



ANÁLISIS COMPARATIVO DE MODELOS CLASIFICADORES

Para determinar la legitimidad de archivos PE
(Portable Executable) para Windows.

Descripción breve

Utilizar dataset de atributos de archivos PE para detectar posibles programas malignos,
mediante el uso de modelos clasificadores.

Jose Tobar

Universidad Tecnológica de Panamá
Facultad de Ingeniería de Sistemas Computacionales

**Proyecto final: ANÁLISIS COMPARATIVO DE MODELOS
CLASIFICADORES**

Docente:
Juan Castillo

Nombre:

Tobar, José

Maestría en Analítica de Datos

2024

Introducción

El análisis y clasificación de archivos ejecutables portables (PE) para Windows es fundamental en la detección de malware, dado que estos archivos, en formatos como .exe, .dll o .sys, son un vehículo común para la propagación de software malicioso. Identificar la legitimidad de estos archivos es posible utilizando datos recabados con los diferentes atributos de estos tipos de archivo y combinado modelos predictivos. En este trabajo, realizaremos un análisis comparativo del rendimiento modelos clasificadores como: Random Forest, Perceptrón Multicapa y otros, con el fin de determinar su eficacia para identificar archivos PE maliciosos, utilizando herramientas como Python y WEKA.

Contenido

Introducción.....	1
Análisis Comparativo de Modelos Clasificadores para Determinar la Legitimidad de archivos PE (Portable Executable) para Windows.	6
Contexto del problema:.....	6
Objetivo del trabajo.....	6
Descripción del Dataset.....	7
Nombre y origen del dataset	8
Características (features):.....	8
Clase objetivo :.....	9
Ejemplo de datos proporcionados en la primera fila:	9
Preparación de los datos	10
Limpieza de datos	16
Descripción de los Modelos Clasificadores.....	17
Random Forest	17
Regresión Logística	17
Perceptrón Multicapa (Multilayer Perceptron)	17
Prueba de modelos.....	18
Prueba de los modelos con el dataset original solo eliminando filas con columnas con datos en blanco.	18
Random Forest	18
Perceptrón Multicapa (Multilayer Perceptron)	19
Regresión Logística	20
Prueba de los modelos con el dataset modificado eliminando limites extremos.	21
Random Forest	21
Perceptrón Multicapa (Multilayer Perceptron)	22
Regresión Logística	23

Prueba de los modelos con el dataset modificado eliminando columnas.	24
Random Forest	24
Perceptrón Multicapa (Multilayer Perceptron)	25
Regresión Logística	26
Evaluación de los Modelos	27
Métricas de Evaluación	27
Comparación de Resultados	27
Comparación de Resultados	28
Prueba de modelo RandomForest en Python	33
Pruebas de modelo con nuevos datos.....	37
Conclusiones.....	39
Referencias	39

Ilustración 1 ejemplo de Cabeceras de Archivos PE	7
Ilustración 2 Se cambio la columna Legitimate de uno 1 a 0 a “uno” y “cero” ..	10
Ilustración 3 Código para borrar filas con columnas en blanco.	10
Ilustración 4 Detalle del contenido del dataset	10
Ilustración 5 Histograma de 3 columnas del dataset	11
Ilustración 6 Histograma de 3 columnas del dataset	12
Ilustración 7 Histograma de 3 columnas del dataset	13
Ilustración 8 Grafico de dispersión de los atributos vs legitimate	14
Ilustración 9 Visualizar cantidad de datos por categorías de las columnas.....	15
Ilustración 10 proceso para eliminar los limites extremos.	16
Ilustración 11 distribución de los datos luego de luego de la limpieza de datos.	16
Ilustración 12 Resultado de modelo Random Forest.....	18
Ilustración 13 Resultado de modelo Perceptrón Multicapa (Multilayer Perceptron)	19
Ilustración 14 Resultado de modelo Regresión Logística.....	20
Ilustración 15 Resultado de modelo Random Forest con datos limpios	21
Ilustración 16 Resultado de modelo Perceptrón Multicapa (Multilayer Perceptron) con datos limpios.....	22
Ilustración 17 Resultado de modelo Regresión Logísticacon datos limpios	23
Ilustración 18 Resultado de modelo Random Forestcon 6 atributos.	24
Ilustración 19 Resultado de modelo Multilayer Perceptron 6 atributos.	25
Ilustración 20 Resultado de modelo Regresión Logística 6 atributos.	26
Ilustración 21 Primeros pasos para crear el modelo, importar librerías, dividir los datos en atributos, clases, entrenamiento y prueba.	34
Ilustración 22 código para crear el modelo, entrenarlo y evaluar su precisión .	35
Ilustración 23 resultado de las métricas del modelo en Python.....	36
Ilustración 24 Código para generar nuevo archivo con información de aplicaciones ejecutables para Windows.....	37
Ilustración 25 Resultado de pruebas del modelo con datos nuevos.....	38

Tabla 1 resultados con las métricas de las tres pruebas realizadas a cada modelo	28
Tabla 2 Promedio de precisión según el modelo y el tipo de dataset utilizado	31
Tabla 3 Promedio de MAD por modelo y tipo de data set	32
 Grafica 1 Relación entre Precisión y MAD	30
Grafica 2 "Precisión " por "Dataset" y "Modelo"	31
Grafica 3 Tabla 3 Promedio de MAD por modelo y tipo de data	33

Análisis Comparativo de Modelos Clasificadores para Determinar la Legitimidad de archivos PE (Portable Executable) para Windows.

Contexto del problema:

Uno de los aspectos más interesantes para enfrentar el análisis de malware es aprender a distinguir entre archivos binarios legítimos entre aquellos que son potencialmente peligrosos para la integridad de las computadoras y la información que estas contiene.

Generalmente nos referimos a archivos binarios más que archivos ejecutables debido a que el malware puede incluso estar oculto en archivos aparentemente inocentes como imágenes.

Los archivos PE (Portable Executable) son un formato de archivo utilizado en sistemas operativos Windows para los archivos ejecutables, bibliotecas de enlace dinámico (DLL), controladores, entre otros. En esta ocasión trabajaremos con archivos ejecutables de la plataforma Windows estos archivos pueden tener la extensión .exe, .dll o sys.

Por lo tanto, es de fundamental importancia ser capaces de identificar efectivamente la presencia de software malicioso con la intención de prevenir o al menos limitar su diseminación dentro de una organización.

Objetivo del trabajo

Evaluar y comparar el rendimiento Modelos Clasificadores para Determinar la Legitimidad o posibles malware de archivos PE para Windows.

Descripción del Dataset

Los archivos PE (ver ilustración 1) tienen muchas secciones incluidas en la imagen binaria y estas características pueden ser utilizadas para esconder software malicioso.

Por ejemplo, una sección PE puede contener un archivo sys (un controlador malicioso) cuyo objetivo es comprometer el kernel, en conjunto con un archivo que contenga parámetros de configuración o enlaces remotos a los cuales se puede conectar este binario para descargar otros artefactos de activación, puertas traseras u otros.

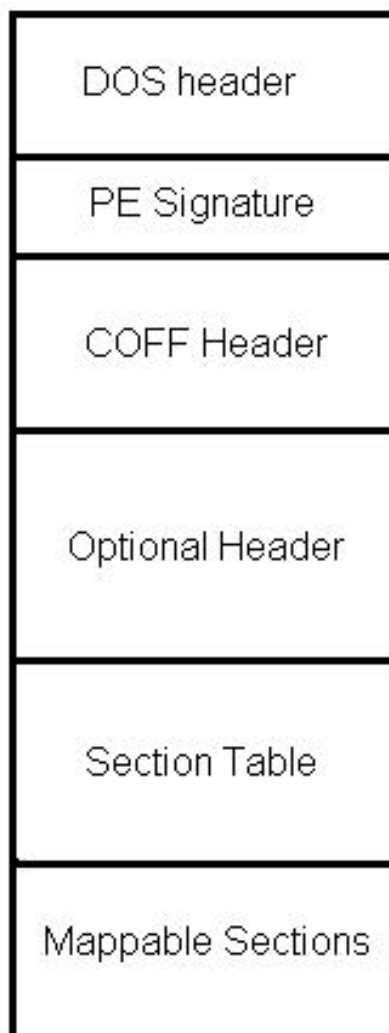


Ilustración 1 ejemplo de Cabeceras de Archivos PE

Nombre y origen del dataset

- MAIwareartifact.CSV
- <https://github.com/PacktPublishing/Hands-On-Artificial-Intelligence-for-Cybersecurity/tree/master/Chapter04/datasets>
- Cuenta con 137445 datos de archivos PE (Portable Executable) con sus distintos atributos y categorías en legítimo (0) o no legítimos (1).

Características (features):

Cabeceras del archivo CSV:

- **AddressOfEntryPoint:** Especifica la dirección (offset) dentro del archivo PE donde comienza la ejecución del código cuando se carga en la memoria.
- **MajorLinkerVersion:** Versión principal del enlazador (programa para combinar objetos en un archivo ejecutable) que se utilizó para crear el archivo ejecutable.
- **MajorImageVersion:** Versión de la imagen del archivo PE.
- **MajorOperatingSystemVersion:** Versión principal del sistema operativo en el cual este archivo está diseñado para ejecutarse.
- **DllCharacteristics:** Un conjunto de características del archivo PE, que puede indicar aspectos como protección de memoria, compatibilidad con ASLR, etc.
- **SizeOfStackReserve:** El tamaño de memoria reservado para la pila del programa.
- **NumberOfSections:** Número de secciones (segmentos) dentro del archivo PE, que pueden contener código, datos o recursos.
- **ResourceSize:** Tamaño de los recursos embebidos dentro del archivo, como íconos, imágenes, etc.
- **Legitimate:** Un valor binario que probablemente indica si el archivo es legítimo (1) o malicioso (0).

Clase objetivo :

El objetivo es clasificar según las características de cada fila en si el programa es legítimo o malicioso según la columna legitimate, siendo legitimo (0) o no legítimos (1).

Ejemplo de datos proporcionados en la primera fila:

- **AddressOfEntryPoint:** 10407: La dirección de entrada donde comenzará la ejecución del archivo.
- **MajorLinkerVersion:** 9: Versión principal del enlazador utilizado.
- **MajorImageVersion:** 6: Versión de la imagen del archivo.
- **MajorOperatingSystemVersion:** 6: Versión principal del sistema operativo.
- **DllCharacteristics:** 33088: Características del archivo PE, codificadas en un valor numérico.
- **SizeOfStackReserve:** 262144: Tamaño reservado para la pila.
- **NumberOfSections:** 4: El archivo PE tiene 4 secciones.
- **ResourceSize:** 952: El tamaño de los recursos es 952 bytes.
- **Legitimate:** 1: Indica que este archivo es legítimo, no malicioso.

Preparación de los datos

Se procedió con el cambio de tipo de dato de la columna Legitimate a tipo cadena en Python ver ilustración 2

```
import pandas as pd

# Cargar el archivo CSV
df = pd.read_csv('/content/MalwareArtifacts.csv')
# Cambiar el tipo de dato de una columna específica de entero a texto
# Reemplaza 'nombre_columna' con el nombre real de la columna
df['legitimate'] = df['legitimate'].replace({1: 'uno', 0: 'cero'})

# Guardar el DataFrame modificado en un nuevo archivo CSV
df.to_csv('MalwareArtifacts_modificado', index=False)

print("Archivo CSV guardado exitosamente.")
```

Ilustración 2 Se cambio la columna Legitimate de uno 1 a 0 a “uno” y “cero”

Se procede a borrar filas con columnas en blanco ver ilustración 3 y se visualiza el resumen del dataset, encontrado cantidad de archivos números promedios máximos y mínimos de las diferentes columnas ver ilustración 3, aparte el detalle de el tipo de dato de cada columna.

```
[ ] # prompt: eliminar filas en blanco de datos

datos = datos.dropna()
datos.info()
```

Ilustración 3 Código para borrar filas con columnas en blanco.

	AddressOfEntryPoint	MajorLinkerVersion	MajorImageVersion	MajorOperatingSystemVersion	DllCharacteristics	SizeOfStackReserve	NumberOfSections	ResourceSize
count	1.374440e+05	137444.000000	137444.000000	137444.000000	137444.000000	1.374440e+05	137444.000000	1.374440e+05
mean	1.722186e+05	8.620784	68.731876	5.098738	22301.043436	9.306841e+05	4.997119	2.474766e+05
std	3.438014e+06	4.095635	1185.709873	99.437584	15444.753219	5.553175e+05	1.917237	2.129516e+07
min	0.000000e+00	0.000000	0.000000	0.000000	0.000000	0.000000e+00	1.000000	0.000000e+00
25%	1.272100e+04	8.000000	0.000000	4.000000	320.000000	1.048576e+06	4.000000	2.216000e+03
50%	5.300800e+04	9.000000	0.000000	5.000000	33088.000000	1.048576e+06	5.000000	9.640000e+03
75%	6.157800e+04	10.000000	6.000000	5.000000	33088.000000	1.048576e+06	5.000000	2.376250e+04
max	1.074484e+09	255.000000	28619.000000	36868.000000	49504.000000	3.355443e+07	40.000000	4.294903e+09


```
[ ] datos.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 137444 entries, 0 to 137443
Data columns (total 9 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   AddressOfEntryPoint                  137444 non-null  int64
1   MajorLinkerVersion                  137444 non-null  int64
2   MajorImageVersion                   137444 non-null  int64
3   MajorOperatingSystemVersion         137444 non-null  int64
4   DllCharacteristics                   137444 non-null  int64
5   SizeOfStackReserve                  137444 non-null  int64
6   NumberOfSections                    137444 non-null  int64
7   ResourceSize                        137444 non-null  int64
8   legitimate                          137444 non-null  object
dtypes: int64(8), object(1)
memory usage: 9.4+ MB
```

Ilustración 4 Detalle del contenido del dataset

Posterior mente los datos son cargados en la aplicación Weka 3.8.6 para análisis y visualizar su comportamiento mediante los histogramas de los datos. Ver ilustración 5 -8

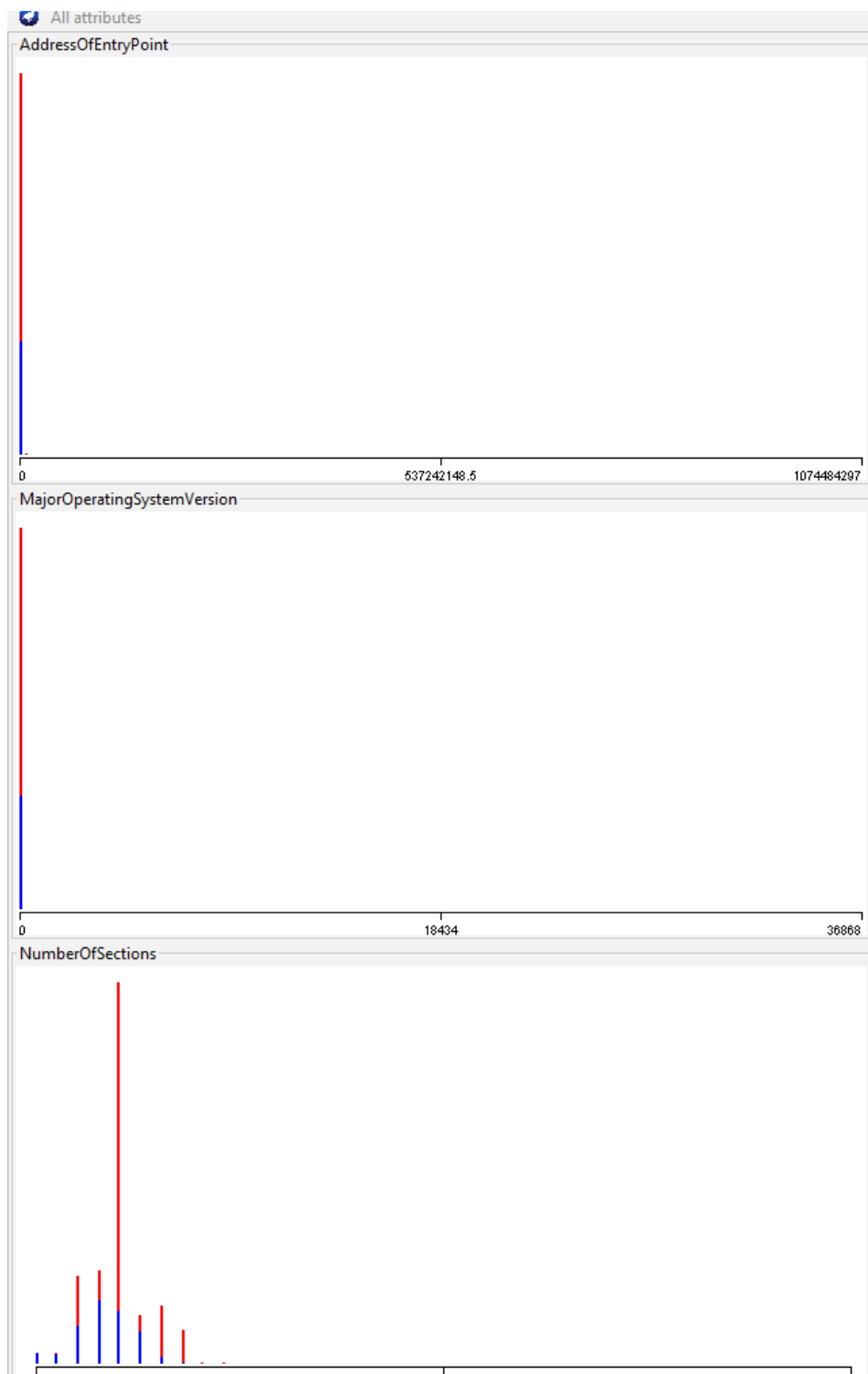


Ilustración 5 Histograma de 3 columnas del dataset

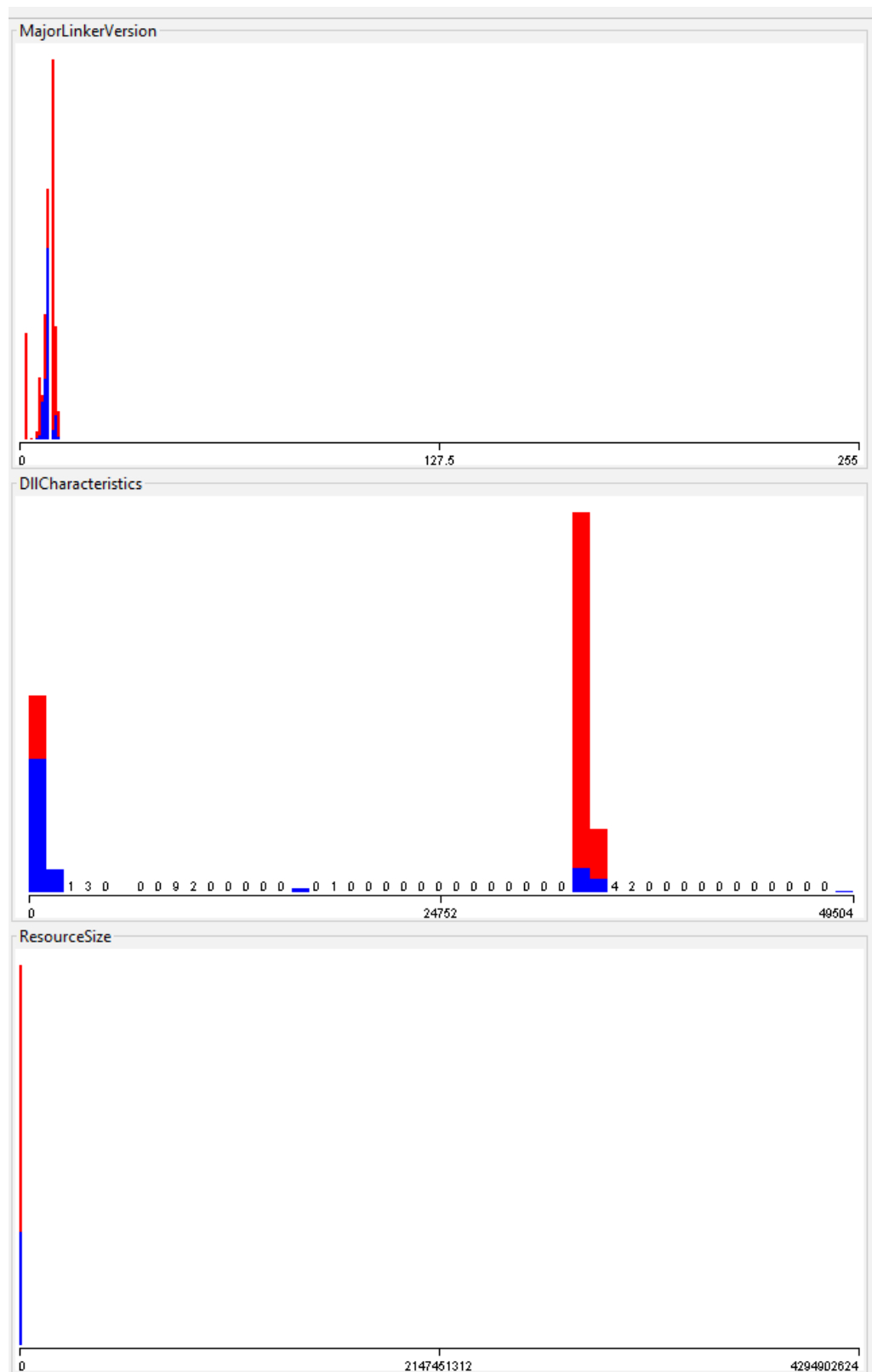


Ilustración 6 Histograma de 3 columnas del dataset

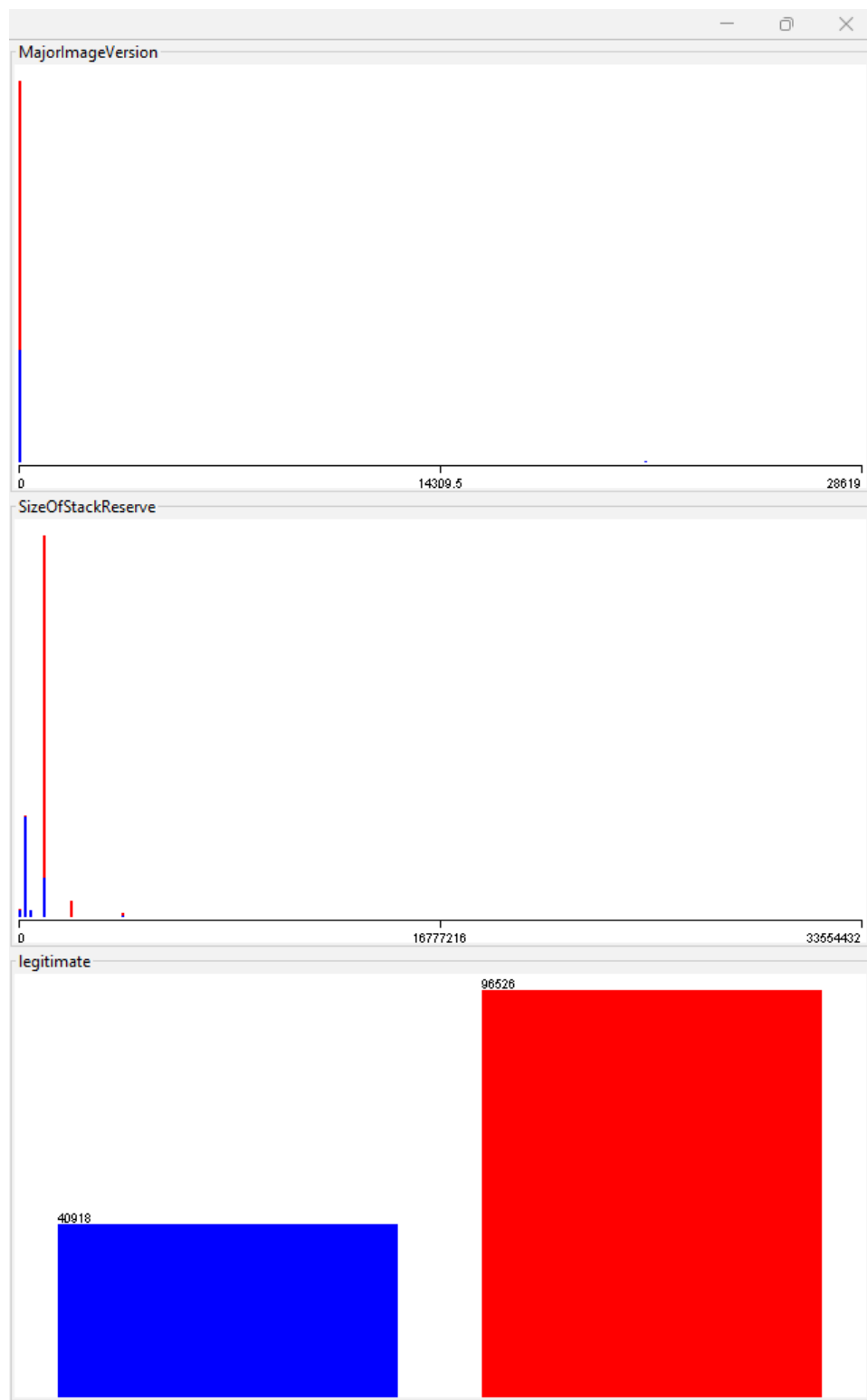


Ilustración 7 Histograma de 3 columnas del dataset



Ilustración 8 Grafico de dispersión de los atributos vs legitimate

Al colocar los datos en Weka se puede ver mediante las gráficas una distribución de los datos donde en la columna legitimate que es nuestra categoría existen 4918 datos que son “cero” y 96526 que son “uno”, también destacan histogramas donde el peso de los datos se ubica a extremos de el histograma lo que podría indicar que puede existir datos no muy representativos para los modelos. Al visualizar en Python podemos ver que estos datos son tipo enteros, pero de categorías, ya que cada uno compone un ejemplo de un archivo PE por lo cual estos datos pueden tener estos tipos de diferencias y como se está comprobando la legitimidad pueden existir archivos con múltiple variedad de características tamaños que pueden ser legítimos o malignos, no presentan una característica que destaque de una de las dos categorías. Ver ilustración 9


```
# prompt: contar categorias de las columnas
```

```
for nombre_col in datos.columns:
    print("Columna:", nombre_col)
    print("cantidad", datos[nombre_col].value_counts())
    print("\n")
```

Columna: SizeOfStackReserve	Columna: AddressOfEntryPoint
cantidad SizeOfStackReserve	cantidad AddressOfEntryPoint
1048576 101316	61532 8787
262144 26529	0 6231
2097152 4412	61578 6176
0 1743	61562 5403
524288 1659	5339 5306
4194304 819	...
32000 149	47166 1
4096 38	1130446 1
327680 36	69230 1
1310720 34	51054 1
10000000 23	8728 1
12582912 22	Name: count, Length: 22622, dtype: int64
65536 19	
16384 12	Columna: MajorOperatingSystemVersion
1000000 9	cantidad MajorOperatingSystemVersion
16777216 7	5 77297
5000000 5	4 30117
131072 4	6 24618
2000000 3	1 4670
10485760 3	10 109
1572864 2	0 43
2048576 2	8 3
1376256 2	7 2
204800 2	9 2
1105920 1	3 2
1081344 1	2 1
1073152 1	Name: count, dtype: int64
983040 1	
3000000 1	
8388608 1	
2162688 1	
1474560 1	
4000000 1	
1100000 1	

Ilustración 9 Visualizar cantidad de datos por categorías de las columnas.

Limpieza de datos

Luego de visualizar se procedió eliminar los límites muy extremos de las columnas, MajorLinkerVersion, MajorOperatingSystemVersion, SizeOfStackReserve en búsqueda de mejorar la distribución de los datos, ver ilustración 10.

```
# Encuentra el valor máximo en la columna deseada
max_valor = datos['MajorOperatingSystemVersion'].max() # Reemplaza 'nombre_columna'
print(max_valor)
# Elimina las filas con el valor máximo
```

36868

+ Código + Texto

```
[ ] datos = datos[datos['MajorOperatingSystemVersion'] != max_valor]
datos.describe()
```

	AddressOfEntryPoint	MajorLinkerVersion	MajorImageVersion	MajorOperatingSystemVersion	DllCharacteristics	SizeOfStackReserve	NumberOfSections	ResourceSize
count	1.372890e+05	137289.000000	137289.000000	137289.000000	137289.000000	1.372890e+05	137289.000000	1.372890e+05
mean	1.722742e+05	8.618389	68.802810	4.829178	22272.548959	9.299476e+05	4.996081	2.476724e+05
std	3.439947e+06	4.096963	1186.377144	0.972151	15426.826649	5.207236e+05	1.917911	2.130718e+07
min	0.000000e+00	0.000000	0.000000	0.000000	0.000000	0.000000e+00	1.000000	0.000000e+00
25%	1.272100e+04	8.000000	0.000000	4.000000	320.000000	1.048576e+06	4.000000	2.216000e+03
50%	5.300800e+04	9.000000	0.000000	5.000000	33088.000000	1.048576e+06	5.000000	9.640000e+03
75%	6.157800e+04	10.000000	6.000000	5.000000	33088.000000	1.048576e+06	5.000000	2.320200e+04
max	1.074484e+09	255.000000	28619.000000	10.000000	36864.000000	1.677722e+07	40.000000	4.294903e+09

Ilustración 10 proceso para eliminar los límites extremos.

Esto generó que el data set pasara a tener 40350 datos que son “cero” y 9614 que son “uno”, y una mejor distribución en ciertas gráficas, ver ilustración 11

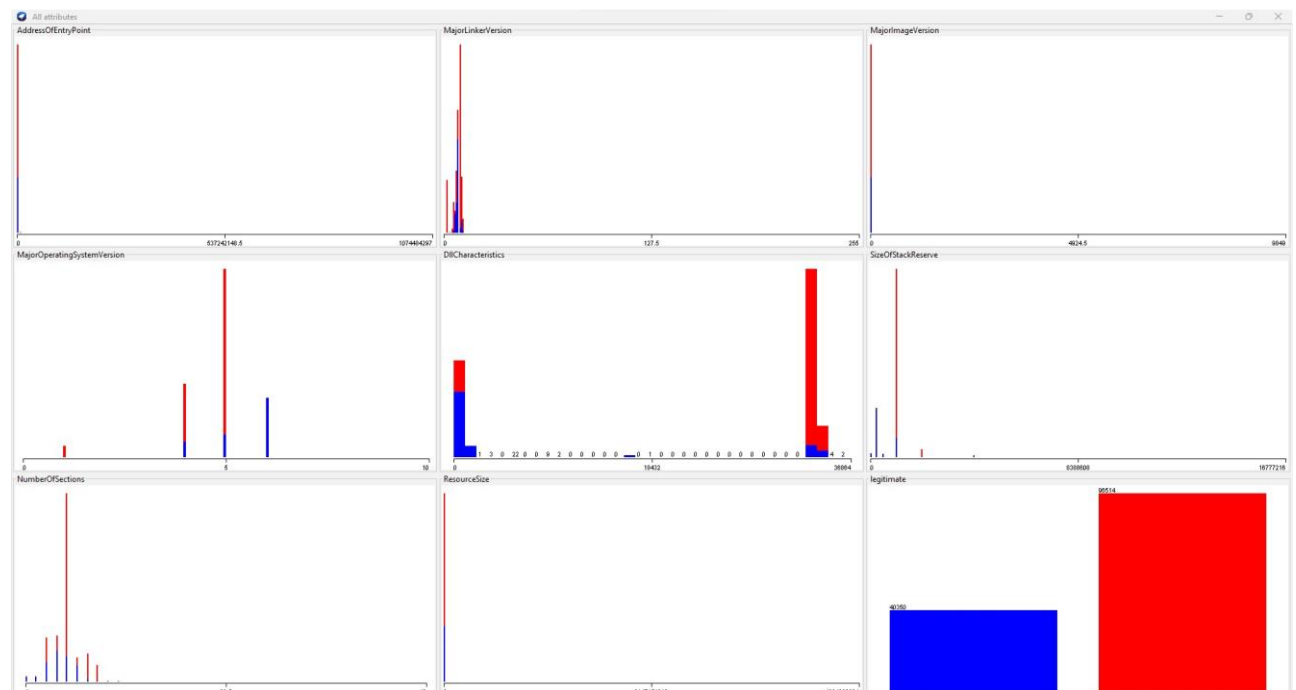


Ilustración 11 distribución de los datos luego de la limpieza de datos.

Descripción de los Modelos Clasificadores

Random Forest

- Descripción del algoritmo: este método combina múltiples árboles de decisión para mejorar la precisión y reducir el sobreajuste al evitar que cualquier árbol individual domine la predicción.
- Aplicación en el dataset: Detalla los parámetros importantes (número de árboles, profundidad máxima, etc.) y cómo se configuró el modelo

Regresión Logística

- Descripción del algoritmo: modelo lineal utilizado para clasificación binaria. Calcula la probabilidad de una clase mediante la aplicación de la función sigmoide a una combinación lineal de las características del dataset.
- Aplicación en el dataset: si las características permiten una separación lineal entre archivos legítimos y maliciosos. Se ajustó la regularización para evitar el sobreajuste y mejorar la generalización en datasets con patrones simples.

Perceptrón Multicapa (Multilayer Perceptron)

- Descripción del algoritmo: Es una red neuronal artificial con múltiples capas de neuronas (una capa de entrada, una o más capas ocultas y una capa de salida). Utiliza la retropropagación para ajustar los pesos y minimizar el error en las predicciones.
- Aplicación en el dataset: icaaz para capturar relaciones no lineales entre las características del archivo PE. se pueden configurar parámetros como el número de capas ocultas y neuronas en cada capa, así como la tasa de aprendizaje y el número de iteraciones (epochs).

Prueba de modelos

Prueba de los modelos con el dataset original solo eliminando filas con columnas con datos en blanco.

Random Forest

El modelo se probó en Weka 3.8.6 con los 9 atributos dando como resultado una precisión de 99.2% ver detalles ilustración 12

```
Scheme:      weka.classifiers.trees.RandomForest -P 100 -I 100 -num-slots 1 -K 0 -M 1.0 -V 0.001 -S 1
Relation:    archivo_salida (2)
Instances:    137444
Attributes:   9
              AddressOfEntryPoint
              MajorLinkerVersion
              MajorImageVersion
              MajorOperatingSystemVersion
              DllCharacteristics
              SizeOfStackReserve
              NumberOfSections
              ResourceSize
              legitimate
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===

RandomForest

Bagging with 100 iterations and base learner

weka.classifiers.trees.RandomTree -K 0 -M 1.0 -V 0.001 -S 1 -do-not-check-capabilities

Time taken to build model: 16.86 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances      136275                99.1495 %
Incorrectly Classified Instances      1169                0.8505 %
Kappa statistic                     0.9797
Mean absolute error                  0.0134
Root mean squared error              0.0819
Relative absolute error              3.1977 %
Root relative squared error          17.9214 %
Total Number of Instances           137444

=== Detailed Accuracy By Class ===

              TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
              0.987    0.007    0.984     0.987    0.986     0.980    0.999    0.997    uno
              0.993    0.013    0.995     0.993    0.994     0.980    0.999    0.999    cero
Weighted Avg.   0.991    0.011    0.992     0.991    0.991     0.980    0.999    0.999

=== Confusion Matrix ===

      a    b  <-- classified as
40402  516 |    a = uno
 653 95873 |    b = cero
```

Ilustración 12 Resultado de modelo Random Forest

Perceptrón Multicapa (Multilayer Perceptron)

El modelo se probó en Weka 3.8.6 con los 9 atributos dando como resultado una precisión de 95.3% ver detalles ilustración 13

```
=== Run information ===

Scheme:      weka.classifiers.functions.MultilayerPerceptron -L 0.3 -M 0.2 -N 500 -V 0 -S 0 -E 20 -H a
Relation:    archivo_salida (2)
Instances:    137444
Attributes:   9
              AddressOfEntryPoint
              MajorLinkerVersion
              MajorImageVersion
              MajorOperatingSystemVersion
              DllCharacteristics
              SizeOfStackReserve
              NumberOfSections
              ResourceSize
              legitimate
Test mode:    10-fold cross-validation
Class uno
  Input
  Node 0
Class cero
  Input
  Node 1

Time taken to build model: 31.72 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances      131064           95.3581 %
Incorrectly Classified Instances     6380           4.6419 %
Kappa statistic                     0.8879
Mean absolute error                  0.0617
Root mean squared error              0.1912
Relative absolute error              14.7668 %
Root relative squared error          41.8256 %
Total Number of Instances           137444

=== Detailed Accuracy By Class ===

              TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
              0.906   0.026   0.936     0.906   0.921     0.888   0.975    0.958    uno
              0.974   0.094   0.961     0.974   0.967     0.888   0.975    0.983    cero
Weighted Avg.   0.954   0.074   0.953     0.954   0.953     0.888   0.975    0.975

=== Confusion Matrix ===

      a    b  <-- classified as
37063  3855 |    a = uno
2525  94001 |    b = cero
```

Ilustración 13 Resultado de modelo Perceptrón Multicapa (Multilayer Perceptron)

Regresión Logística

El modelo se probó en Weka 3.8.6 con los 9 atributos dando como resultado una precisión de 92.9% ver detalles ilustración 14

```
=== Run information ===

Scheme:      weka.classifiers.functions.Logistic -R 1.0E-8 -M -1 -num-decimal-places 4
Relation:    MalwareArtifacts_modificado
Instances:   137444
Attributes:  9
              AddressOfEntryPoint
              MajorLinkerVersion
              MajorImageVersion
              MajorOperatingSystemVersion
              DllCharacteristics
              SizeOfStackReserve
              NumberOfSections
              ResourceSize
              legitimate
Test mode:   10-fold cross-validation

=== Classifier model (full training set) ===

Logistic Regression with ridge parameter of 1.0E-8
Coefficients...

Variable          Class
=====
AddressOfEntryPoint  -0
MajorLinkerVersion   0.127
MajorImageVersion    0.0001
MajorOperatingSystemVersion -0.0001
DllCharacteristics   -0.0001
SizeOfStackReserve   -0
NumberOfSections     -0.3165
ResourceSize         -0
Intercept           6.2876

Odds Ratios...

Variable          Class
=====
AddressOfEntryPoint  1
MajorLinkerVersion   1.1354
MajorImageVersion    1.0001
MajorOperatingSystemVersion 0.9999
DllCharacteristics   0.9999
SizeOfStackReserve   1
NumberOfSections     0.7287
ResourceSize         1

Time taken to build model: 3.45 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances 127733      92.9346 %
Incorrectly Classified Instances 9711      7.0654 %
Kappa statistic 0.8252
Mean absolute error 0.1224
Root mean squared error 0.2368
Relative absolute error 29.278 %
Root relative squared error 51.7859 %
Total Number of Instances 137444

=== Detailed Accuracy By Class ===

          TP Rate  FP Rate  Precision  Recall  F-Measure  MCC  ROC Area  PRC Area  Class
          0.824    0.026    0.931    0.824    0.874    0.828    0.959    0.923    uno
          0.974    0.176    0.929    0.974    0.951    0.828    0.959    0.967    cero
Weighted Avg.  0.929    0.131    0.929    0.929    0.928    0.828    0.959    0.954

=== Confusion Matrix ===

      a    b  <-- classified as
33709 7209 |    a = uno
 2502 94024 |    b = cero
```

Ilustración 14 Resultado de modelo Regresión Logística

Prueba de los modelos con el dataset modificado eliminando limites extremos.

Random Forest

El modelo se probó en Weka 3.8.6 con los 9 atributos dando como resultado una precisión de 99.2% ver detalles ilustración 15

```

Scheme:      weka.classifiers.trees.RandomForest -P 100 -I 100 -num-slots 1 -K 0 -M 1.0 -V 0.001 -S
Relation:    MalwareArtifacts_limpio
Instances:   136864
Attributes:  9
              AddressOfEntryPoint
              MajorLinkerVersion
              MajorImageVersion
              MajorOperatingSystemVersion
              DllCharacteristics
              SizeOfStackReserve
              NumberOfSections
              ResourceSize
              legitimate
Test mode:   10-fold cross-validation

=== Classifier model (full training set) ===

RandomForest

Bagging with 100 iterations and base learner

weka.classifiers.trees.RandomTree -K 0 -M 1.0 -V 0.001 -S 1 -do-not-check-capabilities

Time taken to build model: 82.59 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances      135730                99.1714 %
Incorrectly Classified Instances      1134                0.8286 %
Kappa statistic                     0.9801
Mean absolute error                  0.0133
Root mean squared error              0.0813
Relative absolute error              3.1947 %
Root relative squared error          17.8278 %
Total Number of Instances           136864

=== Detailed Accuracy By Class ===

              TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
              0.988    0.007    0.984     0.988    0.986     0.980    0.999    0.998     uno
              0.993    0.012    0.995     0.993    0.994     0.980    0.999    0.999     cero
Weighted Avg.   0.992    0.011    0.992     0.992    0.992     0.980    0.999    0.999

=== Confusion Matrix ===

      a    b  <-- classified as
39858  492 |      a = uno
 642 95872 |      b = cero

```

Ilustración 15 Resultado de modelo Random Forest con datos limpios

Perceptrón Multicapa (Multilayer Perceptron)

El modelo se probó en Weka 3.8.6 con los 9 atributos dando como resultado una precisión de 95.7% ver detalles ilustración 16

```
=== Run information ===

Scheme:      weka.classifiers.functions.MultilayerPerceptron -L 0.3 -M 0.2 -N 500 -V 0 -S 0 -E 20 -H a
Relation:    MalwareArtifacts_limpio
Instances:   136864
Attributes:  9
              AddressOfEntryPoint
              MajorLinkerVersion
              MajorImageVersion
              MajorOperatingSystemVersion
              DllCharacteristics
              SizeOfStackReserve
              NumberOfSections
              ResourceSize
              legitimate
Test mode:   10-fold cross-validation
Time taken to build model: 93.51 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances      131020           95.7301 %
Incorrectly Classified Instances      5844           4.2699 %
Kappa statistic                     0.8956
Mean absolute error                  0.0559
Root mean squared error              0.192
Relative absolute error              13.4466 %
Root relative squared error          42.1181 %
Total Number of Instances           136864

=== Detailed Accuracy By Class ===
```

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.900	0.019	0.953	0.900	0.926	0.896	0.944	0.948	uno
	0.981	0.100	0.959	0.981	0.970	0.896	0.944	0.943	cero
Weighted Avg.	0.957	0.076	0.957	0.957	0.957	0.896	0.944	0.944	

```
=== Confusion Matrix ===

  a    b  <-- classified as
36307 4043 |    a = uno
1801 94713 |    b = cero
```

Ilustración 16 Resultado de modelo Perceptrón Multicapa (Multilayer Perceptron) con datos limpios

Regresión Logística

El modelo se probó en Weka 3.8.6 con los 9 atributos dando como resultado una precisión de 93.1% ver detalles ilustración 17

```
=== Run information ===

Scheme:      weka.classifiers.functions.Logistic -R 1.0E-8 -M -1 -num-decimal-places 4
Relation:    MalwareArtifacts_limpio
Instances:   136864
Attributes:  9
              AddressOfEntryPoint
              MajorLinkerVersion
              MajorImageVersion
              MajorOperatingSystemVersion
              DllCharacteristics
              SizeOfStackReserve
              NumberOfSections
              ResourceSize
              legitimate
Test mode:   10-fold cross-validation

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances    127380           93.0705 %
Incorrectly Classified Instances    9484           6.9295 %
Kappa statistic                   0.8273
Mean absolute error               0.117
Root mean squared error          0.2324
Relative absolute error           28.1376 %
Root relative squared error       50.979 %
Total Number of Instances        136864

=== Detailed Accuracy By Class ===

              TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
              0.822   0.024   0.935     0.822   0.875     0.831   0.938    0.934    uno
              0.976   0.178   0.929     0.976   0.952     0.831   0.938    0.949    cero
Weighted Avg.   0.931   0.133   0.931     0.931   0.929     0.831   0.938    0.945

=== Confusion Matrix ===

      a    b  <-- classified as
33168  7182 |    a = uno
 2302 94212 |    b = cero
```

Ilustración 17 Resultado de modelo Regresión Logística con datos limpios

Prueba de los modelos con el dataset modificado eliminando columnas.

Se eliminaron las columnas de **AddressOfEntryPoint**, **MajorImageVersion**, **ResourceSize** donde la distribución de los datos no parecía representativa, para realizar la prueba con los distintos modelos.

Random Forest

El modelo se probó en Weka 3.8.6 con 6 atributos dando como resultado una precisión de 97.8% ver detalles ilustración 18

```
=== Run information ===

Scheme:      weka.classifiers.trees.RandomForest -P 100 -I 100 -num-slots 1 -K 0 -M 1.0 -V 0.001 -S 1
Relation:    MalwareArtifacts_limpio-weka.filters.unsupervised.attribute.Remove-R1,3,8
Instances:    136864
Attributes:   6
              MajorLinkerVersion
              MajorOperatingSystemVersion
              DllCharacteristics
              SizeOfStackReserve
              NumberOfSections
              legitimate
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===

RandomForest

Bagging with 100 iterations and base learner

weka.classifiers.trees.RandomTree -K 0 -M 1.0 -V 0.001 -S 1 -do-not-check-capabilities

Time taken to build model: 62.63 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances      133785                97.7503 %
Incorrectly Classified Instances      3079                2.2497 %
Kappa statistic                    0.9463
Mean absolute error                  0.0322
Root mean squared error              0.1277
Relative absolute error              7.7463 %
Root relative squared error          28.0044 %
Total Number of Instances           136864

=== Detailed Accuracy By Class ===

              TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
              0.974    0.021    0.951     0.974    0.962      0.946    0.997     0.993     uno
              0.979    0.026    0.989     0.979    0.984      0.946    0.997     0.998     cero
Weighted Avg.   0.978    0.024    0.978     0.978    0.978      0.946    0.997     0.997

=== Confusion Matrix ===

      a      b  <-- classified as
39311 1039 |      a = uno
 2040 94474 |      b = cero
```

Ilustración 18 Resultado de modelo Random Forest con 6 atributos.

Perceptrón Multicapa (Multilayer Perceptron)

El modelo se probó en Weka 3.8.6 con 6 atributos dando como resultado una precisión de 95.6% ver detalles ilustración 19

```
=== Run information ===

Scheme:      weka.classifiers.functions.MultilayerPerceptron -L 0.3 -M 0.2 -N 500 -V 0 -S 0 -E 20 -H a
Relation:    MalwareArtifacts_limpio-weka.filters.unsupervised.attribute.Remove-R1,3,8
Instances:   136864
Attributes:  6
             MajorLinkerVersion
             MajorOperatingSystemVersion
             DllCharacteristics
             SizeOfStackReserve
             NumberOfSections
             legitimate
Test mode:   10-fold cross-validation

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances   130879           95.627 %
Incorrectly Classified Instances    5985           4.373 %
Kappa statistic                   0.8928
Mean absolute error               0.0634
Root mean squared error          0.1965
Relative absolute error          15.2394 %
Root relative squared error      43.0994 %
Total Number of Instances       136864

=== Detailed Accuracy By Class ===

             TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
             0.893   0.017   0.956     0.893   0.923     0.894   0.948    0.941    uno
             0.983   0.107   0.956     0.983   0.969     0.894   0.948    0.948    cero
Weighted Avg.   0.956   0.081   0.956     0.956   0.956     0.894   0.948    0.946

=== Confusion Matrix ===

      a    b  <-- classified as
36024  4326 |    a = uno
 1659  94855 |    b = cero
```

Ilustración 19 Resultado de modelo Multilayer Perceptron 6 atributos.

Regresión Logística

El modelo se probó en Weka 3.8.6 con 6 atributos dando como resultado una precisión de 95.6% ver detalles ilustración 20

```
Scheme:      weka.classifiers.functions.Logistic -R 1.0E-8 -M -1 -num-decimal-places 4
Relation:    MalwareArtifacts_limpio-weka.filters.unsupervised.attribute.Remove-R1,3,8
Instances:   136864
Attributes:  6
             MajorLinkerVersion
             MajorOperatingSystemVersion
             DllCharacteristics
             SizeOfStackReserve
             NumberOfSections
             legitimate
Test mode:   10-fold cross-validation
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances      127900           93.4504 %
Incorrectly Classified Instances     8964           6.5496 %
Kappa statistic                     0.836
Mean absolute error                  0.1189
Root mean squared error              0.2354
Relative absolute error              28.5948 %
Root relative squared error          51.6186 %
Total Number of Instances           136864

=== Detailed Accuracy By Class ===
```

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.821	0.018	0.950	0.821	0.881	0.840	0.938	0.930	uno
	0.982	0.179	0.929	0.982	0.955	0.840	0.938	0.950	cero
Weighted Avg.	0.935	0.131	0.935	0.935	0.933	0.840	0.938	0.945	

```
=== Confusion Matrix ===

      a      b  <-- classified as
33143  7207 |      a = uno
1757  94757 |      b = cero
```

Ilustración 20 Resultado de modelo Regresión Logística 6 atributos.

Evaluación de los Modelos

Métricas de Evaluación

- **Precisión:** Este valor indica que el modelo clasificó correctamente.
- **Recall:** Es sinónimo de la tasa de verdaderos positivos (TP Rate).
- **MSE (Mean Squared Error):** promedio de los errores al cuadrado, sugiere que, en promedio, las predicciones del modelo se desvían del valor real por una cantidad cuadrática pequeña. El RMSE es la raíz cuadrada del MSE por lo tanto $MSE = (RMSE)^2$.
- **MAD (Mean Absolute Error):** en promedio, las predicciones del modelo están desviadas en unidades respecto a los valores reales. Un error pequeño, indica que el modelo está bastante cerca en sus predicciones en la mayoría de los casos.

Comparación de Resultados

Muestra en una tabla o gráfica los resultados obtenidos por cada clasificador según las métricas mencionadas.

Comparación de Resultados

Dataset	Modelo	Precisión	Recall	MSE	MAD
Con menos columnas	Logistic Regression	93.5%	93.5%	0.05081	0.1189
Limpio	Logistic Regression	93.0%	93.0%	0.05401	0.117
Original	Logistic Regression	92.9%	93%	0.05607	0.1224
Con menos columnas	Multilayer Perceptron	95.6%	95.6%	0.03861	0.0634
Limpio	Multilayer Perceptron	95.7%	95.7%	0.03686	0.0559
Original	Multilayer Perceptron	95.3%	95.4%	0.03656	0.0617
Con menos columnas	RandomForest	97.8%	97.8%	0.01631	0.0322
Limpio	RandomForest	99.2%	99.2%	0.00661	0.0133
Original	RandomForest	99.20%	99.10%	0.00671	0.0134

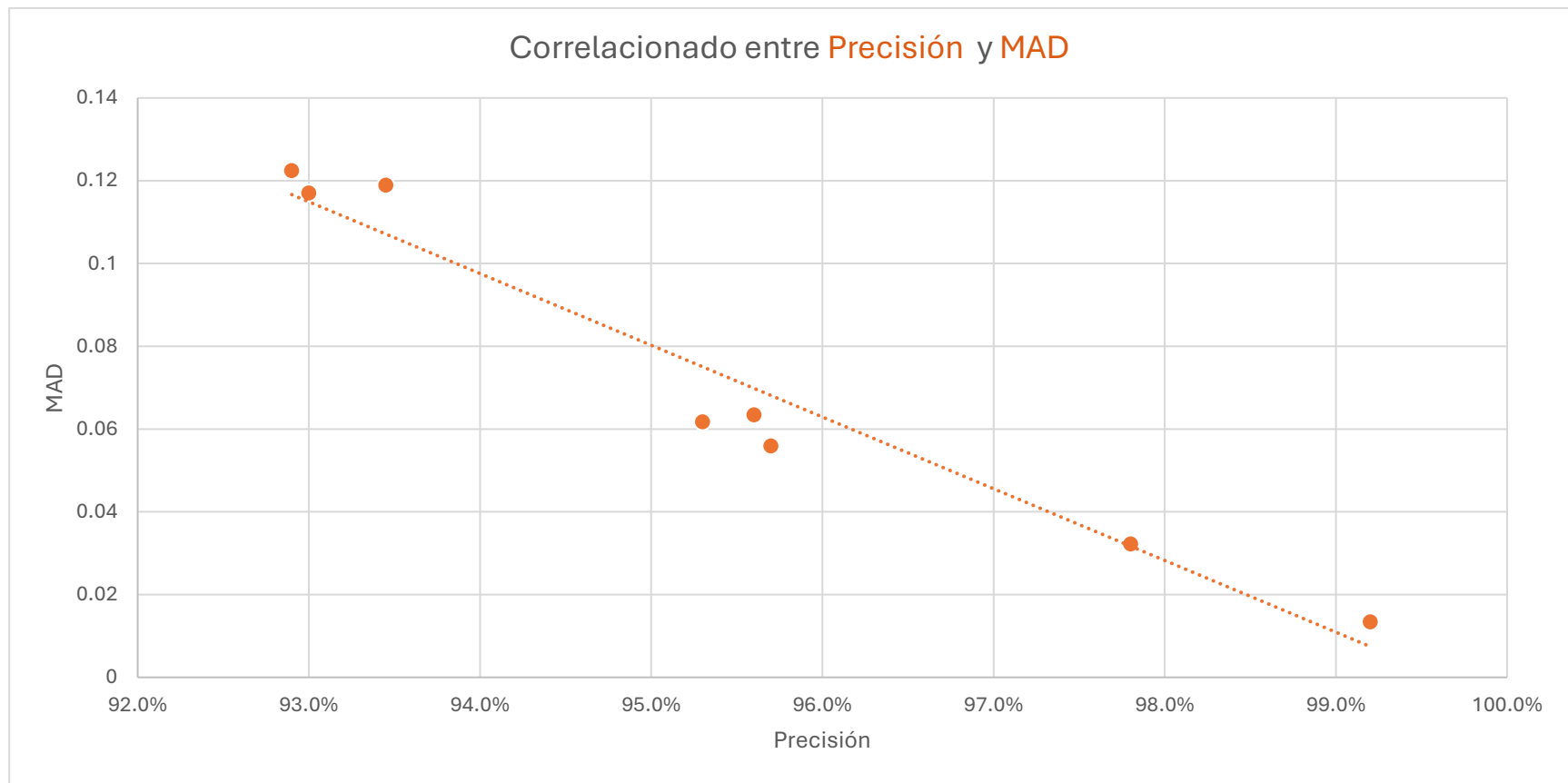
Tabla 1 resultados con las métricas de las tres pruebas realizadas a cada modelo

A partir de los resultados ver tabla 1 podemos destacar que el modelo con mayor precisión es de RandomForest con una precisión arriba del 97%, el modelo menos preciso es el de Logistic Regression con un 93% de precisión, pero el mismo obtuvo un comportamiento contrario al de RandomForest, ya que a medida que se fueron excluyendo datos y columnas la precisión de Logistic Regression aumento, por lo cual es un modelo menos sensible al cambio de características, en cambio la de RandomForest, disminuyo, lo que lo hace más sensible a los cambios de características.

En general los tres modelos presentan números bastante adecuados para hacer pruebas con nuevos datos. Con un MSE y un MAD bajos por lo cual indica un ajuste más preciso y menor error en las clasificaciones

Estos resultados también nos permiten identificar que los datos fueron preprocesados adecuadamente antes del análisis, para este tipo de modelos de clasificación ya que con los resultados de las modificaciones al dataset podemos decir que no era necesario aplicar transformaciones significativas, ya que los valores estaban limpios, sin anomalías importantes.

Con el resultado de la tabla podemos ver la relación en estos modelos de la precisión y el MAD donde se destaca que entre mejor la precisión más pequeña el valor del MAD ver grafica 1

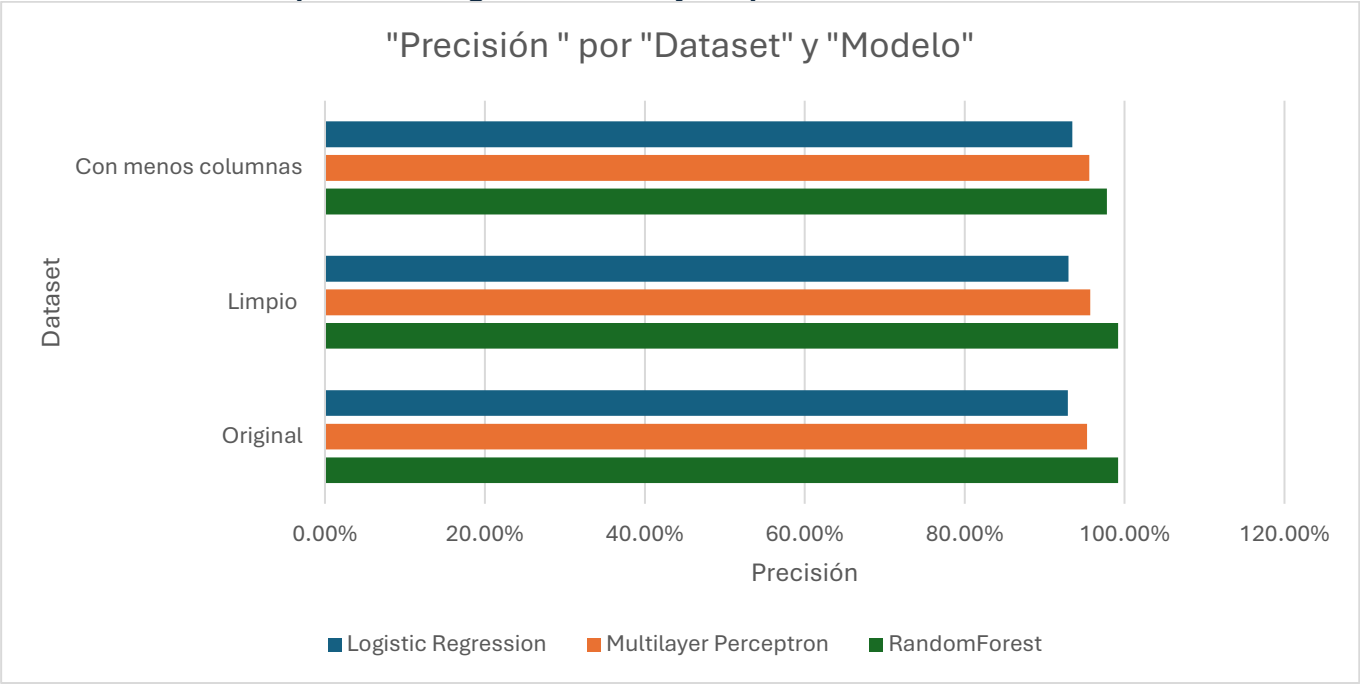


Grafica 1 Relación entre Precisión y MAD

Al igual los resultados nos permiten comparar los modelos según el dataset utilizado, destacando el RandomForest con el promedio de precisión más alto ver tabla 2 y grafica 2.

Promedio de Precisión	Modelo		
Dataset	Logistic Regression	Multilayer Perceptron	RandomForest
Con menos columnas	93.45%	95.60%	97.80%
Limpio	93.00%	95.70%	99.20%
Original	92.90%	95.30%	99.20%
Total general	93.12%	95.53%	98.73%

Tabla 2 Promedio de precisión según el modelo y el tipo de dataset utilizado

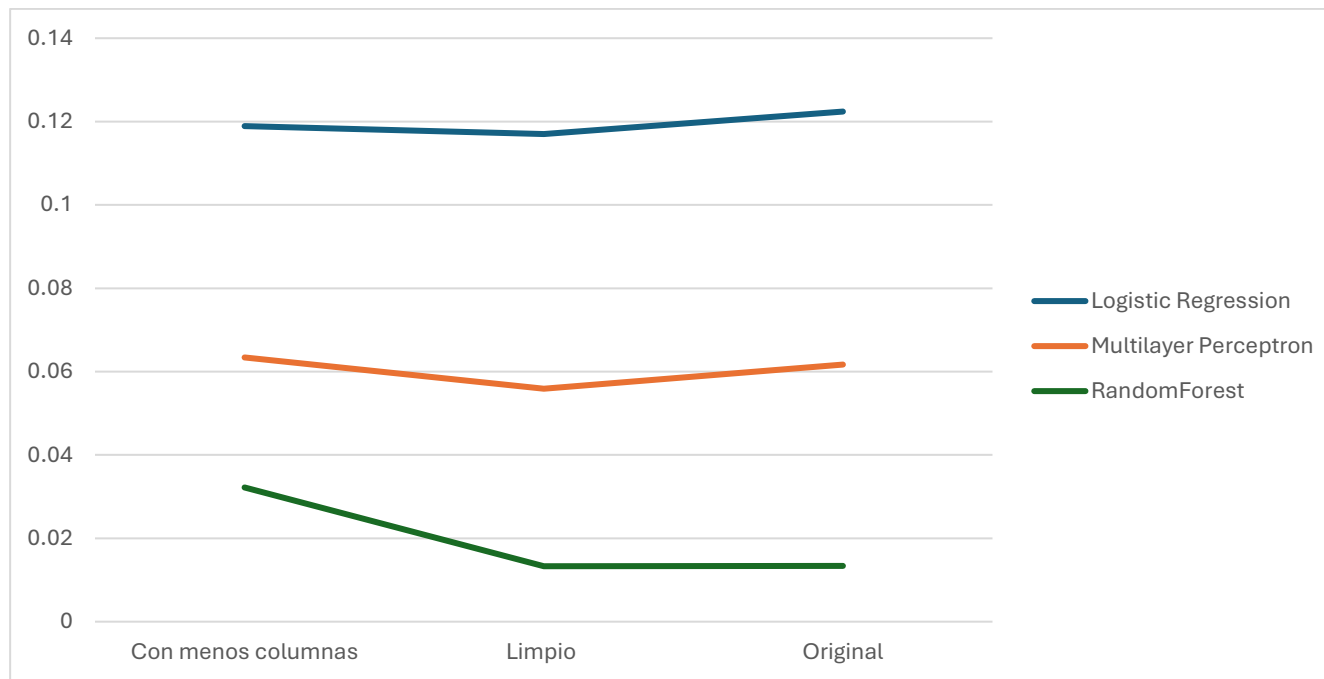


Grafica 2 "Precisión " por "Dataset" y "Modelo"

Dichos resultados nos permiten comparar los modelos según el dataset utilizado, lo susceptible a los cambios en las características de este dataset, dando el RandomForest el aumento significativo en el MAD al eliminar columnas en el dataset más alto ver tabla 3 y grafica 3.

Promedio de MAD	Modelo		
Dataset	Logistic Regression	Multilayer Perceptron	RandomForest
Con menos columnas	0.1189	0.0634	0.0322
Limpio	0.117	0.0559	0.0133
Original	0.1224	0.0617	0.0134

Tabla 3 Promedio de MAD por modelo y tipo de data set



Grafica 3 Tabla 3 Promedio de MAD por modelo y tipo de data

Prueba de modelo RandomForest en Python

Se creo el modelo RandomForest en Python utilizando las librerías sklearn ver ilustración 21

```
[83] # importar librerías necesarias
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, mean_absolute_error, mean_squared_error, confusion_matrix
from sklearn import *
import seaborn as sns
import matplotlib.pyplot as plt
```

```
[84] # Cargar los datos desde un archivo CSV
data = pd.read_csv('/content/MalwareArtifacts.csv')
# Selecciona las columnas deseadas
x_atributos = data[['AddressOfEntryPoint', 'MajorLinkerVersion', 'MajorImageVersion', 'MajorOperatingSystemVersion', 'DllCharacteristics', 'SizeOfStackReserve', 'NumberOfSections', 'ResourceSize' ]]
y_categorias = data['legitimate']
# Dividir los datos en conjuntos de entrenamiento y prueba
x_train, x_test, y_train, y_test = train_test_split(x_atributos, y_categorias, test_size=0.2, stratify=y_categorias, random_state=1) # estratificación según las etiquetas
```

```
#una grafica de pastel de y_categorias
# Contar la frecuencia de cada categoría
counts = y_categorias.value_counts()
# Crear la gráfica de pastel
plt.figure(figsize=(2, 2))
plt.pie(counts, labels=counts.index, autopct='%1.1f%%', startangle=90)
plt.axis('equal') # Asegura que la gráfica sea un círculo
plt.title('Distribución de Categorías')
plt.show()
```

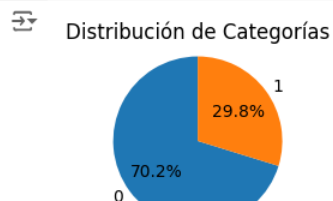


Ilustración 21 Primeros pasos para crear el modelo, importar librerías, dividir los datos en atributos, clases, entrenamiento y prueba. Se procede a crear el modelo, entrenarlo y evaluar su precisión. Ver ilustración 22

```
rfc = ensemble.RandomForestClassifier(n_estimators=100) #Crear el clasificador Random Forest
rfc.fit(x_train, y_train) #Entrenar el modelo
accuracy = rfc.score(x_test, y_test) #Evaluar la precisión del modelo, utiliza x_test y compara los resultados con y_test
print("Random Forest Classifier accuracy: " + str(accuracy*100) )
```

Random Forest Classifier accuracy: 99.2324202408236

Ilustración 22 código para crear el modelo, entrenarlo y evaluar su precisión

Se realizan las validaciones de las métricas como la precisión, MAD MSE. Ver ilustración 23

```

# Realizar predicciones en el conjunto de prueba
y_pred = rfc.predict(x_test)

# Calcular el MAD
mad = mean_absolute_error(y_test, y_pred)
print("Mean Absolute Deviation (MAD): ", mad)

# Calcular el MSE
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error (MSE): ", mse)

# Imprimir el reporte de clasificación
print(classification_report(y_test, y_pred))

# Calcular y mostrar la matriz de confusión
conf_matrix = confusion_matrix(y_test, y_pred)
#print("Matriz de Confusión:")
#print(conf_matrix)
# Visualizar la matriz de confusión
plt.figure(figsize=(10,7))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicción')
plt.ylabel('Real')
plt.title('Matriz de Confusión')
plt.show()

```

Mean Absolute Deviation (MAD): 0.007675797591763978

Mean Squared Error (MSE): 0.007675797591763978

	precision	recall	f1-score	support
0	1.00	0.99	0.99	19305
1	0.99	0.99	0.99	8184
accuracy			0.99	27489
macro avg	0.99	0.99	0.99	27489
weighted avg	0.99	0.99	0.99	27489

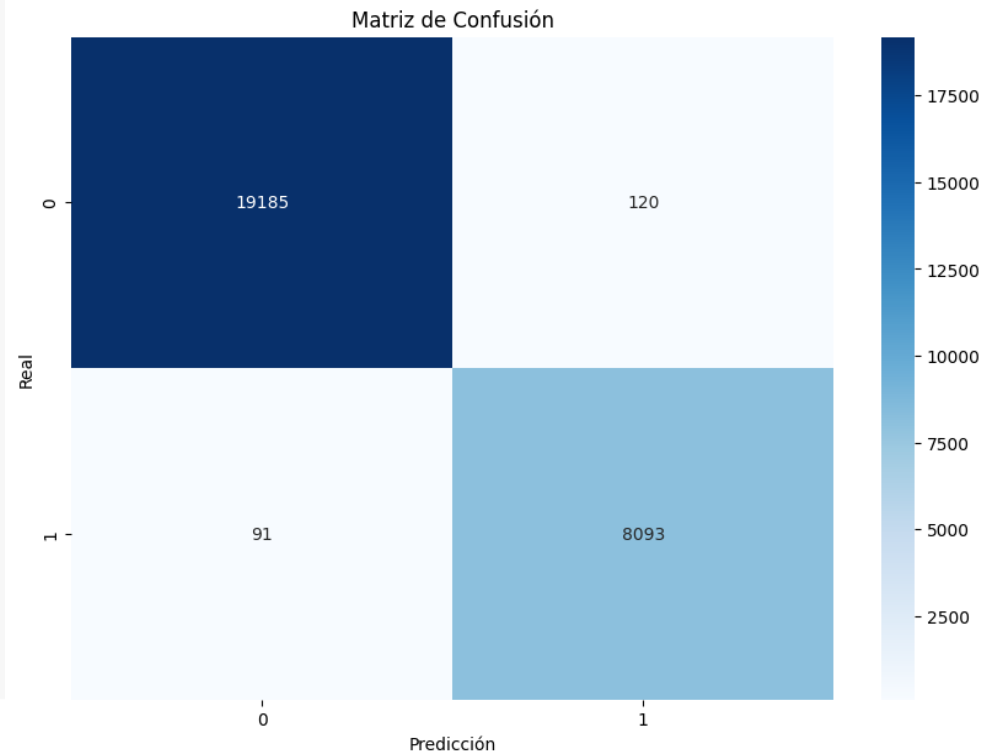


Ilustración 23 resultado de las métricas del modelo en Python.

Pruebas de modelo con nuevos datos

El modelo en Python arroja como resultado similar a lo visto en Weka con un 99% de precisión. Por lo cual se procede hacer pruebas con datos nuevos que no están en los datos de entrenamiento ni prueba, para ello se utiliza una lista de nuevos programas, los cuales se le extrae la información y se crea un nuevo archivo CSV con un código Python, ver ilustración 24

```
import os
import pefile
import glob
import csv

# Abrir el archivo CSV para escritura
with open('datosarchivosPE.csv', 'w', newline='') as csvfile:
    # Crear un escritor CSV
    csvwriter = csv.writer(csvfile)









    # Escribir la fila de encabezado
    csvwriter.writerow([
        "AddressOfEntryPoint",
        "MajorLinkerVersion",
        "MajorImageVersion",
        "MajorOperatingSystemVersion",
        "DllCharacteristics",
        "SizeOfStackReserve",
        "NumberOfSections",
        "ResourceSize"
    ])

# Encontrar archivos .exe
files = glob.glob(r'C:\Users\JOSE TOBAR\Documents\Mestria en analitica de datos\ejemplos_archivos\*.exe')

for file in files:
    try:
        suspect_pe = pefile.PE(file)

        # Escribir los datos de cada archivo PE
        csvwriter.writerow([
            suspect_pe.OPTIONAL_HEADER.AddressOfEntryPoint,
            suspect_pe.OPTIONAL_HEADER.MajorLinkerVersion,
            suspect_pe.OPTIONAL_HEADER.MajorImageVersion,
            suspect_pe.OPTIONAL_HEADER.MajorOperatingSystemVersion,
            suspect_pe.OPTIONAL_HEADER.DllCharacteristics,
            suspect_pe.OPTIONAL_HEADER.SizeOfStackReserve,
            suspect_pe.FILE_HEADER.NumberOfSections,
            suspect_pe.OPTIONAL_HEADER.DATA_DIRECTORY[2].Size
        ])
    except Exception as e:
        print(f"Error procesando el archivo (file): {e}")

print("listo")
```

	Dev-Cpp 5.6.2 TDM-GCC x64 4.8.1 Setup	04/12/2014 11:35 a. m.	Aplicación	45,886 KB
	everest	10/12/2009 12:43 a. m.	Aplicación	2,382 KB
	KMSPico_setup	11/10/2013 4:55 p. m.	Aplicación	2,735 KB
	PSCS6	12/09/2016 9:14 p. m.	Aplicación	6,281 KB
	Setup.X86.es-ES_O365ProPlusRetail_0b1a...	08/26/2016 5:32 p. m.	Aplicación	3,696 KB
	setup_Project64_1.6	01/11/2012 4:55 p. m.	Aplicación	2,033 KB
	USB Show	05/01/2009 10:56 p. m.	Aplicación	114 KB
	zinjai-w32-20140620	11/16/2014 10:24 a. m.	Aplicación	57,053 KB

A	B	C	D	E	F	G	H
AddressOfEr	MajorLinker\	MajorImage\	MajorOperat	DllCharacter	SizeOfStackF	NumberOfSe	ResourceSize
12851	6	0	4	0	1048576	5	18608
12668320	2	0	4	0	16777216	3	320464
42488	2	6	1	32768	1048576	8	23136
2376651	9	0	5	32768	1048576	4	1379676
1517857	14	0	5	33088	1048576	6	569780
103660	6	0	4	0	1048576	4	41696
7852	6	1	4	0	1048576	2	12696
12491	6	6	4	32768	1048576	5	18792
10407	9	6	6	33088	262144	4	952
17415	7	0	4	0	327680	4	992

Ilustración 24 Código para generar nuevo archivo con información de aplicaciones ejecutables para Windows.

AL hacer la prueba con nuevos datos el modelo pudo detectar cuales eran legítimos y cuales podían ser archivos maliciosos. Ver ilustración 25

```
# Cargar los datos nuevos desde un archivo CSV
datos_nuevos = pd.read_csv('/content/datosdearchivosPE.csv')
# Selecciona las columnas deseadas
x_nuevos = datos_nuevos[['AddressOfEntryPoint', 'MajorLinkerVersion', 'MajorImageVersion', 'MajorOperatingSystemVersion', 'DllCharacteristics', 'SizeOfStackReserve', 'NumberOf...]]
y_nuevos = datos_nuevos['legitimate']
# Realizar predicciones en los nuevos datos
y_pred_nuevos = rfc.predict(x_nuevos)

# Imprimir las predicciones
print("Predicciones para los nuevos datos:")
for i, prediccion in enumerate(y_pred_nuevos):
    if prediccion == 1:
        print(f"El archivo {i+1} es malicioso.")
    else:
        print(f"El archivo {i+1} es legitimo.")
```

Predicciones para los nuevos datos:
El archivo 1 es legitimo.
El archivo 2 es legitimo.
El archivo 3 es legitimo.
El archivo 4 es legitimo.
El archivo 5 es legitimo.
El archivo 6 es legitimo.
El archivo 7 es legitimo.
El archivo 8 es legitimo.
El archivo 9 es malicioso.
El archivo 10 es malicioso.

Ilustración 25 Resultado de pruebas del modelo con datos nuevos.

Conclusiones

Al finalizar el trabajo podemos comparar tres modelos diferentes con una buena precisión, para clasificar el dataset de posibles archivos PE ilegítimos, a la vez que se puede diferenciar las diferentes características de los modelos en cuanto a la variación de resultados si se cambian los datos.

Igualmente podemos destacar que el modelo Random Forest fue el que genero mejor resultado en todas sus métricas, y un resultado igual al implementarlo en Python y probar nuevos datos.

Este tipo de modelados debe ser igualmente analizado más afondo con más datos de pruebas e ir corrigiendo posibles errores, o sobreajuste que pueda tener.

Referencias

[GitHub - PacktPublishing/Hands-On-Artificial-Intelligence-for-Cybersecurity: Hands-On Artificial Intelligence for Cybersecurity, publised by Packt](#)

Implementación en Python

<https://colab.research.google.com/drive/1O130kZcflcw5sCLtGPKaBQhirsPv9O3B?usp=sharing>

[Random Forest - Analytics Lane](#)