An Android Malware Multi-class Classification Explained Through Genetic Programming

Gianni D'Angelo, Francesco Palmieri, and Antonio Robustelli(✉)

Department of Computer Science, University of Salerno, Fisciano, Italy
{giadangelo,fpalmieri,arobustelli}@unisa.it

Abstract. The interest in applying Artificial Intelligence algorithms within security contexts is rapidly growing, particularly for the tasks related to malware detection and classification. Over the last decade, numerous Machine Learning (ML) and Deep Learning (DL)-based techniques have been proposed to address the growth of malicious applications, focusing on utilizing features derived from dynamic malware analysis. However, these approaches are often considered black boxes due to their limited ability to explain the results they produce. On the contrary, this study seeks to develop a new model for identifying malware families in an interpretable manner. The methodology employs Genetic Programming to construct a multi-class classifier characterized by a mathematical formula expressing the relationship between dynamic features and the considered malware families. Experimental results, based on Android applications from Unisa Malware Dataset (UMD), showcase the effectiveness of our approach in achieving comparable average scores to the most famous Machine Learning techniques.

Keywords: Android Malware · Explainable Classification · Genetic Programming · Artificial Intelligence · API-Calls

1 Introduction

Due to their popularity, Android OS-based devices have attracted billions of end-users around the World, becoming one of the main targets for malware threats. Indeed, as highlighted by famous cybersecurity companies, unprotected devices (e.g. smartphones, tablets, smart TVs, and wearable devices) represent one of the principal targets on which to perform several malicious activities, such as stealing sensitive information and launching cyberattacks. For instance, according to the last McAfee Mobile Threat Reports [25,26], cyber-criminals increased the effectiveness of their activities with the support of a wide variety of methods, such as back-doors and fake cryptocurrencies. Consequently, the number of malicious applications has drastically increased, reaching 40 million in 2020 and growing by at least 4 million yearly.

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2024
O. Gervasi et al. (Eds.): ICCSA 2024 Workshops, LNCS 14816, pp. 53–70, 2024.
https://doi.org/10.1007/978-3-031-65223-3_4

Therefore, on the basis of these outcomes [30], the interest in Artificial Intelligence algorithms has rapidly grown, especially for detection and classification purposes. As is known, in the last decade, many Machine Learning (ML) and Deep Learning (DL)-based techniques have been proposed to counteract the exponential growth of malicious applications by leveraging features coming from both static and dynamic analysis [23,24,32]. In detail, static-based approaches can extract relevant features and signatures without executing applications but, instead, by performing several reverse engineering operations [29,34].

However, such static approaches can be strongly affected by obfuscation tools. Also, they become ineffective against most existing malware capable of evading pattern-matching detection mechanisms, namely metamorphic and polymorphic ones [18]. For this reason, dynamic analysis-based solutions are widely employed to analyze the behavior of an application during its execution and collect different features, such as system parameters and API-Calls [10]. Moreover, in order to capture more significant and relevant insights, dynamic features have been leveraged to face malware detection as corresponding image classification tasks [6,28]. For instance, D'Angelo et al. [4] proposed a sparse matrix representation of API-Calls sequences. More precisely, they demonstrated the effectiveness of the proposed API-Images by obtaining an average accuracy of 95% in performing a binary classification task.

Nevertheless, similar to other mainstream AI methodologies, Deep Learning (DL)-based approaches tend to lack transparency because they do not reveal the underlying processes behind the provided outputs. For this reason, the issue of interpretability has become increasingly significant, giving rise to two emerging research areas: Explainable Artificial Intelligence (XAI) [8] and Interpretable Machine Learning (IML) [28]. In this context, Genetic Programming (GP) stands out as a promising solution for XAI, offering the unique ability to 'explain' outputs by providing a mathematical formula. As a result, GP has proven effective in numerous application domains, with a particular focus on health care [2,7]. Of course, GP has also been employed to face several malware classification tasks [17], but a multi-class model has yet to be proposed in the literature.

In this work, intending to improve the explainability of malware classification, we propose a GP-based model capable of spotting different malware families using dynamic features. To this end, we employ Genetic Programming to provide a mathematical formula expressing the relationship between the previously mentioned API-Images and predicted families. More specifically, we combine the derived formula with the threshold concept in order to gain a powerful multi-class classifier.

The main contributions of this work can be summarized as follows:

1. A malware multi-class classifier, which combines GP with the threshold concept, is proposed and compared with the most famous ML-based techniques.
2. A mathematical formula describing the relationship between API-Images and predicted families is provided to improve the explainability behind the classification process.

The remainder of the paper is organized as follows. Section 2 will present the related works on malware classification. Section 3 will report the background concepts related to the proposed approach, namely the API-Images and the symbolic regression process. Then, Sect. 4 will describe the proposed GP-based classifier by including the threshold concept. Finally, Sect. 5 will report the experimental results, while Sect. 6 will show the conclusions and future work.

2 Related Works

Nowadays, many detection frameworks and methodologies based on static and dynamic analysis have been proposed [23,24,32] to counter the continuous diffusion of malware. In detail, static-based approaches can extract useful features (e.g. signatures) without executing the analyzed applications. To this end, such approaches disassemble the applications by performing several reverse engineering operations. For instance, Zhang et al. [34] proposed a semantic-based approach using dependency graphs created from the application code. Instead, Onwuzurike et al. [29] presented MaMaDROID, a malware detection system that finds the sequences of methods potentially invoked and maps them to their corresponding packages and classes.

However, static approaches can be strongly affected by obfuscation tools. Also, they become ineffective against most existing malware capable of evading pattern-matching detection mechanisms, namely metamorphic and polymorphic ones [18]. Therefore, in order to overcome these issues, several dynamic analysis-based solutions have been proposed to analyze the behavior of an application and consider several related aspects, such as parameter monitoring and API-Calls tracing [10,13]. For instance, Kolosnjaji et al. [19] leveraged the capability of Deep Neural Networks (DNNs) to analyze the sequence of system calls extracted during the app's execution. To accomplish this, they combined convolutional and recurrent layers by obtaining an average accuracy of 89.0% on 10 Android malware families. Instead, Ficco [12] analyzed the effectiveness of several Machine Learning (ML)-based classifiers in spotting benign and malicious applications using Markov chains. To this end, they arranged API-Call sequences as a graph summarizing the execution flow. The experimental results, obtained by collecting 22K benign and 24K malware applications, have shown an average accuracy of 89%. Next, D'Angelo et al. [3,5] have improved this Markov chains-based approach by employing associative rules (i.e. pairs of API-Calls) as corresponding nodes in the related graphs. The achieved results, derived by considering a federated scenario with non-Independent and Identically Distributed (non-IID) data, have proven the effectiveness of the proposed approach in classifying several malware families with an average accuracy of 99%.

Moreover, in order to capture more significant and relevant insights, dynamic features have been leveraged to face malware detection as corresponding image classification tasks [6,28]. For instance, Abderrahmane et al. [1] proposed a CNN to perform the malware analysis by considering a matrix representation of system calls. The obtained results, with the aid of an unbalanced dataset, have shown

an average accuracy of 93.3%. Next, D'Angelo et al. [4] have improved this representation technique by considering the API-Calls sequences. More precisely, they demonstrated the effectiveness of their approach, called API-Images, by obtaining an average accuracy of 95% in performing a binary classification task through Sparse Autoencoders (SAEs).

Nevertheless, despite the satisfactory accuracy demonstrated in these studies, the proposed methodologies often lack the transparency needed to comprehend the processes generating the provided outputs. For this reason, recent advances in AI have shifted their focus towards models falling under the Explainable Artificial Intelligence (XAI) category. A promising avenue in implementing XAI is Genetic Programming (GP), known for its ability to provide a mathematical formula explaining the rationale behind a given output. Consequently, GP has proven effective in several application domains, with a particular emphasis on healthcare [2,7]. Of course, GP has also been employed to face several malware classification tasks [17], but a multi-class model has yet to be proposed in the literature. In detail, Le et al. [22] employed GP as a pure binary malware classifier. The experimental results, derived from 800 malicious and 1600 benign applications, have shown an average accuracy of 93%. Instead, Harahsheh et al. [16] proposed a GP-based feature section technique. To this end, they analyzed the effectiveness of this approach by considering different famous classifiers and other feature selection techniques. The best experimental results, achieved by the Random Forest algorithm, have shown an average accuracy of 97%.

Therefore, with the aim to enhance the explainability of malware classification, this study combines the effectiveness of Genetic Programming with a multi-class threshold concept to establish an explainable correlation between API-Images and several Android malware families.

3 Background

This Section briefly describes the preliminary concepts employed for our classification purposes. For this reason, we recall the definition of API-Images and the related workflow. Then, we report some information about the Symbolic Regression (SR), which represents the core activity performed by GP.

3.1 API-Images

API-Images are a set of sparse matrices proposed by D'Angelo et al. [4], whose peculiarity is to represent the dynamic behavior of analyzed applications. In detail, given a sequence of API-Calls, an API-Image can be built through the following two main steps:

1. Identification of each API-Call with a unique ID.
2. Matrix creation through consecutive pairs of API-Calls.

The first step can be accomplished using a dictionary in which a numeric ID is assigned to each API-Call. Then, from the traced API-Calls sequence, every

two consecutive calls can be used as coordinates to build the related API-Image matrix. Hence, each pair of API-Calls is employed to draw a fixed point in the API-Image. For instance, let a_1, a_2, and a_3 be three APIs where a_2 is called after a_1 and a_3 is called after a_2. On the basis of the described process, we can consider only two pairs of coordinates, $P1 = (a_1, a_2)$ and $P2 = (a_2, a_3)$, respectively. Also, since an application may invoke different times the same API-Calls sequence, some pairs can occur more than once. This situation can be managed through different colour scales (e.g. RGB or Gray-scale) in which each colour represents a frequency. Figure 1 summarizes the workflow to generate an API-Image.

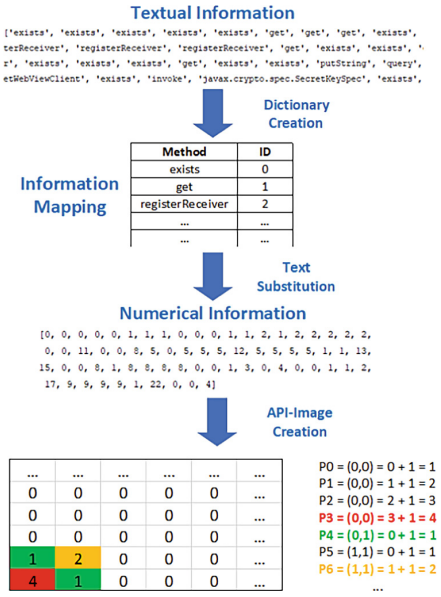

Fig. 1. The workflow to obtain an API-Image [15].

3.2 Symbolic Regression in Genetic Programming

Symbolic Regression (SR) is a well-established method for uncovering the optimal mathematical expression that best fits a multivariate dataset. The core rationale behind adopting SR lies in its ability to produce an interpretable model, which diverges from classic AI approaches operating as black boxes. In the last decades, SR has witnessed increased utilization through the Evolutionary Algorithms (EAs) [9,11]. These algorithms, inspired by Darwinian Theory, excel in generating a set of candidate solutions. Each solution represents a population, with its members also known as individuals. Consequently, the primary objective of an EA is to iteratively navigate through populations to identify the best individual or the optimal solution.

The concept of applying Symbolic Regression (SR) through Evolutionary Algorithms (EAs) was introduced by John Koza, who employed a Genetic Algorithm (GA) [27] in a methodology known as Genetic Programming (GP) [21]. Similar to EAs, GP involves multiple individuals forming a population. In detail, each individual corresponds to a mathematical function represented by a tree structure where leaves represent operands, while other nodes represent mathematical operations. The effectiveness of each individual, computed across all dataset entries, undergoes evaluation through a fitness function that measures the error between the expected outcome and the predicted one.

4 The Employed GP-Based Classifier

As previously mentioned, this work aims to provide an explainable malware classification. To this end, we employ Genetic Programming to derive a mathematical formula describing the relationship between API-Images and the considered malware families. Indeed, differently from other mainstream AI methodologies, Genetic Programming provides an explainable, transparent, and intuitive solution, making the reasoning behind the model's decisions more accessible. Moreover, although Genetic Programming has been employed to face several malware classification tasks [17], no effective multi-class models have been proposed in the literature. Therefore, leveraging the API-Images' capabilities in representing the dynamic behavior of applications, this Section shows how to implement the mathematical formula derived by GP as an effective multi-class classifier. In detail, to better appreciate the provided formulation, we first describe how to build the classifier to face a binary classification task. Next, we extend the provided formulation for a multi-class scenario.

Accordingly, let X^n be the set of one-dimensional API-Images and let C be the set of corresponding output classes, the proposed GP-based classifier aims to provide a function $f^* : X^n \rightarrow C$, which represent the best mathematical formula describing the relationship between input/output pairs of a training dataset S. Notice that $S = (X^n \cup C)$, that is any item of S includes input parameters $(x_i \in X^n)$ and its corresponding class $(c_i \in C)$.

For a binary classification task, the problem to solve can be formally defined as follows:

$$f^* = arg\ min_{f \in F}\ \ \{\mathcal{D}(f(x_i), c_i)\}\ \ \forall (x_i, c_i) \in S \tag{1}$$

where F is the set of all admissible functions, c_i is the ground truth (i.e., the considered malware family), and $\mathcal{D}$ is a distance metric expressing the discrepancy between the predicted and expected outcomes, which can be defined as follows:

$$\mathcal{D} = \frac{1}{|S|} \sum_{(x_i, c_i) \in S} P(x_i, c_i) \tag{2}$$

where

$$P(x_i, c_i) = \begin{cases} 1, f(x_i) = c_i \\ 0, f(x_i) \neq c_i \end{cases} \tag{3}$$

Nevertheless, since the GP-derived mathematical formula yields numerical outputs, certain assumptions are necessary for implementing the classifier effectively. First, the expected output values must be changed into two class values (e.g. 0 and 1). This approach enables GP to discover a formula that fits all numeric inputs to the corresponding output values.

Second, since for an arbitrary input vector x_i the resulting output may not precisely correspond to the expected output class, the concept of a threshold is frequently employed to overcome this issue. More specifically, for a given formula, f, the associated threshold, T_f, is determined as the point of maximum separation among the predicted outcomes of f^* across the entire testing dataset. In detail, a threshold value discretizes the search space into two distinct areas, each representing a specific class. Next, the classification of an input sample is determined by assessing the area in which the provided output falls.

Generally, the selection of the threshold value aims to maximize specific evaluation metrics, such as F-Score or AUC (See Eqs. 10 and 11 below). As expressed in Eq. 4, for a given input vector x_i, if the outcome of the formula $f^*(x_i)$ greater than T_{f^*}, the vector belongs to Class 1; otherwise, it belongs to Class 0.

$$Class = \begin{cases} 1, f^*(x_i) > T_{f^*} \\ 0, f^*(x_i) \leq T_{f^*} \end{cases} \tag{4}$$

Once the definition of the binary classifier is provided, we can derive the related multi-class one by extending the assumptions discussed above with m classes ($m > 2$). To this end, the concept of multiple threshold values is employed. More specifically, let m be the number of classes belonging to C and let $\mathcal{T} = \{T^1_{f^*}, T^2_{f^*}, ..., T^{m-1}_{f^*}\}$ be the set of derived threshold values, then the outcome provided by f^* can be assigned to the related class as follows:

$$Class = \begin{cases} c_{m-1}, f^*(x_i) > T^{m-1}_{f^*} \\ \vdots \\ c_1, T^1_{f^*} < f^*(x_i) \leq T^2_{f^*} \\ c_0, f^*(x_i) \leq T^1_{f^*} \end{cases} \tag{5}$$

5 Experimental Results

This Section aims to demonstrate the effectiveness of the GP-based classifier in recognizing malware categories starting from API-Images. To accomplish this,

we first perform a preliminary study to analyze the capabilities of the most notable ML-based techniques in spotting several malware families. Then, we prove the effectiveness of Genetic Programming by comparing its results with those derived from the preliminary study. Finally, in order to explain the correlation between API-Images and the given category, we provide the mathematical formula describing the employed classifier. All the experiments were conducted by using a MacBook Pro equipped with an Apple M1 CPU and 16 GB of unified memory.

5.1 Dataset and Pre-processing

The dataset used in the following experiments was derived by the Unisa Malware Dataset (UMD) [15]. In detail, we considered 1403 malware applications belonging to 4 notable Android families: Anydown (Class 0), BaseBridge (Class 1), FakeDoc (Class 2), and Opfake (Class 3).

First, we performed an Exploratory Data Analysis (EDA) [20] to accurately study the malware distribution in relationship to the considered classes. As shown in Table 1, the following analysis has counted 326 applications for Anydown, 313 for BaseBridge, 149 for FakeDoc, and 615 for Opfake.

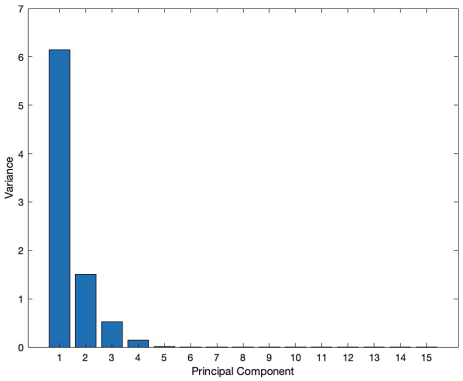

Fig. 2. The distribution of principal components according to their eigenvalues.

Next, by following the workflow shown in Fig. 1, we transformed each sequence of API-Calls into a corresponding API-Image (70×70). For the sake of clarity, since no image classification task is faced, we did not consider a particular colour scale. Instead, due to the unbalanced nature of the employed dataset, we scaled each API-Image for the total number of applications belonging to the related family. Then, we arranged API-Images in a corresponding one-dimensional space of 4900 features.

However, due to the high dimensions of the resulting vectors, we performed an additional dimensionality reduction step by using the Principal Compo-

nent Analysis (PCA) [14]. Hence, we mapped each flattened image in a one-dimensional space of 10 features by also applying a subsequent data standardization with $\mu = 0$ and $\sigma = 1$. In detail, to better choose the output dimensions, we considered the rank of principal components according to their eigenvalues (See Fig. 2). Notice that we conducted the entire data pre-processing phase using several dedicated Python scripts on the whole dataset.

Finally, as shown in Table 1, we have split the resulting dataset with the 70/30 criterion. Therefore, for each family, we employed the 70% for training and the remaining 30% for testing.

Table 1. An overview of the employed dataset splitted with the 70/30 criterion.

	Training	Testing	Total
Anydown	228	98	326
BaseBridge	218	95	313
Fakedoc	104	45	149
Opfake	431	184	615
Total	981	422	1403

5.2 Evaluation Metrics

To appreciate the classification quality of the employed models, we used the following evaluation metrics derived from the confusion matrix for a multi-class classifier: Accuracy (Acc.), Sensitivity (Sens.), Specificity (Spec.), Precision (Prec.), F-Score (F-Mea.), and Area Under the ROC Curve (AUC).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{6}$$

$$Sensitivity = \frac{TP}{TP + FN} \tag{7}$$

$$Specificity = \frac{TN}{TN + FP} \tag{8}$$

$$Precision = \frac{TP}{TP + FP} \tag{9}$$

$$F - Score = \frac{2 * Sens * Prec}{Sens + Prec} \tag{10}$$

$$AUC = \frac{Sens + Spec}{2} \tag{11}$$

For each category, TPs (True Positives) refer to the applications correctly classified, while TNs (True Negatives) refer to the applications correctly identified in another category. Instead, FPs (False Positives) are the applications

mistakenly identified as the considered category, while FNs (False Negatives) are the applications mistakenly identified in another category. Also, to provide a global perspective of the employed models, the average performance values (Avg.) among all the considered malware classes have been derived.

5.3 Preliminary Results

To compare the effectiveness of the proposed classifier, we have first analyzed the capabilities of the most notable ML-based techniques in classifying the considered malware families. To this end, we used some famous algorithms provided by WEKA [33], namely Naive Bayes (NB) classifier, Logistic Regression (LOG), Multi-Layer Perceptron classifier (MLP), Support Vector Machine (SMO), J48 Decision Trees (J48), and Random Forest (RF) algorithm. Tables 2, 3, 4, 5, 6, and 7 report the testing results related to each ML-based technique, respectively. Instead, Table 8 compares the average (Avg.) results.

Table 2. Results achieved by Naive Bayes classifier (NB).

	Acc.	Sens.	Spec.	Prec.	F-Mea.	AUC
Anydown	0.9521	0.9540	0.9516	0.8469	0.8973	0.9528
BaseB	0.9175	0.8861	0.9249	0.7368	0.8046	0.9055
FakeDoc	0.9869	1.0000	0.9852	0.9000	0.9474	0.9926
Opfake	0.9265	0.9783	0.8839	0.8738	0.9231	0.9311
Avg.	0.9458	0.9546	0.9364	0.8394	0.8931	0.9455

Table 3. Results achieved by Logistic Regression (LOG).

	Acc.	Sens.	Spec.	Prec.	F-Mea.	AUC
Anydown	0.9855	0.9894	0.9844	0.9490	0.9688	0.9869
BaseB	0.9714	0.9053	0.9908	0.9663	0.9348	0.9480
FakeDoc	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
Opfake	0.9761	0.9485	1.0000	1.0000	0.9735	0.9742
Avg	0.9833	0.9608	0.9938	0.9788	0.9693	0.9773

As shown in Table 8, the Random Forest (RF) algorithm, the Logistic regression (LOG), and Decision Trees (J48) achieved the best results with a 98% average accuracy. In detail, they outperformed Multi-Layer Perceptron (MLP) by 1%, Naive Bayes (NB) by 4%, and Support Vector Machine (SMO) by 5%, respectively. Furthermore, it is also possible to appreciate the peculiarity of such models to have a discrete number of false positives and false negatives. This, in

Table 4. Results achieved by Multi-Layer Perceptron classifier (MLP).

	Acc.	Sens.	Spec.	Prec.	F-Mea.	AUC
Anydown	0.9733	0.9307	0.9871	0.9592	0.9447	0.9589
BaseB	0.9548	0.9750	0.9500	0.8211	0.8914	0.9625
FakeDoc	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
Opfake	0.9709	0.9388	1.0000	1.0000	0.9684	0.9694
Avg.	0.9748	0.9611	0.9843	0.9451	0.9511	0.9727

Table 5. Results achieved by Support Vector Machine (SMO).

	Acc.	Sens.	Spec.	Prec.	F-Mea.	AUC
Anydown	0.9510	0.9877	0.9414	0.8163	0.8939	0.9645
BaseB	0.9134	1.0000	0.8983	0.6316	0.7742	0.9491
FakeDoc	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
Opfake	0.8765	0.7797	1.0000	1.0000	0.8762	0.8898
Avg.	0.9352	0.9418	0.9599	0.8620	0.8861	0.9509

Table 6. Results achieved by J48 Decision Trees (J48).

	Acc.	Sens.	Spec.	Prec.	F-Mea.	AUC
Anydown	0.9808	0.9588	0.9875	0.9588	0.9588	0.9731
BaseB	0.9737	0.9468	0.9815	0.9368	0.9418	0.9642
FakeDoc	0.9976	1.0000	0.9973	0.9778	0.9888	0.9986
Opfake	0.9855	0.9785	0.9912	0.9891	0.9838	0.9849
Avg.	0.9844	0.9710	0.9894	0.9656	0.9683	0.9802

Table 7. Results achieved by Random Forest algorithm (RF).

	Acc.	Sens.	Spec.	Prec.	F-Mea.	AUC
Anydown	0.9881	0.9697	0.9938	0.9796	0.9746	0.9817
BaseB	0.9834	0.9889	0.9819	0.9368	0.9622	0.9854
FakeDoc	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
Opfake	0.9857	0.9684	1.0000	1.0000	0.9840	0.9842
Avg.	0.9893	0.9818	0.9939	0.9791	0.9802	0.9878

fact, is highlighted by the obtained F-Score values, which range from 88% to 98%.

However, as discussed in Sect. 2, the considered ML-based techniques do not explain why they generate a particular result (i.e., the predicted class) [2]. Consequently, in order to improve the model explainability, we investigated the

Table 8. A summarize of the average results of the most notable ML-based techniques powered by WEKA tool.

	Acc.	Sens.	Spec.	Prec.	F-Mea.	AUC
RF	0.9893	0.9818	0.9939	0.9791	0.9802	0.9878
LOG	0.9833	0.9608	0.9938	0.9788	0.9693	0.9773
J48	0.9844	0.9710	0.9894	0.9656	0.9683	0.9802
MLP	0.9748	0.9611	0.9843	0.9451	0.9511	0.9727
NB	0.9458	0.9546	0.9364	0.8394	0.8931	0.9455
SMO	0.9352	0.9418	0.9599	0.8620	0.8861	0.9509

effectiveness of a GP-based classifier in correlating the considered API-Images with the given output (i.e., the prediction of a class for an arbitrary malware application).

5.4 GP-Based Classifier and Results

To implement the proposed GP-based classifier, we used a MATLAB library for symbolic data mining, namely GPTIPS (Genetic Programming Toolbox for the Identification of Physical Systems) [31]. It is a variant of the GP approach that employs the Multi-Gene-based Genetic Programming (MGGP) algorithm to derive the mathematical formula. In detail, MGGP performs the symbolic scaling regression, whose peculiarity is capturing the behavior of individuals more effectively than the classic SR discussed in Sect. 3.

Therefore, in our experiments, we implemented the GP-classifier by varying different hyper-parameters. For the sake of clarity, only the set-up that provided the best results is shown below:

- pop_size: the dimension of considered population = 100;
- $runs$: the number of algorithm iterations = 100;
- num_gen: the number of generations at each iteration = 35;
- $tournament.size$: the size of tournament selection criteria = 9;
- max_depth: the max depth of solution trees = 11;
- $bias$: the range of bias values $[-10; 10]$;
- $functions$: the considered arithmetic operators $[\times, -, +, /, sqrt, exp]$.

Table 9 reports the results obtained by the GP-based classifier on the whole testing set, while Table 10 compares average results with those derived from the preliminary study (i.e., the ML-based techniques provided by WEKA).

Table 9. Results achieved by GP-based classifier.

	Acc.	Sens.	Spec.	Prec.	F-Mea.	AUC
Anydown	0.9579	0.9765	0.9530	0.8469	0.9071	0.9647
BaseB	0.9281	0.8495	0.9506	0.8316	0.8404	0.9000
FakeDoc	0.9847	0.9333	0.9914	0.9333	0.9333	0.9624
Opfake	0.9579	0.9196	0.9951	0.9946	0.9556	0.9574
Avg.	0.9572	0.9197	0.9725	0.9016	0.9091	0.9461

Table 10. Comparison among the GP classifier and ML-based techniques.

	Acc.	Sens.	Spec.	Prec.	F-Mea.	AUC
RF	0.9893	0.9818	0.9939	0.9791	0.9802	0.9878
LOG	0.9833	0.9608	0.9938	0.9788	0.9693	0.9773
J48	0.9844	0.9710	0.9894	0.9656	0.9683	0.9802
MLP	0.9748	0.9611	0.9843	0.9451	0.9511	0.9727
GP	0.9572	0.9197	0.9725	0.9016	0.9091	0.9461
NB	0.9458	0.9546	0.9364	0.8394	0.8931	0.9455
SMO	0.9352	0.9418	0.9599	0.8620	0.8861	0.9509

As shown in Table 10, all the metrics achieved comparable scores with those derived by ML-based techniques provided by WEKA. In detail, with an average accuracy of 95%, the employed GP-based classifier obtained a 1% more on Naive Bayes and a 2% on Support Vector Machine (SMO). On the contrary, our approach has achieved a 2% less on the Multi-Layer Perceptron (MLP) and a 3% on Decision Trees (J48) and Random Forest (RF) algorithm. Consequently, this confirms the effectiveness of the GP-based classifier in spotting multiple malware families.

5.5 Formula Extraction and Explainability

Despite the confirmed effectiveness of the GP-based classifier in the preceding experiments, notice that our proposal distinguishes itself for its ability to present a mathematical formula expressing the connection between API-Images and the given category. To this end, we derived f^* by running the MGGP algorithm on the whole training dataset.

The outcome is represented by the following equations:

$$f^* = Z_1 + Z_2 + Z_3 \tag{12}$$

$$Z_1 = 0.3431 \cdot x_3 + 0.3431 \cdot x_4 + 0.02946 \cdot x_5 + 0.1568 \cdot x_6 \tag{13}$$

$$Z_2 = -0.07404 \cdot x_9 - x_4 \cdot e^{exp(exp(x_4) - (3 \cdot x_5 - 2 \cdot x_3 + exp(x_4))^4)} \cdot 0.6738 \tag{14}$$

$$Z_3 = \frac{0.07404 \cdot x_1}{2 \cdot x_3 - 3 \cdot x_5} + 0.8997 \tag{15}$$

Next, as shown in Fig. 3 and with reference to Table 4, we considered three threshold values ($T_{f^*} = 0.74$, 1.47, and 2.45) to distinguish the malware families provided by the testing dataset, that are: Anydown - Class 0, BaseBridge - Class 1, FakeDoc - Class 2, and Opfake - Class 3.

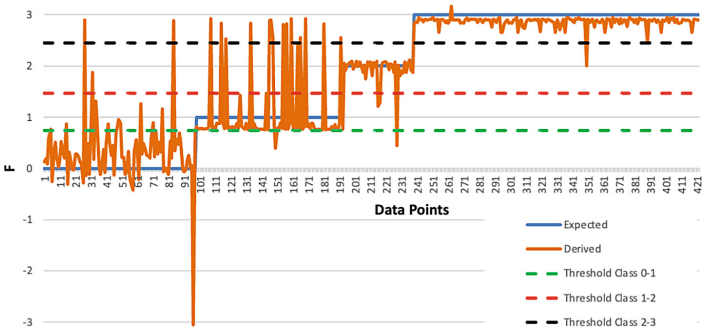

Fig. 3. Predicted vs expected classes. The threshold values are also depicted.

More formally, this means that we adapted the definition provided by Eq. 5 for the considered malware families as follows:

$$Class = \begin{cases} 3, f^*(x_i) > T^3_{f^*} \\ 2, T^2_{f^*} < f^*(x_i) \leq T^3_{f^*} \\ 1, T^1_{f^*} < f^*(x_i) \leq T^2_{f^*} \\ 0, f^*(x_i) \leq T^1_{f^*} \end{cases} \tag{16}$$

with $T^1_{f^*} = 0.74$, $T^2_{f^*} = 1.47$, and $T^3_{f^*} = 2.45$.

As depicted, the derived formula is given by the sum of three functions, namely Z_1, Z_2, and Z_3. Although the interpretation of this formula is not trivial, we highlight some helpful considerations to foster a better comprehension of the correlation between the API-Images and the predicted malware families.

Firstly, the selection of 6 out of 10 features (x_i) demonstrates that the adopted PCA decomposition has provided an effective dimensionality reduction of input images. As previously remarked, this step is fundamental to overcoming bias situations arising from the flattening process of API-Images.

Second, with reference to Fig. 3 and Table 9, it is possible to appreciate how the employed formula has correctly classified most malware applications by obtaining an accuracy score between 92% and 98%.

Finally, regarding the presence of false positives and false negatives, it is also possible to appreciate how the derived formula limits the number of applications incorrectly classified. Indeed, for 3 families out of 4, the derived formula has achieved an F-Score between 90% and 95%. Whereas, only for BaseBridge (Class 1), the number of both false positives (16) and false negatives (14) is a little bit

high. Nevertheless, such values do not adversely affect the overall validity of the provided formula and, thus, of this proposal.

Therefore, from these considerations, we can state that the derived formula can effectively explain how malware applications belong to a specific family. Consequently, its employment could offer more targeted solutions capable of mathematically justifying their results instead of performing only a simple classification task.

6 Conclusions and Future Work

In this paper, we proposed a Genetic Programming (GP)-based model to classify malware families in an explainable way. More precisely, since classic Machine Learning and Deep Learning-based approaches do not provide any information about how they generate the related outputs, we employed Genetic Programming to provide a mathematical formula expressing the relationship between malware applications and the corresponding family. To this end, we first considered a famous feature representation technique, named API-Images, to effectively arrange the dynamic behavior of applications. Then, we have shown the validity of the proposed multi-class classifier by using multiple thresholds to turn the given predictions into corresponding classes. Experimental results, based on 4 Android malware families representative of the Unisa Malware Dataset (UMD), have shown comparable scores with those derived by the most famous Machine Learning techniques. Finally, we derived the mathematical formula expressing the relationship between API-Images and the given outcomes to provide an understandable interpretation of the employed classifier.

Therefore, on the basis of the achieved outcomes, we would like to employ the GP-based approach to spot new malware families, such as those not only related to Android OS. Moreover, we will investigate the effectiveness of GP in classifying malware applications by considering static and Run-time features, which represent two highly considered parameters in malware analysis. All of this could offer several benefits for both end-users and developers, such as more targeted solutions, faster detection and mitigation, and reduction of caused damages.

Acknowledgments. This work was partially supported by project SERICS (PE00000014) under the NRRP MUR program funded by the EU - NGEU.

Author contributions. Gianni D'Angelo: Supervision, Methodology, Reviewing, and Editing.
Francesco Palmieri: Supervision, Methodology, Reviewing, and Editing
Antonio Robustelli: Conceptualization, Investigation, Software, and Writing.

Conflict of interest. The authors declare that they have no conflict of interest.

Informed Consent. Informed consent was obtained from all individual participants included in the study.

Ethical Approval. This article does not contain any studies with human participants or animals performed by any of the authors.

References

1. Abderrahmane, A., Adnane, G., Yacine, C., Khireddine, G.: Android malware detection based on system calls analysis and cnn classification. In: 2019 IEEE Wireless Communications and Networking Conference Workshop (WCNCW), pp. 1–6 (2019). https://doi.org/10.1109/WCNCW.2019.8902627
2. D'Angelo, G., Della-Morte, D., Pastore, D., Donadel, G., De Stefano, A., Palmieri, F.: Identifying patterns in multiple biomarkers to diagnose diabetic foot using an explainable genetic programming-based approach. Future Gener. Comput. Syst. **140**, 138–150 (2023). https://doi.org/10.1016/j.future.2022.10.019. https://www.sciencedirect.com/science/article/pii/S0167739X2200334X
3. D'Angelo, G., Farsimadan, E., Ficco, M., Palmieri, F., Robustelli, A.: Privacy-preserving malware detection in android-based iot devices through federated markov chains. Future Gener. Comput. Syst. **148**, 93–105 (2023). https://doi.org/10.1016/j.future.2023.05.021. https://www.sciencedirect.com/science/article/pii/S0167739X23002054
4. D'Angelo, G., Ficco, M., Palmieri, F.: Malware detection in mobile environments based on autoencoders and api-images. J. Parallel Distrib. Comput. **137**, 26–33 (2020). https://doi.org/10.1016/j.jpdc.2019.11.001. https://www.sciencedirect.com/science/article/pii/S0743731519302436
5. D'Angelo, G., Ficco, M., Palmieri, F.: Association rule-based malware classification using common subsequences of api calls. Appl. Soft Comput. **105**, 107234 (2021). https://doi.org/10.1016/j.asoc.2021.107234. https://www.sciencedirect.com/science/article/pii/S1568494621001575
6. D'Angelo, G., Palmieri, F., Robustelli, A.: Effectiveness of video-classification in android malware detection through API-streams and CNN-LSTM autoencoders. In: You, I., Kim, H., Youn, T.-Y., Palmieri, F., Kotenko, I. (eds.) MobiSec 2021. CCIS, vol. 1544, pp. 171–194. Springer, Singapore (2022). https://doi.org/10.1007/978-981-16-9576-6_13
7. D'Angelo, G., Scoppettuolo, M.N., Cammarota, A.L., Rosati, A., Palmieri, F.: A genetic programming-based approach for classifying pancreatic adenocarcinoma: the SICED experience. Soft Comput. **26**(19), 10063–10074 (2022). https://doi.org/10.1007/s00500-022-07383-3
8. Došilović, F.K., Brčić, M., Hlupić, N.: Explainable artificial intelligence: a survey. In: 2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), pp. 0210–0215 (2018). https://doi.org/10.23919/MIPRO.2018.8400040
9. D'Angelo, G., Palmieri, F.: A co-evolutionary genetic algorithm for robust and balanced controller placement in software-defined networks. J. Netw. Comput. Appl. **212**(C) (2023). https://doi.org/10.1016/j.jnca.2023.103583
10. Egele, M., Scholte, T., Kirda, E., Kruegel, C.: A survey on automated dynamic malware-analysis techniques and tools. ACM Comput. Surv. **44**(2) (2008). https://doi.org/10.1145/2089125.2089126
11. Elia, S., et al.: A machine learning evolutionary algorithm-based formula to assess tumor markers and predict lung cancer in cytologically negative pleural effusions. Soft Comput. **24**(10), 7281–7293 (2019). https://doi.org/10.1007/s00500-019-04344-1
12. Ficco, M.: Detecting IoT malware by Markov chain behavioral models. In: 2019 IEEE International Conference on Cloud Engineering (IC2E), pp. 229–234 (2019)

13. Ficco, M.: Malware analysis by combining multiple detectors and observation windows. IEEE Trans. Comput. **71**(6), 1276–1290 (2022). https://doi.org/10.1109/TC.2021.3082002
14. F.R.S., K.P.: Liii. on lines and planes of closest fit to systems of points in space. Lond. Edinburgh Dublin Phil. Maga. J. Sci. **2**(11), 559–572 (1901). https://doi.org/10.1080/14786440109462720
15. Gianni D'Angelo, Francesco Palmieri, A.R., Castiglione, A.: Effective classification of android malware families through dynamic features and neural networks. Connect. Sci. **33**(3), 786–801 (2021). https://doi.org/10.1080/09540091.2021.1889977
16. Harahsheh, H., Shraideh, M., Sharaeh, S.: Performance of malware detection classifier using genetic programming in feature selection. Informatica **45**(4) (2021). https://doi.org/10.31449/inf.v45i4.3819
17. Heba Harahsheh, Mohammad Alshraideh, S.A.S., Al-Sayyed, R.: Improving classification performance for malware detection using genetic programming feature selection techniques. J. Appl. Secur. Res. **18**(3), 627–647 (2023). https://doi.org/10.1080/19361610.2022.2067459
18. Hosmer, C.: Polymorphic and metamorphic malware - black hat briefings. https://www.blackhat.com/presentations/bh-usa-08/Hosmer/BH_US_08_Hosmer_Polymorphic_Malware.pdf
19. Kolosnjaji, B., Zarras, A., Webster, G., Eckert, C.: Deep learning for classification of malware system call sequences. In: Kang, B.H., Bai, Q. (eds.) AI 2016. LNCS (LNAI), vol. 9992, pp. 137–149. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-50127-7_11
20. Komorowski, M., Marshall, D.C., Salciccioli, J.D., Crutain, Y.: Exploratory data analysis. In: Secondary Analysis of Electronic Health Records, pp. 185–203. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-43742-2_15
21. Koza, J.: Genetic programming as a means for programming computers by natural selection. Stat. Comput. **4**(2) (1994). https://doi.org/10.1007/bf00175355
22. Le, T.A., Chu, T.H., Nguyen, Q.U., Nguyen, X.H.: Malware detection using genetic programming. In: the 2014 Seventh IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA), pp. 1–6 (2014). https://doi.org/10.1109/CISDA.2014.7035623
23. Marra, A.L., Martinelli, F., Mercaldo, F., Saracino, A., Sheikhalishahi, M.: D-bridemaid: a distributed framework for collaborative and dynamic analysis of android malware. J. Wirel. Mobile Netw. Ubiq. Comput. Depend. Appl. (JoWUA) **11**(3), 1–28 (2020)
24. Martín, A., Rodríguez-Fernández, V., Camacho, D.: Candyman: classifying android malware families by modelling dynamic traces with Markov chains. Eng. Appl. Artif. Intell. **74**, 121–133 (2018). https://doi.org/10.1016/j.engappai.2018.06.006. https://www.sciencedirect.com/science/article/pii/S0952197618301374
25. McAfee: Mcafee 2022 consumer mobile threat report. https://www.mcafee.com/blogs/mobile-security/mcafee-2022-consumer-mobile-threat-report/. Accessed 24 Jan 2022
26. McAfee: Mcafee mobile threat report 2021. https://www.mcafee.com/content/dam/global/infographics/McAfeeMobileThreatReport2021.pdf. Accessed 24 Jan 2024
27. Mitchell, M.: An Introduction to Genetic Algorithms. The MIT Press, Cambridge (1998). https://doi.org/10.7551/mitpress/3927.001.0001

28. Molnar, C., Casalicchio, G., Bischl, B.: Interpretable machine learning – a brief history, state-of-the-art and challenges. In: Koprinska, I., et al. (eds.) ECML PKDD 2020. CCIS, vol. 1323, pp. 417–431. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-65965-3_28
29. Onwuzurike, L., Mariconti, E., Andriotis, P., Cristofaro, E.D., Ross, G., Stringhini, G.: Mamadroid: detecting android malware by building markov chains of behavioral models (extended version). ACM Trans. Priv. Secur. **22**(2) (2019). https://doi.org/10.1145/3313391
30. Potha, N., Kouliaridis, V., Kambourakis, G.: An extrinsic random-based ensemble approach for android malware detection. Connect. Sci. **33**(4), 1077–1093 (2021). https://doi.org/10.1080/09540091.2020.1853056
31. Searson, D.P.: GPTIPS 2: an open-source software platform for symbolic data mining. In: Gandomi, A.H., Alavi, A.H., Ryan, C. (eds.) Handbook of Genetic Programming Applications, pp. 551–573. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-20883-1_22
32. Taheri, L., Kadir, A.F.A., Lashkari, A.H.: Extensible android malware detection and family classification using network-flows and api-calls. In: 2019 International Carnahan Conference on Security Technology (ICCST), pp. 1–8 (2019). https://doi.org/10.1109/CCST.2019.8888430
33. Witten, I.H., Frank, E., Hall, M.A., Pal, C.J.: Data Mining, Fourth Edition: Practical Machine Learning Tools and Techniques, 4th edn. Morgan Kaufmann Publishers Inc., San Francisco (2016)
34. Zhang, M., Duan, Y., Yin, H., Zhao, Z.: Semantics-aware android malware classification using weighted contextual api dependency graphs. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, CCS 2014, pp. 1105–1116. Association for Computing Machinery, New York (2014). https://doi.org/10.1145/2660267.2660359