IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received March 6, 2021, accepted April 3, 2021, date of publication April 7, 2021, date of current version April 20, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3071450

Android Ransomware Detection Based on a Hybrid Evolutionary Approach in the Context of Highly Imbalanced Data

IMAN ALMOMANI[1,5], (Senior Member, IEEE), RANEEM QADDOURA[2], MARIA HABIB[3], SAMAH ALSOGHYER[4], ALAA AL KHAYER[1], IBRAHIM ALJARAH[5], AND HOSSAM FARIS[5,6]

[1]Security Engineering lab, Computer Science Department, Prince Sultan University, Riyadh 11586, Saudi Arabia
[2]Department of Information Technology, Philadelphia University, Amman 11118, Jordan
[3]Altibbi.com, Amman 11118, Jordan
[4]King Abdulaziz City for Science and Technology, Riyadh 11442, Saudi Arabia
[5]King Abdullah II School for Information Technology, The University of Jordan, Amman 11118, Jordan
[6]School of Computing and Informatics, Al Hussein Technical University, Amman 11118, Jordan

Corresponding author: Hossam Faris (hossam.faris@ju.edu.jo)

This work was supported by the Prince Sultan University.

ABSTRACT In recent years, Ransomware has been a critical threat that attacks smartphones. Ransomware is a kind of malware that blocks the mobile's system and prevents the user of the infected device from accessing their data until a ransom is paid. Worldwide, Ransomware attacks have led to serious losses for individuals and stakeholders. However, the dramatic increase of Ransomware families makes to the process of identifying them more challenging due to their continuously evolved characteristics. Traditional malware detection methods (e.g., statistical-based prevention methods) fail to combat the evolving Ransomware since they result in a high percentage of false positives. Indeed, developing a non-classical, intelligent technique to safeguarding against Ransomware is of significant importance. This paper introduces a new methodology for the detection of Ransomware that is depending on an evolutionary-based machine learning approach. The binary particle swarm optimization algorithm is utilized for tuning the hyperparameters of the classification algorithm, as well as performing feature selection. The support vector machines (SVM) algorithm is used alongside the synthetic minority oversampling technique (SMOTE) for classification. The utilized dataset is collected from various sources, which consists of 10,153 Android applications, where 500 of them are Ransomware. The performance of the proposed approach SMOTE-*t*BPSO-SVM achieved merits over traditional machine learning algorithms by having the highest scores in terms of sensitivity, specificity, and g-mean.

INDEX TERMS Ransomware, evolutionary algorithms, imbalanced, particle swarm optimization, support vector machines, SMOTE, ADASYN.

I. INTRODUCTION

Recently, the market share of Android mobile operating system (OS) has approximately reached 72.97% by Q4 2020.[1] However, this rapid evolution of the Android market has attracted many attackers to gain illegal access to Android devices and data using malware applications. Malware is a malicious application that is developed to cause harmful attacks on mobile devices. Many forms of malware applications can infect the victim's device, including but not limited to Trojans, Spyware, and Ransomware. Among the aforementioned types, Ransomware has been recording a dramatic increase by recent studies [1]–[3]. Ransomware intrudes the device's OS by using a malicious code that blocks the access of the victim's data unless a ransom is paid [4]. Ransomware developers have diversely created well-established methods to cause monetary damages to their victims. Consequently, this type of malware has constituted as one of the most threatening attacks targeting both individuals and organizations

The associate editor coordinating the review of this manuscript and approving it for publication was Mostafa M. Fouda.

[1]https://gs.statcounter.com/os-market-share/mobile/worldwide

with financial losses to billions of dollars, which harmed one million Android users in one month [5], [6]. Additionally, Ransomware's recent success results in the manifestation of new families [4].

Several approaches have been proposed to enforce the security on the Android platform, including malware detection, vulnerability detection, and application reinforcement [7]. Among the suggested security protection approaches, malware detection is broadly implemented to prevent malicious applications from being published in the Android marketplace. Several malware detection approaches have been suggested that can be classified into three main categories: static analysis approaches, dynamic approaches, and hybrid approaches [8]. The static analysis identifies the malware application by scanning the source code of the application without running it. It uses reverse engineering techniques to retrieve the source code of the Android application package (APK). Based on the implemented reverse engineering technique, various static features can be extracted (e.g., the permissions and API calls features) and further utilized in the malware detection process [9]. On the other hand, in the dynamic approach, the APK is executed and equipped to examine the app's behavior and create executions and data flows of the application. However, executing the malware has to be performed in a virtual environment to mitigate the influence of any external factors that might affect the malware behavior [10]. To improve the malware detection efficiency, the hybrid detection approach can be implemented since it combines a variety of run-time and application features. Indeed, in the hybrid approach both static and dynamic techniques are utilized in the classification process [11].

Generally, malware detection approaches can be classified into signature-based, and anomaly-based approaches. The former, depends on a database of predefined characteristics of such threats in order to identify the malicious behaviours. However, even this approach can identify accurately the previously known malware, but it lacks the ability to recognize new unseen malicious behaviours. Whereas, the latter, attempts to identify the malicious behaviours by continuously measuring any deviations in the network from the known normal behaviours. As the anomaly-based approaches do not require a predefined knowledge of malware, they are more efficient in detecting novel unseen malware. In comparison to classical (i.e., statistical, and knowledge-based) techniques, the performance of malware detection algorithms that are based on machine learning-approaches surpass the traditional methods [12]–[15].

A key aspect when developing a machine learning model is to utilize a set of relevant, non-redundant features, since the quality of the features might promote or deteriorate the performance of the algorithm [16], [17]. Primarily, having a number of n features leads to a search space of size of 2^n. Selecting the optimal set of features from such search space demands the adoption of search algorithms that optimize and maximize the performance, while finding the optimal solution in reasonable amount of time. A well-regarded type of search algorithms is the metaheuristic algorithms. Metaheuristics are stochastic search algorithms that integrate a randomization process and consist of two major components; the exploration and exploitation.

Evolutionary algorithms are type of metaheuristics that are inspired by different natural phenomena, such as the Darwinian principles of evolution and natural selection, as well as, the collective swarming behaviour of birds, insects, or other living organisms in ecological systems. Evolutionary algorithms are classified into population-based, and trajectory-based algorithms. Population-based methods are more exploration-oriented, while the trajectory-based are more exploitation-oriented. Particle swarm optimization (PSO) is an evolutionary, population-based, and swarm-intelligence algorithm. It is attributed to Kennedy and Eberhart [18], which is developed to mimic the social and collective behaviour of bird flocking. The potential solutions are known by particles that also play the role of birds, where each particle has a velocity and position components. This paper utilizes the PSO algorithm to search for the optimal set of features, as well as to optimize several hyperparameters of the classification algorithm.

The proposed approach aims to detect the Ransomware by developing an evolutionary machine learning-based approach. The proposed method utilizes the support vector machines (SVM) algorithm [19] for identifying the Ransomware, while the PSO is to optimize the search process by optimizing the number of features and other hyperparameter coefficients. The integrated data was collected from different tools, such as: the Google play, VirusTotal, Ransomware-Proper, and Koodous. The collected data was decompiled into Smali files, which then parsed to extract different types of features (e.g., the permissions and API calls). Certainly, the collected set of data is imbalanced dataset, where the normal (non-Ransomware) is the dominant class. This poses challenges for the classification algorithm during the learning to not bias toward the major class and results in overfitting, but to have a balanced performance at each class. Hence, different oversampling algorithms were adopted and experimented, such as the synthetic minority over-sampling technique (SMOTE) [20], borderline-SMOTE [21], and the adaptive synthetic (ADASYN) sampling [22]. Overall, iteratively, the PSO algorithm searches for the optimal set of features, the optimal number of nearest neighbors, and the sampling ratio of the oversampling method, in addition to the cost (C) of the linear SVM. The proposed model is assessed by the sensitivity, the specificity, and the g-mean, which achieved outperforming results and merits over classical machine learning algorithms.

The objective of this paper is to achieve a high performance malware detection with an immensely imbalanced dataset. As in Android market store, the percent of Ransomware applications is low in comparison to the benign applications [23]. Accordingly, the collected dataset simulates the status of the current market by creating an imbalanced dataset. The main contributions of the paper are summarized as follows.

1) Present a comprehensive discussion on the state-of-the-art in Ransomware detection systems.
2) Provide an up-to-date dataset of the permissions and API calls of Android OS by considering the latest Android release (version 11, API level 30). This will be conducted by mimicking the real-market status by creating an imbalanced dataset of benign and Ransomware applications.
3) Proposing a swarm-based machine learning detection system that combines PSO with SVM and an oversampling technique for performing classification, feature selection and data balancing, simultaneously.
4) The proposed approach automatically tunes the parameters of SVM and the incorporated oversampling technique to overcome any effort needed for this task.
5) In addition to its detection power, the proposed approach will help in identifying the most influencing features in the detection process.

The rest of the paper is organized as follows. Section II presents a comprehensive survey of the related work. Section III discusses preliminaries and the theories of the utilized algorithms in the proposed approach. Section IV outlines the creation of the imbalanced dataset including the parsing and decompiling phases. Section V describes the proposed classification approach for Ransomware detection. In Section VI, the model evaluation metrics are presented. Section VII discusses the experiments and results. Finally, Section VIII concludes the paper and points out potential future works.

II. RELATED WORKS

Several studies have investigated the impact of utilizing machine learning approaches in detecting Android Ransomware. This section highlights recent studies that applied machine learning in their solutions.

Static-based detection approaches applied different machine learning to enhance the accuracy of their detection systems. The authors in [24] conducted a deep analysis of Android APIs for benign and Ransomware apps. The objective was to determine the list of APIs that significantly impact identifying Ransomware. Consequently, using them to build a detection system based on the selected feature set. In [25], the authors extracted the opcodes in native instructions. They computed the opcodes' frequency information from Android applications and used them as features that are then forwarded to a classifier for evaluation. Whereas the authors in [26] detected a locker-based Ransomware using a feature set that combines the displayed text and background operations. Such operations include admin, window, system, priority, and permissions. Then they evaluated the extracted features on different machine learning algorithms.

Moreover, [27] investigated the appearance of some information that is related to Ransomware operations. The authors examined 48 permissions, 4 intents and 34 APIs in inspected applications before integrating some supervised machine learning models to classify the applications. Kim *et al.* [28] utilized also static features to reflect malware and benign apps' properties. They used a multimodal deep learning method, which was designed to deal with various kinds of feature types. Additionally, Pektaş and Acarman [29] analyzed applications to construct a flow graph, then extracting features. After that, building a deep neural network for malware detection. On the other hand, the research done by [30] extracted the system APIs information through packages, classes, and methods. Then, the authors employed the detection model using different classifiers. Moreover, Singh *et al.* [31] identified malware using Latent Semantic Indexing as an information retrieval technique with a lower-dimensional representation of opcodes.

The work in [4] compared the permissions frequency obtained by Ransomware/benign apps by analyzing the AndroidManifest.xml file. Wheareas, Alsoghyer and Almomani [32] conducted a deep analysis of permissions requested by apps. They proposed a permissions-based Ransomware detection system and evaluated it using different classification algorithms. On the other hand, in [33] and in [34], the classification of Android apps into malware/benign was done by examining both the permissions and API calls frequency distributions. The approach in [33] in specific detected Ransomware apps using a proposed hybrid swarm-intelligent machine learning approach to optimize the hyperparameters and perform feature selection.

Dynamic-based detection approached have also applied machine learning techniques in their systems. Chen *et al.* [35] recognized user interface differences between benign and Ransomware apps with coordinates of the user's finger movements. The authors used user interface and information entropy of files before and after the encryption. Where the dynamic mechanism in [36] was comprised of preprocessing and detection. The preprocessing phase was included with the extraction of important features using eight machine learning filtration techniques. Additionally, the detection phase detected malicious behavior of application using deep learning algorithm. 19 important features related to network packets and headers were selected through a simple majority voting process by comparing all feature filtration techniques.

In other context, the dynamic detection system in [10] was based on selecting the most significant system calls that were used by a classification algorithm to discriminate Ransomware from benign app. Besides, a hybrid static/dynamic solution proposed in [37] distinguished Ransomware by comparing the structural similarity of an app and known Ransomware. Threatening text and images were traced in this regard.

Table 1 summarizes and compares different Ransomware detection systems. The comparison was conducted in terms of the type of detection system whether static or dynamic or hybrid, the used features list, the applied machine learning approach(s), the accuracy of the detection system, the source of the studied Android apps, the resulted dataset size and whether this dataset was balanced or not.

As can be observed from the table, most of the related works have considered balanced datasets. Even in the limited studies that considered imbalanced datasets in their Ransomware detection solution, the imbalance ratio was low except in work [21]. But, in this work the authors have focused only on the Chinese market and the locker-based ransomware. Additionally, when the authors compared their approach with other related study, they have chosen only 1500 benign apps. They mainly compared their work with R-PackDroid, which we already compared within our previous research work [19]. Moreover, the existing methods have considered different datasets from different sources with also different features set.

In this study, an imbalanced dataset with only 5% of Ransomware apps are included in the overall dataset of apps. Consequently, making it even more challenging to detect Ransomware with high accuracy. As shown in Table 1, most of the existing methods have applied standard and well-known classifiers and compared their detection performance. This was also implemented by our research while considering more appropriate evaluation metrics in case of imbalanced datasets to examine the proposed Ransomware detection system's performance, including g-mean, sensitivity, and specificity.

III. PRELIMINARIES

In this section, we describe the algorithms utilized in the proposed approach for ransomware detection.

A. PARTICLE SWARM OPTIMIZATION

Particle swarm optimization (PSO) is a swarm-based and naturally-inspired algorithm that is created by Kennedy and Eberhart in the 90s [18]. PSO is a stochastic optimization and metaheuristic algorithm inspired by the collective social behavior of bird flocking. As the birds search in swarms for the food, the PSO searches stochastically for the optimal solutions in the search space of an objective function in a similar approach. The PSO algorithm consists of a swarm of random particles, where each particle is characterized by two components: the velocity and position. Each particle might represent an optimal solution, while simultaneously the algorithm adjusts and assess the particles in order to find the best solution (particle) that maximizes the learning performance. The velocity and position of a particle depend on the experience of the particle itself (the cognitive component), and the other particles' experiences (the social component). Mathematically, the particles adjust their velocities and positions and move toward the optimal region. The particle that is in the optimal region is an optimal solution, thereby the algorithm evaluates the particles iteratively during the learning process. Equation 14 represents the velocity of a particle, while Equation 15 represents the position.

$$v_{id}(t+1) = w * v_{id}(t) + r_1 * c_1 * (p_{id}(t) - x_{id}(t)) + r_2 * c_2 * (g_d(t) - x_{id}(t)) \quad (1)$$

$$x_{id}(t+1) = x_{id}(t) + v_{id}(t+1) \quad (2)$$

For which, the d is the d^{th} dimension, where $d \in D$. The parameter w is the inertia weight of a particle that tunes the balance between the cognitive and the social components, r_1 and r_2 are two random numbers in the range from 0 to 1. c_1 and c_2 are the acceleration constants, which control the influence of the personal and global best particles. The p_{id} is the personal best position of a particle (*pbest*), and g_d is the best global particle (*gbest*) [18].

Algorithm 1 present the implementation of a typical PSO algorithm.

Algorithm 1 Basic PSO Algorithm

```
1:  procedure PSO Algorithm
2:      Initialize the population POP
3:      for each particle do
4:          Initialize the particle
5:      end for
6:      while maximum iterations is not satisfied do
7:          for each particle do
8:              Evaluate the fitness value (v)
9:              if v is better than the best fitness value (pBest) then
10:                 set v as the new pBest
11:             end if
12:             Choose the particle with pBest and set it as gBest
13:         end for
14:         for each particle do
15:             Calculate particle velocity as in equation 14
16:             Update particle position as in equation 15
17:         end for
18:         Increment the loop counter
19:     end while
20: end procedure
```

In essence, the classical PSO algorithm developed to deal with continuous problems that have continuous variables [50]. However, handling problems of discrete search spaces demands transforming the optimization algorithm into a discrete problem. The PSO algorithm can be considered as a binary search algorithm when it represents the values of the positions of the particles as binary values, while the velocities of the particles represent the probabilities of a position to have a value of 1. The earlier implementation to transform the PSO algorithm into binary algorithm was the in [51], which utilized a kind of transfer function that is known by the Sigmoid function. The Sigmoid function (S1) is represented in Equation 3, in which the $v_{id}(t)$ is the velocity of particle i at dimension d and iteration t.

$$T(v_{id}(t)) = \frac{1}{1 + e^{-2 * v_{id}(t)}} \quad (3)$$

If a randomly generated number is greater than $T(v_{id}(t))$, the position holds a value 1, and 0 otherwise (as shown in Equation 4).

$$x_{id}(t+1) = \begin{cases} 1 & \text{if } rand \geq T(v_{id}(t)) \\ 0 & \text{if } rand < T(v_{id}(t)) \end{cases} \quad (4)$$

The analysis of the time complexity of the PSO algorithm is based on Algorithm 1. The time complexity for lines 2-5 to initialize N particles with D number of dimensions equals

TABLE 1. Summary and comparison among different Ransomware detection systems.

Related work	Type	Feature set	ML approach[1]	Accuracy	Apps' source	Dataset size[2]	Balanced
[38]	Dynamic	User Interface and info entropy of files	Not stated	99%	HelDroid [39] and own collected samples	9,238 B 2,721 R	No
[37]	Static/ Dynamic	Threatening text/image	Not stated	91%	ID-Rans [40],thezoo [41],Contagio [42]	200 B 100 R	No
[24]	Static	API calls frequencies	RF, DT, SMO and NB	97%	Koodous [43], RProber [38], and VirusTotal [44]	500 B 500 R	Yes
[25]	Static	A dictionary contains unique opcodes present	RF, SVM, KNN and ANN	99.80%	AMD dataset [45]	15126 B 2418 R	No
[26]	Static	Text and background operations [admin, window, system, priority, permissions]	SVM, RF, DT and LR	99.98%	ransomware-transaction QQ groups [46]	15751 B 301 R	No
[27]	Static	48 Permissions, 4 Intents and 34 APIs	KNN,LR,SVM, XGBoost, and RF	97.62%	Contagio [42], Kharon [47]	more than 200 B 259 R	No
[28]	Static	String, opcode, API,permission, and envi. features	Multimodal deep learning,SVM, and RF	98%	VirusTotal [44]	19,747 B 13,075 M	No
[29]	Static	Sequence of opcodes	proposed deep learning model	91.91%	AMD dataset [45]	25,000 B 24,650 M	Yes
[30]	Static	API packages, classes and methods	RF	97%	HelDroid [39], VirusTotal [44]	18396 B 3017 R	No
[36]	Dynamic	Network packets and headers	LSTM	97.08%	CI-CAndMal2017 dataset [48]	1048574 B 460976 M	No
[31]	Static	Opcodes, permissions and intents	RF, SVM, KNN and LR	93.92%	CIC_ InvesAndMal2019 [49]	1147 B 905 M	No
[33]	Static	Permission requests and API calls	RF, J48, RT, KNN and NB	F-measure of 94.3%	AndroZoo, Contagio [42],MalShare,VShare and VirusTotal [44]	14,172 B 13,719 M	No
[4]	Static	Permissions	N/A	N/A	RansomProber [38] dataset	2,050 B 2,050 R	Yes
[32]	Static	Permissions	RF,J48, SMO, and NB	96.9%	Alsoghyer & Almomani [24]	500 B 500 R	Yes
[10]	Dynamic	System calls	RF, J48, and NB	98.31%	VirusTotal [44]	400 B 400 R	Yes
[34]	Static	Permissions and API calls	SSA,PSO,NB,J48, RF,Adaboost, Bagging and XGBoost.	98%	Alsoghyer & Almomani [24]	500 B 500 R	Yes

NB:Naïve Bayes, RF:Random Forest, DT:Decision Tree, SMO:Sequential Minimal Optimization, SVM:Support Vector Machine, KNN:K-Nearest Neighbors, ANN:Artificial Neural Network, LR:Logistic Regression, LSTM:Long Short-Term Memory, RT:Random Tree, J48:Decision Tree C4.5,SSA:Salp Swarm Algorithm, PSO:Particle Swarm Optimization, B: Benign, R: Ransomware, M: Malware

$O(N \times D)$. The loops in lines 7-13 and 14-17 have the time complexity of $O(N \times D)$ for each iteration in I. Line 18 is of constant time. Thus, the time complexity of the binary PSO is $O(N \times D \times I)$ [52].

B. SUPPORT VECTOR MACHINES

The support vector machines algorithm is a statistical and supervised machine learning algorithm that is attributed to Vapnik who coined it in 1992 [19] by the research community. Originally, it was developed to address binary classification problems, where it uses support vectors and hyperplanes to create decision boundaries that separate the classes by maximizing the marginal distance between them. As it is known by the large margin classifier, the objective of SVM is to maximize the distance between the data points of the different classes. The maximum distance between the data points (support vectors) from the two classes is known by the margin. Whilst, the support vectors are the points that are closer to the decision boundaries and used to distinguish the classes.

Loosely speaking, the SVM has been adopted into various real-world problems [53]–[55], however, not all problems are linearly separable. Though, the SVM algorithm maps the data into higher-dimensional spaces (the feature space) in order to make such non-linear input data linearly separable. As the SVM algorithm can handle the linear and non-linear problems; having a linearly-separable dataset can be handled by a linear hyperplane that is formulated as in Equation 5, where w is a weight parameter, and b is a threshold value between the hyperplane and the origin plane.

$$f(x) : w \cdot x + b = 0 \tag{5}$$

The objective of SVM is to maximize the marginal-distance by minimizing an objective function which represents the

cost. Equation 6 describes the objective of the SVM. In which, C is a regularization parameter, A is the objective function that is formulated in Equations 7, and B represents the regularization function as in Equation 8, where n is the number of features, and m in Eq. 7 is the number of training instances.

$$Min\ C \cdot A + B \tag{6}$$

$$A = \sum_{i=1}^{m} \left[y^{(i)} cost_1(\theta^T x^{(i)}) + (1 - y^{(i)}) cost_0(\theta^T x^{(i)}) \right] \tag{7}$$

$$B = \frac{1}{2} \sum_{j=1}^{n} \theta_j^2 \tag{8}$$

Even that SVM had shown very successful results in classification over the literature [56], but its performance is highly sensitive to various hyperparameters. Mainly, the cost (C) which is the regularization parameter, and the (γ) parameter in the case of a non-linear SVM. Since the improper settings of the C and γ degrades the generalization power of the algorithm, tuning them is a critical issue while training an SVM algorithm. In other words, initializing the C parameter with a large value; results in a low bias and high variance, which leads to overfitting. Meanwhile, initializing the C parameter with a small value; increases the bias and minimizes the variance, which is prone to underfitting. Therefore, optimizing the hyperparameters of C and γ has a significant influence on the performance of the SVM algorithm.

The time complexity of the SVM algorithm is $O(max(n, m) \times min(n, m)^2)$ [57] where n is the number of points and m is the number of dimensions.

C. OVERSAMPLING TECHNIQUES

Until recently, the problem of imbalanced data has been studied widely and several approaches have been proposed to address it, including data-oriented, and algorithmic-oriented methods. Various approaches of algorithmic-oriented methods have been developed, such as SMOTE [20], SVM-SMOTE [58], borderline-SMOTE [21], and ADASYN [22].

1) SMOTE

The synthetic minority over-sampling technique is an approach to overcome the imbalanced data problem by creating artificial data points that are analogous to the real one. SMOTE is one of the early proposed techniques, which is developed by Chawla in 2002, and used over various domains, such as the medical [59], the industrial [60], and others [61].

Assuming an imbalanced dataset d with n examples, where, C is a major class, and c is a minor class. Primarily, SMOTE balances the distribution of classes by increasing the ratio of the minority examples by synthesizing new interpolated examples from the existing data. Iteratively, for each data example i_c of the minority class, SMOTE selects a k nearest points from the minor class to the example i_c. For which, the shortest path between any two points is calculated based on the euclidean distance. Relying on a sampling rate defined based on the imbalanced ratio, a number of the k nearest neighbors is randomly chosen. Thus, each of the nominated nearest examples creates a line that links it with the example i_c. Subsequently, from every formed link, a random example is picked to be the new synthetic point.

However, a serious drawback of SMOTE is that the synthesized minority examples do not consider the majority classes during the oversampling process, especially, when there is a strong correlation between them [20].

2) BORDERLINE-SMOTE

Borderline-SMOTE is an evolved version of SMOTE, which is proposed in 2005 [21] as an extension minority-based oversampling method. The comparison of Borderline-SMOTE with its precedent (SMOTE) shows more powerful performance at different learning approaches [62], [63]. Two variants of borderline-SMOTE were proposed to overcome the original difficulty of the imbalanced data problem of identifying the correct border that distinguishes two classes. The two variants are the borderline-SMOTE1 and borderline-SMOTE2.

Borderline-SMOTE performs two operations: first, classifies the examples into three types of regions (i.e., safe, danger, and noise) to recognize the borderline examples. Second, synthesizes new examples. In Borderline-SMOTE1, the new data examples merely created from the recognized borderline data points. For a given data point i from the minority class, a k number of nearest neighbors is selected regardless of whether the neighbors are from the major or the minor class. The ratio of the neighbors that belongs to the majority class decides to which region the point i should be classified. If all neighbors belong to the majority class, then the point i is classified into the noise region. If the number of the majority examples of the neighbors is greater than $a_c/2$, where a_c is the number of the minority examples, then i belongs to the danger region. Otherwise, when the number of the majority examples is less than $a_c/2$, then i is in the safe region. Upon this, all minority examples that are classified into the danger region form the borderline examples.

To synthesize new examples; for each point in the danger region i_{danger}, a k number of nearest neighbors which belongs to the minor class is selected. From which, an s random examples were picked. The difference between all points in s and the corresponding point in the danger region is computed and denoted by $diff_j$. Hence, the new point is synthesized based on Equation 9, where r_j is a random number in the range from 1 to s.

$$i\_new = i_{danger} r_j \times diff_j \tag{9}$$

In Borderline-SMOTE2, the neighbors of the points that are in the danger region are considered from the major and minor classes and not just from the minor class as in the first version (Borderline-SMOTE1).

Further, SVM-SMOTE is a variant of SMOTE that is proposed by Hien in [58]. SVM-SMOTE balances the class distribution by oversampling the minority class instances which

are located at the borderline between the two classes. The SVM algorithm can ameliorate the performance by identifying the separating decision boundary and then predict new minority instances.

3) ADAPTIVE SYNTHETIC SAMPLING

The ADASYN method was proposed to reduce the learning bias by adaptively adjusting the decision boundary of the minority data points that are hard to learn. Considering the major class C with a number of instances a (a^C), and the minor class c, with the number of instances b (b^c). The ratio of the minority points to the majority points is used to compute the number of synthetic points S for the minority class c, as in Eq. 10.

$$S = (b^c - a^C) \times \beta, \quad \beta \in [0, 1] \tag{10}$$

For each minority point, a number of k of neighboring points is selected which is known by the neighboring set. Each minority point has a density distribution based on its neighbors of the majority class from the neighboring set. Identifying the density distribution of the points assists in computing the number of required synthetic data points, as described by Equation 11. Where the density distribution (r_i') is a normalized version of $\frac{a^C}{k}$.

$$g_i = r_i' \times S, \ where \ r_i' = \frac{a^C}{k} \tag{11}$$

For each minority point x_i, a randomly-selected minority point from the neighboring set is selected x_j to create the synthetic data g_i. The data created based on Equation 12, where $diff_j$ is the difference between the points x_i and x_j, while the *lambda* is a random number in the range from 0 to 1.

$$x_{new} = x_i + diff_j \times \lambda \tag{12}$$

IV. IMBALANCED DATASET CREATION

The dataset used in this research was built based on 10,153 Android application samples (500 malicious (Ransomware) applications and 9653 benign applications). This dataset is publicly accessible on the Security Engineering Lab (SEL) website.[2] The process began by collecting the applications (benign and ransomware) from different market stores. After that, the collected files were decompiled to extract the original code. Finally, the code parsing phase was applied and then resulted in creating the imbalanced dataset. Figure 1 displays the process of creating the dataset.

A. DATASET COLLECTION PHASE

In this paper, four data sources have been used to collect Android applications. The Android applications were downloaded as Android Packages (APK). The APK file is a zipped file that contains all the application's code including assets, resources and the manifest file. The APK files of Ransomware samples were collected from several repositories

[2]https://sel.psu.edu.sa/Research/datasets/2020_RansIm-DS.php

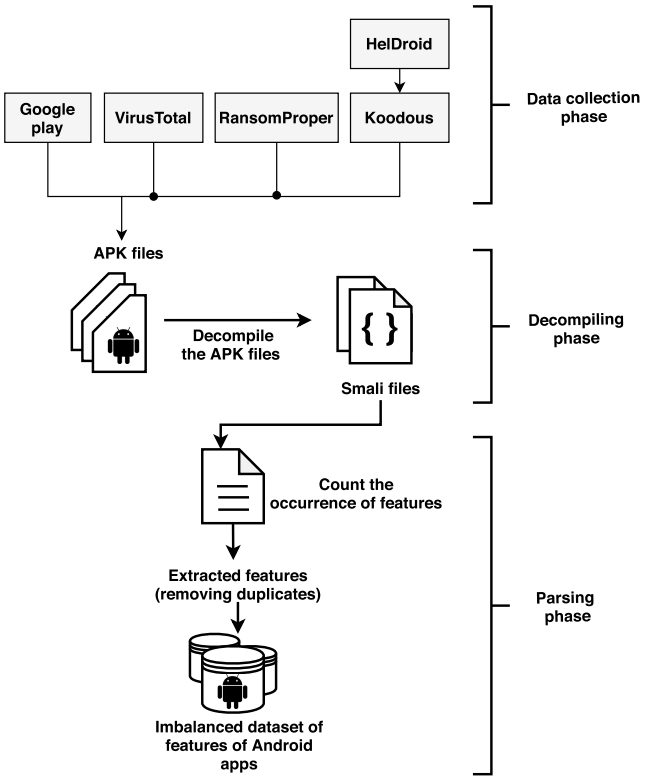

FIGURE 1. The process of creating the imbalanced dataset.

including Koodous, RansomProper Project and VirusTotal. Whereas, the APK files of benign samples were downloaded from Google Play store.

RansomProper Project: Categorizes the Ransomware apps chronologically based on their malicious features. The dataset of RansomProper Project contains of 2,721 ransomware applications that represent fifteen different families [38]. Furthermore, the project provides a real-time scanning by executing the Ransomware application's user interface (UI) widgets.

Koodous: Is an Android malware platform that provides online analysis services and a large-scale repository for Android malicious apps with over 65 million APKs [64]. To collect Ransomware families from Koodous database, the hashes of the APK files were captured from HelDroid project [24]. Afterward, the APK files were retrieved using the searching services of koodous platform. Further, additional Ransomware samples were captured by searching the tag *(tag:ransomware)*. Then, they have been tested on VirusTotal's scanning tools to ensure they are identified as Ransomware.

VirusTotal: Is an online malware scanning engine which uses over 70 security vendors to conduct malicious detection services [65]. In addition, it provides an API service to upload and scan APK files without the need for the web-interface. However, VirusTotal lacks of the presence of labelling standardization, which makes it challenging to precisely classify a Ransomware family [66]. Consequently, to overcome this

challenge, the obtained samples from the VirusTotal database have been filtered and labeled manually to accurately build the Ransomware database.

Google Play: Is the official Android market store which can be accessed using a browser or the Play Store of Android applications. Google Play developed a Bouncer that is a security mechanism aims to automatically detect malware applications [67]. Furthermore, Google Play introduced a security event called *Google Play Protect* which provides continuous running scans ensuring the harmless of the installed applications [68]. Based on the aforementioned mechanisms applied by Google play, it has been chosen as the main source of benign apps of this study. The total number of downloaded apps is 12,036 applications that are chosen from the top-ranking applications of main categories in Google Play store. These main categories are Kids (41%), Game (28%), and others (31%).

B. DECOMPILING PHASE

In this phase, the executable code of the APK file is converted to an intermediate readable language [8]. The main file to be extracted is the *AndroidManifest.xml* file which contains the application's metadata and all the defined permissions by the applications. This file can be extracted using APKtool[3] that decompiles the zipped APK file to the manifest file and.smali files. Each smali file represents a defined class in the original code of the application. All the required features of the application can be extracted by parsing the manifest file and theses smali files.

C. PARSING PHASE

Constructing a well-designed features set has a major influence on the efficiency of malware detection. The used features set of this paper includes 389 features; 228 API packages, and 161 permissions belong to the most recent release of Android (API level 30). The occurrences of these features have been counted on the collected samples using a Python script by extracting the features from the.smali files and the manifest file. Algorithm 2 illustrates the parsing procedure. The Parsing algorithm scans all the applications' folders allocated in the *Root Directory*. For each application, the scanning is performed in two stages, first is parsing the manifest file to count the defined permissions, and second is parsing the.smali files to count the used API packages. Finally, the overall parsed features is stored in the database.

The extracted features of the parsed applications have to be serialized for further analysis. The representation of the serialized features can be implemented using the $U-dimensional$ format where U represents the feature set [69]. As shown in Figure 5, all the features under investigation (F) are stored in one list of length $|F|$. Then each application's decompiled APK file is represented as a record of length $|F|$. The entries of these records are filled with a value of 0 if the application never uses these features. Otherwise, the Parsing algorithm fills the application's record with c, where c is the total count of the usage of a specific feature [70].

Finally, different apps with same features (16% of the total apps) were filtered and removed to avoid any duplication in the dataset. Figure 3 shows the number of removed apps.

[3]https://ibotpeaches.github.io/Apktool/

Algorithm 2 Parsing Algorithm

```
procedure Parsing Algorithm
    Input: Android applications
    Output: Feature parsing records
Initialisation:
    Initialize the API_packages set
    Initialize the Permissions set
    Initialize the Root_Directory
    for each application_folder in Root_Directory do
        get Manifest_File
        for each element in Permissions set do
            Find_Occurrence (Manifest_File )
            permissions_count ← Total occurring
        end for
        for each smali_file in smali_Directory do
            for each element in API_packages set do
                Find_Occurrence (.smali file )
                API_Packages_count ← Total occurring
            end for
        end for
        Total ←permissions_count+ API_Packages_count
        Store Total number of features in the database
    end for
end procedure
```

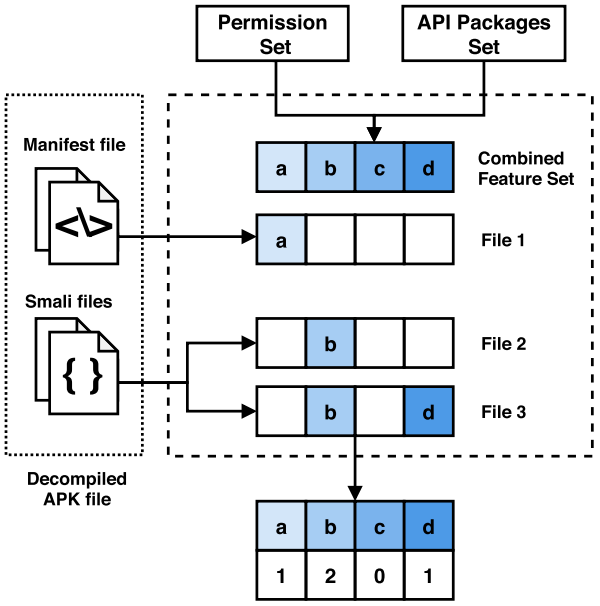

FIGURE 2. The feature serializing model.

V. PROPOSED CLASSIFICATION APPROACH

The ransomware detection process of the proposed approach is discussed in detail in this section. The proposed approach is based on a hybrid evolutionary process of optimizing an imbalanced data, an oversampling technique, and a classification technique.

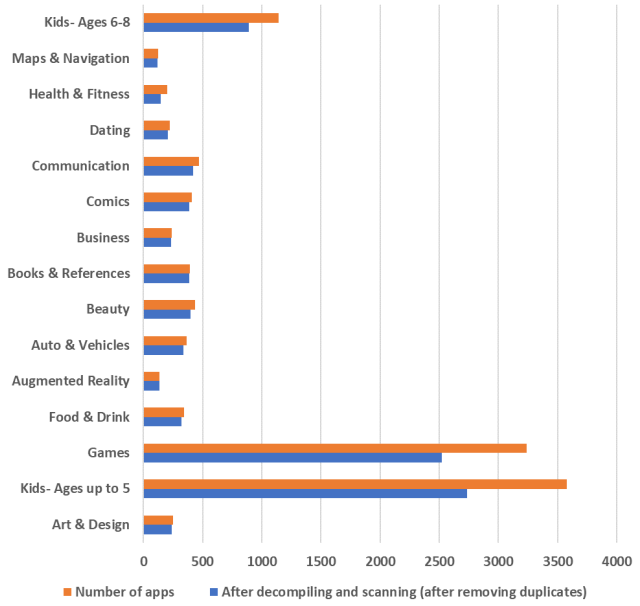

FIGURE 3. The total apps and the apps after removing duplicates.

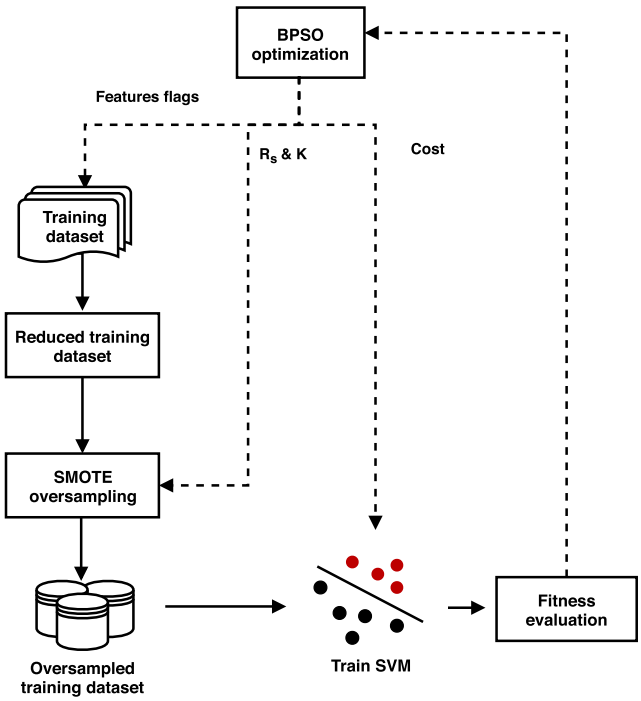

FIGURE 4. High-level description of the flow of the proposed approach.

Figure 4 illustrates the proposed approach, which we refer to as ovr-*t*BPSO-SVM. The approach includes iterative optimization of the classification process and an evaluation process of the classification by a binary PSO optimization algorithm. The main components of the proposed approach as described in the figure are mainly the feature selection, the oversampling technique, the classification technique, the fitness evaluation, and the optimization technique.

In the feature selection component, not all features are considered for classification, but rather, a selected set of features which are representing a reduced version of the training dataset. Hence, features are selected according to the flags generated by the optimization process based on a transfer function, which results in a reduced training dataset. This dataset reduces the computational cost and enhances the performance of the classification. Regarding the oversampling technique, the Ransomware class is enlarged by the SMOTE oversampling techniques to balance the dataset for better evaluation of the performance. Thereby, an oversampled training dataset is generated from this process. For the classification technique, the SVM algorithm is used to generate a classification model that correctly identifies Ransomware instances from benign instances. Moreover, it is also clear from the figure that the evolutionary algorithm optimizes the k-neighbours and the sampling ratio parameters of the oversampling technique. In addition, the cost parameter of the SVM algorithm is further boosted by the evolutionary algorithm to find an optimal fit of the model to the dataset while avoiding overfitting. Finally, the generated model by the optimization process is evaluated by the fitness function for further enhancements of later iterations. The objective function of the proposed algorithm is assessed by the fitness evaluation criterion. For which, the classification model is evaluated using the geometric mean score to promote the optimization process at later iterations. Finally, the optimization technique is implemented by the binary PSO algorithm that is used to optimize the selection of features, the parameters of the oversampling technique, and the cost coefficient of the SVM algorithm.

A. STRUCTURE OF THE PARTICLE

As primarily the PSO algorithm developed to deal with continuous search spaces, adopting it to address discrete search problems requires to transform its search space from continuous into discrete (binary). This is implemented by utilizing the (S2) transfer function. S2 is a sigmoidal function that is given by Equation 13, which integrated to produce particles represented by binary vectors.

$$S(x) = \frac{1}{1 + e^{-x}} \tag{13}$$

At each iteration, several particles are generated by the proposed approach presenting candidate solutions to the ransomware classification problem. Each particle consists of three parts of binary values which are illustrated in Figure 5. These parts reflect the following tasks:

- Feature selection: a set of features is selected from the dataset for training and testing the model. This set is selected according to a list of n binary values represented by the first part of the particle. The list can be represented by Equation 14.

$$F = [f_1, f_2 \ldots f_n] \tag{14}$$

  Each binary value in the list reflects the presence or absence of the corresponding feature where a value of 0 represents the absence of the feature, while the value of 1

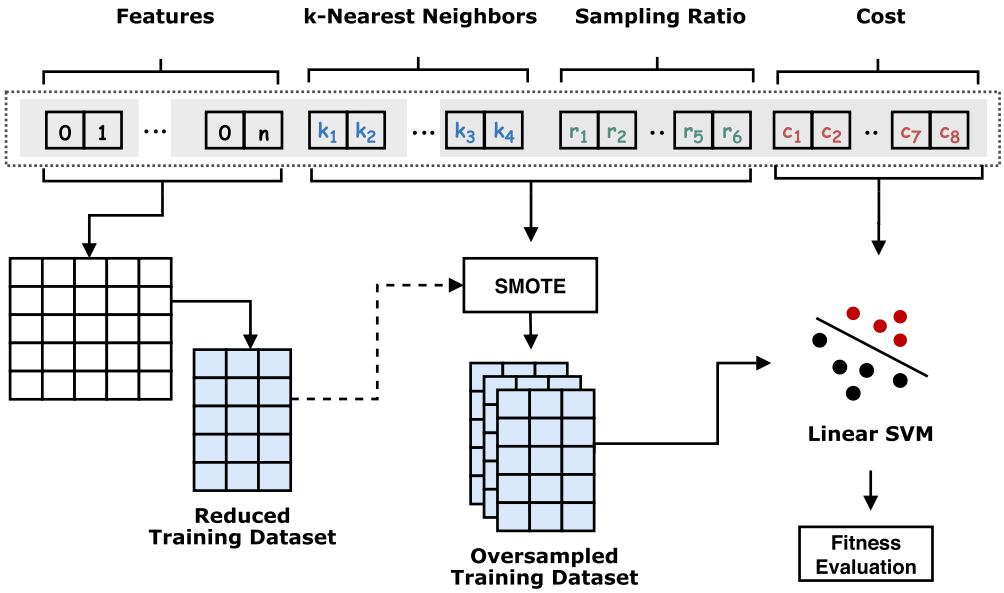

FIGURE 5. An illustration of the adopted structure of the particle in the proposed classification approach, where R_S is the ratio of oversampling, and K is the number of nearest neighbors.

represents the presence of the feature. Thus, the number of selected features for a particle is the number of the 1s of the first part of the particle.

- SMOTE oversampling technique parameters optimization: the number of neighbours and the oversampling ratio for the SMOTE oversampling technique are represented by the second part of the particle. The number of neighbours is used to synthesise new instances of the Ransomware class by considering the neighbours of the class. It is represented by four binary digits of a decimal value between 0 and 15 reflecting a binary range of numbers between 0000 to 1111, respectively. The list of digits is represented as in Equation 15.

$$K = [k_1, k_2, k_3, k_4] \tag{15}$$

On the other hand, the oversampling ratio represents the number of oversampling instances of the Ransomware class divided by the number of instances of the benign class. It is represented by six binary digits of a float value between 0.0 and 0.63 reflecting a binary range of numbers between 000000 to 111111, respectively. The list of digits is represented by Equation 16.

$$r = [r_1, r_2, r_3, r_4, r_5, r_6] \tag{16}$$

- The optimization of the cost parameter of the linear SVM: the cost parameter value of the linear SVM classification technique is represented by the third part of the particle. This reflects the extent to which the model is fitting the training data where a low value of the cost parameter indicates a more fitted model and a small number of misclassifications on the training instances, while a high value indicates a less fitted model and a large number of misclassifications of the training instances. A trade-off between fitting the model on the training data and fitting it on possible upcoming unseen data must be taken into consideration when choosing the value of the cost parameter to avoid overfitting. The cost parameter is represented by eight binary digits of a float value between 0.0 and 2.55 referring to a binary range of numbers between 00000000 to 11111111. The list of digits is represented as in Equation 17.

$$C = [C_1, C_2, C_3, C_4, C_5, C_6, C_7, C_8] \tag{17}$$

B. FITNESS FUNCTION (INTERNAL EVALUATION)

Each particle in the proposed approach is evaluated internally by a fitness function to measure its quality in leading the algorithm toward optimality and avoiding the stagnation in local regions. The utilized fitness function is a single-objective optimization problem that aims to maximize the performance by minimizing the fitness. Essentially, the internal evaluation of the candidate particles determine the global best and the local best at each iteration. Hence, this guides the search process of the algorithm toward the optimal regions of the search space gradually over the course of iterations. The fitness is given by Equation 18, which is the geometric-mean (g-mean) of classification subtracted from 1. The g-mean measure is the square root of the product of class-wise sensitivity, which equals the square root of the multiplication of sensitivity and specificity in the case of binary classification. The sensitivity and specificity are based on the confusion matrix, where the sensitivity is defined by $TP/(TP+FN)$, and the specificity by $TN/(TN+FP)$. The TP is the true-positive, the TN is the true-negative, the FP is the false-positive, and the FN represents the false-negative.

$$Fitness = 1 - \sqrt{sensitivity \times specificity} \tag{18}$$

C. PROCEDURE OF THE ALGORITHM

The procedure of the proposed ovr-*t*BPSO-SVM can be described by the following steps:

1) Initialization: A set of particles are formed for the first iteration of the PSO optimization technique, where each particle includes binary values of the presence or absence of features, k, sampling ratio, and the cost as observed by Figure 5 and discussed in Section V-A.
2) Update: Through the course of iterations, particles are evolved to explore better set of solutions for the Ransomware data, which is further detailed by Section III-A.
3) Particles mapping: As discussed in Section V-A, a set of values are extracted from each particle and are used to select features on one hand, and as an input to both the oversampling and classification techniques on the other hand.
4) Fitness evaluation: The g-mean score is used to assess the evolved particles of each iteration according to Equation 18.
5) End of procedure: The procedure terminates when a predefined value of iterations is reached. As a result, the fittest particle is retrieved having a combination of selected features and optimized values of k, sampling ratio, and cost parameters.
6) Testing: The testing instances are used to evaluate the model generated from ovr-*t*BPSO-SVM algorithm using several evaluation techniques including the sensitivity, specificity, and g-mean, which are further discussed in the next section.

Figure 6 illustrates the process by which ovr-*t*BPSO-SVM follows. It starts by dividing the dataset into training and testing parts. Both parts are reduced by the selected features but only the training part is oversampled and used to generate a prediction model with the enhanced values of parameters. In contrast, the testing part of reduced features is used to evaluate the optimized model.

VI. MODEL EVALUATION METRICS

Various evaluation measures were utilized to assess the performance of the proposed methodology for ransomware prediction. The evaluation metrics are derived from the confusion matrix, which are the true positive rate (sensitivity), the true negative rate (specificity), and the g-mean. The confusion matrix is represented in Table 2, where the positive class corresponds to the ransomware, while the negative class is the normal class. Hence, the *TP* represents the examples that are actually ransomware and they are predicted ransomware. The *TN* indicates the examples that are predicted normal and they are actually normal. Whereas the *FP*, represents the instances that are predicted ransomware but they are actually normal, and the *FN* are the instances that are predicted normal but they are ransomware.

The sensitivity also is known by the true positive rate, the recall, and the likelihood of prediction rate.

TABLE 2. Confusion matrix.

	Actual	
	Ransomware	Non-Ransomware
Predicted Ransomware	**TP**	**FP**
Predicted Non-Ransomware	**FN**	**TN**

The sensitivity reflects the capacity of the classifier in recognizing the ransomware instances. It is computed as the ratio of the recognized ransomware over the true ransomware instances, as defined in Equation 19.

$$sensitivity = \frac{TP}{TP + FN} \tag{19}$$

The specificity also known as the true negative rate. It represents the ability of the classification algorithm to recognize the negative instances of data (Non-Ransom), which is the ratio of the identified true negatives over the overall number of actual non-Ransomware instances. It is described as in (Eq. 20).

$$specificity = \frac{TN}{TN + FP} \tag{20}$$

The g-mean metric corresponds to the ratio of balance of classification performances from the ransomware and normal classes. Mathematically, the g-mean is calculated as the square root of the multiplication of the recall of both classes; the ransomware and the normal, as illustrated in Equation 21.

$$G - mean = \sqrt{\frac{TP}{TP + FN} \times \frac{TN}{TN + FP}} \tag{21}$$

VII. EXPERIMENTS AND RESULTS

A. ENVIRONMENTAL AND PARAMETERS' SETTINGS

All experiments were conducted on a workstation with the following specifications: Intel core i7-1065G7 CPU and 1.30GHz / 16 GB RAM. For experiments Python 3.8 is used. Scikit Learn [71] and imbalanced-learn [72] Python libraries are used for the oversampling techniques, classification techniques and the evaluation of the algorithm.

The experiments are repeated 30 times with different values of the population size and number of iterations which have the values of 25, 50, and 100 for both the population size and the number of iterations. The values of 6, 0.9, 0.6, 2, and 2 are determined for PSO parameters for the Vmax, wMax, wMin, c1, and c2, respectively [73]. The parameters settings are illustrated in Table 3. For the parameters of the oversampling methods they are automatically tuned using the PSO algorithm.

B. EFFECT OF OVERSAMPLING TYPE AND PARAMETERS TUNING

In this experiment, the effect of the oversampling technique on the performance of the proposed approach is investigated for Ransomware detection in the context of imbalanced data distribution. For this, five SMOTE variants were experimented; the classic SMOTE, ADASYN, SVM-SMOTE,

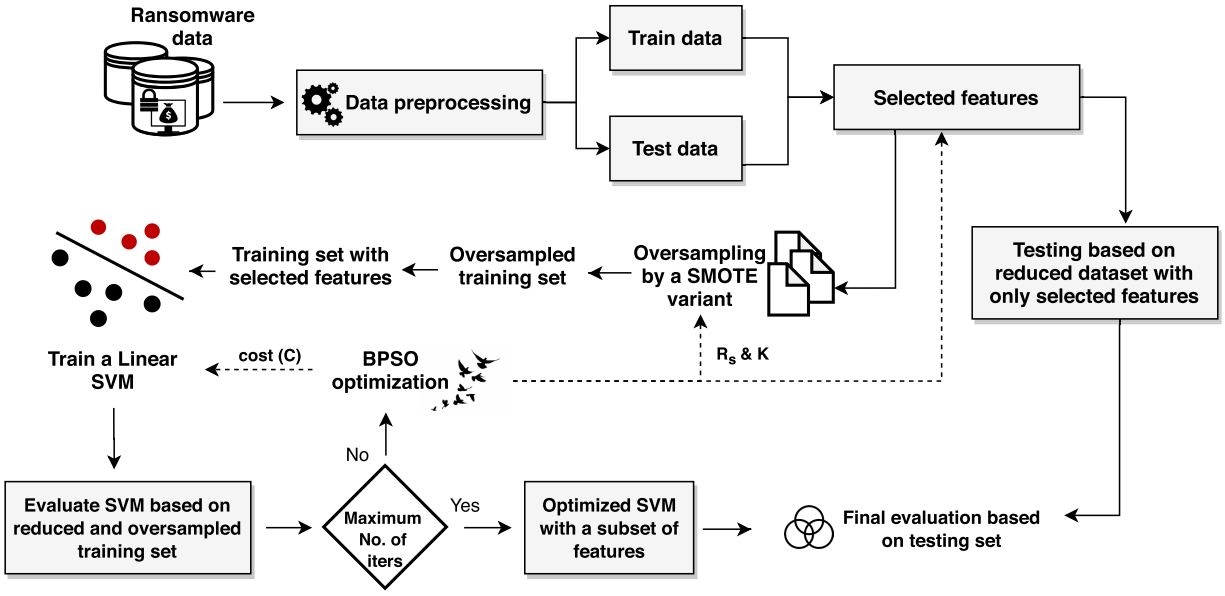

FIGURE 6. Flowchart describes the procedure of the proposed ovr-*t*BPSO-SVM.

TABLE 3. Parameters settings.

Parameter	Value
Runs	30
Population size	25, 50, and 100
Iterations	25, 50, and 100
Vmax	6
wMax	0.9
wMin	0.6
c1	2
c2	2

Borderline-SMOTE1, and Borderline-SMOTE2. Each technique was embedded in the proposed approach and experimented under different settings of the swam size, and number of iterations. The results of this experiment in terms of sensitivity, specificity, and g-mean are shown in Table 4. The table also presents the best values obtained for the cost parameter of the SVM and the minimum number of selected features. It is clear from the table that SMOTE obtained the best results in comparison to the other oversampling methods. For instance, when the population size is 50 and the number of iteration is 50, too, SMOTE gained the best sensitivity, specificity, g-mean, and the least number of features by having 96.4%, 98.7%, 97.5%, and 182, respectively. Yet, it achieved the best specificity among all when the population size was 25, and 100, gaining 98.3%, and 98.4%, respectively. However, it is noticeable that Borderline-SMOTE2 can achieve outperforming results in terms of sensitivity and g-mean when the population size is 25 by holding 97%, and in terms of sensitivity when the population size is 100, by having a percentage of 95.8%. Additionally, ADASYN accomplished the highest g-mean when the population size is 100 (96.3%), and the minimum number of features (192) at a population size of 25. Overall, the SMOTE method outperformed them mainly when the population size is 50, and the number of iterations is 50, where the best cost parameter also was 1.18.

Furthermore, Figures 7-9 show the convergence curves of the PSO algorithm over different settings of the population size, and the number of iterations. The convergence curves present the capacity of the algorithm to reach the optimal value of the objective (fitness) function during the training period and over the course of iterations. Hence, the curves were depicted with the fitness projected on the y-axis, and the number of iterations on the x-axis. Figure 7 shows the convergence when the number of iterations and the population size are 25. It is clear from the figure as the proposed algorithm minimizes the fitness, that the oversampling methods converge steadily over the iterations, where markedly, the Borderline-SMOTE1 exhibits the smoothest convergence toward the optimal value. Whereas, the SVM-SMOTE demonstrated the poorest convergence toward the optimal.

Regarding Figure 8, it shows the convergence when the population size, and the number of iterations are 50. All of the methods converge gradually up to the 30^{th} iteration where they leveled out. Clearly, after the 30^{th} iteration, the Borderline-SMOTE1 obtained the minimum fitness, which is approximately 0.01. Even though, it is clear noticing that SMOTE achieved slightly better than Borderline-SMOTE2, ADASYN, and SVM-SMOTE, which obtained nearly 0.018 of fitness after the 27^{th} iteration.

Further, Figure 9 explains the convergence when the population size, and the number of iterations are 100. From the figure, it is observed that Borderline-SMOTE1, Borderline-SMOTE2, and ADASYN demonstrated a premature convergence where they had fallen down in a local optimal

TABLE 4. Results of the proposed approach with a comparison of different types oversampling techniques at different values of the swarm size and the number of iterations. The values marked by the **boldline** style are indicating the best results.

	Pop size	Iters	Sensitivity	Specificity	G-mean	Cost	No. of features
ADASYN	25	25	0.933	0.975	0.954	0.62	**192**
	50	50	0.933	0.981	0.957	1.44	198
	100	100	0.945	0.980	**0.963**	2.05	205
SMOTE	25	25	0.939	**0.983**	0.961	0.76	207
	50	50	**0.964**	**0.987**	**0.975**	1.18	**182**
	100	100	0.915	**0.984**	0.949	0.97	211
SVM-SMOTE	25	25	0.952	0.976	0.964	0.65	208
	50	50	0.945	0.981	0.963	0.66	194
	100	100	0.939	0.974	0.957	1.07	220
Borderline1	25	25	0.958	0.960	0.959	0.45	201
	50	50	0.933	0.986	0.959	2.51	219
	100	100	0.933	0.977	0.955	1.25	**187**
Borderline2	25	25	**0.970**	0.970	**0.970**	2.55	197
	50	50	0.952	0.959	0.955	1.55	226
	100	100	**0.958**	0.964	0.961	1.02	198

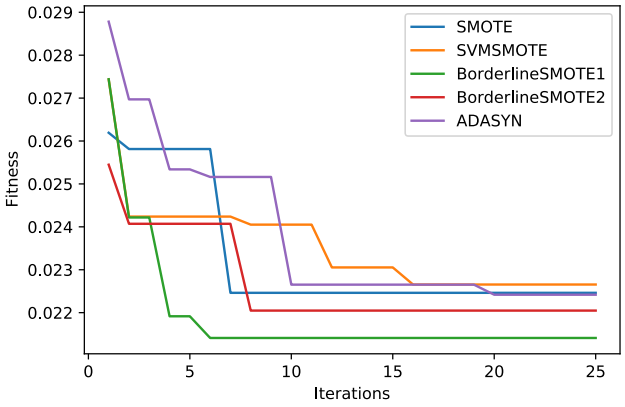

FIGURE 7. Convergence of different oversampling techniques for 25 iterations with a population size value of 25.

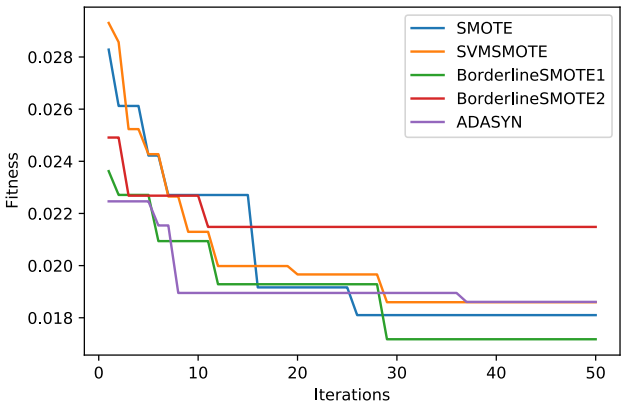

FIGURE 8. Convergence of different oversampling techniques for 50 iterations with a population size value of 50.

early in the iterations then remained constant. For instance, Borderline-SMOTE1 reached a minimal at the 17^{th} iteration, then could not converge any further. Similarly is for the Borderline-SMOTE2 that converged to a local at the 20^{th} iteration, while ADASYN converged at the 37^{th} iteration. On the contrast, SMOTE and SVM-SMOTE showed a better convergence over the iterations, where both of them dropped to an optimal after the 60^{th} iteration achieving a fitness value less than 0.018.

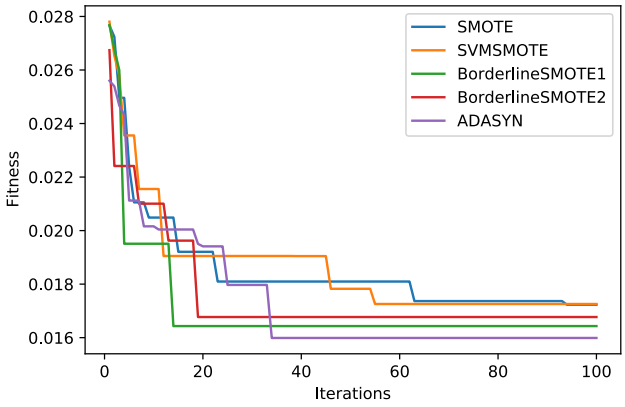

FIGURE 9. Convergence of different oversampling techniques for 100 iterations with a population size value of 100.

To sum up, the SMOTE oversampling showed the best soft and steady convergence when the number of iterations and the population size were 50. Therefore, it is considered the baseline in the subsequent experiments.

C. COMPARISON WITH STANDARD ALGORITHMS

This subsection presents the experimental results of the proposed ovr-tBPSO-SVM approach when compared with standard and well-known classifiers. The best version of ovr-tBPSO-SVM which is the SMOTE-tBPSO-SVM will be used in the following comparison. The classifiers are including the k-nearest neighbours (k-NN), Naïve Bayes (NB), MultiLayer Perceptron (MLP), and Random Forests (RF). k-

TABLE 5. Comparison with standard classifiers.

	Specificity	Sensitivity	G-mean
1-NN	0.997	0.912	0.953
3-NN	0.997	0.882	0.938
5-NN	0.997	0.871	0.931
NB	0.95	0.935	0.943
MLP	**0.999**	0.894	0.945
RF	0.998	0.871	0.932
SMOTE-*t*BPSO-SVM	0.987	**0.964**	**0.975**

TABLE 6. Comparison with other feature selection metaheuristics.

	Specificity	Sensitivity	G-mean
SMOTE-*t*BFFA-SVM	0.986	0.939	0.962
SMOTE-*t*BMVO-SVM	0.983	0.927	0.955
SMOTE-*t*BGWO-SVM	0.976	0.952	0.964
SMOTE-*t*BPSO-SVM	**0.987**	**0.964**	**0.975**

NN was applied with different number of neighbours (i.e., 1, 3 and 5) and denoted as 1-NN, 3-NN, and 5-NN, respectively. For the MLP algorithm, the number of hidden neurons was set to 10, and for the RF algorithm, 100 trees were used. The results of comparison are shown in Table 5. The results exhibits that SMOTE-*t*BPSO-SVM achieved the highest g-mean with a value of 97.5%, even that 1-NN achieved relatively closely good results (95.3%). Regarding the sensitivity, the proposed (SMOTE-*t*BPSO-SVM) also obtained the best score of 96.4%, while the 5-NN, and the RF algorithms achieved the worst results by having 87.1%. In terms of specificity, the MLP algorithm obtained the best score of 99.9%. Even that the proposed SMOTE-*t*BPSO-SVM method did not accomplish the best result in terms of specificity, but it obtained slightly closely to the MLP by having a score of 98.7%. Generally speaking, the proposed (SMOTE-*t*BPSO-SVM) achieved better than classical classification algorithms in terms of sensitivity and g-mean.

D. COMPARISON WITH OTHER FEATURE SELECTION METAHEURISTICS

This section compares between the results obtained from using PSO for feature selection and using other feature selection metaheuristic techniques including the Fire-Fly Algorithm (FFA), Multi-Verse Optimizer (MVO), and Grey Wolf Optimizer (GWO). The same settings are considered for the different metaheuristic algorithms having 50 iterations and a population size value of 50. Table 6 shows the comparative results for these algorithms. The results exhibits that SMOTE-*t*BPSO-SVM achieved the value of 98.7%,96.4%, and 97.5% for the specificity, sensitivity, and g-mean, respectively, which are the highest values compared to the other feature selection metaheuristic algorithms.

TABLE 7. Comparison with other feature selection metaheuristics.

	Time (sec)
1-NN	39.01
3-NN	41.06
5-NN	42.03
NB	28.40
MLP	27.98
RF	27.34
SMOTE-*t*BFFA-SVM	177.15
SMOTE-*t*BMVO-SVM	145.36
SMOTE-*t*BGWO-SVM	287.03
SMOTE-*t*BPSO-SVM	157.48

E. TIME ANALYSIS WITH THE OTHER ALGORITHMS

This section shows the running time obtained by the proposed approach and the other standard and metaheuristic algorithms on the same workstation. Table 7 shows the running time in seconds for these algorithms. It is observed that the standard algorithms have close values, and that K-NN takes more time than NB, MLP, and RF. Metaheuristic algorithms, on the other hand, also have close values, and GWO is consuming increased amount of time compared to the other metaheuristic algorithms.

At another level, the standard algorithms take a recognizable smaller amount of time than any other metaheuristic algorithms, but still, metaheuristic algorithms take a reasonable amount of time. Since the complexity of finding the best combination of features makes it impossible to search for every possible combination, the aim is to find the best combination of features with high quality results in reasonable time, which can be achieved using metaheuristic algorithms. It is well-known that metaheuristic algorithms take reasonable amount of time to produce high quality of results of the acceptable solutions [74]. Such high quality results is observed and discussed in the previous sections, which justifies the increased amount of time compared to the standard algorithms.

F. FEATURE IMPORTANCE ANALYSIS

This part of the experiment identifies the most important features in the process of ransomware detection according to the proposed classification approach. An importance score is given for each feature according to the number of times it appears in the best solution over 30 independent experiments. Table 8 presents the features that have an importance score greater than 70%. It is clear from the table that the features are divided evenly, where half of the features refer to permission features, and the other half is API calls features. Moreover, it can be deduced from the table that the highly scored features are almost permission features more than they are API call features, as will be justified in the following paragraphs.

It is not surprising to see ''SYSTEM_ALERT_WINDOW'' permission with a high score. Ransomware apps usually

TABLE 8. List of features that have importance score ≥ 0.70.

Feature	Score	Feature	Score
SYSTEM_ALERT_WINDOW	1.0	android/content/pm	0.7
READ_PHONE_STATE	1.0	javax/xml/namespace	0.7
android/os	0.9	WRITE_CALENDAR	0.7
ACCESS_NETWORK_STATE	0.9	RESTART_PACKAGES	0.7
KILL_BACKGROUND_PROCESSES	0.8	RECEIVE_SMS	0.7
android/system	0.8	MANAGE_DOCUMENTS	0.7
READ_CALL_LOG	0.7	INSTALL_LOCATION_PROVIDER	0.7
INSTALL_SHORTCUT	0.7	GET_TASKS	0.7
javax/microedition/khronos/egl	0.7	BROADCAST_PACKAGE_REMOVED	0.7
java/awt/font	0.7	java/util/jar	0.7
android/telephony/mbms	0.7	java/nio/charset/spi	0.7
android/opengl	0.7	java/lang/ref	0.7
android/hardware/camera2	0.7	android/speech	0.7
UNINSTALL_SHORTCUT	0.7	android/net/wifi/p2p/nsd	0.7
BLUETOOTH_ADMIN	0.7	android/hardware/camera2/params	0.7
BIND_VPN_SERVICE	0.7	android/graphics/fonts	0.7

abuse it either to display the ransom message or block access to the device by overlaying the screen such that the victim could not dismiss or circumvent. Moreover, many attack scenarios require the permission ''READ_PHONE_STATE'' to collect sensitive user information. Therefore, it is common for ransomware apps to abuse this dangerous permission to gather information, including phone state, cellular network information, and contacts list. Additionally, this information could be utilized by ransomware to send SMS messages to the contacts list to spread itself to other devices.

''ACCESS_NETWORK_STATE'' permission was in the top list as well with 90% of importance score. Ransomware could abuse this permission to monitor network connections at the victim's device to facilitate the communication with Command and Control (C&C) server.

Moreover, ''KILL_BACKGROUND_PROCESSES'' is another typical permission abused by attackers and is not popular in benign apps. It had an importance factor of 80% as the attacker could mistreat this permission to block processes such as Anti-virus apps to prevent its detection. Similarly, the ransomware application can terminate all other background processes that belong to other apps using the ''RESTART_PACKAGES'' permission or shut down the app using ''BROADCAST_PACKAGE_REMOVED'' permission. Further, it was expected to see ''GET_TASKS'' permission in the list too. This permission permits the attacker to retrieve details about all running tasks in the victim device, which then allows the attacker to perform activity hijacking.

The permission ''READ_CALL_LOG'' is not supposed to be requested by any app as attackers can use it to access phone call records secretly. Some permissions are usually exploited by ransomware applications to gain root access to the victim's device and monopolize its resources, such as ''INSTALL_SHORTCUT'' and ''UNINSTALL_SHORTCUT'' that enable an application to add/remove shortcuts to the main screen without the user involvement. Another permission provides root access is ''BLUETOOTH_ADMIN'' which allows users to scan and pair with other Bluetooth devices. However, the ransomware utilizes it to gather technical information regarding the network topology. Furthermore, ''BIND_VPN_SERVICE'' permission is usually abused by many ransomware apps to create VPN clients in the network. Consequently, the attacker can manipulate the traffic by re-routing it to a remote VPN server.

In some scenarios, an attacker might send messages to the victim's contacts using ''WRITE_CALENDAR'' permission, enabling the app to modify the calendar data and send messages. Consequently, these messages are sent as legitimate messages of the calendar owner. Additionally, the requested ransom by the ransomware application may require ''RECEIVE_SMS'' permission to receive and process SMS messages to complete the online payment. To further force its control, ransomware application could utilize ''MANAGE_DOCUMENTS'' permission, which can give access to locations of documents that can be then read and encrypted; mainly, it functions as a document picker.

In other scenarios, the ransomware application tries to access the device's location to retrieve further information about the victim. However, if the victim uses Google apps, the ransomware application might be prevented from accessing the device's location using a location provider because it is not signed with the same key of Google apps. In this case, the ransomware app has to request ''INSTALL_LOCATION_PROVIDER'' permission, which was in the list with 70% importance, to impose installing its location provider into Android Location Manager.

On the other hand, API calls features had more influence in discriminating benign apps. Benign apps heavily

used the listed API calls in comparison to Ransomware apps. For example, ''android/system'' package was used by 73.35% of the benign apps and by only 0.6% of the Ransomware apps. Furthermore, ''android/opengl'' and ''android/graphics/fonts'' were used by 80.84% and 74.82% of the benign apps, respectively. However, ''android/opengl'' was requested by only 4% of Ransomware samples, and ''android/graphics/fonts'' was never called by them. This applies to the rest of the listed API calls features.

VIII. CONCLUSION AND FUTURE WORKS

Ransomware attacks have been causing serious damages across continents. This demands developing more resilient intelligent methods for the detection of such malware. This paper presented an evolutionary-based machine learning approach for the detection of Ransomware. The proposed approach (SMOTE-*t*BPSO-SVM) used the linear SVM for the classification and detection, while the BPSO is the optimization and search algorithm. The BPSO is integrated to optimize the cost coefficient of the SVM, in addition, to tune the k neighbors and the sampling ratio of the oversampling method. The constructed dataset comprises 10,153 applications, where 500 of them are Ransomware, which is covering various features of permissions and API calls. Meanwhile, the created dataset is a highly imbalanced dataset which is imitating reality. Several oversampling methods were adopted and compared, including SMOTE, Borderline-SMOTE1, Borderline-SMOTE2, ADASYN, and SVM-SMOTE. The SMOTE accomplished the best in regard to the sensitivity, specificity, and g-mean. Thereby, the performance of the proposed SMOTE-*t*BPSO-SVM compared with traditional machine learning algorithms, such as the K-NN, NB, MLP, and RF. The results have shown the merits of the proposed approach capacity in detecting Ransomware efficiently (97.5% of g-mean), yet justified with smooth convergence over the training iterations. Even though, this research study can be extended into additional research by utilizing more data as well as advanced models to handle the obtained big data, such as the deep learning algorithms that are more capable to infer accurate patterns of relationships.

ACKNOWLEDGMENT

The authors would like to acknowledge the support of Prince Sultan University for paying the Article Processing Charges (APC) of this publication.

REFERENCES

[1] C. Bansal, P. Deligiannis, C. Maddila, and N. Rao, ''Studying ransomware attacks using Web search logs,'' 2020, *arXiv:2005.00517*. [Online]. Available: http://arxiv.org/abs/2005.00517

[2] M. Paquet-Clouston, B. Haslhofer, and B. Dupont, ''Ransomware payments in the bitcoin ecosystem,'' *J. Cybersecur.*, vol. 5, no. 1, Jan. 2019, Art. no. tyz003.

[3] R. Fang, M. Xu, and P. Zhao, ''Should the ransomware be paid?'' 2020, *arXiv:2010.06700*. [Online]. Available: http://arxiv.org/abs/2010.06700

[4] S. Sharma, R. Kumar, and C. R. Krishna, ''RansomAnalysis: The evolution and investigation of Android ransomware,'' in *Proc. Int. Conf. IoT Inclusive Life (ICIIL)*, NITTTR Chandigarh, India: Springer, 2020, pp. 33–41.

[5] N. Perlroth. (Aug. 2014). *Android Phones Hit by Ransomware*. [Online]. Available: https://www.nytimes.com

[6] A. H. Mohammad, ''Ransomware evolution, growth and recommendation for detection,'' *Mod. Appl. Sci.*, vol. 14, no. 3, p. 68, Feb. 2020.

[7] K. Liu, S. Xu, G. Xu, M. Zhang, D. Sun, and H. Liu, ''A review of Android malware detection approaches based on machine learning,'' *IEEE Access*, vol. 8, pp. 124579–124607, 2020.

[8] I. Almomani and A. Khayer, ''Android applications scanning: The guide,'' in *Proc. Int. Conf. Comput. Inf. Sci. (ICCIS)*, Apr. 2019, pp. 1–5.

[9] Y. Pan, X. Ge, C. Fang, and Y. Fan, ''A systematic literature review of Android malware detection using static analysis,'' *IEEE Access*, vol. 8, pp. 116363–116379, 2020.

[10] Z. Abdullah, F. W. Muhadi, M. M. Saudi, I. R. A. Hamid, and C. F. M. Foozy, ''Android ransomware detection based on dynamic obtained features,'' in *Proc. Int. Conf. Soft Comput. Data Mining*. Cham, Switzerland: Springer, 2020, pp. 121–129.

[11] M. N. Rajkumar, V. V. Kumar, and R. Vijayabhasker, ''A hybrid approach to detect the malicious applications in Android-based smartphones using deep learning,'' in *Handbook of Research on Machine and Deep Learning Applications for Cyber Security*. Hershey, PA, USA: IGI Global, 2020, pp. 176–194.

[12] P. Agrawal and B. Trivedi, ''Machine learning classifiers for Android malware detection,'' *Data Manage., Anal. Innov.*, vol. 1, p. 311, Mar. 2020.

[13] A. Amouri, V. T. Alaparthy, and S. D. Morgera, ''A machine learning based intrusion detection system for mobile Internet of Things,'' *Sensors*, vol. 20, no. 2, p. 461, Jan. 2020.

[14] M. S. Hussain and K. U. R. Khan, ''A survey of IDS techniques in MANETs using machine-learning,'' in *Proc. 3rd Int. Conf. Comput. Intell. Inform.* Singapore: Springer, 2020, pp. 743–751.

[15] E. Mugabo and Q.-Y. Zhang, ''Intrusion detection method based on support vector machine and information gain for mobile cloud computing,'' *IJ Netw. Secur.*, vol. 22, no. 2, pp. 231–241, 2020.

[16] J. Long, S. Zhang, and C. Li, ''Evolving deep echo state networks for intelligent fault diagnosis,'' *IEEE Trans. Ind. Informat.*, vol. 16, no. 7, pp. 4928–4937, Jul. 2020.

[17] J. Long, J. Mou, L. Zhang, S. Zhang, and C. Li, ''Attitude data-based deep hybrid learning architecture for intelligent fault diagnosis of multi-joint industrial robots,'' *J. Manuf. Syst.*, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0278612520301436, doi: 10.1016/j.jmsy.2020.08.010.

[18] R. Eberhart and J. Kennedy, ''A new optimizer using particle swarm theory,'' in *Proc. 6th Int. Symp. Micro Mach. Hum. Sci. (MHS)*, Oct. 1995, pp. 39–43.

[19] B. E. Boser, I. M. Guyon, and V. N. Vapnik, ''A training algorithm for optimal margin classifiers,'' in *Proc. 5th Annu. Workshop Comput. Learn. Theory*. New York, NY, USA: Association for Computing Machinery, 1992, p. 144–152, doi: 10.1145/130385.130401.

[20] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, ''SMOTE: Synthetic minority over-sampling technique,'' *J. Artif. Intell. Res.*, vol. 16, pp. 321–357, Jun. 2002.

[21] H. Han, W.-Y. Wang, and B.-H. Mao, ''Borderline-SMOTE: A new over-sampling method in imbalanced data sets learning,'' in *Proc. Int. Conf. Intell. Comput.* Berlin, Germany: Springer, 2005, pp. 878–887.

[22] H. He, Y. Bai, E. A. Garcia, and S. Li, ''ADASYN: Adaptive synthetic sampling approach for imbalanced learning,'' in *Proc. IEEE Int. Joint Conf. Neural Netw. (IEEE World Congr. Comput. Intell.)*, Jun. 2008, pp. 1322–1328.

[23] R. Oak, M. Du, D. Yan, H. Takawale, and I. Amit, ''Malware detection on highly imbalanced data through sequence modeling,'' in *Proc. 12th ACM Workshop Artif. Intell. Secur.*, 2019, pp. 37–48.

[24] S. Alsoghyer and I. Almomani, ''Ransomware detection system for Android applications,'' *Electronics*, vol. 8, no. 8, p. 868, Aug. 2019.

[25] N. Lachtar, D. Ibdah, and A. Bacha, ''The case for native instructions in the detection of mobile ransomware,'' *IEEE Lett. Comput. Soc.*, vol. 2, no. 2, pp. 16–19, Jun. 2019.

[26] D. Su, J. Liu, X. Wang, and W. Wang, ''Detecting Android locker-ransomware on Chinese social networks,'' *IEEE Access*, vol. 7, pp. 20381–20393, 2019.

[27] A. Alzahrani, H. Alshahrani, A. Alshehri, and H. Fu, ''An intelligent behavior-based ransomware detection system for Android platform,'' in *Proc. 1st IEEE Int. Conf. Trust, Privacy Secur. Intell. Syst. Appl. (TPS-ISA)*, Dec. 2019, pp. 28–35.

[28] T. Kim, B. Kang, M. Rho, S. Sezer, and E. G. Im, "A multimodal deep learning method for Android malware detection using various features," *IEEE Trans. Inf. Forensics Security*, vol. 14, no. 3, pp. 773–788, Mar. 2019.

[29] A. Pektaş and T. Acarman, "Learning to detect Android malware via opcode sequences," *Neurocomputing*, vol. 396, pp. 599–608, Jul. 2020.

[30] M. Scalas, D. Maiorca, F. Mercaldo, C. A. Visaggio, F. Martinelli, and G. Giacinto, "On the effectiveness of system API-related information for Android ransomware detection," *Comput. Secur.*, vol. 86, pp. 168–182, Sep. 2019.

[31] A. K. Singh, G. Wadhwa, M. Ahuja, K. Soni, and K. Sharma, "Android malware detection using LSI-based reduced opcode feature vector," *Procedia Comput. Sci.*, vol. 173, pp. 291–298, 2020.

[32] S. Alsoghyer and I. Almomani, "On the effectiveness of application permissions for Android ransomware detection," in *Proc. 6th Conf. Data Sci. Mach. Learn. Appl. (CDMA)*, Mar. 2020, pp. 94–99.

[33] M. Alazab, M. Alazab, A. Shalaginov, A. Mesleh, and A. Awajan, "Intelligent mobile malware detection using permission requests and API calls," *Future Gener. Comput. Syst.*, vol. 107, pp. 509–521, Jun. 2020.

[34] H. Faris, M. Habib, I. Almomani, M. Eshtay, and I. Aljarah, "Optimizing extreme learning machines using chains of salps for efficient Android ransomware detection," *Appl. Sci.*, vol. 10, no. 11, p. 3706, May 2020.

[35] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2016, pp. 785–794.

[36] I. Bibi, A. Akhunzada, J. Malik, G. Ahmed, and M. Raza, "An effective Android ransomware detection through multi-factor feature filtration and recurrent neural network," in *Proc. UK/China Emerg. Technol. (UCET)*, Aug. 2019, pp. 1–4.

[37] A. Alzahrani, A. Alshehri, H. Alshahrani, R. Alharthi, H. Fu, A. Liu, and Y. Zhu, "RanDroid: Structural similarity approach for detecting ransomware applications in Android platform," in *Proc. IEEE Int. Conf. Electro/Inf. Technol. (EIT)*, May 2018, pp. 0892–0897.

[38] J. Chen, C. Wang, Z. Zhao, K. Chen, R. Du, and G.-J. Ahn, "Uncovering the face of Android ransomware: Characterization and real-time detection," *IEEE Trans. Inf. Forensics Security*, vol. 13, no. 5, pp. 1286–1300, May 2018.

[39] N. Andronio, S. Zanero, and F. Maggi, "HelDroid: Dissecting and detecting mobile ransomware," in *Proc. Int. Symp. Recent Adv. Intrusion Detection*. Cham, Switzerland: Springer, 2015, pp. 382–404.

[40] ID Ransomware. *List of Ransomware*. Accessed: Apr. 9, 2021. [Online]. Available: https://id-ransomware.blogspot.com/2016/07/ransomware-list.html

[41] theZoo. *theZoo—A Live Malware Repository*. Accessed: Apr. 9, 2021. [Online]. Available: https://thezoo.morirt.com/

[42] Contagio. *Contagio Malware Dump*. Accessed: Apr. 9, 2021. [Online]. Available: http://contagiodump.blogspot.com/

[43] Koodous. *Koodous APKs*. Accessed: Apr. 9, 2021. [Online]. Available: https://koodous.com/apks

[44] VirusTotal. *Analyze Suspicious Files and URLs to Detect Types of Malware, Automatically*. Accessed: Apr. 9, 2021. [Online]. Available: https://www.virustotal.com/

[45] F. Wei, Y. Li, S. Roy, X. Ou, and W. Zhou, "Deep ground truth analysis of current Android malware," in *Proc. Int. Conf. Detection Intrusions Malware, Vulnerability Assessment*. Cham, Switzerland: Springer, 2017, pp. 252–276.

[46] Tencent. *I'm QQ*. Accessed: Apr. 9, 2021. [Online]. Available: https://im.qq.com/

[47] K. Project. *Kharon Malware Dataset*. Accessed: Apr. 9, 2021. [Online]. Available: http://kharon.gforge.inria.fr/dataset/

[48] A. H. Lashkari, A. F. A. Kadir, L. Taheri, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark Android malware datasets and classification," in *Proc. Int. Carnahan Conf. Secur. Technol. (ICCST)*, Oct. 2018, pp. 1–7.

[49] U.-C. I. for Cybersecurity. *Investigation of the Android Malware (CICInvesAndMal2019)*. Accessed: Apr. 9, 2021. [Online]. Available: https://www.unb.ca/cic/datasets/invesandmal2019.html

[50] C. A. C. Coello, G. T. Pulido, and M. S. Lechuga, "Handling multiple objectives with particle swarm optimization," *IEEE Trans. Evol. Comput.*, vol. 8, no. 3, pp. 256–279, Jun. 2004.

[51] J. Kennedy and R. C. Eberhart, "A discrete binary version of the particle swarm algorithm," in *Proc. IEEE Int. Conf. Syst., Man, Cybern. Comput. Cybern. Simulation*, vol. 5, Oct. 1997, pp. 4104–4108.

[52] X. Zhang, D. Zou, and X. Shen, "A novel simple particle swarm optimization algorithm for global optimization," *Mathematics*, vol. 6, no. 12, p. 287, 2018.

[53] A. M. Sarhan, "Brain tumor classification in magnetic resonance images using deep learning and wavelet transform," *J. Biomed. Sci. Eng.*, vol. 13, no. 6, p. 102, 2020.

[54] M. Zulqarnain, R. Ghazali, Y. M. Mohmad Hassim, and M. Rehan, "Text classification based on gated recurrent unit combines with support vector machine," *Int. J. Electr. Comput. Eng. (IJECE)*, vol. 10, no. 4, p. 3734, Aug. 2020.

[55] A. Şentaş, İ. Tashiev, F. Küçükayvaz, S. Kul, S. Eken, A. Sayar, and Y. Becerikli, "Performance evaluation of support vector machine and convolutional neural network algorithms in real-time vehicle type and color classification," *Evol. Intell.*, vol. 13, no. 1, pp. 83–91, Mar. 2020.

[56] J. Cervantes, F. Garcia-Lamont, L. Rodríguez-Mazahua, and A. Lopez, "A comprehensive survey on support vector machine classification: Applications, challenges and trends," *Neurocomputing*, vol. 408, pp. 189–215, Sep. 2020.

[57] O. Chapelle, "Training a support vector machine in the primal," *Neural Comput.*, vol. 19, no. 5, pp. 1155–1178, May 2007.

[58] H. M. Nguyen, E. W. Cooper, and K. Kamei, "Borderline over-sampling for imbalanced data classification," *Int. J. Knowl. Eng. Soft Data Paradigms*, vol. 3, no. 1, pp. 4–21, Apr. 2011.

[59] Z. Xu, D. Shen, T. Nie, and Y. Kou, "A hybrid sampling algorithm combining M-SMOTE and ENN based on random forest for medical imbalanced data," *J. Biomed. Informat.*, vol. 107, Jul. 2020, Art. no. 103465.

[60] S. Matsukawa, T. Nakayama, C. Ninagawa, and J. Morikawa, "Dynamic SMOTE training of neural networks used in real-time pricing control for building air-conditioners," *IEEJ Trans. Elect. Electron. Eng.*, vol. 14, no. 11, pp. 1727–1728, 2019.

[61] K. Chen and B. Liu, "Loan risk prediction method based on SMOTE and XGBoost," *Comput. Mod.*, no. 2, pp. 26–30, 2020. [Online]. Available: http://www.c-a-m.org.cn/EN/10.3969/j.issn.1006-2475.2020.02.006

[62] H. Al Majzoub, I. Elgedawy, O. Akaydın, and M. K. Ulukök, "HCAB-SMOTE: A hybrid clustered affinitive borderline SMOTE approach for imbalanced data binary classification," *Arabian J. Sci. Eng.*, vol. 45, pp. 3205–3222, Jan. 2020.

[63] X. Zheng, "SMOTE variants for imbalanced binary classification: Heart disease prediction," Univ. California, Los Angeles, Los Angeles, CA, USA, 2020.

[64] G. Suarez-Tangil and G. Stringhini, "Eight years of rider measurement in the Android malware ecosystem," *IEEE Trans. Dependable Secure Comput.*, early access, Mar. 30, 2020, doi: 10.1109/TDSC.2020.2982635.

[65] S. Zhu, J. Shi, L. Yang, B. Qin, Z. Zhang, L. Song, and G. Wang, "Measuring and modeling the label dynamics of online anti-malware engines," in *Proc. 29th USENIX Secur. Symp. (USENIX Secur.)*, 2020, pp. 2361–2378.

[66] A. Salem, S. Banescu, and A. Pretschner, "Maat: Automatically analyzing virustotal for accurate labeling and effective malware detection," 2020, *arXiv:2007.00510*. [Online]. Available: http://arxiv.org/abs/2007.00510

[67] M. Ahmad, V. Costamagna, B. Crispo, F. Bergadano, and Y. Zhauniarovich, "StaDART: Addressing the problem of dynamic code updates in the security analysis of Android applications," *J. Syst. Softw.*, vol. 159, Jan. 2020, Art. no. 110386.

[68] İ. A. Doğru and M. Önder, "AppPerm analyzer: Malware detection system based on Android permissions and permission groups," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 30, no. 03, pp. 427–450, Mar. 2020.

[69] C. Li, K. Mills, D. Niu, R. Zhu, H. Zhang, and H. Kinawi, "Android malware detection based on factorization machine," *IEEE Access*, vol. 7, pp. 184008–184019, 2019.

[70] A. Al Khayer, I. Almomani, and K. Elkawlak, "ASAF: Android static analysis framework," in *Proc. 1st Int. Conf. Smart Syst. Emerg. Technol. (SMARTTECH)*, Nov. 2020, pp. 197–202.

[71] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, Oct. 2011.

[72] G. Lemaître, F. Nogueira, and C. K. Aridas, "Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning," *J. Mach. Learn. Res.*, vol. 18, no. 17, pp. 1–5, 2017. [Online]. Available: http://jmlr.org/papers/v18/16-365.html

[73] M. Habib, I. Aljarah, and H. Faris, "A modified multi-objective particle swarm optimizer-based Lévy flight: An approach toward intrusion detection in Internet of Things," *Arabian J. Sci. Eng.*, vol. 45, pp. 6081–6108, Mar. 2020.

[74] X.-S. Yang, *Nature-Inspired Metaheuristic Algorithms*. U.K.: Luniver Press, 2010.

IMAN ALMOMANI (Senior Member, IEEE) received the bachelor's degree from United Arab Emirates, in 2000, the master's degree in computer science from Jordan, in 2002, and the Ph.D. degree in wireless network security from De Montfort University, U.K., in 2007. She is currently an Associate Professor in cybersecurity. She is also the Associate Director of the Research and Initiatives Centre (RIC) and also the Leader of the Security Engineering Laboratory (SEL), Prince Sultan University (PSU), Riyadh, Saudi Arabia. Before Joining Prince Sultan University, she has worked as an Associate Professor and the Head of the Computer Science Department, The University of Jordan, Jordan. Her research interests include wireless networks and security, mainly wireless mobile ad hoc networks (WMANETs), wireless sensor networks (WSNs), multimedia networking (VoIP), and security issues in wireless networks. She is also interested in the area of electronic learning (e-learning) and mobile learning (m-learning). She has several publications in the above areas in a number of reputable international and local journals and conferences. She is also a Senior Member of IEEE WIE. She is on the organizing and technical committees of a number of local and international conferences. She also serves as a reviewer and a member of the editorial board for a number of international journals.

RANEEM QADDOURA received the Ph.D. degree in computer science in the fields of machine learning and data mining. She is currently an Assistant Professor with Philadelphia University. She combines both academic and industrial experience of a total of 14 years of experience. She is also an active Research Member with the Evolutionary and Machine Learning Group, which focuses on evolutionary algorithms, machine learning, and their applications for solving important problems in different areas. Her current research interests include evolutionary computation and data clustering and classification.

MARIA HABIB received the bachelor's degree in computer engineering from the Faculty of Engineering and Technology, The University of Jordan, and the master's degree in web intelligence from the Department of Information Technology, King Abdullah II School of Information Technology. She was a Research Assistant with The University of Jordan. She is currently a Data Science Engineer and a Researcher with Altibbi, Amman, Jordan. She is also a former Graduate Research Trainee in bioinformatics and big data analysis, the Bioinformatics Laboratory–supervised by Jianguo (Jeff) Xia at the Parasitology Department, McGill University. She is also a member of the Research Group. Her research interests include machine learning and deep learning techniques, natural language processing, evolutionary algorithms, multi-objective optimization, as well as, in biomedical and bioinformatics applications.

SAMAH ALSOGHYER is currently a Researcher with the National Center for Cybersecurity Technologies, KACST. Her research interests include malware analysis as well as the identification and mitigation of cyberattacks.

ALAA AL KHAYER received the B.Eng. degree in information technology engineering from SVU University, Damascus, in 2017, and the bachelor's degree in software engineering from Prince Sultan University (PSU), Riyadh, Saudi Arabia, in 2018. She is currently a Research Engineer with the Security Engineering Laboratory (SEL), PSU. Her research interests include software engineering, networks security, malware analysis, multimedia networking, and computer vision.

IBRAHIM ALJARAH received the bachelor's degree in computer science from Yarmouk University, Jordan, in 2003, the master's degree in computer science and information systems from the Jordan University of Science and Technology, Jordan, in 2006, and the Ph.D. degree in computer science from North Dakota State University, USA, in 2014. He is currently an Associate Professor of big data mining and computational intelligence with the Department of Information Technology, The University of Jordan, Jordan. He participated in many conferences in the field of data mining, machine learning, and big data, such as CEC, GECCO, NTIT, CSIT, IEEE NABIC, CASON, and BIGDATA Congress. Furthermore, he contributed in many projects in USA such as Vehicle Class Detection System (VCDS), Pavement Analysis Via Vehicle Electronic Telemetry (PAVVET), and Farm Cloud Storage System (CSS) projects. He has published more than 80 publications in refereed international conferences and first and second quartile journals with more than 3500 citations and an H-index of 32. His research interests include data mining, data science, machine learning, opinion mining, sentiment analysis, big data, MapReduce, Hadoop, swarm intelligence, evolutionary computation, and large-scale distributed algorithms. He ranked amongst the top 2% scientists in the world and the one of top ten scientists in Jordan in Artificial Intelligence and Image Processing field, as per Stanford University Global Ranking 2020.

HOSSAM FARIS received the B.A. degree in computer science from Yarmouk University, Jordan, in 2004, the M.Sc. degree in computer science from Al-Balqa' Applied University, Jordan, in 2008, and the Ph.D. degree in e-business from the University of Salento, Italy, in 2011. He is currently a Professor with the School of Computing and Informatics, Al Hussein Technical University; and the Information Technology Department, King Abdullah II School for Information Technology, The University of Jordan. In 2016, he has worked as a Postdoctoral Researcher with the Information and Communication Technologies Research Center (CITIC), GeNeura Team, University of Granada, Spain. He co-founded the Evolutionary and Machine Learning Research Group. His research interests include applied machine learning, evolutionary computation, knowledge systems, and data mining. He was awarded a Full-Time Competition-Based Scholarship from the Italian Ministry of Education and Research for his Ph.D. degree.