pe = pefile.PE(malware + fileName)
89                  writer.writerow(PEFileStructure(fileName, pe, 1))
90              except Exception as e:
91                  print(e)
92
93      except Exception as e:
94          print(e)
95
96  # Creacion del dataset de archivos legitimos
97  with open(legitimateDatasetName, 'w+', newline='') as csvfile:
98      try:
99          PEFiles = os.listdir(legitimate)
100         fields = PEFileStructure('test', pefile.PE(legitimate + PEFiles[0], fast_load=True),
                0)
```

```
101         fieldNames = fields.keys()
102         writer = csv.DictWriter(csvfile, fieldnames=fieldNames)
103         writer.writeheader()
104 
105         for fileName in os.listdir(legitimate):
106             try:
107                 pe = pefile.PE(legitimate + fileName)
108                 writer.writerow(PEFileStructure(fileName, pe, 0))
109             except Exception as e:
110                 print(e)
111 
112     except Exception as e:
113         print(e)
```

Listing A.1: Creación del conjunto de datos a partir de ficheros PE

A.2. Bibliotecas necesarias para la creación de los modelos

```
1     # Bibliotecas generales
2     import numpy as np
3     import pandas as pd
4     import seaborn as sns
5     import matplotlib.pyplot as plt
6     from matplotlib.colors import ListedColormap
7     import statistics
8     import sklearn
9     import time
10    import os
11 
12    # Estandarizacion
13    from sklearn.preprocessing import StandardScaler
14 
15    # Division del dataset en un conjunto de entrenamiento y en un conjunto de test
16    from sklearn.model_selection import train_test_split
17 
18    # Sobremuestreo y submuestreo
19    from imblearn.over_sampling import SMOTE
20    from imblearn.under_sampling import NearMiss
21 
22    # Validacion cruzada
23    from sklearn.model_selection import KFold
24    from sklearn.model_selection import cross_val_score
25 
26    # Bibliotecas de Algoritmos
27    from sklearn.linear_model import LinearRegression
28    from sklearn.linear_model import LogisticRegression
29    from sklearn.cluster import KMeans
30    from sklearn.neighbors import KNeighborsClassifier
31    from sklearn.tree import DecisionTreeClassifier, plot_tree
32    from sklearn.ensemble import RandomForestClassifier
33    from sklearn.naive_bayes import GaussianNB
34    from sklearn.svm import SVC
35 
36    # Metricas de precision
37    from sklearn.metrics import accuracy_score
38 
39    # Imprimir datos tabulados
```

```
40    from tabulate import tabulate
41
42    # Almacenar los modelos
43    import pickle
```

Listing A.2: Bibliotecas necesarias

A.3. Exploración del conjunto de datos generado

```
1     # Carga del conjunto de datos
2     for dirname, _, filenames in os.walk('C:/UOC/jtripos/TFG/CU1/'):
3         for filename in filenames:
4             print(os.path.join(dirname, filename))
5
6     malwareDataset = pd.read_csv('C:/UOC/jtripos/TFG/CU1/malwareDataset.csv')
7     legitimateDataset = pd.read_csv('C:/UOC/jtripos/TFG/CU1/legitimateDataset.csv')
8
9     # Fusion de los datasets de malware y de archivos legitimos en un unico dataset llamado
          fullDataset
10    fullDataset = pd.concat([legitimateDataset, malwareDataset], ignore_index=True)
11    fullDataset.to_csv('C:/UOC/jtripos/TFG/CU1/fullDataset.csv')
12
13    # Exploracion del conjunto de datos
14    fullDataset.shape
15
16    # Observamos las 54 caracteristicas
17    fullDataset.info()
```

Listing A.3: Exploración del conjunto de datos

A.4. Creación de funciones que se utilizaran para la creacion de los modelos de aprendizaje

```
1     # Funcion que devuelve aquellas caracteristicas del dataset cuya desviacion estandar es 0
2     def getZeroStdFeatures():
3         global malwareDataset
4         global legitimateDataset
5
6         legitimate = legitimateDataset
7         malware = malwareDataset
8
9         zeroFeat = ['Name', 'Malware']
10        staticsFeats = ['Name', 'Malware']
11
12        leg = legitimate.drop(staticsFeats, axis=1)
13        mal =malware.drop(staticsFeats, axis=1)
14
15        for feature in leg.columns:
16            if leg[feature].std() == 0:
17                zeroFeat.append(feature)
18
19        for feature in mal.columns:
20            if mal[feature].std() == 0:
21                if feature not in zeroFeat:
```

```
22                  zeroFeat.append(feature)
23
24      return zeroFeat
25
26
27  # Funcion para obtener todas las caracteristicas del dataset
28  def getAllFeatures():
29      global fullDataset
30
31      fullData = fullDataset
32
33      allFeats = []
34      staticsFeats = ['Name', 'Malware']
35
36      full = fullData.drop(staticsFeats, axis=1)
37
38      for feature in full.columns:
39          allFeats.append(feature)
40
41      return allFeats
42
43
44  # Funcion de estandarizacion
45  def performStandardization(X_train):
46  # Proceso de estandarizacion de las caracteristicas donde se centran alrededor de cero (
        media = 0) y se escalan para tener una varianza unitaria (desviacion estandar = 1).
        Luego, este objeto final (result) se ajustara a los datos de entrenamiento X_train
        para calcular la media y la desviacion estandar de cada caracteristica.
47  result = StandardScaler().fit(X_train)
48  return result
49
50
51  # Funcion para dividir el dataset en un conjunto de entrenamiento y en un conjunto de test
52  def splitInTrainAndTest(X, y):
53      # Dividimos del conjunto de datos en dos partes: un conjunto de entrenamiento y un
            conjunto de prueba.
54      X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state
            =42)
55
56      # Aplicamos la tecnica de estandarizacion al conjunto de entrenamiento
57      standarization = performStandardization(X_train)
58
59      # Partiendo de los datos de X_train, los transformamos segun las estadisticas de la
            transformacion de estandarizacion calculada en la linea anterior. Esta
            transformacion ajustara las caracteristicas para que tengan una media de 0 y una
            desviacion estandar de 1.
60      X_train = standarization.transform(X_train)
61      X_test = standarization.transform(X_test)
62
63      return X_train, X_test, y_train, y_test
64
65  # Funcion para balancear el conjunto de datos. Recibe todo el conjunto "X" de
        caracteristicas y todo el conjunto "y" de muestras clasificadas, y realiza un balance
        de los datos dependiendo de si el remuestreo (resample) es de oversampling o de
        undersampling.
66  def balanceDataset(X, y, resample):
67
68      if (resample == "o"):
69          # Aplicamos tecnica de oversampling
70          smote = SMOTE(random_state=42)
71          X, y = smote.fit_resample(X, y)
72
73      elif (resample == "u"):
74          # Aplicamos tecnica de undersampling
```

```
75            nearmiss = NearMiss(version=1)
76            X, y = nearmiss.fit_resample(X, y)
77
78        return X, y
```

Listing A.4: Creación de funciones

A.5. División del dataset en un conjunto de entrenamiento y en un conjunto de test

```
1   # Funcion que recibe todo el conjunto "X" de caracteristicas y todo el conjunto "y" de
        muestras clasificadas, del dataset completo de muestras de malware y archivos
        legitimos, y lo divide en dos conjuntos: un conjunto de entrenamiento (80 por ciento)
        y un conjunto de validacion (20 por ciento).
2
3   def splitInTrainAndTest(X, y):
4   # Dividimos del conjunto de datos en dos partes: un conjunto de entrenamiento y un
        conjunto de prueba.
5   X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
6
7   # Aplicamos la tecnica de estandarizacion al conjunto de entrenamiento
8   standarization = performStandardization(X_train)
9
10  # Partiendo de los datos de X_train, los transformamos segun las estadisticas de la
        transformacion de estandarizacion calculada en la linea anterior.
11  # Esta transformacion ajustara las caracteristicas para que tengan una media de 0 y una
        desviacion estandar de 1.
12  X_train = standarization.transform(X_train)
13  X_test = standarization.transform(X_test)
14
15  return X_train, X_test, y_train, y_test
```

Listing A.5: División del dataset

A.6. Balanceo del conjunto de datos

```
1   # Funcion que recibe todo el conjunto "X" de caracteristicas y todo el conjunto "y" de
        muestras clasificadas, y realiza un balance de los datos dependiendo de si el
        remuestreo (resample) es de oversampling o de undersampling.
2   def balanceDataset(X, y, resample):
3
4   if (resample == "o"):
5   # Aplicamos tecnica de oversampling
6   smote = SMOTE(random_state=42)
7   X, y = smote.fit_resample(X, y)
8
9   elif (resample == "u"):
10  # Aplicamos tecnica de undersampling
11  nearmiss = NearMiss(version=1)
12  X, y = nearmiss.fit_resample(X, y)
13
14  return X, y
```

Listing A.6: Balanceo del conjunto de datos

A.7. Análisis de las características del conjunto de datos

```
feats = getAllFeatures()
i=1

plt.style.use('bmh')
plt.rcParams['figure.max_open_warning'] = 60
plt.figure(figsize=(12, 140))

for feat in feats:
plt.figure(figsize=(12, 100))
x1 = plt.subplot(len(feats), 2, i)
sns.kdeplot(data=fullDataset[fullDataset['Malware']==0][feat], bw_adjust=0.1, ax=x1,
    warn_singular=False)
x1.set_title('Legitimate', fontsize=9, color='#44A661')
x2 = plt.subplot(len(feats), 2, i+1)
sns.kdeplot(data=fullDataset[fullDataset['Malware']==1][feat], bw_adjust=0.1, ax=x2,
    warn_singular=False)
x2.set_title('Malware', fontsize=9, color='#E62625')
i= i+2
```

Listing A.7: Análisis de las características del conjunto de datos

A.8. Construcción del dataset final

```
# Dividimos las variables independientes (caracteristicas) de nuestra variable objetivo (
    Malware) en dos datasets.
X = fullDataset.iloc[:, :-1]
y = fullDataset.loc[:, ['Malware']]

# Obtenemos aquellas caracteristicas con desviacion estandar cero y las eliminamos del
    dataset de caracteristicas.
zeroStdFeatures = getZeroStdFeatures()
X = fullDataset.drop(zeroStdFeatures, axis=1)

# Eliminamos, nuevamente del dataset de caracteristicas, aquellas caracteristicas que
    tienen un indice de correlacion superior a +0.8 o inferior a -0.8.
X = X.drop(corrFeatures, axis=1)

# Almacenamos en la variable finalFeats las 30 caracteristicas finales que hemos elegido
    en el apartado anterior. Estas caracteristicas las necesitaremos para luego graficar
    el arbol de decision a partir de esta matriz NumPy.
finalFeats = X.columns

# Dividimos el dataset en el conjunto de entrenamiento y en el conjunto de test.
X_train, X_test, y_train, y_test = splitInTrainAndTest(X, y)
```

Listing A.8: Construcción del dataset final

A.9. Desarrollo de diferentes modelos

```
# Creamos una lista de listas para los datos que contendra la tabla de resultados
resultTable = []

```

```
    # Agregamos encabezados de columna a la tabla de resultados
    headers = ["Algoritmo", "Puntuacion media", "Desviacion estandar", "Tiempo de
        entrenamiento"]

    # Seleccionamos los algoritmos
    algorithmsToUse = []

    # Algoritmos de regresion
    #algorithmsToUse.append(('LinealRegression', LinearRegression()))
    algorithmsToUse.append(('LogisticRegression', LogisticRegression(max_iter=700)))

    # Algoritmos de clasificacion
    algorithmsToUse.append(('K-Nearest-Neighbors_k=2', KNeighborsClassifier(n_neighbors=2)))
    algorithmsToUse.append(('K-Nearest-Neighbors_k=3', KNeighborsClassifier(n_neighbors=3)))
    algorithmsToUse.append(('K-Nearest-Neighbors_k=4', KNeighborsClassifier(n_neighbors=4)))
    algorithmsToUse.append(('K-Nearest-Neighbors_k=5', KNeighborsClassifier(n_neighbors=5)))
    algorithmsToUse.append(('K-Nearest-Neighbors_k=7', KNeighborsClassifier(n_neighbors=7)))
    algorithmsToUse.append(('SupportVectorMachine', SVC()))
    algorithmsToUse.append(('NaiveBayes', GaussianNB()))

    # Algoritmos de agrupamiento o clustering
    algorithmsToUse.append(('K-Means', KMeans()))

    # Algoritmos basados en arboles de decision
    algorithmsToUse.append(('DecisionTree', DecisionTreeClassifier()))
    algorithmsToUse.append(('RandomForest', RandomForestClassifier()))

    # Creamos el vector que almacenara nuestros algoritmos entrenados
    trainedModels = []

    results = []
    names = []

    # Entrenamos a los diferentes algoritmos con nuestros conjuntos de entrenamiento
    for algorithmName, algorithm in algorithmsToUse:
    # Validacion cruzada
    kfold = KFold(7)
    # Medimos el tiempo de entrenamiento
    startTime = time.time()
    trainedModel = algorithm.fit(X_train, np.ravel(y_train))
    endTime = time.time()

    cvScores = cross_val_score(trainedModel, X_train, np.ravel(y_train), cv = kfold, scoring='
        accuracy')
    results.append(cvScores)
    names.append(algorithmName)

    # Limitamos la cantidad de decimales a 7
    meanScore = round(cvScores.mean(), 7)
    stdDev = round(cvScores.std(), 7)
    elapsedTime = round(endTime - startTime, 7)

    # Agregar los datos de cada algoritmo a la lista de datos de la tabla
    resultTable.append([algorithmName, meanScore, stdDev, elapsedTime])

    trainedModels.append((algorithmName, trainedModel))

    # Imprimimos la tabla utilizando tabulate
    print(tabulate(resultTable, headers=headers, floatfmt=".7f"))
```

Listing A.9: Desarrollo de diferentes modelos

Apéndice B

CU2. Detección de ciberataques

B.1. Etiquetas de grupo de ataque de los diferentes tipos de ataques existentes

```
apache2 dos
back dos
buffer_overflow u2r
ftp_write r2l
guess_passwd r2l
httptunnel u2r
imap r2l
ipsweep probe
land dos
loadmodule u2r
mailbomb dos
mscan probe
multihop r2l
named r2l
neptune dos
nmap probe
perl u2r
phf r2l
pod dos
portsweep probe
processtable dos
ps u2r
rootkit u2r
saint probe
satan probe
sendmail r2l
smurf dos
snmpgetattack dos
snmpguess r2l
spy r2l
sqlattack u2r
teardrop dos
udpstorm dos
warezclient r2l
warezmaster r2l
worm probe
worm r2l
xlock r2l
```

```
xsnoop r2l
xterm u2r
```

Listing B.1: Creación de funciones

B.2. Exploración del conjunto de datos generado

```
    # Carga del conjunto de datos
    for dirname, _, filenames in os.walk('C:/UOC/jtripos/TFG/CU2/'):
    for filename in filenames:
    print(os.path.join(dirname, filename))

    dataPath = 'C:/UOC/jtripos/TFG/CU2/'
    trainDataset = pd.read_csv('C:/UOC/jtripos/TFG/CU2/KDDTrain+.txt')

    trainDataset.shape
    testDataset.shape

    trainDataset.head()
    testDataset.head()

    trainDataset.info()
    testDataset.info()
```

Listing B.2: Exploración del conjunto de datos

B.3. Adición del nombre de las características al dataset

```
    features = (['duration',
    'protocol_type',
    'service',
    'flag',
    'src_bytes',
    'dst_bytes',
    'land',
    'wrong_fragment',
    'urgent',
    'hot',
    'num_failed_logins',
    'logged_in',
    'num_compromised',
    'root_shell',
    'su_attempted',
    'num_root',
    'num_file_creations',
    'num_shells',
    'num_access_files',
    'num_outbound_cmds',
    'is_host_login',
    'is_guest_login',
    'count',
    'srv_count',
    'serror_rate',
    'srv_serror_rate',
    'rerror_rate',
```

```
    'srv_rerror_rate',
    'same_srv_rate',
    'diff_srv_rate',
    'srv_diff_host_rate',
    'dst_host_count',
    'dst_host_srv_count',
    'dst_host_same_srv_rate',
    'dst_host_diff_srv_rate',
    'dst_host_same_src_port_rate',
    'dst_host_srv_diff_host_rate',
    'dst_host_serror_rate',
    'dst_host_srv_serror_rate',
    'dst_host_rerror_rate',
    'dst_host_srv_rerror_rate',
    'attack',
    'level'
    ])

    trainDataset.columns = features
    testDataset.columns = features
```

Listing B.3: Creación de funciones

B.4. Clasificación del tráfico de red en el dataset de entrenamiento

```
isAttack = []

for i in trainDataset.attack :
if i == 'normal':
isAttack.append('0')
else:
isAttack.append('1')


labelsGraph = ['Legitimate traffic', 'Attacks']
legitimateTraffic = isAttack.count('0')
attacksTraffic = isAttack.count('1')
allTrafic = [legitimateTraffic, attacksTraffic]

plt.bar(labelsGraph, allTrafic, color = ['#44A661', '#E62625'])
plt.title('Clasificacion del trafico de red en el dataset de entrenamiento')
plt.xlabel('Tipo de trafico')
plt.ylabel('Numero de conexiones')
plt.show()
```

Listing B.4: Clasificación del tráfico de red

B.5. Exploración de los diferentes tipos de ataques existentes

```
trainDataset['attack'].value_counts()
```

```
2    sns.countplot(y=trainDataset.attack, palette='tab20', hue=trainDataset.attack, legend=
         False)
3    plt.title('Cantidad de ataques por tipos')
4    plt.xlabel('Cantidad')
5    plt.ylabel('Tipo de ataque')
6    plt.show()
7
8    testDataset['attack'].value_counts()
9    sns.countplot(y=testDataset.attack, palette='tab20', hue=testDataset.attack, legend=False)
10   plt.title('Cantidad de ataques por tipos')
11   plt.xlabel('Cantidad')
12   plt.ylabel('Tipo de ataque')
13   plt.show()
```

Listing B.5: Exploración de los diferentes tipos de ataques existentes

B.6. Etiquetas de grupo de ataque de los diferentes tipos de ataques existentes

```
1    attackGroups = defaultdict(list)
2    attackGroups['legitimate'].append('normal')
3
4    with open(dataPath + 'training_attack_types.txt', 'r') as attackTypes:
5        for line in attackTypes.readlines():
6            attack, attackGroup = line.strip().split(' ')
7            attackGroups[attackGroup].append(attack)
8
9    for key, value in attackGroups.items():
10       print(key, value)
11
12   attackToGroup = {v: k for k in attackGroups for v in attackGroups[k]}
13
14   # Anadimos los grupos de ataques a nuestro dataset de entrenamiento y al de test
15   trainDataset['attack_group'] = trainDataset['attack'].map(lambda x: attackToGroup[x])
16   testDataset['attack_group'] = trainDataset['attack'].map(lambda x: attackToGroup[x])
```

Listing B.6: Etiquetas de grupo de ataque de los diferentes tipos de ataques existentes

B.7. División entre las características y el objetivo de ambos datasets

```
1    X_train_pre = trainDataset.drop(['attack_group','attack'], axis=1)
2    y_train = trainDataset['attack_group']
3
4    X_test_pre = testDataset.drop(['attack_group','attack'], axis=1)
5    y_test = testDataset['attack_group']
```

Listing B.7: División entre las características y el objetivo de ambos datasets

B.8. Tratamiento de las variables simbólicas y continuas

```
trainDataset['protocol_type'].value_counts()
variableNames = defaultdict(list)

with open(dataPath + 'kddcup.names', 'r') as kddCupNames:
    for line in kddCupNames.readlines()[1:]:
        variableName, variableType = line.strip()[:-1].split(': ')
        variableNames[variableType].append(variableName)

for key, value in variableNames.items():
    print(key, value)

mergedDataset = pd.concat([X_train_pre, X_test_pre])
continuousVariables = variableNames['continuous']

# Eliminamos la variable 'root_shell' porque, como se puede observar en el siguiente
    fragmento de codigo, se trata de una variable binaria (solo toma los valores 0 y 1).
continuousVariables.remove('root_shell')

binarySymbolicVariables = ['land','logged_in','root_shell','su_attempted','is_host_login',
    'is_guest_login']
nominalSymbolicVariables = [variable for variable in variableNames['symbolic'] if variable
     not in binarySymbolicVariables]

# Generamos para cada una de las variables simbolicas nominales, extraidas en el fragmento
     de codigo anterior, una nueva caracteristica binaria para cada categoria posible y
    asignamos un valor de 1 a la caracteristica de cada muestra que corresponda a su
    categoria original.
mergedDatasetWithOneHot = pd.get_dummies(mergedDataset, columns=variableNames['symbolic'],
     drop_first=True)

# Creamos los indices para los conjuntos de datos de entrenamiento y prueba
trainIndices = range(len(X_train_pre))
testIndices = range(len(X_train_pre), len(X_train_pre) + len(X_test_pre))

# Dividimos las caracteristicas y el objetivo de ambos datasets nuevamente. Para ello,
    extraeremos las filas correspondientes a los indices definidos anteriormente.
X_train = mergedDatasetWithOneHot.iloc[trainIndices]
X_test = mergedDatasetWithOneHot.iloc[testIndices]

# Variables binarias que se crearon para representar las categorias de las variables
    nominales.
dummyVariables = [variable for variable in X_train if variable not in mergedDataset]
```

Listing B.8: Tratamiento de las variables simbólicas y continuas

B.9. Estandarización de los datos

```
# Aplicamos la tecnica de estandarizacion al conjunto de entrenamiento
standarization = performStandardization(X_train[continuousVariables])

# Partiendo de los datos de X_train y X_test, los transformamos segun las estadisticas de
    la transformacion de estandarizacion calculada en la linea anterior. Esta
    transformacion ajustara las variables continuas para que tengan una media de 0 y una
    desviacion estandar de 1.
X_train.loc[:, continuousVariables] = standarization.transform(X_train.loc[:,
    continuousVariables]).astype('int64')
```

```
    X_test.loc[:, continuousVariables] = standarization.transform(X_test.loc[:,
        continuousVariables]).astype('int64')
```

Listing B.9: Estandarización de los datos

B.10. Análisis de la relación entre las características del conjunto de datos

```
    # Mapa de calor de correlacion entre caracteristicas del dataset
    corrMatrix = X_train.corr(numeric_only=True)
    plt.figure(figsize=(14, 12))
    sns.heatmap(corrMatrix, cmap='RdBu', linewidths = 0.1)
    plt.title('Mapa de calor de correlaciones entre caracteristicas')
    plt.show()
```

Listing B.10: Relación entre las características del conjunto de datos

B.11. Construcción de diferentes versiones del dataset

```
    # Dataset sin balancear. Aprovechamos todas las modificaciones realizadas hasta esta linea
        .
    X_train = X_train
    X_test = X_test
    y_train = y_train
    y_test = y_test

    # Dataset balanceado con sobremuestreo
    X_train_OS, y_train_OS = balanceDataset(X_train, y_train, "o")
    X_test_OS, y_test_OS = balanceDataset(X_test, y_test, "o")

    # Dataset balanceado con submuestreo
    X_train_US, y_train_US = balanceDataset(X_train, y_train, "o")
    X_test_US, y_test_US = balanceDataset(X_test, y_test, "o")
```

Listing B.11: Construcción de diferentes versiones del dataset

B.12. Creación de un modelo de aprendizaje automático con el algoritmo Árboles de decisión

```
    model = DecisionTreeClassifier()
    startTime = time.time()
    trainingModel = model.fit(X_train, y_train)
    endTime = time.time()
    timeSpent = endTime - startTime

    # Establecemos el valor de iteraciones de validacion cruzada en 7
    kfold = KFold(7)

```

```
10    # Realiza validacion cruzada y evalua el rendimiento del modelo en el conjunto de datos
11    scores = cross_val_score(trainingModel, X_train, y_train, cv = kfold, scoring='accuracy')
12
13    pred_y = model.predict(X_test)
14
15    conMat = confusion_matrix(y_test, pred_y)
16    error = zero_one_loss(y_test, pred_y)
17
18    # Graficamos el arbol de decision
19    plt.figure(figsize=(20, 10))
20    plot_tree(trainingModel, filled=True, feature_names=X_train.columns, class_names=True)
21    plt.show()
22
23    # Guardamos el modelo creado
24    pklFilename = "trained_models/CU2_DecisionTreeModel.pkl"
25
26    with open(pklFilename, 'wb') as file:
27    pickle.dump(trainingModel, file)
```

Listing B.12: Creación de un modelo de aprendizaje automático aplicando el algoritmo Árboles de decisión (Decision Tree) sobre dataset sin balancear

B.13. Automatización en la creación de varios modelos de aprendizaje aplicando diferentes algoritmos

```
1     # Creamos una lista de listas para los datos que contendra la tabla de resultados
2     resultTable = []
3
4     # Agregamos encabezados de columna a la tabla de resultados
5     headers = ["Algoritmo", "Puntuacion media", "Desviacion estandar", "Tiempo de
          entrenamiento"]
6
7     # Seleccionamos los algoritmos
8     algorithmsToUse = []
9
10    # Algoritmos de regresion
11    #algorithmsToUse.append(('LinealRegression', LinearRegression()))
12    algorithmsToUse.append(('LogisticRegression', LogisticRegression(max_iter=700)))
13
14    # Algoritmos de clasificacion
15    algorithmsToUse.append(('K-Nearest-Neighbors_k=2', KNeighborsClassifier(n_neighbors=2)))
16    algorithmsToUse.append(('K-Nearest-Neighbors_k=3', KNeighborsClassifier(n_neighbors=3)))
17    algorithmsToUse.append(('K-Nearest-Neighbors_k=4', KNeighborsClassifier(n_neighbors=4)))
18    algorithmsToUse.append(('K-Nearest-Neighbors_k=5', KNeighborsClassifier(n_neighbors=5)))
19    algorithmsToUse.append(('K-Nearest-Neighbors_k=7', KNeighborsClassifier(n_neighbors=7)))
20    algorithmsToUse.append(('SupportVectorMachine', SVC()))
21    algorithmsToUse.append(('NaiveBayes', GaussianNB()))
22
23    # Algoritmos de agrupamiento o clustering
24    algorithmsToUse.append(('K-Means', KMeans()))
25
26    # Algoritmos basados en arboles de decision
27    algorithmsToUse.append(('DecisionTree', DecisionTreeClassifier()))
28    algorithmsToUse.append(('RandomForest', RandomForestClassifier()))
29
30    # Creamos el vector que almacenara nuestros algoritmos entrenados
31    trainedModels = []
32
```

```
33     results = []
34     names = []
35
36     # Entrenamos a los diferentes algoritmos con nuestros conjuntos de entrenamiento
37     for algorithmName, algorithm in algorithmsToUse:
38     # Validacion cruzada
39     kfold = KFold(7)
40     # Medimos el tiempo de entrenamiento
41     startTime = time.time()
42     trainedModel = algorithm.fit(X_train, np.ravel(y_train))
43     endTime = time.time()
44
45     cvScores = cross_val_score(trainedModel, X_train, np.ravel(y_train), cv = kfold, scoring='
           accuracy')
46     results.append(cvScores)
47     names.append(algorithmName)
48
49     # Limitamos la cantidad de decimales a 7
50     meanScore = round(cvScores.mean(), 7)
51     stdDev = round(cvScores.std(), 7)
52     elapsedTime = round(endTime - startTime, 7)
53
54     # Agregar los datos de cada algoritmo a la lista de datos de la tabla
55     resultTable.append([algorithmName, meanScore, stdDev, elapsedTime])
56
57     trainedModels.append((algorithmName, trainedModel))
58
59     # Imprimimos la tabla utilizando tabulate
60     print(tabulate(resultTable, headers=headers, floatfmt=".7f"))
```

Listing B.13: Automatización en la creación de varios modelos de aprendizaje aplicando diferentes algoritmos