Universidad Zaragoza

Trabajo Fin de Grado

Detección de malware utilizando técnicas de machine learning

Malware detection using machine learning techniques

Autor/es
Julia Varea Palacios

Director/es
Pedro Javier Álvarez Pérez-Aradrós

Codirector/es
Razvan Raducu

Departamento
Informática e Ingeniería de Sistemas

Centro
Escuela de Ingeniería y Arquitectura

Titulación del autor
Grado en Ingeniería Informática

Escuela de Ingeniería y Arquitectura
2024

Resumen

En la era digital actual la ciberseguridad se ha convertido en un elemento crítico para individuos, empresas e instituciones de todo el mundo. Frente al aumento del número de muestras de *malware* recopiladas y de su constante evolución, las técnicas de Machine Learning se presentan como una solución novedosa para la detección efectiva del malware.

Este trabajo se centra en la detección de *malware*, identificando si una muestra es maligna o no, empleando técnicas de Machine Learning para reconocer y clasificar muestras de software maligno en sus categorías o familias correspondientes.

Mediante la automatización del proceso de detección de comportamientos maliciosos en trazas de ejecución de sistemas software y aplicando modelos de Machine Learning, el propósito de este trabajo es investigar la eficacia de técnicas de Machine Learning para la detección de malware.

Para llevar a cabo el proyecto, el primer paso es la selección de un dataset adecuado que contenga muestras de varias familias de *malware* con sus trazas de ejecución. Una vez conseguido, se comenzará el proceso de filtrado y preparado de los datos hasta obtener un dataset que se adecue a las necesidades del proyecto.

Estos datos se someten a una serie de procesos de extracción de n-gramas, de creación de diccionarios y un cálculo de features para obtener características clasificatorias para cada una de las familias de malware.

Para cada conjunto de features calculadas con cada valor de n se crean modelos de Machine Learning. Estos modelos emplean varios clasificadores (KNN, SVM, Gradient Boosting y Regresión Logística) y se calcularán tanto clasificadores simples (un clasificador por familia de malware) como clasificadores múltiples (un solo modelo capaz de clasificar todas las muestras). Una vez obtenidos todos los modelos se calculan métricas para realizar una comparación de todos ellos.

Al analizarlos se ha observado que los resultados obtenidos por los clasificadores múltiples superan los obtenidos por los clasificadores simples.

Para intentar mejorar los valores obtenidos por los modelos se implementará una mejora basada en el filtrado de categorías de interés. Se definió una categoría de interés como una categoría de llamadas al sistema asociada con comportamientos maliciosos o que hagan vulnerable al sistema.

Tras analizar los nuevos resultados se observa que la totalidad de los valores devueltos mejora, tanto en los clasificadores simples como en los clasificadores múltiples. Esto demuestra que la mejora implementada es efectiva a la hora de detectar y clasificar distintas categorías de malware, independientemente del modelo utilizado para la clasificación de las mismas.

Gracias a la automatización del proceso, este proyecto abre paso a poder realizar trabajos futuros, añadiendo más funcionalidades o aplicando el proceso a otros datasets.

Índice

1. Introducción

Este primer capítulo de introducción se abordan aspectos esenciales a la hora de entender el contexto del trabajo. Inicia poniendo en contexto esencial en el que se desarrolla el proyecto e identificando los problemas a abordar, finalizando estableciendo los objetivos del trabajo. Además, se proporcionará una visión general de la organización del documento.

1.1. Contexto del trabajo

El software malicioso, del inglés malicious software (*malware*), es un programa que realiza acciones dañinas en un sistema informático de forma intencionada y sin el conocimiento del usuario. Afecta a la integridad, disponibilidad y confidencialidad de los datos y los sistemas que los contienen, por lo que debe ser un tema que cause gran preocupación a todas las personas que usen un ordenador, ya sea en casas particulares, empresas o instituciones públicas.

En los últimos años, se ha visto un aumento considerable en el número de ciberataques, amenazas y muestras de *malware* recolectadas por los proveedores de antivirus [1] [2]. La gran mayoría de archivos considerados maliciosos son ejecutables de Windows, pudiéndose ver una disminución drástica, del 97 %, en el número de ficheros maliciosos en otros sistemas operativos como MacOS. También se puede apreciar que el número de ataques de día cero nuevos alcanzó un nuevo récord, duplicando los encontrados el año anterior [3].

Así mismo, durante 2021 se observó un aumento del 24 % respecto al año anterior del número de muestras dedicadas a explotar vulnerabilidades, y si añadimos *exploits* [4] aumentaría hasta el 30 % [3]. Por todas estas razones, el número de estudios e investigación en temas de ciberseguridad ha aumentado.

Como se puede ver en [5], el número total de muestras de *malware* y otro tipo de programas no deseados ha ido en constante crecimiento, cada año superando al anterior. En este trabajo nos centraremos en la detección de *malware* cuyo objetivo es el sistema operativo Windows debido a su predominancia y frecuencia de ocurrencia.

La identificación de *malware* se divide en dos grandes subcategorías: la detección [6] y la clasificación [7]. La detección de *malware* es el proceso de identificar si un programa es o no software malicioso y una vez detectado, la clasificación de *malware* se dedica a decidir a qué categoría o familia de *malware* pertenece. Las técnicas empleadas para la clasificación de *malware* serán explicadas en más detalle en secciones futuras.

Es un problema complejo, ya que el software malicioso emplea herramientas para ocultar su presencia y entorpecer su análisis. Las técnicas más comunes [8] incluyen los metamorfismos [9], polimorfismos [10] y la ofuscación de código [11].

Existen técnicas diversas dedicadas a la detección de software malicioso, entre ellas cabe destacar:

- **Análisis de firmas** [12]: basándose en patrones de comportamiento conocidos, identifican si el código es malicioso. Este tipo de técnicas se emplean principalmente en antivirus, manteniendo bases de datos con firmas conocidas que van actualizando.
- **Análisis heurístico** [13]: este tipo de técnicas ayudan a detectar *malware* si no ha sido reconocido por ninguna de las firmas conocidas. No analizará la firma del malware, sino que se centra en análisis estadísticos y de tiempo de ejecución. Esto permite detectar muestras de *malware* previamente desconocidas.
- **Aprendizaje automático** [14]: analiza grandes conjuntos de datos detectando patrones en ellos para tratar de predecir si el software es maligno. Este enfoque es especialmente útil para reconocer muestras de *malware* nuevo y se lleva aplicando durante décadas a la hora de detectar *malware* [15].
- **Análisis de comportamiento** [16]: método que se apoya de técnicas de aprendizaje automático e inteligencia artificial, entre otros, para detectar conductas maliciosas. Observa el comportamiento en tiempo real del código y monitoriza las acciones realizadas por el programa.

Entre las estrategias más utilizadas a la hora de clasificar muestra de *malware* tenemos el Machine Learning y el Deep Learning. Dentro de Machine Learning existen distintos tipos de aprendizaje, que aunque se explicaran en detalle en capítulos posteriores, se mencionarán brevemente a continuación. Los distintos tipos son aprendizaje supervisado, no supervisado, semi-supervisado y reforzado.

El dominio en el que se va a centrar este trabajo es el de la detección de malware, más concretamente empleando técnicas de Machine Learning para el reconocimiento y la clasificación de muestras de software maligno.

1.2. Técnicas de análisis y deteccción

En la actualidad, las técnicas más empleadas para la detección de *malware* se pueden dividir en tres: análisis estático, dinámico e híbrido. Este trabajo hace uso de muestras de código extraídas empleando análisis dinámico.

En el **análisis estático** [17] [18], la muestra de código maligno se estudia sin ser ejecutada, siendo la manera más segura de analizar malware. Principalmente, es empleado para analizar el código binario [19] (o ensamblador) de un programa, generalmente empleado por analistas para buscar indicios y determinar si un programa de origen desconocido es malicioso mediante técnicas como la ingeniería inversa. Este tipo de análisis no solamente examina el código fuente [20] o binario, sino que también emplea otro tipo de técnicas como identificación de patrones conocidos, análisis de dependencias y generación de firmas, entre otros.

Este tipo de análisis trata de comprender la estructura y el comportamiento del software sin ejecutarlo. Sin embargo, esta técnica tiene limitaciones, ya que el *malware* podría implementar técnicas que dificulten su comprensión y camuflen sus intenciones como, por ejemplo, ofuscación de código o técnicas anti-debug.

El **análisis dinámico** [21] ejecuta el código malicioso, idealmente, en entornos aislados, controlados y monitorizados (comúnmente llamados sandbox [22]) para analizar su comportamiento en tiempo real. Este método proporciona más información acerca de las acciones del software bajo análisis, ya que permite detectar, entre otros, cambios en el registro del sistema, en el sistema de ficheros o llamadas al sistema.

Sin embargo, este tipo de análisis se puede ver comprometido si el *malware* implementa técnicas de evasión para detectar la presencia del sandbox y, en ese caso, cambiar su comportamiento o bien finalizar la ejecución directamente, no llegando nunca a ejecutar el comportamiento malicioso. Este tipo de análisis es esencial para detectar variantes del software que quizá hayan empleado técnicas de polimorfismo para modificar su comportamiento en tiempo real y, por lo tanto, dificultar su análisis estático.

El **análisis híbrido** combina técnicas de los análisis estático y dinámico para examinar comportamientos de programas maliciosos. Al integrar ambas técnicas, busca obtener una visión más completa del malware.

Una vez empleado alguno de los análisis mencionados se deben aplicar técnicas de detección para saber si la muestra analizada es o no es maliciosa, las dos principales son la detección basada en firmas y la basada en anomalías.

La **detección basada en firmas** [23] [24] es el enfoque clásico empleado por los antivirus y consiste en generar una firma de la muestra que se está analizando y compararla con las firmas ya presentes en una o varias bases de firmas, que se corresponden con *malware* ya conocido. Estas firmas pueden ser desde el hash del fichero que se está analizando hasta un conjunto de patrones únicos o características específicas asociadas a la muestra que se está analizando.

Este método de detección sigue siendo una herramienta fundamental a la hora de detectar amenazas de manera rápida, pero no es muy efectivo en detectar amenazas desconocidas hasta la fecha. Su combinación con planteamientos más avanzados es empleado para detectar amenazas desconocidas.

A diferencia de la detección basada en firmas, la **detección basada en anomalías** [25] busca encontrar desviaciones en la ejecución normal de un programa. Para conseguirlo se debe conocer previamente cuál es su comportamiento normal y para ello se crean modelos del comportamiento normal del sistema, esta sería la primera fase de la detección. La siguiente fase consiste en la detección y monitorización, en ella se vigila que hace el software y se actúa sobre ellas.

Este enfoque puede detectar nuevas amenazas y ataques de día cero, pero a menudo resulta en una alta tasa de falsos positivos (detecta como *malware* programas que realmente no lo son), ya que se centra en identificar comportamientos inusuales o comportamientos atípicos. Por este motivo hay que ser muy cuidadoso en la configuración del modelo para evitar alertas innecesarias.

1.3. Problema a resolver

El problema que este trabajo busca resolver es la detección y reconocimiento de *malware* empleando trazas de llamadas al sistema, una secuencia ordenada de solicitudes realizadas por un programa a funciones del sistema operativo [26]. Estas contienen las funciones y llamadas al sistema obtenidas durante la ejecución de cada una de las muestras. Las trazas empleadas se obtuvieron del dataset *APIMDS (API-based malware detection system)*, el cual se explicará en más detalle en la sección 3.

Este dataset incluye trazas con su secuencia de llamadas a APIs y al sistema, así como el hash de cada una de ellas y la familia de malware a la que pertenecen. Las APIs son un conjunto de protocolos utilizados en el desarrollo e integración de software en aplicaciones, permitiendo la comunicación entre dos aplicaciones a través del conjunto de protocolos establecido [27].

Para cada una de estas trazas, la recolección de las llamadas a APIs se realizó empleando análisis dinámico en un entorno virtual [28]. Aunque las trazas incluidas en el conjunto de datos seleccionado se hayan conseguido tras emplear análisis dinámico del código, este trabajo realizará un análisis estático de las llamadas al sistema.

Se van a procesar las trazas para extraer n-gramas, sub secuencias de n *ítems* que se superponen entre si. Estos n-gramas representan partes de comportamiento y se utilizarán para detectar patrones comunes entre muestras de la misma familia de *malware*.

Existen muchos tipos de *malware*, pero a continuación se explicarán los tres principales en los que se centra este trabajo. Estos tipos son parte de los presentes en el dataset elegido y la exclusión de los restantes será explicada en capítulos posteriores.

- **Pup** [29] **o programa potencialmente indeseable**: programas que suelen instalarse junto a programas que los usuarios sí quieren instalar. A menudo causan problemas de ralentización, rastreos y saturación del sistema, aunque en algunos casos terminan siendo de utilidad.
- **Trojan** [30] **o Troyano**: a menudo se camufla como software legítimo, pero se emplean para acceder a los sistemas de manera ilegítima y comprometerlos llevando a cabo robos de información mediante el uso de puertas traseras.
- **Worm** [31] **o Gusano**: tipo de software malicioso diseñado para auto-replicarse e infectar el mayor número de dispositivos posible, muchas veces sin necesidad de interacción por parte del usuario.

Todas las APIs que aparecen siguen la especificación Windows API [32] y todas ellas se pueden ver en [33]. Todas ellas se pueden dividir en categorías dependiendo de que funciones lleven a cabo dentro del sistema, estas pueden ser de comunicación entre procesos, almacenamiento, manejo de ficheros, entrada y salida, interfaz de usuario, entre otros, pero no todas estas categorías tienen la misma criticidad dentro de un sistema informático.

Basándonos en su repercusión en el sistema operativo, se han considerado como categorías críticas las de gestión de memoria, registros, ficheros, entrada y salida, comunicaciones, mecanismos de sincronización y de procesos. La gestión efectiva de estos

recursos es esencial para garantizar la estabilidad general de un sistema operativo y su correcta implementación y protección son cruciales para evitar problemas graves que afecten a la vulnerabilidad y la integridad del sistema.

Se emplearán n-gramas calculados a partir de las trazas de APIs para detectar comportamientos comunes entre muestras de la misma familia de *malware* y realizar el entrenamiento de los modelos de clasificación. Finalmente, estos resultados se complementarán con la implementación de un filtro de criticidad para la categoría de cada API que forma el n-grama.

1.4. Objetivos del proyecto

El objetivo principal del trabajo es la detección de comportamiento malicioso en programas a partir de sus trazas de ejecución de sistemas software, aplicando algoritmos de aprendizaje supervisado y buscando los mejores modelos.

Este trabajo busca filtrar y procesar trazas de llamadas al sistema para extraer conocimientos a nivel de comportamiento haciendo uso de n-gramas y de esta manera caracterizar cada una de ellas.

Otro de los objetivos de este proyecto es programar y evaluar modelos de Machine Learning para la detección de *malware* utilizando los resultados obtenidos tras la extracción de n-gramas. Como resultado final, se esperan obtener modelos capaces de clasificar cada muestra de manera correcta y con valores de precisión, recall y f1-score elevados.

Adicionalmente, se extiende la funcionalidad añadiendo un filtrado de criticidad de APIs. Se espera que la mejora implementada mejore los valores obtenidos para todos los clasificadores anteriores, demostrando que realizar filtrados de criticidad de APIs es una buena manera de eliminar datos sin valor del proceso de clasificación.

1.5. Estructura de la memoria

La memoria se ha dividido en los siguientes capítulos:

En la sección *2*, *Detección basado en IA*, se presenta el dominio en el que se va a trabajar en detalle, explicando las técnicas de IA utilizadas en el reconocimiento, entre otros.

La sección *3*, *Modelos para el reconocimiento de malware basado en n-gramas*, está dedicada a explicar el bloque principal del trabajo, desde el dataset empleado hasta los resultados obtenidos.

A continuación, la sección *4*, *Evaluación de mejoras*, describe las mejores realizadas a las primeras versiones de los clasificadores, principalmente las relativas a considerar categorías de interés.

La sección *5*, *Conclusiones y trabajo futuro*, incluye las conclusiones finales del trabajo y posibles trabajos futuros.

En la última sección *6*, *Bibliografía*, se incluyen todos los artículos, webs y documentos referenciados a lo largo de la memoria.

Los anexos incluyen contenido adicional relevante para la memoria, pero que por simplicidad se han separado de la parte principal. El anexo *A*, *Glosario*, incluye una lista con terminología necesaria para la comprensión del contenido de la memoria. En el glosario también se podrá encontrar otra lista con los acrónimos utilizados a lo largo de la memoria.

2. Detección de malware basado en IA

El impacto generado por el *malware* es un problema que lleva existiendo mucho tiempo, pero en los últimos años hemos visto como las amenazas cibernéticas se han vuelto cada vez más frecuentes y como la necesidad de fortalecer nuestra ciberseguridad se ha vuelto imperativa.

El uso de técnicas de Inteligencia Artificial (IA) aplicadas a la Seguridad Informática se está mostrando efectivo [34]. Su uso ha demostrado la mejora de las defensas y en la reducción del tiempo de análisis y detección de amenazas [35].

La IA se presenta como una solución innovadora y prometedora para llevar a cabo trabajos de protección y evaluación de la seguridad de un sistema informático. Estas herramientas pueden simular ataques, detectar vulnerabilidades y aprender de ellos, mejorando de forma continua con el tiempo en su precisión [36].

Una IA es capaz de aprender de los datos y tomar decisiones eficaces y rápidas ante eventos inesperados, por lo que al menos el 40 % de las empresas habrían implementado IA para mejorar su capacidad de detección y respuesta ante ciberdelincuentes [37]. Estas herramientas pueden detectar posibles vulnerabilidades en sistemas informáticos de manera más rápida y eficiente que los métodos tradicionales, transformando la industria de la seguridad informática gracias a su precisión y eficacia [36]. La IA puede ser una muy buena herramienta tanto para identificar vulnerabilidades como para detectar amenazas [38], entre muchas otras aplicaciones posibles.

2.1. Técnicas habituales de Inteligencia Artificial

Las dos principales estrategias de uso de IA para detección de *malware* tienen que ver con el Machine Learning y el Deep Learning. La técnica que se emplea en la realización de este trabajo es la de Machine Learning.

El Machine Learning se centra en el desarrollo de algoritmos y modelos que permiten aprender patrones utilizando técnicas estadísticas para producir clasificaciones, realizar predicciones acerca de los datos y obtener información útil de los mismos. Son más dependientes de la interacción humana para aprender, ya que estos determinan el conjunto de características que se deben de tener en cuenta a la hora de diferenciar los datos y necesitan datos más estructurados para realizar su aprendizaje [39].

La clasificación basada en features se basa en la idea de que muestras con comportamientos maliciosos similares necesitan llamar a las mismas APIs con argumentos similares. El primer paso siempre es ejecutar los programas en un entorno controlado, seguido de un proceso de extracción y selección de características. A continuación, con esas features, se crean modelos de aprendizaje capaces de predecir si una muestra es maliciosa o no. En este trabajo nos centraremos en este tipo de clasificación.

En la sección siguiente se explicarán en más detalle los diferentes tipos de aprendizaje presentes dentro de la rama del Machine Learning.

Los algoritmos de Deep Learning son aplicados a redes neuronales con una estructura en capas formada por tres tipos de capas. Una inicial de entrada, capas ocultas que aplican cálculos matemáticos a los datos y una capa final de salida. Uno de los principales retos al usar este tipo de algoritmos es seleccionar el número adecuado de capas ocultas y de neuronas pertenecientes a cada una de ellas. Las capas iniciales se especializan en identificar detalles simples, mientras que las capas subsiguientes consiguen gradualmente representaciones más complejas al combinar características aprendidas en capas anteriores.

Este tipo de aprendizaje puede ser combinado con otras estrategias para conseguir buenos resultados. Por ejemplo, en [40] se puede ver su uso en conjunto a la representación de cada muestra como una imagen donde las distintas secciones del código se representan con texturas únicas.

Al contrario que los métodos de Machine Learning, los clasificadores de Deep Learning son entrenados mediante aprendizaje de características en lugar de por algoritmos específicos. Esto significa que pueden aprender patrones en lugar de necesitar que una persona le defina los patrones que tiene que buscar en la muestra. Esto resulta en la detección automática de features y clasificación de datos en varias clases [41].

2.2. Técnicas de aprendizaje basado en Machine Learning

Dentro del Machine Learning (ML), según [42] y [43], existen diferentes tipos de aprendizaje:

- **Aprendizaje Supervisado**: tipo de ML que entrena un modelo a partir de datos ya etiquetados. Que un dato esté etiquetado significa que se proporciona la respuesta correcta o la categoría asociada a ese dato. Este tipo de modelos predicen una salida en base al entrenamiento y el hecho de que los datos estén etiquetados permite validar la salida para cada caso. Gracias a este aprendizaje es posible realizar predicciones sobre datos nuevos. Este modelo es muy utilizado en detección de spam [44]. Puede ser categorizado en dos tipos, clasificación y regresión. La clasificación consiste en asignar una etiqueta o categoría a la entrada. Estas etiquetas representan a la totalidad de los datos. El objetivo de la regresión es predecir un valor numérico a partir de unas variables de entrada, buscando una relación funcional entre los datos de entrada y los de salida. Un ejemplo del uso de este tipo de aprendizaje se puede ver en [45].
- **Aprendizaje no Supervisado**: en este tipo de aprendizaje, los modelos son entrenados con datos sin ningún tipo de etiquetado. Estos modelos tratan de obtener información importante de los datos sin conocer con anterioridad su estructura. Existen dos categorías principales dentro del Aprendizaje no Supervisado, el clustering y la reducción de la dimensionalidad. El clustering agrupa un conjunto de datos en subconjuntos con atributos similares. La reducción de la dimensionalidad busca reducir el número de variables o características del conjunto de datos mientras conserva la información relevante con el objetivo de simplificar la representación de los mismos, reduciendo la complejidad computacional de los modelos. Un ejemplo del uso de aprendizaje no supervisado puede verse en [46].

- **Aprendizaje semi-supervisado**: término medio entre los dos tipos anteriores, ya que es entrenado con algunos datos etiquetados y muchos sin etiquetar. En este tipo de aprendizaje el modelo es entrenado sin tener que entrenar un modelo sin tener la totalidad de los datos categorizados. Es especialmente útil cuando la recopilación de datos etiquetados es difícil y se dispone de una gran cantidad de datos no categorizados. El uso de este tipo de aprendizaje se puede apreciar en [47].
- **Aprendizaje Reforzado** [48]: método de entrenamiento basado en recompensar comportamientos mientras penaliza los indeseados. Este modelo no requiere datos de entrenamiento etiquetados, ya que no se conoce de antemano la respuesta correcta. En lugar de ello, depende de un agente de refuerzo, cuya función es evaluar y determinar si la tarea realizada se llevó a cabo de la manera más óptima posible. Este agente toma decisiones secuenciales en un entorno dinámico, aprendiendo a través de la retroalimentación en forma de recompensas o castigos, con el objetivo de maximizar la recompensa acumulativa a lo largo del tiempo. Un ejemplo de la aplicación de aprendizaje reforzado se puede ver en [49].

Este trabajo empleará técnicas de aprendizaje supervisado, ya que se hará uso de un conjunto de datos de trazas de *malware* previamente etiquetados con las familias de *malware* a las que pertenecen.

2.3. Trazas y extracción de n-gramas

Recientemente, el uso de n-gramas se ha convertido en una técnica de uso habitual en el ámbito del estudio de *malware* [50] [51] [52]. Estos estudios presentan resultados prometedores hacia la detección de *malware* haciendo uso de n-gramas.

Esto se debe a que esta técnica permite conocer qué conjunto de n-gramas son representativos, pudiendo así diferenciarlos de otros archivos similares, por lo tanto, permitiendo su clasificación.

El objetivo del uso de n-gramas tiene que ver con analizar las secuencias de operaciones que se realizan de manera conjunta, centrándonos así en comportamientos en vez de en operaciones realizadas de manera individual.

De esta manera es más fácil identificar comportamientos similares dentro de una misma familia de malware, ya que tienden a exhibir comportamientos similares.

El uso de n-gramas junto con IA para la detección de patrones, concretamente su uso con *malware*, comienza con la extracción de una lista de los n-gramas presentes en la muestra maligna. Esta lista es filtrada dependiendo del número o frecuencia de aparición de cada n-grama. Una vez filtrada, esta lista se usa como entrada de los modelos de predicción para que aprendan sobre qué conjuntos de n-gramas son indicios maliciosos.

Este trabajo hace uso de este enfoque, ya las características identificativas de cada familia de malware se calculan haciendo uso de n-gramas. Estas características son empleadas posteriormente para la creación y entrenamiento de modelos para la calcificación de las muestras de *malware*.

2.4. Trabajos similares

En el proceso de preparación para la realización del proyecto se leyó mucha de la literatura acerca de la clasificación de *malware* para entrar al contexto sobre el que se iba a realizar el trabajo. De todos los artículos y bibliografía leída, estos destacan en proximidad.

El primero de ellos es [53]. En este artículo se sigue una estrategia similar a la que se explicara en este documento, ya que también se hace uso de n-gramas para crear features a partir de las que calcular modelos para la clasificación de malware.

Una de las diferencias es el dataset empleado, ya que emplean binarios a partir de los que extraen las APIs. Otra de las similitudes es el empleo de features de frecuencia de n-gramas. Esta similitud no es tan importante, ya que la mayoría de los proyectos de clasificación de *malware* emplean features estadísticas de este tipo.

Otro artículo con características similares es [28]. En este caso, la principal similitud es el uso de features similares, similar al artículo explicado anteriormente. Las features seleccionadas son frecuencias de apariciones de APIs, de APIs en categorías distintas o de número de apariciones totales.

A diferencia del artículo anterior, este no usa n-gramas para extracción de características, pero aplica algoritmos similares a los aplicados en el alineamiento de cadenas de ADN. Este enfoque nos pareció muy novedoso para extraer sub secuencias de patrones en muestras de *malware* de la misma categoría.

En [54] se propone una detección de *malware* mediante la extracción de características comunes entre ellos. Hace uso de n-gramas relevantes, calculados a partir de n-gramas seleccionados por su frecuencia normalizada dentro del documento, y de combinaciones de clasificadores gracias a WEKA, una plataforma software empleada en aprendizaje automático y minería de datos.

El trabajo realizado en [50] también hace uso de trazas de llamadas al sistema a las que posteriormente aplica modelos *bag-of-n-grams* [55]. Este tipo de modelos son comúnmente aplicados en clasificación de documentos, ya que representan textos como un vector de frecuencias de palabras. Otra de las diferencias es el uso de RFE [56], ya que en este trabajo se emplearán tests estadísticos para realizar la función de selección de features.

Otro de los artículos que cabe mencionar es [51]. La principal diferencia es el uso de un método de clasificación y detección de *malware* basado en n-gramas hexadecimales y de bytes. Extraen unigramas de bytes de archivos benignos y malignos, y n-gramas hexadecimales de muestra de *malware*. Las características utilizadas en el proceso de clasificación son la combinación de los unigramas de bytes de bloque y n-gramas hexadecimales.

Por último, en [52] también se hace uso de los n-gramas, aunque de una manera distinta. Cada muestra de *malware* se utiliza para determinar un solo vector de subfamilia al que denominan centroide de familia. Este centroide se construye con los n-gramas que aparecen con mayor frecuencia en la subfamilia

3. Modelos para el reconocimiento de malware basado en n-gramas

En este capítulo se profundiza en la metodología y procesos aplicados en el desarrollo de los modelos. Se explica detalladamente el conjunto de datos utilizado y se describen los sistemas implementados. Finalmente, se presentan los resultados obtenidos, así como una discusión y explicación de los mismos.

3.1. Proceso para programación de modelos de detección

Los pasos seguidos para llevar a cabo el proceso se pueden ver en la Figura 1 y aunque se mencionaran brevemente a continuación con el objetivo de tener una idea general del proceso, se explicaran en más detalle en las siguientes secciones.

El primer paso es encontrar un dataset que se adecue al trabajo que se va a realizar. Una vez seleccionado se comienza el proceso de implementación, empleando el lenguaje de programación Python, de un filtrado del dataset. Este filtrado se lleva a cabo para preparar el dataset a las necesidades específicas del proyecto, eliminando datos que no se van a utilizar.

A continuación se hace uso de una librería de cálculo de n-gramas para obtener los n-gramas más representativos de cada traza y de esa manera crear diccionarios para cada familia de *malware*.

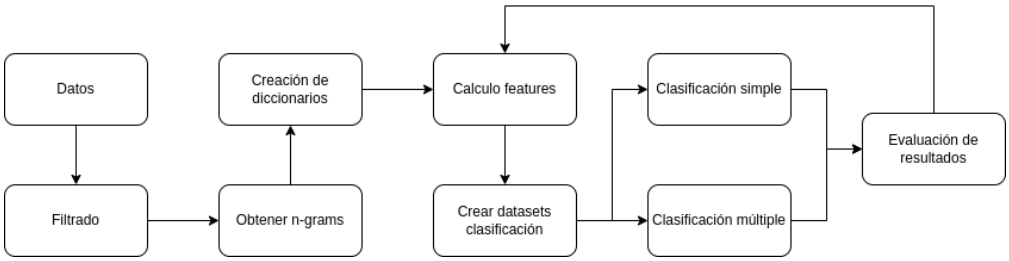

Figura 1: Figura en alto nivel del proceso seguido.

Posteriormente, se calculan las features y se crean los datasets que se emplean para el proceso de detección y clasificación de malware. El proceso de clasificación se lleva a cabo en JupyterLab y se hizo uso de varios tests estadísticos (chi2 [57] y t-test [58]) y distintos tipos de clasificadores (KNN [59], SVM [60], Gradient Boosting [61] y Regresión Logística [62]). En este proceso se hace uso de clasificadores simples y múltiples. Finalmente, se calculan los resultados y métricas para cada uno de ellos y se evalúan. Las siguientes secciones están dedicadas a la explicación en detalle de cada uno de los pasos mencionados.

Una vez analizados se presenta una idea de mejora basada en el filtrado de categorías de interés, el proceso se pueden ver en la Figura 2. Este sistema pretende filtrar el número n-gramas calculados por traza, reduciéndolos y seleccionando solo aquellos que tienen un número de APIs pertenecientes a categorías de interés.

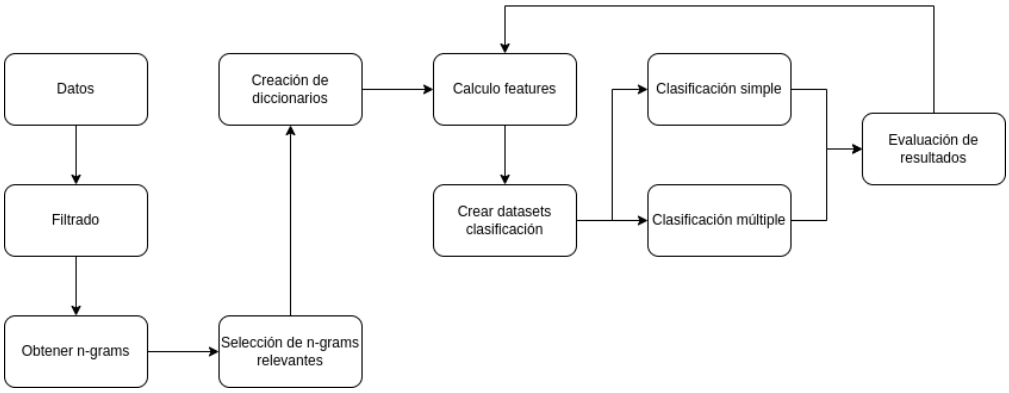

Figura 2: Figura en alto nivel del proceso seguido.

Por último se hace un análisis final de los resultados donde se explicará si han resultado ser favorables y los valores finales para cada clasificador. La sección *4*, *Evaluación de mejoras*, se dedicará a la explicación de la idea de mejora implementada y la discusión de sus resultados.

3.2. Metodología

Antes de comenzar con la explicación del proceso seguido es fundamental hablar de las herramientas que se emplearon en el desarrollo de este trabajo. A continuación se presentarán junto con su aplicación dentro del proyecto.

Python se destaca como el lenguaje de programación principal de este proyecto debido a su facilidad de uso, su gran variedad de librerías especializadas y el conocimiento previo que se tiene del propio lenguaje. Una de las librerías más esenciales en el desarrollo de este proyecto es Scikit-Learn [63], una librería de aprendizaje automático fundamental para desarrollar algoritmos de clasificación y la evaluación de modelos. Será utilizada para entrenar y validar los modelos creados.

Otra de las herramientas fundamentales es JupyterLab, una interfaz de usuario basada en web para Proyecto Jupyter. Proporciona un entorno de desarrollo interactivo para trabajar con Jupyter Notebooks, código y datos [64]. Se va a emplear esta herramienta para el proceso de clasificación de malware, empleando tanto clasificación simple como múltiple.

Para la extracción de información relevante de las trazas de llamadas al sistema se hace uso de la especificación Win32 así como de la librería nltk [65] de n-gramas. Esta librería ofrece recursos para la tokenización y procesamiento del lenguaje natural.

Como se ha mencionado en secciones anteriores, este trabajo realizara un análisis estático de las trazas de llamadas al sistema. Este análisis se lleva a cabo gracias a las herramientas mencionadas.

3.3. Descripción del dataset

El primer paso fue encontrar un dataset adecuado, es decir, que contuviesen muestras de varias familias de *malware* con sus trazas de ejecución y muestras de programas benignos si fuese posible.

Una traza de ejecución describe la secuencia de acciones llevadas a cabo durante la ejecución de un programa. En este contexto específico, nos referimos a trazas de ejecución como la secuencia de llamadas a APIs para cada una de las muestras.

Uno de los primeros datasets que se sopesaron fue *Windows Malware Dataset with PE API Calls* [66]. Este dataset contiene muestras de 8 categorías de malware: Spyware, Downloader, Trojan, Worms, Adware, Dropper, Virus y Backdoor. Contiene un número de trazas similar para la mayoría de ellas, pero no contienen ninguna traza de código benigno.

Otro de los datasets que se tuvieron en cuenta fue el *Microsoft Malware Classification Challenge* [67]. El problema de este conjunto de datos es que solamente contiene los binarios de los ejecutables. Un caso similar es el dataset de *Malware Analysis Datasets: API Call Sequences* [68], ya que contiene trazas de ejecución, pero no aparecen las APIs implicadas en cada llamada, sino un ID identificatorio de cada una de ella, haciendo imposible identificarlas, ya que no se explica.

Por los motivos que se han explicado, ninguno de estos datasets encajaba con los objetivos del trabajo, por ese motivo se decidió utilizar el dataset APIMDS (API-based *malware* detection system) [69]. Este conjunto de datos incluye un total de 23146 trazas de código maligno de 5 categorías: Backdoor, Worm, Packed, PUP, Trojan y Miscelania, así como de subcategorías. La ventaja que tiene es que también contiene los ejecutables de los binarios de código benigno, pudiendo en un futuro poder emplearlos para obtener las trazas de ejecución.

Este conjunto de datos se encuentra en un fichero con formato CSV. El fichero presenta una estructura que consta de una primera columna que contiene el nombre de la clase de *malware* analizado, uno de los mencionados anteriormente, o puede ser que esté la celda vacía, en ese caso, correspondería a la categoría de Miscelania. La segunda columna contiene un hash sha256 y a partir de la tercera columna se encuentra la secuencia de APIs, toda ellas siguiendo el formato Win32.

Como se ha explicado anteriormente, el dataset elegido contiene 5 categorías de malware. Este trabajo nos centraremos en clasificar 3 de ellas. Las dos categorías con las que no se trabajara son Packed y Backdoor. La razón de esta decisión es que ninguna de las dos son un tipo de *malware*. El hecho de que un archivo haya sido empaquetado no indica automáticamente que sea de carácter malicioso. Algo similar pasa con Backdoor, ya que estos se consideran vectores de ataque y no constituyen un tipo específico de malware.

3.4. Sistemas de reconocimiento a programar

En esta sección se explicarán todos los pasos presentados en el apartado anterior en detalle. En la Figura 1 se puede ver un diagrama que pretende servir de apoyo visual a las explicaciones futuras.

3.4.1. Filtrado y preparación de los datos

El primer paso fue trabajar con el dataset y filtrarlo para que se adecuase a las necesidades del proyecto. Para conseguirlo se creó un script de Python para realizar el filtrado de los datos. Queremos obtener trazas organizadas por tipo de *malware* con dos niveles de abstracción: API y categorías de API. Estas abstracciones permiten simplificar la representación de los datos, disminuyendo así su complejidad y de esta forma facilitar la identificación de comportamientos maliciosos.

El objetivo del filtrado es obtener ficheros por familia de *malware*, eliminando el nombre y el hash, dejando presente en la traza solamente las llamadas al sistema. El script crea ficheros individuales en los que aparecen las trazas pertenecientes a la misma familia. Este proceso se realiza recorriendo la colección de datos filtrando cada muestra, aplicando expresiones regulares sobre el hash y el nombre, quedándonos solamente con la lista de llamadas a APIs.

Una vez se tenían las trazas separadas por familia, el siguiente paso fue la extracción de las categorías correspondientes a las que pertenecen cada una de las APIs. Estas categorías hacen referencia al área de funcionalidad en la que se encuentra la API, estas funcionalidades pueden ir desde el control de la exclusión mutua hasta controles del ratón.

Para llevar a cabo este proceso se hizo uso de un fichero [70] JSON donde las claves son cada una de las categorías y los valores son las funciones pertenecientes a esa categoría, tal y como se puede ver en el Listado 1.

Se iteró el fichero JSON, creándose un diccionario donde las APIs actúan como claves y las categorías como valor para optimizar el proceso de realizar consultas. Se crearon ficheros para cada tipo de *malware* con la misma estructura de los generados en el paso anterior, pero en este caso ya no aparecen las APIs sino su categoría.

```
{
  "Dynamic Data Exchange (DDE)": [
      "DdeSetQualityOfService",
      "FreeDDElParam",
      "ImpersonateDdeClientWindow",
      "PackDDElParam",
      "ReuseDDElParam",
      "UnpackDDElParam"
  ],
  "Windows Sockets (Winsock)": [
      "accept",
      "AcceptEx",
      "bind",
  ...
```

Listado 1: Ejemplo del fichero usado para obtener las categorías de cada API.

3.4.2. Reconocedores binarios de 2 a 5 n-gramas

El siguiente paso fue obtener los n-gramas correspondientes a cada traza. Un n-grama es una sub secuencia de 'n' ítems dada una secuencia. Esta 'n' puede ser de distintos tamaños: 1 (unigrama), 2 (bigrama), 3 (trigrama), etc. . En este trabajo se calcularán los resultados para n-gramas de 2 a 5. Muchos estudios han aplicado n-gramas al problema de clasificación de *malware* y se ha demostrado que es una estrategia eficiente para resolver problemas de clasificación [53].

Para poder aplicar n-gramas al proyecto, el primer paso fue encontrar una librería de Python que se adecuase a las necesidades. Entre las librerías que se tuvieron en cuenta se encuentran lpngram [71], python-ngram [72] y nltk [65].

La librería **lpngram** proporciona métodos para la recolección y suavizado de n-gramas, de abstracción de secuencias, para análisis de patrones y es útil para el modelado de lenguajes. La librería **python-ngram** extiende la clase "set", utiliza medidas de similitud entre n-gramas para la comparación de cadenas y es idónea para la búsqueda de similitudes entre ellas, así como de agrupaciones de datos.

Finalmente, contamos con la librería elegida, la librería **nltk**. Esta proporciona amplios recursos para el procesamiento del lenguaje natural, proporciona bibliotecas para la tokenización y clasificación y contiene funcionalidades complejas para el análisis y razonamiento semántico, haciéndola versátil. Así mismo, nltk cuenta con una gran documentación, lo que facilita el aprendizaje y la eficiencia en el desarrollo del proyecto.

Una vez seleccionada la librería se comenzó con el proceso de cálculo de n-gramas, para ello se creó un script de Python encargado de extraer los n-gramas a cada una de las trazas. El proceso de extracción puede verse en el Listado 2, a cada una de las líneas del fichero se les aplica la función *extract ngrams* (Línea 2), esta devuelve la línea separada en n-gramas de tamaño *num*. A estos n-gramas se les aplica la función *FreqDist* (Línea 13) de la librería para obtener la frecuencia de cada uno de los n-gramas en la traza y

seleccionamos los primeros 'n' elementos.

```
# Generar n-grama
def extract_ngrams(data, num):
    n_grams = ngrams(nltk.word_tokenize(data), num)
    return [' '.join(grams) for grams in n_grams]

def create_ngrams(malware_data, fichero, num_ngrams, n):
    fichero = open(fichero, 'w')

    ngrams_malware = []
    for line in malware_data.splitlines():
        lista = extract_ngrams(line, num_ngrams)

        frequency_distribution = FreqDist(lista)
        ngrams = str(frequency_distribution.most_common(n))

        ngrams_malware.append(ngrams)

    # Escribir ngrams en fichero de salida
    for element in ngrams_malware:
        fichero.write(element + "\n")
```

Listado 2: Ejemplo del fichero usado para la extracción de n-gramas.

Una vez calculados los n-gramas por familia, el siguiente paso fue la creación de diccionarios globales para cada familia de malware, donde la clave es cada n-grama y su valor es su número de repeticiones. El porqué del uso de diccionarios proviene de la eficiencia que proporcionan en la búsqueda de un n-grama en específico durante la fase de creación y actualización del diccionario. Proveen una estructura de datos organizada y eficiente para su posterior uso en la creación de modelos.

Utilizando expresiones regulares se extraen los n-gramas y el número de repeticiones de cada uno de ellos. Como se van a calcular diccionarios globales a partir de muchas trazas, puede ser que varios n-gramas coincidan entre ellas. Para solucionarlo, si un n-grama ya ha sido añadido al diccionario, simplemente se actualizará el valor al número de repeticiones existente.

Aprovechando que se tiene el número total de repeticiones por n-grama, se aplicó un paso más de filtrado, calculándose el porcentaje de apariciones de cada uno de ellos y eliminando los que no superen un valor mínimo de aparición. Los restantes se ordenan por porcentaje de aparición, solamente quedándonos con los 'n' primeros, dándoles más importancia a los que más aparecen. Tanto el valor 'n' como el valor mínimo de porcentaje de aparición son introducidos por el usuario, siendo estos valores completamente personalizables.

Como el tamaño del dataset es grande, el valor seleccionado para el porcentaje mínimo de aparición ha sido de 0.05, ya que escogiendo valores mayores el tamaño del diccionario

resultante se reducía considerablemente. Esto puede deberse a que en un conjunto de datos grande, es común que aparezcan n-gramas únicos, pero muchos de ellos contener frecuencias bajas. Filtrando por el umbral de 0.05 nos centramos en los n-gramas con una mayor presencia dentro del dataset.

Como paso final se realizó la automatización de todo el proceso, calculándose todos los resultados para n-gramas de 2 a 5 con longitud máxima 100 y el porcentaje de aparición previamente explicado de 0.05. Las razones por las que se seleccionó 100 como valor máximo tienen que ver conque cuando el valor aplicado a la longitud máxima era mayor se obtenían gran cantidad de n-gramas con frecuencias de aparición muy bajas. Así mismo, en artículos como en [53] también reducen la lista a los 100 primeros. Todos estos parámetros son introducidos por el usuario y se calculan de manera automatizada.

3.4.3. Extracción de features

El objetivo de este paso es obtener un conjunto de datos representativo y útil para su uso en la creación de modelos de aprendizaje. Una buena selección de features es crucial para mejorar la precisión de los modelos.

El enfoque de la selección de features se basa en un análisis de la literatura científica relacionada, así como de una selección personal previa de características que se consideraban de relevancia. Se han incorporado conceptos respaldados por varios estudios en el campo, los cuales se mencionarán a continuación.

Las features empleadas en [73] incluyen el porcentaje de aparición de categorías de cada API presente en las trazas. En [53] podemos ver el uso de la frecuencia de cada n-grama como feature y como se reduce la lista total a los 100 primeros. También emplean las frecuencias de cada llamada. Finalmente, en [28] emplean el número total de llamadas a cada API junto a su frecuencia de uso.

Tras esta revisión, la selección de features adoptada en este trabajo se centra en el recuento total de repeticiones para cada n-grama en cada una de las trazas. Esta elección se basa en las observaciones anteriores, ya que el uso de porcentajes o frecuencias, así como el número de apariciones de APIs han sido empleados y han proporcionado resultados significativos en investigaciones similares.

Como salida generada en este paso se obtiene un fichero por familia de *malware* que contiene una línea por traza en la que aparecen el número de repeticiones de cada n-grama.

3.4.4. Creación de datasets para clasificación

Para iniciar el proceso de clasificación de las muestras y desde este punto en adelante nos sumergimos en la plataforma JupyterLab para llevar a cabo todos los pasos siguientes.

El primer paso fue obtener las features calculadas para cada familia de *malware* en dos partes, una de *train* dedicada a realizar el entrenamiento de los modelos y otra de *test* o validación, destinada a comprobar la eficacia del modelo. Este proceso se realiza

separando el conjunto de datos gracias a un script de Python en el que se introduce el porcentaje de elementos presentes en el conjunto de *train*. En el caso de este proyecto se utiliza un 80 % para datos de entrenamiento y un 20 % para datos de validación. Usando estos elementos se crearon datasets para cada familia de malware.

La primera columna de todos los datasets indica el tipo de *malware* al que corresponde la traza, los tipos siempre serán *other* o uno de los tipos de malware. El resto corresponde el número de repeticiones de cada n-gram. Los datasets contienen un número de trazas equilibrado, esto significa que hay el mismo número de trazas de *other* como del otro tipo y todas tienen la misma longitud. Las trazas clasificadas como *other* no solamente provienen de un tipo de *malware* sino de una mezcla de todos los tipos, excluyendo claramente la categoría de *malware* sobre la que se está creando el dataset.

Una vez obtenidos los datasets, se comenzó con la creación de los modelos.

3.4.5. Programación de modelos

El siguiente paso es la creación de modelos. Este paso consta de varias fases: aplicar tests estadísticos, seleccionar las mejores features, aplicar clasificadores y realizar predicciones. Los modelos creados en este trabajo se construyen mediante clasificadores simples y múltiples. En este apartado se explicarán los clasificadores aplicados, explicando los valores de entrada y salida, así como su comportamiento.

El primer paso es aplicar tests estadísticos. Su objetivo es determinar la eficacia con la que las features describen el conjunto de datos, es decir, funcionan como una hipótesis. Los tests aplicados son el chi2 [57] y el t-test [58], los cuales devuelven las features que mejor describen el conjunto de datos. Una vez obtenidos sus resultados se selecciona las mejores features devueltas por ambos para ser pasadas a los clasificadores. La manera en la que se calculan las mejores features es aplicando estrategias de voting, donde las features devueltas por los tests estadísticos son unificadas seleccionando las mejores de cada uno de ellos.

El siguiente paso consiste en la creación de modelos a partir de diferentes algoritmos de ML. Los algoritmos seleccionados han sido: KNN, SVM, Gradient Boosting y Regresión Logística. Se han seleccionado estos cuatro algoritmos por su diversidad en el enfoque que utilizan y su capacidad para trabajar con diferentes tipos de datos.

- **KNN**: utiliza la proximidad para hacer clasificaciones o predicciones sobre la agrupación de datos [59]. El algoritmo KNN funciona considerando puntos similares cercanos en un espacio de datos, donde la K hace referencia al número de vecinos que se tienen en cuenta. Cuando este algoritmo realiza predicciones, mira a sus vecinos cercanos empleando una medida de distancia entre puntos, como por ejemplo la distancia euclídea.
- **SVM**: encuentra la mejor línea o hiperplano que separe de la mejor forma posible dos clases diferentes de datos [60]. Busca maximizar la distancia entre los puntos más cercanos de las distintas clases. Une vez encontrada la línea o el hiperplano, clasifica los nuevos puntos dependiendo del lado en el que caigan.

- **Gradient Boosting**: técnica de aprendizaje automático utilizada para el análisis de la regresión que produce un modelo predictivo uniendo modelos de predicción débiles [61]. Comienza con un modelo simple y se centra en corregir sus errores agregando modelos débiles. Cada modelo tiene un peso según su capacidad para corregir dichos errores. Estos modelos mejoran gradualmente la precisión del modelo.
- **Regresión Logística**: modelo estadístico que estudia las relaciones entre un conjunto de variables cuantitativas y una variable cualitativa [62] para predecir la probabilidad de pertenencia a una categoría. Ajusta sus parámetros para maximizar dicha probabilidad y establece un umbral de decisión para la clasificación.

Para todos los modelos y clasificadores el entrenamiento se realiza empleando los datos de train y se realizan predicciones con los datos de test, calculándose métricas para comprobar el comportamiento del modelo. Un ejemplo de este proceso se puede ver en el Listado 3.

Sin embargo, para mejorar la robustez de la evaluación también se aplica K-Fold Cross Validation con un valor de k de 5, ya que proporciona un buen equilibrio entre varianza y sesgo de estimación del rendimiento [74] del modelo. Este método permite evaluar los modelos en múltiples divisiones para obtener la más confiable.

La función salida es la clase en la que se ha clasificado cada una de las muestras. En el caso de los clasificadores simples solamente pueden ser dos valores (*other* y la clase de malware con la que se entrena el modelo) y en los clasificadores múltiples pueden ser cuatro (*other*, *trojan*, *pup* o *worm*).

Las métricas calculadas para la evaluación del rendimiento de los modelos son precision, recall, f1-score y matrices de confusión [75]. Todos estos valores se guardaron en ficheros CSV para que su posterior análisis fuese rápido. En estas tablas se puede ver el número de n-gramas que se ha empleado, el modelo de ML utilizado y el número de trazas con las que se ha realizado la creación del modelo. Así mismo también aparecen los valores para cada una de las métricas calculadas.

```
# Crear modelo de KNN
model = KNeighborsClassifier()

# Entrenar con datos de train
model.fit(X_train, y_train)

# Realizar predicciones
y_predict = model.predict(X_test)
```

Listado 3: Ejemplo de creación, entrenamiento y realización de predicciones a modelos.

3.4.6. Clasificadores simples

A continuación se muestran los resultados obtenidos tras la clasificación y el uso de los clasificadores simples de KNN, SVM, Gradient Boosting y Regresión Logística para cada una de las categorías de *malware* presentes en el dataset. Aunque se calcularon resultados para n-gramas desde 2 hasta 5, nos enfocaremos en aquellos que arrojaron mejores resultados. Específicamente, los trigramas y 4-gramas fueron los que ofrecieron mejores resultados.

Las Tablas 1, 2 y 3 contienen los resultados obtenidos con trigramas y las Figuras 3, 4 y 5 presentan las matrices de confusión para *pup*, *trojan* y *worm* respectivamente.

N-grams	Modelo ML	Num. traza	precision	recall	f1	support	mcc
3	KNN	2433	0.357	0.331	0.344	465	-0.214
3	svm	2433	0.5	0.865	0.634	465	0.098
3	gradient	2433	0.438	0.572	0.496	465	-0.101
3	regression	2433	0.516	0.731	0.605	465	0.112

Tabla 1: Resultados obtenidos para pup con trigramas.

N-grams	Modelo ML	Num. traza	precision	recall	f1	support	mcc
3	KNN	7314	0.678	0.825	0.744	1408	-0.011
3	svm	7314	0.754	0.765	0.759	1408	0.236
3	gradient	7314	0.646	0.546	0.592	1408	-0.084
3	regression	7314	0.722	0.695	0.708	1408	0.124

Tabla 2: Resultados obtenidos para trojan con trigramas.

N-grams	Modelo ML	Num. traza	precision	recall	f1	support	mcc
3	KNN	605	0.558	0.653	0.602	118	0.162
3	svm	605	0.361	0.559	0.439	118	-0.447
3	gradient	605	0.49	1	0.657	118	0.063
3	regression	605	0.375	0.585	0.457	118	-0.401

Tabla 3: Resultados obtenidos para worm con trigramas.

```
Elemento pup con clasificador KNN:

              precision    recall  f1-score   support

       other       0.43      0.46      0.44       509
         pup       0.36      0.33      0.34       465

    accuracy                           0.40       974
   macro avg       0.39      0.39      0.39       974
weighted avg       0.39      0.40      0.39       974

Matriz de confusión:
------------------------------
[[232 277]
 [311 154]]

Elemento pup con clasificador svm:

              precision    recall  f1-score   support

       other       0.63      0.21      0.32       509
         pup       0.50      0.86      0.63       465

    accuracy                           0.52       974
   macro avg       0.56      0.54      0.47       974
weighted avg       0.57      0.52      0.47       974

Matriz de confusión:
------------------------------
[[107 402]
 [ 63 402]]

Elemento pup con clasificador gradient:

              precision    recall  f1-score   support

       other       0.46      0.33      0.38       509
         pup       0.44      0.57      0.50       465

    accuracy                           0.45       974
   macro avg       0.45      0.45      0.44       974
weighted avg       0.45      0.45      0.44       974

Matriz de confusión:
------------------------------
[[168 341]
 [199 266]]

Elemento pup con clasificador regression:

              precision    recall  f1-score   support

       other       0.60      0.37      0.46       509
         pup       0.52      0.73      0.60       465

    accuracy                           0.54       974
   macro avg       0.56      0.55      0.53       974
weighted avg       0.56      0.54      0.53       974

Matriz de confusión:
------------------------------
[[190 319]
 [125 340]]
```

Figura 3: Matrices de confusión obtenidas con los clasificadores simples para pup empleando trigramas.

```
Elemento trojan con clasificador KNN:

              precision    recall  f1-score   support

       other       0.31      0.17      0.22       663
      trojan       0.68      0.83      0.74      1408

    accuracy                           0.61      2071
   macro avg       0.49      0.50      0.48      2071
weighted avg       0.56      0.61      0.58      2071

Matriz de confusión:
--------------------------------
[[ 110  553]
 [ 246 1162]]

Elemento trojan con clasificador svm:

              precision    recall  f1-score   support

       other       0.48      0.47      0.48       663
      trojan       0.75      0.76      0.76      1408

    accuracy                           0.67      2071
   macro avg       0.62      0.62      0.62      2071
weighted avg       0.67      0.67      0.67      2071

Matriz de confusión:
--------------------------------
[[ 311  352]
 [ 331 1077]]

Elemento trojan con clasificador gradient:

              precision    recall  f1-score   support

       other       0.27      0.37      0.31       663
      trojan       0.65      0.55      0.59      1408

    accuracy                           0.49      2071
   macro avg       0.46      0.46      0.45      2071
weighted avg       0.53      0.49      0.50      2071

Matriz de confusión:
--------------------------------
[[242 421]
 [639 769]]

Elemento trojan con clasificador regression:

              precision    recall  f1-score   support

       other       0.40      0.43      0.42       663
      trojan       0.72      0.70      0.71      1408

    accuracy                           0.61      2071
   macro avg       0.56      0.56      0.56      2071
weighted avg       0.62      0.61      0.61      2071

Matriz de confusión:
--------------------------------
[[286 377]
 [429 979]]
```

Figura 4: Matrices de confusión obtenidas con los clasificadores simples para trojan empleando trigramas.

```
Elemento worm con clasificador KNN:

              precision    recall  f1-score   support

       other       0.61      0.51      0.55       124
        worm       0.56      0.65      0.60       118

    accuracy                           0.58       242
   macro avg       0.58      0.58      0.58       242
weighted avg       0.58      0.58      0.58       242

Matriz de confusión:
------------------------------
[[63 61]
 [41 77]]

Elemento worm con clasificador svm:

              precision    recall  f1-score   support

       other       0.12      0.06      0.08       124
        worm       0.36      0.56      0.44       118

    accuracy                           0.30       242
   macro avg       0.24      0.31      0.26       242
weighted avg       0.24      0.30      0.25       242

Matriz de confusión:
------------------------------
[[  7 117]
 [ 52  66]]

Elemento worm con clasificador gradient:

              precision    recall  f1-score   support

       other       1.00      0.01      0.02       124
        worm       0.49      1.00      0.66       118

    accuracy                           0.49       242
   macro avg       0.74      0.50      0.34       242
weighted avg       0.75      0.49      0.33       242

Matriz de confusión:
------------------------------
[[  1 123]
 [  0 118]]

Elemento worm con clasificador regression:

              precision    recall  f1-score   support

       other       0.16      0.07      0.10       124
        worm       0.38      0.58      0.46       118

    accuracy                           0.32       242
   macro avg       0.27      0.33      0.28       242
weighted avg       0.26      0.32      0.27       242

Matriz de confusión:
------------------------------
[[  9 115]
 [ 49  69]]
```

Figura 5: Matrices de confusión obtenidas con los clasificadores simples para worm empleando trigramas.

También se incluyen las Tablas 4, 5 y 6 que contienen los resultados obtenidos con 4-gramas y las Figuras 6, 7 y 8 que presentan las matrices de confusión para *pup*, *trojan* y *worm* respectivamente.

N-grams	Modelo ML	Num. traza	precision	recall	f1	support	mcc
4	KNN	2433	0.415	0.645	0.505	468	-0.226
4	svm	2433	0.297	0.111	0.162	468	-0.172
4	gradient	2433	0.464	0.562	0.508	468	-0.039
4	regression	2433	0.27	0.128	0.174	468	-0.229

Tabla 4: Resultados obtenidos para pup con n-gramas de 4.

N-grams	Modelo ML	Num. traza	precision	recall	f1	support	mcc
4	KNN	7314	0.706	0.505	0.589	1474	-0.013
4	svm	7314	0.704	0.904	0.792	1474	-0.054
4	gradient	7314	0.624	0.361	0.457	1474	-0.163
4	regression	7314	0.707	0.943	0.808	1474	-0.041

Tabla 5: Resultados obtenidos para trojan con n-gramas de 4.

N-grams	Modelo ML	Num. traza	precision	recall	f1	support	mcc
4	KNN	605	0.527	0.556	0.541	124	0.031
4	svm	605	0.479	0.452	0.465	124	-0.065
4	gradient	605	0.403	0.452	0.426	124	-0.255
4	regression	605	0.495	0.444	0.468	124	-0.031

Tabla 6: Resultados obtenidos para worm con n-gramas de 4.

```
Elemento pup con clasificador KNN:

              precision    recall  f1-score   support

       other       0.33      0.16      0.21       506
         pup       0.41      0.65      0.51       468

    accuracy                           0.39       974
   macro avg       0.37      0.40      0.36       974
weighted avg       0.37      0.39      0.35       974

Matriz de confusión:
------------------------------
[[ 80 426]
 [166 302]]

Elemento pup con clasificador svm:

              precision    recall  f1-score   support

       other       0.48      0.76      0.59       506
         pup       0.30      0.11      0.16       468

    accuracy                           0.45       974
   macro avg       0.39      0.43      0.37       974
weighted avg       0.39      0.45      0.38       974

Matriz de confusión:
------------------------------
[[383 123]
 [416  52]]

Elemento pup con clasificador gradient:

              precision    recall  f1-score   support

       other       0.50      0.40      0.44       506
         pup       0.46      0.56      0.51       468

    accuracy                           0.48       974
   macro avg       0.48      0.48      0.48       974
weighted avg       0.48      0.48      0.47       974

Matriz de confusión:
------------------------------
[[202 304]
 [205 263]]

Elemento pup con clasificador regression:

              precision    recall  f1-score   support

       other       0.46      0.68      0.55       506
         pup       0.27      0.13      0.17       468

    accuracy                           0.41       974
   macro avg       0.36      0.40      0.36       974
weighted avg       0.37      0.41      0.37       974

Matriz de confusión:
------------------------------
[[344 162]
 [408  60]]
```

Figura 6: Matrices de confusión obtenidas con los clasificadores simples para pup empleando 4-gramas.

```
Elemento trojan con clasificador KNN:

              precision    recall  f1-score   support

       other       0.28      0.48      0.36       597
      trojan       0.71      0.50      0.59      1474

    accuracy                           0.50      2071
   macro avg       0.49      0.49      0.47      2071
weighted avg       0.58      0.50      0.52      2071

Matriz de confusión:
------------------------------
[[287 310]
 [730 744]]

Elemento trojan con clasificador svm:

              precision    recall  f1-score   support

       other       0.21      0.06      0.10       597
      trojan       0.70      0.90      0.79      1474

    accuracy                           0.66      2071
   macro avg       0.46      0.48      0.44      2071
weighted avg       0.56      0.66      0.59      2071

Matriz de confusión:
------------------------------
[[  37  560]
 [ 141 1333]]

Elemento trojan con clasificador gradient:

              precision    recall  f1-score   support

       other       0.23      0.46      0.30       597
      trojan       0.62      0.36      0.46      1474

    accuracy                           0.39      2071
   macro avg       0.43      0.41      0.38      2071
weighted avg       0.51      0.39      0.41      2071

Matriz de confusión:
------------------------------
[[276 321]
 [942 532]]

Elemento trojan con clasificador regression:

              precision    recall  f1-score   support

       other       0.21      0.04      0.06       597
      trojan       0.71      0.94      0.81      1474

    accuracy                           0.68      2071
   macro avg       0.46      0.49      0.44      2071
weighted avg       0.56      0.68      0.59      2071

Matriz de confusión:
------------------------------
[[  22  575]
 [  84 1390]]
```

Figura 7: Matrices de confusión obtenidas con los clasificadores simples para trojan empleando 4-gramas.

```
Elemento worm con clasificador KNN:

              precision    recall  f1-score   support

       other       0.50      0.47      0.49       118
        worm       0.53      0.56      0.54       124

    accuracy                           0.52       242
   macro avg       0.52      0.52      0.52       242
weighted avg       0.52      0.52      0.52       242

Matriz de confusión:
------------------------------
[[56 62]
 [55 69]]

Elemento worm con clasificador svm:

              precision    recall  f1-score   support

       other       0.46      0.48      0.47       118
        worm       0.48      0.45      0.46       124

    accuracy                           0.47       242
   macro avg       0.47      0.47      0.47       242
weighted avg       0.47      0.47      0.47       242

Matriz de confusión:
------------------------------
[[57 61]
 [68 56]]

Elemento worm con clasificador gradient:

              precision    recall  f1-score   support

       other       0.34      0.30      0.32       118
        worm       0.40      0.45      0.43       124

    accuracy                           0.38       242
   macro avg       0.37      0.37      0.37       242
weighted avg       0.37      0.38      0.37       242

Matriz de confusión:
------------------------------
[[35 83]
 [68 56]]

Elemento worm con clasificador regression:

              precision    recall  f1-score   support

       other       0.47      0.53      0.50       118
        worm       0.50      0.44      0.47       124

    accuracy                           0.48       242
   macro avg       0.48      0.48      0.48       242
weighted avg       0.48      0.48      0.48       242

Matriz de confusión:
------------------------------
[[62 56]
 [69 55]]
```

Figura 8: Matrices de confusión obtenidas con los clasificadores simples para worm empleando 4-gramas.

Como se puede apreciar en las tablas, los trigramas consiguen mejores resultados para pup con SVM con f1-score de 0.634 (Tabla 1) y worm con Gradient Boosting con un f1-score de 0.657 (Tabla 3).

En cambio, el mejor modelo para trojan es conseguido empleando 4-grams y Regresión Logística dando un f1-score de 0.808 (Tabla 5). En todos ellos el mejor clasificador varía.

El mejor modelo generado es el de *trojan* utilizando Regresión Logística y 4-gramas (ver Figura 7). En el resultado podemos observar que la clasificación de *trojan* es muy efectiva, con un f1-score de 0.81, en cambio, la clasificación de *other* devuelve un f1-score de 0.06, dejando el valor medio en 0.59. Esto puede deberse a que el número de muestras de *trojan* en comparación con su opuesto es mucho mayor y no contiene solamente muestras de un tipo de malware.

La matriz nos muestra que 1390 muestras han sido categorizadas de manera correcta como *trojan* y solamente 84 han sido mal clasificadas. En el caso de *other* podemos ver que 575 muestras han sido mal clasificadas y solamente 22 han sido identificadas como *other*.

3.4.7. Clasificadores múltiples

Una vez que se obtuvieron clasificadores simples que funcionaban correctamente, se procedió a la implementación de clasificadores múltiples.

El dataset empleado varía un poco de los empleados en los clasificadores simples, ya que en ellos solo existían dos tipos de clases a clasificar, en cambio, en este dataset están presentes todas las categorías de *malware* y *other* que incluye el resto de muestras. Este dataset también contiene un número de muestras balanceado. En cuanto al proceso seguido, este no difiere del empleado en los clasificadores simples.

Como se podrá ver en los resultados obtenidos, no se calculó el resultado para Regresión Logística, ya que el proceso de clasificación era el más largo y JupyterLab cortaba la ejecución. Esto mismo pasaba cuando la longitud de la traza introducida era mayor a 100, de ahí una de las razones de seleccionar ese valor como longitud máxima de cada traza.

La organización de los resultados está organizada de la misma manera que en apartado anterior. La Tabla 7 contiene los resultados obtenidos con trigramas y las Figuras 9, 10 y 11 presentan las matrices de confusión obtenidas para cada clasificador.

N-grams	Modelo ML	precision	recall	f1	support	mcc
3	KNN	0.758	0.782	0.762	2290	0.577
3	svm	0.712	0.737	0.713	2290	0.476
3	gradient	0.78	0.794	0.77	2290	0.591

Tabla 7: Resultados obtenidos para el clasificador múltiple usando trigramas.

También se incluye la Tabla 8 que muestra los resultados obtenidos con 4-gramas y las Figuras 12, 13 y 14 presentan las matrices de confusión obtenidas para cada clasificador..

```
Clasificador KNN:

              precision    recall  f1-score   support

       other       0.43      0.18      0.26       225
         pup       0.69      0.81      0.75       479
      trojan       0.84      0.90      0.87      1458
        worm       0.61      0.35      0.45       128

    accuracy                           0.78      2290
   macro avg       0.64      0.56      0.58      2290
weighted avg       0.76      0.78      0.76      2290

Matriz de confusión:
-------------------------------
[[  41   66  115    3]
 [  12  387   73    7]
 [  38   83 1318   19]
 [   4   23   56   45]]
```

Figura 9: Matriz de confusión obtenida por el clasificador múltiple empleando KNN para trigramas.

```
Clasificador svm:

              precision    recall  f1-score   support

       other       0.46      0.16      0.24       225
         pup       0.62      0.67      0.64       479
      trojan       0.80      0.89      0.84      1458
        worm       0.56      0.28      0.38       128

    accuracy                           0.74      2290
   macro avg       0.61      0.50      0.52      2290
weighted avg       0.71      0.74      0.71      2290

Matriz de confusión:
-------------------------------
[[  36   60  125    4]
 [   6  319  149    5]
 [  35  107 1297   19]
 [   2   32   58   36]]
```

Figura 10: Matriz de confusión obtenida por el clasificador múltiple empleando SVM para trigramas.

```
Clasificador gradient:

              precision    recall  f1-score   support

       other       0.65      0.21      0.32       225
         pup       0.73      0.79      0.76       479
      trojan       0.82      0.93      0.87      1458
        worm       0.71      0.32      0.44       128

    accuracy                           0.79      2290
   macro avg       0.73      0.56      0.60      2290
weighted avg       0.78      0.79      0.77      2290

Matriz de confusión:
------------------------------
[[  47   58  120    0]
 [   3  377   99    0]
 [  21   67 1353   17]
 [   1   13   73   41]]
```

Figura 11: Matriz de confusión obtenida por el clasificador múltiple empleando Gradient Boosting para trigramas.

n-grams	Modelo ML	precision	recall	f1	support	mcc
4	KNN	0.769	0.789	0.772	2290	0.579
4	svm	0.727	0.751	0.726	2290	0.489
4	gradient	0.77	0.791	0.764	2290	0.575

Tabla 8: Resultados obtenidos para el clasificador múltiple usando n-gramas de 4.

Los resultados obtenidos por los clasificadores múltiples también se obtuvieron con trigramas y 4-grams. Como se puede ver en las tablas, los 4-grams consiguen el mejor valor al emplear KNN, ya que devuelve un valor de f1-score de 0.772 (Tabla 8), este valor no tiene gran diferencia con el obtenido con los trigramas, ya que el mejor clasificador, el Gradient Boosting, devuelve un valor de 0.77 (Tabla 7). Tanto en trigramas como en 4-gramas, el clasificador que peores resultados ha devuelto ha sido el SVM.

El mejor modelo fue conseguido tras aplicar KNN (ver Figura 12). Al analizar este resultado podemos ver que el tipo de *malware* que mejor se clasifica es *trojan* con un f1-score de 0.87, seguido de *pup* con 0.74, *worm* con 0.49 y finalmente *other* con 0.28, dejando el valor medio en 0.77. Este orden sigue el del número de trazas disponibles para cada una de las categorías de *malware*.

Otro elemento a tener en cuenta son los valores de la matriz de confusión. Podemos ver que en todos los casos, excepto en el de *other*, la mayor parte de las muestras han sido clasificadas de manera correcta. Podemos ver que 44 muestras han sido clasificadas de manera correcta como *other* mientras que 116 han sido identificadas erróneamente como *trojan*. En el caso de *pup* 370 han sido clasificadas correctamente y 81 como *trojan*. El número de muestras correctamente clasificadas como *trojan* ha sido 1343, mientras que 87 han sido clasificadas como *pup*. Finalmente, 49 muestras de *worm* han sido bien clasificadas, mientras que 40 han sido clasificadas como *trojan*.

```
Clasificador KNN:

              precision    recall  f1-score   support

       other       0.46      0.20      0.28       216
         pup       0.70      0.79      0.74       466
      trojan       0.85      0.90      0.87      1491
        worm       0.59      0.42      0.49       117

    accuracy                           0.79      2290
   macro avg       0.65      0.58      0.60      2290
weighted avg       0.77      0.79      0.77      2290

Matriz de confusión:
--------------------------------
[[  44   54  116    2]
 [   6  370   81    9]
 [  38   87 1343   23]
 [   7   21   40   49]]
```

Figura 12: Matriz de confusión obtenida por el clasificador múltiple empleando KNN para 4-gramas.

```
Clasificador svm:

              precision    recall  f1-score   support

       other       0.49      0.12      0.20       216
         pup       0.63      0.68      0.65       466
      trojan       0.81      0.90      0.85      1491
        worm       0.55      0.33      0.41       117

    accuracy                           0.75      2290
   macro avg       0.62      0.51      0.53      2290
weighted avg       0.73      0.75      0.73      2290

Matriz de confusión:
--------------------------------
[[  27   53  132    4]
 [   4  317  141    4]
 [  22  108 1337   24]
 [   2   26   50   39]]
```

Figura 13: Matriz de confusión obtenida por el clasificador múltiple empleando SVM para 4-gramas.

```
Clasificador gradient:

              precision    recall  f1-score   support

       other       0.56      0.12      0.20       216
         pup       0.72      0.79      0.75       466
      trojan       0.83      0.92      0.87      1491
        worm       0.62      0.38      0.47       117

    accuracy                           0.79      2290
   macro avg       0.68      0.55      0.57      2290
weighted avg       0.77      0.79      0.76      2290

Matriz de confusión:
--------------------------------
[[  27   52  134    3]
 [   1  368   97    0]
 [  19   75 1373   24]
 [   1   18   54   44]]
```

Figura 14: Matriz de confusión obtenida por el clasificador múltiple empleando Gradient Boosting para 4-gramas.

Las muestras no identificadas correctamente como *trojan* han sido confundidas, en su mayoría, con *pup*. Esto también pasa con las muestras de *pup*, *worm* y *other*, mal clasificadas en su mayoría como *trojan*.

3.5. Discusión sobre los resultados

En esta sección se aborda la comparación entre los resultados obtenidos por los clasificadores simples y los clasificadores múltiples. El análisis se centra en evaluar la eficacia de ambos enfoques para identificar la aproximación más efectiva para la clasificación de *malware*.

Tras haber analizado todos los resultados, se puede ver que los resultados obtenidos por los clasificadores múltiples superan a los obtenidos por los clasificadores simples, exceptuando el caso de *trojan* empleando Regresión Logística, ya que devuelve un f1-score de 0.808, mayor que los resultados devueltos por cualquier clasificador múltiple.

La aproximación que se muestra más efectiva es la utilización de clasificación múltiple. En general, los resultados obtenidos superan a los devueltos por los clasificadores simples, indicando una mayor capacidad para manejar la diversidad de las categorías de *malware* que presenta el dataset. Casos como el clasificador simple de *trojan* mencionado, que devuelve un resultado que mejora los obtenidos por los clasificadores múltiples, necesitan de un análisis detallado, ya que puede ser que empleando Regresión Logística y clasificación múltiple en otro entorno se obtengan aún mejores resultados.

Tras analizar la efectividad de los clasificadores, se sugiere la posibilidad de mejorar enfoques mejores al actual. La estrategia decidida es la implementación de un filtrado más específico de los n-gramas.

4. Evaluación de mejoras

Una vez ya analizados los resultados se decidió intentar mejorarlo. La opción por la que se optó fue realizar un filtrado de los n-gramas basado en categorías de interés. Centrarse en categorías que sean más vulnerables a ser relacionadas con comportamientos maliciosos puede tener un impacto significativo en la precisión de los modelos.

4.1. Filtrado basado en categorías de interés

El filtrado basado en categorías de interés busca mejorar los resultados obtenidos por los clasificadores. Para conseguirlo se propuso un filtrado de los n-gramas para quedarnos y calcular features a partir de una selección de n-gramas que se consideren relevantes.

Los paso seguidos para la implementación de esta mejora se pueden ver en la Figura 15. Como se puede ver se ha añadido un paso adicional llamado "Selección de n-gramas relevantes" pero el resto del proceso no ha tenido modificaciones.

El primer paso a realizar para llevar a cabo el filtrado fue hacer una selección de categorías relevantes. Se recopilaron una serie de categorías de APIs más asociadas con comportamientos maliciosos o que su uso hiciese más vulnerable el sistema informático. Todas ellas se obtuvieron del fichero listado en el Listado 1, ya explicado anteriormente. Estas categorías son: Files and I/O (Local file system), Cryptography, Cryptographic Next Generation (CNG), CNG Cryptographic Primitive, Network Management, Windows Networking (WNet), Windows Internet (WinINet), Windows Sockets (Winsock), Memory Management, Processes, Synchronization, Registry y System Information Functions.

A continuación se realiza el filtrado de los n-gramas, seleccionando aquellos n-gramas que contengan dos o más APIs de las categorías relevantes. Una vez realizado el filtrado se calculan el mismo tipo de resultados, estos serán explicados y comentados en las próximas secciones.

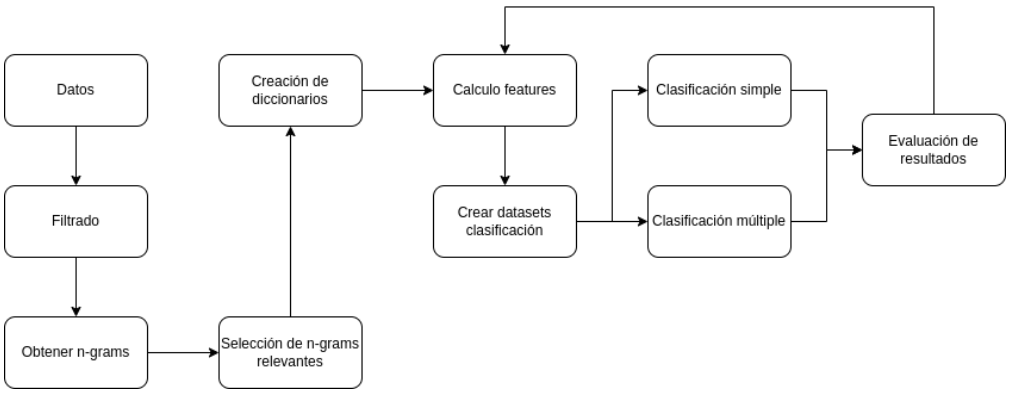

Figura 15: Figura en alto nivel del proceso seguido.

4.2. Sistemas de reconocimiento a programar

El principal sistema a programar en este proceso fue un script de Python encargado de realizar el filtrado y selección de los n-gramas relevantes. Una vez terminado, el script se añadió al proceso automatizado final.

El primer paso fue crear un índice con los contenidos del JSON de categorías. De esta manera es más eficiente realizar búsquedas para cada API que haya en los n-gramas. Este índice se utiliza para obtener la categoría de cada API, creándose una lista con ellas. Esta lista se compara con las categorías previamente seleccionadas como relevantes y si el número de categorías relevantes es igual o mayor a 2 el n-grama se guardará en un diccionario final, utilizado para calcular las features en pasos posteriores.

Finalmente, el resto del sistema se deja igual, añadiéndose este paso como adicional al proceso automatizado que ya se tenía.

4.3. Resultados obtenidos

En esta sección se discutirán y analizarán los resultados obtenidos. Se realizará del mismo modo que en apartados anteriores, comenzando por los resultados de los clasificadores simples y finalizando con los devueltos por los clasificadores múltiples. Los algoritmos seleccionados para el proceso son iguales a los empleados en las clasificaciones anteriores. Estos algoritmos son KNN, SVM, Gradient Boosting y Regresión Logística.

Nos centraremos en explicar los mejores resultados obtenidos con n-gramas desde 2 hasta 5. Tal y como pasaba en el proceso de clasificación anterior, estos resultados fueron obtenidos con trigramas y 4-gramas.

Comenzando con los resultados obtenidos con los clasificadores simples, las Tablas 9, 10 y 11 contienen los resultados obtenidos con trigramas y las Figuras 16, 17 y 18 presentan las matrices de confusión para *pup*, *trojan* y *worm* respectivamente.

N-grams	Modelo ML	Num. traza	precision	recall	f1	support	mcc
3	KNN	2433	0.468	0.497	0.482	465	-0.02
3	svm	2433	0.556	0.927	0.695	465	0.311
3	gradient	2433	0.949	0.4	0.563	465	0.474
3	regression	2433	0.613	0.929	0.738	465	0.439

Tabla 9: Resultados obtenidos para pup con trigramas empleando filtrado de categorías relevantes.

```
Elemento pup con clasificador KNN:

              precision    recall  f1-score   support

       other       0.51      0.48      0.50       509
         pup       0.47      0.50      0.48       465

    accuracy                           0.49       974
   macro avg       0.49      0.49      0.49       974
weighted avg       0.49      0.49      0.49       974

Matriz de confusión:
------------------------------
[[246 263]
 [234 231]]

Elemento pup con clasificador svm:

              precision    recall  f1-score   support

       other       0.83      0.32      0.47       509
         pup       0.56      0.93      0.70       465

    accuracy                           0.61       974
   macro avg       0.69      0.63      0.58       974
weighted avg       0.70      0.61      0.58       974

Matriz de confusión:
------------------------------
[[165 344]
 [ 34 431]]

Elemento pup con clasificador gradient:

              precision    recall  f1-score   support

       other       0.64      0.98      0.78       509
         pup       0.95      0.40      0.56       465

    accuracy                           0.70       974
   macro avg       0.80      0.69      0.67       974
weighted avg       0.79      0.70      0.67       974

Matriz de confusión:
------------------------------
[[499  10]
 [279 186]]

Elemento pup con clasificador regression:

              precision    recall  f1-score   support

       other       0.88      0.46      0.61       509
         pup       0.61      0.93      0.74       465

    accuracy                           0.69       974
   macro avg       0.75      0.70      0.67       974
weighted avg       0.75      0.69      0.67       974

Matriz de confusión:
------------------------------
[[236 273]
 [ 33 432]]
```

Figura 16: Matriz de confusión obtenida por el clasificador simple empleando trigramas para pup.

N-grams	Modelo ML	Num. traza	precision	recall	f1	support	mcc
3	KNN	7314	0.682	0.616	0.648	1408	0.007
3	svm	7314	0.777	0.984	0.868	1408	0.518
3	gradient	7314	0.681	0.996	0.809	1408	0.021
3	regression	7314	0.788	0.976	0.872	1408	0.536

Tabla 10: Resultados obtenidos para trojan con trigramas empleando filtrado de categorías relevantes.

N-grams	Modelo ML	Num. traza	precision	recall	f1	support	mcc
3	KNN	605	0.474	0.839	0.606	118	-0.07
3	svm	605	0.55	0.932	0.692	118	0.272
3	gradient	605	0.496	1	0.663	118	0.126
3	regression	605	0.642	0.941	0.763	118	0.488

Tabla 11: Resultados obtenidos para worm con trigramas empleando filtrado de categorías relevantes.

```
Elemento trojan con clasificador KNN:

              precision    recall  f1-score   support

       other       0.32      0.39      0.35       663
      trojan       0.68      0.62      0.65      1408

    accuracy                           0.54      2071
   macro avg       0.50      0.50      0.50      2071
weighted avg       0.57      0.54      0.55      2071

Matriz de confusión:
-------------------------------
[[259 404]
 [540 868]]

Elemento trojan con clasificador svm:

              precision    recall  f1-score   support

       other       0.92      0.40      0.56       663
      trojan       0.78      0.98      0.87      1408

    accuracy                           0.80      2071
   macro avg       0.85      0.69      0.71      2071
weighted avg       0.82      0.80      0.77      2071

Matriz de confusión:
-------------------------------
[[ 266  397]
 [  23 1385]]

Elemento trojan con clasificador gradient:

              precision    recall  f1-score   support

       other       0.45      0.01      0.01       663
      trojan       0.68      1.00      0.81      1408

    accuracy                           0.68      2071
   macro avg       0.57      0.50      0.41      2071
weighted avg       0.61      0.68      0.55      2071

Matriz de confusión:
-------------------------------
[[   5  658]
 [   6 1402]]

Elemento trojan con clasificador regression:

              precision    recall  f1-score   support

       other       0.90      0.44      0.59       663
      trojan       0.79      0.98      0.87      1408

    accuracy                           0.81      2071
   macro avg       0.84      0.71      0.73      2071
weighted avg       0.82      0.81      0.78      2071

Matriz de confusión:
-------------------------------
[[ 294  369]
 [  34 1374]]
```

Figura 17: Matriz de confusión obtenida por el clasificador simple empleando trigramas para trojan.

```
Elemento worm con clasificador KNN:

              precision    recall  f1-score   support

       other       0.42      0.11      0.18       124
        worm       0.47      0.84      0.61       118

    accuracy                           0.47       242
   macro avg       0.45      0.48      0.39       242
weighted avg       0.45      0.47      0.39       242

Matriz de confusión:
------------------------------
[[ 14 110]
 [ 19  99]]

Elemento worm con clasificador svm:

              precision    recall  f1-score   support

       other       0.81      0.27      0.41       124
        worm       0.55      0.93      0.69       118

    accuracy                           0.60       242
   macro avg       0.68      0.60      0.55       242
weighted avg       0.68      0.60      0.55       242

Matriz de confusión:
------------------------------
[[ 34  90]
 [  8 110]]

Elemento worm con clasificador gradient:

              precision    recall  f1-score   support

       other       1.00      0.03      0.06       124
        worm       0.50      1.00      0.66       118

    accuracy                           0.50       242
   macro avg       0.75      0.52      0.36       242
weighted avg       0.75      0.50      0.36       242

Matriz de confusión:
------------------------------
[[  4 120]
 [  0 118]]

Elemento worm con clasificador regression:

              precision    recall  f1-score   support

       other       0.90      0.50      0.64       124
        worm       0.64      0.94      0.76       118

    accuracy                           0.71       242
   macro avg       0.77      0.72      0.70       242
weighted avg       0.77      0.71      0.70       242

Matriz de confusión:
------------------------------
[[ 62  62]
 [  7 111]]
```

Figura 18: Matriz de confusión obtenida por el clasificador simple empleando trigramas para worm.

También se pueden ver las Tablas 12, 13 y 14 contienen los resultados obtenidos con 4-gramas y las Figuras 19, 20 y 21 que contienen las matrices de confusión para *pup*, *trojan* y *worm* respectivamente.

N-grams	Modelo ML	Num. traza	precision	recall	f1	support	mcc
4	KNN	2433	0.452	0.528	0.487	468	-0.066
4	svm	2433	0.553	0.934	0.695	468	0.301
4	gradient	2433	0.841	0.417	0.557	468	0.403
4	regression	2433	0.621	0.942	0.749	468	0.462

Tabla 12: Resultados obtenidos para pup con n-gramas de 4 empleando filtrado de categorías relevantes.

N-grams	Modelo ML	Num. traza	precision	recall	f1	support	mcc
4	KNN	7314	0.714	0.594	0.649	1474	0.006
4	svm	7314	0.789	0.989	0.878	1474	0.492
4	gradient	7314	0.714	0.995	0.831	1474	0.053
4	regression	7314	0.819	0.927	0.87	1474	0.481

Tabla 13: Resultados obtenidos para trojan con n-gramas de 4 empleando filtrado de categorías relevantes.

N-grams	Modelo ML	Num. traza	precision	recall	f1	support	mcc
4	KNN	605	0.502	0.944	0.655	124	-0.104
4	svm	605	0.515	1	0.679	124	0.066
4	gradient	605	0.521	1	0.685	124	0.133
4	regression	605	0.674	1	0.805	124	0.576

Tabla 14: Resultados obtenidos para worm con n-gramas de 4 empleando filtrado de categorías relevantes.

```
Elemento pup con clasificador KNN:

              precision    recall  f1-score   support

       other       0.48      0.41      0.44       506
         pup       0.45      0.53      0.49       468

    accuracy                           0.47       974
   macro avg       0.47      0.47      0.46       974
weighted avg       0.47      0.47      0.46       974

Matriz de confusión:
--------------------------------
[[206 300]
 [221 247]]

Elemento pup con clasificador svm:

              precision    recall  f1-score   support

       other       0.83      0.30      0.44       506
         pup       0.55      0.93      0.69       468

    accuracy                           0.61       974
   macro avg       0.69      0.62      0.57       974
weighted avg       0.70      0.61      0.56       974

Matriz de confusión:
--------------------------------
[[153 353]
 [ 31 437]]

Elemento pup con clasificador gradient:

              precision    recall  f1-score   support

       other       0.63      0.93      0.75       506
         pup       0.84      0.42      0.56       468

    accuracy                           0.68       974
   macro avg       0.74      0.67      0.65       974
weighted avg       0.73      0.68      0.66       974

Matriz de confusión:
--------------------------------
[[469  37]
 [273 195]]

Elemento pup con clasificador regression:

              precision    recall  f1-score   support

       other       0.90      0.47      0.62       506
         pup       0.62      0.94      0.75       468

    accuracy                           0.70       974
   macro avg       0.76      0.71      0.68       974
weighted avg       0.76      0.70      0.68       974

Matriz de confusión:
--------------------------------
[[237 269]
 [ 27 441]]
```

Figura 19: Matriz de confusión obtenida por el clasificador simple empleando 4-gramas para pup.

```
Elemento trojan con clasificador KNN:

              precision    recall  f1-score   support

       other       0.29      0.41      0.34       597
      trojan       0.71      0.59      0.65      1474

    accuracy                           0.54      2071
   macro avg       0.50      0.50      0.50      2071
weighted avg       0.59      0.54      0.56      2071

Matriz de confusión:
-------------------------------
[[246 351]
 [598 876]]

Elemento trojan con clasificador svm:

              precision    recall  f1-score   support

       other       0.93      0.35      0.51       597
      trojan       0.79      0.99      0.88      1474

    accuracy                           0.80      2071
   macro avg       0.86      0.67      0.69      2071
weighted avg       0.83      0.80      0.77      2071

Matriz de confusión:
-------------------------------
[[ 208  389]
 [  16 1458]]

Elemento trojan con clasificador gradient:

              precision    recall  f1-score   support

       other       0.56      0.02      0.03       597
      trojan       0.71      1.00      0.83      1474

    accuracy                           0.71      2071
   macro avg       0.64      0.51      0.43      2071
weighted avg       0.67      0.71      0.60      2071

Matriz de confusión:
-------------------------------
[[   9  588]
 [   7 1467]]

Elemento trojan con clasificador regression:

              precision    recall  f1-score   support

       other       0.73      0.49      0.59       597
      trojan       0.82      0.93      0.87      1474

    accuracy                           0.80      2071
   macro avg       0.78      0.71      0.73      2071
weighted avg       0.79      0.80      0.79      2071

Matriz de confusión:
-------------------------------
[[ 294  303]
 [ 107 1367]]
```

Figura 20: Matriz de confusión obtenida por el clasificador simple empleando 4-gramas para trojan.

```
Elemento worm con clasificador KNN:

              precision    recall  f1-score   support

       other       0.22      0.02      0.03       118
        worm       0.50      0.94      0.66       124

    accuracy                           0.49       242
   macro avg       0.36      0.48      0.34       242
weighted avg       0.37      0.49      0.35       242

Matriz de confusión:
--------------------------------
[[  2 116]
 [  7 117]]

Elemento worm con clasificador svm:

              precision    recall  f1-score   support

       other       1.00      0.01      0.02       118
        worm       0.51      1.00      0.68       124

    accuracy                           0.52       242
   macro avg       0.76      0.50      0.35       242
weighted avg       0.75      0.52      0.36       242

Matriz de confusión:
--------------------------------
[[  1 117]
 [  0 124]]

Elemento worm con clasificador gradient:

              precision    recall  f1-score   support

       other       1.00      0.03      0.07       118
        worm       0.52      1.00      0.69       124

    accuracy                           0.53       242
   macro avg       0.76      0.52      0.38       242
weighted avg       0.75      0.53      0.38       242

Matriz de confusión:
--------------------------------
[[  4 114]
 [  0 124]]

Elemento worm con clasificador regression:

              precision    recall  f1-score   support

       other       1.00      0.49      0.66       118
        worm       0.67      1.00      0.81       124

    accuracy                           0.75       242
   macro avg       0.84      0.75      0.73       242
weighted avg       0.83      0.75      0.73       242

Matriz de confusión:
--------------------------------
[[ 58  60]
 [  0 124]]
```

Figura 21: Matriz de confusión obtenida por el clasificador simple empleando 4-gramas para worm.

Como se puede ver en las Tablas, los mejores resultados obtenidos para cada clase se han conseguido al emplear 4-grams al conseguir un f1-score de 0.749 para *pup* con Regresión Logística (Tabla 12), para *trojan* con KNN un f1-score de 0.878 (Tabla 13) y para *worm* con Regresión Logística un f1-score de 0.805 (Tabla 14).

El mejor modelo generado por los clasificadores simples fue el de *trojan* utilizando SVM con 4-gramas (ver Figura 20). Se puede observar que la clasificación de *trojan* es efectiva, teniendo un valor de 0.88 en el f1-score. En cambio, la clasificación de *other* resulta con un f1-score de 0.51, mejorando bastante el resultado obtenido sin aplicar el filtrado. Esto deja el valor medio en un 0.77, mejorando claramente la media obtenida anteriormente, de 0.59.

En la matriz podemos ver que 1458 muestras han sido bien clasificadas como *trojan*, dejando 16 mal clasificadas. En el caso de *other* solamente 208 muestras han sido bien identificadas, dejando 389 mal clasificadas. Estos valores son una gran mejora comparándolos con los obtenidos sin aplicar el filtrado de categorías.

En cuanto a los resultados obtenidos por los clasificadores múltiples, la Tabla 15 contiene los resultados obtenidos con trigramas y las Figuras 22, 23 y 24 presentan las matrices de confusión para cada clasificador.

N-grams	Modelo ML	precision	recall	f1	support	mcc
3	KNN	0.789	0.801	0.788	2287	0.614
3	svm	0.716	0.729	0.683	2287	0.43
3	gradient	0.796	0.794	0.76	2287	0.586

Tabla 15: Resultados obtenidos para el clasificador múltiple usando trigramas empleando filtrado de categorías relevantes.

```
Clasificador KNN:

              precision    recall  f1-score   support

       other       0.54      0.40      0.46       227
         pup       0.82      0.82      0.82       481
      trojan       0.83      0.90      0.86      1451
       worm        0.66      0.27      0.39       128

    accuracy                           0.80      2287
   macro avg       0.71      0.60      0.63      2287
weighted avg       0.79      0.80      0.79      2287

Matriz de confusión:
------------------------------
[[  91   19  116    1]
 [   6  395   77    3]
 [  64   61 1312   14]
 [   6    8   79   35]]
```

Figura 22: Matriz de confusión obtenida por el clasificador múltiple empleando KNN para trigramas.

Los resultados obtenidos con 4-gramas pueden verse en la Tabla 16 junto con las matrices de confusión obtenidas para cada clasificador en las Figuras 25, 26 y 27.

```
Clasificador svm:

              precision    recall  f1-score   support

       other       0.31      0.09      0.14       227
         pup       0.85      0.50      0.63       481
      trojan       0.72      0.96      0.82      1451
        worm       0.83      0.15      0.25       128

    accuracy                           0.73      2287
   macro avg       0.68      0.42      0.46      2287
weighted avg       0.72      0.73      0.68      2287

Matriz de confusión:
-------------------------------
[[  20    6  200    1]
 [  15  239  225    2]
 [  29   32 1389    1]
 [   1    3  105   19]]
```

Figura 23: Matriz de confusión obtenida por el clasificador múltiple empleando SVM para trigramas.

```
Clasificador gradient:

              precision    recall  f1-score   support

       other       0.74      0.20      0.31       227
         pup       0.85      0.74      0.79       481
      trojan       0.78      0.96      0.86      1451
        worm       0.84      0.16      0.27       128

    accuracy                           0.79      2287
   macro avg       0.80      0.52      0.56      2287
weighted avg       0.80      0.79      0.76      2287

Matriz de confusión:
-------------------------------
[[  45   10  171    1]
 [   0  356  123    2]
 [  14   42 1394    1]
 [   2    9   96   21]]
```

Figura 24: Matriz de confusión obtenida por el clasificador múltiple empleando KNN para trigramas.

```
Clasificador KNN:

              precision    recall  f1-score   support

       other       0.63      0.27      0.38       234
         pup       0.82      0.80      0.81       511
      trojan       0.81      0.93      0.87      1429
        worm       0.69      0.26      0.37       113

    accuracy                           0.80      2287
   macro avg       0.74      0.56      0.61      2287
weighted avg       0.79      0.80      0.78      2287

Matriz de confusión:
------------------------------
[[  64   13  156    1]
 [  10  407   91    3]
 [  26   61 1333    9]
 [   2   13   69   29]]
```

Figura 25: Matriz de confusión obtenida por el clasificador múltiple empleando KNN para 4-gramas.

```
Clasificador svm:

              precision    recall  f1-score   support

       other       1.00      0.00      0.01       234
         pup       0.81      0.52      0.63       511
      trojan       0.71      0.96      0.82      1429
        worm       0.83      0.18      0.29       113

    accuracy                           0.72      2287
   macro avg       0.84      0.42      0.44      2287
weighted avg       0.77      0.72      0.67      2287

Matriz de confusión:
------------------------------
[[   1    1  230    2]
 [   0  266  245    0]
 [   0   56 1371    2]
 [   0    4   89   20]]
```

Figura 26: Matriz de confusión obtenida por el clasificador múltiple empleando SVM para 4-gramas.

n-grams	Modelo ML	precision	recall	f1	support	mcc
4	KNN	0.787	0.801	0.78	2287	0.612
4	svm	0.768	0.725	0.666	2287	0.431
4	gradient	0.786	0.79	0.758	2287	0.584

Tabla 16: Resultados obtenidos para el clasificador múltiple usando n-gramas de 4 empleando filtrado de categorías relevantes.

```
Clasificador gradient:

              precision    recall  f1-score   support

       other       0.75      0.21      0.33       234
         pup       0.83      0.74      0.78       511
      trojan       0.78      0.95      0.86      1429
        worm       0.72      0.16      0.26       113

    accuracy                           0.79      2287
   macro avg       0.77      0.52      0.56      2287
weighted avg       0.79      0.79      0.76      2287

Matriz de confusión:
------------------------------
[[  50   10  172    2]
 [   1  378  131    1]
 [  16   49 1360    4]
 [   0   16   79   18]]
```

Figura 27: Matriz de confusión obtenida por el clasificador múltiple empleando Gradient Boosting para 4-gramas.

Al analizar las tablas podemos ver que los valores obtenidos por los trigramas superan a los del los 4-gramas, aunque no con una gran ventaja, ya que los valores son muy cercanos en ambos casos. El mejor resultado para ambos se consigue al aplicar KNN, en el caso de los trigramas se obtiene un f1-score de 0.788 (Tabla 15) y en los 4-grams un valor de 0.78 (Tabla 16).

De la misma manera que ocurría en los resultados sin filtrado, el mejor modelo calculado por los clasificadores múltiples fue conseguido tras aplicar KNN (ver Figura 22), pero en este caso con trigramas. Tal y como se ha observado en los resultados obtenidos por los clasificadores simples, aquí también se ve una gran mejoría en los resultados, especialmente en el caso de *other*, subiendo de un valor de 0.28 a uno de 0.46, aunque se puede ver que el valor devuelto por el f1-score para *worm* ha bajado de 0.49 a 0.39.

El mejor resultado vuelve a tenerlo *trojan* con 0.86, seguido de *pup* con 0.82, *other* con 0.46 y finalmente *worm* con 0.39, dejando el valor medio en 0.79, mejorando el resultado anterior de 0.77, aunque no por mucho.

Al comparar estos resultados con los obtenidos sin el filtrado de categorías podemos apreciar que todos ellos han mejorado. Esto demuestra la eficacia de la mejora implementada a la hora de clasificar muestras de malware.

4.4. Discusión sobre los resultados

A continuación se realizará la discusión de los resultados y matrices de confusión obtenidas. Se compararán los resultados obtenidos en la sección anterior y se examinará la efectividad de los modelos implementados respecto a los conseguidos sin la mejora.

Si analizamos los resultados podemos ver que en los casos de *trojan* y *pup* la mayoría de las muestras han sido clasificadas de manera correcta, siendo las dos categorías que obtienen mejores resultados. En los casos de *other* y *worm* la mayoría de las muestras no han sido clasificadas de manera correcta, obteniendo valores del f1-score menores a 0.5 en ambos casos.

Se puede apreciar que *other* y *worm* han sido confundidos mayoritariamente con *trojan* y que *pup* ha sido clasificado bien en su gran mayoría, aunque la clase con la que más se la ha confundido ha sido trojan. Finalmente, *trojan* ha sido bien clasificado en su gran mayoría, aunque la mayor confusión ha ocurrido con *other*.

En general, los valores obtenidos tras aplicar el filtrado han mejorado los anteriores, especialmente a la hora de la clasificación de *other*, subiendo su f1-score de 0.06 a 0.51 en los clasificadores simples y de 0.28 a 0.46 en los clasificadores múltiples. El resto de valores también han visto mejoría.

Esta mejora en los resultados se puede atribuir al filtrado basado en categorías de interés implementado. Este filtrado ha permitido a los modelos enfocarse en las características más relevantes y distintivas entre categorías. Además, al conservar únicamente los n-gramas que contienen dos o más APIs de las categorías relevantes, se consigue una representación de los patrones característicos más precisa. Esto se ve traducido en una mejora considerable en los resultados.

En cuanto a la comparativa entre clasificadores simples y múltiples, no se ve una gran mejoría en los múltiples frente a los simples, ya que el rango de valores en los que se encuentra el f1-score es muy similar.

Si nos centramos en los resultados obtenidos con trigramas podemos ver que solamente un clasificador mejora el valor obtenido por el clasificador múltiple, el modelo que emplea Regresión Logística para la clasificación de *trojan*, cuyo f1-score de 0.872 mejora el 0.788 del clasificador múltiple.

En el caso de los resultados obtenidos con 4-gramas solamente hay un clasificador que no es capaz de superar el resultado obtenido por el clasificador múltiple, este es el modelo que emplea Regresión Logística para la clasificación de *pup*, obtiene un f1-score de 0.749 frente al 0.78 del clasificador múltiple. Aunque no supere el resultado, si que se encuentra mucho más cerca de él si lo comparamos con el conseguido con los trigramas.

5. Conclusiones y trabajo futuros

En esta sección de la memoria se expondrán las conclusiones finales del proyecto, tanto técnicas como personales, así como posibles trabajos futuros.

5.1. Conclusiones técnicas

La solución propuesta a la detección de *malware* utilizando técnicas de Machine Learning ha demostrado ser efectiva a la hora de detectar distintas categorías de malware. Esto ha demostrado que el uso de n-gramas en el ámbito de la detección de *malware* es de gran utilidad, ya que permite detectar comportamientos analizando subsecciones de las secuencias en vez de analizar una a una las APIs que van apareciendo en cada muestra.

En cuanto a la implementación, se han cumplido los objetivos del proyecto, incluso llegando a ampliar la funcionalidad planteada inicialmente. Esto demuestra que la implementación inicial es robusta y admite extensiones y mejoras de la misma.

En una reflexión sobre el trabajo desarrollado, se destaca que la elección de algoritmos de clasificación o técnicas se pudo haber abordado de una manera diferente. Al tomar otro rumbo o elegir algoritmos distintos habría influido directamente en las conclusiones obtenidas.

Por ejemplo, la selección de los n-gramas y categorías de interés podría haber adoptado enfoques completamente distintos. Al añadir o modificar alguna de las APIs consideradas potencialmente maliciosas se hubiesen obtenido resultados distintos. En futuros proyectos se explorarían estas alternativas para mejorar potencialmente los resultados obtenidos.

5.2. Conclusiones personales

El hecho de que este trabajo haya dado los frutos deseados, demostrando la utilidad del uso de n-gramas y técnicas de Machine Learning a la hora de lidiar con problemas de clasificación de malware, hace que todo el esfuerzo dedicado a él merezca la pena.

Desde el inicio del proyecto he intentado esforzarme al máximo y ver que ha dado sus frutos es muy gratificante, no solo por los resultados obtenidos sino por todo el conocimiento adquirido. Este trabajo ha sido un desafío tanto de investigación como de programación y he podido aplicar conocimientos adquiridos durante la totalidad de mi educación universitaria.

En este trabajo he podido aplicar conocimientos de ciberseguridad y Machine Learning, desarrollando habilidades prácticas en la detección de *malware*. La utilización de herramientas como JupyterLab, Scikit-Learn o nltk ha ampliado mis capacidades técnicas, brindándome una perspectiva más completa.

La evaluación de modelos y la comprensión de las métricas de rendimiento han enriquecido mi comprensión sobre cómo abordar problemas complejos y tomar decisiones informadas. Estas experiencias han sido esenciales para mi desarrollo profesional como ingeniera informática, proporcionándome una valiosa oportunidad para enfrentar desafíos reales en el ámbito de la seguridad informática.

5.3. Trabajo futuro

El proyecto sienta las bases para futuros trabajos, estableciendo un punto de partida para futuros desarrollos. Se plantea la posibilidad de ampliar las capacidades del sistema mediante la incorporación de funcionalidades adicionales al proceso. Esto podría enriquecer y ampliar su aplicación en diferentes conjuntos de datos y categorías de *malware*.

La automatización implementada no solo simplifica la modificación de partes del código, sino que también blinda la posibilidad de introducir pasos intermedios. Un ejemplo concreto de la inclusión del filtrado de categorías relevantes, una mejora especifica que ha demostrado ser eficaz. Esta modularidad hace que sea más fácil ajustar el proceso a problemas particulares.

Además de las posibles mejoras previamente mencionadas, cabe la posibilidad de integrar la automatización con otras herramientas y enfoques. Por ejemplo, explorar técnicas adicionales de Machine Learning o incluso de Deep Learning para proporcionar al proyecto una mayor precisión en la detección de *malware*.

Así mismo, el desarrollo de un dataset propio podría mejorar el rendimiento de los clasificadores al incluir datos más específicos o trazas con mayor relevancia. Esto podría involucrar incorporar trazas de código benigno para mejorar la capacidad de clasificación de los modelos.

La implementación de estos cambios implica que el sistema se vuelva más flexible y efectivo. La modularidad facilita añadir nuevas funcionalidades y el dataset propio mejoraría la precisión de los modelos.

6. Bibliografía

[1] W. Tounsi and H. Rais, "A survey on technical threat intelligence in the age of sophisticated cyber attacks," *Computers Security*, vol. 72, pp. 212–233, 2018.

[2] "Más de 4,9 millones de muestras de malware identificadas." https://www.itdigitalsecurity.es/endpoint/2020/02/mas-de-49-millones-de-muestras-de-malware-fueron-identificadas-en-2019. Accedido el 01-12-2023.

[3] "VirusTotal's 2021 Malware Trends Report." https://assets.virustotal.com/reports/2021trends.pdf. Accedido el 11-12-2023.

[4] "¿Qué es un Exploit? Prevención de Exploits." https://www.bitdefender.es/consumer/support/answer/22884/. Accedido el 02-12-2023.

[5] "Malware Statistics Trends Report | AV-TEST." https://www.av-test.org/en/statistics/malware/. Accedido el 12-12-2023.

[6] "What is Malware Detection? | Importance of Malware Tool." https://enterprise.xcitium.com/what-is-malware-detection/. Accedido el 09-01-2024.

[7] "Malware Classification." https://serp.ai/malware-classification/. Accedido el 09-01-2024.

[8] I. You and K. Yim, "Malware obfuscation techniques: A brief survey," pp. 297–300, 2010.

[9] "Metamorfismo (malware) - Wikipedia, la enciclopedia libre." https://es.wikipedia.org/wiki/Metamorfismo_(malware). Accedido el 02-12-2023.

[10] "Polimorfismo (malware) - Wikipedia, la enciclopedia libre." https://es.wikipedia.org/wiki/Polimorfismo_(malware). Accedido el 02-12-2023.

[11] "Ofuscación - Wikipedia, la enciclopedia libre." https://es.wikipedia.org/wiki/OfuscaciÃşn. Accedido el 02-12-2023.

[12] "Qué es una firma de archivo de malware y cómo funciona." https://ciberseguridad.com/amenzas/firma-archivo-malware/. Accedido el 09-01-2024.

[13] "¿Qué es un análisis heurístico?." https://latam.kaspersky.com/resource-center/definitions/heuristic-analysis. Accedido el 09-01-2024.

[14] "¿Qué es el aprendizaje automático? | Glosario | HPE LAMERICA." https://www.hpe.com/lamerica/es/what-is/machine-learning.html. Accedido el 09-01-2024.

[15] E. Raff and C. K. Nicholas, "A survey of machine learning methods and challenges for windows malware classification," *ArXiv*, vol. abs/2006.09271, 2020.

[16] "Análisis del comportamiento | ES." https://www.vmware.com/es/topics/glossary/content/behavioral-analysis.html. Accedido el 09-01-2024.

[17] "Análisis estático de malware – CYBER OPSEC." https://www.cyberopsec.com.mx/blog/analisis-estatico-de-malware/. Accedido el 09-01-2024.

[18] "WeLiveSecurity - Análisis estático." https://www.welivesecurity.com/la-es/2014/01/14/bases-analisis-estatico-malware-bases-desensamblado/. Accedido el 09-01-2024.

[19] "Código binario: la base de todo - DevCamp." https://devcamp.es/codigo-binario-la-base-de-todo/. Accedido el 09-01-2024.

[20] "Qué es Código fuente - Definición, significado y ejemplos." https://www.arimetrics.com/glosario-digital/codigo-fuente. Accedido el 09-01-2024.

[21] "WeLiveSecurity - Análisis dinámico." https://www.welivesecurity.com/la-es/2011/12/22/herramientas-analisis-dinamico-malware/. Accedido el 09-01-2024.

[22] "Sandbox - ¿Qué es y cómo funciona? | Proofpoint ES." https://www.proofpoint.com/es/threat-reference/sandbox. Accedido el 09-01-2024.

[23] "¿Qué es la detección basada en firmas? | phoenixNAP Glosario de TI." https://www.phoenixnap.mx/glosario/detecciÃ§n-basada-en-firmas. Accedido el 09-01-2024.

[24] "¿Qué hace un antivirus para detectar el malware? | Empresas | INCIBE." https://www.incibe.es/empresas/blog/hace-antivirus-detectar-el-malware. Accedido el 09-01-2024.

[25] "Exploración, identificación y detección de malware inteligente para evitar caos ciberepidemiológico y ciberpandemias - Ciberseguridad." https://www.interempresas.net/Ciberseguridad/Articulos/353718-Exploracion-identificacion-deteccion-malware-inteligente-evitar-caos-ciberepidemiologico.html. Accedido el 09-01-2024.

[26] "System call: la importancia de las llamadas al sistema - IONOS." https://www.ionos.es/digitalguide/servidores/know-how/que-son-las-system-calls-de-linux/. Accedido el 09-01-2024.

[27] "Api: qué es y para qué sirve." https://www.xataka.com/basics/api-que-sirve. Accedido el 02-12-2023.

[28] Y. Ki, E. Kim, and H. K. Kim, "A novel approach to detect malware based on api call sequence analysis," *International Journal of Distributed Sensor Networks*, vol. 2015, pp. 1–9, 06 2015.

[29] "Software potencialmente no deseado - Wikipedia, la enciclopedia libre." https://es.wikipedia.org/wiki/Software_potencialmente_no_deseado. Accedido el 02-12-2023.

[30] "Qué es un virus troyano | Definición de virus troyano." https://www.kaspersky.es/resource-center/threats/trojans. Accedido el 02-12-2023.

[31] "Gusano informático: definición y riesgos - Panda Security." https://www.pandasecurity.com/es/security-info/worm/. Accedido el 02-12-2023.

[32] "Win32 API - Wikipedia, la enciclopedia libre." https://es.wikipedia.org/wiki/Win32_API. Accedido el 02-12-2023.

[33] "Servicios del sistema - Win32 apps | Microsoft Learn." https://learn.microsoft.com/es-es/windows/win32/api/_base/. Accedido el 02-12-2023.

[34] "Uso de IA y Machine Learning en ciberseguridad | OpenWebinars." https://openwebinars.net/blog/uso-de-inteligencia-artificial-y-machine-learning-en-ciberseguridad/. Accedido el 20-12-2023.

[35] "Ai brings speed to security." https://www.oreilly.com/content/ai-brings-speed-to-security/. Accedido el 01-12-2023.

[36] "La IA en la automatización de pruebas de seguridad informática." https://www.imediacomunicacion.com/la-ia-en-la-automatizacion-de-pruebas-de-seguridad-informatica/. Accedido el 01-12-2023.

[37] "¿Cómo aporta la inteligencia artificial a la ciberseguridad? - Ikusi ES." https://www.ikusi.com/es/blog/como-aporta-la-inteligencia-artificial-a-la-ciberseguridad/. Accedido el 20-12-2023.

[38] "La revolución de la Inteligencia Artificial en la ciberseguridad y como Auditech se mantiene a la vanguardia | Auditech." https://auditech.es/blog/la-revolucion-de-la-inteligencia-artificial-en-la-ciberseguridad-y-como-auditech-se-mantiene-a-la-vanguardia/. Accedido el 20-12-2023.

[39] "What is Machine Learning? | IBM." https://www.ibm.com/topics/machine-learning. Accedido el 02-12-2023.

[40] S. Venkatraman, M. Alazab, and R. Vinayakumar, "A hybrid deep learning image-based analysis for effective malware detection," *Journal of Information Security and Applications*, vol. 47, pp. 377–389, 2019.

[41] "Malware Detection Using Deep Learning | by Ria Kulshrestha | Towards Data Science." https://towardsdatascience.com/malware-detection-using-deep-learning-6c95dd235432. Accedido el 02-12-2023.

[42] "Machine Learning | Qué es, tipos, ejemplos y cómo implementarlo." https://www.grapheverywhere.com/machine-learning-que-es-tipos-ejemplos-y-como-implementarlo/. Accedido el 02-12-2023.

[43] "Malware Detection Using Machine Learning Techniques." https://www.einfochips.com/blog/malware-detection-using-machine-learning-techniques/#:~:text=%E2%80%9CMachine%20Learning%20has%20improved%20the,a%20wide%20range%20of%20threats. Accedido el 02-12-2023.

[44] S. Trivedi, "A study of machine learning classifiers for spam detection," pp. 176–180, 09 2016.

[45] M. Alazab, S. Venkatraman, P. Watters, and M. Alazab, "Zero-day malware detection based on supervised learning algorithms of api call signatures," p. 171–182, 2011.

[46] A. Tang, S. Sethumadhavan, and S. J. Stolfo, "Unsupervised anomaly-based malware detection using hardware features," pp. 109–129, 2014.

[47] I. Santos, J. Nieves, and P. G. Bringas, "Semi-supervised learning for unknown malware detection," in *International Symposium on Distributed Computing and Artificial Intelligence* (A. Abraham, J. M. Corchado, S. R. González, and J. F. De Paz Santana, eds.), (Berlin, Heidelberg), pp. 415–422, Springer Berlin Heidelberg, 2011.

[48] "What is Reinforcement Learning? – Overview of How it Works | Synopsys." https://www.synopsys.com/ai/what-is-reinforcement-learning.html. Accedido el 02-12-2023.

[49] T. T. Nguyen and V. J. Reddi, "Deep reinforcement learning for cyber security," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 34, no. 8, pp. 3779–3795, 2023.

[50] R. Canzanese, S. Mancoridis, and M. Kam, "System call-based detection of malicious processes," pp. 119–124, 08 2015.

[51] Y. Tang, X. Qi, J. Jing, C. Liu, and W. Dong, "Bhmdc: A byte and hex n-gram based malware detection and classification method," *Computers Security*, vol. 128, p. 103118, 2023.

[52] A. Pektas, M. D. Eriş, and T. Acarman, "Proposal of n-gram based algorithm for malware classification," in *International Conference on Emerging Security Information, Systems and Technologies*, 2011.

[53] M. Alazab, R. Layton, S. Venkatraman, and P. Watters, "Malware detection based on structural and behavioural features of api calls," *International Cyber Resilience conference*, 03 2012.

[54] D. Reddy and A. K. Pujari, "N-gram analysis for computer virus detection," *Journal in Computer Virology*, vol. 2, pp. 231–239, 12 2006.

[55] "Bag-of-n-grams - Machine Learning Glossary." https://machinelearning.wtf/terms/bag-of-n-grams/. Accedido el 15-12-2023.

[56] "Recursive Feature Elimination (RFE) for Feature Selection in Python - MachineLearningMastery.com." https://machinelearningmastery.com/rfe-feature-selection-in-python/. Accedido el 15-12-2023.

[57] "Prueba de Chi-cuadrado - Explicación sencilla - DATAtab." https://datatab.es/tutorial/chi-square-test. Accedido el 15-12-2023.

[58] "La prueba t | Introducción a la estadística | JMP." https://www.jmp.com/es_es/statistics-knowledge-portal/t-test.html. Accedido el 15-12-2023.

[59] "¿Qué es el algoritmo de k vecinos más cercanos? | IBM." https://www.ibm.com/es-es/topics/knn. Accedido el 15-12-2023.

[60] "Conceptos clave de Support Vector Machine (SVM) - MATLAB Simulink." https://es.mathworks.com/discovery/support-vector-machine.html. Accedido el 15-12-2023.

[61] "Gradient boosting - Wikipedia, la enciclopedia libre." https://es.wikipedia.org/wiki/Gradient_boosting. Accedido el 15-12-2023.

[62] "¿Qué es la regresión logística? - DataScientest." https://datascientest.com/es/que-es-la-regresion-logistica. Accedido el 15-12-2023.

[63] "Scikit-learn: machine learning in Python — scikit-learn 1.4.0 documentation." https://scikit-learn.org/stable/. Accedido el 01-12-2023.

[64] "JupyterLab información general." https://experienceleague.adobe.com/docs/experience-platform/data-science-workspace/jupyterlab/overview.html?lang=es#:~:text=JupyterLab%20es%20una%20interfaz%20de,Jupyter%20Notebooks%2C%20c%C3%B3digo%20y%20datos. Accedido el 01-12-2023.

[65] "NLTK :: Natural Language Toolkit." https://www.nltk.org/index.html. Accedido el 01-12-2023.

[66] "Public malware dataset generated by Cuckoo Sandbox based on Windows OS API calls analysis for cyber security researchers." https://github.com/ocatak/malware_api_class. Accedido el 15-12-2023.

[67] "Microsoft Malware Classification Challenge (BIG 2015) | Kaggle." https://www.kaggle.com/competitions/malware-classification/data. Accedido el 15-12-2023.

[68] "Malware Analysis Datasets: API Call Sequences | IEEE DataPort." https://ieee-dataport.org/open-access/malware-analysis-datasets-api-call-sequences. Accedido el 15-12-2023.

[69] "HCRL - [HIDE]APIMDS-dataset." https://ocslab.hksecurity.net/apimds-dataset. Accedido el 15-12-2023.

[70] "RazviOverflow winapi categories json." https://github.com/RazviOverflow/winapi_categories_json/blob/main/winapi_functions_by_category.json. Accedido el 15-12-2023.

[71] "lingpy/lpngram: Python library for ngram collection and frequency smoothing." https://github.com/lingpy/lpngram. Accedido el 15-12-2023.

[72] "gpoulter/python-ngram: Set that supports searching by ngram similarity." https://github.com/gpoulter/python-ngram. Accedido el 15-12-2023.

[73] P. Trinius, T. Holz, J. Göbel, and F. C. Freiling, "Visual analysis of malware behavior using treemaps and thread graphs," in *2009 6th International Workshop on Visualization for Cyber Security*, pp. 33–38, 2009.

[74] "Sesgo y Varianza en Machine Learning - Aprende IA." https://aprendeia.com/bias-y-varianza-en-machine-learning/. Accedido el 15-12-2023.

[75] "Evaluando los modelos de Clasificación en Aprendizaje Automático: La matriz de confusión.| profesorDATA.com." https://profesordata.com/2020/08/07/evaluando-los-modelos-de-clasificacion-en-aprendizaje-automatico-la-matriz-de-confusion-claramente-explicada/. Accedido el 15-12-2023.

Apéndice A Glosario

Terminología

Malware Software o programa maligno que realiza acciones dañinas en un sistema informático de forma intencionada y sin el conocimiento del usuario.

Categoría de API Clasifica las APIs en función de sus funciones específicas dentro del sistema.

Traza de API Una traza que muestra las categorías a las que pertenece cada una de las APIs en lugar de listar cada API individualmente.

Ataques de día cero Ataque contra un sistema informático con el objetivo de ejecutar código maligno gracias a vulnerabilidades desconocidas tanto para los usuarios como para el fabricante del producto.

Features Características o rasgos interesantes o importantes que identifican cada una de las instancias de un conjunto de datos.

Tree map Metodo para mostrar datos jerárquicos utilizando figuras anidadas, normalmente rectángulos. Se emplean rectángulos de distintos tamaños para transmitir valores numéricos.

Thread graph Muestra información en forma de hilos (threads) de en forma de gráfico.

Dense Pixel Display Técnica de exploración visual de los datos, busca poder analizar una gran cantidad de datos multidimensionales detectando patrones.

Dataset Conjunto de datos corresponde a los contenidos de una tabla, donde cada columna representa una variable y cada fila una muestra.

N-gramas Sub secuencia de 'n' elementos de una secuencia dada.

Acrónimos

API Aplication Programming Interfaces

ML Machine Learning

RFE Recursive Feature Elimination

CSV Comma Separated Values

JSON JavaScript Object Notation

KNN K Nearest Neighbors

SVM Support Vector Machine