

**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

**junio, 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	IX
Índice de tablas	XI
Lista de Acrónimos	XIII
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
2.2. Ciberseguridad . . . . .	3
2.2.1. <i>Malware</i> . . . . .	3
3. Formulación del problema	5
4. Objetivos	7
5. Metodología de trabajo	9
6. Desarrollo y experimentación	11
6.1. Selección del conjunto de datos . . . . .	11
6.2. Procesamiento del conjunto de datos . . . . .	12
6.2.1. Clasificación multiclase . . . . .	12
6.2.2. Reducción del conjunto de datos . . . . .	13
7. Resultados y discusión	17

<b>8. Conclusiones y recomendaciones</b>	<b>19</b>
<b>Bibliografía</b>	<b>21</b>
<b>A. Código del programa</b>	<b>23</b>
A.1. Codificación de las categorías <i>malware</i> . . . . .	23
A.2. Reducción de la dimensionalidad . . . . .	24
A.3. Pruebas para la elección del conjunto de datos . . . . .	25

# Índice de figuras



# Índice de tablas

6.1. Codificación de las clases <i>malware</i> . . . . .	13
6.2. Clasificación binaria con <i>PCA</i> . . . . .	14
6.3. Clasificación binaria con <i>PCA</i> y <i>Undersampling</i> . . . . .	15
6.4. Clasificación multiclase con <i>PCA</i> . . . . .	15









# Capítulo 1

## Introducción

En el Capítulo ?? puedes ver códigos de ejemplo de latex para ver como se insertan imágenes, fórmulas, listas enumeradas, etc.

En estas primeras líneas de texto se puede ver como se cita una referencia bibliográfica contenida en el archivo “referencias.bib”. En [?] se hace referencia a un artículo, en [?] se hace referencia a un libro y en [?] se hace referencia a una web.

## 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

#### 2.2. Ciberseguridad

##### 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

## CAPÍTULO 5. METODOLOGÍA DE TRABAJO

---

## Capítulo 6

# Desarrollo y experimentación

### 6.1. Selección del conjunto de datos

En lo que a *malware* se refiere, *BODMAS* [1] 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* [2], 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* [3]. 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* [4]. 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 de datos considerados, hemos decidido usar *BODMAS*, ya que es el que mejor se adapta a las necesidades del estudio

## 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.

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* [5] 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* [6], 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* [7]. 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.

## CAPÍTULO 6. DESARROLLO Y EXPERIMENTACIÓN

---

Tabla 6.1: Codificación de las clases *malware*.

Categoría	Codificación
<i>benign</i>	0
<i>trojan</i>	1
<i>worm</i>	2
<i>backdoor</i>	3
<i>downloader</i>	4
<i>informationstealer</i>	5
<i>dropper</i>	6
<i>ransomware</i>	7
<i>rootkit</i>	8
<i>cryptominer</i>	9
<i>pua</i>	10
<i>exploit</i>	11
<i>virus</i>	12
<i>worm</i>	13
<i>trojan-gamethief</i>	14

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 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* [8] 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,

## CAPÍTULO 6. DESARROLLO Y EXPERIMENTACIÓN

---

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* [9] 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* [10] 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*.
- 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	Precisión	Tiempo (s)
<i>Decission tree</i>	0.89	0.91
<i>Random forest</i>	0.90	24.82
<i>K-NN</i>	0.954	0.088

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

## CAPÍTULO 6. DESARROLLO Y EXPERIMENTACIÓN

---

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

Clasificador	Precisión	Tiempo (s)
<i>Decission tree</i>	0.90	0.17
<i>Random forest</i>	0.94	4.54
<i>K-NN</i>	0.928	0.016

Tabla 6.4: Clasificación multiclase con *PCA*

Clasificador	Precisión	Tiempo (s)
<i>Decission tree</i>	0.77	1.05
<i>Random forest</i>	0.84	28.90
<i>K-NN</i>	0.91	0.090

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 objeto de este estudio. La decisión final ha sido elegir el conjunto con el que se han hecho las pruebas previas ya que los resultados han sido buenos, aunque no tanto como en clasificación binaria.

## CAPÍTULO 6. DESARROLLO Y EXPERIMENTACIÓN

---



## Capítulo 7

# Resultados y discusión

## CAPÍTULO 7. RESULTADOS Y DISCUSIÓN

---

## Capítulo 8

# Conclusiones y recomendaciones

## CAPÍTULO 8. CONCLUSIONES Y RECOMENDACIONES

---

## Bibliografía

- [1] Bodmas. URL <https://whyisyoung.github.io/BODMAS/>.
- [2] Virusshare. URL <https://virusshare.com/>.
- [3] thezoo. URL <https://github.com/ytisf/theZoo>.
- [4] Microsoft malware classification challenge. URL <https://www.kaggle.com/c/malware-classification/data>.
- [5] pandas.dataframe.merge. URL <https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.merge.html>.
- [6] pandas.dataframe.fillna. URL <https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.fillna.html>.
- [7] pandas.dataframe.map. URL <https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.map.html>.
- [8] 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](https://www.researchgate.net/publication/383827671_A_Low_Complexity_ML-Based_Methods_for_Malware_Classification).
- [9] Randomundersampler. URL [https://imbalanced-learn.org/stable/references/generated/imblearn.under\\_sampling.RandomUnderSampler.html](https://imbalanced-learn.org/stable/references/generated/imblearn.under_sampling.RandomUnderSampler.html).
- [10] sklearn.decomposition.pca. URL <https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html>.

## BIBLIOGRAFÍA

---

## 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}')
```