Journal of Data Acquisition and Processing
ISSN: 1004-9037
https://sjcjycl.cn/
DOI: 10.5281/zenodo.776609

ANDROID FORTIFY: ELEVATING ANDROID SECURITY THROUGH THE SYNERGY OF MACHINE LEARNING AND BINARY PARTICLE SWARM OPTIMIZATION

[1]R. M. Sharma ,[2]Chaitanya P Agrawal
[1]Department of Computer Science and Applications, Makhanlal Chaturvedi University, Bhopal, MP 462001 India
[2]Department of Computer Science and Applications, Makhanlal Chaturvedi University, Bhopal, MP 462001 India

Abstract

Smartphone applications (APPs) play essential roles in daily activities like online shopping, mobile banking, online transactions, etc. The exponential growth of online transactions has attracted the attention of hackers. Hackers are increasing their efforts to deploy malicious applications to users to steal sensitive information such as ATM PINs and bank account detail. The Traditional malware detection systems (MDS) require significant computational overload and time to analyze malware behavior patterns and find invasive tendencies. This research aims to expose the dangerous behavior of android malware to detect them quickly. We offer a negotiation by examining several types of static behavior patterns using BPSO (Binary Particle Swarm Optimization) to reduce computational complexity and pick the most optimal subset. The BPSO is hybridized with six machine learning techniques to get the complete solution for feature optimization and malware detection. The Binary Particle Swarm Optimization (BPSO) technique chooses an optimal subset from behavioral feature sets and provides the best fitness values. The Six machine learning techniques are utilized with BPSO to generate MDS models. The anticipated system has been empirically tested with three benchmark android datasets: DREBIN, MALGENOME, and the MENDELEY dataset. The proposed method achieved an accuracy of 96% with a 94% recall rate and 96 % f1 score. The high values of true positive (TP) and true negative (TN), indicate the model's effectiveness in both primary and secondary classes. The suggested technique has a meager computational cost, allowing for real-time application analysis.

Keywords: Machine Learning, Android Feature Selection, Malware Detection, security, Binary Particle Swarm Optimization

1. Introduction

Smartphones are now extremely important for e-shopping, internet payments, web surfing, social media, and other internet activities. Statista reports 3.8 billion smartphone users worldwide. It may reach several hundred million in the coming years [1]. In comparison to other platforms, the Android OS currently holds the most significant portion of the market share. In June 2021, Android smartphones continued to hold a market share of 73.3 percent across the globe [2]. The Google Play store is currently the biggest app store in the world, and

by the beginning of the first quarter of 2022, it is expected to have 3.6 million apps easily accessible [3]. Malware is part of a program created to harm the system functions or embed them to infect other applications. The report from the AV-TEST institute states that in 2021, there will be more than 450,000 new pieces of malicious software (also known as malware) and potentially unwanted applications (PUAs)[4]. According to Statista, the cumulative amount of malicious samples for Android devices that are newly revealed each month equates to 482,579[5].

Android Smartphone have become the preferred medium of infiltration for cybercriminals. Along with the growth of benign ware (gentle apps), malware has also grown at an uncontrollable pace. Similarly, the number of cybercrimes through Smartphone is also increasing continuously. Some mobile apps can be infected with malicious code, and such apps are capable of obtaining sensitive information and breaching privacy. The malicious apps have the ability to steal private information, including financial data such as passwords, credit card and debit card details, and other similar information. Traditionally, malware recognition methods are based on signatures generated from the malware's source code. Therefore, the signature-based method requires a massive database of known malware. Nevertheless, it is impossible to maintain signature records due to the rapid and large-scale growth of malware and its variants. Signature-based approaches offer the advantages of being simple and efficient and having high accuracy. However, they cannot new malware [6], [7].

The use of behavior-based detection has become more commonplace as a direct response to the constraints that exist in the world today regarding signature-based detection. This method looks at how malware that has already been found behaves in patterns [8]. Many statistical attributes are extracted from the APK files of the apps to detect malware through behavioral analysis, and datasets are created from these attributes. Hence, malware detection requires the analysis of large datasets that require large amounts of memory and high computational power. Therefore, the selection of optimal features has more impact on classification accuracy. In this process, the optimization criteria are set so that the optimal subset N is obtained from the entire set M where $N < M$. In recent decades, many diverse methods for selecting features have been developed. These methods make use of a variety of search and evaluation strategies. The procedures utilized for feature selection can be classified into one of three primary groups: filter, wrapper, or embedded methods. Ranking scores are implemented in filtering methods. Analysis of variance, Pearson's correlation, chi-square, mutual information, information gain, and a host of other examples are some of the statistical methods used. The wrapper methods use ML accuracy to select the features with the best overall performance. The properties of the wrapper methods and the filter methods are combined into the embedded methods. The embedded methods use various techniques, such as ridges, lassos, and elastic nets, to prevent the model from over fitting. These methods require more time to compute, and most are inefficient because they only consider local optimal solutions. This has resulted in the development of many meta-heuristic techniques. In recent decades, various meta-heuristics algorithms (GA, ACO, BCO, PSO, etc.) have proven their usefulness in optimization in many domains. The authors present a hybrid method for detecting behavioral malware that combines

machine learning and a lightweight meta-heuristic algorithm (BPSO). The primary contributions that the proposed work will make are outlined below.

- In this paper, we demonstrate how BPSO and machine-learning techniques can be used for the detection of behavior-based malware.
- This article presents six different iterations of BPSO, each of which uses a unique machine-learning classification.
- We targeted three publicly available android benchmark datasets that researchers extensively use (i) MALGEMONE, (ii) DREBIN, (iii), and MENDELEY to authenticate the proposed approach.
- In addition to this, we contrast our findings with the various methods currently in use.

The remaining portions of the paper are structured as follows: The second section is dedicated to previous research that has been conducted in this field; the third section presents the BPSO algorithm that has been proposed. The procedure for choosing BPSO features is broken down and explained in the fourth section. In Section 5, we talk about the datasets, the pre-processing steps, and the experimental environment. In Section 6, we talk about the various performance evaluation metrics used in these experiments. In Section 7, the outcomes of the work that was proposed are broken down in great detail, and the section then draws to a close by discussing potential steps that could be taken next.

2. Related Work

The previous research on feature optimization and selection, machine learning-based Android malware detection, and other domains are covered in the following section. In the article [9], Taheri et al. proposed the hamming distance-based similarity approach, eliminating most similar malware and benign ware features from the dataset. In the article [10], Wu et al. presented an 'MVIIDroid' mechanism for malware detection. They used the Multiple Kernel Learning (MKL) models for classification. They compared the results with other methods, such as RF, JS, and SVM. The paper by Santosh Jhansi et al. [11] illustrated the gain ratio-based ranking method to choose pertinent attributes from the dataset and used the J48, RC, MLP, SMO, and randomizable filtered methods for malware classification.

The "DroidTrace" method was developed by Zheng et al. [12] and is based on Ptrace-based dynamic analysis. It helps monitor specific system calls for the target process executing the dynamic payloads. A hybrid malware detection method called "StaDART" was developed by Ahmed et al. [13] to work with dynamic code update features. In another study, Min Yang et al. [14] present a DT+SVM-based malware recognition technique using the DREBIN dataset. In a different research, L. Sun et al. [15] proposed SIGPID, which stands for Significant Permission Identification for ML-based AMD. SVM was used to classify the data.. In another paper, S. Wang et al. [16] offered a multi-view neural network method for malware detection. This method recognized the malicious apps based on URLs visited by the apps. In their paper, X. Jiang et al. [17] offered malware detection and classification utilizing SVM, J48, KNN, and NB based on fine-grained dangerous permission (FDP), based on fine-grained dangerous permission (FDP). Singh et al. [18] proposed a hybrid model for feature selection, using a

firefly bio-inspired algorithm and random forest ensemble classifier (HBRF) for credit card fraud detection. The suggested HBRF model has 96.23% accuracy and 3.7% inaccuracy.

In another study, M. A. Jerlin et al. [19] anticipated a Multi-Dimensional Naive Bayes (MDNB rete) ML technique for malware detection using API call sequence as a characteristic. In another paper, F. Idrees et al. [7] offered the intent and permission-based approach (PINdroid) where SMO, RF, NB, ML, and DT-based ensemble classifiers are used for classification. The rule-based attribute selection technique was proposed by A. Mehtab et al. [20] based on Contagio Dump and using the VirusShare dataset. In another study, M. Alzaylaee et al. [21] illustrated DL-Droid, a deep-learning model for malware discovery through dynamic analysis using stateful input generation. Arvind Mahindru et al. [22] presented the rough set feature selection method with four different machine learning classifiers for subset evaluation. These classifiers include deep learning, farthest first clustering, Y-MLP, and the nonlinear ensemble decision tree forest.

Jiayin Feng et al. [23] developed a new approach that cascades CNN and AutoEncoder known as CACNN, which is used to detect malware through the network traffic characteristics of APPs. Selvakumar B and Muneeswaran K [24] offered a firefly algorithm for feature selection from the KDD CUP 99 network dataset and C4.5, Bayesian Networks (BN), used for subset evaluation. In another work, S. Alam et al.[25] proposed a dominance tree of API calls (a DroidDomTree method) to find analogous patterns in Android apps for malware detection. Some more related work based on meta-heuristic methods is presented in Table 1.

Table 1. Some of the previous work is related to meta-heuristic methods

S. No.	Authors and Ref.	Domain	Classification Approach	Dataset	Feature Selection Meta
1.	[26]	Android	AdaBoost	Drebin	PSO
2.	[27]	Intrusion Detection	RF, C4.5, Forest BA	NSL-KDD, AWID	CFS-BA
3	[28]	Intrusion Detection	KNN	Hacker-Earth Network attack	Whale Pearson
4.	[29]	Different domain	KNN	UCI	IHHO
5.	[30]	Different Domain	NB	UCI	SOMI-GANB(GA)
6.	[31]	Different Domain	KNN	UCI	WOA
7.	[32]	Different Domain	SVM	UCI	PSO+SVM
8.	[33]	Android	NB, FT, MLP,	Self-Dataset	GS
9.	[34]	Medical	J48	UCI	ACO+BCO

10.	[35]	Android	SVM, NN	Self-dataset.	GA
11.	[36]	Android	DT, NB,	Contagio	PSORS-FS
12.	[37]	Android	KNN, L.R.	Self and UCI	Self-Variant
13.	[38]	Android	KELM	Self-dataset	SSA-KELM

In the related works cited above, it was observed that meta-heuristic approaches are more suitable for feature selection than traditional methods, and their hybridization with machine learning methods makes them more efficient. Therefore, a hybrid system has been proposed in the present work.

3. Proposed BPSODroid

The related works cited above showed that meta-heuristic approaches are more suitable for feature selection than traditional methods, and their hybridization with machine learning methods makes them more efficient. Therefore, a hybrid system has been proposed in the present work. This section initially describes the standard PSO algorithm, followed by binary particle swarm optimization. The last section describes the preparation of hybrid variants.

Particle swarm optimization (PSO) is a meta-heuristic intelligent population-based optimization algorithm. It is based on the prototype of swarm intelligence. The collective behavior of animals such as birds, fish, and ants inspires it. The PSO has been successfully used in various science and engineering optimization problems such as image processing, data mining, machine learning, etc. The PSO was initially introduced in 1995 by James Kennedy and Russel C. Eberhart. However, they were working to develop a model to define the social behavior of animals, such as herds of birds and schools of fish. However, they felt that the proposed model was proficient at optimization. Hence, they proposed a new model based on their experiment called Particle Swarm Optimization. The PSO algorithm simulates an animal's intelligent behavior, such as an ant, a bird, or a fish, as an intelligent agent (particle).

The particle represents an individual animal, while the swarm represents a group of animals. Swarm refers to the population of potential alternate solutions that are contained within the PSO. Every individual particle represents a potential answer to the optimization problem. Every particle occupies a particular location in the search space, which can be thought of as a collection of the various possible solutions to the optimization problem. The concept of the whole process of basic PSO is presented in Fig. 2. If X denotes the search space, then the location of particle i in the search space is represented by xi. The xi is a vector member of the search space. This position vector gets an additional component called the time index to distinguish between different time intervals, denoted by (t). The (t) is the discrete-time interval, indicating the iteration number of the algorithm. The xi (t) represents the position of a particle i in the time step (t) in the search space X.

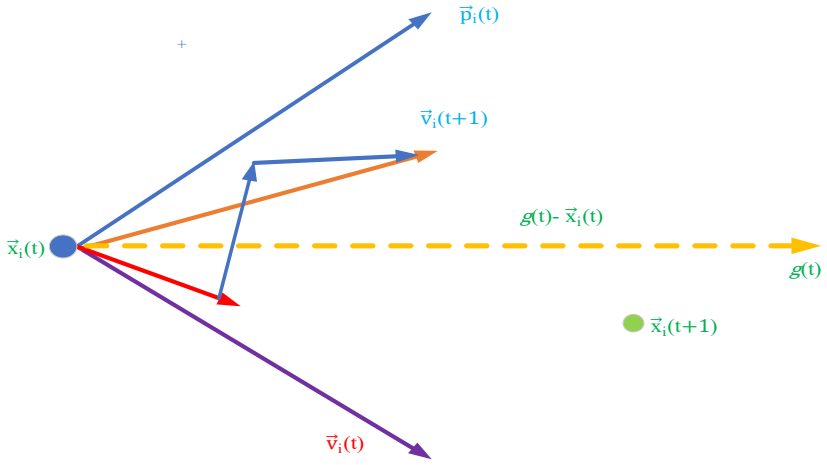

Figure 1. The working process of Binary Swarm Optimization

Besides the particle's position, each particle has a velocity in the time step (t), denoted by vi (t), a vector, and a member of the same search space. The velocity shows the movement and direction of the particle. Each particle is a member of the swarm. Each particle communicates with or learns from one another by exchanging information and operating according to a predetermined set of rules to find the optimal solution to the optimization problem. Each particle retains a memory of the position or solution that is optimal for it. It is represented by the individual particle's best experience, denoted by the symbol pi (t). In addition to the best solution for the individual, there is also the best solution for the entire swarm, denoted by the g (t). It is important to note that it does not have index I because it is related to the experience of the swarm as a whole rather than the experience of a particle in particular. g(t) is a global experience that all members of the swarm share in common. Therefore, there is the best solution for each particle and the best solution for the entire swarm, which is the best possible experience for all particles.

As shown in Fig. 1, the equation for the vector of initial position and personal best ($x_i(t)$ to $\vec{p}_i(t)$) will be ($\vec{p}_i(t)$ - $\vec{x}_i(t)$). Similarly, the equation for the vector between the current location and the global best ((t)to$\vec{x}_i(t)$) will be g(t)-$\vec{x}_i(t)$. If the particle moves from its current location to a new location, it uses all the previous information. The new spot of the particle is determined by the position to parallel the last three vectors, which include the former velocity $v_i(t)$, vector ($\vec{x}_i(t)$ to $\vec{p}_i(t)$), and ($g(t)to\vec{x}_i(t)$)) vector. The particle's new position and velocity after the time interval t is denoted by($\vec{x}_i(t+1)$), and ($\vec{v}_i(t+1)$). Hence the new location is created according to the previous velocity, individual best, and global best. So, this is probably a better position because it uses the previous experience of the particle, and it uses the prior knowledge of the whole swarm. All swarm particles obey the above rules to find the best location in the search space and collaborate to update the information. No swarm particle knows the best global solution, but collaboration makes it possible. This algorithm works on two principles: communication and learning. Through communication, particles communicate their experiences to each other. At the same time, the art of learning allows the particle to learn the

best location of the other particle. Thus, the principle of learning from the best place of others and getting the best solution from cooperation constitutes the intelligent behavior of the particle. Eq. (1) represents the new velocity, and the particle's new location is stated as follows.

$$\vec{v}_i(t+1)=w_{incof}\vec{v}_i(t)+r_{1rand}C_{1f}(\vec{p}_i(t)-\vec{x}_i(t)) + r_{2rand}C_{2f}(g(t)-\vec{x}_i(t)) \quad (1)$$

$$\vec{x}_i(t+1)=\vec{x}_i(t)+\vec{v}_i(t+1) \quad (2)$$

As given in Eq. (1), the new speed ($\vec{v}_i(t+1)$) of element i at time step (t+1) is prepared up of three vector mechanisms; the first vector component is parallel to the particle's prior velocity $\vec{v}_i(t)$, the second vector element, which is parallel to the vector of the particle's individual best ($\vec{p}_i(t)$-$\vec{x}_i(t)$) and the third element is that parallel to the vector of the global best ($g_i(t)$-$\vec{x}_i(t)$) of the swarm. Where w stands for the inertia coefficient, and (r_{1rand}, r_{2rand}) symbolizes the random numbers distributed evenly across the (0,1). The (C_{1f},C_{2f}) denotes the acceleration coefficient. In the velocity update equation, the first phrase ($w_{incof}\vec{v}_i(t)$) is recognized as inertia. The second phrase ($r_{1rand}C_{1f}(\vec{p}_i(t)-\vec{x}_i(t))$) is referred to as a cognitive term, and the last phrase $r_{2rand}C_{2f}\left(g_i(t)-\vec{x}_i(t)\right)$ is recognized as a social term. The cognitive phrase provides the particle's personal experience, while the social term provides the combined experience of the swarm. The best inertia coefficient w_{incof} value is between 0.3 and 0.72984, and the acceleration coefficient is ($C_{1f}+C_{2f}\geq 4$), which is more precisely (C1f = C2f = 2.05).

The position update equation, as given in Eq. (2), is made up of two components; the first component is the particle's previous location ($\vec{x}_i(t)$), while the second component is the updated velocity ($\vec{v}_i(t+1)$) obtained from the first Eq. (1). Each element moves from its earlier location to the new location with the updated velocity. In this way, each particle gets a better location than before. Every particle in the swarm obeys the rules defined in Eq. (1) and (2) to get the best solution to the optimization problem. The output of the PSO is a continuous stream of values. However, continuous values do not provide accurate information for selecting any attribute. Therefore, the ideal solution can be obtained by making the following changes to the above algorithm. The updating velocity in Binary Particle Swarm Optimization (PBSO) works the same way in Basic Particle Swarm Optimization. In addition, the notable change in BPSO is that the following expression determines the position update.

$$\vec{x}_i(t+1)=\begin{cases}0 & \text{if rand\_n } (\partial) \geq S_{ig}(\vec{v}_i(t+1)) \\ 1 & \text{if rand\_n } (\partial) < S_{ig}(\vec{v}_i(t+1))\end{cases} \quad (3)$$

Where (rand_n (∂)) is a random number between 0 and 1 and (S_{ig} (x)) is a sigmoid function defined in the following expression.

$$S_{ig}(\vec{v}_i(t+1))=\left\{\frac{1}{1+e^{-\vec{v}_i(t+1)}}\right\} \quad (4)$$

These modifications enable the algorithm's output to be in binary format rather than as a continuous stream of values. The binary output provides accurate information about the selected features, where 1 represents the selected attribute and 0 means the unselected feature. After receiving the result in binary format, it is converted into feature numbers by a small program. A reduced dataset is prepared from the obtained feature numbers. The six different classification methods in the Python Scikit-learn machine learning library are combined in the BPSO to make the different types of the BPSO. For this hybridization, K-Nearest Neighbor (KNN), Decision Tree (DT), Logistic Regression (LR), Support Vector Machine (SVM), Random Forest (RF), and multi-layer perceptions (MLPs) have been used. In this way, a total of six (BPSO + kNN), (BPSO + DT), (BPSO + LR), (BPSO + SVM), (BPSO + RF), and (BPSO + MLP) variants were prepared.

4. Feature selection process using BPSO

The above-described BPSO algorithm selects the best attributes from the given dataset. The BPSO acquires the best global solution ($g_i(t)$), an array of binary numbers 0 and 1, obtained after N iterations. Output 1 represents the particular attribute that is selected, and 0 denotes the attribute that is masked. The number of available attributes in the datasets corresponds to the number of particles in the swarm. The total number of particles was 301 for dataset III, while datasets I and II each had 215 particles. In equations (1) and (3), the time step t represents one iteration, and N represents the total number of iterations. The vector ($\vec{v}_i$) corresponds to the velocity of an ith particle in the swarm. The velocity ($\vec{v}_i(t+1)$) is applied to the sigmoid function ($S_{ig}(x)$), which translates the output into binary format. Each particle has its own personal best experience and conveys its personal best experience ($\vec{p}_i(t)$) to the other particles to acquire the best global solution defined in the algorithm. The BPSO algorithm is executed N times, and the outcome of each iteration is achieved to produce a list of the dataset's features that are considered the best overall. After eliminating the unneeded components from the dataset, we are left with the optimal subset of the dataset. Then we apply six classification techniques to the reduced dataset. 70% of the reduced dataset is used for training the models, and 30% is used for testing models. Fig. 2 depicts the overall architecture of the proposed BPSODroid. The algorithm for BPSOdroid is given in Table 2. The various BPSO parameters are given in Table 3.

Table 2.BPSOdroid Algorithm

S. No.	BPSODorid Algorithm
	Input: Dataset (DREBIN, MALGENOME, MENDELY)
	Output: Optimal subset
1.	Initialize BPSO parameters
2.	Generate Artificial particles (Population)

3. Iteration T =1
4. Do
5. FOR every individual particle I that makes up the swarm
6. Compute Fitness value
7. Evaluate velocity and position
8. IF the current fitness value is better than the Personal Best in history
9. Set current fitness value to Personal Best
10. END IF
11. END FOR
12. Update Global best-based on Personal Best
13. For each particle i in the swarm
14. Compute Velocity
15. Compute Position
16. END FOR
17. T=T+1
18. WHILE Maximum iteration reached
19. Convert the output to binary format using Eq. 4.
20. Create an Optimal dataset using selected features from BPSO
21. Create Training Dataset
22. Create Testing Dataset
23. Train ML models (RF, MLP, SVM, LR, DT, KNN)
24. Test Predictive model
25. Get Confusion Matrix
26. Evaluate the performance of each model

Table 3. The BPSO parameter setting

BPSO (Parameters)	BPSO(Parameter Values)

N	No of iteration =200
w_{incof}	inertia coefficient w_{incof}=0.3
c_{1f}	cognitive parameter (acceleration coefficient) c_{1f} =2.05
c_{2f}	social parameter (acceleration coefficient c_{1f} =2.05)
No. of particles (search space) (X)	215 for a dataset I and II
No. of particles (search space) (X)	301 for dataset III
$r_{1rand,}r_{2rand}$	a number between '0' and '1' selected at random
$rand\_n(\partial)$	a number between '0' and '1' chosen at random
$\vec{v}_i$	The initial velocity for all particles is set to zero.

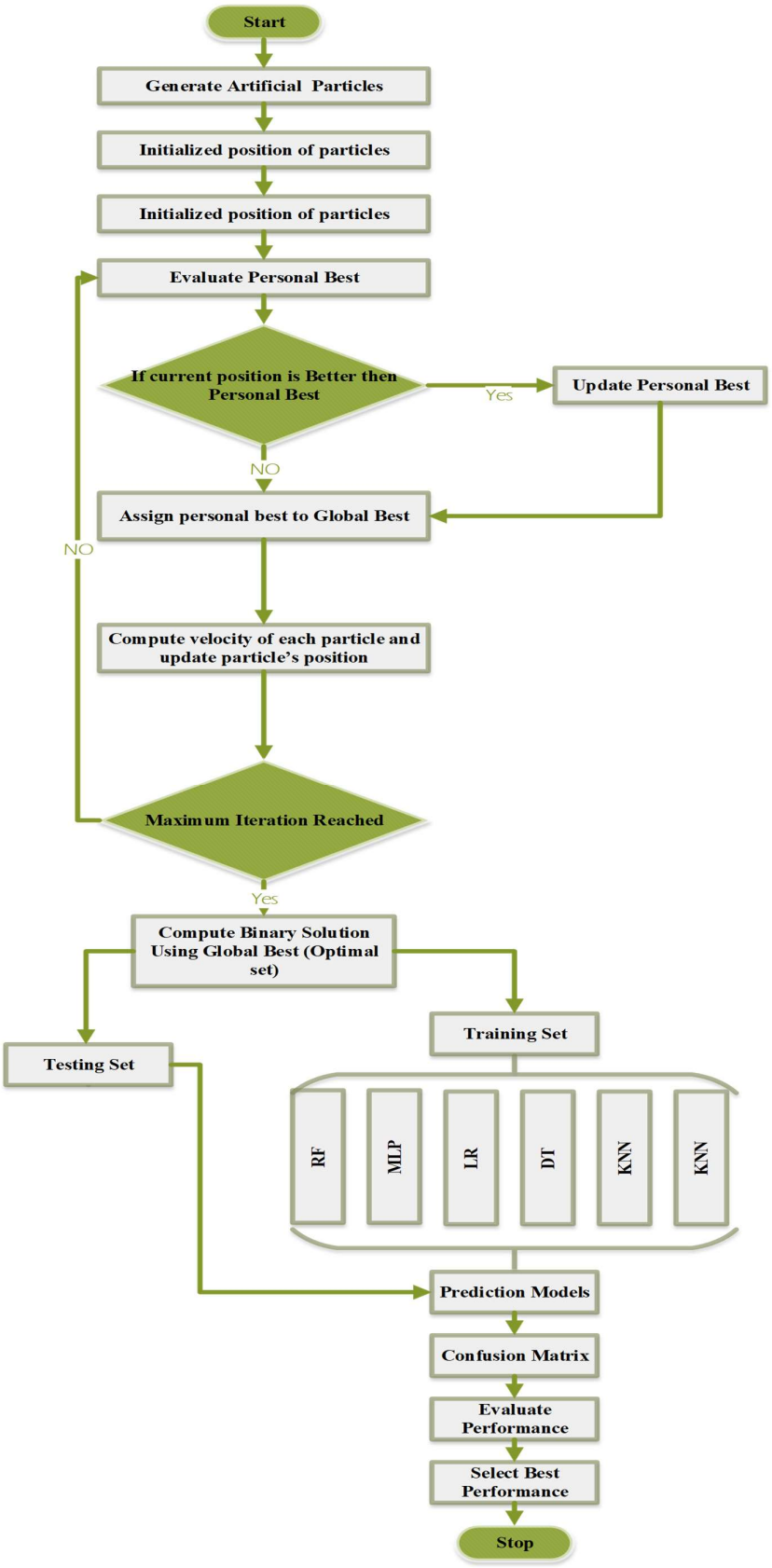

Figure 2. Architecture of BPSODroid

5. Datasets Description, Prepossessing and Experimental Setup:

We examined the proposed approach using three publicly available android datasets in this proposed work. A brief explanation of the datasets, their pre-processing steps, and the experimental environment is given in this section.

5.1 Employed Datasets

The investigation executed to estimate the proposed approach was made using three datasets. The first dataset (Malgenome-215) contains 3799 app samples, of which 2539 are benign apps, and 1260 malware samples were taken from the genome project [39]. The second one is the DREBIN-215 dataset, in which a total of 15036 app samples, of which 9476 are benign, and 5560 malware samples, were taken from the DREBIN Project [40]. The third dataset is the MENDELEY-301 dataset, consisting of 18,850 benign apps and 10,000 malware samples [41].

5.2 Data Pre-processing Steps

All entries are examined in the pre-processing data phase, and the duplicate instances are removed from the dataset. If an entry contains a NaN value, then such instances are removed from the dataset, and the constant value attributes are also removed from the dataset. The proposed BPSO method selects the most valuable features, removing unusable ones from the dataset. The dataset is reduced in size as a result of this process. The resulting dataset is then used for training and testing purposes.

5.3 Experimental environment

The experiments were done using Python 3.8 on a Jupyter notebook using the Anaconda platform. The implementation was done on a machine with an Intel (R) Core (TM) i-7 8550U @ 1. 80 GHz processor, 8 G.B. of RAM, and Windows 10 Home version 21H1.5.4

6. Performance Evaluation Matrix

The performance of the proposed method is evaluated using a confusion matrix. The confusion matrix is an N*N matrix that represents the performance of the classifiers using true positive, false positive, true negative, and false negative terms. The confusion matrix summarises the model's predictive performance and describes which classes are correctly and incorrectly classified.

A True Positive (TP) : A true positive represents the number of malware predicted by the classifier from all malware samples.

A False Positive (FP) : A false positive denotes the number of benign-ware samples predicted as malware.

A True Negative (TN) : A true negative represents the number of benign-ware samples detected as benign ware.

A False Negative (FN): A false negative denotes the number of malware samples detected as benign ware.

To evaluate the performance of the planned method, we used the following seven parameters: The confusion metrics drive the five parameters ranging from 6.1 to 6.5.

6.1 Recall (R):

Recall measures how well the model is appropriate for detecting events in the positive class. The recall is the number of relevant predictions relevant to the total number. Eq. (5) is used to calculate the recall value.

$$\text{Recall} = TP/(TP+FN) \tag{5}$$

6.2 Precision (P):

Precision is also known as positive predictive value. The precision is represented by the ratio of true positive predictions to the total predicted positive. The formulas of precision are given in Eq. (6).

$$\text{Precision} = TP/(TP+FP) \tag{6}$$

6.3 Accuracy

Accuracy is a metric that describes the performance of the classifier. This is useful when all classes are of equal significance. It is computed as the ratio between correct predictions and the total number of predictions. The classification accuracy is represented as Eq. (7).

$$\text{Accuracy} = (TP+TN)/(TP+TN+FP+FN) \tag{7}$$

6.4 F1-Score:

The weighted average of accuracy and recall is known as the f1-score. Therefore, both false positives and false negatives are considered for this score. It is not as simple as accuracy but is more valuable than accuracy for unequal class distributions. The F1-Score can be described as Eq. (8).

$$\text{F1Score} = 2(R*P)/(R+P) \tag{8}$$

6.5 MCC :

The Matthews Correlation Coefficient (MCC) measures the quality of binary classifications. Eq. (9) was used to calculate the MCC.

$$\text{MCC}=(\text{TN}*\text{TP}-\text{FN}*\text{FP})/\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)} \quad (9)$$

6.6 Mean Absolute Error :

The MAE is used to measure the prediction error in the classification problem. The MAE is used to calculate performance on continuous data. It gives a linear value that averages the weighted individual differences equally. The absolute difference ignores the negative value and is not sensitive to outliers. The MAE goes from 0 to 1, and values near 0 represent the best performance. Eq. (10) illustrates the Mean Absolute Error. Where y_i denotes the true value, n represents the total instance, and y_p represents the predicted value.

$$\text{MAE}=\frac{1}{n}\sum_{i=0}^{n}|yi - y_p| \quad (10)$$

6.7. Root Mean Squared Error :

The RMSE () represents the standard deviation of the residual errors. The errors are measured by subtracting the true value from predicted values. The errors are squared before they are averaged. The values near 0 indicate the better performance of the model. The Eq. (11) define the Root Mean Squared Error.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=0}^{n}(yi - y_p)\,2} \quad (11)$$

Where y_i denotes the true value, n represents the total instance, and y_p represents the predicted value.

7. RESULTS AND DISCUSSION:

To compare the performance of the proposed techniques with other existing systems, a total of six hybrid models were generated by combining the six classification methods with the BAPSO algorithm. These include Support Vector Machine (SVM), Random Forest (RF), K-Nearest Neighbors (KNN), Decision Tree (DT), Logistic Regression (LR), and the Multi-Layer Perception (MLP) classification method. Table 4 shows the precision, recall, TPR, F-1 score, MCC, MAE, RMSE, and weighted average accuracy, of all the applied classification techniques in the selected subset of all three datasets. It can be understood from Fig. 7. that the two variants of the proposed hybrid approaches, (BPSO + kNN), and (BPSO + SVM), have achieved higher accuracy than other applied approaches.

It is also clear from Table 4, and Fig. 8. that (BPSO + RF) performed better than other commonly used methods in terms of MCC. If the precision of detecting malware is compared, as per Table 4 and Fig. 3., (BPSO + LR) and (BPSO + LR) performed the best. The best recall values for malware detection were obtained in (BPSO+MLP) as shown in Fig. 4. As shown in Fig. 6., the BPSO+RF approach proved to be more effective with the least amount of error. Fig. 5. shows that the (BPSO+LR) and (BPSO+MLP) provide the highest F1-Score in malware detection. Fig. 8. compares MCC in three datasets and offers the highest MCC score obtained in the (BPSO+RF) variant.

Table 4. Performance of the proposed approaches

Datasets	Models	FS	B or M	Precision	Recall	F1-score	MCC	MAE	RMSE	Avg. Accuracy
I	BPSO+KNN	67	B	97	98	98	86.1	0.04	0.201	96
			M	92	85	88				
	BPSO+DT		B	91	92	92	84.4	0.047	0.217	90
			M	89	87	88				
	BPSO+LR		B	85	99	92	87.9	0.035	0.187	91
			M	99	85	91				
	BPSO+SVM		B	97	97	97	86	0.04	0.201	95
			M	86	89	87				
	BPSO+RF		B	94	96	95	93.1	0.02	0.142	94
			M	94	91	92				
	BPSO+MLP		B	93	98	95	89.9	0.029	0.172	95.4
			M	98	93	95				
II	BPSO+KNN	68	B	97	99	98	79.8	0.097	0.312	96
			M	95	85	90				
	BPSO+DT		B	93	94	94	87.1	0.062	0.249	93
			M	92	90	91				
	BPSO+LR		B	93	98	95	84.7	0.074	0.242	95
			M	98	93	96				
	BPSO+SVM		B	96	99	98	85.3	0.071	0.266	96
			M	95	82	88				
	BPSO+RF		B	92	96	94	86.5	0.065	0.255	93
			M	94	89	91				
	BPSO+MLP		B	82	99	89	82.9	0.083	0.288	89
			M	99	80	88				
III	BPSO+KNN	103	B	98	99	99	83.9	0.085	0.292	98
			M	96	93	94				
	BPSO+DT		B	94	95	94	90.4	0.048	0.22	93
			M	93	91	92				

	BPSO+LR		B	93	98	96	91.1	0.045	0.212	96
			M	98	94	96				
	BPSO+SVM		B	98	99	98	79.6	0.11	0.332	97
			M	93	90	92				
	BPSO+RF		B	93	92	93	91.4	0.043	0.208	92
			M	89	91	90				
	BPSO+MLP		B	94	98	96	91.3	0.043	0.208	96
			M	98	94	96				

The accuracy rate increase implies that the machine learning model's categorization is connected to the low positive rate. When identifying malware also provides an accurate result. However, the strong recall indicates that the malware traits are comparable to benign. Consequently, it was found that the findings show accuracy precision with a high recall value and that it is successful in identifying malware precisely.

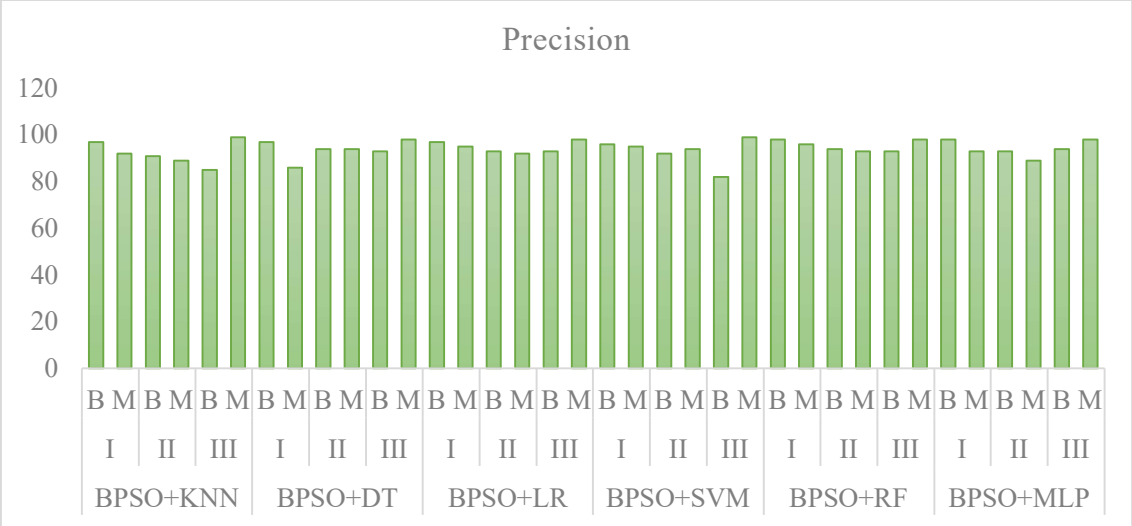

Figure 3. The obtained precision of all six variants in three datasets

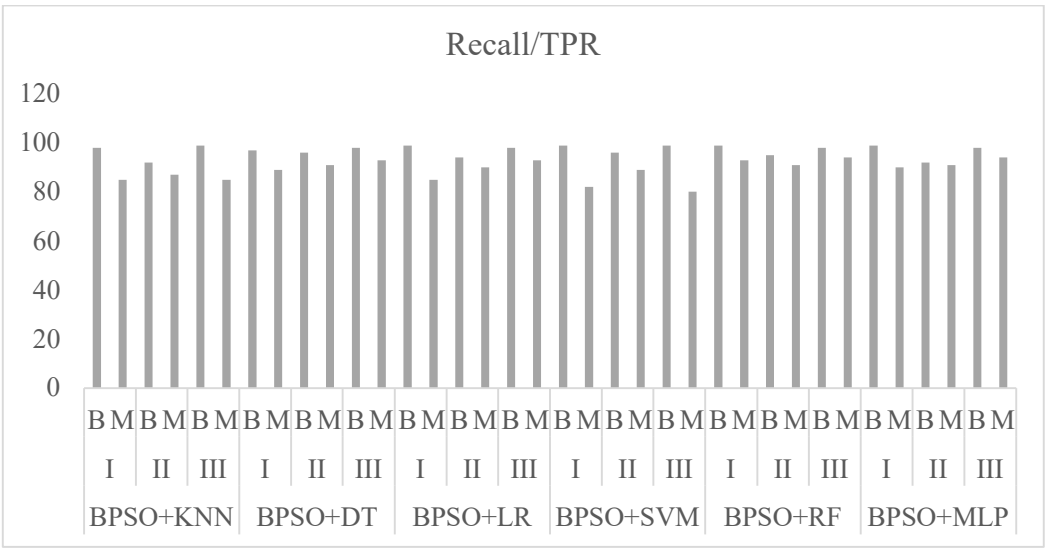

Figure 4. Obtained recall of all variants

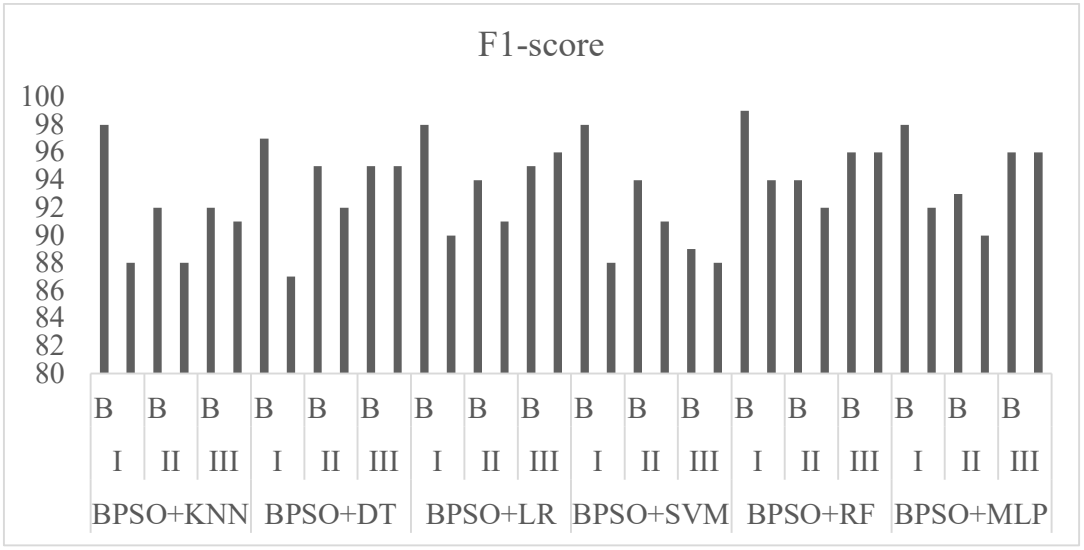

Figure 5. Obtained F1-Score of all applied hybrid

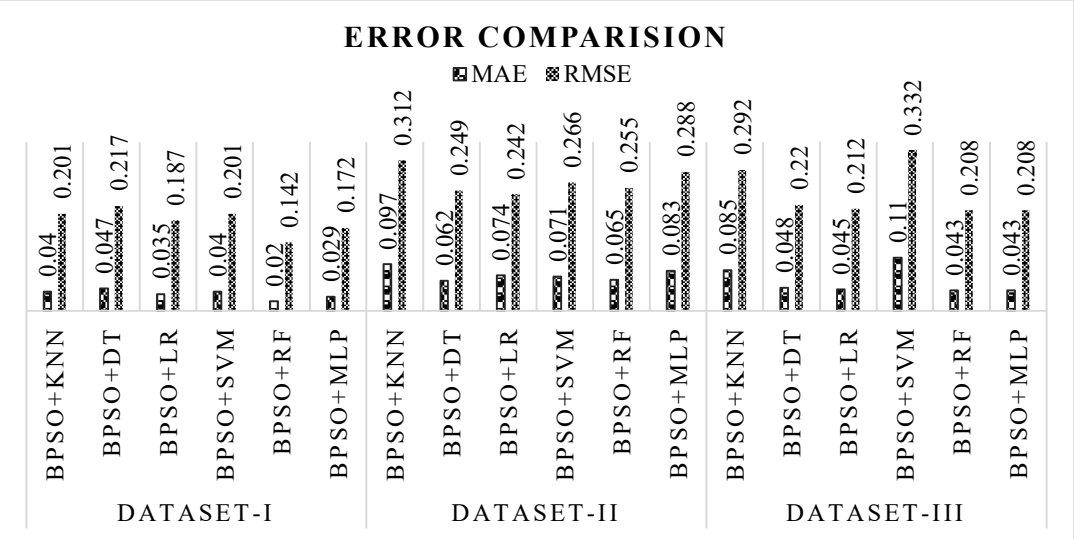

Figure 6. The comparison of MAE and RMSE in all reduced dataset

According to the trial results, kNN surpasses the other machine learning classifiers. This is because it builds a robust classifier from training data and then produces a second model to rectify errors in the first model. This technique is repeated until the training set is perfectly predicted. BPSO is an optimization technique that can yield promising results. Scanning an entire high-dimensional problem space is an efficient optimization approach. It seeks the optimal solution by distributing the particles according to their topology. It achieves the most outstanding performance in predicting malware utilizing permissions and other features by integrating kNN and the BPSO algorithm to attain 98 percent accuracy.

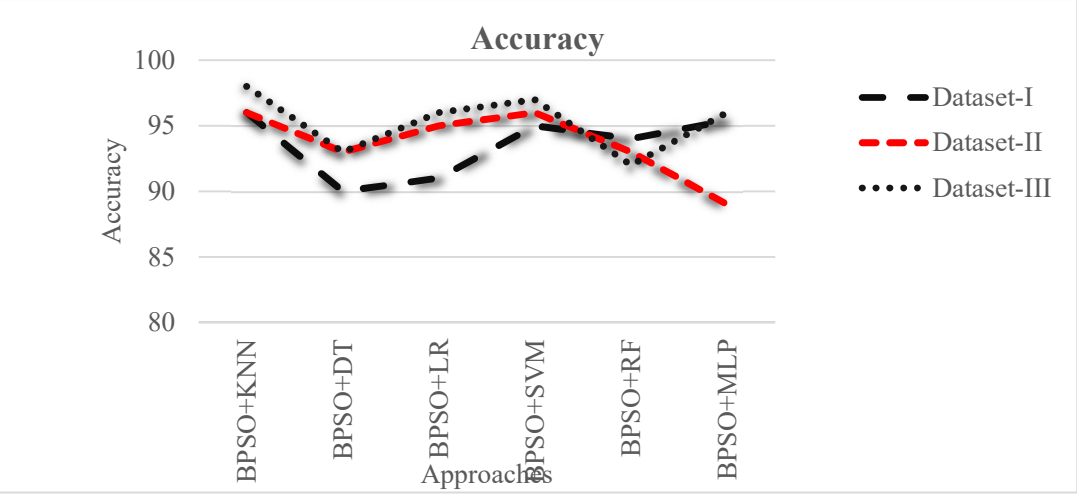

Figure 6. The comparison of accuracy in all applied techniques

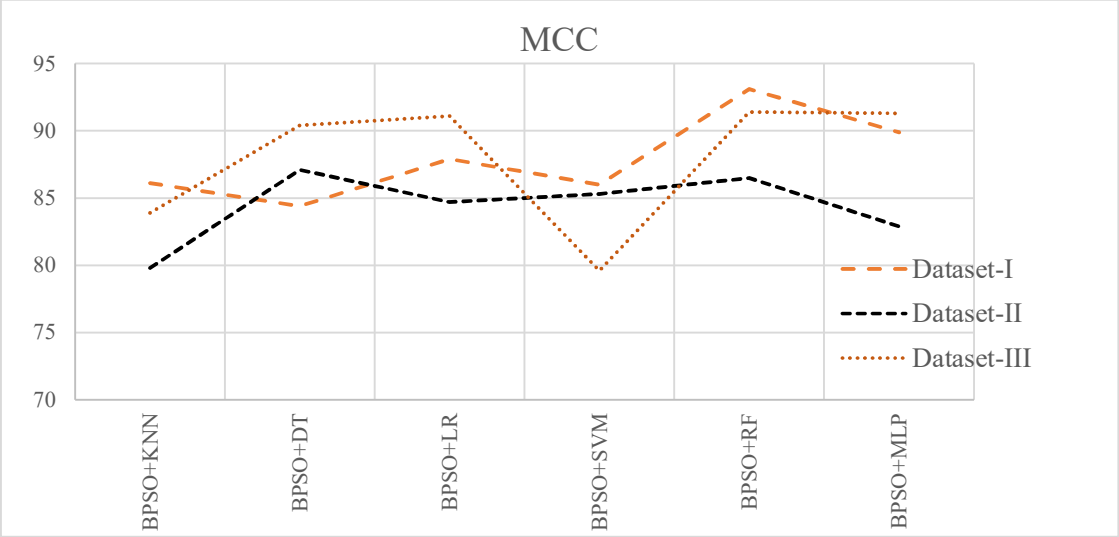

Figure 7. The comparison of MCC in all applied techniques

7.1 Comparison of the proposed approach with the earlier approach

The comparison of the proposed approach with the earlier methods is shown in Table 5. Except for [42] and [43], the performance of the suggested technique is better, as shown in Table 5. Heuristic algorithms do not re-examine previously covered pathways or actions since they always aim to obtain an ideal solution by learning from past steps. Instead, using the knowledge of the last steps, they search for new promising solutions and quickly find an optimal solution. Thus, one of the benefits of using meta-heuristic optimization is that they significantly reduce the size of the dataset by selecting the appropriate features, thereby reducing the detection time and complexity.

The proposed BPSO algorithm chooses 68 optimal features from the first dataset and 67 and 103 from the second and third datasets, respectively, which reduces the dataset size by more than 70%. Thus, it can be established that the proposed framework is more effective than many other existing approaches. Table 6 displays the top 20 potentially harmful behavior patterns chosen by the BPSO algorithm from the DREBIN dataset. Table 7 presents the twenty most risky behavior patterns chosen by the BPSO algorithm from the MALGENOME dataset. Table 8 displays the top 20 potentially hazardous behavior patterns chosen by the BPSO algorithm from the MENDELEY dataset.

Table 5. The performance comparison of the proposed approaches with the earlier methods.

Reference	Dataset	Approaches	Accuracy	F1-score	Recall/TPR	Precision
Proposed approach	MALGENOME DREBIN, MENDELEY	(BPSO and RF, KNN, MLP, SVM, DT, LR	98%	96%	94%	99%
[44]	DREBIN	SVM(Linear)	94%	__	__	__
	MALGENOME		95%			
[45]	DREBIN	SVM (2 class)	93.7%	__	__	__
[26]	DREBIN and Androzoo	PSO and MLP AdaBoost RF, KNN, J48	___	91.6	95.6%	__
[46]	DREBIN	PCA + RELIEF	95.2%	___	94.7%	__
[42]	DREBIN	(BNS + L-VM)	99.5%	___	99.6%	__
[43]	MALGENOME	ORGB	96.9%	97.09 %	98.55%	__
[47]	DREBIN	CNN	94.8%	___	93.6%	__
[35]	Self dataset	(GA +SVM) (GA+NN)	95% and 94.1%	__	__	__
[17]	MALGENOME	J48,KNN,SVM,NB	___	94.5%	94.5%	__
[48]	Self dataset	Gain Ratio + RF	91.7%	__	90.0%	__
[49]	Self dataset	CNN,RF SVM	95.7%	__	__	__

Table 5. Top 20 Dangerous Behavior patterns selected by BPSO from DREBIN Dataset

S.No.	Top 20 Dangerous Behavior (DREBIN DATSET)	Types
1	ServiceConnection	API call signature
2	Ljava.lang.Class.getCanonicalName	API call signature
3	Ljava.lang.Class.getMethods	API call signature
4	READ_PHONE_STATE	Manifest Permission
5	getBinder	API call signature
6	ClassLoader	API call signature

7	Landroid.content.Context.registerReceiver	API call signature
8	Landroid.content.Context.unregisterReceiver	API call signature
9	getCallingUid	API call signature
10	MANAGE_ACCOUNTS	Manifest Permission
11	SecretKey	API call signature
12	WRITE_SMS	Manifest Permission
13	android.telephony.gsm.SmsManager	API call signature
14	mount	Commands
15	INSTALL_PACKAGES	Manifest Permission
16	Runtime.getRuntime	API call signature
17	Ljava.lang.Object.getClass	API call signature
18	WRITE_SYNC_SETTINGS	Manifest Permission
19	android.intent.action.SEND_MULTIPLE	Intent
20	createSubprocess	API call signature

Table 6. Top 20 dangerous Behavior patterns selected by BPSO algorithm from MALGENOME dataset

S. No.	Dangerous Behavior Pattern	Types
1	transact	API call signature
2	attachInterface	API call signature
3	ServiceConnection	API call signature
4	android. Os.Binder	API call signature
5	Ljava.lang.Class.getMethods	API call signature
6	Class_Loader	API call signature
7	Ljava.lang.Class.getDeclaredField	API call signature
8	READ_SMS	Manifest Permission
9	Key_Spec	API call signature
10	DexClassLoader	API call signature
11	HttpGet.init	API call signature
12	Secret _Key	API call signature
13	System.load library	API call signature
14	android.telephony.gsm.SmsManager	API call signature
15	mount	Commands signature
16	INSTALL_PACKAGES	Manifest Permission
17	CAMERA	Manifest Permission
18	READ_HISTORY_BOOKMARKS	Manifest Permission
19	INTERNET	Manifest Permission
20	android.intent.action.PACKAGE_REPLACED	Intent

Table 1. Top 20 behaviour Pattern selected by BPSO from Mendely dataset

S. No	Dangerous Behavior Pattern	Types
1	Advanced download manager functions.	Manifest Permission
2	Read Google settings	API call signature
3	access checkin properties	API call signature
4	access to passwords for Google accounts	API call signature
5	INTERNET	Manifest Permission
6	android.intent.action.PACKAGE_REPLACED	Intent
7	Landroid.content.Context.registerReceiver	API call signature
8	Landroid.content.Context.unregisterReceiver	API call signature
9	enable or disable application components	Intent
10	force application to close	API call signature
11	force device reboot	API call signature
12	full Internet access	Manifest Permission
13	monitor and control all application launching	Manifest Permission
14	permission to install a location provider	API call signature
15	read frame buffer	Intent
16	read instant messages	API call signature
17	run in factory test mode	Intent
18	make all background applications close	API call signature
19	control vibrator	Manifest Permission
20	reset the system to factory defaults	Manifest Permission

8. Conclusion and future work

In this paper, we first determine the most dangerous behaviour pattern of android malware that is significantly responsible for malware attacks. The hazardous behaviour pattern of android malware is determined using the feature Selection mechanism of Binary particle swarm optimization (BPSO). The feature selection problem is described as an NP-hard problem with the challenge of picking a minimal-size subset of variables that contain all the information needed to create an ideally predictive model for a target variable. The BPSO algorithm identifies and eliminates unnecessary or detrimental pattern extraction characteristics to achieve better, quicker, and more intelligible data mining solutions. The BPSO is a stochastic optimization approach that identifies the optimum subset using swarm intelligence and mobility with minimal computational cost.

The BPSO optimizes a problem by iteratively improving a potential solution's quality. It solves a problem by moving candidate solutions, called particles, in the search space according to basic mathematical equations for location and velocity. Each particle's movement is directed by its local best-known location and the best-known positions in the search space, which are updated when better places are identified. The swarm should migrate toward the best solution. The proposed approach combines the BPSO with six machine learning techniques to find the complete solution for feature optimization and android malware detection. The proposed method achieved 98 % accuracy in malware detection, 96 % of the F1 score, and a recall rate of 94%. It also optimized the dangerous behaviour pattern by 70 % and outperformed in

comparison with several previous algorithms. The primary objective of future work will be to create other efficient hybrid models by utilizing deep learning techniques.

Reference

[1] "Smartphone users 2026 | Statista."https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/

[2] "Mobile OS market share 2021 | Statista."https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/

[3] "Google Play Store: number of apps 2021 | Statista." https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/

[4] "Malware Statistics & Trends Report | AV-TEST." https://www.av-test.org/en/statistics/malware/

[5] Joseph Johnson, "• Global Android malware volume 2020 | Statista," 2021. https://www.statista.com/statistics/680705/global-android-malware-volume/

[6] Y. Suleiman, S. Sezer, G. McWilliams, and I. Muttik, "New Android malware detection approach using Bayesian classification," in *Proceedings - International Conference on Advanced Information Networking and Applications, AINA*, 2013, pp. 121–128, 2013.

[7] F. Idrees, M. Rajarajan, M. Conti, T. M. Chen, and Y. Rahulamathavan, "PIndroid: A novel Android malware detection system using ensemble learning methods,"*Computers and Security*, vol. 68, pp. 36–46, 2017.

[8] A. Saracino, D. Sgandurra, G. Dini, and F. Martinelli, "MADAM: Effective and Efficient Behavior-based Android Malware Detection and Prevention,"*IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 1, pp. 83–97, 2018.

[9] R. Taheri, M. Ghahramani, R. Javidan, M. Shojafar, Z. Pooranian, and M. Conti, "Similarity-based Android malware detection using Hamming distance of static binary features,"*Future Generation Computer Systems*, vol. 105, pp. 230–247, 2020.

[10] Q. Wu, M. Li, X. Zhu, and B. Liu, "MVIIDroid: A Multiple View Information Integration Approach for Android Malware Detection and Family Identification,"*IEEE Multimedia*, vol. 27, no. 4, pp. 48–57,2020.

[11] K. Santosh Jhansi, S. Chakravarty, and R. K. P. Varma, "Feature Selection and Evaluation of Permission-based Android Malware Detection," in *Proceedings of the 4th International Conference on Trends in Electronics and Informatics, ICOEI 2020*, Jun. 2020, pp. 795–799, 2020.

[12] M. Zheng, M. Sun, and J. C. S. Lui, "DroidTrace: A ptrace based Android dynamic analysis system with forward execution capability," in *IWCMC 2014 - 10th International Wireless Communications and Mobile Computing Conference*, Sep. 2014, pp. 128–133, 2014.

[13] 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,"*Journal of Systems and Software*, vol. 159, 2020.

[14] M. Yang, X. Chen, Y. Luo, and H. Zhang, "An Android Malware Detection Model Based on DT-SVM,"*Security and Communication Networks*, vol. 2020, pp. 1–11, 2020.

[15] L. Sun, Z. Li, Q. Yan, W. Srisa-An, and Y. Pan, "SigPID: Significant permission identification for android malware detection,"*2016 11th International Conference on Malicious and Unwanted Software, MALWARE 2016*, pp. 59–66, Mar. 2017.

[16] S. Wang *et al.*, "Deep and Broad URL Feature Mining for Android Malware Detection,"*Information Sciences, Elsevier*, vol. 513, pp. 600–613, 2020.

[17] X. Jiang, B. Mao, J. Guan, and X. Huang, "Android Malware Detection Using Fine-Grained Features,"*Scientific Programming*, vol. 2020, no. 5190138, pp. 1–13, 2020.

[18] A Singh, Ajeet, and Anurag Jain. "Hybrid bio-inspired model for fraud detection with correlation based feature selection." Journal of Discrete Mathematical Sciences and Cryptography 24, vol. no. 5 (2021): pp. 1365-1374.

[19] M. A. Jerlin and K. Marimuthu, "A New Malware Detection System Using Machine Learning Techniques for API Call Sequences,"*Journal of Applied Security Research*, vol. 13, no. 1, pp. 45–62, 2018.

[20] A. Mehtab *et al.*, "AdDroid: Rule-Based Machine Learning Framework for Android Malware Analysis,"*Mobile Networks and Applications*, vol. 25, no. 1, pp. 180–192, 2020.

[21] M. K. Alzaylaee, S. Y. Yerima, and S. Sezer, "DL-Droid: Deep learning based android malware detection using real devices,"*Computers and Security*, vol. 89, Feb. 2020.

[22] A. Mahindru and A. L. Sangal, *SemiDroid: a behavioral malware detector based on unsupervised machine learning techniques using feature selection approaches*, vol. 12, no. 5. Springer Berlin Heidelberg, 2021.

[23] J. Feng, L. Shen, Z. Chen, Y. Wang, and H. Li, "A Two-Layer Deep Learning Method for Android Malware Detection Using Network Traffic,"*IEEE Access*, vol. 8, pp. 125786–125796, 2020.

[24] B. Selvakumar and K. Muneeswaran, "Firefly algorithm based feature selection for network intrusion detection,"*Computers and Security*, vol. 81, pp. 148–155, 2019.

[25] S. Alam, S. Alharbi, S. Y.-C. Networks, and undefined 2020, "Mining nested flow of dominant APIs for detecting android malware,"*Elsevier*, Accessed: May 02, 2022.

[26] M. F. A. Razak, N. B. Anuar, F. Othman, A. Firdaus, F. Afifi, and R. Salleh, "Bio-inspired for Features Optimization and Malware Detection,"*Arabian Journal for Science and Engineering*, vol. 43, no. 12, pp. 6963–6979, 2018.

[27] Y. Zhou, G. Cheng, S. Jiang, and M. Dai, "Building an efficient intrusion detection system based on feature selection and ensemble classifier,"*Computer Networks*, vol. 174, p. 107247, Jun. 2020.

[28] V. Ravindranath, … S. R.-2020 I. C., and undefined 2020, "Swarm intelligence based feature selection for intrusion and detection system in cloud infrastructure,"*ieeexplore.ieee.org*.

[29] R. Sihwail, K. Omar, K. A. Z. Ariffin, and M. Tubishat, "Improved Harris Hawks Optimization Using Elite Opposition-Based Learning and Novel Search Mechanism for Feature Selection,"*IEEE Access*, vol. 8, pp. 121127–121145, 2020.

[30] B. K. Khotimah, M. Miswanto, and H. Suprajitno, "Optimization of feature selection using genetic algorithm in naïve Bayes classification for incomplete data,"*International Journal of Intelligent Engineering and Systems*, vol. 13, no. 1, pp. 334–343, Feb. 2020.

[31] M. M. Mafarja and S. Mirjalili, "Hybrid Whale Optimization Algorithm with simulated annealing for feature selection,"*Neurocomputing*, vol. 260, pp. 302–312, Oct. 2017.

[32] S. W. Lin, K. C. Ying, S. C. Chen, and Z. J. Lee, "Particle swarm optimization for parameter determination and feature selection of support vector machines,"*Expert Systems with Applications*, vol. 35, no. 4, pp. 1817–1824, 2008.

[33] A. Firdaus, N. B. Anuar, A. Karim, and M. F. A. Razak, "Discovering optimal features using static analysis and a genetic search based method for Android malware detection,"*Frontiers of Information Technology and Electronic Engineering*, vol. 19, no. 6, pp. 712–736, 2018.

[34] P. Shunmugapriya and S. Kanmani, "A hybrid algorithm using ant and bee colony optimization for feature selection and classification (AC-ABC Hybrid),"*Swarm and Evolutionary Computation*, vol. 36, pp. 27–36, 2017.

[35] A. Fatima, R. Maurya, M. K. Dutta, R. Burget, and J. Masek, "Android malware detection using genetic algorithm based optimized feature selection and machine learning,"*2019 42nd International Conference on Telecommunications and Signal Processing, TSP 2019*, pp. 220–223, 2019.

[36] A. Bhattacharya, R. T. Goswami, and K. Mukherjee, "A feature selection technique based on rough set and improvised PSO algorithm (PSORS-FS) for permission based detection of Android malwares,"*International Journal of Machine Learning and Cybernetics*, vol. 10, no. 7, pp. 1893–1907, Jul. 2019.

[37] L. Wang, Y. Gao, S. Gao, and X. Yong, "A New Feature Selection Method Based on a Self-Variant Genetic Algorithm Applied to Android Malware Detection,"*Symmetry 2021, Vol. 13, Page 1290*, vol. 13, no. 7, p. 1290, Jul. 2021.

[38] H. Faris, M. Habib, I. Almomani, M. Eshtay, and I. Aljarah, "Optimizing Extreme Learning Machines Using Chains of Salps for Efficient Android Ransomware Detection,"*Applied Sciences 2020, Vol. 10, Page 3706*, vol. 10, no. 11, p. 3706, May 2020.

[39] "Android malware dataset for machine learning 1." https://figshare.com/articles/dataset/Android_malware_dataset_for_machine_learning_1/5854590/1.

[40] "Android malware dataset for machine learning 2." https://figshare.com/articles/dataset/Android_malware_dataset_for_machine_learning_2/5854653.

[41] "Research Data - Mendeley Data." https://data.mendeley.com/research-data/?type=DATASET&search=android malware.

[42] A. Kumar, S. C. D. Jaidhar, and M. A. A. Kumara, "Experimental analysis of Android malware detection based on combinations of permissions and API-calls,"*Journal of Computer Virology and Hacking Techniques*, vol. 15, no. 3, pp. 209–218, 2019.

[43] W. Zhang, H. Wang, H. He, P. L.-I. T. On, and U. 2020, "DAMBA: detecting android malware by ORGB analysis,"*ieeexplore.ieee.org*, vol. vol 69, no. 1, pp. 55–69, 2020.

[44] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck, “Drebin: Effective and Explainable Detection of Android Malware in Your Pocket,” *NDSS*, vol. 14, pp. 23–26, 2014.

[45] S. Lou, S. Cheng, J. Huang, and F. Jiang, "Tfdroid: Android malware detection by topics and sensitive data flows using machine learning techniques," in *2019 IEEE 2nd International Conference on Information and Computer Technologies, ICICT 2019*, pp. 30–36, 2019.

[46] "An Android malware detection system based on machine learning," vol. 020136, no.2018 2018.

[47] Y. Ding, X. Zhang, J. Hu, and W. Xu, "Android malware detection method based on bytecode image,"*Journal of Ambient Intelligence and Humanized Computing*, no. 0123456789, 2020.

[48] R. Thangaveloo, W. W. Jing, C. K. Leng, and J. Abdullah, "DATDroid: Dynamic analysis technique in android malware detection,"*International Journal on Advanced Science, Engineering and Information Technology*, vol. 10, no. 2, pp. 536–541, 2020.

[49] P. Yadav, N. Menon, V. Ravi, S. Vishvanathan, and T. D. Pham. "EfficientNet convolutional neural networks-based Android malware detection." Computers & Security vol. 115 , no.2022 pp.102622, 2022.