



UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA SUPERIOR DE INGENIERÍA INFORMÁTICA

GRADO EN INGENIERÍA INFORMÁTICA

TECNOLOGÍA ESPECÍFICA DE COMPUTACIÓN

TRABAJO FIN DE GRADO

**Detección y Clasificación de Malware usando técnicas
inteligentes**

Rubén Donate Serrano

Julio de 2018





UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA SUPERIOR DE INGENIERÍA INFORMÁTICA

GRADO EN INGENIERÍA INFORMÁTICA

TECNOLOGÍA ESPECÍFICA DE COMPUTACIÓN

TRABAJO FIN DE GRADO

**Detección y Clasificación de Malware usando técnicas
inteligentes**

Autor: Rubén Donate Serrano

Directores: José Luis Martínez Martínez
José Miguel Puerta Callejón

Julio de 2018

Declaración de Autoría

Yo, Rubén Donate Serrano con DNI 70519349S, declaro que soy el único autor del trabajo fin de grado titulado *Detección y Clasificación de Malware usando técnicas inteligentes* y que el citado trabajo no infringe las leyes en vigor sobre propiedad intelectual y que todo el material no original contenido en dicho trabajo está apropiadamente atribuido a sus legítimos autores.

Albacete, a 9 de julio de 2018

Fdo.: Rubén Donate Serrano

Resumen

Este sería el resumen del TFG.

Índice general

ÍNDICE DE FIGURAS	VII
Lista de Figuras	IX
ÍNDICE DE TABLAS	IX
Lista de Tablas	1
1. INTRODUCCIÓN	1
1.1. Motivación	2
1.2. Objetivos	4
1.3. Estructura de la memoria	5
2. TÉCNICAS Y HERRAMIENTAS UTILIZADAS	7
3. ANTECEDENTES Y ESTADO DE LA CUESTIÓN	11
3.1. Solución 1	11
3.2. Solución 2	13
4. METODOLOGÍA Y DESARROLLO	27
5. EXPERIMENTO Y RESULTADOS	47
6. CONCLUSIONES Y PROPUESTAS	55
6.1. Conclusiones	55
6.2. Trabajo futuro	55
BIBLIOGRAFIA	61
CONTENIDO DEL CD	63

ÍNDICE DE FIGURAS

3.1. Lista de instrucciones ASM manejadas.	14
3.2. Lista de secciones ASM manejadas.	16

ÍNDICE DE TABLAS

4.1. Clase de malware en la base de datos train.	30
5.1. Número de características por tipo de la base de datos de train.	50
5.2. Ratio entre características diferentes y características por tipo de la base de datos de train.	50
5.3. Ratio entre el acumulado total de las características y las características diferentes por tipo de la base de datos de train.	50
5.4. Número de características discretizadas por tipo de la base de datos de train.	51
5.5. Ratio entre número de características distintas y número de características discretizadas por tipo de la base de datos de train.	51
5.6. Tabla de características seleccionadas.	51
5.7. Comparación de los resultados de las distintas soluciones.	52
5.8. Número de características por tipo de la base de datos de test.	52
5.9. Ratio entre características diferentes y características por tipo de la base de datos de test.	52
5.10. Ratio entre el acumulado total de las características y las características diferentes por tipo de la base de datos de test.	53
5.11. Tabla con la predicción realizada para la base de datos de test.	53

Capítulo 1

INTRODUCCIÓN

Actualmente en el mundo hiperconectado en el que vivimos y con la enorme cantidad de datos y archivos que manejamos, hace cada vez mas probable que nos tengamos que enfrentar ante el problema del malware. El malware es un fichero o código, generalmente distribuido a través de la red, que infecta, explora, roba o conduce virtualmente algún procedimiento a donde quiere el atacante [1]. Este problema afecta tanto a los usuarios particulares, pequeñas y medianas empresas e incluso a grandes empresas y gobiernos, pudiendo ocasionar tanto daños financieros como daños emocionales.

Si no todos, la inmensa mayoría de los usuarios utilizamos sistemas antimalware, software especializado para evitar estas amenazas. El problema viene ante la poca eficacia de estos ante las amenazas nuevas. Esto es debido, a los arduos procesos de investigación llevados a cabo por los atacantes para encontrar nuevos vectores de ataque, vulnerabilidades no conocidas, las cuales se conocen como 0days, técnicas mediante las cuales evitar ser detectados por los sistemas antimalware, etc. Dentro de las técnicas mediante las cuales evitar ser detectados por los sistemas antimalware, me resulto interesante una expuesta en la DEFCON 25 por Hyrum Anderson en 2017 [2], en la que se explicaba como usando técnicas de Deep Learning para mutar un malware ya existe y detectado por los sistemas antimalware, conseguía que fuese indetectable por estos. Aunque esta es una técnica es la que mas me llamo la atención debido a la utilización de técnicas de Deep Learning, existen otras muchas. De hecho, en este Trabajo de Final de Grado, de ahora en adelante TFG, me he encontrado con varias de ellas, las cuales se explicarán llegado el momento.

Por lo cual, en el año 2015, Microsoft que se percató de todo esto, decidió publicar un reto en la plataforma Kaggle, especializada en retos de ciencia de datos, para la detección y clasificación de malware con una recompensa económica de 12.000\$ para el primer clasificado, 3.000\$ para el segundo y 1.000\$ para el tercero, para que la co-

unidad de científica desarrollará modelos bajo licencia open source que permitieran clasificar un conjunto de muestras reales que nos proporcionaban. Siendo la descripción para este reto [3]:

Uno de los principales desafíos a los que se enfrenta el antimalware hoy en día es la gran cantidad de datos y archivos que deben evaluarse para detectar posibles intenciones maliciosas. Por ejemplo, los productos antimalware de detección en tiempo real de Microsoft están presentes en más de 160 millones de computadoras en todo el mundo e inspeccionan más de 700 millones de equipos mensualmente. Esto genera decenas de millones de muestras diarias para ser analizadas como posibles malware. Una de las razones principales de estos grandes volúmenes de archivos diferentes es el hecho de que, para evadir la detección, los autores de malware introducen polimorfismo en los componentes maliciosos. Esto significa que los archivos maliciosos que pertenecen a la misma ‘familia’ de malware, con las mismas formas de comportamiento malicioso, se modifican y/u ofuscan constantemente mediante diversas tácticas, de modo que se parecen a muchos archivos diferentes.

Para ser efectivos al analizar y clasificar cantidades tan grandes de archivos, debemos ser capaces de agruparlos e identificar a la familia que le corresponde. Además, dichos criterios de agrupamiento se pueden aplicar a los archivos nuevos que se encuentran en las computadoras para detectarlos como maliciosos y determinar a que familia corresponde.

Por todo esto, Microsoft decidió proporcionar a la comunidad de científicos de datos un conjunto de datos de malware sin precedentes [4] y fomento la creación de técnicas efectivas y open source para agrupar variantes de archivos de malware en sus respectivas familias.

A continuación, en el siguiente punto explicaré mis motivos para seleccionar este reto para mi TFG.

1.1. Motivación

La motivación que me hizo decantarme seleccionar este reto como propuesta para este TFG, fue aunar en este los dos conjuntos de capacidades tan distintas como son por un lado la Minería de Datos, Big Data, etc., capacidades que me motivaron a escoger la Intensificación de Computación, y la Ciberseguridad, la cual me ha hecho escoger opta-

tivas orientadas a este ámbito, como puede ser Auditoría de Sistemas de Información, Criptografía y Dispositivos y Redes Inalámbricos, y además de otras cuestiones. Por todo esto, a la hora de elegir el tema de mi TFG busque opciones que englobaran estos dos conjuntos de capacidades por eso cuando, se me ofreció la posibilidad de realizar este TFG sobre este reto no lo dude.

El motivo de esta elección desde el punto de vista de la ciberseguridad, es que se trata un tema muy habitual y muy interesante como es el tema de los malware, tanto que una compañía tan importante a nivel Internacional como es Microsoft decidió lanzar este reto como bien explican en su descripción. Esto hace que en este TFG sea necesario trabajar con la información obtenida mediante técnicas de reversing de las muestras de malware. Estas técnicas de reversing nos proporcionan información en lenguaje ensamblador y en lenguaje binario, siendo necesario manejar y familiarizarse con esta información, estos lenguajes y las posibles características que se puedan extraer de ellos y nos pueden ser de utilidad.

En cambio, en lo que respecta a las capacidades relacionadas con la Intensificación de Computación y conforme se puede deducir de la cifras que se indican en la descripción del reto por parte de Microsoft, nos encontramos ante un problema de Big Data. También, es necesario la utilización de técnicas de Minería de Datos para conseguir la clasificación de las distintas muestras en su correspondiente familia. Pero no solo eso, de las cifras que se observan también se puede llegar a concluir que debido a la gran cantidad de variantes dentro de una misma familia de malware, sea necesario la utilización de una gran cantidad de características para la correcta identificación de la familia a la que pertenece, haciendo así que nos encontremos ante uno de los mayores problemas a la hora de realizar procesos de Minerías de Datos como es la alta dimensionalidad. Este problema nos hace que sea necesario utilizar técnicas para reducir el número de características a un número que sea abordable y permita la clasificación en la familia de manera correcta.

Debido a todas estas consideraciones, cuando se me presentó la oportunidad de realizar este TFG me pareció tremendamente interesante, ya que me permitiría aprender muchas capacidades interesantes de estos dos grupos. Por ese motivo, fue que no dude en seleccionar este tema para mi TFG una vez se me propuso.

1.2. Objetivos

Según el reto que se va a llevar a cabo en este TFG, el objetivo principal es al creación de un sistema experto para la agrupación de las distintas muestras proporcionadas en la base de datos de malware en sus correspondientes familias.

Debido a que este objetivo es muy genérico, se puede subdividir en sub-objetivos, siendo estos:

- Familiarizarse con las seguridad informática.
- Estudio de las distintas familias de malware utilizadas en el reto. Esto es necesario para su conocer su comportamiento y ser capaz de identificar las posibles características y de ese modo seleccionar las mas apropiadas para estas familias de malware.
- Buscar y analizar las soluciones propuestas por otros participantes en el reto. Esto es posible ya que una de la condiciones del reto es que las soluciones sigan la filosofía open source. Esto me permite obtener ideas de los que van realizando otros equipos y la posibilidad de comparar los resultados con los obtenidos en mi solución.
- Estudiar y analizar distintos métodos para el procesado de fichero de texto. Esto es necesario porque la bases de datos proporcionadas para el reto se encuentran almacenadas dos ficheros comprimidos formados por ficheros de texto donde se almacenan las información de las muestras proporcionadas, por lo cual es necesario manejar estos ficheros para la extracción de características.
- Diseño e implementación de un algoritmo de extracción de las características que van a ser utilizadas para la identificación de las muestras proporcionadas para el reto.
- Estudio y análisis de distintas técnicas para el procesamiento de lenguaje natural. Es necesario ya que uno de los tipos de ficheros proporcionados contiene las instrucciones del malware desensambladas con IDA, lo que supone el manejo de las ocurrencias de palabras utilizadas en ensamblador junto otras como pueden ser las llamadas al sistema. Esto hace que sea necesario la utilización de técnicas que permitan la obtención de la probabilidad de estas palabras en todos los documentos proporcionados.
- Estudio de distintos sistemas para el almacenaje de grandes cantidades de datos. El motivo por el que es necesario este objetivo es porque tenemos que utilizar

una gran cantidad de información en todo momento, por lo cual no es posible mantenerlo en todo momento en memoria, haciendo obligatorio la utilización de algún sistema de almacenaje adecuado para grandes cantidades de datos.

- Estudio e implementación de técnicas para la reducción de la dimensionalidad. Esto, al igual que el punto anterior, es en gran parte debido a la gran cantidad de información disponible para su análisis, por lo cual es necesario para conseguir hacer la implementación abordable, sobre todo con los recursos que se van a utilizar.
- Implementación de un sistema experto utilizando técnicas de Minería de Datos y Aprendizaje Automático.
- Prueba del sistema experto construido con la bases de datos proporcionadas.
- Comparación de los resultados obtenidos con otras soluciones propuestas por otros equipos.
- Analizar la posibilidad de extender el algoritmo implementado a un escenario mas general de detección de malware. Esto sub-objetivo solo se analizara de manera teórica y se realizar ya que viene establecido en el reto su realización.
- Propuesta de posibles mejoras.

1.3. Estructura de la memoria

La memoria a partir de este punto se organizará del siguiente modo. En el Capítulo 2 se explicarán las técnicas y herramientas utilizadas durante el y algunos términos, necesarios para tender correctamente el contenido de este TFG. En el Capítulo 3 se explicará dos soluciones que fueron presentadas en su momento en el reto al que hacemos mención. La primera de estas soluciones es la mas simple de las que se comentarán tanto que es abordable con los recursos que dispongo. En cambio, la segunda y ultima solución se trata de una solución muy compleja haciendo que no sea abordable con los recursos que dispongo de ninguna de las maneras. En el Capítulo 4 se explicará el proceso que se ha llevado para construcción de mi solución para el reto que estamos tratando. En un primer paso, se explica los tipos de malware que forman el reto, y el proceso que llevaré acabo para la extracción de las características. A continuación, se explicará el código desarrollado y la decisión por las cuales se ha optado por esa opción. Para explicar el código, se utilizará como guía el código principal, con el cual se inicia el ejecución de la solución. En el capítulo 5 se explicarán los opciones con las que se ha ejecutado mi solución, también se explicará el el resultado de cada uno de

las etapas de la ejecución del código, mostrando y explicando los resultados obtenidos. Finalmente, en el Capítulo 6 se explicarán las conclusiones extraídas de los resultados obtenidos de la ejecución de mi solución. También, se compararán los resultados con los obtenidos por las soluciones expuestas en el Capítulo 3.

Capítulo 2

TÉCNICAS Y HERRAMIENTAS UTILIZADAS

Durante el desarrollo de este TFG, me he encontrado o he utilizado las siguientes herramientas y técnicas. A continuación, las enumero y explico antes de que sean utilizadas en los antecedentes o en mi propia solución, para una mejor comprensión de estos puntos.

1. Ngram es una secuencia de n características que pueden ser de varios tipos. Para este reto, utilizaremos ngrams binarios, estando formado un ngram por dos caracteres en hexadecimal a lo que ahí que añadir otro valor representado por ?? que representa cuando el valor binario no ha sido mapeado en memoria.
2. Sklearn es una módulo en Python para desarrollar modelos de aprendizaje automático construido sobre SciPy [21](#) y distribuido bajo licencia open source BSD license [\[5\]](#).
 - a) RandomForestClassifier es una implementación del algoritmo Random Forest que es un ensembler de tipo bagging [20](#) en el cual durante el proceso de selección de las muestras no solo se lección los elementos de manera aleatoria sino que selecciona de manera aleatoria un subconjunto de las características originales.
 - b) DictVectorizer es una función que transforma un diccionario array de diccionarios en una matriz de tamaño el número de elementos del array por el número de claves distintas que forman los diccionario, de este modo establece una relación entre la clave y la posición de la columna y guardando esa relación para todos los elementos, ya tuviese clave o no el elemento original [\[6\]](#).

- c) TF-IDF es la abreviación de term frequency – inverse document frequency en español frecuencia de término – frecuencia inversa de documento. Es una implementación del algoritmo con este mismo nombre. Se trata de un algoritmo para el procesamiento de lenguaje natural. Consiste en la utilización de dos técnicas, por un lado la frecuencia del término dentro de un documento y la frecuencia inversa del documento. para lo cual se utiliza la formula que se muestra a continuación [7].

$$p_i(t_j) = f_{ij} * \log \left(\frac{N}{n_j} \right)$$

- d) Non-negative Matrix Factorization es una implementación de este algoritmo. Este originalmente, fue diseñado para el tratamiento de imagen y consiste la unión de dos tecnicas Vector Quantization, la cual compara un imagen con un conjunto de imagenes y selecciona la mas parecida comparando las características, y Principal Components Analysis el compara un conjunto de características con una imagen y establece el nivel de acierto respecto al original. Uniendo estas dos técnicas conseguimos que se compara cada una de las carecterísticas de un conjunto con el original y recibiendo el nivel de acierto respecto a esta [8] [9].
- e) Linear Support Vector Classification es una implementación del modelo LI-BLINEAR 14 [10].
3. Numpy es un paquete fundamental para el cálculo científico con Python y esta distribuido bajo una licencia open source BSD License [11].
 4. Pickle es un módulo de Python que implementa un protocolo binario para la serialización y deserialización de una estructura de objetos de Python [12].
 5. CPickle es un módulo de Python2 que al igual que pickle 4 es utilizado para la serialización y deserialización del objetos de Python [13].
 6. Glob es un módulo de Python mediante el cual se puede buscar todos los nombre que coinciden de acuerdo con un patrón especifico como el que se utiliza por las Shell de Unix [14].
 7. Subprocess es un módulo de Python que permite generar nuevos procesos, conectarlos a ellos mediante pipes de entrada, salida y error, y obtener el resultado del código [15].
 8. Funciones Linux. Las funciones linux que se utilizan durante este TFG son:
 - a) Echo es una función que muestra una linea de texto [16].

- b) Cat es una función que concatena ficheros e imprimé en el dispositivo de salida estándar [17].
 - c) Wc es una función que imprimé el número de líneas, de palabras y de bytes para cada fichero [18].
 - d) Grep es una función que imprimé líneas que coinciden con un patrón [19].
- 9. Pandas es una librería que proporciona eficiencia, estructuras de datos fáciles de utilizar y herramientas de análisis para el lenguaje de programación Python y está distribuido bajo una licencia open source BSD License [20].
- 10. DataFrame es la estructura de datos principal de pandas 9. Es una estructura de datos bidimensional de tamaño variable, potencialmente heterogénea y tabulada con etiquetas en las filas y las columnas [21].
- 11. Matriz Dispersa es una matriz donde muchos de sus elementos son ceros [22].
- 12. Os es un módulo de Python que proporciona una manera compatible de utilizar funciones propias del sistema operativo [23].
- 13. Joblib es un paquete en Python con un conjunto de utilidades que proporciona el manejo de hilos de manera ligera [24].
- 14. LibLinear: Una librería para completa clasificación lineal es un modelo de predicción fácil de usar, mediante el cual calcula una línea que divide el conjunto de entrenamiento para obtener el mejor nivel de acierto [25].
- 15. Hickle es un paquete en Python que al igual que pickle 4 y cPickle 5 es utilizado para la serialización y la deserialización de objetos de python [26].
- 16. XGBoost es la abreviación de ‘Extreme Gradient Boosting’, el cual es una implementación basado en el Gradient Boosting 17 [27].
- 17. Gradient Boost Tree es un ensembler que a diferencia del bagging 20 usa los el conjunto de datos original. Con este conjunto de datos, construye un árbol de decisión, para a continuación repetir el proceso ponderando las características respecto a los resultados obtenidos del árbol de decisión obtenido. Este proceso se repite tantas veces como modelo se le establezca al ensembler. Una vez concluidos todos los modelos, para obtener el acierto del modelo se pondera el resultado de todos los modelos en función de su acierto individual.
- 18. MLogloss es la métrica utilizada en este reto de Kaggle y la versión para múltiples valores de clase de la pérdida logarítmica. Esta es el logaritmo de la función de probabilidad para una distribución aleatoria de Bernoulli, pudiendo observar la

formula a continuación. Para esta métrica, se tiene que intentar reducir su valor siendo mejor cuanto mas próximo a cero [28] [29] .

$$logloss = -\frac{1}{N} \sum_{i=1}^N \sum_{j=1}^M y_{ij} \log(p_{ij})$$

19. Random es un módulo de python que implementa un generador de números pseudo-aleatorios para varias distribuciones [30].
20. Bagging es un un tipo de ensembler (conjunto de modelos), en el cual se obtiene n muestra con o sin reemplazo de igual tamaño o no del conjunto de datos iniciales y se construyen n modelos, y el resultado de este modelo es la media de los n modelos construidos.
21. Scipy es una librería en python que proporciona de manera amigable y eficiente algunas calculos numéricos como pueden ser el calculo de integrales numéricas, algoritmos de optimización, etc., la cual esta distribuido bajo una licencia open source BSD License [31]. Uno de los módulos que proporciona esta librería es Stats mediante el cual el usuario puede utilizar un gran número de distribuciones de probabilidad además de funciones estadísticas [32].
22. MongoDB es una base de datos open source de documentos que proporciona alta eficacia, alta disponibilidad y autoescalado. Que se trate de una base de datos de documentos supone que los registros sean objetos BSON, el cual es muy similar JSON 23 siendo la principal diferencia que se almacena el contenido en binario [33].
23. JSON (JavaScript Object Notation) es una formato estándar. Es un formato de datos intercambiables ligero, compatible con muchas lenguajes y almacena la información en objetos de manera desordenada y sin estructura predefinida y utiliza un sistema de clave valor para almacenar las informacion [34].
24. Docker es una sistema de virtualización ligero basado en contenedores, mediante el cual permite el aislado de la ejecución de todo o de parte del código del que disponemos. Además, cada contenedor ha de incluir todas las librería, paquetes, etc. que sean necesarias para su correcta ejecución [35].
25. RE es un modulo de Python mediante el cual se puede utilizar expresiones regulares para realizar comprobaciones [36].

Capítulo 3

ANTECEDENTES Y ESTADO DE LA CUESTIÓN

Para el reto de Kaggle que se ha seleccionado para este TFG, se he realizado una búsqueda para intentar encontrar soluciones presentadas del mismo. De esta búsqueda, se ha conseguido encontrar varias soluciones presentadas. A continuación, se explicaran las soluciones presentadas que se han encontrado.

3.1. Solución 1

Esta primera solución que se va a comentar, fue realizada por Vishnu Chevli [37] y en la clasificación del Reto de Kaggle obtuvo un resultado en el ranking público de 0,023121984 en la posición 90 y en el ranking privado 0,018856579 en la posición 72.

Esta solución se desarrolló para que pudiera ser ejecutada de manera indistinta en Python 2 y Python 3, es muy sencilla y consiste en:

En primer lugar, se descomprimen las dos bases de datos que se proporcionan. Estas bases de datos contienen dos tipos de ficheros que contienen la información obtenida de IDA al desensamblar las muestras en dos formatos, siendo estos formatos ASM para los ficheros con extensión .asm y binario en formato hexadecimal para los ficheros con extensión .bytes. Para esta solución solo se utilizarán los ficheros con extensión .bytes. Después, estos ficheros son nuevamente comprimidos en formato gzip de manera separada y separando las bases de datos en dos carpetas distintas. El motivo de realizar esto es para ahorrar espacio en disco, ya que el espacio de la base de datos completas con todos los tipos de ficheros sin comprimir es de aproximadamente 1 Tb.

El siguiente paso, consiste en extraer las características de los ficheros que se han generado en el paso anterior. Esta extracción consiste en generar un array donde cada posición corresponde a cada uno de los posibles uno ngrama 1, formado por dos caracteres en formato hexadecimal, incluyendo el ngram '??' que indica que el valor correspondiente no esta mapeado, y contar las veces que aparece cada uno de ellos en el fichero, para concluir, guardando los resultados en un fichero csv de manera separada para cada una de las bases de datos. Este proceso ser realizara de manera conjunta para las dos bases de datos y de manera paralela para 2 ficheros.

El siguiente paso, es construir el modelo que para este caso es un RandomForest-Classfier 2a. Para ello, lo primero se tiene que realizar es obtener la clase para cada uno de los fichero, para lo cual se tiene que abrir y recorrer el fichero 'trainLabels.csv' que se proporciona y almacenar la información del mismo en un diccionario que posteriormente se utilizará. A continuar, se crea una matriz con numpy 3 de tamaño el numero de ficheros de la base de datos de train por el número de uno ngramas 1 mas uno adicional para la clase, en este caso $10868 * 258$. Estos datos son obtenidos del fichero csv correspondiente a la base de datos de train para los valores de los uno ngramas 1 y del diccionario creado anteriormente para la clase. Después, se crea el modelo con los valores por defecto, excepto la semilla que utiliza 123, el número de hilos que utiliza 5 y el nivel de información que muestra que utiliza la mas detallada 2. Por ultimo, solo queda entrenar el modelo pasando le la matriz de numpy 3 separando las características y la clase, o lo que es lo mismo la matriz sin la ultima columna y la ultima columna por separado, siendo esta ultima columna la clase.

Para terminar, solo nos queda realizar la previsión para la base de datos de test. Para ello, ahí que repetir el proceso de recuperar las características creando una matriz con numpy 3 pero en este caso sin añadir un columna para la clase, también se crea un array donde se almacena el nombre del fichero en el mismo indice que la fila de la matriz, esto es así porque la predicción ha de ser identificada con es valor. Lo siguiente, es realizar la predicción para la base de datos de test. Para concluir, escribiendo los resultados en un fichero comprimido en el que guarda la información con el formato que se nos proporciona en el ejemplo, siendo este el id del fichero seguido de la probabilidad que sea cada una de las clases separado todo por comas.

3.2. Solución 2

Esta segunda solución que se va a comentar, fue realizada de manera conjunta por Mikhail Trofimov, Dmitry Ulyanov y Stanislav Semenov [38] y en la clasificación del Reto de Kaggle obtuvo un resultado en el ranking público de 0,005984430 en la posición 14 y en el ranking privado 0,003969846 en la posición 3.

Esta solución es mucho más compleja como se puede suponer de la posición que obtuvo en el ranking, de hecho según sus creadores utilizaron una máquina con 16 cores y 120 Gb de RAM para su ejecución. Ahora, se analizará en qué consiste esta solución:

Lo primero que realizar es crear los directorios, para ello existe un script en bash llamado `'create_dirs.sh'` para generar los directorios necesarios. A continuación, toca ejecutar otro script en bash llamado `'main.sh'` para ejecutar esta solución, siendo el primer paso, la extracción de las características de los ficheros con extensión `'.asm'` que posteriormente se van a utilizar para construir el modelo. Pero antes de eso, ahí que establecer en el fichero llamado `'set_up.py'` los parámetros fijos para esta solución en variables, como pueden ser el número de hilos, y las distintas direcciones de las carpetas con las que trabaja la solución. A continuación, se comentará este proceso de extracción que consisten en:

El primer paso en el proceso de extracción de características de los ficheros con extensión `'.asm'`, consiste en contar las veces que aparece cada una de las secciones en cada uno de las muestras de las bases de datos y la suma total del número de ocurrencias de cada sección para cada muestra. Para llevar a cabo esto, se recorre cada fichero con extensión `'.asm'`, siendo pasado el nombre de éste por parámetro a la función, y de cada línea, se tiene que coger la primera palabra, estando esta delimitada con dos puntos, y se mantiene en un diccionario el conteo de ocurrencias que aparece cada una de las secciones y el número total de ocurrencias de cada sección. Para proporcionar persistencia y no tener que almacenar toda la información en memoria, se guarda en un archivo serializado con el paquete `pickle` [4](#) el diccionario obtenido para ese fichero. Para obtener el nombre del fichero que se le pasa para hacer la extracción, se consigue la dirección completa de los ficheros con extensión `'.asm'` en cada una de las carpetas donde se almacenan las bases de datos, usando para esto la función `glob` del paquete `glob` [6](#). Posteriormente, a los elementos de esta lista se les aplica una función `map` para dejar solo el nombre del fichero que es el identificador de la muestra. Para terminar, se crea una lista de procesos cuya longitud es el número de hilos establecidos. Esto es así para realizar el procesamiento de las muestras de manera paralela. A estos procesos, se les pasa una sublista de los ficheros y ejecuta una función `worker` que se encarga de

procesar cada uno de los elementos de esa sublista y realizar la extracción comentada anteriormente. La manera de formar estas sublista es mediante el resto del índice de la posición en la lista original entre el número de hilos que se ejecutan, siendo este el índice del proceso en el cual la muestra tiene que formar parte de su sublista.

El segundo paso en el proceso de extracción de características en los ficheros con extensión '.asm', consiste en llevar un conteo de las líneas que posee el fichero de la muestra y en cuantas de estas, aparece los elementos de un conjunto establecido de instrucciones asm. Este conjunto de instrucciones que se utiliza en la solución, se puede observar en la figura 3.1, está formado por 23 instrucciones ASM, siendo estas las mas habituales, almacenadas en una lista de strings.

```
specter = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 'sub', 'inc', 'dec', 'add',
          'imul', 'xchg', 'or', 'shr', 'cmp', 'call', 'shl', 'ror', 'or', 'rol', 'jnb']
```

Figura 3.1: Lista de instrucciones ASM manejadas.

Se comienza por almacenar en el fichero llamado 'fnames' el nombre de la muestra, usando la función echo [8a](#) de linux, que se le pasa a la función, siendo este nombre generado de la misma manera que en le paso anterior. Realizarlo de está manera implica que no sea posible la paralelización de este proceso, ya que la línea dentro de cada fichero que se va a utilizar estable la muestra a la que se hace mención. Se continua realizando el conteo de la líneas del fichero con extension '.asm' de la muestra. Para llevar acaba el proceso de conteo, se utiliza la función call del paquete subprocess [7](#) para de esa manera, utilizar las utilidades de linux. En este caso, para contar el número total de líneas del fichero de muestra se utiliza la función cat [8b](#) para recorrer el y se le pasa esa información mediante una tubería a la utilidad wc [8c](#) con el parametro '-l', la cual cuenta la líneas que se le han pasado y añade esa información en un fichero llamado 'line_count' en la posición correspondiente para esa muestra. A continuación, se realizará el conteo de la líneas en las que aparece cada una de las instrucciones en esa muestra. Para conseguir este objetivo, hace uso de la función grep [8d](#) para seleccionar las líneas dentro de la muestra que contiene la instrucción correspondiente, para posteriormente pasárselo mediante una tubería a la función wc [8c](#), igual que en el paso anterior, y añadir la información en el fichero correspondiente a la instrucción en la posición que le corresponde. Este paso se repite, para cada una de las instrucciones establecida. En este proceso, se realizar de manera serizalizada para todos los ficheros de la base de datos y en el caso de ser el primer fichero y no existir los ficheros todos ellos son generados cuando se añade el primer elemento.

El siguiente paso en el proceso de extracción de características en los ficheros con

extensión '.asm', consiste en seleccionar de la muestra las líneas en las que aparece la cadena de caracteres '__stdcall' y almacenarlas en un fichero por cada muestra. Para llevar esto a cabo, hace uso de la función `call` del paquete `subprocess` 7 y de la función `grep` 8d, igual que en el paso anterior, pero en esta ocasión se almacena por nombre de la muestra, la cual se le pasa a la función del mismo modo que en los pasos anteriores. Esto implica que en este caso si sea posible paralelizar el proceso de extracción de características, por lo cual se realiza de la misma manera que se llevo a cabo en el primer paso, creando una sublista que procesara un proceso para todas las muestras de las bases de datos.

Por ultimo en el proceso de extracción de características en los ficheros con extensión '.asm', consiste en seleccionar de las muestras las líneas en la que aparece la cadena de caracteres 'FUNCTION'. Como se puede observar, consiste en repetir el proceso anterior una nueva cadena de caracteres.

Una vez terminado el proceso de extracción de características de los ficheros con extensión '.asm', toca preprocesar la información obtenida de los ficheros con extensión '.asm'. Para ello, lo primero que se realiza es obtener dos dataframes a partir los ficheros 'trainLabel.csv' y 'sampleSubmission.csv', los cuales son proporcionados en el reto, con `pandas` 9. Éstos contienen los identificadores junto con otra información de las muestras de las dos bases de datos train y test respectivamente, y se usarán para que durante el proceso de preprocesado todas las características obtenidas de las muestras guarden el mismo orden, hecho por el cual esto implica que este procedimiento no puede ser paralelizado. Además, los procesos que se comentara a continuación se repite para estos dos dataframes. Este procedimiento consiste en:

En primer lugar, se tratan las secciones obtenidas, para ello se recorre el dataframe fila a fila, y para el campo `Id` se abre el fichero de las secciones calculado correspondiente con el paquete `cPickle` 5, el cual se trata de un diccionario en python, por lo cual se recorre y si pertenece a un conjunto establecido, el cual se puede observar en la figura 3.2, se incluye en otro diccionario, para terminar guardándolo en la posición del array que viene establecida por el dataframe. En este primer paso, cuando se trata del dataframe de la base de datos de train también se crea un array, en el cual se almacena la etiqueta correspondiente de la muestra en el índice establecido por el dataframe. Para continuar, transformando el array de diccionarios en una matriz, para esto usa el modelo `DictVectorizer` 2b del paquete `sklearn` 2 con el parámetro `sparse` a `false` para que no almacenar los datos en una matriz dispersa 11. Este modelo se entra con el array generado con el dataframe de la base de datos de train y posteriormente se transforman los dos arrays en su correspondiente matriz. Con estas matrices se procede a calcular

la entropía correspondiente a cada una de ellas. Terminando así el tratamiento de las secciones.

```
section_whitelist = set(['.bss', '.data', '.edata', '.idata', '.rdata', '.reloc',
                        '.rsrc', '.text', '.tls', 'bss', 'code', 'data', 'header'])
```

Figura 3.2: Lista de secciones ASM manejadas.

En segundo lugar, consiste en obtener el tamaño de los archivos con extension '.asm' y '.bytes' para después obtener también el ratio entre estos. De este modo, se crea una matriz con la longitud del dataframe y dos columnas una para cada fichero, donde se almacena con la ayuda de la función `getsize` del paquete `os.path` ?? el tamaño estos correspondientes a la muestra en la posición establecida por el dataframe. A partir de esta matriz, se genera otra nueva con una sola columna, la cual es el resultado de hacer la división de float, ya que para garantizarlo se multiplica en numerador por 1,0, del tamaño obtenido del fichero con extensión '.bytes' entre el tamaño obtenido del fichero con extensión '.asm', concluyendo así este paso.

En tercer lugar, consisten en almacenar la información obtenida de las instrucciones asm en una matriz de numpy 3 y el calculo de la entropía de la misma. Para ello, primero se tiene que abrir y recorrer el fichero 'fnames' que se genero en el proceso de extracción de características para almacenar en un diccionario donde se almacena en el valor de indice la muestra se almacena un diccionario con los valores obtenidos para las instrucciones manejadas, vease la figura 3.1, para esa muestra. Una vez se recuperado la información, se procede a almacenar la en una matriz de numpy 3 conforme se va haciendo, donde el indice que el dataframe establece la fila y en indice del array de figura 3.1 indica la columna. Una vez realizado esto se procede se calcula la entropía de cada característica. Este proceso se repite para los dos dataframe y con esto se termina este paso.

El siguiente paso, consiste recuperar la información del número de lineas de los ficheros con extensión '.asm' y almacenarlo en un diccionario. Esto se realiza de este modo debido a que se tiene que recorrer el fichero 'fnames' para obtener el Id de valor almacenado en el fichero 'line_count'. Una vez hemos recuperado la información se procede a almacenarla en una matriz con una sola columna en el orden establecido y por el dataframe y se calcula la entropía para todos los valores de la misma. Este proceso se repite de igual manera para los dos dataframe, terminando lo así.

A continuación, toca obtener la información de las llamadas a las funciones, estando estas en las lineas seleccionadas en las que aparecía la cadena '__stdcall' almacenadas

en el proceso anterior. Para esto, se procede a recorrer los ficheros obtenidos anteriormente en el orden establecido por el dataframe y en un primer paso, partir esa linea para obtener el nombre de la función para a continuación concatenar los y almacenarlos en un string separadas por espacios para cada muestra, el cual es almacenada en un array en la posición establecida por el dataframe. Una vez obtenido este array para los dos dataframe, se procede a crear un TfidfVectorizer [2c](#) mediante el cual se obtienen como máximo las 10000 características mas significativas según este modelo. Para ello, se entrena el modelo con la unión de los dos array para posteriormente, transformar estos una matriz dispersa [11](#) con la probabilidad de como máximo las 10000 características mas significativas para cada una de las muestras. Una vez obtenida esta matriz, se le realiza una factorización de matriz no negativa [2d](#), para de ese modo concluir obteniendo las 10 características mas representativas representativas. Por lo cual, se construye en modelo NMF para 10 componentes y se entrena con una matriz dispersa [11](#), consistente en la unión de las dos matriz obtenidas en el paso anterior, para concluir transformado cada una de de las matrices y así obtener una matriz dispersa con la probabilidad factorizada de las características de las dos bases de datos.

Para terminar con el proceso de preprocesado de los fichero con extensión '.asm', solo nos queda tratar las llamadas al sistema, siendo las lineas seleccionados en las que aparecen la cadena 'FUNCTION'. Este proceso es idéntico al llevado a cabo en el paso anterior salvo que se cogen los nombres de las funciones en la que en la linea aparece la palabra 'PRESS'.

Para concluir, se almacena en una matriz de numpy [3](#) donde se incluye en columnas la información preprocesada de la secciones, el ratio del tamaño, las instrucciones, el tamaño de los ficheros, la entropía de las secciones, las características obtenidas de las llamadas a las funciones y por ultimo las características de las llamadas la sistema. Este proceso se repite de igual manera para las dos bases de datos. Por ultimo se almacena en un fichero serializado con joblib [13](#) llamado 'X_basepack' guardando las dos matrices en una tupla. Con esto se termina el proceso de preprocesado de las características de los ficheros con extensión '.asm'.

Llegados a este punto, el siguiente paso consiste en extraer las características de los ficheros con extensión '.bytes'. Este paso comienza por, crear un diccionario mediante el cual traducir un n-grama de dos caracteres en hexadecimal a decimal. A continuación, se abre el fichero de la muestra y cada linea se transforma en un array teniendo como delimitador el carácter espacio en blanco ' '. De los arrays generados, se selecciona todos los elementos salvo en primero, que corresponde con la dirección en memoria, para a continuación transformar a decimal los n-gramas con el diccionario

generado en el primer paso, y concatenar todos en un nuevo array donde se encuentran todos los n-gramas en formato decimal del fichero ordenados. Una vez tenemos este último array, lo recorremos en grupos de 4 n-gramas, para continuar por transformar el 4 n-grama en un índice número y guardar en un diccionario las ocurrencias de estos índices en el fichero. Para concluir, transformando la información almacenada en diccionario en dos arrays, uno para los índices y otro para las ocurrencias del mismo. Para ello, recorre las keys del diccionario y almacena en la misma posición en ambos arrays el índice y el valor de este, respectivamente. Para terminar, almacena en un archivo serializado con pickle [4](#) una tupla con estos dos arrays. Este proceso es llevado a cabo para las dos bases de datos. Para ello, se obtienen los ficheros que los ficheros en la ubicación de la base de datos con la función glob de glob [6](#). Una vez tenemos la lista de ficheros de la base de datos, se procesan de manera paralela todos ellos haciendo uso de todas las CPU disponibles utilizando la función parallel ?? del paquete joblib [13](#).

En este momento, se comienza con el preprocesado de la información de los ficheros con extensión '.bytes', siendo este primer paso contar las veces que aparece cada n-grama representado por un índice en formato decimal en los primeros 4000 ficheros. Para ello, se obtienen los ficheros de características obtenidas en el paso anterior con la función glob del paquete glob [6](#). A continuación, se genera un array de numpy [3](#) de ceros con tamaño el número de 4 n-gramas posibles, o lo que es lo mismo 257^4 . Para continuar, obtener la información obtenida para a continuación sumar uno en el array de numpy [3](#) generado en los índices que posee la muestra. Una vez, procesados 4000 de estos ficheros se termina guardando los resultados en un archivo serializado con la función pickle del paquete cPickle [5](#).

El segundo paso que se realiza en este preprocesado, de la información conseguida del paso anterior selecciona la posición de los que hayan obtenido un valor superior a 10 y se crea un diccionario donde se almacena el índice original y su posición en el nuevo índice. Con esta información, el siguiente paso consiste en modificar las características obtenidas seleccionando solo las que forman parte del diccionario que se acaba de obtener. Para ello, se recuperan las características obtenidas, recorriéndolas y si pertenecen al diccionario de características frecuentes se incluye en los nuevos arrays, ya que se almacena mediante dos arrays uno primero para almacenar las características y otro segundo para almacenar el valor de esa característica en la misma posición, del mismo modo que se almacena en un primer momento. Este proceso se realiza para todos los archivos que se han generado en la extracción de características. Además, se paraleliza con la función parallel ?? del paquete joblib [13](#) para 15 de los 16 CPUs disponibles.

Por último, se procede a recuperar las características y almacenarlas en una matriz

dispersa 11, para a continuación pasársela al modelo Linear Support Vector Classification 2e para reducir los outliers que hayan detectado. Para ello, en primer lugar se recuperan los datos en el formato correcto, por lo cual se abre el archivo 'trainLabels.csv' correspondiente a la base de datos de train, el cual se proporciona. Llevando a cabo esto con la función `read_csv` del paquete pandas 9, obteniendo un dataframe, el cual recorreremos y obtenemos los valores que se han generado en el paso anterior. Estos datos están formados por los valores los cuales se incluyen en un array llamado 'data', los índices de los n-gramas que se incluyen en un array llamado 'índices' y por último en un array llamado 'ptr' donde se almacena el índice donde empieza cada una de los valores de cada una de las muestra, para conseguir esto, se acumula la longitud de los valores recuperados en una variable llamada 'cur_bound' y se añade el valor de esta variable en cada momento en el susodicho array. Una vez formados estos arrays se guarda en una tupla formada por arrays de numpy 3 de los tres array obtenidos en un archivo serializado con el paquete hickle 15 y a continuación, se crea una matriz dispersa con la función `csr_matrix` ?? del paquete 11, la cual también es guardada en un archivo serializado con el paquete joblib 13.

A continuación, se repite el proceso anterior para la base de datos de test, por lo cual se genera un dataframe a partir del fichero 'sampleSubmission.csv' proporcionado con la función `read_csv` de paquete pandas 9 y repitiendo con este el proceso de recuperación y almacenaje en una matriz dispersa de las características comentado en el paso anterior.

Llegado a este punto, ahí que construir y entrenar el modelo Linear Support Vector Classification 2e, por lo cual, se recupera la matriz dispersa de las características generada en el paso anterior para la base de datos de train, mediante el paquete joblib 13. También se obtiene de nuevo el dataframe de la base de datos de train, mediante la función `read_csv` del paquete pandas 9 leyendo el fichero 'trainLabels.csv' proporcionado. A continuación se crea el modelo Linear Support Vector Classification 2e, con los siguientes parámetros `penalty=l1` con la cual se establece que la penalización utiliza en el modelo sea la 'l1' del módulo liblinear 14, `max_iter=20` con el que se establece el número máximo de iteraciones que va a realizar el modelo, `dual=False` con la cual se establece que se utiliza el problema de optimización dual ?? y `verbose=1` con el que se establece que el nivel de detalle de información que se muestra mientras se crea el modelo siendo que no muestre información. Para después, entrenar el modelo con la matriz dispersa y las etiquetas correspondientes a las muestras, obtenidas del dataframe y guardando el modelo una vez entrenado en un archivo serializado con el paquete 13. Para terminar, transformando la matriz a partir del modelo aprendido eliminar los outliers, en caso de que la matriz no se haya recuperado, en el caso de la matriz

dispersa de las características de test, primero se obtiene la misma mediante el paquete `joblib` 13. Además, las dos nuevas matrices son guardadas en un archivo serializado con el paquete `joblib` 13, terminando de este modo el preprocesado de las características.

El siguiente paso en esta solución, es reducir la dimensionalidad de las características de los ficheros con extensión `'.bytes'`. Para ello, se recuperan los datos preprocesados en el paso anterior para las base de datos de train y de test y transformando la matriz dispersa en una matriz, también se obtiene mediante la función `read_csv` del paquete `pandas` 9 el dataframe de la base de datos de train del archivo `'trainLabels.csv'` proporcionado. A continuación, se parte la base de datos de train almacenada en la matriz y la etiquetas correspondientes en dos grupos, uno de train y otro de test de un 70 % y 30 % respectivamente. Estos dos grupos se subdividen en características representada con un `X` al principio y etiquetas representada con un `y` al principio, quedando almacenadas en las siguientes variables respectivamente, `X_train`, `X_test`, `y_train`, `y_test`. El siguiente paso, consiste en construir el modelo `Random Forest Classifier` 2a con los parámetros `n_jobs=-1` que indica que se usen todas la CPUs y `n_estimators=1000` que establece el número de árboles de decisión que formaran el modelo y entrenándolo con las variables `X_train` y `y_train`. A continuación, de la característica `features_importances_` del modelo se seleccionan las que tengan un valor superior a 0,0014. Para terminar, construyendo una tupla formada por las dos matrices de los datos preprocesado y que formen parte de las características seleccionadas, terminando por guardar esta tupla en un archivo serializado con el paquete `cPickle` 5. Terminando así el proceso de selección de variables de los archivos con extensión `'.bytes'`.

En este punto, es cuando se comienza a la construcción de los modelos necesarios para realizar la predicción en el reto. Este proceso se puede dividir en 4, siendo estos, la creación un primer modelo base, seguido de otros dos modelos que serán la base de la predicción final y por ultimo, una ponderación de los resultados obtenidos por estos dos últimos modelos. A continuación se explicará cada uno de pasos.

El primero paso, conforme se ha indicado es construir un modelo inicial que servirá de apoyo en para la predicción posterior de otro modelo. Para ello, se comienza por obtener los datos generados tras sus correspondientes preprocesados y selección de variables de los ficheros con extensión `'.asm'` y `'.bytes'`, para a continuación, juntar los en una matriz llamada `'Xtrain'` y `'Xtest'` respectivamente para la base de datos de train y test, haciendo uso de la función `hstack` del paquete `numpy`, esto es posible ya que todas las informaciones han sido almacenadas según las posiciones establecidas por el dataframe correspondiente a cada base de datos. También se obtiene la etiqueta de clase de las muestras pertenecientes a la base de datos de train, para ello se obtiene

el valor de las clase del dataframe del fichero 'trainLabels.csv' proporcionado con la función `read_csv` del paquete `pandas` 9 para almacenarlo en una array de `numpy` 3 restándole 1 para las etiquetas comiencen en el valor 0 y guardándolas en una variable llamada 'ytrain'.

Una vez contamos con los datos necesarios el siguiente paso consiste en establecer los parámetros necesarios y construir el modelo, siendo este un XGBoost ???. Los parámetros son guardados en un diccionario transformar los item que la forman en una lista almacenándolo en una variable llamada `plst`. Los parámetros que se establecen son `booster=gbtree` con el que se establece que el modelo boost que se creará sea un Gradient Boost Tree 17, `objective=multi:softprob` con el que se establece que el resultado sera una matriz donde para cada elemento se mostrara la probabilidad de cada una de las posibles etiquetas de clase, `num_class=9` que establece que el número de etiquetas para clasificar la clase ese de 9 según se establece en el reto, `eval_metric=logloss` con el cual se establece que la métrica a utilizar es `logloss` 18, `scale_pos_weight=1.0` con el que se establece que el control del balance de los pesos positivos y negativos que se utiliza habitualmente para desequilibrar la clase ??, `bst:eta=0.3` con el cual se establece un peso en tanto por 1, en este caso 0,3, con el que se ponderan los pesos que va generando durante la construcción del modelo para establecer la importancia del modelo construido según su acierto a la hora de predecir las muestras proporcionadas al modelo, con el valor que se le ha proporcionado hace que el comportamiento del modelo sea conservador teniendo mas importancia las características ya aprendidas que las nuevas características, `bst:max_depth=6` con el cual se establece la profundidad máxima de los árboles que construirá el modelo, `bst:colsample_bytree=0.5` con el que se establece el porcentaje en tanto por 1 el número de características que van a ser seleccionadas para construir el árbol de decisión en cada iteración, `silent=1` con el cual se establece que no se muestre información durante la generación del modelo y `nthread=16` con el que se establece el número de hilos que se usarán para generar el modelo. Además, aparte de los parámetros establecidos en el diccionario también se establecen una variable llamada `num_round=100` que establece el número de rondas que realiza el modelo y una lista vacía llamado `watchlist` que establece que no se revisa los resultados del modelo con un conjunto de muestras que no es utilizado en el entrenamiento.

Llegado a este punto, se genera un array con los índices de las muestras y una matriz de `numpy` 3 de ceros con tamaño el número de muestras a clasificar por el número de clases posibles, en este caso 9. Para continuar creando 20 modelos, comenzando por asignar al diccionario de parámetros el valor de la semilla 'seed', siendo este el número del modelo añadiéndole uno más, y transformando los elementos en una lista para pasárselo al modelo. A continuación, se genera un nuevo array que corresponderá

a los índices de las muestras de la base de datos de training que se van a utilizar en el modelo. Para lo cual, lo primero se genera una permutación del array de índices de las muestras que se ha generación al inicio de este punto, haciendo uso de la función `sample` del paquete `random` 19, para a continuación, añadir de manera aleatoria índices del array permutado hasta dejar un array de 8 veces el tamaño de la base de datos.

El siguiente paso, es crear dos `DMatrix`, una para cada base de datos, que es utilizada por pasar al modelo `XGBoost` ?? los datos. La `DMatrix` de la base de datos de Train se construye seleccionando los valores de las características de la matriz `'Xtrain'` y de la etiquetas de `'ytrain'` de esa base de datos con los índices creados en el paso anterior y en el caso de la base de datos de test, la matriz `DMatrix` se construye únicamente pasándole los valores de `'Xtest'`. A continuación, se construye el modelo `XGBoost` ?? propiamente dicho, para lo cual se le pasan los parametros en forma de lista la Matriz `DMatrix` correspondiente a la base de datos de train, las etiquetas de los índices seleccionados, el numero de rondas establecido en la variable `num_round`, y los valores con los que se comprobará establecido en la variable `watchlist`.

A continuación, toca realizar la predicción de este modelo para los valores de la base de datos de test almacenados en la `DMatrix` y reordenarlos para que se ajusten a la matrix de resultados que se genero, donde se acumularan los resultados de todas la predicciones, para a continuación, obtener la media de todos los modelos generados en este primer proceso.

Los resultados obtenidos serán guardados en un fichero llamado `'release0.csv'`, para lo cual se abre el fichero `'sampleSubmission.csv'` proporcionado para este reto, con la función `read_csv` del paquete `pandas` 9. Para a continuación, asignar los resultados obtenidos y terminar guardándolo en el fichero ya indicado con la ayuda de la función `to_csv` del paquete `pandas` 9 terminando así este primer modelo, siendo este una implementación propia de un algoritmo de Bagging 20 donde el modelo base es un `XGBoost` ??.

El siguiente paso, consiste en construir el primero de los modelos que se utilizarán para obtener la predicción. Para lo cual, se obtienen los datos obtenidos de los ficheros `'bytes'` de los ngramas 1 preprocesados pero sin reducir, los cuales son dos matrices dispersas por lo cual son transformadas en dos arrays, los cuales se llaman `Xtrain` y `Xtest` respectivamente con la bases de datos de train y test. También se recupera la etiqueta de la clase del mismo modo que se realizó en le modelo anterior. También los parámetros son los mismos que en el modelo anterior, con la salvedad que el número de rondas ahora es 150 en lugar de los 100 que se utilizo en le modelo anterior. La

construcción del modelo también es muy parecida siendo todo igual salvo que no se amplía la base de datos de train para construir el modelo. Para concluir, se guardan los resultados de la predicción en un fichero 'release0.5.csv' de la misma manera que en el modelo anterior, terminando así este modelo.

Ahora, solo queda construir el ultimo modelo, para lo cual, se recuperan los mismo datos y de la misma manera que en el primero de los modelos. Además, también se recupera la predicción realizada por dicho modelo, para realizar esto se hace uso de la función `read_csv` del paquete `pandas` 9. El siguiente paso, consiste en array de `numpy` 3 según corresponda respectivamente a la etiquetas de la clase y el resto de los datos, quitando si aparece el nombre de la muestra. Se continua, creando un array donde se almacena distribuciones de probabilidad para cada una de la muestras de la base de datos de test, usando para ello la función `rv_discrete`, mediante la cual se genera una distribución de probabilidad discreta, del paquete `stats` 21 dentro de `scipy` 21, al cual se le pasa las posibles etiquetas y la predicción para cada una de estas etiquetas de cada muestra. Después, se generan 10 particiones de tamaño $\frac{1}{10}$ de la base de datos de test donde en cada partición un de las partes es seleccionada para test sin ser repetidas y las restantes para train, donde el contenido son los indices de correspondientes a las muestras, haciendo uso de `KFold` y se realiza una copia del array de predicción generado al principio de este paso. Seguidamente para cada una de las partes generadas, se procede a añadir a los datos de la base de datos de train la parte seleccionada para cada caso de train de la base de datos de test, utilizando la función `concatenate` del paquete `numpy` 3, seleccionando los datos correspondientes a la parte de test de esta partición y inicializando un array llamado `pr_vals` con un array vacio donde se irán añadiendo la predicción de cada partición. A continuación, se construyen 20 modelo para cada conjunto de datos, para lo cual lo primero que se realiza, es obtener a partir de las distribuciones de probabilidad generadas anteriormente un valor aleatorio para la etiqueta de la clase para las muestras de la base de datos de test, debido a que es necesaria esa etiqueta, ya que se ha aplicado la base de datos de train con muestras de la base de datos de test y les faltan a estas ultimas ese valor, y guardan en una variable llamada `y_train` en un array de `numpy` 3 los valores de la etiqueta clase de la base de datos train y los valores obtenidos para las muestras seleccionadas de la base de datos de test. El siguiente paso consiste en generar dos arrays de indices uno de el tamaño de los datos ampliados y otro con el tamaño de los datos seleccionados de la base de datos de test. A continuación, se procede a realizar una permutación del array de indices completo, para posteriormente ampliarlo con siete veces del tamaño de los datos seleccionados de la base de datos de test, añadiendo de manera aleatoria muestras de este subconjunto. Se continua, generando las `DMatrix` que se utilizaran en el modelo, siendo una con los datos completos para train con su correspondiente

valores para la etiqueta y la correspondientes a los valores que se han dejado para el test. También se establecen los parámetros para correspondiente modelo utilizando un diccionario, siendo para este caso `num_round` con el que se establece que el número de rondas, `seed` con el que se establece la semilla para la generación de números aleatorios, siendo en este caso $123 + 13141 * i$ donde i es el índice del modelo, `max_depth` por el cual se establece que la profundidad de los árboles que construirá el modelo sea como máximo de 3, `gamma` con el que se establece el nivel de pérdida mínima necesaria para que se sigue expandiendo un nodo según una función de pérdida, `eta` por el cual se establece un peso en tanto por 1, en este caso 0,22, con el que se ponderan los pesos que va generando durante la construcción del modelo para establecer la importancia del modelo construido según su acierto a la hora de predecir las muestras proporcionadas al modelo, con el valor que se le ha proporcionado hace que el comportamiento del modelo sea conservador teniendo más importancia las características ya aprendidas que las nuevas características, `silent` con el que se establece que no se muestre información durante la generación del modelo, `objective` con `multi:softprob` por el cual se establece que el resultado será una matriz donde para cada elemento se mostrara la probabilidad de cada una de las posibles etiquetas de clase, `num_class` con un 9 con el que establece que el número de etiquetas para clasificar la clase es de 9 según se establece en el reto, `subsample` por el cual se establece el porcentaje en tanto por 1 de muestras que son seleccionadas para la construcción del modelo, `colsample_bytree` con el que se establece el porcentaje en tanto por 1, el número de características que van a ser seleccionadas para construir el árbol de decisión en cada iteración y `nthread` por el cual se establece que se generen 16 hilos para construcción de los modelos. También, se crea una array vacío que se llama `watchlist` que establece que no se revisa los resultados del modelo con un conjunto de muestras que no es utilizado en el entrenamiento. A continuación, se entrena el modelo XGBoost ?? con los datos y parámetros generados y después se realiza la predicción con los datos de test seleccionados, para a continuación añadirlos al array `pr_vals`. Una vez terminados los 20 modelos se obtiene la media de los resultados almacenados en el array `pr_vals` y se concluye por guardar este resultado en la copia del array de predicciones. Para concluir, una vez completado todo este proceso para todas las particiones se abre el fichero 'sampleSubmission.csv' con la función `read_csv` del paquete `pandas` 9, se le asignan los resultados guardados en la copia del array de predicciones y se guardan los resultados en un fichero llamado 'release1.csv'. Habiendo terminado así la construcción de todos los modelos necesarios para esta solución.

En este punto, solo queda abrir los dos archivos de predicción y el fichero 'sampleSubmission.csv' con la función `read_csv` del paquete `pandas` 9 para guardar en el dataframe de este último fichero una media ponderada de los dos ficheros de predicción obtenidos, siendo un 95 % el último modelo y el resto el otro modelo utilizado para la

predicción y se concluye guardando los resultados en un fichero llamado 'release2.csv', el cual es el que obtuvo la calificación indica al principio.

Capítulo 4

METODOLOGÍA Y DESARROLLO

Llegados a este punto, procederé a explicar el procedimiento llevado a cabo para la obtención de la solución del reto propuesto.

Conforme se estableció en la planificación del anteproyecto, el primer paso era la familiarización con la seguridad informática y concretamente con los conceptos necesarios para el abordaje de este reto. Por esto, además de realizar una búsqueda de soluciones propuestas por otros equipos, las cuales se ha explicado en el apartado de antecedentes y estado de la cuestión, también se buscó artículos en los que se mostrarán métodos de análisis que pudieran ser aplicados a este reto. De la cual, se obtuvo información de las familias de malware utilizados en este reto y sus correspondientes características identificativas, las cuales ayudarán a elegir cuáles serán las características más adecuadas a utilizar para el reto, siendo los tipos de malware utilizados los siguientes, según la página de Glosarios de Microsoft Windows Defender Security Intelligence [39]:

1. Worm o Gusano es un tipo de malware que se propaga a otros ordenadores. Este puede usar para propagarse uno o más de los siguientes métodos: Archivo adjunto o link en un email, envío a través de un sistema de mensajería instantánea como puede ser Skype, en los ficheros descargado o subidos en redes de intercambio de ficheros p2p, enviando un mensaje a todos tus contactos de una red social, en el auto-arranque de un disco extraíble o explotando una vulnerabilidad de un software.
2. Adware es un tipo de malware que muestra publicidad adicional que no puedes controlar estas usando el ordenador.
3. Trojan es un tipo de malware que a diferencia del worm [1](#) no se propaga por sí solo. Esto hace que tenga que aparentar ser un software inofensivo y que la víctima lo descargue e instale para de ese modo infectar su ordenador. Un vez, el

sistema esta comprometido puede robar tu información personal, descargar otro malware y pudiendo llegar a permitir al atacante tener acceso y/o control sobre tu ordenador.

4. Backdoor este tipo de malware es un clase de trojan 3 que permite al atacante tener acceso y control a tu ordenador.
5. TrojanDownloader, al igual que en el caso del backdoor 4, este tipo de malware es una clase de trojan 3, la cual instala otros ficheros maliciosos, incluido malware, en tu ordenador. El puede descargar el fichero desde un ordenador remoto o instalarlo directamente desde un copia incluida dentro de él.
6. Any kind of obfuscated malware no se trata de un tipo de malware, sino que son técnicas usadas para ocultar o hacer parecer limpio un malware, aunque en la página de glosarios de términos [39] que estamos utilizando si lo identifican como un tipo de malware. Lo que realizar es ocultar su código para hace mas difícil que los programas de seguridad lo detecten o eliminen.

De estos tipos de malware, se han seleccionado las familias que utilizaremos en el reto, siendo estas:

En primer lugar la familia Ramnit que según la información del reto es de tipo Worm 1 aunque en la propia página de Microsoft Defender Security [40] no se llega a clasificar, aunque se puede deducir ya que si se menciona que el vector de entrada más habitual es mediante la utilización de un disco extraíble, generalmente pendrive, ya infectado. Este malware roba información sensible del usuario, como pueden ser usuario y contraseña de acceso a su banco. Además, puede proporcionar acceso y control de nuestro ordenador al atacante.

En segundo lugar nos encontramos la familia Lollipop que según la información del reto y la propia página de Microsoft Windows Defender Security [41] es de tipo Adware 2. Esta familia de malware lo que realiza es mostrar publicidad emergente mientras se esta utilizando el navegador y redirige los resultados de tu motor de búsqueda. Debido a que redirige los resultados del motor de búsqueda, le permite mostrar la publicidad basándose en las palabras claves utilizadas en esto además de su ubicación geográfica.

En tercer lugar es el turno de Kelihos_ver3 que según la información del reto es de tipo Backdoor 4 y según la propia página de Microsoft Windows Defender Security [42] lo clasifican como un trojan 3 que puede dar acceso y control de nuestro ordenador al atacante, que conforme se puede observar es la definición de backdoor 4. Es la tercera versión de este malware y se propaga mediante el envío de correos electrónicos

no deseados que contienen enlaces a otros malware. Además, puede comunicarse con otros ordenadores para intercambiar información sobre el envío de correos electrónicos no deseados, robar tu información sensible o descargar y ejecutar ficheros maliciosos.

El siguiente es Vundo que según la información del reto es de tipo Trojan 3 aunque en la propia página de Microsoft Windows Defender Security [43] lo clasifica como una familia de malware formada por múltiples familias, el cual muestra ventanas emergentes con publicidad fuera de contexto. Además, algunas de sus variantes puede descargar y ejecutar otros ficheros, incluyendo otros tipos de malware. El vector de entrada más habitual es mediante un complemento para el navegador. También, utiliza técnicas avanzadas para defenderse y mantenerse a salvo de las detecciones y ocultarse cuando se eliminan.

A continuación, nos encontramos con Simda que según la información del reto y la propia página de Microsoft Defender Security [44] es de tipo Backdoor 4. Lo que realizar es robar la contraseñas del usuario del sistema infectado.

El siguiente en la lista es Tracur que según la información del reto es de tipo TrojanDownloader 5 pero en la propia de página de Microsoft Defender Security [45] lo clasifica como una familia de Trojans 3 que puede redirigir tus buscadores web. Realizan esto para ganar dinero con la publicidad fraudulenta. Este malware también puede descargar y ejecutar otros ficheros, esto incluye otros posibles malware, y de esta manera proporcionar al atacante el control de tu ordenador. Los vectores de entrada más habituales por este malware son que sea instalado por otro malware, cuando haces click en enlaces sospechosos o mediante un archivo adjunto en un email.

A continuación, es el turno de Kelihos_ver1 este tipo de virus es la primera versión del ya mencionado Kelihos_ver3 4, el cual en el reto es clasificado como de tipo Backdoor, mientras en la propia página de Microsoft Windows Defender Security [46] lo clasifican como un trojan 3, igual que ocurría con Kelihos_ver3.

Siguiendo con la lista nos encontramos con Obfuscator.ACY que según la información del reto y la propia página de Microsoft Windows Defender Security [47] se trata de una técnica mediante la cual se intenta ocultar 6 el malware y así evitar que los sistemas de seguridad la detecten.

Y por último, nos encontramos con Gatak que según la información del reto es de tipo Backdoor 4 y la propia página de Microsoft Windows Defender Security [48] es de tipo Trojan 3. Según la página mencionada, esta familia de malware recoge información

del equipo infectado y enviarla posteriormente al atacante y el punto de entrada suele ser a través de un generador de llaves para una aplicación o mediante una aparente actualización legítima de una aplicación. Una de las acciones más significativas que realiza que además permite identificar que estamos infectados con este malware es modificar el registro de Microsoft Windows para que la aplicación de mensajería y videollamada Google Talk se arranque de manera automática y Skype se inicie de manera automática de manera minimizada y sin mostrar la pantalla durante esta arrancando. Esto es llamativo porque estas aplicaciones pueden ser utilizadas para exfiltrar la información recabada del usuario.

Según se nos explica en el siguiente artículo [49], para este reto se nos proporcionan dos bases de datos, *train* y *test*, las cuales contienen muestras de malware de las clases mencionadas anteriormente. Estas están formadas por dos tipos de fichero, por un lado un fichero en binario, al cual se le ha eliminado la cabecera para hacerlo inofensivo, representado en formato hexadecimal y otro que contiene metadatos obtenidos mediante el desensamblado de las muestras con la utilidad IDA dissassembler tool. Donde la base de datos de *train* está compuesta por 10868 muestras de malware las cuales se reparten por las clases de malware según muestra en la tabla 4.1, en cuanto a la base de datos de *test* como esta formada por 10873 muestras de malware, las cuales ahí que clasificar en la clase que le corresponde.

Id	Clase	Muestras	Tipo
1	Rammit	1541	Worm
2	Lollipop	2478	Adware
3	Kelihos_ver3	2942	Backdoor
4	Vundo	475	Trojan
5	Simda	42	Backdoor
6	Tracur	751	TrojanDownloader
7	Kelihos_ver1	398	Backdoor
8	Obfuscator.ACY	1228	Any kind of obfuscated malware
9	Gatak	1013	Backdoor

Tabla 4.1: Clase de malware en la base de datos *train*.

Además, se consiguió el artículo ‘A scalable multi-level feature extraction technique to detect malicious executables’ de Mohammad M. Masud, Latifur Khan y Bhavani Thuraisingham de 2007 [50], en el cual se explica la investigación que han realizado con respecto a la detección de malware. Esta investigación demuestra que una extracción de características mixta mejora los resultados de los modelos. Esta perspectiva híbrida supone extraer y manejar características de los ficheros binarios en función

del número de ngrams [1](#), instrucciones en ensamblador, llamadas a funciones tanto del sistema como propias, etc. Esto supone incrementar y mucho el número de características, haciendo necesario realizar un proceso de selección de las mas significativas para que el modelo a construir sea abordable. Resulta interesante que se trata de la misma perspectiva utiliza en la solución 2 [3.2](#). Para mi solución, he decidido utilizarla también.

Llegado a este punto, comenzaré a explicar el proceso llevado para la construcción de mi solución.

La primera decisión, después de tener claro que se va a utilizar un sistema de características híbridas, es el número de ngrams [1](#) que se se van a utilizar. Una vez estudiado tanto el artículo anterior como la solución 2 [3.2](#), tomé la decisión de hacer un esfuerzo para utilizar en mi solución los mismos 4 ngrams que se utilizaba en la solución 2 [3.2](#). Esta decisión es más importante de lo que parece, ya que va repercutir en todas las decisión posteriores, debido a los recursos de los que voy a hacer uso son solamente mi portátil personal (Toshiba L850-150) y mi disco duro externo, siendo las características las siguientes:

- Procesador Intel Core i7-3610QM a 2,30 GHz.
- 8 Gb RAM DDR3.
- Tarjeta Gráfica Radeon 7670M.
- Disco Duro SSD Samsung EVO 850 de 250 Gb.
- Disco Duro externo Seagate Expansion de 4 Tb.

Estos recursos implican no poder tener todas las características extraídas en memoria ni tan siquiera de una sola muestra. Esto hace que sea necesario utilizar alguna tecnología que permita almacenar de manera persistente esta información. Para solucionar este problema, la idea fue utilizar una base de datos, concretamente MongoDB [22](#). El motivo por el que me decante por está me decidí por utilizar esta base de datos fue que se trataba de una base de datos de documentos con lo facilidad que esto implica ya que te permite almacenar en los atributos estructuras como pueden ser arrays y objetos BSON ??, además tiene una buenas compatibilidad con el lenguaje de programación elegido Python 3 y es una opción muy estable, consolidada, eficaz y open source.

Otra cuestión a tratar antes de entrar a explicar la configuración de la base de datos MongoDB, es explicar el motivo por el cual se ha tomado la decisión de utilizar docker y la configuración que se ha llevado a acabo. Para empezar el motivo, ha sido para fomentar la portabilidad de la solución y facilitar su ejecución y prueba. También, se ha buscado de alguna manera controlar los recursos que dispongo y aislar la ejecución del

sistema, ya que se manejan muestras de malware y aunque se hayan tomado medidas para convertirlos en inofensivos es mejor aislarlos. Por estos motivos y por tratarse de una sistema de virtualización ligero se ha optado por la utilización de Docker [24](#).

En primer lugar, se tiene que instalar Docker y Docker Compose conforme se puede en los siguientes enlaces [\[51\]](#) y [\[52\]](#) respectivamente. En este punto, comenzaremos con la configuración para lo cual se utilizar un fichero de tipo yaml donde se establecerán la configuración que posteriormente utilizará Docker Compose. En este fichero, se crearán dos servicios, db y tfg, y una red, cuyo nombre se tfg, para posibilitar la comunicación entre los servicios.

La red se llama tfg y se le ha indicado que el driver es bridge, mediante el cual se establece que los contenedores se mantienen aislados y utilizan esta red virtual para comunicarse entre ellos. También se le configura que el sistema administrador de direcciones IP utiliza como driver el que tiene establecido por defecto y que el segmento de red que utilizarán los contenedores es 172.16.0.0/24.

El servicio db se establece la configuración de la base de datos de MongoDB [22](#), para lo cual se establece que la imagen que se va a utilizar es la última disponible de mongo mediante el parámetro image:mongo:3.6.5. A continuación los parámetros hostname y container_name establece el nombre de la maquina y el nombre del contenedor que se creará respectivamente, en mi caso son mongo y mongo_tfg. El siguiente parámetro es command mediante el cual se establece el comando que se ejecutará al lanzar el servicio, en mi caso los que se ejecutará el servidor de MongoDB [22](#) con su correspondiente configuración. La configuración que se ha decidido establece para el servidor de base de datos MongoDB [22](#) es utilicé como motor de base de datos wiredTiger con el parámetro storageEngine, que cree un directorio por cada base de datos con el parámetro directoryperdb, que guarde en un directorio los índices y en otro las colecciones con el parámetro wiredTigerDirectoryForIndexes, que comprima las colecciones con el algoritmo que se establece con el parámetro wiredTigerCollectionBlockCompressor, en mi caso se ha establecido zlib, ya que es el mas eficaz a la hora de reducir espacio, que comprima también los índices con el parámetro wiredTigerIndexPrefixCompression y por ultimo se le establece la cantidad de RAM que utilizará con el parámetro wiredTigerCacheSizeGB, en caso le he establecido 4 Gb. El siguiente parámetro que se establece en el servicio es volume mediante el cual se establece un almacenamiento permanente al contenedor y en este caso también se le establece la dirección dentro del propio ordenador donde se almacenará, en este caso se ha optado por almacenarlo en el directorio home de mi ordenador en la carpeta TFGDataBase que esta ubicada en el disco solido.

Por ultimo, ya solo nos queda establecer la configuración de red que tendrá el contenedor para la red que se ha establecido para los contenedores, siendo la configuración una dirección IP dentro del rango establecido en la red tfg con el parámetro `ipv4_address` y alias con el parámetro `aliases`, siendo los valores para estos 172.16.0.2 y mongo respectivamente.

Por ultimo, el servicio tfg al igual que en el comando db se le ha establecido el nombre a la maquina, el nombre al contenedor, los volúmenes que establecen la ubicación del almacenamiento permanente y la configuración de red del contenedor. Además de esta configuración, se utiliza el parámetro `depends_on` para indicar que este servicio necesita de manera obligatoria el servicio db, con lo cual tiene que ejecutarse del servicio db antes de que comience la ejecución de éste, y el parámetro `enviroment` donde se establecen una listas de variables de entorno con las que se ajustará la ejecución de la solución. Estas variables de entorno son `SAMPLE` con la cual se indica que seleccione una muestra de de los fichero de train, `PERCENTAGE_SAMPLE` con el que se le indica el porcentaje concreto de la muestra, solo es utilizado conjuntamente con la variable `SAMPLE`, `PROJECT` con la cual se le asigna un nombre al proyecto, `IP_SERVER_MONGO` con la que se le dice la dirección IP que se le ha asignado al servicio db, `DB_NAME` con el cual se le indica el nombre de la base datos que va a manejar el proyecto, `ORDER_EXTRACT_FEUTURES_ASM` con la que se establece el número de ngrams [1](#) que se van a utilizar en la solución, `DIR_DATA_RAW` con la cual se le indica la dirección donde se encuentran los datos que maneja la solución, `SEED` con el que se le establece una semilla, `THREAD` con el cual se le indica el número de hilos que se utilizará en los procesos de paralización, `PERFORM_METHOD` con el que se le indica que parte de la solución se ejecutara y `NUM_FEATURES` con la cual se se le indica el número de características que se utilizara en el modelo que se construya. También se le establece que el nombre del contenedor sea `code_tfg`, el nombre de la maquina sea `code` y la imagen que utilice para crear el contenedor sea tfg, aunque en este caso esta no es una imagen oficial, como en el caso de MongoDB [22](#), esto implica que se tiene que construir está, por lo cual también se tiene el parámetro `build`. Este parámetro hace que tengamos que crear un fichero llamado `Dockerfile` donde estableceremos la configuración especifica para generar esta imagen. En este fichero se le configura el parámetro `FROM` con el cual se le indica que utilice como base la imagen oficial `python:3.6.5`, el parámetro `MAINTAINER` mediante el cual se le establece el autor de la imagen, el parámetro `LABEL` con el que se le establece una etiqueta y se le establece un nombre al proyecto, siendo este `detectorMalware`, el parámetro `ADD` con el que se copia los ficheros de la maquina huésped a la maquina virtual dentro del contenedor, el parámetro `RUN` con el que se ejecutan comandos dentro de la maquina

del contenedor, en mi caso va al directorio que se ha copiado y instala pip3 install -r requirements.txt los paquetes necesarios para ese proyecto, el comando WORKDIR con el que se establece como directorio inicial el directorio donde hemos copiado los ficheros y por ultimo el parámetro CMD con el cual se establece el comando que va a ejecutar cuando se inicie el contenedor, en mi caso python3 detectorMalware.py. Con esto se termina la configuración de Docker [24](#).

El directorio, que se copia en el servicio tfg [4](#) que contiene la implementación de mi solución, se estructura de la siguiente manera. En este directorio tenemos el fichero requirements.txt en el cual establecemos los paquetes de python necesarios para ejecutar mi solución, el fichero __init__.py en el que se establece metainformación del proyecto, como puede ser nombre del paquete, versión y autor, el fichero detectorMalware.py el cual es el encargado de ejecutar y controlar la ejecución de mi solución y por ultimo el directorio src el cual se subdivide en 4 directorios, siendo los siguientes extract_features el cual contiene los ficheros necesarios para llevar a cabo la fase de extracción de características, misc en el que se almacena los ficheros varios que se utilizarán en la solución, por ejemplo sample, split entre otros, model el cual contiene los ficheros que manejan desde el preprocesado de los datos hasta la predicción del modelo posando por la construcción del mismo y mongo donde se guarda el fichero que establece la conexión con la base de datos MongoDB [22](#).

Comenzaré la explicación de mi solución siguiente el proceso de ejecución del fichero detectorMalware.py, siendo el primer paso recuperar los valores de las variables de entorno establecidos en el fichero docker-compose.yml que se comentaron anteriormente. El siguiente paso, es comprobar si se desea ejecutar la solución sobre una muestra, esto se realiza así debido a que el volumen de información a manejar es excesivamente grande y durante el proceso de implementación sería muy tedioso tener que esperar el tiempo que supondría ejecutarlo sobre el conjunto completo de datos, por ese motivo se ha optado por esta opción para reducir el tiempo que supondría. Para seleccionar la muestra se ha creado un fichero sample.py dentro del directorio misc donde se ha implementado la función selectSample, el cual selecciona una muestra estratificada del fichero trainLabels.csv, para lo cual hace uso de pandas [9](#) creando un DataFrame [10](#) y seleccionando de manera aleatoria una muestra de los valores de la clase 1 y repitiendo ese proceso para el resto de valores de la clase y concatenándolos en el dataframe [10](#) del primer valor con el comando de pandas [9](#) concat para terminar guardando el dataframe [10](#) en un nuevo fichero csv y terminado por devolver la dirección con esta almacenado el mismo.

A continuación, se abre el origen ya sea el proporcionado por en el reto o en el gene-

rado durante el proceso de muestra, para lo cual se hace uso de la función `open_sources` la cual abre el el fichero de donde se van a leer los datos que se pasa a la función por parámetro y leyendo la información del fichero. Se continua separando esta información por líneas y desechando la primera porque son las cabeceras del fichero y la ultima porque en el fichero ahí una linea la final del documento sin información. Después, se obtiene el número de lineas que tiene el documento y se parte cada una de las lineas con una función aplicándole una función que se le pasa por parámetro a la tupla obtenida de aplicar la función `enumerate` a los datos. Por ultimo, se cierra el fichero y se devuelve la longitud y los datos ya particionados. Las funciones para realizar el particionado de los ficheros origen se encuentran en fichero `split` en `misc`. En este fichero tenemos la función para particionar los datos de origen de `train`, de `test` y una función genérica común para ambas funciones. La función `split_Source` lo que hace generar un array en el que la primera posición es el indice que tiene ese elemento en el fichero origen seguido de todos los elementos que posee el elemento después de dividirlos por la coma y quitarle las comillas dobles. En el caso de `split_Source_Test` se devuelve una tupla de los dos primeros elementos del array que general la función general. En cambio, en `split_Source_Train` se hace igual pero el tercer elemento del array se transforma de `string` a un entero.

El siguiente paso, es comprobar el valor que tiene la variable de entorno `PERFORM_METHOD`. Los posibles valores que se le puede asignar son `EXTRACT_TRAIN`, `DISCRETE`, `SELECT`, `MODEL`, `EXTRACT_TEST`, `MODEL_PREDICT`, `PREDIT`, `EXTRACT_ALL` Y `ALL` que establece si se extraen las características de la base de datos de `train`, si se discretizan las características de la base de datos de `train`, si se seleccionan las características que mayor información proporcionan de la base de datos de `train`, si se crea el modelo con las características seleccionadas de la base de datos de `train`, si se extraen las características de la base de datos de `test`, si se crea el modelo y se predice el resultado de la base de datos de `test`, si se predice el resultado de la base de datos de `test`, si se extrae las características de las dos bases de datos o todo lo anterior respectivamente. Esto se ha realizado así para durante el proceso de implementación poder excluir los pasos ya concluidos y ahorrar todo el tiempo que fuese posible.

El primero de estos pasos que ahí que ejecutar, por orden es `EXTRACT` el cual, lo primero que realizar establecer la conexión con la base de datos para los cual se usa la clase creada en el fichero `database.py` dentro de la carpeta `mongo`. En este fichero, contiene una clase llamada `DataBaseMongo` la cual se ha creado para manejar y controlar la conexión con la base de datos, para lo cual cuando se crea una instancia de la misma ahí que indicarle tanto la dirección del servidor de base de datos MongoDB [22](#) y el nombre de la base de datos a la que nos queremos conectar. Esta clase también posee

un método para seleccionar una colección pasándole como parámetro el nombre de la misma. Esto se ha realizado así para garantizar de que no se le pase ningún parámetro erróneo y para manejar los posibles excepciones tanto de la conexión con el servidor como la selección de la colección.

Una vez, establecida la conexión con la base de datos se procede a borrar todas las colecciones que tiene establecidas, para lo cual selecciona cada colección y la elimina con el método `drop` proporcionado para la API de MongoDB [22](#) para python.

A continuación, se procede a la extracción de las características de la base de datos `train`, proceso que se realiza de manera paralela utilizando el número de hilos que se ha establecido en la configuración, y realizando así la extracción de las características de manera simultánea. Este proceso de extracción consiste en llamar en la función `extract_features` dentro del fichero `extract_features.py`, al cual ahí que pasarle por parámetro la ip del servidor de base de datos MongoDB [22](#), el nombre de la base de datos a la que nos conectaremos, la ubicación dentro del contenedor donde se puede acceder a los datos proporcionado, el prefijo y el nombre de las colecciones que se utilizarán, el número de ngrams [1](#) a extraer y por último del fichero del que se extraerán las características, siendo todos los parámetros fijos salvo este último que cada hilo tratará uno de estos ficheros. Los datos proporcionados se les pasan al sistema en un fichero comprimido en zip de donde se seleccionan el elemento a procesar. Esto se ha realizado de esta manera por ahora espacio en el disco duro y ha sido sacada la idea de la solución [1](#) [3.1](#), ya que de esta manera el peso de las dos bases de datos es de algo más de 56 Gb mientras si se trata de manera descomprimida ocuparía más de 400 Gb complicando donde almacenar tanta información, aunque como no se podía manejar los archivos comprimidos que se suministran a tenido que descomprimirse el fichero `7z` y posteriormente comprimirlo de nuevo en formato zip, el cual si es posible manejarlo con python. Este proceso se realiza por separado tanto a la base de datos de `train` como para `test`. Llegados a este punto, el primer paso es establecer la conexión con la base de datos y seleccionar la colección `train_total` donde se inserta un registro con el id, nombre y etiqueta del elemento que se va a procesar en dicha colección. El siguiente paso consiste en realizar las modalidades de extracción que se van a realizar, extracción de binario y de ensamblador, para lo cual se utiliza la función `extract_Features_Byte` dentro del fichero `extract_features_bytes.py` y la función `extract_Features_Asm` dentro del fichero `extract_features_asm.py` respectivamente. Empezaré explicando la función `extract_Features_Byte`, la cual se ha adaptado la implementación de la solución [2](#) [3.2](#) que se esta parte de la solución a mi solución, siendo lo primero se conecta con la base de datos de MongoDB [22](#) y seleccionar tanto la colección `total` como la `features` donde se almacenará la información extraída. Después, se abre de dentro del fichero comprimido

en zip para esa base de datos el fichero establecido en el origen y se acumulan todos los ngrams 1 en un array para posteriormente recorrer este array en bloque establecidos por el parámetro order que establece el número de ngrams 1 que se extraerán llevando el conteo en un diccionario. Se continua recorriendo este diccionario para obtener el número total de ngrams 1 y actualizar el valor del registro insertando anteriormente en la colección total con el número total de estos del fichero, esto es realizado así para tener los totales y realizar una normalización de los conteos respecto del fichero. Para concluir, insertando todos los elementos del diccionario en la colección features donde se almacena para cada bloque de ngrams 1 del fichero en id del fichero, el nombre del fichero, la característica que esta formada por "b_" seguido del bloque de ngrams 1 que corresponde, el número de veces que aparece en el fichero tanto normalizado respecto del fichero como sin normalizar y el tipo de característica de que se trata, siendo en este caso bytes.

En cambio en la función `extract_Features_Asm`, donde se realiza la extracción de la información de los ficheros en ensamblador, el proceso que realiza empieza igual que en el caso de bytes donde se establece la conexión con la base de datos, se seleccionan las colecciones que se utilizarán que son total y features y se obtiene de dentro del fichero comprimido en zip el fichero establecido como origen para extraer los datos, aunque en este punto se le establece que se encuentra codificado en iso8859 debido a que en el paso anterior no era necesario al no manejar caracteres especiales. A continuación, se parte el fichero en líneas haciendo uso del paquete `re` 25 con la función `findall` y la opción `multiline`, esto se realiza así para reconocer varios tipos de salto de línea y evitar así de esta manera posibles problema de codificación. En este punto, se elimina de la línea los posibles saltos de carro, los comentarios, los espacios en blanco al inicio y los tabuladores, para continuación partiendo la línea una vez por el separador ":" y obteniendo así la sección y la línea que queda pendiente de tratar. El siguiente paso, es partir la línea una vez teniendo en este caso como separador obteniendo en este paso la dirección y almacenando en un diccionario la sección extraída anteriormente un conjunto donde se guardaran las direcciones encontradas, esto es así para solo contemplar una vez cada una de las direcciones. A continuación, si la línea pendiente de procesar no esta compuesta solo de espacios en blanco, entonces se divide la parte binaria para lo cual se utiliza la `partition_bytes`, debido a la complejidad que tienen las comprobaciones necesarias se creo esta función para no complicar mas la función principal. En esta función, se comprueban todos los posibles patrones reconocibles, siendo que sean un conjunto de ngrams 1 que terminan con al menos dos espacios en blanco, que sea un conjunto ngrams 1 y termine con el signo +, que sea un conjunto de ngrams 1 seguido únicamente espacios en blanco o por ultimo que contenga al menos un espacio en blanco. En todos los casos se parte según el patrón que se ha obtenido, salvo en el

ultimo de los casos que se parte por el espacio en blanco y se comprueba si cada parte corresponde con el patrón de un ngram 1 y con una variable se controla la posición que tenemos en el array hasta ese momento y terminando por formar un array donde el primer valor son los elementos hasta la ultima posición que coincide con el patrón de los ngrams 1 y transformando el array en string y como segundo valor el resto de los elementos también transformándolos de array a string. Para esta transformación se utiliza la función `arrayToString` dentro del fichero de `string` dentro de `misc`. Esta función lo que realiza es coger el primer elemento del array y concatenarle el resto de elementos con el un espacio en blanco entre medias. Un vez terminada obtenido la parte binaria, solo nos queda partir la linea pendiente de procesar teniendo como separador al menos un espacio en blanco. De esta ultima partición obtenemos como primer elemento la instrucción y el resto de elemento son argumentos de esta. A continuación, se parte la instrucción si contiene algún guión bajo que no este al principio o al final y se selecciona como instrucción el primer de los elementos y concatenando el resto una vez transformado en string con la función `arrayToString` con la linea sin procesar con un espacio en blanco entre medias. Se continua limpiando de la instrucción de los posible guiones bajos, de los dos punto, de los iguales y de las comas los espacios en blanco que tuvieran alrededor. Ahora ya solo queda limpiar de espacio en blanco al inicio y al final la linea pendiente de procesar y partirla teniendo como separador al menos un espacio en blanco y obteniendo un array de argumentos, en el caso de no tener espacio en blanco se tiene un array con un solo elemento que es al linea pendiente de procesar. Por ultimo, en el caso de que la instrucción sea 'call' se le añade a la instrucción el primero de los elementos de los argumentos y guardando la instrucción en un diccionario. Para terminar los dos diccionarios para obtener el número total de instrucciones y secciones y actualizar el valor del registro insertando anteriormente en la colección total con los totales del fichero, esto es realizado así para tener los totales y realizar una normalización de los conteos respecto del fichero. Para concluir, insertando todos los elementos de los diccionarios en la colección `features` donde se almacena para cada bloque de instrucción o sección del fichero en `id` del fichero, el nombre del fichero, la característica que esta formada por 'a_' o 's_' seguido del bloque de de instrucción o sección respectivamente, el número de veces que aparece en el fichero tanto normalizado respecto del fichero como sin normalizar y el tipo de característica de que se trata, siendo en este caso `asm` o `section`.

Un vez se termina de la extracción de todos las características de la base de datos de `train` se crea un indice de las características `feature` y `index_file` ambos en orden ascendente. Terminado así el proceso de extracción de las características del a base de datos de `train`.

En este punto, es cuando me doy cuenta que todas las variables que voy a utilizar tienen valor continua complicando de ese modo el manejo de los mismo. Se buscan posibles soluciones, como son calculo de la entropía de Shannon a través de una función de probabilidad, discretización, etc., terminando por seleccionar la discretización de las variables como mejor opción. Para ser concreto el sistema de distretización que se utilizará es el utilizado por el árbol de decisión C45, el cual se estudio durante la especialización en la asignatura Sistemas Basados en el Conocimiento. Es un sistema muy sencillo y esta comprobado que obtiene buenos resultados. Mi implementación se llevará a acabo en la función `calculateDiscreeting` dentro del fichero `selection_filter.py` apoyándome en todo momento en la base de datos MongoDB [22](#).

En esta función, lo primero que realiza conectarse con la base de datos, seleccionar las colección `features` y `discreeting`, y borrar la colección `discreeting` todas las colecciones con su prefijo correspondiente. A continuación, se crea un array de numpy [3](#) de 1 por el número de elementos de la base de datos a tratar con valores de tipo entero del 0 hasta el tamaño de la base de datos menos 1 y una variable llamada `processed` donde se almacenan el número de elementos procesados hasta el momento, como es lógico el valor inicial de esta variable es de 0. El siguiente paso es un bucle infinito que terminará cuando se termine el cursor sin que se produzca el error `CursorNotFound`, el cual ocurre cuando el cursor supera el tiempo máximo de espera establecido. El cursor al que se hacia mención anteriormente es el obtenido de un array con los indices y su correspondiente etiqueta insertados por orden ascendente del número de apariciones normalizadas y etiqueta, ademas de ordenador por característica, y saltándose los elementos ya procesados hasta el momento. A continuación, se recorren todos los elementos del cursor dentro de `try except` para manejar en el caso de que de error `CursorNotFound` mencionado anteriormente. Para cada uno de estos elementos, se incrementa en 1 la variable `processed`, se comprueba si la característica aparece al menos en dos muestras sino se salta a la siguiente característica. A continuación, se crea un array para las partes discretizadas y otro para la partición actual y una variable llamada `num_part` donde se almacena el indice de la partición actual empezando desde 1. Además, se crea otros dos array uno con todos los posibles indice y otro de numpy [3](#) donde se almacenará en que partición se encuentra cada indice, esto se hace así para durante el proceso de discretización calcular también la entropía de la característica que se utilizará posteriormente. En este punto, en cuando comenzamos con el proceso de discretización, teniendo como primer paso seleccionando la información del primer elemento, indice y etiqueta, y almacenándolos en `index_element_prior` y `label_element_prior` respectivamente. A continuación, se elimina el indice del array de indices y se añade al array de la partición actual. Después, se recuperan los datos indice y etiqueta del segundo elemento y se guardan en `index_element` y `label_element` respectivamente, y se elimi-

na el índice del array de índices. El siguiente paso recorrer el resto elementos del array de la característica. Ahora, toca comprobar si `label_element_prior` y `label_element` o `label_element` y la etiqueta del siguiente elemento son distintos, en ese caso se añade la partición actual al array de partes discretizadas, se crea un nuevo array para la parte actual y se incrementa en 1 la variable `num_part`. A continuación independientemente del resultado de la comprobación anterior, se añade `label_element` al array de la partición actual, se le coloca en el array de numpy 3 de partición según índice se guardada en la posición establecida por `index_element` el valor de `num_part`, se guardan los valores de `index_element` y `label_element` en `index_element_prior` y `label_element_prior` respectivamente, y en `index_element` y `label_element` se guardan los valores del siguiente elemento, eliminado el índice de la lista índices. Una vez terminados todos los elementos toca añadir el ultimo elemento a la partición actual, actualizar el valor de posición del índice en el array de particiones de índices con el valor de `num_part` y se añade la partición actual al array de partes discretizadas. A continuación, si quedan elementos en la lista de índices de añade al array de lista de particiones se incrementa en 1 la variable `num_part` y se actualiza con este valor los índices del array de numpy 3 de particiones de índices. Llegado a este punto solo nos queda calcular la entropía de la característica con la función `calculateEntropy`, a la cual se le pasa como parámetros en número de particiones, el array de numpy 3 de partes de índices y el array de numpy 3 con todos los índices de la base de datos y veremos ahora después, insertar en la colección `discreting` los valores para esta característica, que son el nombre de la característica, en número de muestras en el que aparece la característica, la entropía de la característica, la discretización que se ha realizado y por ultimo el número de partición que se han realizado, cerrar el cursor, salir del bucle infinito con `break` y terminar eliminando la conexión con la base de datos.

La función `calculateEntropy` se utiliza para calcular la entropía aplicando el suavizado de Laplace, pudiendo calcularse de una características, de la clase o de una características conociendo la clase, eso solo depende de los parámetros que se le pasen. Los parámetros que se le tienen que pasar son los datos que a de ser un array de numpy, por ejemplo para la característica los datos serían el índice de la partición a la que corresponde el índice, en la clase sería la etiqueta que correspondiente al índice, y el número de valores distintos que forman los datos. Está consiste en crear un array de numpy 3 de ceros con tamaño el número de particiones que se le pasan para almacenar los pasos de la entropía y una variable donde se guarda el numero total de elementos. Después, se recorre el array de la particiones aplicándole un `enumerate` para obtener el índice y el elemento del array de particiones. A continuación, se utiliza la función `calculate_num_labels` la cual crea un array de ones, mediante la cual se aplica el suavizado de Laplace, y se cuenta en el array de numpy 3 el número de elementos de cada

etiqueta y se devuelve éste. A continuación, se suman el número total de elementos del array de etiquetas y se añade al total elementos. El siguiente paso, consiste en dividir el array de etiquetas entre el número total de elementos que contiene este array para después multiplicar este número por el logaritmo de el mismo, terminado por sumar todos los valores y cambiándole el signo y se guarda en array de entropía en el índice de la partición. Una vez terminadas todos los elementos del array de partes se divide el array de entropía entre el total de elementos contados, sumando todos los valores y terminando por devolver este ultimo valor.

En este punto que ya se ha terminado el proceso de discretización, ahora nos toca realizar un proceso de selección filter, concretamente un proceso forward, para seleccionar las características que mayor información proporcionen a la hora de clasificar las muestras proporcionadas, por lo cual se implemento el método calculateRanking en el fichero selection_filter.py. A este método se le pasa como parámetros el prefijo a utilizar para las colecciones, la dirección IP del servidor de base de datos MongoDB 22, el nombre de la base de datos, la lista con los nombre utilizados para las colecciones, el número de características a seleccionar y el número total de muestras de la base de datos proporcionada.

Para evaluar la información que proporcionaba cada variable para realizar el proceso de selección filter forward de características, en un principio se pensó y se probó a implementar el calculo de incertidumbre simétrica haciendo uso de ganancia de información y utilizando la probabilidad conjunta de la clase y las características seleccionadas hasta el momento, ya que el número de muestras es mucho menor que el número de características, ya que se probó con un muestra de 5 de las características y 4 ngrams 1. Una vez se fue a ejecutar esa solución con todo los datos, se llego a comprobar que el tiempo necesario para mantener esta propuesta lo hiciera inabordable. En es punto, fue cuando concluí que la opción mas eficaz sería calcular la información mutua de las características mediante la aproximación de Battiti sustituyendo dentro de la misma la información mutua por la incertidumbre simétrica y apoyándome en MongoDB 22.

Mi implementación del método calculateRanking consiste en primer lugar conectarse a la base de datos MongoDB para después conectarse a la colección total y generar un array de numpy 3 donde se almacena en el índice del fichero la etiqueta que le corresponde. A continuación, se seleccionan la colecciones discreeting, ranking, aux_ranking, ranking_su y ranking_mi y se borrar todas excepto discreeting. Después, se realizar una copia de la colección discreeting en la coleccion aux_ranking, para mantener mantener intacta esta y poder así eliminar en cada pasada la característica seleccionada, y se crea un índice único en esta colección en la característica feature y otro índice único

en la colección `ranking_su` en la característica `feature_prior` y después por `feature`, siendo el orden para todas las características de los índices ascendente. Ahora, toca guardar en `feature_prior` el nombre de la característica seleccionada anteriormente, su particionado y su entropía de la clase, para este primer paso se guarda como característica 'label', como particionado un array con un único array con todos los índices y como entropía la obtenida de utilizar la función `calculateEntropy` con los datos y el particionado que tenemos y como número de características el número de etiqueta que tiene el reto. A continuación, se calculará la incertidumbre simétrica para todos los elementos de la colección `aux_ranking`, para lo cual usamos un contador de elementos procesados, un bucle infinito, recuperar los elementos que no han sido procesados, calcular la incertidumbre simétrica de cada elemento respecto al elemento anterior almacenado en `feature_prior` y terminando por cerrar el cursor y salir del bucle infinito si se han procesado todos los elementos sin que ocurra el error `CursorNotFound`, en caso de que ocurra se repite este proceso descartando los elementos ya procesados.

Para calcular la incertidumbre simétrica se utiliza la función `calculateRankingSU`, a la cual se le pasa como parámetro la dirección IP del servidor de base de datos MongoDB 22, el nombre de la base de datos a la que se conecta, el nombre la colección que utilizará para almacenar los resultados, el array de numpy 3 que almacena la etiquetas de las muestras, el elemento anterior y el elemento actual, y lo que hace es conectarse a la base de datos de MongoDB 22, conectarse a la colección establecida, se calcula la entropía de la clase conociendo la características, para lo cual se utiliza la función `calculateEntropy` con los datos, la partición de la característica y en número de etiquetas que se manejan, se calcula la incertidumbre simétrica utilizando la ganancia siendo esta el doble de la entropía de la clase menos la entropía de la clase conociendo la característica calculada anteriormente y todo esto dividido entre la suma de la entropía de la clase y la de la característica y se termina por insertar en la colección la característica, la característica anterior, el particionado, la incertidumbre simétrica y la entropía de la clase sabiendo la característica.

Una vez terminado el cálculo el cálculo de la incertidumbre simétrica de todas las características, nos toca a partir de estos valores calcular la información mutua aplicando la fórmula de la aproximación de Battiti con la incertidumbre simétrica, por lo cual lo primero que se realiza es realizar una consulta a la colección `ranking_su` de los elementos que forman la colección `ranking` y totalizar el valor de la incertidumbre simétrica de esos elementos en esa colección y un array con los elementos que lo forman. A continuación, se utilizará un bucle infinito con un contador de elementos procesados y se seleccionaran los que se han generado en el paso anterior saltándose los elementos que ya han sido procesados para manejar el posible error `CursorNotFound`. Después,

elimina la base de datos `ranking_mi` y le crear un indice para la característica `mi` en orden descendiente y en caso de empate lo ordena por `feature` en orden ascendente. Se continua, se procede al calculo de la información mutua de cada característica para lo cual se utiliza la función `calculateRankingMI`, a la cual se le pasa como parámetro la dirección IP del servidor de base de datos [22](#), en nombre de la base de datos, el prefijo utilizado en las colecciones, el nombre de la colección que se utilizará, el acumulado de los elementos de `ranking_su` que aparecen en la colección `ranking` y la característica a calcular.

Mi implementación para la función `calculateRankingMI` consiste en conectarse a la base de datos y se seleccionan las colecciones `ranking_mi` y `ranking_su`. A continuación, se recupera el array de los elementos que forman parte del acumulado de los elementos de `ranking_su` que aparecen en la colección `ranking` y se le añade el elemento actual, teniendo así el array de características a utilizar. Después, se obtiene el acumulado de la incertidumbre simétrica de todos las características que aparecen en el array de características y no tienen como elemento anterior `'label'`. Se continua, calculando el valor de la incertidumbre simétrica utilizando el total de la incertidumbre simétrica de los características que de `ranking_su` que se encuentran en la colección `ranking` mas el valor de la incertidumbre simétrica de la colección `ranking_su` de la característica actual con elemento anterior `'label'` menos 0,5 por el acumulado obtenido aquí, para concluir, añadiendo el valor obtenido de la información mutua al elemento de la característica y añadiendo este a la colección `ranking_mi`.

Una vez se termina el calculo de información mutua de todos las características, no toca seleccionar la característica con el mayor valor para la información mutua. A esta característica, se le añade el valor de la posición que le corresponde dentro del `ranking` y se añade en la colección `ranking`. Para terminar, actualizando los valores de `feature_prior` con los del elemento seleccionado, siendo esto el nombre de la característica, el particionado de la característica y la entropía de la clase sabiendo la característica seleccionada, y eliminando la característica de la colección `aux_ranking`.

Una vez terminado el calculo del `ranking`, nos tocar el código con la opción `MODEL`, con la cual se construye el modelo que realizará la predicción de la base de datos de test posteriormente. Para conseguir esto se utiliza la función `create_Model` dentro del fichero `create_model.py`, a la cual se le pasa como parámetro el prefijo que se utiliza para las colecciones, la dirección IP del servidor de base de datos MongoDB [22](#), la tupla con los nombre de las colecciones, el número total de muestras que forma la base de datos suministrados para el reto, el número de características que se van a utilizar y la semilla para obtener los números aleatorios que se ha establecido.

Una vez dentro de la función `create_Model`, lo primero que se realiza es seleccionar las características de todas las muestras y sus correspondientes etiquetas, para lo cual se utiliza la función `select_features` dentro del fichero `select_features_kind.py`, a la cual se le pasa como parámetro la dirección IP del servidor de base de datos MongoDB [22](#), el nombre de la base de datos, el prefijo utilizado para las colecciones, la tupla de nombre de las colecciones, el número de características que se va a seleccionar y el número de muestras que posee la base de datos que se nos ha proporcionado. Donde, lo primero es conectarse a la base de datos, para continuar creando las estructuras donde se almacenarán las características y las labels, siendo estas una matriz de numpy [3](#) de ceros de tamaño el número de muestras de la base de datos suministrada por el número de características que se han seleccionado y un array de numpy [3](#) de ceros de longitud el número de muestras de la base de datos suministrada respectivamente. A continuación, selecciona la colección `ranking`, se seleccionan de la colección `features` los elementos que aparecen en `ranking` y se crea una variable booleana en la que se marca se representa como cierto si las características de la base de datos contienen el campo 'label'. Llegado a este punto, toca recorrer los elemento de la consulta y dentro de este elemento toca recorrer los fichero que forman cada característica, para almacenar en conteo normalizado respecto del fichero de la característica en la posición indicada por el índice del fichero y la posición de la característica en el `ranking`. En el caso de que en la información de la colección `features` apareciera el elemento 'label' se guardaría en la posición del de índice del fichero el valor de 'label' y se modificaría la variable booleana indicándole que en la base de datos aparecen la etiquetas. En el caso de que aparezcan las etiquetas en la colección directamente se devuelve únicamente la matriz con las características. en cambio, si aparecen las etiquetas en la colección se comprueba si algún índice no tiene la etiqueta porque no contenga ninguna de las características establecidas, en ese caso selecciona la etiqueta correspondiente al índice del fichero almacenado en la colección total y terminando por devolver tanto las características como las etiquetas.

Ahora de vuelta en la creación del modelo, después de seleccionar las características y las etiquetas toca en primer lugar reducir en 1 las etiquetas para que en lugar de estar desde 1 a 9 para que este de 0 a 8 conforme maneja el modelo. A continuación, se establecen los parámetros los cuales son un diccionario con `num_class` que establece el número de etiquetas que tiene la clase, en este caso 9, `eval_metric` que es un array con las métricas que tiene que utilizar, en este caso solo se utilizará una que es `mlogloss` [18](#), el objetivo que tiene el modelo que en este caso `multi:softprob` para que me devuelva la probabilidad de que etiqueta para la muestra, `booster` con el que se establece el tipo de booster que se utiliza, en este caso `gbtree` con el cual se utilizar un árbol de decisión,

silent que establece que no muestra información por pantalla al construir el modelo y por ultimo la semilla la cual se establece que sea el doble de la que se le ha pasado al ejecutar el código. Además de los parámetros que se almacenan en el diccionario, se tiene una variable con la que se establece el número de iteraciones para construir el modelo y un array vacío el cual establece los elementos sobre los que se va a comprobar el modelo, en este caso se ha optado por un array vacío ya que para la comprobación se va a utilizar un sistema de validación cruzada en lugar de esta opción. El siguiente paso, es construir DMatrix con las características y las etiquetas, ya que es la estructura mediante la cual se le pasan los datos al modelo XGBoost `??`. Ahora, es el turno de realizar la validación cruzada del modelo para comprobar los resultados para lo cual utilizamos la función `crossvalidationXGBoost` en el fichero `validation.py` a la cual se le pasan el número de pasos a repetir para construir el modelo, el diccionario con los parámetros y la DMatrix construida con los datos y mostrando por pantalla los resultados obtenidos la media del ultimo paso del modelo tanto para las particiones de train como de test. Llegado a este punto, solo nos queda construir y entrenar el modelo para lo cual utilizamos el método `train` que nos proporciona XGBoost para este objetivo para lo cual ahí que pasarle como parámetro el diccionario de parámetros, la DMatrix con los datos a utilizar para construir y entrenar el modelo, el número de estimadores y el array con los elementos sobre los que se comprobará el modelo, y terminando por devolver el modelo que se ha construido.

En cuanto a la función `crossValidationXGBoost`, con la que se lleva a cabo la validación cruzada para validar el modelo se realiza en primer lugar se le establece a `numpy` [3](#) una semilla, que en este caso es la que se le ha pasado en el diccionario de parámetros mas el número de estimadores para a continuación utilizando el método `cv` que proporciona XGBoost realizar una validación cruzada para lo cual se le pasa como parámetro el diccionario con los parámetros, la DMatrix con los datos para procesar, el número de particiones, que para este caso es 10, en número de rondas que se realizara se repetirá el modelo, una variable booleana para indicar que no devuelva los resultado con `pandas` y establecer la semilla que en este caso es la establecida en el diccionario de parámetros mas dos veces el número de rondas, y terminado por devolver el diccionario con los resultados generados.

Una vez ya tenemos el modelo construido, es el momento de generar los datos correspondientes a la base de datos de test para lo cual se establece la dirección donde se encuentra el fichero, se establece que ahora el prefijo de las colecciones ahora sea `test` y se abre el origen de datos con la dirección donde se encuentran los datos de test y utilizando la función `split_Source_Test` para partir los datos y obteniendo así el número de muestras que forma esta base de datos como los datos de las muestras listos para ser

procesador. En este punto, ejecutar el código con el modalidad de `EXTRACT_TEST` y realiza el mismo proceso que el realizado con la base de datos de train para extraer las características.

Ahora, para terminar se ejecuta el código con la modalidad `PREDICT` para lo que se usa la función `prediction` del fichero `prediction.py` a la cual se le pasa como parámetro el modelo, la dirección IP del servidor del servidor de base datos MongoDB [22](#), el nombre de la base de datos, el prefijo utilizado para las colecciones, el número de características que se utilizarán, el número de muestras que forma la base de datos proporcionados y la dirección donde se encuentra el ejemplo de como realizar la predicción. Ahora, dentro de la función nos toca seleccionar los datos con la función `select_features` que vimos anteriormente y que en este caso solo devuelve los datos. A partir, de esos datos recuperados se genera una `DMatrix` para pasárselos al modelo y utilizando el método `predict` al que se le pasa la `DMatrix` anterior se obtiene la predicción para la base de datos de test. Llegados a este punto, toca abrir con `pandas` [9](#) el fichero de ejemplo, para a continuación guardar la predicción generada y terminar por guardar los resultado en el fichero generado. Quedan así terminado la ejecución del código de mi solución. En el siguiente punto, se explicará como se ha ejecutado esta solución y los resultados que ha obtenido.

Capítulo 5

EXPERIMENTO Y RESULTADOS

Para ejecutar el experimento el requisito fundamental es tener instalado Docker y Docker Compose conforme ya se ha indicado anteriormente. Una vez se tiene estos recursos la disponible existen dos posibilidades, que disponga de conexión a internet y se descargue las imágenes originales utilizadas como base o en su defecto importar la imágenes utilizadas en el código, las cuales están disponibles en el CD proporcionado con el TFG dentro de la carpeta Imágenes Docker. En esta carpeta de proporcionan las imágenes necesarias en archivos comprimidos en formato tar. Para importa estos archivos en Docker es necesario utilizar el comando *‘docker load -i {NOMBRE DEL FICHERO}’*.

Una vez tenemos la imágenes que utilizaremos ya disponibles, toca configurar los volúmenes que utilizarán estas imágenes, mediante los cuales se le proporcionará persistencia a la información almacenada y disponibilidad a los mismo desde fuera del contenedor, los cuales se encuentran dentro del fichero docker-compose.yml. Los volúmenes que utilizaremos para cada uno de los servicios son:

- Para el servicio db que corresponde a la imagen mongo y proporciona la configuración del servidor de base de datos MongoDB [22](#), el volumen es:
 - /mnt/Data/TFGDataBase/data/:/data/db/, mediante el cual se monta la dirección /mnt/Data/TFGDataBase/data/ de mi equipo dentro del contenedor en la dirección /data/db/. Este volumen se utiliza para almacenar de manera persistente las base de datos del contenedor.
- Para el servicio tfg que corresponde a la imagen tfg y es el código de mi solución, los volúmenes son:
 - ./detectorMalware/:/detectorMalware/, mediante el cual se monta el directorio detectorMalware ubicado dentro de mi posición actual a la dirección

/detectorMalware/ dentro del contenedor. Este volumen es para poder suministrar al contenedor el código de mi solución que tiene que ejecutar.

- \$HOME/TFG/results/:/data/results/, mediante el cual se monta el directorio results dentro de TFG en mi carpeta personal en el directorio /data/results/. Este volumen es para poder tener acceso a los resultados de la predicción fuera del contenedor.
- \$HOME/TFG/:/data/raw/, mediante el cual se monta el directorio TFG en mi carpeta personal en el directorio /data/raw/. Este volumen es para poder suministrar al contenedor la información suministrada para el reto como son los ejemplos de la predicción, el fichero con la etiquetas de las muestras de la base de datos de train y las dos bases de datos suministradas pero en formato zip, en lugar del formato en el que se suministraban, ya que en ese formato fui incapaz de manejarlo en Python.

A continuación, toca en el mismo fichero docker-compose.yml configurar las variables de entorno utilizadas para el contenedor Docker tfg, siendo estas:

- SAMPLE que se le establece el valor a False para establecer que no seleccione una muestra de la base de datos de train. Esta variable se incluyó para realizar las pruebas del código implementado.
- PERCENTAGE_SAMPLE que se establece el valor a 5 para establecer que se utilice el porcentaje de la base de datos de train que se seleccionara. Esta variable solo es utilizada cuando la variable SAMPLE está a True, con lo cual para esta ejecución es irrelevante.
- PROJECT con la que se le establece un nombre al proyecto, siendo en este caso detectorMalware.
- IP_SERVER_MONGO se le establece con esta variable la dirección IP del contenedor Docker donde esta la base de datos MongoDB [22](#), para este caso el valor es 172.16.0.2.
- DB_NAME con esta variable se le establece el nombre de la base de datos que utilizaremos durante la ejecución, siendo en este caso tfg.
- ORDER_EXTRACT_FEATURES_ASM que establece el número de ngrams [1](#) binarios que se extraerán de las muestras proporcionadas, para esta ejecución se ha utilizado el valor de 2 debido a las implicaciones de espacio que tiene durante el proceso y los problemas a la hora de utilizar el disco duro externo en el contenedor de MongoDB [22](#).

- `DIR_DATA_RAW` con esta variable se establece la dirección donde se encuentran los datos proporcionados para el reto. Para esta ejecución se le establece la dirección dentro del contenedor TFG, siendo esta `/data/raw/`, ya que es la dirección establecida en los volúmenes del contenedor para proporcionarle acceso a los datos fuera del contenedor.
- `SEED` con el que se establece el valor de la semilla que utilizaremos para el experimento, siendo para este caso 1234.
- `THREAD` que establece el número de hilos con los que se abordara el proceso de extracción de características tanto de la base de datos de train como de test, único punto en el que se ha paralelizado la ejecución, en mi caso lo he ejecutado con 4 hilo ya que en mi opinión es el número óptimo de hilos con los recursos que disponía.
- `PERFORM_METHOD` con esta variable se le establece que parte del código se ejecuta. Existen dos posibilidades ejecutar todo el código con la opción `'ALL'` o como se ha utilizado en mi caso paso por paso utilizando las opciones `'EXTRACT_TRAIN'`, `'DISCRETE'`, `'SELECT'`, `'MODEL'`, `'EXTRACT_TEST'` Y `'PREDICT'`, en este mismo orden.
- `NUM_FEATURES` con esta variable le indicamos el número de características que nos quedaremos en el proceso de selección filter forward, en mi caso he seleccionado solamente 15 debido a que la alta repercusión que tenía la colección `rankingFeatures_su` y el tiempo que suponía la ejecución me limitaba y obligaban a utilizar tan pocas características.

Conforme se ha comentado, ahora ejecutare el código de mi solución por cada una de sus etapas y analizaré los resultados obtenidos en cada una de ellas. Se comenzará por ejecutar el código con la opción `'EXTRACT_TRAIN'`, la cual se ocupa de extraer las características de las muestras suministradas para la base de datos de train. Del proceso de extracción de características, se obtienen dos colecciones `train_features` y `train_total` de las que podemos observar en la tabla 5.1 el número de características de la colección `train_features`, el número de características diferentes de la colección `train_features` y el número acumulado total de la colección `train_total` en este mismo orden y separados por tipo de característica.

De los valores obtenidos en la tabla 5.1, se han obtenido dos ratios. El primero, se obtiene de dividir el número de características diferentes entre el número de características, con el cual puede observarse el porcentaje en tanto por uno que representa cada característica diferente respecto del total y se puede ver los valores en la tabla 5.2. En

	Asm	Byte	Sección	Total
Num. Caract.	4.547.005	502.155.365	61.945	506.764.315
Num. Caract. Dif.	1.016.037	66.049	610	1.082.696
Num. Total Caract.	3.193.596.229	14.026.873.393	3.180.566.078	20.401.035.700

Tabla 5.1: Número de características por tipo de la base de datos de train.

el segundo, se obtiene en número acumulado total en media de ocurrencias para ese tipo de características y se puede ver los valores en la tabla 5.3.

	Asm	Byte	Sección	Total
Ratio	0,223452	0,000132	0,009847	0,002136

Tabla 5.2: Ratio entre características diferentes y características por tipo de la base de datos de train.

	Asm	Byte	Sección	Total
Ratio	3.143,19	212.370,72	5.214.042,75	18.842,81

Tabla 5.3: Ratio entre el acumulado total de las características y las características diferentes por tipo de la base de datos de train.

De los valores obtenidos de los ratios, podemos observar que las características de tipo asm tienen una repercusión muy alta respecto del total, haciéndose también evidente en su alto número de características diferentes. Además, en el ratio de los acumulados podemos observar que la media para este tipo de características es mucho menor que el de los otros tipos de características. De estas observaciones, podemos concluir que existe una alta probabilidad de que en las características de tipo asm exista ruido.

A continuación, pasaremos a la siguiente etapa de mi solución, que no es otra que el proceso de discretización de las características obtenida, debido a que los valores de estas son valores continuos. Una vez realizado el proceso de discretización, mediante el cual se descartan las características que poseen dos o menos elementos, esto es así ya que en el caso de que solo tenga un elemento o los dos sean iguales las variables no tienen incertidumbre y no sirven para clasificar, en el caso de dos elementos y ambos con distinta etiqueta nos encontramos con una incertidumbre total, haciendo que esas características tampoco sean útiles a la hora de clasificar. Por lo cual, una vez terminado de discretización obtenemos que el número de características por tipo es el que se puede observar en la tabla 5.4.

	Asm	Byte	Sección	Total
Num. Caract.	201.523	66.049	213	267.785

Tabla 5.4: Número de características discretizadas por tipo de la base de datos de train.

De estos datos, se ha generado un ratio para observar el porcentaje en tanto por uno que representa las características que se han quedado después del proceso de discretización, pudiendo verse en la tabla 5.5.

	Asm	Byte	Sección	Total
Ratio	0,198342	1,000000	0,349180	0,247332

Tabla 5.5: Ratio entre número de características distintas y número de características discretizadas por tipo de la base de datos de train.

En este ratio, se puede observar que se ha seleccionado no llega a un 20 % de las características de tipo asm, hecho que no sorprende ya que se había comentado anteriormente que existía una probabilidad muy alta de que existiera ruido en ese tipo de características. El hecho que si sorprende es la existencia de ruido dentro de las características de tipo sección, de la cuales se ha seleccionado solamente algo menos de un 35 %. Otra hecho sorprendente es que nos se haya descartado ninguna característica de tipo byte.

SELECCION DE VARIABLES.

Posición	Característica	Información Mutua
0	a_call __encoded_null	0,29614472313950563
1	a_amhhh	0,5949614041910611
2	a_aawior	0,6976168373317981
3	a_aawiorasmxsdl	0,3193735025963903
4	a_call sub_431a09	-0,05871628941614593
5	b_?? 1a	-0.43731234185149903
6	a_call sub_401100	-0.812551447732242

Tabla 5.6: Tabla de características seleccionadas.

Llegado a este punto, se ha terminado el proceso de selección de variables y empezaremos la construcción del modelo. De esta etapa nos obtenemos el valor de la métrica mlogloss 18 de nuestro modelo. el cual compara el valor obtenido en el ranking publico de las soluciones presentes en los antecedentes, pudiendo observarlos en la tabla 5.7.

	Solución 1 3.1	Solución 2 3.2	Mi Solución
Ranking Público	0,023121984	0,00598443	

Tabla 5.7: Comparación de los resultados de las distintas soluciones.

Ahora, es el turno de extraer las características de la base de datos de test, del cual, al igual que ocurrió con este proceso con la base de datos de train, se han obtenido dos colecciones `test_features` y `test_total`, de las que podemos observar en la [tabla 5.8](#) el número de características de la colección `test_features`, el número de características diferentes de la colección `test_features` y el número acumulado total de la colección `test_total` en este mismo orden y separados por tipo de característica.

	Asm	Byte	Sección	Total
Num. Caract.	4.296.105	501.182.428	61.784	505.540.317
Num. Caract. Dif.	978.646	66.049	645	1.045.340
Num. Total Caract.	3.361.243.484	14.019.613.237	3.350.741.732	20.731.598.453

Tabla 5.8: Número de características por tipo de la base de datos de test.

Del mismo modo que en el proceso de extracción de características de la base de datos train, se han generados dos ratios que se pueden observar en las [tablas 5.9](#) y [5.10](#). En los cuales, se puede observar que nos encontramos con resultados muy parecidos a los obtenidos en sus equivalentes ratios para la base de datos de train que se pueden observar en las [tablas 5.2](#) y [5.3](#). De lo cual, podemos concluir que también nos encontramos con ruido, aunque en este caso no es problema, debido a que solo se utilizarán las características seleccionadas en el proceso anterior.

	Asm	Byte	Sección	Total
Ratio	0,227798	0,000132	0,010440	0,002068

Tabla 5.9: Ratio entre características diferentes y características por tipo de la base de datos de test.

Para terminar ya solo, que da el proceso de predicción, el cual genera un fichero csv con la probabilidad de cada muestra para ser clasificada con cada etiqueta, pudiendo ver esta en la [tabla 5.11](#).

	Asm	Byte	Sección	Total
Ratio	3.443,59	212.260,79	5.194.948,42	19.832,40

Tabla 5.10: Ratio entre el acumulado total de las características y las características diferentes por tipo de la base de datos de test.

Muestra	1	2	3	4	5	6	7	8	9
---------	---	---	---	---	---	---	---	---	---

Tabla 5.11: Tabla con la predicción realizada para la base de datos de test.

Capítulo 6

CONCLUSIONES Y PROPUESTAS

6.1. Conclusiones

6.2. Trabajo futuro

Algunas posibles propuestas, como trabajo futuros para mejorar esta solución, sería diseñar un algoritmo para la optimización de los parámetros que construyen el modelo. Otra propuesta, sería mejorar la paralización del código de las partes en las que ya se utiliza y utilizarlo en otros lugares, y su adaptación para la ejecución en un cluster. Para este proceso se me ha ocurrido la utilización de microservicios y la orquestación de los mismos, de hecho en un principio también estaba contemplada la opción pero al final se dejó para trabajos futuros. Todos estos trabajos futuros, son necesarios para este último que nada menos que ejecutar el código con 4 ngrams [1](#) de binario igual que se utilizó con la solución [2](#) [3.2](#) de los antecedentes y comparar los resultados obtenidos por las dos soluciones.

Bibliografía

- [1] What is malware? - palo alto networks. Palo Alto Networks, Inc. [Online]. Available: <https://www.paloaltonetworks.com/cyberpedia/what-is-malware> 1
- [2] H. Anderson. Defcon-25-hyrum-anderson-evading-next-gen-av-using-ai-updated.pdf. DEF CON Communications, Inc. [Online]. Available: <https://media.defcon.org/DEFCON25/DEFCON25presentations/DEFCON-25-Hyrum-Anderson-Evading-Next-Gen-AV-Using-AI-UPDATED.pdf> 1
- [3] (2015) Microsoft Malware Classification Challenge (BIG 2015). Microsoft Malware Protection Center and Microsoft Azure Machine Learning and Microsoft Talent Management. [Online]. Available: <https://www.kaggle.com/c/malware-classification> 2
- [4] R. Ronen, M. Radu, C. Feuerstein, E. Yom-Tov, and M. Ahmadi, “Microsoft malware classification challenge,” *arXiv preprint arXiv:1802.10135*, 2018. 2
- [5] Scikit-learn/README.rst at master · scikit-learn/scikit-learn · github. [Online]. Available: <https://github.com/scikit-learn/scikit-learn/blob/master/README.rst> 7
- [6] sklearn.feature_extraction.dictvectorizer - scikit-learn 0.19.1 documentation. Scikit-learn developers. [Online]. Available: http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.DictVectorizer.html 7
- [7] R. Montiel Soto, Y. Ledeneva, R. A. García-Hernández, and R. Cruz Reyes, “Comparación de tres modelos de texto para la generación automática de resúmenes,” *Procesamiento del Lenguaje Natural*, no. 43, 2009. 8
- [8] D. D. Lee and H. S. Seung, “Learning the parts of objects by non-negative matrix factorization,” *Nature*, vol. 401, no. 6755, p. 788, 1999. 8
- [9] G. Cobo, X. Sevillano, F. Alías, and J. C. Socoró, “Técnicas de representación de textos para clasificación no supervisada de documentos.” *Procesamiento del lenguaje natural*, vol. 37, 2006. 8

- [10] sklearn.svm.linearsvc - scikit-learn 0.19.1 documentation. Scikit-learn developers. [Online]. Available: <http://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html> 8
- [11] Numpy. [Online]. Available: <http://www.numpy.org/> 8
- [12] 17.5. subprocess - subprocess management - Python 3.7.0 documentation. [Online]. Available: <https://docs.python.org/3/library/pickle.html> 8
- [13] 11.1. pickle - python object serialization - Python 2.7.15 documentation. [Online]. Available: <https://docs.python.org/2.7/library/pickle.html> 8
- [14] 11.7. glob — unix style pathname pattern expansion - Python 3.7.0 documentation. [Online]. Available: <https://docs.python.org/3/library/glob.html> 8
- [15] 12.1. pickle — python object serialization - Python 3.7.0 documentation. [Online]. Available: <https://docs.python.org/3/library/subprocess.html> 8
- [16] echo(1): line of text - Linux man page. [Online]. Available: <https://linux.die.net/man/1/echo> 8
- [17] cat(1): concatenate files/print on stdout - Linux man page. [Online]. Available: <https://linux.die.net/man/1/cat> 9
- [18] wc(1) - Linux man page. [Online]. Available: <https://linux.die.net/man/1/wc> 9
- [19] grep(1): print lines matching pattern - Linux man page. [Online]. Available: <https://linux.die.net/man/1/grep> 9
- [20] Python Data Analysis Library; pandas: Python Data Analysis Library. [Online]. Available: <https://pandas.pydata.org/> 9
- [21] pandas.dataframe - pandas 0.23.1 documentation. [Online]. Available: <https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.html> 9
- [22] Wikipedia. Sparse matrix - Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Sparse_matrix 9
- [23] 16.1. os — miscellaneous operating system interfaces - Python 3.7.0 documentation. [Online]. Available: <https://docs.python.org/3/library/os.html> 9
- [24] Joblib: running python functions as pipeline jobs - joblib 0.11 documentation. [Online]. Available: <https://pythonhosted.org/joblib/> 9

- [25] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, “LibLinear: A library for large linear classification,” *Journal of machine learning research*, vol. 9, no. Aug, pp. 1871–1874, 2008. 9
- [26] hickle/README.md at master · telegraphic/hickle · Github. [Online]. Available: <https://github.com/telegraphic/hickle/blob/master/README.md> 9
- [27] Distributed (Deep) Machine Learning Community. Introduction to Boosted Trees - xgboost 0.72 documentation. [Online]. Available: <https://xgboost.readthedocs.io/en/latest/model.html> 9
- [28] Logarithmic Loss | kaggle. [Online]. Available: <https://www.kaggle.com/wiki/LogarithmicLoss> 10
- [29] Multi Class Log Loss | kaggle. [Online]. Available: <https://www.kaggle.com/wiki/MultiClassLogLoss> 10
- [30] 9.6. random — generate pseudo-random numbers - Python 3.7.0 documentation. [Online]. Available: <https://docs.python.org/3/library/random.html> 10
- [31] SciPy library - scyPy.org. [Online]. Available: <https://github.com/scikit-learn/scikit-learn/blob/master/README.rst> 10
- [32] Statistical functions (scipy.stats) - SciPy v1.1.0 Reference Guide. [Online]. Available: <https://docs.scipy.org/doc/scipy/reference/stats.html> 10
- [33] Introduction to mongodb - mongodb manual. MongoDB Inc. [Online]. Available: <https://docs.mongodb.com/manual/introduction/> 10
- [34] JSON. [Online]. Available: <https://www.json.org/> 10
- [35] Docker enterprise edition platform - docker documentation. Docker Inc. [Online]. Available: <https://docs.docker.com/install> 10
- [36] 6.2. re — Regular expression operations - Python 3.7.0 documentation. [Online]. Available: <https://docs.python.org/3/library/re.html> 10
- [37] V. Chevli. Beating the benchmark for Microsoft Malware Classification Challenge (BIG 2015). [Online]. Available: <https://github.com/vrajs5/Microsoft-Malware-Classification-Challenge> 11
- [38] M. Trofimov, D. Ulyanov, and S. Semenov. Microsoft Malware Classification Challenge third place solution. [Online]. Available: <https://github.com/geffy/kaggle-malware> 13

- [39] Glossary. Microsoft Windows Defender Security Intelligence. [Online]. Available: <https://www.microsoft.com/en-us/wdsi/help/antimalware-security-glossary> 27, 28
- [40] Win32/Ramnit. Microsoft Windows Defender Security Intelligence. [Online]. Available: <https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Win32/Ramnit> 28
- [41] Adware:Win32/Lollipop. Microsoft Windows Defender Security Intelligence. [Online]. Available: <https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Adware:Win32/Lollipop> 28
- [42] Backdoor:Win32/Kelihos.F. Microsoft Windows Defender Security Intelligence. [Online]. Available: <https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Backdoor:Win32/Kelihos.F> 28
- [43] Win32/Vundo. Microsoft Windows Defender Security Intelligence. [Online]. Available: <https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Win32/Vundo> 29
- [44] Win32/Simda. Microsoft Windows Defender Security Intelligence. [Online]. Available: <https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Win32/Simda> 29
- [45] Win32/Tracur. Microsoft Windows Defender Security Intelligence. [Online]. Available: <https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Win32/Tracur> 29
- [46] Win32/Kelihos. Microsoft Windows Defender Security Intelligence. [Online]. Available: <https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Win32/Kelihos> 29
- [47] Virtool:Win32/Obfuscator.ACY. Microsoft Windows Defender Security Intelligence. [Online]. Available: <https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=VirTool:Win32/Obfuscator.ACY> 29
- [48] Win32/Gatak. Microsoft Windows Defender Security Intelligence. [Online]. Available: <https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?name=Win32/Gatak> 29
- [49] R. Ronen, M. Radu, C. Feuerstein, E. Yom-Tov, and M. Ahmadi, “Microsoft malware classification challenge,” *CoRR*, vol. abs/1802.10135, 2018. [Online]. Available: <http://arxiv.org/abs/1802.10135> 30

- [50] M. M. Masud, L. Khan, and B. Thuraisingham, “A scalable multi-level feature extraction technique to detect malicious executables,” *Information Systems Frontiers*, vol. 10, no. 1, pp. 33–45, 2008. 30
- [51] Install docker. Docker Inc. [Online]. Available: <https://docs.docker.com/ee> 32
- [52] Install docker compose. Docker Inc. [Online]. Available: <https://docs.docker.com/compose/install/> 32

CONTENIDO DEL CD

Acompañando al presente TFG, se adjunta un CD, el cual contiene los siguientes recursos:

- Memoria del trabajo en los formatos PDF dentro del directorio Memoria.
- Código fuente del trabajo dentro del directorio Código fuente.
- Imágenes de los contenedores Docker utilizadas dentro del directorio Imágenes Docker.