

**ESCUELA POLITÉCNICA
SUPERIOR DE CÓRDOBA**
Universidad de Córdoba



TRABAJO FIN DE GRADO
Grado en Ingeniería Informática

**Estudio comparativo de métodos de aprendizaje automático
en la detección de malware**

Autor: Manuel Jesús Mariscal Romero
Directores: D. David Guijo Rubio
D. Víctor Manuel Vargas Yun

agosto, 2025



UNIVERSIDAD
DE
CÓRDOBA

Resumen

rellenar

Palabras clave: Palabra-1, Palabra-2, Palabra-3, Palabra-4.

Abstract

rellenar

Keywords: Palabra-1, Palabra-2, Palabra-3, Palabra-4.

Índice general

Resumen	III
Abstract	V
Índice de figuras	XI
Índice de tablas	XIII
Lista de Acrónimos	XV
1. Introducción	1
2. Estado de la técnica	3
2.1. Aprendizaje automático	3
2.1.1. Balanceo de datos	3
2.1.2. Reducción de la dimensionalidad	3
2.1.3. Métricas de evaluación	3
2.2. Ciberseguridad	5
2.2.1. <i>Malware</i>	5
3. Formulación del problema	7
4. Objetivos	9
5. Metodología de trabajo	11
5.1. Enfoque metodológico	11
5.2. Procedimiento seguido	11
5.3. Técnicas y herramientas empleadas	11
5.4. Criterios de selección de datos y modelos.	11
5.4.1. Selección del conjunto de datos	11
5.4.2. Selección de los modelos	12
5.5. Procedimientos de evaluación y validación.	12

6. Desarrollo y experimentación	13
6.1. Modelos utilizados	13
6.2. Procesamiento del conjunto de datos	14
6.2.1. Clasificación multiclase	14
6.2.2. Reducción del conjunto de datos	15
6.3. Preparación del entorno	21
6.3.1. Herramientas y librerías	21
6.3.2. Hardware	22
6.3.3. Métricas de evaluación	23
6.3.4. Protocolo de experimentación y validación	23
6.4. Implementación y pruebas	23
6.4.1. Estructura del código	23
6.4.2. Procedimiento de entrenamiento y evaluación	23
6.4.3. Preparación y uso de los conjuntos de datos	23
6.4.4. Métricas y análisis de resultados	23
7. Resultados y discusión	25
7.1. Clasificación binaria	25
7.1.1. Árboles de decisión	27
7.1.2. <i>Random forest</i>	27
7.1.3. <i>K-NN</i>	27
7.1.4. Máquinas de vectores de soporte	27
7.1.5. <i>Ridge</i>	27
7.1.6. Redes neuronales: Perceptrón multicapa	27
7.1.7. <i>Light Gradient Boosting Machine</i>	27
7.2. Clasificación multiclase	27
7.2.1. Árboles de decisión	27
7.2.2. <i>Random forest</i>	27
7.2.3. <i>K-NN</i>	27
7.2.4. Máquinas de vectores de soporte	27
7.2.5. <i>Ridge</i>	28
7.2.6. Redes neuronales: Perceptrón multicapa	28
7.2.7. <i>Light Gradient Boosting Machine</i>	28
8. Conclusiones y recomendaciones	35
Bibliografía	37

A. Código del programa	39
A.1. Codificación de las categorías <i>malware</i>	39
A.2. Reducción de la dimensionalidad	40
A.3. Pruebas para la elección del conjunto de datos	41

Índice de figuras

6.1. Matriz de confusión para la clasificación multicaso	19
--------------------------------------------------------------------	----

Índice de tablas

6.1. Codificación de las clases <i>malware</i>	15
6.2. Clasificación binaria con <i>PCA</i>	17
6.3. Clasificación binaria con <i>PCA</i> y <i>Undersampling</i>	17
6.4. Clasificación multiclase con <i>PCA</i>	17
6.5. Nueva codificación de las clases <i>malware</i>	21
6.6. Clasificación multiclase con la nueva codificación.	21
7.1. Clasificación binaria con <i>DecisionTreeClassifier</i>	26
7.2. Clasificación binaria con <i>RandomForestClassifier</i>	27
7.3. Clasificación binaria con <i>KNeighborsClassifier</i>	28
7.4. Clasificación binaria con <i>SVC</i>	29
7.5. Clasificación binaria con <i>RidgeClassifier</i>	29
7.6. Clasificación binaria con <i>MLPClassifier</i>	30
7.7. Clasificación binaria con <i>LGBMClassifier</i>	30
7.8. Clasificación multiclase con <i>DecisionTreeClassifier</i>	31
7.9. Clasificación multiclase con <i>RandomForestClassifier</i>	31
7.10. Clasificación multiclase con <i>KNeighborsClassifier</i>	32
7.11. Clasificación multiclase con <i>RidgeClassifier</i>	32
7.12. Clasificación multiclase con <i>MLPClassifier</i>	33
7.13. Clasificación multiclase con <i>LGBMClassifier</i>	33

Capítulo 1

Introducción

CAPÍTULO 1. INTRODUCCIÓN

Capítulo 2

Estado de la técnica

2.1. Aprendizaje automático

2.1.1. Balanceo de datos

2.1.1.1. Sobremuestreo

2.1.1.2. Submuestreo

2.1.2. Reducción de la dimensionalidad

2.1.2.1. Análisis de componentes principales

2.1.2.2. Análisis factorial

2.1.2.3. Descomposición en valores singulares

2.1.3. Métricas de evaluación

La elección de métricas de evaluación adecuadas es esencial para valorar de forma precisa el rendimiento de los modelos. No todas las métricas ofrecen la misma información. En esta sección se revisan las métricas más empleadas en la literatura especializada, destacando su utilidad, limitaciones y el tipo de información que aportan para la comparación de modelos.

2.1.3.1. Exactitud

La exactitud o *Accuracy* se corresponde con el porcentaje de aciertos que se han producido, es decir, los patrones clasificados correctamente respecto al total. Se calcula como la suma de verdaderos positivos (TP) y verdaderos negativos (TN) respecto al número total de patrones de entrada (N) [1].

$$CCR = \frac{TP + TN}{N} \quad (2.1)$$

2.1.3.2. Precisión

La precisión es una métrica que evalúa la proporción de patrones clasificados como positivos que realmente pertenecen a la clase positiva, es decir, mide como de confiable es el modelo cuando predice un positivo. Es muy relevante cuando el coste de clasificar erróneamente un negativo como positivo es alto.

$$\text{Precisión} = \frac{TP}{TP + FP} \quad (2.2)$$

Donde TP representa el número de verdaderos positivos, y FP corresponde al número de falsos positivos.

2.1.3.3. Sensibilidad

También conocida como exhaustividad o *recall* en inglés, mide la capacidad del modelo para detectar correctamente los positivos de un conjunto de datos. Como se muestra en la ecuación 2.3, se calcula como la proporción entre el número de verdaderos positivos (TP) y la suma de verdaderos positivos y falsos negativos (FN) [1]. Un valor alto de sensibilidad indica que se han obtenido pocos falsos negativos.

$$\text{Sensibilidad} = \frac{TP}{TP + FN} \quad (2.3)$$

2.1.3.4. Mínima sensibilidad

La mínima sensibilidad mide cómo de bien se clasifica la clase peor clasificada. Es útil en clasificación multiclase o con conjuntos de datos desbalanceados, ya que permite identificar si existe alguna clase que el modelo no está clasificando correctamente. Un valor alto indica que el modelo mantiene un buen rendimiento en todas las clases, mientras que un valor bajo revela que, al menos, una de ellas presenta un bajo grado de acierto. Si el modelo se deja una clase sin clasificar, el valor será 0.

Sea S_i la sensibilidad de la clase i , con n el número total de clases, la mínima sensibilidad se calcula como se muestra en la ecuación 2.4.

$$MS = \min_{i \in \{1, 2, \dots, n\}} S_i \quad (2.4)$$

Donde la sensibilidad de cada clase S_i se obtiene mediante la ecuación 2.3

CAPÍTULO 2. ESTADO DE LA TÉCNICA

2.1.3.5. Valor-F

El valor-F o *F1-score* mide el equilibrio entre la precisión y la sensibilidad [1]. Se calcula como la media armónica entre ambas, lo que penaliza de forma más severa los valores extremos y proporciona una medida equilibrada del rendimiento del modelo. Es especialmente útil en problemas con clases desbalanceadas, ya que evita que un alto rendimiento en una sola métrica distorsione la evaluación global.

2.2. Ciberseguridad

El MS es la mínima sensibilidad calculada por clases. Es decir, mide cómo de bien se clasifica la clase peor clasificada. Es interesante porque un valor alto indica que el modelo es capaz de clasificar correctamente todas las clases. En el momento en el que haya una clase que se clasifique mal, el valor será bajo. Si por ejemplo el modelo se deja una clase sin clasificar, el valor sería 0.

2.2.1. *Malware*

CAPÍTULO 2. ESTADO DE LA TÉCNICA

Capítulo 3

Formunación del problema

CAPÍTULO 3. FORMUNACIÓN DEL PROBLEMA

Capítulo 4

Objetivos

Debe de ser igual a los mencionados en el anteproyecto.

Describir el objetivo principal y los objetivos específicos llevados a cabo para conseguir el objetivo principal.

CAPÍTULO 4. OBJETIVOS

Capítulo 5

Metodología de trabajo

5.1. Enfoque metodológico

5.2. Procedimiento seguido

5.3. Técnicas y herramientas empleadas

5.4. Criterios de selección de datos y modelos.

5.4.1. Selección del conjunto de datos

En lo que a *malware* se refiere, *BODMAS* [2] es uno de los conjuntos de datos más completos en la actualidad, con la ventaja para este proyecto de ya estar procesado y tener una amplia bibliografía. Otra opción interesante puede ser *VirusShare* [3], ya que cuenta con más de 99 millones de muestras de *malware* actualizadas pero tiene varios inconvenientes para este proyecto. El primero es que no incluye muestras de *software* no malicioso y el segundo que necesita un procesamiento previo para extraer las características. Todo esto conlleva un aumento considerable para la realización del proyecto. Otra de las opciones estudiadas ha sido *theZoo* [4]. En cuanto a este repositorio hemos podido observar que tiene los mismos inconvenientes que *VirusShare* y no tiene sus ventajas. Por último tenemos *Microsoft Malware Classification* [5]. En este caso tenemos un conjunto de datos muy amplio con casi medio *terabyte*, pero además de los inconvenientes ya comentados en los anteriores conjuntos, solo incluye *malware* que afecta a equipos *Windows*, lo que limitaría considerablemente el alcance del estudio.

Teniendo en cuenta todo lo comentado hasta ahora sobre los distintos conjuntos

CAPÍTULO 5. METODOLOGÍA DE TRABAJO

de datos considerados, hemos decidido usar *BODMAS*, ya que es el que mejor se adapta a las necesidades del estudio

5.4.2. Selección de los modelos

5.5. Procedimientos de evaluación y validación.

Capítulo 6

Desarrollo y experimentación

En esta fase se lleva a cabo la implementación práctica del estudio, haciendo uso de los modelos de aprendizaje automático implementados principalmente en la librería *Scikit-Learn* de *python*. Para ello se realiza un procesamiento de los datos, necesario para obtener un conjunto reducido y otro apto para la clasificación multiclase. Además, se configuran los entornos necesarios para su entrenamiento y evaluación, se establecen las métricas de rendimiento, los procedimientos de prueba y los escenarios de experimentación que permitirán obtener resultados consistentes y comparables. El objetivo es verificar, mediante pruebas controladas, la efectividad de cada método en la detección de *malware*.

La parte experimental se aborda desde dos perspectivas complementarias. En primer lugar, se evalúa la capacidad de los modelos para la detección de *malware* mediante pruebas de clasificación binaria, determinando si un patrón corresponde a software malicioso o legítimo. En segundo lugar, se analiza la viabilidad de realizar una clasificación multinivel sobre esos mismos patrones, identificando el tipo específico de *malware* al que pertenecen, lo que permite un análisis más detallado y aplicable a entornos de ciberseguridad avanzada.

6.1. Modelos utilizados

En este proyecto se han empleado diversos algoritmos de aprendizaje automático, seleccionados en función de los criterios mencionados en el capítulo 5 y con el objetivo de representar diferentes enfoques.

Se han utilizado los siguientes modelos implementados en *scikit-learn* y *LightGBM*:

- *DecisionTreeClassifier*
- *RandomForestClassifier*

- *KNeighborsClassifier*
- *RidgeClassifier*
- *MLPClassifier*
- *SVC*
- *LGBMClassifier* (de *LightGBM*)

Todos los modelos se han ajustado y evaluado utilizando *GridSearchCV*, lo que permite explorar sistemáticamente distintas combinaciones de hiperparámetros y asegurar comparaciones consistentes entre los distintos métodos de clasificación. La descripción teórica de estos modelos se presenta en el capítulo 2.

6.2. Procesamiento del conjunto de datos

Dadas las limitaciones *hardware* y la cantidad de datos, aproximadamente 135000 patrones y 2400 atributos por cada patrón, es necesario hacer un procesamiento previo del conjunto de datos. Para ello hemos tenido en cuenta varios enfoques. Por un lado, *BODMAS* nos permite hacer una distinción entre clasificación binaria y clasificación multiclase, pero para ello es necesario reordenar los datos, ya que se encuentran distribuidos en varios archivos. Por otro lado, es necesario reducir la cantidad de datos. A continuación veremos los distintos enfoques.

6.2.1. Clasificación multiclase

El conjunto de datos seleccionado se divide en varios archivos:

- *bodmas.npz*: incluye la matriz de patrones de entrada en formato de matriz de *python* y la matriz de salidas deseadas.
- *bodmas_metadata.csv*: la información relevante para nuestro problema es la columna *sha* que contiene la función *hash* de todo el conjunto de datos.
- *bodmas_malware_category.csv*: contiene la función *hash* del *malware* y la categoría a la que pertenece.

Dado que las distintas categorías se encuentran en formato texto, es necesario codificarlas para poder trabajar con ellas. La codificación elegida ha sido la representada en la tabla 6.1.

CAPÍTULO 6. DESARROLLO Y EXPERIMENTACIÓN

Tabla 6.1: Codificación de las clases *malware*.

Categoría	Codificación	Nº de patrones
<i>benign</i>	0	77142
<i>trojan</i>	1	29972
<i>worm</i>	2	16697
<i>backdoor</i>	3	7331
<i>downloader</i>	4	1031
<i>informationstealer</i>	5	448
<i>dropper</i>	6	715
<i>ransomware</i>	7	821
<i>rootkit</i>	8	3
<i>cryptominer</i>	9	20
<i>pua</i>	10	29
<i>exploit</i>	11	12
<i>virus</i>	12	192
<i>p2p-worm</i>	13	16
<i>trojan-gamethief</i>	14	6

Para obtener una nueva matriz de salidas deseadas que incluya los tipos de *malware*, una vez cargados los datos en sus correspondiente variables de *python*, usamos la función *merge* [6] perteneciente a la clase *pandas.DataFrame* para incluir en *metadata* los datos de *mw_category['category']* en las entradas donde coincide la columna *sha*.

Antes de codificar necesitamos darle una etiqueta a los datos vacíos, los cuales significan que esa muestra es benigna. Para ello usamos la función *pandas.DataFrame.fillna* [7], que nos permite completar datos vacíos de distintas formas. Para nuestro caso usamos la etiqueta *benign*. También eliminamos las columnas que no vamos a necesitar, dejando solo la categoría a la que pertenece cada muestra.

Ahora podemos codificar los datos usando la función *pandas.DataFrame.map* [8]. Este método aplica una función que acepta y devuelve un valor escalar a cada elemento del *DataFrame*, lo que permite asignar un valor numérico a cada clase.

El código utilizado para esta tarea se encuentra en el Anexo A.1.

6.2.2. Reducción del conjunto de datos

Reducir el número de datos con el que vamos a trabajar tiene el objetivo de principal de disminuir el tiempo que los algoritmos van a necesitar para procesar la

CAPÍTULO 6. DESARROLLO Y EXPERIMENTACIÓN

información sin perjudicar la integridad de los datos, ya que los resultados del estudio podrían verse afectados y llevar a unas conclusiones erróneas. Esta tarea se puede enfrentar desde dos planteamientos distintos: condensar el número de patrones o el número de características. Ambos planteamientos se han estudiado de forma teórica en esta memoria en las secciones 2.1.1 y 2.1.2 respectivamente. Las técnicas elegidas son *undersampling* por simplicidad y *PCA* porque según el estudio *A Low Complexity ML-Based Methods for Malware Classification* [9] se obtienen unos resultados algo más precisos que con otros métodos.

El código utilizado se encuentra en el anexo A.2. A continuación se explicarán los pasos seguidos.

6.2.2.1. Número de patrones

Como ya hemos estudiado en la sección 2.1.1.2, el submuestreo o *undersampling* en inglés, es una técnica para abordar el desbalance de clases en un conjunto de datos, especialmente cuando una de las clases tiene muchos más patrones que la otra. En nuestro caso, el desbalance no es demasiado grande ya que *BODMAS* contiene 57293 muestras *malware* y 77142 muestras benignas.

El método *RandomUnderSampler* [10] de la librería *Imbalanced learn* nos permite varias formas de actuar, siendo la que nos interesa para este estudio la que nos permite elegir manualmente el número de patrones de cada clase. Hemos elegido una cantidad de 15000 patrones en por clase.

6.2.2.2. Número de características

Este método, también conocido como reducción de la dimensionalidad, consiste en reducir el número de variables de las que consta el problema. Para aplicar el método matemático-estadístico de análisis de componentes principales, *PCA* por sus siglas en inglés, usamos la clase *PCA* [11] perteneciente a *sklearn.decomposition*. Esta clase nos permite entrenar el modelo y transformar el conjunto de datos tanto para el conjunto de entrenamiento como para el de test. Para ello será necesario separar previamente los datos, ya que *BODMAS* no cuenta con esta división.

6.2.2.3. Elección final del nuevo conjunto de datos

Para poder decidir como será el conjunto de entrenamiento final se han hecho distintos conjuntos de datos sobre los que se probarán algunos algoritmos. Los conjuntos son los siguientes:

- Clasificación binaria con *PCA*.

CAPÍTULO 6. DESARROLLO Y EXPERIMENTACIÓN

- Clasificación binaria con *PCA* y *Undersampling* con 15000 patrones por clase.
- Clasificación multiclase con *PCA*.

Los resultados obtenidos se reflejan en las tablas 6.2, 6.3 y 6.4 respectivamente.

Tabla 6.2: Clasificación binaria con *PCA*.

Clasificador	Tiempo (s)	Entrenamiento			Test		
		Acc	MS	F1	Acc	MS	F1
<i>Decission tree</i>	0.885	1.000	1.000	1.000	0.972	0.971	1.000
<i>Random forest</i>	25.91	1.000	1.000	1.000	0.984	0.976	1.000
<i>K-NN</i>	0.095	0.973	0.970	1.000	0.963	0.963	1.000

Tabla 6.3: Clasificación binaria con *PCA* y *Undersampling*.

Clasificador	Tiempo (s)	Entrenamiento			Test		
		Acc	MS	F1	Acc	MS	F1
<i>Decission tree</i>	0.184	1.000	1.000	1.000	0.945	0.936	1.000
<i>Random forest</i>	4.926	1.000	1.000	1.000	0.963	0.957	1.000
<i>K-NN</i>	0.016	0.954	0.948	1.000	0.938	0.931	1.000

Tabla 6.4: Clasificación multiclase con *PCA*

Clasificador	Tiempo (s)	Entrenamiento			Test		
		Acc	MS	F1	Acc	MS	F1
<i>Decission tree</i>	1.059	0.999	0.895	0.999	0.939	0.000	0.976
<i>Random forest</i>	30.04	0.999	0.895	0.999	0.955	0.000	0.981
<i>K-NN</i>	0.088	0.951	0.000	0.981	0.936	0.000	0.975

En cuanto a la clasificación binaria, hemos decidido usar el conjunto de datos en el que se ha aplicado tanto *PCA* como *undersampling*, ya que, aunque los resultados son similares en ambos conjuntos, el tiempo es considerablemente más bajo y dadas las limitaciones del equipo disponible puede ser beneficioso a la hora de probar algoritmos más complejos.

Para la clasificación multiclase hay varios métodos que podemos usar para reducir el tamaño del conjunto de datos, como el *clustering* o variantes del método de *undersampling* ya utilizado en clasificación binaria. A pesar de ello, estos métodos tienen una mayor complejidad de aplicación y la reducción de las dimensiones no es el

CAPÍTULO 6. DESARROLLO Y EXPERIMENTACIÓN

objeto de este estudio. Por otro lado, esta decisión puede suponer algunos problemas al usar técnicas como *GridSearchCV* o la validación cruzada, ya que incrementan considerablemente el tiempo de entrenamiento.

También en referencia a la clasificación multiclase, podemos ver en la tabla 6.4 que la métrica de mínima sensibilidad es 0 para todos los casos de test. Como ya se ha explicado en esta memoria, mide cómo de bien se clasifica la clase peor clasificada y un valor de 0 indica que alguna de las clases no se ha clasificado bien. Como podemos ver en la matriz de confusión representada en la imagen 6.1, algunas de las clases con menos patrones tienen dificultades para obtener una buena clasificación debido a la falta de información en el entrenamiento. Algunos clasificadores tienen la opción de asignar un peso a los patrones de cada clase inversamente proporcional al número de patrones de la clase, de manera que todas las clases tengan el mismo peso en el entrenamiento, pero no se consiguen mejores resultados.

CAPÍTULO 6. DESARROLLO Y EXPERIMENTACIÓN

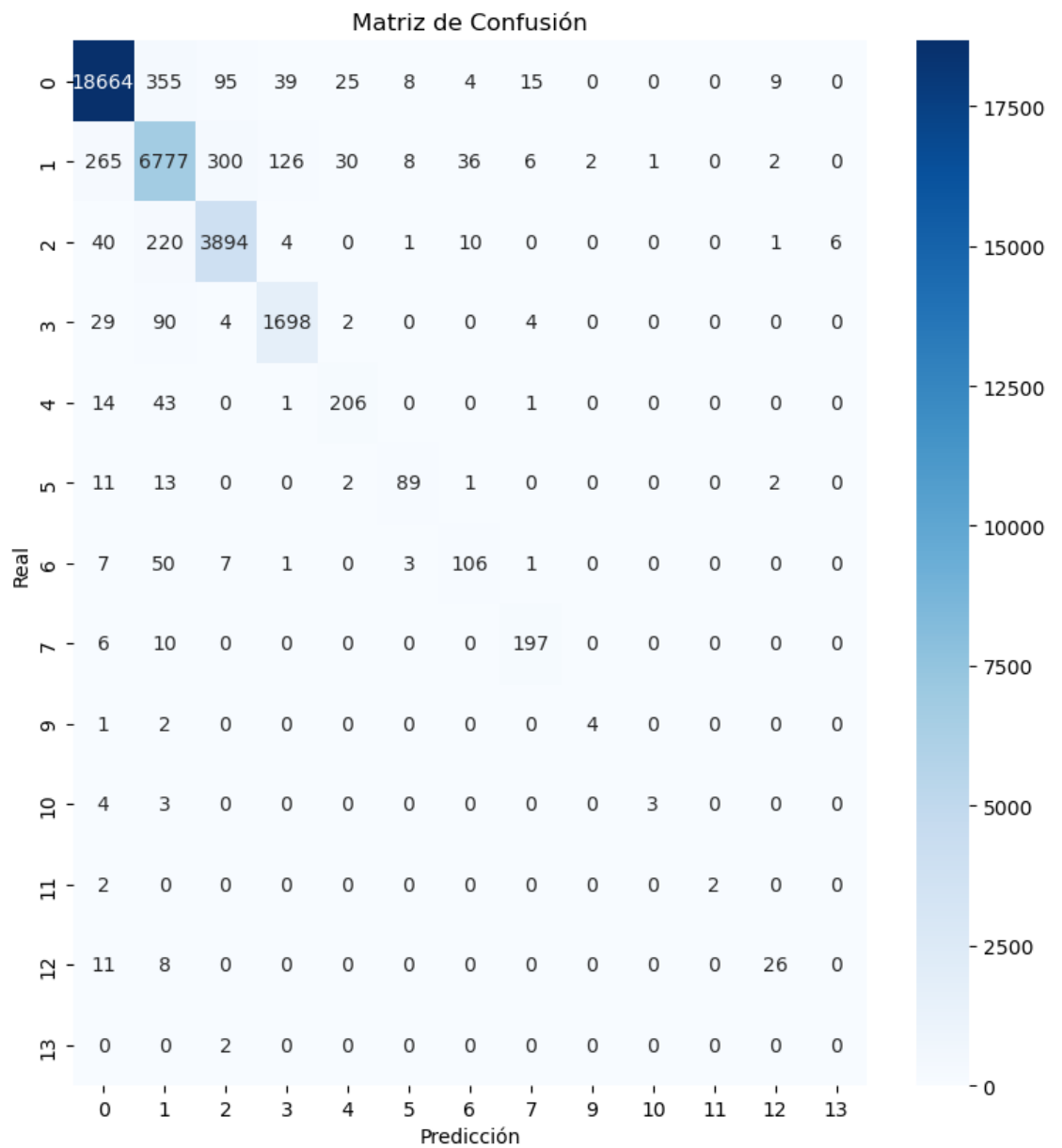


Figura 6.1: Matriz de confusión para la clasificación multicaso

CAPÍTULO 6. DESARROLLO Y EXPERIMENTACIÓN

Según el estudio *Malware Behavior Analysis: Learning and Understanding Current Malware Threats* [12], algunos de los tipos de *malware* que tenemos con menos patrones, se pueden agrupar en algunas de las clases más representadas de nuestro conjunto de datos. En este estudio se comenta que *p2p-worm* añade un comportamiento específico al comportamiento de un gusano, generando problemas de red y de pérdida de datos. Algo similar pasa con *Gamethief trojan*. De esta forma podemos agrupar estos patrones a sus respectivas clases similares sin perder efectividad a la hora de clasificar y además eliminar así dos de las clases que nos pueden dar problemas por falta de información.

Por otro lado, se han planteado dos formas de solucionar este problema, aunque ambas presentan inconvenientes:

- Eliminar las clases menos representadas. Tiene el riesgo de no reconocer un nuevo patrón si es de un tipo distinto de *malware*.
- Agruparlas en una nueva clase que represente varios tipos de *malware*. En este caso estamos suponiendo que los patrones agrupados tienen unas características similares.

Finalmente hemos decidido agrupar las clases con menos de 30 patrones en una nueva categoría *otros*. Por número de patrones sería recomendable agrupar también la clase *virus*, pero podría tener demasiado peso en la categoría *otros* y hemos considerado que es lo suficientemente relevante como para estudiarla por separado. En la tabla 6.6 podemos ver que, aunque mejoramos la mínima sensibilidad, no se producen unas mejoras significativas en la precisión de clasificación pero dada la alta precisión presentada por los modelos y la mejora en la mínima sensibilidad puede considerarse una buena actualización. Podemos ver la nueva codificación en la tabla 6.5

Por último, se han considerado otras opciones para mejorar la clasificación de las clases minoritarias, pero podrían exceder la complejidad de este proyecto:

- Utilizar métodos de sobremuestreo, ya mencionados en la sección 2.1.1.1, que consisten en aumentar la cantidad de patrones de estas clases de forma sintética.
- Utilizar métodos jerárquicos que primero clasifiquen usando la categoría *otros*, para después dividirla en sus diferentes clases y entrenar un modelo específico.

CAPÍTULO 6. DESARROLLO Y EXPERIMENTACIÓN

Tabla 6.5: Nueva codificación de las clases *malware*.

Categoría	Codificación	Nº de patrones
<i>benign</i>	0	77142
<i>trojan</i>	1	29978
<i>worm</i>	2	16713
<i>backdoor</i>	3	7331
<i>downloader</i>	4	1031
<i>informationstealer</i>	5	448
<i>dropper</i>	6	715
<i>ransomware</i>	7	821
<i>virus</i>	8	192
<i>otros</i>	9	64

Tabla 6.6: Clasificación multiclase con la nueva codificación.

Clasificador	Tiempo (s)	Entrenamiento			Test		
		Acc	MS	F1	Acc	MS	F1
<i>Decision tree</i>	1.220	0.998	0.992	0.998	0.938	0.670	0.975
<i>Random forest</i>	31.201	0.998	0.993	0.998	0.953	0.670	0.980
<i>K-NN</i>	0.083	0.951	0.431	0.980	0.936	0.333	0.974

6.3. Preparación del entorno

En esta sección se describe el entorno de trabajo utilizado por el alumno para la implementación de los modelos y la realización de las pruebas. El entorno se debe preparar de forma correcta, ya que puede afectar a la ejecución de los algoritmos y a la reproducibilidad de los experimentos. A continuación, se explican elementos del entorno como el lenguaje de programación, las librerías y las características del equipo.

6.3.1. Herramientas y librerías

El desarrollo y la experimentación de este proyecto se han llevado a cabo empleando un conjunto de herramientas y librerías muy utilizadas en la ciencia de datos. *Python* ha sido el lenguaje de programación de este trabajo, ya que ofrece una fácil implementación de modelos, manipulación de datos y visualización de resultados. Su popularidad se debe a su sintaxis sencilla, escalabilidad y amplia variedad de herramientas y bibliotecas [13].

CAPÍTULO 6. DESARROLLO Y EXPERIMENTACIÓN

En este proyecto se ha utilizado la versión 3.12 de *Python*, elegida principalmente por su compatibilidad con las librerías empleadas, en particular, con *GridSearchCV*, que aprovechan la paralelización de procesos para mejorar el rendimiento. El problema encontrado es que los hilos no se cierran correctamente, es un comportamiento típico asociado a lo que en programación concurrente se denomina *thread leakage* o hilos huérfanos. provoca que la memoria *RAM* y la *CPU* sigan siendo consumidas incluso después de que la ejecución haya terminado. En teoría, este problema no afecta al rendimiento de los clasificadores, pero puede afectar al tiempo de ejecución.

Las librerías utilizadas para construir un modelo y analizar los datos son *Scikit-learn*, *DLOrdinal*, *Matplotlib*, *NumPy*, *Pandas*, *LightGBM* y *Seaborn*.

6.3.1.1. *Scikit-learn*

Scikit-learn es un paquete de código abierto en *Python* que ofrece una gran variedad de métodos de aprendizaje automático rápidos y eficientes, gracias a que usan bibliotecas compiladas en lenguajes como *C++*, *C* o *Fortran*. Tiene detrás una comunidad activa que mantiene la documentación, corrige errores y asegura la calidad. Aunque no incluye todos los algoritmos usados en este proyecto, es una herramienta muy recomendable si necesitamos: transformación de datos, aprendizaje supervisado o evaluación de modelos [14].

6.3.1.2. *DLOrdinal*

dlordinal [15]

6.3.1.3. *Matplotlib*

6.3.1.4. *NumPy*

6.3.1.5. *Pandas*

6.3.1.6. *LightGBM*

6.3.1.7. *Seaborn*

6.3.2. Hardware

El entrenamiento y evaluación de los modelos se ha realizado en el equipo del estudiante con las siguientes características: procesador *Intel Core i7-4712MQ*, tarjeta gráfica *NVIDIA GeForce 920M*, 16 GB de *RAM DDR3* y almacenamiento compuesto por un *SSD Crucial MX500* de 250 GB y un *HDD* de 1 TB. Este hardware permite la paralelización de los algoritmos en múltiples núcleos del procesador, lo

que reduce significativamente los tiempos de entrenamiento, pero se encuentra muy limitado respecto al conjunto utilizado para clasificación multiclase y modelos más costosos como puede ser *SVM*.

6.3.3. Métricas de evaluación

Para evaluar la efectividad de los modelos implementados, se han utilizado las siguientes métricas, cuya descripción teórica se encuentra en la sección 2.1.3:

- **Accuracy**: proporción de predicciones correctas sobre el total de patrones.
- **Mínima Sensibilidad**: sensibilidad de la clase peor clasificada.
- **F1-score**: media armónica entre precisión y sensibilidad.

6.3.4. Protocolo de experimentación y validación

texto

6.4. Implementación y pruebas

6.4.1. Estructura del código

6.4.2. Procedimiento de entrenamiento y evaluación

6.4.3. Preparación y uso de los conjuntos de datos

6.4.4. Métricas y análisis de resultados

CAPÍTULO 6. DESARROLLO Y EXPERIMENTACIÓN

Capítulo 7

Resultados y discusión

7.1. Clasificación binaria

En esta fase se lleva a cabo la implementación práctica del estudio, haciendo uso de los modelos de aprendizaje automático implementados principalmente en la librería *Scikit-Learn* de *python* y descritos en el capítulo 5. Para ello se configuran los entornos necesarios para su entrenamiento y evaluación, se establecen las métricas de rendimiento, los procedimientos de prueba y los escenarios de experimentación que permitirán obtener resultados consistentes y comparables. El objetivo es verificar, mediante pruebas controladas, la efectividad de cada método en la detección de *malware*.

La parte experimental de este proyecto se estudiará desde dos enfoques distintos. Por un lado se evaluarán los modelos seleccionados en la detección de *malware*, es decir, se realizarán pruebas de clasificación binaria donde se estudiará si un patrón corresponde a un programa malicioso o no. Por otro, se estudiará si, para estos mismos patrones, es posible realizar una clasificación más exhaustiva y reconocer con que tipo de *malware* se corresponde cada patrón.

CAPÍTULO 7. RESULTADOS Y DISCUSIÓN

Tabla 7.1: Clasificación binaria con *DecisionTreeClassifier*

Estado aleatorio	Entrenamiento			Test		
	Acc	MS	F1	Acc	MS	F1
0	1.000	1.000	1.000	0.951	0.942	1.000
1	1.000	1.000	1.000	0.944	0.935	1.000
2	1.000	1.000	1.000	0.942	0.935	1.000
3	1.000	1.000	1.000	0.948	0.938	1.000
4	1.000	1.000	1.000	0.953	0.946	1.000
5	0.998	0.997	1.000	0.947	0.936	1.000
6	0.998	0.997	1.000	0.947	0.941	1.000
7	1.000	1.000	1.000	0.949	0.946	1.000
8	0.999	0.998	1.000	0.948	0.937	1.000
9	1.000	1.000	1.000	0.950	0.940	1.000
Mean	0.999	0.999	1.000	0.948	0.940	1.000
STD	0.001	0.001	0.000	0.003	0.004	0.000

CAPÍTULO 7. RESULTADOS Y DISCUSIÓN

Tabla 7.2: Clasificación binaria con *RandomForestClassifier*

Estado aleatorio	Entrenamiento			Test		
	Acc	MS	F1	Acc	MS	F1
0	0.980	0.928	0.989	0.939	0.524	0.972
1	0.980	0.929	0.990	0.939	0.500	0.973
2	0.980	0.927	0.989	0.941	0.429	0.974
3	0.979	0.923	0.989	0.939	0.444	0.973
4	0.982	0.931	0.990	0.939	0.500	0.974
5	0.980	0.929	0.990	0.937	0.609	0.971
6	0.978	0.920	0.988	0.938	0.550	0.971
7	0.980	0.929	0.990	0.939	0.562	0.972
8	0.982	0.934	0.991	0.938	0.489	0.971
9	0.989	0.956	0.992	0.936	0.400	0.972
Mean	0.981	0.931	0.990	0.939	0.501	0.972
STD	0.003	0.010	0.001	0.001	0.064	0.001

7.1.1. Árboles de decisión

7.1.2. *Random forest*

7.1.3. *K-NN*

7.1.4. Máquinas de vectores de soporte

7.1.5. *Ridge*

7.1.6. Redes neuronales: Perceptrón multicapa

7.1.7. *Light Gradient Boosting Machine*

7.2. Clasificación multiclase

7.2.1. Árboles de decisión

7.2.2. *Random forest*

7.2.3. *K-NN*

7.2.4. Máquinas de vectores de soporte

En este caso, el proceso de entrenamiento presentó una mayor complejidad y dificultad para obtener resultados comparables con los de otros modelos evaluados,

CAPÍTULO 7. RESULTADOS Y DISCUSIÓN

Tabla 7.3: Clasificación binaria con *KNeighborsClassifier*

Estado aleatorio	Entrenamiento			Test		
	Acc	MS	F1	Acc	MS	F1
0	1.000	1.000	1.000	0.947	0.935	1.000
1	1.000	1.000	1.000	0.949	0.938	1.000
2	1.000	1.000	1.000	0.939	0.926	1.000
3	1.000	1.000	1.000	0.949	0.937	1.000
4	1.000	1.000	1.000	0.949	0.936	1.000
5	1.000	1.000	1.000	0.946	0.932	1.000
6	1.000	1.000	1.000	0.946	0.931	1.000
7	1.000	1.000	1.000	0.944	0.934	1.000
8	1.000	1.000	1.000	0.947	0.935	1.000
9	1.000	1.000	1.000	0.949	0.940	1.000
Mean	1.000	1.000	1.000	0.947	0.934	1.000
STD	0.000	0.000	0.000	0.003	0.004	0.000

principalmente debido a las limitaciones del equipo utilizado. El elevado tiempo requerido para el entrenamiento sin ajuste de parámetros, junto con los resultados poco satisfactorios obtenidos para las dos semillas empleadas —con una precisión aproximada del 20 %—, motivaron la decisión de no continuar con las máquinas de vectores de soporte para la clasificación multiclase. No obstante, estos resultados no indican que el modelo sea inadecuado para el problema planteado, sino que tiene una mayor exigencia en cuanto a los recursos necesarios para su entrenamiento.

7.2.5. *Ridge*

7.2.6. Redes neuronales: Perceptrón multicapa

7.2.7. *Light Gradient Boosting Machine*

CAPÍTULO 7. RESULTADOS Y DISCUSIÓN

Tabla 7.4: Clasificación binaria con *SVC*

Estado aleatorio	Entrenamiento			Test		
	Acc	MS	F1	Acc	MS	F1
0	0.757	0.656	1.000	0.764	0.672	1.000
1	0.761	0.671	1.000	0.769	0.681	1.000
2	0.766	0.702	1.000	0.757	0.699	1.000
3	0.762	0.700	1.000	0.766	0.704	1.000
4	0.760	0.684	1.000	0.768	0.696	1.000
5	0.761	0.663	1.000	0.753	0.655	1.000
6	0.762	0.702	1.000	0.762	0.683	1.000
7	0.763	0.699	1.000	0.759	0.697	1.000
8	0.766	0.704	1.000	0.758	0.693	1.000
9	0.760	0.662	1.000	0.758	0.666	1.000
Mean	0.762	0.684	1.000	0.762	0.685	1.000
STD	0.003	0.020	0.000	0.005	0.016	0.000

Tabla 7.5: Clasificación binaria con *RidgeClassifier*

Estado aleatorio	Entrenamiento			Test		
	Acc	MS	F1	Acc	MS	F1
0	0.649	0.549	1.000	0.648	0.530	1.000
1	0.645	0.558	1.000	0.655	0.569	1.000
2	0.652	0.573	1.000	0.645	0.564	1.000
3	0.649	0.567	1.000	0.653	0.570	1.000
4	0.651	0.573	1.000	0.651	0.573	1.000
5	0.647	0.562	1.000	0.648	0.558	1.000
6	0.648	0.556	1.000	0.650	0.573	1.000
7	0.651	0.571	1.000	0.650	0.573	1.000
8	0.651	0.564	1.000	0.639	0.551	1.000
9	0.650	0.563	1.000	0.645	0.551	1.000
Mean	0.649	0.564	1.000	0.648	0.561	1.000
STD	0.002	0.008	0.000	0.005	0.014	0.000

CAPÍTULO 7. RESULTADOS Y DISCUSIÓN

Tabla 7.6: Clasificación binaria con *MLPClassifier*

Estado aleatorio	Entrenamiento			Test		
	Acc	MS	F1	Acc	MS	F1
0	0.783	0.771	1.000	0.789	0.778	1.000
1	0.788	0.736	1.000	0.792	0.740	1.000
2	0.788	0.750	1.000	0.782	0.739	1.000
3	0.733	0.605	1.000	0.737	0.609	1.000
4	0.767	0.759	1.000	0.769	0.760	1.000
5	0.790	0.736	1.000	0.783	0.730	1.000
6	0.777	0.772	1.000	0.783	0.781	1.000
7	0.774	0.767	1.000	0.770	0.763	1.000
8	0.778	0.704	1.000	0.772	0.705	1.000
9	0.788	0.762	1.000	0.784	0.751	1.000
Mean	0.776	0.736	1.000	0.776	0.736	1.000
STD	0.017	0.051	0.000	0.016	0.050	0.000

Tabla 7.7: Clasificación binaria con *LGBMClassifier*

Estado aleatorio	Entrenamiento			Test		
	Acc	MS	F1	Acc	MS	F1
0	0.984	0.981	1.000	0.953	0.952	1.000
1	0.984	0.980	1.000	0.951	0.947	1.000
2	0.985	0.983	1.000	0.949	0.946	1.000
3	0.985	0.982	1.000	0.952	0.951	1.000
4	0.984	0.981	1.000	0.950	0.945	1.000
5	0.985	0.981	1.000	0.949	0.948	1.000
6	0.985	0.982	1.000	0.952	0.949	1.000
7	0.986	0.984	1.000	0.948	0.947	1.000
8	0.984	0.979	1.000	0.953	0.952	1.000
9	0.989	0.989	1.000	0.953	0.950	1.000
Mean	0.985	0.982	1.000	0.951	0.949	1.000
STD	0.002	0.003	0.000	0.002	0.002	0.000

CAPÍTULO 7. RESULTADOS Y DISCUSIÓN

Tabla 7.8: Clasificación multiclase con *DecisionTreeClassifier*

Estado aleatorio	Entrenamiento			Test		
	Acc	MS	F1	Acc	MS	F1
0	0.980	0.928	0.989	0.939	0.524	0.972
1	0.980	0.929	0.990	0.939	0.500	0.973
2	0.980	0.927	0.989	0.941	0.429	0.974
3	0.979	0.923	0.989	0.939	0.444	0.973
4	0.982	0.931	0.990	0.939	0.500	0.974
5	0.980	0.929	0.990	0.937	0.609	0.971
6	0.978	0.920	0.988	0.938	0.550	0.971
7	0.980	0.929	0.990	0.939	0.562	0.972
8	0.982	0.934	0.991	0.938	0.489	0.971
9	0.989	0.956	0.992	0.936	0.400	0.972
Mean	0.981	0.931	0.990	0.939	0.501	0.972
STD	0.003	0.010	0.001	0.001	0.064	0.001

Tabla 7.9: Clasificación multiclase con *RandomForestClassifier*

Estado aleatorio	Entrenamiento			Test		
	Acc	MS	F1	Acc	MS	F1
0	0.981	0.926	0.990	0.951	0.524	0.977
1	0.981	0.926	0.990	0.953	0.735	0.978
2	0.981	0.926	0.990	0.954	0.429	0.978
3	0.980	0.923	0.990	0.954	0.500	0.978
4	0.981	0.926	0.990	0.954	0.500	0.979
5	0.981	0.927	0.990	0.952	0.638	0.977
6	0.980	0.923	0.990	0.952	0.550	0.977
7	0.981	0.927	0.991	0.954	0.500	0.978
8	0.981	0.926	0.990	0.953	0.471	0.978
9	0.981	0.926	0.990	0.953	0.400	0.978
Mean	0.981	0.926	0.990	0.953	0.525	0.978
STD	0.000	0.002	0.000	0.001	0.098	0.001

CAPÍTULO 7. RESULTADOS Y DISCUSIÓN

Tabla 7.10: Clasificación multiclase con *KNeighborsClassifier*

Estado aleatorio	Entrenamiento			Test		
	Acc	MS	F1	Acc	MS	F1
0	0.994	0.811	0.997	0.940	0.524	0.976
1	0.994	0.794	0.996	0.943	0.500	0.977
2	0.994	0.811	0.997	0.940	0.357	0.977
3	0.994	0.815	0.996	0.942	0.389	0.978
4	0.994	0.810	0.997	0.941	0.375	0.976
5	0.994	0.807	0.996	0.940	0.435	0.976
6	0.994	0.802	0.996	0.941	0.500	0.976
7	0.994	0.849	0.996	0.940	0.500	0.976
8	0.994	0.817	0.996	0.939	0.529	0.976
9	0.994	0.834	0.996	0.940	0.400	0.976
Mean	0.994	0.815	0.996	0.941	0.451	0.976
STD	0.000	0.016	0.000	0.001	0.067	0.001

Tabla 7.11: Clasificación multiclase con *RidgeClassifier*

Estado aleatorio	Entrenamiento			Test		
	Acc	MS	F1	Acc	MS	F1
0	0.189	0.000	0.301	0.186	0.000	0.299
1	0.195	0.000	0.308	0.191	0.000	0.307
2	0.173	0.000	0.284	0.176	0.000	0.287
3	0.172	0.000	0.283	0.172	0.000	0.280
4	0.185	0.000	0.297	0.189	0.000	0.305
5	0.187	0.000	0.300	0.189	0.000	0.303
6	0.170	0.000	0.280	0.166	0.000	0.274
7	0.186	0.000	0.299	0.191	0.000	0.303
8	0.187	0.000	0.300	0.187	0.000	0.301
9	0.171	0.000	0.282	0.173	0.000	0.284
Mean	0.182	0.000	0.293	0.182	0.000	0.294
STD	0.009	0.000	0.010	0.009	0.000	0.012

CAPÍTULO 7. RESULTADOS Y DISCUSIÓN

Tabla 7.12: Clasificación multiclase con *MLPClassifier*

Estado aleatorio	Entrenamiento			Test		
	Acc	MS	F1	Acc	MS	F1
0	0.725	0.000	0.885	0.722	0.000	0.883
1	0.724	0.000	0.901	0.724	0.000	0.900
2	0.724	0.000	0.885	0.723	0.000	0.885
3	0.679	0.000	0.885	0.681	0.000	0.888
4	0.730	0.000	0.904	0.735	0.000	0.902
5	0.721	0.000	0.888	0.717	0.000	0.884
6	0.724	0.000	0.889	0.723	0.000	0.888
7	0.711	0.000	0.885	0.711	0.000	0.884
8	0.719	0.000	0.910	0.720	0.000	0.910
9	0.716	0.000	0.886	0.718	0.000	0.885
Mean	0.717	0.000	0.892	0.718	0.000	0.891
STD	0.014	0.000	0.009	0.014	0.000	0.009

Tabla 7.13: Clasificación multiclase con *LGBMClassifier*

Estado aleatorio	Entrenamiento			Test		
	Acc	MS	F1	Acc	MS	F1
0	0.938	0.821	0.965	0.916	0.600	0.953
1	0.936	0.820	0.964	0.916	0.735	0.953
2	0.890	0.749	0.941	0.884	0.357	0.936
3	0.323	0.000	0.460	0.327	0.000	0.460
4	0.888	0.747	0.940	0.880	0.500	0.936
5	0.938	0.828	0.967	0.917	0.565	0.955
6	0.893	0.758	0.943	0.881	0.550	0.935
7	0.936	0.821	0.964	0.917	0.562	0.952
8	0.891	0.750	0.941	0.880	0.588	0.933
9	0.893	0.760	0.942	0.883	0.467	0.936
Mean	0.853	0.706	0.903	0.840	0.492	0.895
STD	0.187	0.250	0.156	0.181	0.198	0.153

CAPÍTULO 7. RESULTADOS Y DISCUSIÓN

Capítulo 8

Conclusiones y recomendaciones

CAPÍTULO 8. CONCLUSIONES Y RECOMENDACIONES

Bibliografía

- [1] Joaquim Moré. Evaluación de la calidad de los sistemas de reconocimiento de sentimientos. URL <https://openaccess.uoc.edu/server/api/core/bitstreams/6ff15a78-47c1-45ba-9475-442a6e8d19cc/content>.
- [2] Bodmas. URL <https://whyisyoung.github.io/BODMAS/>.
- [3] Virusshare. URL <https://virusshare.com/>.
- [4] thezoo. URL <https://github.com/ytisf/theZoo>.
- [5] Microsoft malware classification challenge. URL <https://www.kaggle.com/c/malware-classification/data>.
- [6] pandas.dataframe.merge. URL <https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.merge.html>.
- [7] pandas.dataframe.fillna. URL <https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.fillna.html>.
- [8] pandas.dataframe.map. URL <https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.map.html>.
- [9] A low complexity ml-based methods for malware classification. URL https://www.researchgate.net/publication/383827671_A_Low_Complexity_ML-Based_Methods_for_Malware_Classification.
- [10] Randomundersampler. URL https://imbalanced-learn.org/stable/references/generated/imblearn.under_sampling.RandomUnderSampler.html.
- [11] sklearn.decomposition.pca. URL <https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html>.

BIBLIOGRAFÍA

- [12] Mohamad Zolkipli and Aman Jantan. Malware behavior analysis: Learning and understanding current malware threats. URL https://www.researchgate.net/publication/232657598_Malware_Behavior_Analysis_Learning_and_Understanding_Current_Malware_Threats.
- [13] Reema Patel Akshit J. Dhruv and Nishant Doshi. Python: The most advanced programming language for computer science applications. URL <https://www.scitepress.org/Papers/2020/103079/103079.pdf>.
- [14] J. Hao and T. K. Ho. Machine learning made easy: A review of scikit-learn package in python programming language. *Journal of Educational and Behavioral Statistics*, 44(3):348–361, 2019. doi: 10.3102/1076998619832248.
- [15] Francisco Bérchez-Moreno, Rafael Ayllón-Gavilán, Víctor M. Vargas, David Guijo-Rubio, César Hervás-Martínez, Juan C. Fernández, and Pedro A. Gutiérrez. dlordinal: A python package for deep ordinal classification. *Neurocomputing*, 622:129305, 2025. ISSN 0925-2312. doi: <https://doi.org/10.1016/j.neucom.2024.129305>. URL <https://www.sciencedirect.com/science/article/pii/S0925231224020769>.

Anexo A

Código del programa

A.1. Codificación de las categorías *malware*

```
X, y      = load('bodmas/bodmas.npz')
metadata  = pd.read_csv('bodmas/bodmas_metadata.csv')
mw_category = pd.read_csv('bodmas/bodmas_malware_category.csv')

# Incluimos los valores de 'category' en metadata cuando coinciden
# los valores de 'sha'
mw_category = metadata.merge(mw_category, on = 'sha', how = 'left')

# Rellenamos los huecos como software benigno
mw_category['category'] = mw_category['category'].fillna('benign')

# Eliminamos todas las columnas excepto 'category'
mw_category = mw_category['category']

# Codificamos las categorías de malware
category = {
    'benign': 0, 'trojan': 1, 'worm': 2, 'backdoor': 3,
    'downloader': 4, 'informationstealer': 5, 'dropper': 6,
    'ransomware': 7, 'rootkit': 8, 'cryptominer': 9, 'pua': 10,
    'exploit': 11, 'virus': 12, 'p2p-worm': 13, 'trojan-gamethief':
    14
}

mw_category = mw_category.map(category)

y = mw_category.to_numpy()

save('bodmas/bodmas_multiclass.npz', X, y)
```

A.2. Reducción de la dimensionalidad

```
def resampling(X, y, n_components = 5, size = 15000, u = False):  
    if u:  
        rus = RandomUnderSampler(sampling_strategy = {0: size, 1: size  
        })  
        # rus = RandomUnderSampler(sampling_strategy = 'majority')  
        X, y = rus.fit_resample(X, y)  
  
    X_train, X_test, y_train, y_test = train_test_split(  
        X, y, test_size = 0.25, random_state = 1  
    )  
  
    pca = PCA(n_components)  
    X_train = pca.fit_transform(X_train)  
    X_test = pca.transform(X_test)  
  
    return X_train, X_test, y_train, y_test
```

A.3. Pruebas para la elección del conjunto de datos

```
file = {'pca_binary', 'resampling_binary', 'pca_multiclass'}
clf = None

print('clasificador,dataset,n patrones,n características,accuracy,
      tiempo')

for i in range(3):
    if i == 0: clf = DecisionTreeClassifier()
    elif i == 1: clf = RandomForestClassifier()
    else: clf = KNeighborsClassifier()

    for train_file in file:

        X_train, y_train = load('bodmas/' + train_file + '_train.npz')
        X_test, y_test = load('bodmas/' + train_file + '_test.npz')

        # Entrenar el modelo
        inicio = time.time()
        clf.fit(X_train, y_train)
        tiempo = time.time() - inicio

        # Predecir sobre el conjunto de prueba
        y_pred = clf.predict(X_test)

        # Evaluar
        accuracy = accuracy_score(y_test, y_pred)

        print(f'{i},{train_file},{X_train.shape},{accuracy:.3f},{tiempo:.3f}')
```