computers

MDPI

Article

Achieving High Accuracy in Android Malware Detection through Genetic Programming Symbolic Classifier

Nikola Anđelić *,† **and Sandi Baressi Šegota** †

Department of Automation and Electronics, Faculty of Engineering, University of Rijeka, 51000 Rijeka, Croatia; sandi.segota@riteh.uniri.hr

* Correspondence: nikola.andjelic@riteh.uniri.hr

† These authors contributed equally to this work.

Abstract: The detection of Android malware is of paramount importance for safeguarding users' personal and financial data from theft and misuse. It plays a critical role in ensuring the security and privacy of sensitive information on mobile devices, thereby preventing unauthorized access and potential damage. Moreover, effective malware detection is essential for maintaining device performance and reliability by mitigating the risks posed by malicious software. This paper introduces a novel approach to Android malware detection, leveraging a publicly available dataset in conjunction with a Genetic Programming Symbolic Classifier (GPSC). The primary objective is to generate symbolic expressions (SEs) that can accurately identify malware with high precision. To address the challenge of imbalanced class distribution within the dataset, various oversampling techniques are employed. Optimal hyperparameter configurations for GPSC are determined through a random hyperparameter values search (RHVS) method developed in this research. The GPSC model is trained using a 10-fold cross-validation (10FCV) technique, producing a set of 10 SEs for each dataset variation. Subsequently, the most effective SEs are integrated into a threshold-based voting ensemble (TBVE) system, which is then evaluated on the original dataset. The proposed methodology achieves a maximum accuracy of 0.956, thereby demonstrating its effectiveness for Android malware detection.

Keywords: Android malware detection; genetic programming symbolic classifier; oversampling techniques; random hyperparameters values method

Citation: Anđelić, N.; Baressi Šegota, S. Achieving High Accuracy in Android Malware Detection through Genetic Programming Symbolic Classifier. *Computers* **2024**, *13*, 197. https://doi.org/10.3390/computers13080197

Academic Editor: Paolo Bellavista

Received: 15 July 2024
Revised: 5 August 2024
Accepted: 13 August 2024
Published: 15 August 2024

1. Introduction

Android malware detection (AMD) is crucial because it protects users' sensitive information from being compromised. Android devices often store a wealth of personal data, including financial details, personal communications, and location information. Malware can exploit vulnerabilities to steal these data, leading to identity theft, financial loss, and privacy violations. Effective malware detection helps safeguard users by identifying and mitigating threats before they can cause harm, ensuring the integrity and confidentiality of personal information.

Moreover, robust malware detection enhances the overall security of the Android ecosystem. As Android is the most widely used mobile operating system, it is a prime target for cybercriminals. Widespread malware can degrade user trust and undermine the platform's reputation. By preventing the proliferation of malicious software, malware detection tools help maintain a secure and reliable environment for both users and developers. This encourages continued innovation and adoption of Android devices, contributing to a healthier digital landscape.

In the last decade, various artificial intelligence (AI) algorithms were applied for AMD. In [1], the authors proposed deep convolutional neural network (CNN). Using this method, the accuracy (ACC) achieved was 98%. The authors in [2] used an online deep-learning-based AMD engine, which can automatically detect if the application is malware or not.

Using this approach, the highest ACC achieved was 96.76%. The multi-level anomaly detector for Android Malware (MADAM) was proposed in [3]. The detector was based on the random forest classifier (RFC) that achieved 96.9%. In [4], the robustness of the support vector machines (SVM) model for AMD against adversarial attacks was explored. It demonstrates how secure SVM can be achieved by integrating adversarial training and robust feature selection. The model achieved an ACC of 89%. Adversarial examples that can fool any deep learning model (DNN) in various domains, including AMD, were examined in [4]. The authors discussed various attack and defense strategies. The accuracy of their adversarial trained models reached up to 92%. The paper [5] introduces a deep learning-based approach for AMD using recurrent neural networks (RNN) to analyze API call sequences. The model effectively distinguishes between benign and malicious applications. The reported ACC is 98.2%. The paper [6] presents MAMADroid, a tool that detects Android malware by constructing Markov chains of app behavior based on API calls. The classification was performed using different classification algorithms: RFC, artificial neural network (ANN), k-Nearest Neighbors (kNN), and support vector classifier. The reported ACC was 99%. The Deep4maldroid was introduced in [7], a deep learning framework that uses Linux kernel system call graphs for AMD. It combines CNNs with graph embedding techniques. The model achieves an ACC of 94.6%. The paper [8] explores the structural detection of Android malware using embedded call graphs, comparing the graphs of benign and malicious apps. The approach is effective in identifying structural anomalies. The reported accuracy is 96.5%. The paper [9] presents a machine learning-based classification approach for Android malware, employing techniques like RFC and SVM. The study highlights the importance of feature engineering. The model achieves an ACC of 96.2%. The deep learning framework for malware detection (DL4MD) was proposed in [10]. The DL4MD is a deep learning framework designed for intelligent malware detection in Android applications, utilizing autoencoders and deep neural networks. The framework emphasizes automatic feature learning. The reported accuracy is 95.8%. The study in [11] presents a method for AMD based on bytecode image, transforming the app's bytecode into images and using CNNs for classification. This novel representation aids in effective detection. The ACC reported is 94.9%.

The study presented in [12] employs a structured pipeline for reverse engineering-based malware analysis, utilizing Call Graphs, Control Flow Graphs (CFGs), and Data Flow Graphs (DFGs) in conjunction with Deep Learning (DL) models. The authors introduce the Canonical Executable Graph (CEG) for Portable Executable (PE) files, which integrates syntactical, semantic, and structural information. Additionally, they develop the Genetic Algorithm-based Graph Explainer (GAGE) to enhance the interpretability of Explainable Artificial Intelligence (XAI). Through this innovative approach, they achieve an accuracy (ACC) of 87%, demonstrating the effectiveness of their methodology.

In [13], the authors propose a method for Android malware detection (AMD) known as GA-StackingMD. This approach combines five different base classifiers using the Stacking technique and optimizes the framework's hyperparameters with a Genetic Algorithm. This method addresses challenges related to high feature dimensionality and the limited accuracy of single classification algorithms. Experimental results show that GA-StackingMD achieves accuracy rates of 98.43% and 98.66% on the CIC-AndMal2017 and CICMalDroid2020 datasets, respectively, underscoring its effectiveness and practical feasibility.

The research detailed in [14] underscores the growing importance of robust anti-malware measures in the expanding Android smartphone market. The authors develop a machine learning model designed to detect subtle malicious patterns, emphasizing the significance of coexistence patterns in malware detection. They enhance data representativeness through the use of the SMOTE technique and optimize feature selection with the Extra Trees Classifier (ETC). The proposed methodology achieves 98% accuracy using a voting classifier, representing a significant advancement in adaptable and precise Android malware detection.

In response to the rising cyber-threats associated with increased smartphone usage, the authors of [15] introduce a Time-Aware Machine Learning (TAML) framework for Android malware detection. Utilizing the KronoDroid dataset, which comprises over 77,000 apps, their framework identifies the 'LastModDate' feature as critical. The study demonstrates that time-aware models significantly outperform traditional machine learning models, achieving a 99.98% F1 score in a time-agnostic setting and up to a 99% F1 score in annual evaluations. Moreover, the research highlights that real-device detection surpasses emulator-based detection, underscoring the superior performance of their time-aware classifier. The results of previously described research papers are summarized in Table 1.

Table 1. The summary of research papers.

Reference	Methods	Accuracy (%)
[1]	CNN	98
[2]	DNN	96.76
[3]	RFC	96.9
[4]	SVM	89
[4]	DNN	92
[5]	RNN	98.2
[6]	RFC, ANN, kNN, SVC	99
[7]	CNN	94.6
[8]	embedded call graphs	94.6
[9]	RFC, SVM	96.2
[10]	autoencoders + DNN	95.8
[11]	CNN	94.9
[12]	GAGE	87
[13]	GA-StackingMD	98.43, 98.66
[14]	ETC	98
[15]	TAML	F1-Score: 99.98

The results presented in Table 1, derived from other research studies, indicate that machine learning and deep learning methods can achieve high evaluation metrics for malware detection. However, these approaches have notable drawbacks, including the substantial computational resources required for training and model storage, as well as the inability to express them in mathematical form. This is particularly relevant for large deep learning neural networks, which consist of numerous interconnected neurons.

To address these limitations, this paper proposes the application of a Genetic Programming Symbolic Classifier (GPSC) to generate symbolic expressions (SEs) capable of detecting Android malware with high classification accuracy. The initial dataset used in this research is the publicly available Android Malware Dataset, which is inherently unbalanced. The primary challenge with an unbalanced dataset in machine learning is the bias it introduces towards the majority class, leading to suboptimal performance and inaccurate predictions, especially for the minority class.

Consequently, this paper explores whether the application of oversampling techniques can balance the dataset and, in turn, improve the GPSC's ability to generate SEs with superior classification performance. Given the extensive range of hyperparameters associated with GPSC, a Random Hyperparameter Values Search (RHVS) method will be developed and employed to identify optimal hyperparameter combinations, enabling the GPSC to produce SEs with enhanced classification capabilities.

The GPSC training process will utilize 10-fold cross-validation (10FCV) to generate 10 SEs per training session, as each SE corresponds to one split of the 10FCV. After determining the best SE sets for each balanced dataset variation, these SEs will be integrated into a threshold-based voting ensemble (TBVE). The research will further investigate whether adjusting the threshold value can enhance the classification performance of the TBVE.

Based on the review of other research papers, the disadvantages of other research papers, and the idea and novelty in this paper, the following questions arise:

- Can the Genetic Programming Symbolic Classifier (GPSC) achieve high classification performance in detecting Android malware?
- Do oversampling techniques effectively balance the dataset, thereby enabling GPSC to produce symbolic expressions (SEs) with high classification performance in AMD?
- Is the development and application of the Random Hyperparameter Values Search (RHVS) method justified in optimizing GPSC hyperparameters for achieving high-performance SEs in AMD?
- Does the application of 10-fold cross-validation (10FCV) in training GPSC lead to the generation of robust and highly accurate SEs for AMD?
- Can the combination of SEs using a threshold-based voting ensemble (TBVE) result in even higher classification performance for AMD?

The structure of this paper is organized into the following sections: Materials and Methods, Results, Discussion, and Conclusions. In the Materials and Methods section, the research methodology is detailed, including the dataset description, statistical analysis, oversampling techniques, GPSC and RHVS, evaluation metrics, training–testing procedures, and threshold based-voting ensemble. The Results section presents the optimal combinations of GPSC hyperparameter values obtained using the RHVS method, as well as the classification performance of each of the best SEs. Additionally, this section reports the classification performance of the TBVE ensemble consisting of the best SEs when applied to the original dataset.

The Discussion section provides an in-depth analysis of the proposed research methodology and the results obtained. The Conclusions section addresses the hypotheses outlined in the introduction, drawing on insights from the detailed discussion. It also highlights the advantages and limitations of the proposed research methodology and suggests directions for future research based on these limitations. Lastly, the Appendix provides supplementary information on the GPSC, along with instructions on how to download and utilize the SEs generated in this study.

2. Materials and Methods

In this section, the research methodology, dataset description and statistical analysis, oversampling techniques, genetic programming symbolic classifier, evaluation metrics, training–testing procedure, and different ensemble methods of best mathematical expressions are described.

2.1. Research Methodology

The research methodology is graphically represented in Figure 1.

As illustrated in Figure 1, the research methodology is composed of the following steps:

- The initial dataset contains a target variable with two classes, each with a different number of samples. To address this imbalance, various oversampling techniques were employed to create balanced dataset variations, ensuring equal representation of each class in the target variable.
- Once the balanced dataset variations were established, each was utilized in the GPSC to generate SEs capable of detecting Android malware with high classification performance. The datasets were split into training and testing subsets in a 70:30 ratio. The RHVS method was employed to identify the optimal combination of GPSC hyperparameters for each dataset variation. GPSC was trained using a 10FCV approach, where each of the ten splits generated one SE. In each split, the GPSC was trained

on nine folds and validated on one. Following the completion of 10FCV with the optimal hyperparameters, the resulting 10 SEs were evaluated on the testing dataset. If the classification performance on the testing dataset was consistent with that on the training dataset, the training/testing procedure was deemed successful, resulting in 10 SEs with robust classification performance.

- After identifying the best SEs from each balanced dataset variation, they were combined into a TBVE. The threshold was adjusted to determine if further improvements in classification performance could be achieved.

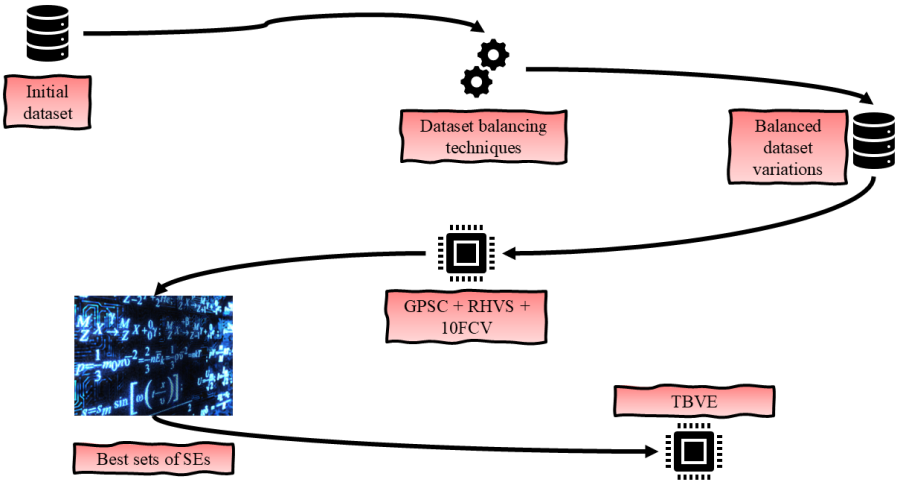

Figure 1. The graphical representation of the research methodology.

2.2. Dataset Description and Statistical Analysis

As stated, in this paper, the publicly avaliable dataset for AMD was used (for reference, see Data Availability at the end of the manuscript). The dataset consists of 327 input variables, 1 target variable, and 4644 samples. The problem with the target variable is that it consists of only strings, i.e., benign and malware values. Using the label encoder, the string values benign and malware were transformed to classes 0 and 1.

Due to an extremely large number of input variables, the initial statistical analysis will be avoided. Instead, the Pearson's correlation analysis will be performed, and a scatter plot will be shown in which input variables that exhibit some correlation (lower than -0.3 and higher than 0.3) with the target variable will be shown. It should be noted that GPSC will automatically label each input variable from the dataset as X variable, with the corresponding index in the range of $i = 0, \ldots, 327$.

Pearson's correlation analysis [16] was performed to examine the relationship between two variables, with correlation values ranging from -1 to 1. A value of -1 indicates a perfect negative correlation, meaning that as one variable increases, the other decreases, and vice versa. Conversely, a value of 1 signifies a perfect positive correlation, where an increase in one variable corresponds to an increase in the other. A correlation value of 0 indicates no relationship between the variables, meaning that changes in one variable have no effect on the other. Figure 2 presents the results of this analysis, showing the correlation between each input variable and the target variable.

From Figure 2, it can be noticed that "android.permision.READ_PHONE_STATE" has the highest positive correlation value (0.76) with the target variable. This high positive correlation suggests that malware applications often request this permission, making it a significant indicator of malicious behavior. The android.permission.RECEIVE_BOOT_COMPLETED and RECEIVE_BOOT_COMPLETED variables have a 0.591 and 0.432 correlation value with the target variable. This strong correlation indicates that malware often requests to start after the device boots, likely to ensure persistence on the device. The INSTALL_SHORTCUT has a 0.4518 correlation value with the target variable. The relatively high correlation value indicates that malware apps might use this permission to place shortcuts on the home screen,

possibly as part of a phishing or adware strategy. The ACCESS_COARSE_LOCATION and ACCESS_FINE_LOCATION have 0.406 and 0.39 correlation values with the target variable. Access to location data is moderately associated with malware, suggesting these apps might track user movements. The RECEIVE_SMS and READ_SMS have a bot 0.307 correlation value with the target variable. The SMS permissions are moderately correlated with malware, indicating that malicious apps might intercept or read messages. The GET_TASKS have a 0.323 correlation value with the target variable. This permission allows an app to see what tasks are running, which can be used by malware to monitor user activity or other applications. The SEND_SMS has a correlation value of 0.306 with the target variable. The ability to send SMS is associated with malware, potentially for sending premium SMS messages or spreading spam.

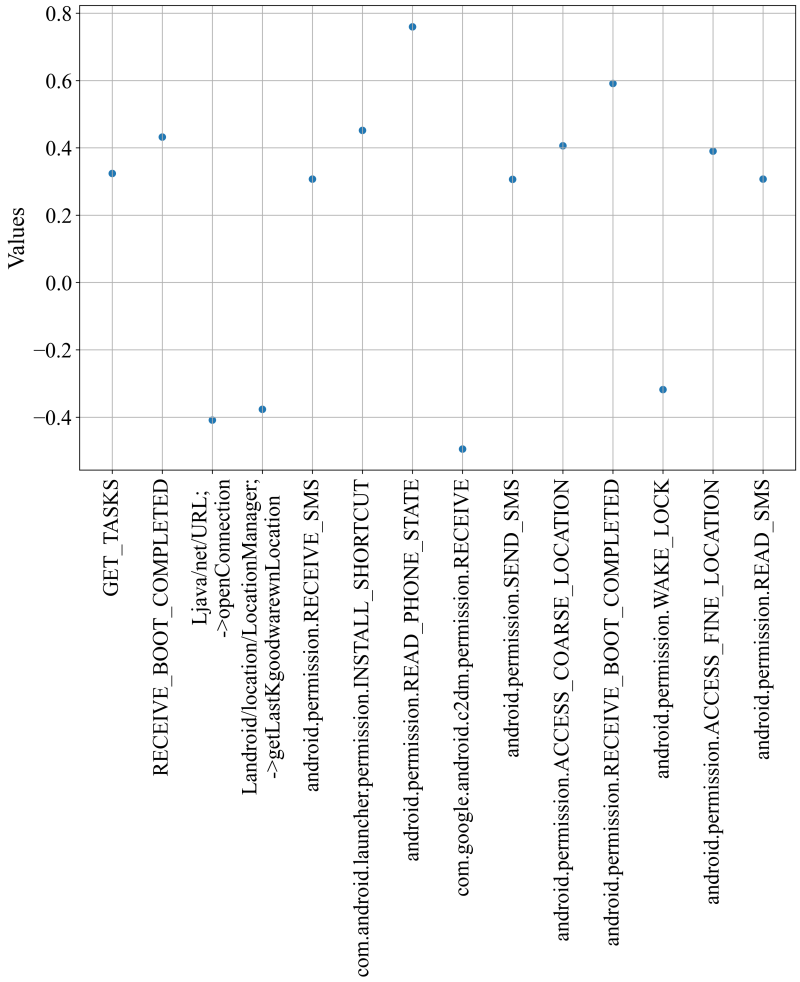

Figure 2. The results of the Pearson's correlation analysis between the input variables and the output (target) variable are displayed. The plot highlights only those input variables that exhibit a correlation value greater than 0.3 or less than −0.3 with the target variable.

The negative correlation indicates that the presence of certain permissions is more common in the benign applications. The com.google.android.c2dm.permission.RECEIVE has a correlation of −0.49 with the target variable. This correlation value suggests that benign apps are more likely to use Google's cloud messaging service, whereas malware might avoid it to evade detection. The Ljava/net/URL;->openConnection variable has a correlation value of −0.408 with the target variable. The benign apps might use standard network connections more frequently, while malware could use alternative methods to avoid detection. The Landroid/location/LocationManager;->getLastKnownLocation variable has a −0.376 correlation value with the target variable. Accessing the last known

location is more common in benign apps, possibly for legitimate location-based services. The WAKE_LOCK has a −0.317 correlation value with the target variable. The benign applications might use this permission to keep the device awake for legitimate purposes, whereas malware might not use it frequently.

The number of samples per class could indicate if the dataset is balanced or imbalanced. In this dataset, there are two classes, i.e., benign (labeled 0) and malware (labeled 1). The graphical representation of both class samples are shown in Figure 3.

Figure 3. The initial imbalance between class samples.

As seen from Figure 4, the initial investigation of target variable shows that class_0 has 1931 samples, while class_1 has 2533 samples. This is a huge imbalance between the minority (class_0) class and majority (class_1) class, which suggests that the application of dataset balancing techniques is mandatory.

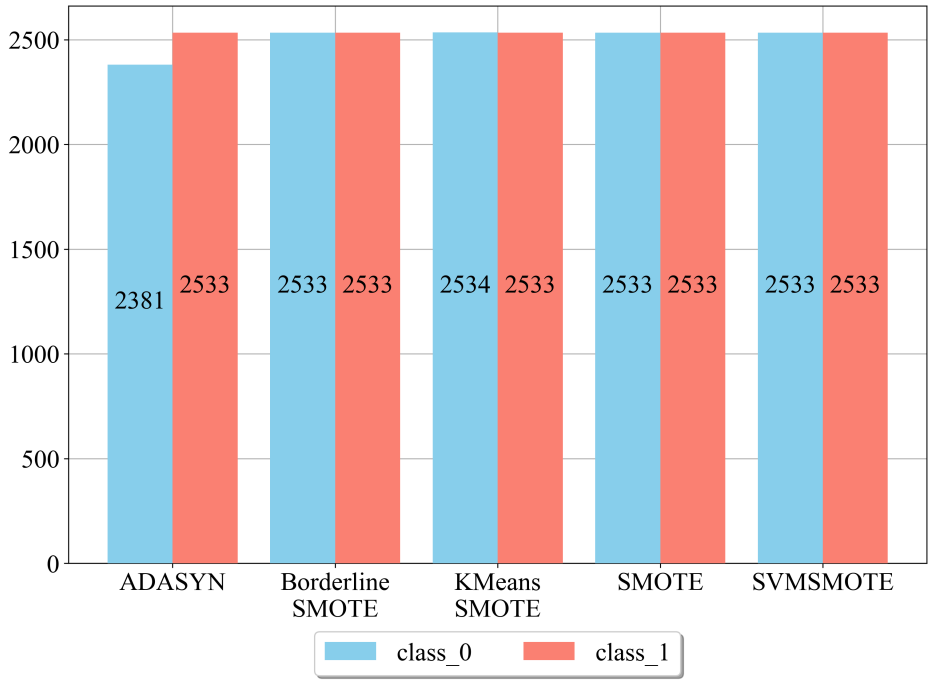

Figure 4. The balance between class samples achieved using different oversampling techniques.

2.3. Oversampling Techniques

In an unbalanced dataset for a binary classification problem, the two classes are typically referred to as the minority and majority classes. The minority class has fewer samples, while the majority class has a larger number of samples, leading to an imbalance. Training a machine learning algorithm on such an imbalanced dataset is generally not recommended, as it may result in the model becoming biased towards the majority class. To mitigate this, it is considered best practice to balance the classes, either by increasing the number of samples in the minority class or by reducing the number of samples in the majority class.

Increasing the number of minority class samples is achieved through oversampling techniques, while reducing the number of majority class samples is accomplished using undersampling techniques. The advantage of oversampling methods is that they generate additional samples for the minority class until its size is equal to or nearly equal to that of the majority class. This process typically does not require extensive parameter tuning and can be executed quickly, often within seconds.

In contrast, undersampling techniques reduce the number of samples in the majority class to match or approximate the number of minority class samples. However, undersampling is more time-consuming, as it involves identifying which majority class samples should be removed from the dataset. Additionally, undersampling methods usually require careful parameter tuning to achieve an appropriate balance between the classes.

Given the ease of application, shorter execution time, and the ability to balance class samples without extensive parameter tuning, the following oversampling techniques were employed in this research: ADASYN, BorderlineSMOTE, KMeansSMOTE, SMOTE, and SVMSMOTE.

2.3.1. ADASYN

The Adaptive Synthetic Sampling (ADASYN) technique, as described in [17], is a method designed to address class imbalance in datasets, particularly in machine learning classification tasks. ADASYN generates synthetic data samples for the minority class to achieve a more balanced class distribution. Unlike other oversampling techniques, ADASYN adaptively targets minority class samples that are more difficult to classify, producing a higher number of synthetic samples in regions where the minority class density is low and fewer samples where the density is high. This targeted approach enhances the classifier's performance on the minority class by focusing on the areas where the model is most likely to struggle.

2.3.2. BorderlineSMOTE

The Borderline Synthetic Minority Oversampling Technique (BorderlineSMOTE) is an advanced method for addressing class imbalance in datasets. Unlike traditional oversampling techniques, BorderlineSMOTE specifically focuses on generating synthetic samples for the minority class near the decision boundary between classes [18]. This approach is designed to improve the classifier's performance on difficult borderline instances that are more likely to be misclassified.

BorderlineSMOTE identifies minority class samples that are at a higher risk of misclassification—those situated close to the majority class—and generates new synthetic samples around these critical points. By concentrating on these challenging regions, BorderlineSMOTE enhances the classifier's ability to accurately differentiate between classes, making it particularly effective for complex and imbalanced datasets.

2.3.3. KMeansSMOTE

The KMeans Synthetic Minority Oversampling Technique (KMeansSMOTE) is an oversampling technique designed to handle class imbalance in datasets by combining k-means clustering with the Synthetic Minority Oversampling Technique (SMOTE) [19]. The technique involves the following steps:

1. Clustering: The minority class samples are clustered using the k-means algorithm.
2. SMOTE Application: Within each cluster, synthetic samples are generated using SMOTE, which creates new instances by interpolating between existing minority class samples.

This clustering step ensures that synthetic samples are generated in a more structured manner, respecting the natural data distribution and improving the classifier's performance on the minority class. By focusing on local structures within clusters, KMeansSMOTE can produce more relevant and diverse synthetic samples, enhancing the overall model accuracy.

2.3.4. SMOTE

The Synthetic Minority Oversampling Technique (SMOTE) is a widely used method for addressing class imbalance in datasets, particularly in machine learning classification tasks. SMOTE enhances class balance by generating synthetic samples for the minority class. The process begins by selecting a sample from the minority class and identifying its k-nearest neighbors within the same class [20]. Synthetic samples are then created along the line segments connecting the original sample to its neighbors. This approach effectively increases the diversity of the minority class, avoiding mere duplication of existing samples, and thereby improving the model's performance on imbalanced datasets.

2.3.5. SVMSMOTE

The Support Vector Machines–Synthetic Minority Oversampling Technique (SVMSMOTE) is an oversampling technique that combines the principles of SVM and SMOTE to address class imbalance in datasets [21]. The method focuses on generating synthetic samples for the minority class, particularly around the decision boundary between classes.

The SVMSMOTE consists of the following steps:

1. SVM Training: An SVM classifier is trained on the dataset to identify the support vectors, which are the samples that lie closest to the decision boundary.
2. Sample Selection: The support vectors of the minority class are identified, as they are the most informative and challenging samples.
3. SMOTE Application: Synthetic samples are generated using SMOTE by interpolating between these support vectors and their nearest neighbors within the minority class.

By concentrating on the support vectors, SVMSMOTE aims to improve the classifier's performance on the most difficult-to-classify minority class samples, thus enhancing the overall model accuracy on imbalanced datasets.

2.3.6. The Results of Oversampling Techniques

After applications of oversampling techniques, the balanced dataset variations were created. Some balanced dataset variations have a slight difference between class samples; however, these imbalances are negligible compared to the initial imbalanced dataset. The results of the application of balancing techniques is shown in Figure 4.

As seen from Figure 4, the application of BorderlineSMOTE, SMOTE, and SVMSMOTE generated perfectly balanced datasets. However, the ADASYN generated class_0 with 2381 samples, which is a 6% lower number of samples than the class_1 number of samples. It should be noted that, initially, class_0 has a 23.76% lower number of samples than class_1. The KMeansSMOTE oversampled class_0; however, there is a slight difference between samples: class_0 contains 2534, while class_1 contains 2533 samples.

2.4. Genetic Programming Symbolic Classifier

The Genetic Programming Symbolic Classifier (GPSC) is an evolutionary algorithm that begins with an initialization phase of the population. This initial population is created by randomly selecting mathematical functions, constant values, and input variables derived from the dataset. These components are organized into a primitive set, which includes terminals (i.e., input variables and constant values) and a function set (i.e., mathematical

functions). The size of the initial population is determined by the GPSC hyperparameter popSize. In this study, the constant values are defined within a range specified by the GPSC hyperparameter constantRange. The mathematical functions utilized include addition, subtraction, multiplication, division, natural logarithm, absolute value, square root, cube root, and logarithms with bases 2 and 10. Notably, to ensure the correct execution of the GPSC, some functions—specifically, division, square root, and logarithms with bases 2 and 10—were slightly modified to prevent errors; details of these modifications are provided in Appendix A. Each member of the initial population in GPSC is represented as a three-structure. For example, the symbolic expression add(mul(x_0,x_1), div(x_5, 43)) is illustrated in its three-structure form in Figure 5.

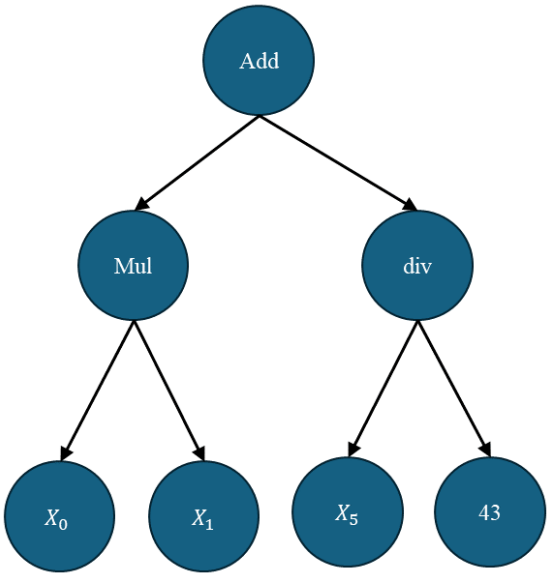

Figure 5. The tree structure of the population member in GP.

In Figure 5, the node labeled "Add" represents the root node, which is at depth 0. The functions "mul" and "div" are positioned at depth 1, while the variables x_0, x_1, x_5, and the constant 43 are located at depth 2. The size of a symbolic expression (SE) is quantified by its length, defined as the total number of elements it contains, including mathematical functions, constants, and variables. For the SE shown in Figure 5, which comprises 3 mathematical functions, 3 variables, and 1 constant, the total length is 7.

In GPSC, the initial population is generated using the ramped half-and-half method, which merges two traditional genetic programming methods: the full method and the grow method [22]. The term "ramped" refers to the range of depths for the population members. For example, a depth range of 5–20 means that the initial population will include members with tree depths varying between 5 and 20.

In the full method, half of the initial population is created by selecting nodes exclusively from the function set until the maximum depth is achieved; only terminals (variables and constants) are then used to fill the remaining nodes. This method often results in trees where all leaves are at the same depth. Conversely, the grow method generates the other half of the population by selecting nodes from the entire primitive set (function set plus terminals) until the maximum depth is reached, at which point only terminals are randomly selected. The grow method tends to produce trees with greater diversity compared to the full method. The depth range used in the ramped half-and-half method is defined by the hyperparameter InitDepth, which specifies the depth range for the population members.

After the initial population is created using the random half-and-half method, the population members (SEs) are evaluated using the fitness function. This fitness function consists of the following steps:

1. Using the training dataset to calculate the output of each SE.

2. Using the output of SEs inside the sigmoid function, which can be written as:

$$S(x) = \frac{1}{1 + e^{-x}} \tag{1}$$

where x is the output sample of the SE.

3. The final step is to calculate the Log-Loss function value, which is the fitness function value, and can be written as:

$$LogLoss = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(p_i) + (1 - y_i)\log(1 - p_i)] \tag{2}$$

where N represents the total number of samples or instances in the dataset. y_i denotes the true label or ground truth of the i-th instance. It is typically binary (0 or 1) for binary classification tasks. p_i represents the predicted probability that the i-th instance belongs to the positive class (often denoted as class 1).

In GPSC, there are two primary termination criteria: the fitness function threshold, known as stopCrit, and the maximum number of generations, denoted as maxGen. The GPSC execution terminates if the fitness function value of any population member falls below the stopCrit value. If this threshold is not reached, the algorithm continues until the maxGen limit is achieved. In this study, all GPSC runs were concluded upon reaching the maximum number of generations.

After evaluating the population, the Tournament Selection (TS) process is repeatedly conducted to determine which members advance to the next generation. Each TS involves randomly selecting a subset of population members, where the size of the subset is defined by the GPSC hyperparameter TS. The member with the lowest fitness value within this subset is chosen as the winner of the tournament.

Genetic operations, such as crossover and mutation, are then applied to these TS winners to produce offspring for the next generation. The crossover operation requires two TS winners. It involves randomly selecting subtrees from both winners and exchanging these subtrees to generate new population members. In GPSC, three types of mutations are utilized: subtree mutation, hoist mutation, and point mutation.

- Subtree Mutation (subMute): Involves selecting a random subtree from a TS winner and replacing it with a newly generated subtree from the primitive set. This operation requires only one TS member.
- Hoist Mutation (hoistMute): Randomly selects a subtree from the TS winner and replaces it with a random node, resulting in the creation of a new subtree and, consequently, a new population member.
- Point Mutation (pointMute): Randomly selects nodes within the TS winner and replaces them with nodes from the primitive set. This includes replacing constants with other constants, variables with other variables, and mathematical functions with equivalent functions of the same arity (number of arguments).

The stopping criterion stopCrit is a predefined fitness value. If any population member achieves this value, GPSC terminates. In this study, stopCrit was set to a very low value to ensure that the algorithm would terminate only upon reaching maxGen.

The constant range, constRange, defines the range of constant values used in GPSC for both initializing the population and during genetic operations. The maximum samples parameter, maxSamples, specifies the percentage of the training dataset used for each generation. For example, a value of 0.99 means that 99% of the dataset is used for training, while the remaining 1% is used for additional evaluation. The fitness value obtained from the 99% of the data is compared with the out-of-bag fitness value from the remaining 1% to ensure consistency in the evolution of GPSC.

Lastly, the parsimony coefficient, parsCoeff, is a crucial hyperparameter used to manage the size of the population members and mitigate the bloat phenomenon—where

members grow excessively without improving fitness. During the TS process, if a member's length is 50 times greater than another but has a lower fitness value, its fitness is penalized by its size multiplied by the parsimony coefficient. A high parsCoeff value prevents growth, while a low value may exacerbate bloat. Thus, careful tuning of parsCoeff is essential to balance the growth of population members and maintain effective evolution.

Since the GPSC has a lot of hyperparameter values, the problem is how to find the optimal combination of its hyperparameters, in which the use of the GPSC will produce the SEs that will generate high classification performance. Developing the RHVS method from scratch consists of the following steps:

1. Define initial lower and upper boundary value for each hyperparameter;
2. Test the GPSC with each boundary and see it will successfully execute;
3. If the GPSC execution in the previous step fails, then adjust the boundaries.

The boundaries of GPSC hyperparameter values used in the RHVS method is shown in Table 2.

Table 2. The boundaries of GPSC hyperparameter values used in the RHVS method.

GPSC Hyperparameter	Lower Boundary	Higher Boundary
popSize	1000	2000
maxGen	100	200
InitDepth	(3, 7)	(12, 18)
TS	100	500
Cross	0.001	0.1
subMute	0.95	1
hoistMute	0.001	0.1
pointMute	0.001	0.1
stopCrit	1×10^{-6}	1×10^{-3}
maxSamp	0.99	1
constRange	−10,000	10,000
parsCoeff	1×10^{-4}	1×10^{-3}

It should be noted that the sum of genetic operations should be equal to or less than 1. If the value is lower than 1, then some population members enter the next generation unchanged. As seen from Table 2, it can be noticed that the subMute coefficient will be a dominating genetic operation when compared to others. The initial investigation showed that the subMute greatly reduces the fitness function value. The benefit of using this genetic operation is that it does not require two TS winners, i.e., only one, and due to the large population in every GPSC execution, it is not memory intensive. On the other hand, this genetic operation can be described as more aggressive when compared to other genetic operations, such as hoistMute or pointMute, due to the fact that it replaces the entire subtree of the TS winner with a randomly generated subtree. The parsCoeff range initially proved to be a suitable range for this research since the coefficient value did not prevent the growth of the population members and the bloat phenomenon did not occur.

2.5. Evaluation Metrics

In this research, the accuracy score (ACC), area under receiver operating characteristics curve (AUC), precision score, recall score, $F1-score$, and confusion matrix were used to evaluate the obtained SEs and TBVE. Before explaining each evaluation metric, it is important to define the basic elements—which are used to calculate the aforementioned evaluation metrics—and these are true positive (TP), false positive (FP), true negative (TN), and false negative (FN).

- TP are instances where the model (trained ML model or SE) correctly predicts the positive class. In the case of AMD, this could be a sample that is classified as Android malware (actual positive) and the model correctly identifies this is Android malware (predicted positive). TP is the count of such correct positive predictions.
- FN are instances where the model (trained ML model or SE) incorrectly predicts the negative class for a true positive case. In the case of AMD, this is the sample that is classified as Android malware (actual positive) but the model fails to detect it (predicted negative). FN is the count of incorrect negative predictions.
- FP are instances where the model (trained ML model or SE) incorrectly predicts the positive class for the true negative case. In this case, the sample is not classified as Android malware (actual negative) but the model incorrectly predicts the sample as being Android malware (predicted postivie). FP is the count of incorrect positive predictions.
- TN are instances where the model (trained ML model or SE) correctly predicts the negative class. In this investigation, the sample is not classified as Android malware and the model correctly identifies that this is not Android malware (predicted negative). The TN is the count of correct negative predictions.

The ACC, according to [23], can be defined as the ratio of correctly predicted observations to the total number of observations. The ACC is the basic evaluation metric that gives the information about the overall effectiveness of the trained ML model or obtained SE. The equation for calculating the ACC can be written as:

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} \tag{3}$$

The range of ACC is between 0 and 1, where one is the perfect accuracy of the model or, in other words, that the trained model or obtained SE predicts all dataset samples correctly. The AUC score, according to [24], is the evaluation metric used in classification problems. This is the area under the ROC curve, and the ROC is the probability curve. The AUC represents the degree or measure of separability. This metric provides the information of how much the model is capable of distinguishing between classes. The components of ROC are the true positive rate (TPR) or recall and false positive rate (FPR). The TPR is calculated using the expression written as:

$$TPR = \frac{TP}{TP + FN} \tag{4}$$

while the FPR can be calculated using the expression

$$FPR = \frac{FP}{FP + TN} \tag{5}$$

The AUC value can be between 0 and 1, where 1 is the perfect ML model or obtained SE. The AUC of 0.5 is the model with no discriminative ability, which is similar to random guessing. A higher AUC value indicates a better performing model in distinguishing between the positive and negative class.

According to [25], the precision score is the ratio of correctly predicted positive observations to the total predicted positives. The precision score measures the accuracy of the positive predictions. The precision score is calculated using the expression written as:

$$Precision = \frac{TP}{TP + FP} \tag{6}$$

A high precision means that the model has a low FPR. This evaluation metric is very important in scenarios where FP are more costly than FN (example: email spam detection, AMD).

According to [25], the recall score, also known as the sensitivity or TPR, is the ratio of correctly predicted positive observations to all observations from the dataset. The recall score is calculated using the expression:

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

A high recall score means that the model captures most of the positive samples. This evaluation metric is crucial in situations where FN are more costly than the FP, for example, in disease detection.

The F1-score is the harmonic mean of the precision and recall score, according to [25]. This evaluation metric is the balance between precision and recall and is useful when a single metric that balances both concerns is required.

The confusion matrix [26] is a table used to describe the performance of a classification model on a set of test data, of which the true values are known. The components of the confusion matrix are TP, TN, FP, and FN, and are usually represented in a table, as in Table 3.

Table 3. The general form of the confusion matrix.

	Predicted Positive	**Predicted Negative**
Actual Positive	TP	FN
Actual Negative	FP	TN

The CM provides a complete picture of how the classification model is performing. It helps in identifying not just errors made by the model, but also provides the information on the error type (FP and FN).

2.6. Training–Testing Procedure

The training–testing procedure used in this research is graphically shown in Figure 6.

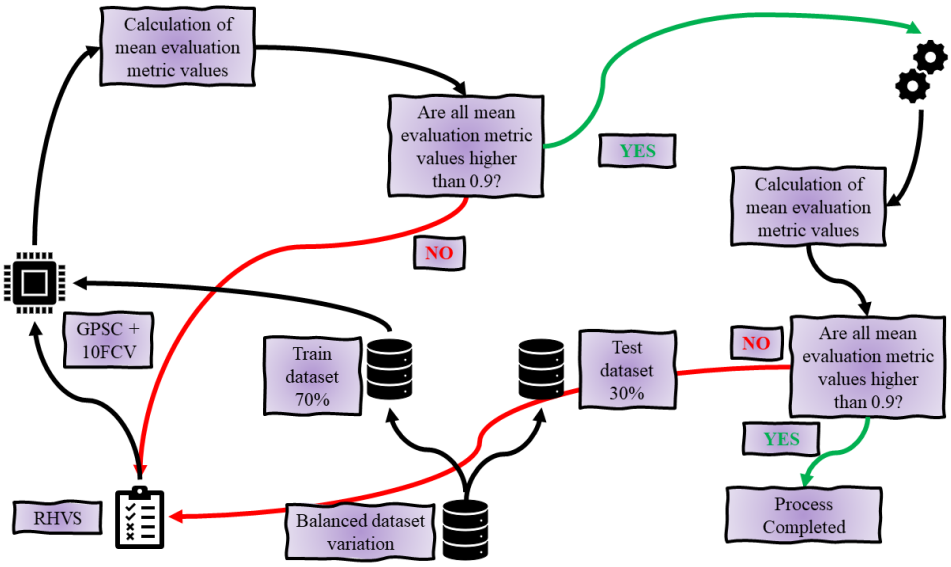

Figure 6. The graphical representation of the train/test procedure used in this research.

The training and testing procedure (Figure 6) outlined in this paper involves the following steps:

1. Dataset Splitting: Each balanced dataset variation is divided into training and testing subsets with a 70:30 ratio. The training subset is used for GPSC training with a 10-fold cross-validation (10FCV), while the testing subset is reserved for final evaluation. The SEs are tested only if the training metrics exceed a threshold of 0.9.

2. GPSC Training with 10FCV: Training begins after determining the GPSC hyperparameters using the Random Hyperparameter Values Search (RHVS) method. For each fold of the 10FCV, one SE is generated, resulting in a total of 10 SEs after completing all folds. Evaluation metrics, including Accuracy (ACC), Area Under the Curve (AUC), Precision, Recall, and F1-score are calculated for each SE on both training and validation folds.
3. Evaluation of Training Metrics: Once training is complete, the mean and standard deviation of the evaluation metrics are computed. If all metrics exceed the 0.9 threshold, the process advances to the testing phase. If any metric falls below this threshold, the procedure restarts with a new set of hyperparameters determined through RHVS.
4. Testing Phase: The set of 10 SEs is evaluated on the testing dataset, which was not used during training. The performance is assessed by comparing the SEs' predictions with the actual values in the testing dataset using the same evaluation metrics. If the metrics remain above 0.9, the process concludes. If not, the procedure is restarted with new hyperparameters selected via RHVS.

2.7. Threshold-Based Voting Ensemble

After the set of best SEs are obtained using GPSC on each balanced dataset variation, the threshold-based voting ensemble (TBVE) will be formed based on these SEs [27]. The TBVE is a method used in ensemble learning where multiple ML models/SEs (classifiers) vote on the classification of a sample, and a decision is made based on the predefined threshold of votes. The threshold could be defined as the minimum number of accurate predictions made by SEs in an ensemble per sample. This threshold must be investigated to find for which threshold value the classification performance is the highest possible.

So, the TBVE consists of multiple SEs that were, in this case, trained on multiple balanced dataset variations. Each of the SEs in the TBVE makes a prediction for a given dataset sample. The predictions are usually in the form of class labels. A threshold is set to determine how many SEs need to agree on a particular class label for it to be assigned to the sample. The threshold can be a majority vote, i.e., more than 50% of models, a plurality vote (most votes), or a fixed number of votes. However, in this paper, the threshold will be investigated from the minimum (one SE accurate prediction) to the maximum threshold (maximum number of SEs obtained). The idea is to find the threshold interval or specific threshold value for which the classification performance has the maximum value. The class label that meets or exceeds the threshold of votes is assigned to the sample.

3. Results

The results of the conducted investigation will be presented in the following order. First, the best SE performance obtained using GPSC on each balanced dataset variation will be presented. Then, the classification performance of the best SEs combined in TBVE will be presented.

3.1. Classification Performance Analysis of Obtained SE on Balanced Dataset Variations

For the best SEs obtained from each balanced dataset variation, first, the optimal combination of GPSC hyperparameter values will be listed using which of the GPSC generated the SEs with the highest possible classification accuracy.

From Table 4, it can be noticed that the popSize and the maxGen vary significantly across optimal combinations of GPSC hyperparameters obtained from each balanced dataset variation. The highest popSize (1691) and the high number of maxGen (146) were used to obtain SEs on the ADASYN dataset. The moderate popSize (1193) and the highest maxGen (197) were used to obtain SEs on the BorderlineSMOTE dataset. The SEs obtained from KMeansSMOTE, SMOTE, and SVSMOTE used a similar popSize value (1200–1600) but with a different maxGen value. The maxGen in the case of the SMOTE dataset used the lowest value to obtain SEs.

Table 4. The optimal combination of GPSC hyperparameters obtained from each balanced dataset variation using the RHVS method.

Dataset Name	GPSC Hyperparameters (popSize, maxGen, InitDepth, TS, Corss, subMute, hoistMute, pointMute, stopCrit, maxSamp, constRange, parsCoeff)
ADASYN	1691, 146, 436, (5, 13), 0.0089, 0.979, 0.001, 0.0093, 0.000656, 0.998, (−7285.5, 6882.54), 0.0004
BorderlineSMOTE	1193, 197, 281, (3, 13), 0.021, 0.96, 0.0096, 0.0071, 0.000176, 0.996, (−9967.55, 97.21), 0.00091
KMeansSMOTE	1557, 145, 423, (5, 18), 0.0031, 0.966, 0.016, 0.013, 0.000214, 0.998, (−4757.9004, 9024.429), 0.00054
SMOTE	1561, 100, 349, (3, 18), 0.0056, 0.955, 0.0015, 0.036, 0.00099, 0.994, (−3247.41, 8602.23), 0.00041
SVMSMOTE	1234, 103, 120, (7, 15), 0.022, 0.958, 0.0028, 0.015, 0.000647, 0.997, (−5775.98, 291.91), 0.0006

The cross rates are consistently low (0.0031 to 0.022), which is expected since the initial range in Table 2 is very small and narrow. The same is valid of the hoistMute and poitMute coefficients. The subMute coefficients dominate across in all optimal GPSC hyperparameter value combinations in the range (0.955 to 0.979).

The stoppCrit was extremely low and was not used as one of the GPSC termination criteria. Instead, all GPSC executions stopped after the maxGen value was reached. MaxSamp are almost constant across all datasets (around 0.994 to 0.998). The constRange shows significant variability, suggesting different ranges of constants' values used in GPSC executions on different balanced dataset variations. The parsCoeff is very low across all balanced dataset variations (0.0004 tp 0.00091), which enabled the stable growth of population members in all GPSC executions without the occurrence of the bloat phenomenon.

As seen from Figure 7, the $\overline{ACC}$ ranges from approximately 0.9402 to 0.9526 across different datasets and sampling techniques. The standard deviation of $\overline{ACC}$ is relatively low, which indicates consistent performance with minor variability (ranging from 0.0002 to 0.00031). The $\overline{AUC}$ ranges from about 0.9402 to 0.9524, showing a consistently high discriminating ability of the classifier. The standard deviation of $\overline{AUC}$ is similarly low, which indicates a stable performance (ranging from 0.0004 to 0.003). The $\overline{Precision}$ ranges from 0.9405 to 0.9595, which indicates a high accuracy in positive instance predictions. The standard deviation of $\overline{Precision}$ is generally low, with values ranging from 0.0005 to 0.0042, suggesting a relatively consistent precision performance. The $\overline{Recall}$ varies slightly from 0.9366 to 0.9495, showing the classifier's ability to correctly identify positive instances. The standard deviation of $\overline{Recall}$ ranges from 0.0001 to 0.002, which indicates stable performance across datasets. The $\overline{F1-Score}$ ranges from 0.9405 to 0.952, combining the precision and recall into a single metric. The standard deviation $\overline{F1-Score}$ ranges from 0.0006 to 0.0032, showing consistent performance across datasets.

The depth and length as well as the average depth and length of obtained SEs in this investigation are listed in Table 5.

The classification performance of the best SEs obtained from balanced dataset variations is graphically shown in Figure 7.

As seen from Table 5, the SEs obtained from the ADASN dataset showed moderately high values for both depth and length. The high variation in length (32 to 132) could indicate some instances in bloat, where certain SEs have grown excessively large without a corresponding increase in depth. In the case of SEs obtained from the BorderlineSMOTE dataset, the SEs showed a more balanced profile with relatively consistent lengths and depths. However, the average length is higher than KMeansSMOTE and SMOTE. The SEs obtained from the KMeansSMOTE dataset achieved the lowest average depth and a relatively low average length. The obtained SEs are relatively simple when compared

to the SEs obtained from other balanced dataset variations. The SEs obtained from the SMOTE dataset exhibit a low average depth and length, similar to KMeansSMOTE. The SEs obtained from the SVMSMOTE dataset achieved high average values for both depth and length, similar to ADASYN. The SEs obtained from SVMSMOTE also exhibit higher variability in length (32 to 98 range).

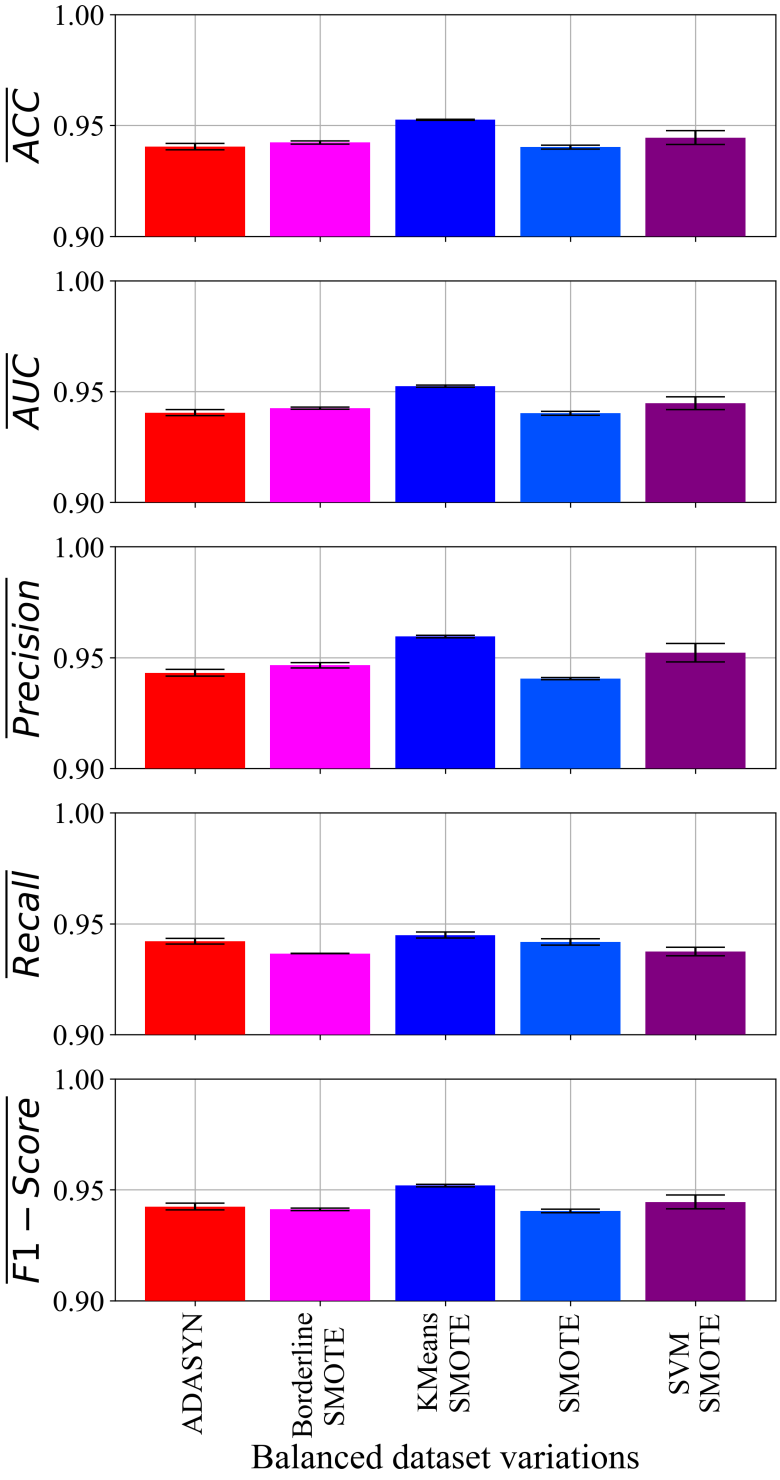

Figure 7. Classification performance of the best sets of SEs obtained from balanced dataset variations.

Table 5. The depth and length of SEs obtained from balanced dataset variations.

Dataset	Depth/Length	SE1	SE2	SE3	SE4	SE5	SE6	SE7	SE8	SE9	SE10	Average
ADASYN	Depth	14	11	14	11	20	20	6	21	12	15	14.4
	Length	54	39	97	53	123	86	33	32	51	40	60.8
BorderlineSMOTE	Depth	13	13	11	7	14	13	14	9	18	17	24.7
	Length	44	35	40	35	45	42	45	32	49	42	40.9
KMeansSMOTE	Depth	8	10	14	11	8	15	8	9	11	10	10.4
	Length	22	50	51	38	33	27	28	32	45	43	36.9
SMOTE	Depth	9	12	12	9	12	18	6	11	12	7	10.8
	Length	23	69	49	29	39	41	20	41	48	23	38.2
SVMSMOTE	Depth	13	12	20	23	15	12	11	10	9	18	14.3
	Length	80	55	92	91	98	32	34	41	41	47	61.1

The most effective (have the highest classification performance) set of SEs obtained from balanced dataset variations were those obtained from the dataset balanced with the KMeansSMOTE technique.

$$\begin{aligned} y_0 &= \mathrm{X258} - 1.\sec\left(\sqrt[4]{\tan(\mathrm{X266})}\right)\left(\frac{\mathrm{X280} - 1.\max(\mathrm{X149}, \tan(\mathrm{X306}))}{\mathrm{X262}}\right. \\ &- \left. 1.4427\log(\cos(\mathrm{X220})) + \mathrm{X280}\right). \end{aligned} \tag{8}$$

In Equation (8), the secans function was used to shorten the equation. Initially, the GPSC generated an SE where $\sec\left(\sqrt[4]{\tan(\mathrm{X266})}\right)$ was $\frac{1}{\cos\left(\sqrt[4]{\tan(\mathrm{X266})}\right)}$.

$$\begin{aligned} y_1 &= -\tan(\mathrm{X220}) - 2\mathrm{X228} - \mathrm{X246} + 2\mathrm{X247} - \frac{2\mathrm{X254}}{\mathrm{X262}} + 2\tan(\mathrm{X258}) - \frac{\mathrm{X304}}{\mathrm{X262}} \\ &+ \mathrm{X262} + \tan(\mathrm{X275}) - \mathrm{X280} - \tan(\mathrm{X280}) + \mathrm{X306} + 2\mathrm{X308} - 2\mathrm{X310}, \end{aligned} \tag{9}$$

$$\begin{aligned} y_2 &= -\cos(\mathrm{X149}) - \tan(\mathrm{X220}) - \tan(\mathrm{X228}) - \mathrm{X253} + 2\mathrm{X258} + 4\mathrm{X262} - \mathrm{X266} \\ &+ \tan(\mathrm{X275}) - 3\mathrm{X280} + \mathrm{X295} - \mathrm{X304} + \mathrm{X306} - \mathrm{X307} + \mathrm{X308} - \mathrm{X309} \\ &- \tan(\mathrm{X310}) - \tan(\mathrm{X316}), \end{aligned} \tag{10}$$

$$\begin{aligned} y_3 &= -\frac{2(\mathrm{X280} - \mathrm{X149})}{\mathrm{X262}} - 2\mathrm{X220} - \mathrm{X228} - \tan(\mathrm{X246}) + 2\mathrm{X258} - 2\mathrm{X280} + \mathrm{X285} \\ &- \mathrm{X304} + \mathrm{X306} + \mathrm{X308} - \mathrm{X310}, \end{aligned} \tag{11}$$

$$\begin{aligned} y_4 &= -\frac{\mathrm{X314} - \mathrm{X149}}{\mathrm{X306}} - \tan(\mathrm{X220}) - \mathrm{X246} + \tan(\mathrm{X258}) + 3\mathrm{X262} - \mathrm{X266} \\ &- 2\tan(\mathrm{X280}) - \mathrm{X304} + \mathrm{X308} + \mathrm{X325}, \end{aligned} \tag{12}$$

$$\begin{aligned} y_5 &= 1.4\log\left(0.43\log\left(\log\left(0.43\right.\right.\right. \\ &\left.\left.\left.\log\left(\cos\left(\frac{\log(0.43\log(\tan(\min(\mathrm{X152} - \mathrm{X228}, \mathrm{X258} - \mathrm{X280}, \mathrm{X280} - \mathrm{X316}))))}{\mathrm{X262}}\right)\right)\right)\right. \\ &+ \left.\mathrm{X262}\right)\right) + \mathrm{X149} + \mathrm{X262}, \end{aligned} \tag{13}$$

$$\begin{aligned} y_6 &= -\frac{1.\sec(\mathrm{X266})(\mathrm{X228}-1.\mathrm{X294}+\mathrm{X310})}{\mathrm{X149}}+\tan(\mathrm{X258}) \\ &- \frac{1.(\tan(\mathrm{X280})-1.\mathrm{X306})\sec(\mathrm{X322}-0.43\log(\cos(\cos(\mathrm{X298}))))}{\mathrm{X262}}, \end{aligned} \tag{14}$$

$$\begin{aligned} y_7 &= -\sqrt[3]{\sec(\tan(\mathrm{X228}))\max\left(\frac{\mathrm{np.abs(X237)}}{\mathrm{X306}},\tan(\tan(\mathrm{X280}))\right)} \\ &- \frac{1.4427\log(\cos(\mathrm{X149}+\sin(\cos(\mathrm{X258}))))}{\mathrm{X262}}-1.\mathrm{X246}+\mathrm{X308}+\mathrm{X325}+\sqrt{\mathrm{X325}}, \end{aligned} \tag{15}$$

$$\begin{aligned} y_8 &= -\tan(\max(\mathrm{X220},\mathrm{X280}))+\sqrt[3]{-\frac{\mathrm{X250}+\mathrm{X316}}{\mathrm{X149}}+\sin(\mathrm{X262})-\mathrm{X280}} \\ &- \cos(\mathrm{X149})-\mathrm{X228}-1.\mathrm{X246}+\sqrt[3]{\mathrm{X258}-\cos(\mathrm{X262})}+2.\mathrm{X262}-\mathrm{X280} \\ &+ \mathrm{X285}-1.\mathrm{X304}+\mathrm{X306}+\mathrm{X308}-\mathrm{X310}, \end{aligned} \tag{16}$$

$$\begin{aligned} y_9 &= -\max\left(\frac{\mathrm{X287}}{\mathrm{X149}}+\mathrm{X316},-\tan(\mathrm{X275}-\mathrm{X304})\right)-\max(\mathrm{X228}, \\ &- \tan(\mathrm{X275}-\mathrm{X304}))-\frac{(-\mathrm{X149}+\tan(\mathrm{X254})-\mathrm{X306}+\mathrm{X316})}{\mathrm{X262}} \\ &- \tan(\mathrm{X220})-\frac{\mathrm{X280}}{\mathrm{X262}}+\mathrm{X262}-2\mathrm{X280}+\mathrm{X306}. \end{aligned} \tag{17}$$

Equations (8)–(17) are the SEs obtained using GPSC on the dataset balanced with the KMeansSMOTE technique. To determine if the new sample is Android malware or not, the output of each equation must be input to a sigmoid function (Equation (1)). The output of the sigmoid function provides the information if the initial sample is Android malware (value near 1) or not (value near 0). After an output is obtained from all 10 equations and sigmoid functions using the average value, the final decision is made. The confusion matrix based on the previously described procedure is shown in Figure 8.

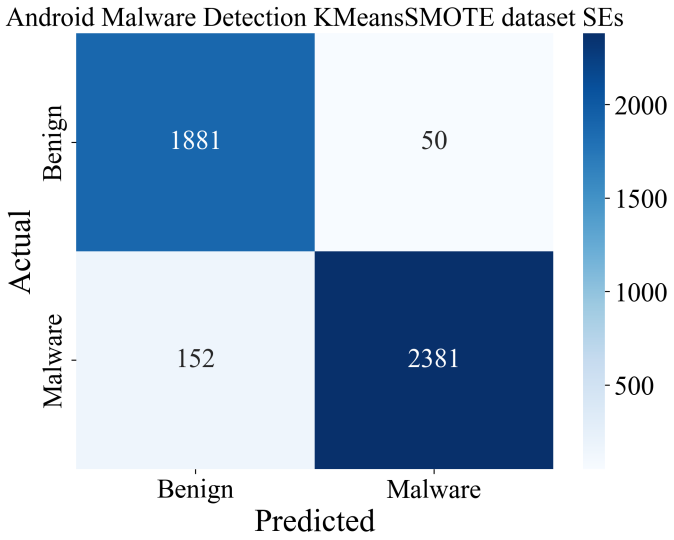

Figure 8. The confusion matrix for the best SEs obtained from KMeansSMOTE dataset.

As illustrated in Figure 8, the set of 10 symbolic expressions (SEs) generated using the KMeansSMOTE balancing technique successfully identified 2381 out of 2533 samples as Android malware and 1881 out of 1931 samples as benign.

To compute the output from these SEs (Equations (8)–(17)), 30 input variables are required. These variables, denoted in GPSC as X149, X152, X220, X228, X237, X246, X247, X250, X253, X254, X258, X262, X266, X275, X280, X285, X287, X294, X295, X298, X304, X306, X307, X308, X309, X310, X314, X316, X322, and X325, correspond to the following Android permissions and methods:

- RECEIVE_BOOT_COMPLETED: Allows an application to receive the BOOT_COMPLETED broadcast after the system finishes booting, enabling the app to perform operations when the device starts.
- RECEIVE_SMS: Allows an application to receive and read SMS messages, and trigger actions based on their content.
- Ljava/net/URL;->openConnection: Method reference for opening a connection to a specified URL.
- Landroid/location/LocationManager;->getLastKnownLocation: Method reference for retrieving the device's last known location.
- Landroid/telephony/TelephonyManager;->getSimOperatorName: Method reference for obtaining the name of the mobile network operator associated with the SIM card.
- android.permission.READ_EXTERNAL_STORAGE: Allows an application to read from external storage such as an SD card.
- android.permission.RECEIVE_SMS: Similar to RECEIVE_SMS above, allows the application to receive and process SMS messages.
- com.google.android.providers.gsf.permission.READ_GSERVICES: Allows reading Google service configuration data.
- android.permission.WRITE_EXTERNAL_STORAGE: Allows writing to external storage for saving files.
- android.permission.RECORD_AUDIO: Enables recording audio using the device's microphone.
- com.android.launcher.permission.INSTALL_SHORTCUT: Allows creating home screen shortcuts without user intervention.
- android.permission.READ_PHONE_STATE: Allows access to the phone's state, including phone number and cellular network information.
- android.permission.INTERNET: Allows the application to access the internet by opening network sockets.
- android.permission.CHANGE_CONFIGURATION: Enables changing the current configuration, such as locale or orientation.
- com.google.android.c2dm.permission.RECEIVE: Allows receiving messages from Google Cloud Messaging (GCM) or Firebase Cloud Messaging (FCM).
- android.permission.SEND_SMS: Enables sending SMS messages programmatically.
- android.permission.REQUEST_INSTALL_PACKAGES: Allows requesting the installation of packages.
- android.permission.ACCESS_COARSE_LOCATION: Allows accessing approximate location data based on network sources.
- android.permission.READ_LOGS (X295): Allows reading system log files used for debugging.
- android.permission.SYSTEM_ALERT_WINDOW: Allows creating windows that appear on top of all other applications.
- com.android.vending.BILLING: Allows using in-app billing through Google Play.
- android.permission.RECEIVE_BOOT_COMPLETED: Allows receiving the BOOT_COMPLETED broadcast (same as RECEIVE_BOOT_COMPLETED above).
- android.permission.WAKE_LOCK: Prevents the device from going to sleep, keeping the screen on, or performing tasks in the background.

- android.permission.ACCESS_FINE_LOCATION: Allows accessing precise location data using GPS.
- android.permission.BLUETOOTH: Enables connecting to paired Bluetooth devices.
- android.permission.CAMERA: Allows accessing the device's camera for taking photos or recording videos.
- android.permission.VIBRATE: Controls the device's vibration for notifications and alerts.
- android.permission.RECEIVE_USER_PRESENT: Detects when the user is present after the device wakes up.
- android.permission.ACCESS_NETWORK_STATE: Views the state of all network connections to check internet connectivity.
- android.permission.READ_SMS: Allows reading SMS messages (similar to RECEIVE_SMS above).

3.2. Classification Performance of TBVE

The previously described best SEs obtained from balanced dataset variations were used to develop the TBVE. The idea of TBVE is to see if the classification performance can be improved when compared to the individual SE. This investigation is performed by adjusting the threshold value or, in other words, the number of correct SE predictions per dataset sample. Since there are 50 SEs, the threshold value can be in a 1 to 50 range. The classification performance of the TBVE, which consists of 50 SEs, versus threshold value is shown in Figure 9.

From Figure 9, it can be noticed that ACC, AUC, and F1-Score have a similar performance. The ACC, AUC, and F1-Score value increases from 0.5 up to 0.94 as the threshold value increases fro 1 to 10. In a 10 to 40 threshold value range, the ACC, AUC, and F1-Score values increase to almost 0.959, and then they begin to decrease. The Precision score value increases from an initial 0.57 for the threshold value of 1 and increases as the threshold value increases; and when the threshold value reaches 50, the precision score value is equal to 0.995. The recall score is equal to 1 when the threshold value is equal to 1, and as the threshold value increases, the recall score value decreases and reaches a minimum of 0.855 for the threshold value of 50.

The challenge in this TBVE is to determine the range in which the classification performance is the highest. If the ACC, AUC, and F1-Score maximum values are examined, it can be noticed that for threshold values in the 27–29 (the red dashed lines in Figure 9) range, these evaluation metric values are the highest. The Precision and Recall do not have the highest values in that range butare rather acceptable. The highest classification performance in terms of ACC, AUC, and F1-Score is recorded for a threshold value of 28, and the confusion matrix for this threshold value is shown in Figure 10.

As seen from Figure 10, the TBVE was able to accurately predict the 2409 samples out of 2533 dataset samples classified as malware. This means that 124 samples were misclassified as benign. The TBVE was also able to accurately predict 1863 benign samples out of 1931 samples in total, which means 68 dataset samples were misclassified as malware.

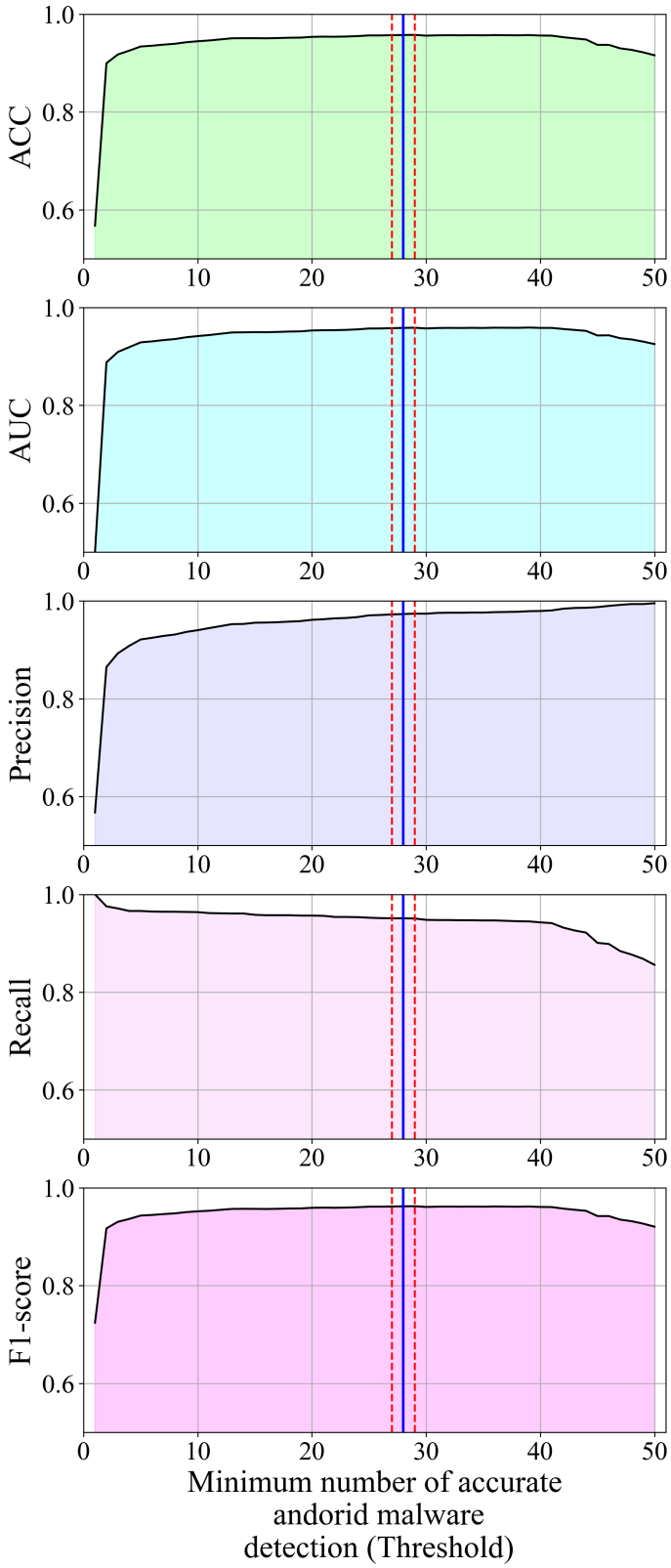

Figure 9. Classification performance of TBVE consisting of 50 SEs versus threshold value. The red dotted lines represents the interval with high classification performance while blue line represents the highest classification performance achieved.

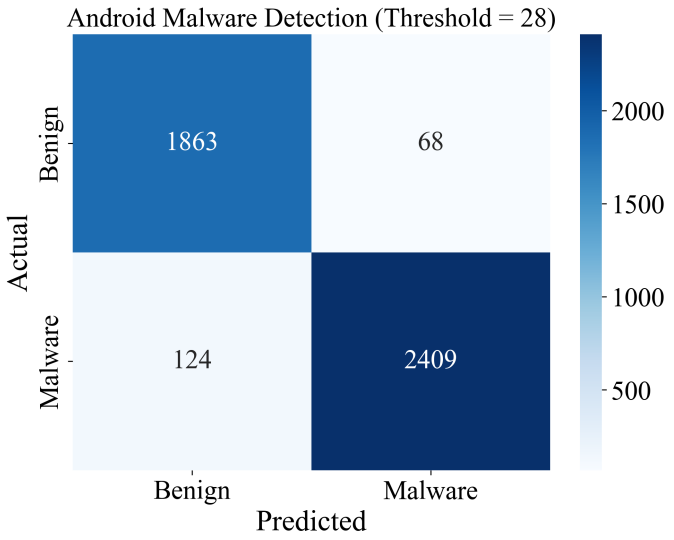

Figure 10. The confusion matrix of TBVE for a threshold value of 28.

4. Discussion

In this paper, the publicly available dataset for AMD was used in GPSC to obtain SEs that could detect Android malware with a high detection accuracy. The initial dataset consists of 328 variables, where 327 are input variables and 1 is a target variable. The dataset contains 4464 samples in total. Due to a large number of input variables, statistical analysis is almost impossible. The problem was that initially, the research was conducted with all input variables to see which one would end up in the SEs. So only the Pearson's correlation analysis was performed, and only correlated variables with target variables are shown. From Figure 2, the highly positive correlation permissions, permissions like READ_PHONE_STATE and RECEIVE_BOOT_COMPLETED, are strong indicators of malware. Security systems should flag apps requesting these permissions for further inspection. The moderate positive correlation permissions are permissions related to location (ACCESS_COARSE_LOCATION, ACCESS_FINE_LOCATION), SMS (RECEIVE_SMS, READ_SMS, SEND_SMS), and task management (GET_TASKS) and suggest a higher likelihood of the app being malware, warranting closer scrutiny. The negative correlation permissions are permissions and methods like com.google.android.c2dm.permission.RECEIVE, openConnection, and getLastKnownLocation and are more common in benign apps; their presence might indicate a lower risk of the app being malware. These correlations provide valuable insights into which permissions and features are most indicative of malware, helping in the development of better heuristics and machine learning models for malware detection.

The problem with this dataset is that it was imbalanced, so the application of dataset balancing methods was necessary. In this investigation, oversampling techniques were chosen due to the fact that they are easily executed and usually do not require additional parameter tuning. The chosen oversampling techniques (ADASYN, BorderlineSMOTE, KMeansSMOTE,SMOTE, and SVMSMOTE) achieved a balanced dataset variation. However, it should be noticed that ADASYN has a slight imbalance. However, this slight imbalance was over the initial imbalance, so this dataset was used in further investigations.

On each balanced dataset variation, the GPSC with the RHVS method was applied and the GPSC was trained using the 10FCV. This means that the GPSC was trained 10 times on 10 split variations of the train dataset. Each time the GPSC was trained, the SE was obtained by the GPSC; so in total, after one 10FCV training, 10 SEs were obtained. The evaluation metric values obtained during the training determine if the SEs will continue to the testing phase, i.e., where the test dataset will be applied on obtained SEs. If the evaluation metric values were all higher than 0.9, the process was completed and the best SEs for that balanced dataset variation were obtained.

The variability of optimal GPSC hyperparameter values across different balanced dataset variations shows the importance of the application of an RHVS method that can quickly find optimal GPSC hyperparameter values. As seen from Table 4, the larger popSizes (e.g., ADASYN dataset) may help in exploring a more extensive solution space, while a higher number of generations (e.g., BorderlineSMOTE) allows more time for evolution, potentially leading to better optimization. The variation in TS and tree depth settings suggest that different datasets benefit from varying selection pressures and model complexities. This could be due to the intrinsic complexity and noise levels within each dataset. The near-constant training size implies a consistent approach to utilizing the full capacity of the training data. The wide range of constant values reflects the need for different numerical constants to fit various datasets optimally. A low parsCoeff across all datasets emphasizes the focus of enabling the growth of the population members, which could improve the diversity between population members.

So after the application of the training–testing procedure, a total of 50 SEs were obtained. These SEs were all used in TBVE to see if the classification performance could be improved. The investigation showed that using TBVE, the highest classification performance was achieved. The best SEs obtained from balanced dataset variations (Figure 7) showed consistency across metrics. The best SEs show robust performance across all evaluated metrics (ACC, AUC, Precision, Recall, and F1-Score). The low standard deviation indicates that the best SEs' performance is relatively stable across different datasets and oversampling techniques. The balanced dataset variations were obtained using different oversampling techniques, and from each dataset, the SEs achieved high classification performance. This suggests that the SEs can generalize well across different distributions of data achieved by these oversampling methods. The high mean values across all metrics indicate that the best SEs are effective in distinguishing between classes and making accurate predictions. This effectiveness is crucial in practical applications, where high accuracy and reliable predictions are essential.

To compute the output of TBVE, the best SEs require 101 input dataset variables, and these are: X1, X6, X7, X8, X13, X14, X17, X20, X28, X29, X31, X34, X39, X41, X47, X49, X55, X62, X72, X78, X86, X87, X89, X95, X99, X103, X105, X108, X114, X118, X119, X128, X131, X134, X138, X140, X147, X149, X152, X154, X155, X156, X165, X167, X175, X180, X181, X188, X191, X200, X213, X216, X218, X219, X220, X228, X236, X240, X246, X247, X249, X253, X254, X257, X258, X262, X266, X267, X268, X269, X270, X272, X275, X278, X280, X282, X284, X285, X286, X289, X293, X294, X295, X298, X301, X302, X303, X304, X306, X307, X308, X309, X310, X312, X313, X314, X316, X319, X322, X323, X325.

These input variables are: ACCESS_CACHE_FILESYSTEM, ACCESS_LOCATION_EXTRA_COMMANDS, ACCESS_MOCK_LOCATION, ACCESS_MTK_MMHW, ACCESS_SUPERUSER, ACCESS_ SURFACE_FLINGER, ACTIVITY_ RECOGNITION, ANT, BIND_ APPWIDGET, BIND_ CARRIER_ MESSAGING_ SERVICE, BIND_ DREAM_ SERVICE, BIND_ NFC_ SERVICE, BIND_ TV_ INPUT, BIND_ VPN_ SERVICE, BRICK, BROADCAST_ SMS, CAMERA, CHANGE_ NETWORK_ STATE, DEVICE_ POWER, EXPAND_ STATUS_ BAR, GET_ TASKS, GET_ TOP_ ACTIVITY_ INFO, GOOGLE_ AUTH, INSTALL_ SHORTCUT, JPUSH_ MESSAGE, MANAGE_ APP_ TOKENS, MAPS_ RECEIVE, MEDIA_ CONTENT_ CONTROL, NFC, PLUGIN, PROCESS_ OUTGOING_ CALLS, READ_ DATABASES, READ_ GMAIL, READ_ INPUT_ STATE, READ_ PHONE_ STATE, READ_ SETTINGS, REBOOT, RECEIVE_ BOOT_ COMPLETED, RECEIVE_ SMS, RECEIVE_ WAP_ PUSH, RECORD_ AUDIO, REORDER_ TASKS, SET_ ALARM, SET_ ANIMATION_ SCALE, SET_ WALLPAPER, SUBSCRIBED_ FEEDS_ READ, SUBSCRIBED_ FEEDS_ WRITE, USE_ FINGERPRINT, WAKE_ LOCK, WRITE_ EXTERNAL_ STORAGE, WRITE_VOICEMAIL, Ljava/lang/Runtime;->exec, Ldalvik/system/DexClassLoader;->loadClass, Ljava/lang/ System;->loadLibrary, Ljava/net/URL;->openConnection, Landroid/location/LocationManager;->getLastKgoodwarewn Location, Landroid/telephony/TelephonyManager;->getSimOperator, Lorg/apache/ http/impl/client/DefaultHttpClient;->execute, android. permission. READ_ EXTERNAL_ STORAGE, android. permission. RECEIVE_ SMS, android. permission. WRITE_ SETTINGS, android.

permission. WRITE_ EXTERNAL_ STORAGE, android. permission. RECORD_ AUDIO, android. permission. CHANGE_ NETWORK_ STATE, com. android. launcher. permission. INSTALL_ SHORTCUT, android. permission. READ_ PHONE_ STATE, android. permission. INTERNET, android. permission. MOUNT_ UNMOUNT_ FILESYSTEMS, com. majeur. launcher. permission. UPDATE_ BADGE, android. permission. AUTHENTICATE_ ACCOUNTS, com. htc. launcher. permission. READ_ SETTINGS, android. permission. FLASHLIGHT, android. permission. CHANGE_ CONFIGURATION, com. anddoes. launcher. permission. UPDATE_ COUNT, com. google. android. c2dm. permission. RECEIVE, com. sonymobile. home. permission. PROVIDER_ INSERT_ BADGE, android. permission. WRITE_ CALENDAR, android. permission. SEND_ SMS, com. huawei. android. launcher. permission. WRITE_ SETTINGS, android. permission. SET_ WALLPAPER, android. permission. ACCESS_ MOCK_ LOCATION, android. permission. ACCESS_ COARSE_ LOCATION, android. permission. READ_ LOGS, android. permission. SYSTEM_ ALERT_ WINDOW, me. everything. badger. permission. BADGE_ COUNT_ READ, android. permission. CHANGE_ WIFI_ STATE, android. permission. READ_ CONTACTS, com. android. vending. BILLING, android. permission. RECEIVE_ BOOT_ COMPLETED, android. permission. WAKE_ LOCK, android. permission. ACCESS_ FINE_ LOCATION, android. permission. BLUETOOTH, android. permission. CAMERA, android. permission. FOREGROUND_ SERVICE, android. permission. BLUETOOTH_ ADMIN, android. permission. VIBRATE, android. permission. RECEIVE_ USER_ PRESENT, com. sec. android. iap. permission. BILLING, android. permission. ACCESS_ NETWORK_ STATE, com. google. android. finsky. permission. BIND_ GET_ INSTALL_ REFERRER_ SERVICE, and android. permission. READ_ SMS.

The dataset variables represent various Android permissions and API functionalities. ACCESS_ CACHE_ FILESYSTEM allows an application to read from the file system cache, while ACCESS_ LOCATION_ EXTRA_ COMMANDS provides access to additional location provider commands. ACCESS_ MOCK_ LOCATION enables applications to create mock locations for testing, and ACCESS_ MTK_ MMHW is related to MediaTek hardware features, typically requiring special hardware. ACCESS_ SUPERUSER provides elevated access levels for superuser operations, and ACCESS_ SURFACE_ FLINGER allows access to low-level graphics operations. ACTIVITY_ RECOGNITION allows applications to recognize physical activities, like walking or cycling.

ANT is related to the ANT wireless protocol used in health and fitness devices. BIND_ APPWIDGET and BIND_ CARRIER_ MESSAGING_ SERVICE are used to bind to specific system services, like widgets or carrier messaging services. BIND_ DREAM_ SERVICE and BIND_ NFC_ SERVICE bind are applications for the dream service (screensavers) and NFC service, respectively. BIND_ TV_ INPUT and BIND_ VPN_ SERVICE are used for TV input and VPN services. BRICK is a dangerous permission that can permanently disable a device.

BROADCAST_ SMS allows applications to send SMS messages, and CAMERA grants access to the device's camera. CHANGE_ NETWORK_ STATE allows applications to change network connectivity states, and DEVICE_ POWER allows for managing device power settings. EXPAND_ STATUS_ BAR lets applications expand or collapse the status bar. GET_ TASKS provides access to information about running tasks, and GET_ TOP_ ACTIVITY_ INFO provides access to information about the top activity.

GOOGLE_ AUTH is related to Google authentication services. INSTALL_ SHORTCUT allows apps to install shortcuts on the home screen, and JPUSH_ MESSAGE is specific to the JPush messaging service. MANAGE_ APP_ TOKENS allows for managing app tokens, and MAPS_ RECEIVE is used to receive map data. MEDIA_ CONTENT_ CONTROL allows controlling media playback, and NFC enables NFC communication.

PLUGIN supports plugin integration, and PROCESS_ OUTGOING_ CALLS allows for managing outgoing calls. READ_ DATABASES allows for reading databases, READ_ GMAIL allows for reading Gmail, and READ_ INPUT_ STATE allows for reading input state. READ_ PHONE_ STATE grants access to phone state information, and READ_

SETTINGS permits reading system settings. REBOOT allows the app to reboot the device, and RECEIVE_ BOOT_ COMPLETED allows apps to start after the device boots. RECEIVE_ SMS and RECEIVE_ WAP_ PUSH allow for receiving SMS and WAP push messages. RECORD_ AUDIO grants access to record audio, and REORDER_ TASKS allows for the reordering of running tasks.

SET_ ALARM allows for setting alarms, SET_ ANIMATION_ SCALE allows for setting animation scale, and SET_ WALLPAPER allows for setting wallpapers. SUBSCRIBED_ FEEDS_ READ and SUBSCRIBED_ FEEDS_ WRITE enable the reading and writing of subscribed feeds. USE_ FINGERPRINT allows for using fingerprint hardware, and WAKE_ LOCK keeps the device from sleeping. WRITE_ EXTERNAL_ STORAGE and WRITE_ VOICEMAIL allow for writing to external storage and voicemail.

API calls like Ljava/lang/Runtime;->exec, Ldalvik/system/DexClassLoader;->loadClass, Ljava/lang/System;->loadLibrary, and Ljava/net/URL;->openConnection are related to executing commands, loading classes, loading native libraries, and opening network connections. Landroid/location/LocationManager;->getLastKnownLocation provides the last known location, Landroid/telephony/TelephonyManager;->getSimOperator provides SIM operator information, and Lorg/apache/http/impl/client/DefaultHttpClient;->execute executes HTTP requests.

Permissions such as android.permission.READ_ EXTERNAL_ STORAGE, android. permission. RECEIVE_ SMS, android.permission.WRITE_ SETTINGS, and android. permission. RECORD_ AUDIO grant essential access to storage, SMS, settings, and audio recording. Other permissions like com.android. launcher. permission. INSTALL_ SHORTCUT and android.permission.MOUNT_ UNMOUNT_ FILESYSTEMS handle shortcut installation and file system mounting. Various permissions like com.majeur.launcher. permission. UPDATE_ BADGE, android. permission. AUTHENTICATE_ ACCOUNTS, and com. htc. permission. READ_ SETTINGS manage badges, account authentication, and settings for specific launchers.

android.permission.FLASHLIGHT controls the flashlight, and android.permission. CHANGE_ CONFIGURATION allows for changing system configuration. com. anddoes. launcher. permission. UPDATE_ COUNT and com. google. android. c2dm. permission. RECEIVE enable badge count updates and receive data messages. Manufacturer-specific permissions like com.sonymobile. home.permission.PROVIDER_ INSERT_ BADGE and android.permission. WRITE_ CALENDAR handle badge insertion and calendar writing.

android.permission.SEND_ SMS allows for sending SMS messages, and com.huawei. android.launcher. permission.WRITE_ SETTINGS allows for writing settings for Huawei launchers. android.permission.SET_ WALLPAPER allows for setting the wallpaper, android.permission. ACCESS_ MOCK_ LOCATION allows for creating mock locations, and android.permission. ACCESS_ COARSE_ LOCATION provides access to coarse location data. android.permission. READ_ LOGS allows for reading system logs, android.permission.SYSTEM_ ALERT_ WINDOW allows for creating alert windows, and me.everything.badger.permission. BADGE_ COUNT_ READ allows for reading badge counts.

android.permission.CHANGE_ WIFI_ STATE allows for changing the Wi-Fi state, android.permission.READ_ CONTACTS allows for reading contacts, and com.android.vending. BILLING enables in-app billing. android.permission.RECEIVE_ BOOT_ COMPLETED allows apps to start after booting, android.permission.WAKE_ LOCK keeps the device from sleeping, and android.permission.ACCESS_ FINE_ LOCATION provides access to precise location data. android.permission.BLUETOOTH and android.permission.BLUETOOTH_ ADMIN manage Bluetooth connections and settings. android.permission.CAMERA grants access to the camera, android.permission.FOREGROUND_ SERVICE allows for starting foreground services, and android.permission.VIBRATE enables vibration control.

android.permission.RECEIVE_ USER_ PRESENT detects when the user is present, and com.sec.android.iap.permission.BILLING enables in-app billing for Samsung devices. android.permission.ACCESS_ NETWORK_ STATE allows access to the network state, and com.google.android.finsky.permission.BIND_ GET_ INSTALL_ REFERRER_ SERVICE

binds to the install referrer service. Finally, android.permission.READ_ SMS grants access to read SMS messages.

The final comparison of classification performance is given in Table 6.

Table 6. Comparison of accuracy achieved in other research papers with accuracy achieved in this research paper.

Reference	Methods	Accuracy (%)
[1]	CNN	98
[2]	DNN	96.76
[3]	RFC	96.9
[4]	SVM	89
[4]	DNN	92
[5]	RNN	98.2
[6]	RFC, ANN, kNN, SVC	99
[7]	CNN	94.6
[8]	embedded call graphs	94.6
[9]	RFC, SVM	96.2
[10]	autoencoders + DNN	95.8
[11]	CNN	94.9
[12]	GAGE	87
[13]	GA-StackingMD	98.43, 98.66
[14]	ETC	98
[15]	TAML	F1-Score: 99.98
This research	GPSC + 10FCV + RHVS, TBVE	95.76

As seen from Table 6, this research outperforms the majority of research reported in other literature. Although it is not the highest classification performance achieved, the benefit of utilizing this approach is that SEs are obtained that can easily be integrated and can execute using lower computational resources when compared to CNN and DNNs.

5. Conclusions

In this paper, the AMD dataset was used in GPSC to obtain SEs with high detection performance. The initial dataset was imbalanced, so various oversampling techniques were chosen to achieve the balance between class samples. The following oversampling techniques were chosen: ADASYN, BorderlineSMOTE, KMeansSMOTE, SMOTE, and SVMSMOTE. With the application of oversampling techniques, five different balanced dataset variations were created and were used in GPSC to obtain SEs. Since GPSC has 12 hyperparameters, finding the optimal combination of these hyperparameters can be a challenge. This is why the RHVS method was applied to find the optimal combination of GPSC hyperparameter values. The training was conducted using 10FCV, which means that after training, GPSC generated 10 SEs (one per each split). After the best set of SEs were obtained for each balanced dataset variation, these were combined in TBVE to investigate if the the classification performance could be improved. The TBVE showed an improvement of classification performance by adjusting the threshold to the specific range. Based on the previous description, the following conclusions are:

- The GPSC-generated symbolic expressions (SEs) demonstrated high classification performance in detecting Android malware.

- The use of oversampling techniques effectively balanced the initial dataset, allowing the GPSC to produce SEs with high classification performance in Android malware detection (AMD).
- The Random Hyperparameter Value Selection (RHVS) method proved effective in optimizing GPSC hyperparameters, leading to the generation of SEs with superior classification performance in AMD.
- The application of 10-fold cross-validation (10FCV) during GPSC training resulted in a robust set of SEs, exhibiting high accuracy in AMD, thus validating the use of 10FCV.
- Combining SEs through the Training-Based Variable Evaluation (TBVE) method further enhanced classification performance for AMD, achieving even higher accuracy.

The pros and cons of the conducted research are given below. Pros of the Conducted Research:

- The application of oversampling techniques effectively balanced the original dataset, resulting in the creation of diverse dataset variations. This approach facilitated the generation of more symbolic expressions (SEs) using the GPSC, leading to highly accurate SEs that enhanced the robustness of Android malware detection (AMD).
- The use of the Random Hyperparameter Value Selection (RHVS) method streamlined the process of identifying optimal GPSC hyperparameters, requiring fewer executions to achieve optimal configurations.
- Employing 10-fold cross-validation (10FCV) allowed for the generation of a substantial number of SEs, contributing to a robust detection framework for Android malware.
- Combining GPSC with the Training-Based Variable Evaluation (TBVE) method proved effective in improving classification performance compared to the initial SEs.

Cons of the Proposed Research Methodology:

- The methodology was time-consuming, primarily due to the large population size (popSize) and high number of generations (maxGen) used. This extended the time required to obtain all SEs.

Future Work:

- Future research will explore the reduction of the values of popSize and maxGen to determine if similar levels of classification performance can be maintained with fewer resources.
- Given the large sample size of the dataset, the application of undersampling techniques will also be investigated. This could potentially lead to the development of additional SEs and the formation of an even larger TBVE.
- The proposed methodology will be tested on additional datasets, such as Androzoo or CICMalDroid 2020, to evaluate its effectiveness and generalizability.

Author Contributions: Conceptualization, N.A. and S.B.Š.; methodology, N.A. and S.B.Š.; software, N.A. and S.B.Š.; validation, N.A. and S.B.Š.; formal analysis, N.A. and S.B.Š.; investigation, N.A. and S.B.Š.; resources, N.A. and S.B.Š.; data curation, N.A. and S.B.Š.; writing—original draft preparation, N.A. and S.B.Š.; writing—review and editing, N.A. and S.B.Š.; visualization, N.A. and S.B.Š.; supervision, N.A. and S.B.Š.; project administration, N.A. and S.B.Š.; funding acquisition, N.A. and S.B.Š. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: In this research, the publicly available dataset from Kaggle was used: https://www.kaggle.com/datasets/dannyrevaldo/android-malware-detection-dataset (accessed on 1 July 2024).

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

10FCV	10-Fold Cross-Validation
AMD	Android Malware Detection
ADASYN	Adaptive Synthetic Sampling
BorderlineSMOTE	Borderline Synthetic Minority Oversampling Technique
Cross	Crossover
constRange	constants range
GPSC	Genetic programming symbolic classifier
hoistMute	hoist mutation
InitDepth	inital depth
KMeansSMOTE	KMeans Synthetic Minority Oversampling Technique
maxGen	maximum number of generations
maxSamp	maximum number of samples
parsCoeff	parsimony coefficient
popSize	Population Size
pointMute	point mutation
RHVS	Random hyperparameter value search method
SMOTE	Synthetic Minority Oversampling Technique
subMute	subtree mutation
stopCrit	stopping criteria
SVMSMOTE	Support Vector Machines Synthetic Minority Oversampling Technique
SE	Symbolic expression
TS	Tournament size

Appendix A. The Modified Versions of Mathematical Functions Used in GPSC

As stated in the description of the GPSC, some mathematical functions have to be modified in order to avoid nan or imaginary values, which could lead to early termination and failure of GPSC execution. These functions are: division, square root, natural logarithm, and logarithm with bases 2 and 10. The division function is defined as:

$$y_{div}(x_1, x_2) = \begin{cases} \frac{x_1}{x_2} & if|x_2| > 0.001 \\ 1 & |x_2| < 0.001 \end{cases} \tag{A1}$$

It should be noted that x_1 and x_2 are two variables introduced solely for demonstration purposes and do not pertain to the dataset variables.

The square root function in this context first computes the absolute value of the argument before applying the square root. This function can be expressed as:

$$y_{sqrt} = \sqrt{|x|} \tag{A2}$$

The natural logarithm and logarithms with bases 2 and 10 are defined as follows:

$$y_{\log}(x) = \begin{cases} \log_i(x) & if|x| > 0.001 \\ 0 & if|x| < 0.001 \end{cases} \tag{A3}$$

where i indicates the natural logarithm, bases 2 or 10, respectively. So all three logarithms work the same way, i.e., if the $|x|$ is greater than 0.001, the function will calculate the logarithm value. Otherwise, it will return 0. Where i denotes the base of the logarithm: $i = e$ for the natural logarithm, $i = 2$ for the logarithms with base 2, and $i = 10$ for logarithms with base 10. All these logarithms are implemented similarly: if $|x|$ is greater than 0.001, the function computes the logarithm of x, otherwise, it returns 0.

Appendix B. Aquisition of Obtained SEs from This Resarch

The Symbolic Expressions (SEs) generated in this research are available for download at: https://github.com/nandelic2022/AndroidMalwareDetection_MDPI (accessed on 15 July 2024). To apply these SEs to new data, follow these steps:

1. Download and prepare: Download the SEs and prepare the corresponding Python script for use with your new dataset.
2. Variable correspondence: Ensure that the variables in your dataset align with those defined in the Discussion section of this paper.
3. Class determination: After obtaining the output from the SEs, apply the sigmoid function (Equation (1)) to determine the class.

References

1. McLaughlin, N.; Martinez del Rincon, J.; Kang, B.; Yerima, S.; Miller, P.; Sezer, S.; Safaei, Y.; Trickel, E.; Zhao, Z.; Doupé, A.; et al. Deep android malware detection. In Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy, Scottsdale, AZ, USA, 22–24 March 2017; pp. 301–308.
2. Yuan, Z.; Lu, Y.; Xue, Y. Droiddetector: Android malware characterization and detection using deep learning. *Tsinghua Sci. Technol.* **2016**, *21*, 114–123. [CrossRef]
3. Saracino, A.; Sgandurra, D.; Dini, G.; Martinelli, F. Madam: Effective and efficient behavior-based android malware detection and prevention. *IEEE Trans. Dependable Secur. Comput.* **2016**, *15*, 83–97. [CrossRef]
4. Demontis, A.; Melis, M.; Biggio, B.; Maiorca, D.; Arp, D.; Rieck, K.; Corona, I.; Giacinto, G.; Roli, F. Yes, machine learning can be more secure! a case study on android malware detection. *IEEE Trans. Dependable Secur. Comput.* **2017**, *16*, 711–724. [CrossRef]
5. Yuan, X.; He, P.; Zhu, Q.; Li, X. Adversarial examples: Attacks and defenses for deep learning. *IEEE Trans. Neural Netw. Learn. Syst.* **2019**, *30*, 2805–2824. [CrossRef] [PubMed]
6. Mariconti, E.; Onwuzurike, L.; Andriotis, P.; De Cristofaro, E.; Ross, G.; Stringhini, G. Mamadroid: Detecting android malware by building markov chains of behavioral models. *arXiv* **2016**, arXiv:1612.04433.
7. Hou, S.; Saas, A.; Chen, L.; Ye, Y. Deep4maldroid: A deep learning framework for android malware detection based on linux kernel system call graphs. In Proceedings of the 2016 IEEE/WIC/ACM International Conference on Web Intelligence Workshops (WIW), Omaha, NE, USA, 13–16 October 2016; IEEE: Piscataway, NJ, USA, 2016; pp. 104–111.
8. Gascon, H.; Yamaguchi, F.; Arp, D.; Rieck, K. Structural detection of android malware using embedded call graphs. In Proceedings of the 2013 ACM Workshop on Artificial Intelligence and Security, Berlin, Germany, 4 November 2013; pp. 45–54.
9. Milosevic, N.; Dehghantanha, A.; Choo, K.K.R. Machine learning aided Android malware classification. *Comput. Electr. Eng.* **2017**, *61*, 266–274. [CrossRef]
10. Hardy, W.; Chen, L.; Hou, S.; Ye, Y.; Li, X. DL4MD: A deep learning framework for intelligent malware detection. In Proceedings of the International Conference on Data Science (ICDATA), Cochin, India, 23–25 August 2016; The Steering Committee of the World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp): Seattle, WA, USA, 2016; p. 61.
11. Ding, Y.; Zhang, X.; Hu, J.; Xu, W. Android malware detection method based on bytecode image. *J. Ambient Intell. Humaniz. Comput.* **2023**, *14*, 6401–6410. [CrossRef]
12. Saqib, M.; Fung, B.C.; Charland, P.; Walenstein, A. GAGE: Genetic Algorithm-Based Graph Explainer for Malware Analysis. In Proceedings of the 2024 IEEE 40th International Conference on Data Engineering (ICDE), Utrecht, The Netherlands, 13–16 May 2024; IEEE: Piscataway, NJ, USA, 2024; pp. 2258–2270.
13. Xie, N.; Qin, Z.; Di, X. GA-StackingMD: Android malware detection method based on genetic algorithm optimized stacking. *Appl. Sci.* **2023**, *13*, 2629. [CrossRef]
14. Jyothsna, V.; Dasari, K.P.; Inuguru, S.; Gowni, V.B.R.; Kudumula, J.T.R.; Srilakshmi, K. Unified Approach for Android Malware Detection: Feature Combination and Ensemble Classifier. In Proceedings of the International Conference on Computational Innovations and Emerging Trends (ICCIET-2024), Amalapuram, India, 4–5 April 2024; Atlantis Press: Amsterdam, The Netherlands, 2024; pp. 485–495.
15. AlSobeh, A.M.; Gaber, K.; Hammad, M.M.; Nuser, M.; Shatnawi, A. Android malware detection using time-aware machine learning approach. *Clust. Comput.* **2024**, 1–22. [CrossRef]
16. Sedgwick, P. Pearson's correlation coefficient. *BMJ* **2012**, *345*, e4483. [CrossRef]
17. He, H.; Bai, Y.; Garcia, E.A.; Li, S. ADASYN: Adaptive synthetic sampling approach for imbalanced learning. In Proceedings of the 2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence), Hong Kong, China, 1–6 June 2008; IEEE: Piscataway, NJ, USA, 2008; pp. 1322–1328.
18. Han, H.; Wang, W.Y.; Mao, B.H. Borderline-SMOTE: A new over-sampling method in imbalanced data sets learning. In Proceedings of the International Conference on Intelligent Computing, Hefei, China, 23–26 August 2005; Springer: Berlin/Heidelberg, Germany, 2005; pp. 878–887.
19. Fonseca, J.; Douzas, G.; Bacao, F. Improving imbalanced land cover classification with K-Means SMOTE: Detecting and oversampling distinctive minority spectral signatures. *Information* **2021**, *12*, 266. [CrossRef]

20. Fernández, A.; Garcia, S.; Herrera, F.; Chawla, N.V. SMOTE for learning from imbalanced data: Progress and challenges, marking the 15-year anniversary. *J. Artif. Intell. Res.* **2018**, *61*, 863–905. [CrossRef]
21. Miftahushudur, T.; Sahin, H.M.; Grieve, B.; Yin, H. Enhanced SVM-SMOTE with Cluster Consistency for Imbalanced Data Classification. In Proceedings of the International Conference on Intelligent Data Engineering and Automated Learning, Évora, Portugal, 22–24 November 2023; Springer: Cham, Switzerland, 2023; pp. 431–441.
22. O'Neill, M.; Poli, R.; Langdon, W.B.; McPhee, N.F. *McPhee: A Field Guide to Genetic Programming: Lulu. com*; Springer: Berlin/Heidelberg, Germany, 2008; 250p, ISBN 978-1-4092-0073-4.
23. Ilse, M.; Tomczak, J.M.; Welling, M. Chapter 22—Deep multiple instance learning for digital histopathology. In *Handbook of Medical Image Computing and Computer Assisted Intervention*; Zhou, S.K., Rueckert, D., Fichtinger, G., Eds.; The Elsevier and MICCAI Society Book Series; Academic Press: Cambridge, MA, USA, 2020; pp. 521–546. [CrossRef]
24. Mandrekar, J.N. Receiver Operating Characteristic Curve in Diagnostic Test Assessment. *J. Thorac. Oncol.* **2010**, *5*, 1315–1316. [CrossRef] [PubMed]
25. Goutte, C.; Gaussier, E. A probabilistic interpretation of precision, recall and F-score, with implication for evaluation. In Proceedings of the European Conference on Information Retrieval, Santiago de Compostela, Spain, 21–23 March 2005; Springer: Berlin/Heidelberg, Germany, 2005; pp. 345–359.
26. Singh, P.; Singh, N.; Singh, K.K.; Singh, A. Chapter 5—Diagnosing of disease using machine learning. In *Machine Learning and the Internet of Medical Things in Healthcare*; Singh, K.K., Elhoseny, M., Singh, A., Elngar, A.A., Eds.; Academic Press: Cambridge, MA, USA, 2021; pp. 89–111. [CrossRef]
27. Anđelić, N.; Baressi Šegota, S. An Advanced Methodology for Crystal System Detection in Li-Ion Batteries. *Electronics* **2024**, *13*, 2278. [CrossRef]