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

Abril de 2018

UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA SUPERIOR DE INGENIERÍA INFORMÁTICA

GRADO EN INGENIERÍA INFORMÁTICA

TECNOLOGÍA ESPECÁFICA 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

Abril de 2018

La dedicatoria que quiera hacer.

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 6 de abril de 2018

Fdo.: Rubén Donate Serrano

Resumen

Este sería el resumen del TFG.

Agradecimientos

Agradezco el apoyo de mi familia amigos y de mis tutores.

Índice general

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

Capítulo 1

INTRODUCCIÓN

1.1. Motivación

1.2. Objetivos

1.3. Estructura de la memoria

Capítulo 2

TÉCNICAS UTILIZADAS

1. ngram
2. sklearn
    a) randomForrestClassifier
    b) DictVectorizer
    c) TF-IDF [1]
    d) NMF [2] [3]
    e) Linear Support Vector Classification 13
    f) KFold
3. numpy
    a) Concatenate
4. pickle
5. cPickle
6. glob
7. subprocesss
8. funciones linux:
    a) echo
    b) cat
    c) wc
    d) grep
9. pandas

   a) read_csv
   b) to_csv
10. Matriz Dispersa
   a) csrMatrix
11. os
   a) path
12. joblib
   a) parallel
   b) delayed
13. LibLinear: Una librería para completa clasificación lineal [4]
14. hickle
15. XGBoost [5] [6]
16. Gradient Boost Tree
17. logloss
18. Dual Optimization Problem [7]
19. Random
   a) Sample
   b) Choice
20. Bagging
21. Scipy
   a) Stats
      1) rv_discrete

Capítulo 3

ANTECEDENTES Y ESTADO DE LA CUESTIÓN

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

3.1. Solución 1

Esta primera solución que se va a comentar, fue realizada por Vishnu Chevli [8] y en la clasificación del Reto de Kaggle obtuvo una resultado en el ranking publico de 0,023121984 en la posición 90 y en el ranking privado 0,018856579 en la posición 72.

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

En primero lugar, se descomprimir 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 la 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 utilizaran 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 RandomForrestClassifier 2*a*. 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 [9] y en la clasificación del Reto de Kaggle obtuvo una resultado en el ranking publico de 0,005984430 en la posición 14 y en el ranking privado 0,003969846 en la posición 3.

Esta solución es mucho mas compleja como se puede suponer de la posición que obtuvo en el ranking, de hecho según sus creadores utilizaron una maquina con 16 cores y 120 Gb de RAM para su ejecución. Ahora, se analizará en que 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 acabo esto, se recorre cada fichero con extensión '.asm', siendo pasado el nombre de éste por parámetro a la función, y de cada linea, se tiene que coger la primera palabra, estando esta delimitada con dos punto, 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 estas lista se les aplica una función map para dejar solo el nombre del fichero que es el identificador la muestra. Para terminar, se crea una lista de procesos cuya longitud es el número de hilos establecidos. Esto es así para realizar el procesado 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 indice de la posición en la lista original entre el número de hilos que se ejecutan, siendo este el indice 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 lineas 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 8*a* 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 linea 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 lineas 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 lineas del fichero de muestra se utiliza la función cat 8*b* para recorrer el y se le pasa esa información mediante una tubería a la utilidad wc 8*c* con el parametro '-l', la cual cuenta la lineas 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 lineas en las que aparece cada una de las instrucciones en esa muestra. Para conseguir este objetivo, hace uso de la función grep 8*d* para seleccionar las lineas dentro de la muestra que contiene la instrucción correspondiente, para posteriormente pasárselo mediante una tubería a la función wc 8*c*, 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 lineas en las que aparece la cadena de caracteres '__stdcall' y almacenarlas en un fichero por cada muestra. Para llevar esto acabo, hace uso de la función call del paquete subprocess 7 y de la función grep 8*d*, 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 anterior. 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 acabo en el primer paso, creando una sublista que procesara un proceso para todos 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 la muestras las lineas 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 otro 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 la 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 datafame. Este procedimiento consiste en:

En primer lugar, se trataran 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 es 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 datos de train también se crea un array, en el cual se almacena la etiqueta correspondiente de la muestra en el indice establecido por el dataframe. Para continuar, transformando el array de diccionarios en una matriz, para esto usa el modelo DictVectorizer 2*b* del paquete sklearn 2 con el parámetro sparse a false para que no almacenar los datos en una matriz dispersa 10. Este modelo se entran con el array generado con el dataframe de la base de datos de train y posteriormente se transforman los dos array 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 11*a* 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 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 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 2*c* 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 unión de los dos array para posteriormente, transformar estos una matriz dispersa 10 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 2*d*, 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 10, 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 prepocesado 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 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 12 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

le primer paso, y concatenar todos en un nuevo array donde se encuentras todos los n-gramas en formato decimal del fichero ordenados. Una vez tenemos este ultimo array, lo recorremos en grupos de 4 n-gramas, para continuar por transformar el 4 n-grama en un indice número y guardar en un diccionario las ocurrencias de estos indices en el fichero. Para concluir, transformando la información almacenada en diccionario en dos array, uno para los indices y otro para las ocurrencias del mismo. Para ello, recorre las keys del diccionario y almacena en la misma posición en ambos array el indice y el valor de este, respectivamente. Para terminar, almacena en un archivo serializado con pickle 4 un tupla con estos dos array. Este proceso es llevado acabo para las dos bases de datos. Para ello, se obtiene los ficheros que los ficheros en la ubicación de la base de datos con la función glob de glob 6. Un vez tenemos la lista de ficheros de la base de datos, se procesan de manera paralela todos ellos haciendo uso de todos las CPU disponibles utilizando la función parallel 12*a* del paquete joblib 12.

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 indice 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 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 generado en los indices 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 realizar 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 indice original y su posición en el nuevo posición. 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, recorriendolas y si pretenecen al diccionario de características frecuentes se incluye en los nuevos array, ya que se almacena mediante dos array uno primero para almacenarlas características y otro segundo para almacenar el valor de esa característica en la misma posición, del mismo modo que se almaceno en un primero 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 12*a* del paquete joblib 12 para 15 de los 16 CPUs disponibles.

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

dispersa 10, para a continuación pasársela al modelo Linear Support Vector Classification 2*e* 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 cualse 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 indices de los n-gramas que se incluyen en un array llamado índices' y por ultimo en un array llamado 'ptr' donde se almacena el indice 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 de los tres array obtenidos en un archivo serializado con el paquete hickle 14 y a continuación, se crea una matriz dispersa con la función csr_matrix 10*a* del paquete 10, la cual también es guardada en un archivo serializado con el paquete joblib 12.

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 2*e*, por lo cual, se recuperar la matriz dispersa de las características generada en el paso anterior para la base de datos de train, mediante el paquete joblib 12. 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 2*e*, con los siguientes parámetros penality=l1m con la cual se establece que la penalización utiliza en el modelo sea la 'l1' del modulo liblinear 13, max_iter=20 con el que se establece el número máximo de iteracciones que va ha realizar el modelo, dual=False con la cual se establece que se utiliza el problema de optimización dual 18 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 12. Para terminar, transformando las 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 12. Además, las dos nuevas matrices son guardadas en un archivo serializado con le paquete joblib 12, 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 2*a* 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 hpstack 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 9*a* del paquete pandas 9 para almacenarlo en una array de numpy 3 restándole 1 para las etiquetan 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 15. 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 16, 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 17, scale_pos_weight=1.0 con el que se estable que el control del balance de los pesos positivos y negativos que se utiliza habitualmente para desequilibrar la clase [6], 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 generan 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 iteracción, 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 indices de las muestras y una matriz de numpy 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 un lista para pasárselo al modelo. A continuación, se genera un nuevo array que corresponderá

a los indices 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 indices de las muestras que se ha generación al inicio de este punto, haciendo uso de la función sample 19*a* del paquete random 19, para a continuación, añadir de manera aleatoria indices 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 15 los datos. La DMatrix de la base de datos de Train se construye seleccionado los valores de las características de la matriz 'Xtrain' y de la etiquetas de 'ytrain' de esa base de datos con los indices creados en el paso anterior y en el caso de la base de datos de test, la matriz DMatrix se construye únicamente pasandole los valores de 'Xtest'. A continuación, se construye el modelo XGBoost 15 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 indices seleccionados, el numero de rondas establecido en la variable num_roound, 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 predicicones, 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 9*a* 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 9*b* 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 15.

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 amplia 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 9*a* del paquete pandas 9. El siguiente paso, consiste en array de numpy 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 problabilidad para cada una de la muestras de la base de datos de test, usando para ello la función rv_discrete 21*a*1 del paquete stats 21*a* 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 él 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 2*f* 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 3*a* del paguete 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 indice del modelo, max_depth por el cual se establece que la profundidad de los arboles que construirá el modelo sea como máximo de 3, gamma con el que se establece el nivel de perdida mínima necesaria para que se sigue expandiendo un nodo según una función de perdida, 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 generan 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, 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 sera 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 ese 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 iteracción y nthread por el cual se establece que se generen 16 hilos para construcción de los modelos. También, se crea una array vació 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 15 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 resultados en el 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 9*a* del paquete pandas 9, se le asignan los resultados guardado 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 dos los archivos de predicción y el fichero 'sampleSubmission.csv'con la función read_csv 9*a* del paquete pandas 9 para guardar en el dataframe de este ultimo fichero una media pondera de los dos ficheros de predicción obtenidos, siendo un 95 % el ultimo 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

Capítulo 5

EXPERIMENTOS Y RESULTADOS

Capítulo 6

CONCLUSIONES Y PROPUESTAS

6.1. Conclusiones

6.2. Trabajo futuro

Bibliografía

[1] 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. 3

[2] 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. 3

[3] 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. 3

[4] 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. 4

[5] 2018. [Online]. Available: https://github.com/dmlc/xgboost 4

[6] 2018. [Online]. Available: https://xgboost.readthedocs.io/en/latest/parameter.html#parameters-for-tree-booster 4, 15

[7] I. Singer, "A general theory of dual optimization problems," *Journal of Mathematical Analysis and Applications*, vol. 116, no. 1, pp. 77–130, 1986. 4

[8] V. Chevli. (2015) Beating the benchmark for Microsoft Malware Classification Challenge (BIG 2015). [Online]. Available: https://github.com/vrajs5/Microsoft-Malware-Classification-Challenge 5

[9] M. Trofimov, D. Ulyanov, and S. Semenov. (2015) Microsoft Malware Classification Challenge third place solution. [Online]. Available: https://github.com/geffy/kaggle-malware 7

CONTENIDO DEL CD