Fast and Robust Detection of Adversarial Attacks in the Problem Space using Machine Learning

MUHAMMAD LUQMAN NASEEM (mluqmannaseem@outlook.com)
Northeastern University https://orcid.org/0000-0002-0331-2347
Uroosa Sehar
University of Engineering and Technology

Research Article

Keywords: adversarial attack, machine learning, malware detection

Posted Date: March 3rd, 2022

DOI: https://doi.org/10.21203/rs.3.rs-1311205/v1

REVIEW

Fast and Robust Detection of Adversarial Attacks in the Problem Space using Machine Learning

Abstract

Machine learning is widely accepted as an accurate statistical approach for malware detection to cope with the rising uncertainty risk and complexity of modern intrusions. Not only has machine learning security been asked, but it has also been challenged in the past. However, it has been identified that machine learning contains intrinsic weaknesses that may be exploited to avoid detection during testing. So, look at it another way, machine learning can become an intelligence system bottleneck. We use the related attack methodology to classify different types of attacks using learning-based malware detection techniques in this research by evaluating attackers with unique abilities and talents. After this, to carefully identify the security of Drebin, Android malware detection has been performed. We implemented and did a set of comparable malware detection using the linear SVM and other relevant techniques, including Sec-SVM, Reduced SVM, Reduced Sec-SVM, Naïve Bayes, Random Forest Classifier, and some deep neural networks. The main agenda of this paper is the presentation of a scalable and straightforward secure-learning methodology that reduces the effect of adversarial attacks. In the presence of an attack, the detection accuracy is only a bit worsened. Finally, we evaluate that our robust technique may be accurately adapted to additional intrusion prevention tasks.

Keywords: adversarial attack; machine learning; malware detection

Introduction

Most recent research on adversarial ML promotes the domain direction where feature mapping is nondifferentiable or noninvertible. We change the problem space in this adversary requirement to prevent them from obstructing the feature space. The mapping feature is known as the inverse problem [1-4]. This research must be managed experiential and tracked to determine contingent best practices. Although they have become part of cohesions, they are fuzzy about how we associate them or which features they most utmost resemble in the flow of methodology to deal with the attacking mechanism. This evasion approach is also applicable to similar Android classifiers. Their problem-space transformation leaves two primary artifacts that should be detected through software assessment, in addition to the regional focus on API-based sequence features. However, hardcoding is no longer required for our attack and, with the help of the format, is robust by standard non-ML application evaluation methods. Problem-space attacks have been investigated in recent research efforts on the detection of adversarial ML attacks, given the rising diversity and current attacks, focusing on the development of benign and Malignant. Machine learning is extensively used as a statistically robust initiative for detecting malware. Machine learning has innate imperfections that can be used to evaluate during testing. Then, to adequately study and analyze Drebin's security, we create and implement a set of related evasion attacks, an Android malware detector. The availability of a scalable and straightforward secure-learning paradigm that alleviates the impact of evasion attacks is the already developed method. In the absence of an attack, the detection rate is only marginally worsened. According to the recent work, over 16,061,035 harmful malware apps and 119 new malware families have been detected.

The architecture of Drebin Malwares is featured schematically. We begin by extracting the feature and any associated malware, family, and labeling data. In a d-dimensional feature space, applications are represented as a vector. So after, an available set of labeled applications is used to train a linear SVM, Sec-SVM, Reduced SVM, Reduced Sec-SVM classifier to discriminate between malignant and benign applications. The classifier evaluates unseen applications during classification. They are categorized as malignant if the output is $f(x) \geq 0$, and benign otherwise. Drebin also explains its decision by accentuating the most suspicious or benign characteristic features that contributed to the decision. The problem may be represented geometrically inside an N-dimensional vector space. N represents the

Full list of author information is available at the end of the article

attributes of the set of all characteristics gathered from across all applications in the dataset. A program is an N-dimensional vector in this space:

$$v \in R^N \text{ s.t } v = \begin{cases} 1 \text{ if feature with index i appear in } v \\ 0 \text{ if feature with index idont appear} \end{cases}$$

This approach shows a program, the first index of all aspects from 1 to N. The problem solution is the hyperplane, which better divides the R^N space into two subspaces:

$$R^N = A \bigoplus B$$

As a result, the program x may be classified as come after according to this partition:

$$x = \begin{cases} \text{Malignant if } x \in A \\ \text{Benign if } x \in B \end{cases}$$

Now, we obtain the hyperplane equation:

$$y(x) = (\omega_1, \omega_2, \ldots, \omega_N)\, x + \omega_0 \tag{1}$$

It is simple to classify a new object:

$$x = \begin{cases} \text{Malignant if } y(x) > 0 \\ \text{Benign if } y(x) \leq 0 \end{cases}$$

As a result, it knows the weights ω_1 of the hyperplane is the first step in learning. There's an underlying assumption in this problem statement: the programs must be differentiable, which means we're assuming a line that correctly splits occurrences of the two classifications.

There are several techniques for determining the hyperplane coefficients ω_i to bisect the occasion in the training dataset. We will apply SVM in this implementation, which would be a method for determining the best value for each weight by maximizing the distance between the hyperplane and its closest point. This approach is reliable since it can discover the best answer even with outliers.

In this paper, we have worked step by step. In Section I, we have introduced the critical information about malware detection in problem space and defined its mathematical overview. Section II is about the motivation and contribution of our work. Section III is about the related works and state-of-the-art methods in this specific topic. After that, Section IV comprises the overall design methodology to experiment. Moreover, the performance matrices are evaluated and compared in this section more detailedly. Section V will explain the discussion about the result and conclusion of the overall work.

Table 1 Dataset arrangement for Drebin

Prefix		Feature set
feature	S1	Hardware components
permission	S2	Requested permission
activity service_receiver provider service	S3	App Components
intent	S4	Filtered Intent
api_call	S5	Restricted API calls
real_permission	S6	Uses Permission
call	S7	Suspicious API calls
URL	S8	Network addresses

The used Drebin dataset for malware recognition and prevention would have been to collect many features from such an application manifests and code, arrange and integrate those features inside a feature vector space where each characteristic is classified into sets, and eventually organize the program. Furthermore, using machine learning algorithms, patterns in these prefixes are detected. Feature, permission, activity, service receiver, provider, service, intent, API call, real permission, call, and URL are among properties gathered according to each program. Hardware components, requested permission, app components, filtered intents, restricted API calls, used permission, suspicious API calls, and network addresses have all been split into groups.

Motivation And Contribution

Malware is rising at an increasing rate [59], with millions of new attacks every month. Overall total number of malware attacks in 2013 was estimated to be over 30 million [60]. Many new malicious programs exceeded 94.6 million in February 2017 [61]. Such extraordinary rise is attributed to the simplicity with which harmful applications may be developed, mainly by packaging dangerous applications to create new variants. Even though there are significant variations in the estimations on daily malware attacks [60, 61], these are primarily in accordance. In 2013, new malware was estimated to be over 82k each day [60]. Researchers [62] predicted 250k unique malicious files every day in 2017. Android malware, several are dealt with security techniques have indeed been suggested. There seems to be an apparent demand for methods to protect mobile and IoT gadgets from malicious applications, including special needs to address the limits of current Android malware detection platforms. First importantly, the Android malware detection platform must have a high level of accuracy with such a minimal number of false alerts. Secondly, that should be able to manage on a diversity of deployment levels, including (1) server machines, (2) host computers, (4) IoT gadgets. and (3)

smartphones and tablets, Finally, identifying a malicious program would not be enough since more knowledge about the risk is expected to prioritize mitigating operations. Knowing what sort of attack was used might be essential in preventing the intended harm. As a result, having a solution that takes a step beyond and allocates the malware to such a particular family specifies the possible risk an infected system faces. Finally, manual human interaction should be minimized as often as feasible, with detection mainly focusing on the application samples for automatic extraction of features and patterns detection. The vulnerability team should keep up with the news of harmful programs growing stealthier. In this context, a manual assessment of such data is necessary with each new malware family to discover its structure and characteristics that distinguish it from benign programs.

Machine learning applications are becoming increasingly important in everyday life. On the other hand, such machine learning systems are subject to adversarial attacks. In our understanding, there was not a significant analysis of adversarial training that includes various forms of adversarial attacks and associated defenses. Such a limited set of machine learning [62] offered a detailed study on adversarial attacks against Machine learning. A few evaluations on information security of specific machine learning techniques have been published [63], [64], [65], [1]. Furthermore, our key emphasis is on malware detection using machine learning methods. We have also included detection scenarios using Support Vector Machines (SVM) due to their huge-scale use in real-world applications. The major contribution is as under

1. I have used unsupervised learning algorithms relevant to the SVM and its variation, compared their results, and found which is better to adopt.

2. Determining this perturbation is a bit trickier. We did malware detection using android apks from Drebin Dataset and found promising state-of-the-art accuracy by working more in pre-processing and encoding decoding.

Related Works

The researchers classified Android malware families depending on the graphic similarities between predictable and unpredictable Android apps. Some innovative and remarkable methodologies outperform established systems regarding malware classification. They mainly focused on extracting various characteristics and implemented several categorization methods. The similarity among malware families is quantified by the technique used carefully selected features that usually appeared in the same family and a machine learning approach.

Faced with the threat posed by malicious Android apps, an increasing percentage of scientists and researchers from across the world have been working on ways to recognize harmful software. The vital related approaches to Android malware family classification are discussed in this section. Several research activities focused on decoding the information from code explored a fine-grained family classification of Android malware to obtain better results on identifying malware.

Due to the various file types and parts inside the Android application package (Apk), scientific work on Android malware classification concentrates primarily on feature extraction and model optimization. Figure 1 presents the fundamental types. Appropriate characterization approaches can help increase classification accuracy and efficiency by mitigating run time and compute related costs. Prototype optimization can significantly help adjust algorithms and hyperparameters fit sample features, which can help to improve impacts. Most research currently focuses on program behavior characteristics, with API being among the most commonly used aspects. Dendroid [5] developed a text mining approach for analyzing the program code architecture of Android malware and categorizing it into families based on code architecture similarity. Droidminer [6] used static code analysis to automatically mining process malicious application code from recognized Android malware by transforming this into a sequence of threat patterns. Afterward, executing Android malicious app diagnosis and Android malware family classification based on whether the app carried out the correlating threat patterns. After that, some research works classified Android malware by features extracted from graphs and creating dependency graphs. Based on a supervised classification method and an unsupervised segmentation approach, EC2 [8] presented an Android harmful application family classification method. The API call execution sequence [9,10] aids in having understood the target attack application's behavior and thread information. Tao et al. [11] examined the importance of sensitive APIs shielded by privileges and extracted malware patterns from known malware specimens to efficiently identify malicious. The feature representation approaches can mutate malware identification into classifications, and then they can use modern techniques and models in the app to recognize and evaluate the application's malicious behaviors. The behavioral sequencing can be transformed into a dependency graph [12,13] or

Figure 1 Android Malware Categorization

even a flow graph [14,15] during malware detection. To control the family categorization of massive Android malware, [16] proposes a weighted-sensitive-API-call-based graph matching technique and a mechanism entitled FalDroid. The researchers of [17] proposed a method for classification malware based on common behavioral characteristics. Another model was used as the classifiers in Android-oriented Metrics, using a collection of Android-oriented coding metrics for static detection. The researchers of [18] extracted characteristics from source code and manifest files, such as systems API calls, hardware, authorization, and Android modules, and afterward utilized the vector support machine classifier as the final classification for Android malware feature extraction and classification. This method successfully divides Android malware samples into different-sized malicious applications families. The researchers of [15] collected features from the control-flow graph and data access graph, then used deep learning to build family classification algorithms. In [19], the researchers initiated a unique ground-truth dataset with features from 129013 benign applications, a total of 545334 features, and 5 560 compiled Android malware files from 79 families, using a multi-label classification algorithm to annotate the malicious functionality of the applicant malware attacks. Malicious applications belonging to almost the same malware family have similar behaviors carried out through numerous API calls.

On the other hand, most existing research focuses on evident characteristics that make it very difficult to express common harmful behaviors in such a malware family. Relying on vulnerable APIs for classification, Omid et al. [20] looked at the resemblance behaviors of various malware families. To attain family classification, Jiang et al. [21] examined the malicious program's specific semantic information, then built and converted sequences of vulnerable machine code into semantically similar vectors. Android malware detection and classification techniques are being developed to integrate and bagging to improve the model algorithm's robustness. The classification methods assist in analyzing the application's behavior sequences and mining the malicious proposed methodology. Sercan et al. [22] suggested a steady Android malware family classification scheme that uses many machine learning classification techniques. Techniques are mainly relevant to classify unfamiliar malware families (support vector machine, decision tree, logistic regression, K-nearest neighbor, random forest, AdaBoost, and multilayer Perceptron classifier).

Regarding Android detecting attacks and family identification, MVIIDroid [23] used a Multiple Viewing Information Integrated Approach. Permits, APIs, Dalvik opcodes, and native library opcodes were extracted as multiple kinds of characteristics, and the classification method used the Multiple Kernel Learning algorithm. Furthermore, as the quantity of new malware develops, linear classifiers become less adaptable, limiting the efficiency of Android malware classification.

[24] a unified optimization methodology for both evasion and poisoning attacks and a technical classification of their transferability. [25] The researcher believes the RSVM accuracy rate is comparatively lesser than standard SVM. [26] DREBIN performs similar techniques in an assessment using 123,453 programs and 5,560 malware samples, detecting 94% of the malware with fewer false errors. The purposes given with each detection expose important components of the detected malware. [27] Experiments were performed, and the outcomes illustrate that the proposed technique may be used to find universal adversarial perturbations across different classification machine learning models. [28] were the first to clearly describe and evaluate this weakness, showing when an attack could be mitigated exclusively by enhancing the learning algorithm's security or when additional aspects should be considered. [29] Many ideas for improving the robustness of neural network models against adversarial malware evasion attacks were standardized. [30] Researchers determined that "adversarial-malware as a service" is a plausible threat. However, we can gradually generate thousands of feasible and invisible adversarial apps at scale, with an average life span of only a few minutes. [31] Classify a well-established methodology in adversarial machine learning classified threats targeted against training PDF malware detection. [32] demonstrated that adversarial training in restricted domains can improve DNN robustness to these attacks. [33] Compared to state-of-the-art PDF malware detection techniques, researchers claim that their conventional abstract interpretation approach achieved similar accuracy, is much more flexible to evasion attacks, and gives accountable results. [34] The unique adversarial training vastly enhanced the robustness of deep learning models. The purpose of using a deep learning model is to use against a cyberattack, supervised learning enhances robustness even before classifiers are robust enough, and ensemble cyberattacks can effectively avoid the amplified malware detectors, perhaps devaluing the VirusTotal service. [35] have also seen an increase in interest in pursuing attack and defensive methods for Neural network models on several datasets, such as imagery, graphs, and text. [36] Their

Figure 2 Generic Flow of Methodology

investigation is the first to explain the actual issues of performing the start of the fall adversarial attacks within the cyber security industry, navigate them in a coherent classification, and utilize the classification system to identify future research prospects, as per the researchers. [37] suggested two new techniques that change a subset of sensed data by taking advantage of the learned physical limits. [38] In SoK research, it was proposed that ML is a suitable technique for learning insights. Those revelations give light to possible research areas in the future. [39] claimed that their function optimization approaches significantly outperform purely gradient-based approaches. [40] The actual issues of performing the start of the fall adversarial attacks in the cyber defense industry were discussed. Also, the classification was used to identify directions for future research. [41] Without learning how organizers constructed the adversarial cases, I used the defense mechanism to achieve the AICS'2019 Challenge. [42] Using techniques like hyperparameter optimization, it is recommended to generate an ensemble featuring profound models different from the attacker's expected framework (i.e., targeting model). [43] proved that the unique adversarial training greatly improves the robustness of deep learning models against a large variety of attacks. Ensembles encourage robustness when base classifiers seem to be robust sufficiently, and outfit attacks could successfully evade the improved malware detection, even scaling down the Malware detection service. [44] created a realistic adversarial example that can reliably mislead the opcode classifier to focused misclassification with such a 75% accuracy rate, an API classifier with such an 83.3% value, and the service function classifier with such a 91.7% value. [45] proposed an estimated certifying method for tree ensemble, which quickly evaluates the least accuracy on such a particular dataset without any need for time-consuming evasion attack calculations. [46] proposed an adaptable white-box threat that is conscious of the defense mechanism and tries to overcome it. [47] Moreover, they spoke about the restrictions of their methodology and how it may be improved in the future to attack malware classifications based on the dynamic assessment. [48] generated a series of problem-space modifications that produce UAPs in the appropriate feature-space embedding and analyzed their performance across attack models of various rates of actual attacker information. [49] According to the researchers, research findings on 12 families inside the Drebin dataset to the most samples show that, on average, program accuracy has varied from 0.976 to 0.989. Also, its robustness coefficient has enhanced from 0.857 to 0.917, illustrating the efficiency of the adversarial training method. [50] FamDroid could accurately classify 98.92% of malicious apps to different families, with such an F1- Score of 99.12%.

Design Methodology

To apply this first strategy, we will be using the LinearSVC SVM library by "scikit-learn." Scikit-learn is a Python package that efficiently implements different machine learning techniques. The LinearSVC class contains two primary methods: fit and predict. Fit is used to train the SVM using a provided dataset while predicting new instances.

Rather than considering one example at a time, the suitable technique accepts a batch of samples (X parameter) together with their correct classification (y parameter). The following are the values for these two parameters:

$$x_{M\times N} = \{x_{i,j}\} \text{ and } y \in R_M \tag{2}$$

Where M is the number of the training set, and N is the total number of features.

$$x_{i,j} = \begin{cases} 1 \text{ if i has feature index of } j \\ 0 \text{ else} \end{cases}$$

$$y_i = \begin{cases} 1 \text{ if malware is } i \\ 0 \text{ else} \end{cases}$$

The y axis reflects actual ground truth, which is the accurate classification (1 for malignant and 0 for non-malignant) for each application in the x matrices. The suitable technique modifies the SVM weights to accurately identify the program in the x matrices based on the value in the y vectors. Only the x matrices are sent to the predicted feature, which returns a y vectors with the data set (0 or 1) by each program in the x matrices.

We will use the following methods to create x and y matrices: Initially, we repeat across each program to implement a list of all features with no repetitions.

Next, we will use a dictionary with $< key|values >=< feature|index >$ to implement each features dataset quickly.

Finally, we will split the dataset into a training phase (actually containing 70% of total programs) and a testing dataset (usually includes 30% of overall programs) that provides 30% of comprehensive programming. Then we will extract all features with each

Figure 3 Top 20 Families in the Drebin Dataset

training scheme, and if application i has the feature j, we will set the $x_{i,j}$ Component of the x matrices to

1. The i^{th} component of the y vectors will be set to 1 or 0 if the i programs emerge in the sha256 families.csv file (which contains all types of malware with the appropriate family). When we provide input feature sets showing new features to label, the malware detectors must preserve the indexing list of features to create new x matrices.

Although the dataset encompasses a set of attributes and the number of features for each type could be unequal, we will not classify or preprocess them, even if intuition says some are much more helpful than others. The shreds of evidence will not influence our assumption gathered from data by the program, which may find connections that we hadn't considered. Another critical aspect to remember would be that this algorithm is not a black box; we could see which features were the most significant in choosing. This is due to the hyperplane's shape, or "discriminant function":

$$y(x) = w_1x_1 + w_2x_2 + \cdots + w_Nx_n + \omega_0$$

Follow as x is Classified:

$$x = \begin{cases} \text{Malignant if } y(x) > 0 \\ \text{Benign if } y(x) \leq 0 \end{cases}$$

This indicates that features with such a more excellent value of w assisted the most in classifying an application as malware. As per this feature, to describe the classifications, we will list the k highest weighted features, then calculate the intersection between both the app's features and the k highest weighted features for each application classified as Malicious. The features that assisted the most in classifying that application as malicious are the outcomes of the intersection. Now, let's discuss how to choose the value of k (the number of most weighted features to select), a hyper-parameter in our approach. By experimenting with different numbers of k and comparing results, we can see that:

Suppose k is too few relatives to the total amount of features. In that case, the intersection between the features of x as well as the k highest weighted features for a feature of x identified as malware may be empty, making the algorithm is unable to describe why x was classified as malicious. Experiments confirmed that this happens when $k < 500$ since there are 545327 distinct features.

If k is too high, the intersection will have too many features, and the description will be insufficient.

A figure of k of approximately 1,000 is hardly too less nor too high, and it will give high accuracy if used for family classifications, as we will see from the following section. As previously stated, to classify the program p, we should answer the following equation.

$$\text{argmax}_f\{P(f \mid p)\} = \text{argmax}_f \left\{\Pi_{i=1}^m P\left(s_i \mid f\right) * P(f)\right\} \tag{3}$$

In computing the equation above, the method must study all of the terms on the right side among all features and families. We will be looking at how the algorithms can learn each of them on a deeper level.

The first phase is the possibility of a specific feature provided by the family, calculated as the number of malicious apps inside the dataset with the feature s_i of family f divided by the total of malicious in the dataset also with feature s_i of family f. First, this word is known as "prospect" in Bayes terminology, as well as the algorithm should study the prospect for each selection of features during training.

$foreach family f$:

$foreach features$:

$$P(s \mid f) = \frac{\text{No. of malwares with features s of family } f}{\text{number of malwares of familiy } f}$$

The second term is also known as "prior probability," and it is computed by dividing the numbers of malignant in family f by the actual quantity of malicious apps in the dataset. For each family, the algorithm should learn the "prior probability" as tries to follow:

$foreach family f$:

$$P(f) = \frac{\text{number of malwares of family } f}{\text{number of malwares in dataset}}$$

The family of a program P could be evaluated once algorithms have learned all of the other terminologies:

$$f = \text{argmax}_f \left\{\Pi_{i=1}^m P\left(s_i \mid f\right) * P(f)\right\} \tag{4}$$

Where m is the number of features in program P, i.e., $P = s_1, \ldots, s_m$. Now, let us look at possible modifications to understand how the algorithm is implemented. The first step will be to reduce features that represent a program to a realistic size. Recall that we calculated the set of its most "dangerous" attributes in the detecting section, i.e., the collection of all the features assisted the most in detecting a program as malicious. Furthermore, we computed the set of harmful features as that of the intersection of the program's feature set with the setting of all dangerous features for a specific program.

As a result, rather than representing a program with all its aspects, we will describe it in this following part with a set of the most harmful aspects. The application is assumed to be malicious to classify the family - unless it's a false positive, i.e., a harmless software identified as malware by our detection technique - thus, applying the selected features with one of the most dangerous features set is not a fault. This choice

was reached for 2 purposes: one is time efficiency, as the equation calculation above needs fewer repetitions with a smaller number of features. The second reason is that precision is essential as only features that helped identify the application as malicious must be used to classify its families, not the "benign" features found in both malicious and benign applications. The second suggestion would be that families with very few malicious apps are not considered because the algorithms require many samples of each family to identify instances of that families accurately. As a result, families with few then N programs will be reduced to a single "unidentified" family. We can see that with more than samples were taken from each family, the classification is accurate enough with experimenting with new values of N.

DREBIN requires a comprehensive yet lightweight representation of apps to identify malware on a smartphone, allowing it to determine typical indicators of malicious activity. Our approach accomplishes this by applying a widespread static analysis that extracts feature sets across many sources and evaluates them in an expressive vector space. This procedure is visualized in Figure 2 and described in detail.

Feature Extraction: In the first step, DREBIN inspects an Android application statically and extracts different feature sets from the manifest and dex code.

Class Labels: The extracted feature sets are then delineated to a combined vector space, which allows geometrical analysis of patterns and combinations of features.

Training: We are supposed to train the model using a different variation of SVM such as Linear SVM, Sec-SVM, Reduced SVM, Reduced Sec-SVM, etc.

Results: The last phase identifies features that contribute to detecting a malicious application and presents them to the user to explain the detection process.

Our Secure SVM methodology (Sec-SVM) is defined as follows:

$$\min_{w,b} \frac{1}{2} w^T w + C \sum_{i=1}^{n} \max\left(0, 1 - y_i f\left(x_i\right)\right) \tag{5}$$

$$\text{s.t } w_k^{lb} \leq w_k \leq w_k^{ub} k = 1, \ldots, d \tag{6}$$

The matrices specify the lowest and highest bounds for w. $w^{lb} = \left(w_1^{lb}, \ldots, w_d^{lb}\right)$ and $w^{ub} = \left(w_1^{ub}, \ldots, w_d^{ub}\right)$ In the following, we will show the constraint given in Equation.(5) as $w \in W \subseteq R^d$ for mathematical notation simplification. The algorithm flow figure illustrates the meaningful learning algorithm. It is a restricted Stochastic Gradient Descent (SGD) that uses a simple line-search technique to smooth the gradient step size throughout the optimization. SGD is a trivial gradient-based method for effective learning with very large-scale datasets [56], [57]. It functions by approaching the sub-gradients of the objective function to use a single data point or a tiny subset of the training data, which is randomly selected at each loop. In our scenario, the target function sub-gradients (Equation. (6)) are as described in the following:

$$\nabla_\omega \mathcal{L} \cong \omega + C \sum_{i \in S} \nabla_l^i x_i \tag{7}$$

$$\nabla_b \mathcal{L} \cong C \Sigma_{i \in S} \nabla_l^i \tag{8}$$

As S is the subset of the training dataset that used to calculate the estimate, while li seems to be the hinges loss's gradient concerning $f\left(x_i\right)$, that equals if y_i, if $y_i f\left(x_i\right) < 1$,0 or 1 elsewhere. The starting gradients phase margin$\eta^{(0)}$ and a suitable decaying function $s(t)$, i.e., a function that systematically lowers the gradient scale factor during the optimization procedure, are two critical issues for assuring fast convergence of SGD. These factors must be determined based on preliminary testing on a subset of the training phase, as described in [56], [57]. Linear and decaying exponential functions are popular alternatives for function $s(t)$. We finalize this part by emphasizing that our approach is pretty standard; various failure and regularization functions can improve different safe variations of other linear classification algorithms. As with the traditional SVM [58], our Sec-SVM learning algorithm is just an example that considers the hinge failure and l_2 Regularization. Also, it is essential to note that when the high and low boundaries decrease in actual number, our approach tends to provide (dense) results with weights equivalent to the upper or lower factor.

Figure 4 Flow of Sec-SVM

When decreasing the l_∞ directly, a corresponding result is achieved [59]. We concluded from our study that security and sparsity had had an implicit trade-off: although a sparse learning method ensures an efficient illustration of the acquired decision function, it can be easily overcome by manipulating several features. A secure learning method, on either hand, relies on the availability of many, perhaps redundant, qualities that make evading the decision function more difficult, but at the cost of a dense representation. The flow of

Figure 5 Variation of the values using of Sec SVM

Methodology-Comparison of Linear SVM and RSVM With such a parameter variable, we study the support vector machine (SVM). Given a classification of feature groups for training,$(x_i, y_i), i = 1, \ldots, l$ wherever $x_i \in R^n$ and $y_i \in \{-1, 1\}$ The proposed optimization issue is solved using the support vector methodology:

$$\begin{aligned} &\min_{w,b,\varepsilon} \frac{1}{2}\omega^T\omega + C\left(\sum_{i=1}^{l}\varepsilon_i^2\right) \\ &\text{Subject to } y_i\left(\omega^T\phi(x_i) + b\right) \geq 1 - \varepsilon_i \end{aligned} \tag{9}$$

Since, $\phi(x)$ will be mapped into finite-dimensional space. First, we deal with a quadratic problem with l as several variables.

$$\begin{aligned} &\min_{\alpha} \frac{1}{2}\alpha^T\left(Q + \frac{1}{2C}\right)\alpha - e^T\alpha \\ &\text{Subject to } y^T\alpha = 0 \\ &0 \leq \alpha_i i = 1, \ldots, l, \end{aligned} \tag{10}$$

The critical difference between linear SVM and RSVM in [23]. They begin by including a second term, $b^2/2$, in the fitness function of (19):

$$\min_{w,b,\varepsilon} \frac{1}{2}\left(\omega^T\omega + b^2 + C\left(\sum_{i=1}^{l}\varepsilon_i^2\right)\right) \tag{11}$$

$$\text{Subject to } y_i\left(\omega^T\phi(x_i) + b\right) \geq 1 - \varepsilon_i \tag{12}$$

Its corresponding form is a more fundamental bound-constrained threat:

$$\begin{aligned} &\min_{\alpha} \frac{1}{2}\alpha^T\left(Q + \frac{1}{2C} + yy^T\right)\alpha - e^T\alpha \\ &\text{Subject to } 0 \leq \alpha_i i = 1, \ldots, l \end{aligned} \tag{13}$$

The technique of summing up $b^2/2$ and subsequently getting a bounded duality was presented in [52] and [53]. This is the same as applying a constant attribute to the instruction objective of finding a hyperplane that separates going through the source. When $b^2/2$ is introduced, numerical studies in [54] indicate that the precision does not change substantially.

It is well known because ω seems to be a linear combination of training the model at the best solution:

$$\omega = \sum_{i=1}^{l} y_j\alpha_i\phi(x_i) \tag{14}$$

If we use ω instead of (21), we get

$$\begin{aligned} y_i\omega^T\phi(x_i) &= \sum_{j=1}^{l} y_i y_j\alpha_j\phi(x_j)^T\phi(x_j) \\ &= \sum_{j=1}^{l} Q_{ij}\alpha_j = (Qa)_i, \text{ and} \\ \omega^T\omega &= \sum_{i=1}^{l} y_i\alpha_j\phi(x_j)^T\omega \end{aligned} \tag{15}$$

$$\begin{aligned} \omega^T\omega &= \sum_{i=1}^{l} y_i\alpha_j\phi(x_j)^T\omega \\ &= \sum_{i=1}^{l} \alpha_j(Qa)_i = \alpha^T Qa \end{aligned} \tag{16}$$

As a result, we have a new optimization problem:

$$\begin{aligned} &\min_{a,b,\varepsilon} \tfrac{1}{2}\left(a^T Qa + b^2 + C\left(\textstyle\sum_{i=1}^{l}\epsilon_i^2\right)\right) \\ &\text{Subject to } Q\alpha + by \geq e - \varepsilon \end{aligned} \tag{17}$$

Due to the fact of (9) is not the same as (5), for dual problems, we can prove that for every solution α of (9), the associated ω : given by (6) is also an appropriate solution of (3), permitting us to solve (9) rather than (5). [55] Would be the one to apply the transformation of (3) to (9).

RSVM is such a technique for reducing the number of support vectors. It is accomplished by selecting a subsequence of m samples at probability sampling for the construction of ω :

$$\omega = \sum_{i \in R} y_i\alpha_j\phi(x_j) \tag{18}$$

Where R is a list of index values for this subset. Putting (10) in (4), would give us the same problem as in (9), and the significant variable like α is changed into m

$$\begin{aligned} &\min_{\bar{\alpha},b,\varepsilon} \frac{1}{2}\left(\bar{\alpha}^T Q_{RR}\bar{\alpha} + b^2\right) + C\left(\sum_{i=1}^{l}\epsilon_i^2\right) \\ &\text{Subject to } Q_{:R}\bar{\alpha} + \text{ by } \geq e - \varepsilon \end{aligned} \tag{19}$$

here $\bar{\alpha}$ would be the set of all $\alpha_i, and, i\epsilon R$. We have to keep in mind that m is the size of the Real number (R).

We can evaluate the sub-matrix of the column concerning R in $Q_{:,R}$.Subsequently, we still have l number of constraints. In generalized SVM, [23] has clarified

Figure 6 Variation of the values using of Reduced Sec SVM

the same term $\frac{1}{2}\bar{\alpha}^T Q_{RR}\bar{\alpha}$ to $\frac{1}{2}\bar{\alpha}^T\bar{\alpha}$. So, it solves the RSVM.

$$\min_{\bar{\alpha},b,\varepsilon} \frac{1}{2}\left(\bar{\alpha}^T\bar{\alpha} + b^2\right) + C\left(\sum_{i=1}^{l}\varepsilon_i^2\right) \quad \text{Subject to } Q_{:,R}\bar{\alpha} + by \geq e - \varepsilon \tag{20}$$

Performances Evaluation: The number of programs successfully classified among the overall number of programs is evaluated by accuracy. This metric is insufficient to assess the machine learning technique, especially when the two unbalanced classifications. Because there are much more benign applications than malware in this scenario, the algorithm might achieve high accuracy even if it classified all applications as benign.

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

As a result, we've included the accuracy metric, which shows how often we can rely on the algorithm when it identifies an application as malware. There are few false positives, so precision is high.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

The recall measures the ability to detect malware. If there are few false negatives, recall is high.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

The false-positive rate is a metric that shows how susceptible the algorithms are to making errors when classifying benign applications as malware. If there are few false positives, the false positive rate is low.

$$\text{False Positive Rate} = \frac{\text{FP}}{\text{FP} + \text{TN}}$$

To measure the result, we separated the dataset into two parts: training test (70% of the dataset) and testing sets (30% of the dataset). The dataset is a collection of malware attacks that were correctly classified. The performance is calculated as the number of well-classified malware divided by the total number of malware:

$$\text{True Positive rate} = \frac{\text{true prediction}}{\text{total predictions}}$$

The following is the definition of the confusion matrix:

$$M = \{a_{ij}\} \text{ s.t. } a_{ij} \tag{21}$$

=number of malwares from the family i classified as malwares from the family j

Results: To apply the algorithm, we assume that splitting of the dataset, i.e., is divided into two parts: the training data set, which contains 70% of the overall programs, and also the testing sets, which includes the remaining programs. The procedure to be followed will be collected after computing the predicted values y for the training dataset and comparing them to true values. *True Positives:* malwares accurately classified as malwares *True Negatives:* programs that are considered benign *False Positives:* Malicious programs are classified as benign programs. *False Negatives:* Malware that has been labeled as harmless. Where percentages of successfully classified programs (true positive rate and true negatives rate) first diagonal, while the numbers of misclassified programs second diagonal (starting to the left bottom, we have false positives and false negatives).Other indicators useful for computing algorithm performance are listed in table 2. The number of precise predictions is represented by tracing the confusion matrix, whereas the sum of other components represents the number of wrong predictions. That's the algorithm's confusion matrix:

Figure 7 Ground Truth Table positive and negative ratios of the Malware

Figure 8 Variation of the Malware Confusion Matrix

Table 2 Results evaluation using SVM

Dataset	Accuracy	Precision	Recall	FPRate	TPRate
DREBIN	99.35%	94.03%	91.70%	0.25%	89.21%

This study deeply evaluated the state-of-the-art methods to deal with this problem. Typically, the followed methods are mainly SVM, Sec-SVM, Reduced SVM, Reduced Sec-SVM. After implementing all the algorithms, the results are computed as below. Currently, I have these results you can see

Figure 9 Comparison of the values using SVM, Sec-SVM, Reduced-SVM, Reduced-Sec-SVM

Table 3 Comparison of Different Techniques

Techniques	SVM	Sec-SVM	RSVM	RSSVM
Accuracy (%)	**99.35%**	99.12%	98.56%	98.42%

Table 4 Comparison between SVM, Random Forest Classifier, Naïve Bayes and Deep Learning using Drebin Dataset

The Classifier	Accuracy	Precision	Recall	F1 Score
SVM	**99.35%**	94.03%	91.70%	92.33%
RFC	98.79%	**98.79%**	78.50%	84.40%
Naïve Bayes	92.23%	92.23%	46.29%	34.21%
Deep Learning	91.12%	91.00%	**96.00%**	**94.00%**

Now, let us briefly compare SVM with the supervised learning algorithm using a deep neural network. As SVM was bounded in the concept of autoencoder, the Deep learning classifier always looks into the true labels and computes the better classification on the test dataset samples. Figure 8 explains how both learning methods are different from each other. The blue, orange, grey, and yellow bars represent and every performance metric like accuracy, precision, recall, and

Figure 10 Comparison between SVM, Random Forest Classifier, Naïve Bayes and Deep Learning using Drebin Dataset

f1 score, respectively, using SVM, deep learning model, RFC, and Naïve Bayes. The DNN was computed, taking advantage of the convolution neural network.
The input is calculated and reshaped using convolution layers and the pooling layer, and then it is estimated in the fully connected network layer. Finally, SoftMax regression computes the classes based on the specified probability. Therefore, we can say that unsupervised learning can give us promising results, and the experimental values can also be seen in Table 4. Also, we compared the result using other techniques like Naïve Bayes and Random Forest Classifiers.

Conclusion

This paper presents a comprehensive overview of different techniques for classifying Android malware families throughout this research. We perform many experiments and determine that our methodology has significant benefits in the detection of Android malware families. As our research focuses on malware detection, SVM has an accurate result of 99.35%, precision at 94.03%, and a recall score at 91.70% than other representations. Machine learning algorithms, on either hand, learn to detect malware via samples and thus may be more helpful in detecting malicious activity according to their more generic method. Recent studies in the fields of adversarial machine learning with computer security show that machine learning may take a keen interest in vulnerability in such a security system, thus mitigating the fundamental security of the system. The source of the problem would be that machine-learning methods were not designed to deal with knowledgeable and adapting attackers who could change their behavior to mislead learning and classification methods. RSVM will be faster than the regular SVM on significant problems or challenging cases with many support vectors. The SVM has been researched in terms of multi-class implementations. Experiments demonstrate that perhaps the accuracy rate of the RSVM is slightly lower than the traditional SVM. Although the RSVM complies with almost the same constraints as such primary form of SVM, reducing support vectors to a random selection subset reduces performance. Based on actual implementation techniques, we demonstrate that RSVM would be better than standard SVM on significant or unexpected issues with several support vectors than standard SVM on substantial issues or perhaps some difficult cases with several support vectors.
In conclusion, standard SVM should be utilized for moderate challenges, although RSVM would be an attractive method for significant problems. Still, it can appropriately limit the number of learning support vectors. When dealing with secure learning methods, Sec SVM could focus on the availability of numerous, perhaps redundant, features that make evading the decision function more complex but at the expense of a density representation. We expect such research to inspire interest in additional research towards adversarial malware detection. Possible future research areas also include exciting features of various adversarial malware samples, including the search for effective cyber-attacks, robust feature extraction, harmful feature assessment, defense evaluation metrics, and the design of much more strong robust defenses.

Author details

References

1. Biggio, B., Fumera, G., Roli, F. (2017). Security Evaluation of Pattern Classifiers under Attack. https://doi.org/10.1109/TKDE.2013.57.
2. Association for Computing Machinery., ACM Conference on Computer and Communications Security (18th: 2011: Chicago, Ill.). (n.d.). AISec'11: proceedings of the 4th ACM Workshop Security and Artificial Intelligence: October 21, 2011, Chicago, Illinois, USA.
3. Quiring, E., Maier, A., Rieck, K., & Braunschweig, T. U. (n.d.). Misleading Authorship Attribution of Source Code using Adversarial Learning. https://www.usenix.org/conference/usenixsecurity19/presentation/quiring
4. Biggio, B., Corona, I., Maiorca, D., Nelson, B., Nedim˘, N., Nedimšrndi´c, N., Laskov, P., Giacinto, G., Roli, F. (n.d.). LNAI 8190 - Evasion Attacks against Machine Learning at Test Time. http://prag.diee.unica.it/
5. Suarez-Tangil, G., Tapiador, J. E., Peris-Lopez, P., Blasco, J. (2014). Dendroid: A text mining approach to analyzing and classifying code structures in Android malware families. Expert Systems with Applications, 41(4 PART 1), 1104–1117. https://doi.org/10.1016/j.eswa.2013.07.106
6. Yang, C., Xu, Z., Gu, G., Yegneswaran, V., Porras, P. (2014). LNCS 8712 - DroidMiner: Automated Mining and Characterization of Fine-grained Malicious Behaviors in Android Applications.

7. Zhang, M., Duan, Y., Yin, H., Zhao, Z. (2014). Semantics-aware Android malware classification using weighted contextual API dependency graphs. Proceedings of the ACM Conference on Computer and Communications Security, 1105–1116. https://doi.org/10.1145/2660267.2660359
8. Chakraborty, T., Pierazzi, F., Subrahmanian, V. S. (2020). EC2: Ensemble Clustering and Classification for Predicting Android Malware Families. IEEE Transactions on Dependable and Secure Computing, 17(2), 262–277. https://doi.org/10.1109/TDSC.2017.2739145
9. Onwuzurike, L., Mariconti, E., Andriotis, P., de Cristofaro, E., Ross, G., Stringhini, G. (2019). Mamadroid: Detecting android malware by building Markov chains of behavioral models (extended version). ACM Transactions on Privacy and Security, 22(2). https://doi.org/10.1145/3313391
10. Sun, Y. S., Chen, C. C., Hsiao, S. W., Chen, M. C. (2018). ANTSdroid Automatic malware family behaviour generation and analysis for Android apps. Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 10946 LNCS, 796 804. https://doi.org/10.1007/978-3-319-93638-3-48
11. Tao, G., Zheng, Z., Guo, Z., Lyu, M. R. (2018). MalPat: Mining Patterns of Malicious and Benign Android Apps via Permission-Related APIs. IEEE Transactions on Reliability, 67(1), 355–369. https://doi.org/10.1109/TR.2017.2778147
12. Ikram, M., Beaume, P., Kaafar, M. A. (2019). DaDiDroid: An Obfuscation Resilient Tool for Detecting Android Malware via Weighted Directed Call Graph Modelling. http://arxiv.org/abs/1905.09136
13. Zhao, B. (n.d.). MAPPING SYSTEM LEVEL BEHAVIORS WITH ANDROID APIS VIA SYSTEM CALL DEPENDENCE GRAPHS.
14. Xu, Z., Ren, K., Qin, S., Craciun, F. (n.d.). CDGDroid: Android Malware Detection Based on Deep Learning using CFG and DFG.
15. Xu, Z., Ren, K., Song, F. (n.d.). Android Malware Family Classification and Characterization Using CFG and DFG.
16. Fan, M., Liu, J., Luo, X., Chen, K., Tian, Z., Zheng, Q., Liu, T. (2018). Android malware familial classification and representative sample selection via frequent subgraph analysis. IEEE Transactions on Information Forensics and Security,13(8),1890–1905. https://doi.org/10.1109/TIFS.2018.2806891
17. Kim, H. M., Song, H. M., Seo, J. W., Kim, H. K. (2019). Andro-Simnet: Android Malware Family Classification Using Social Network Analysis. https://doi.org/10.1109/PST.2018.8514216
18. Li, C., Mills, K., Niu, D., Zhu, R., Zhang, H., Kinawi, H. (2019). Android Malware Detection Based on Factorization Machine. IEEE Access, 7, 184008–184019. https://doi.org/10.1109/ACCESS.2019.2958927
19. Mirzaei, O., Suarez-Tangil, G., de Fuentes, J. M., Tapiador, J., Stringhini, G. (2019). AndrEnsemble: Leveraging API ensembles to characterize android malware families. AsiaCCS 2019 - Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security, 307–314. https://doi.org/10.1145/3321705.3329854
20. Qiu, J., Zhang, J., Luo, W., Pan, L., Nepal, S., Wang, Y., Xiang, Y. (2019). A3CM: Automatic Capability Annotation for Android Malware. IEEE Access, 7, 147156–147168. https://doi.org/10.1109/ACCESS.2019.2946392
21. Jiang, J., Li, S., Yu, M., Li, G., Liu, C., Chen, K., Liu, H., Huang, W. (n.d.). Android Malware Family Classification Based on Sensitive Opcode Sequence.
22. Wu, Q., Li, M., Zhu, X., Liu, B. (2020). MVIIDroid: A Multiple View Information Integration Approach for Android Malware Detection and Family Identification. IEEE Multimedia,27(4),48–57. https://doi.org/10.1109/MMUL.2020.3022702
23. Lee, Y.-J., Mangasarian, O. L. (n.d.). RSVM: Reduced Support Vector Machines. https://epubs.siam.org/page/terms
24. Demontis, A., Melis, M., Pintor, M., Jagielski, M., Biggio, B., Oprea, A., Nita-Rotaru, C., Roli, F. (n.d.). Why Do Adversarial Attacks Transfer? Explaining Transferability of Evasion and Poisoning Attacks.
25. Lin, K.-M., Lin, C.-J. (n.d.). A Study on Reduced Support Vector Machines.
26. Arp, D., Spreitzenbarth, M., Hübner, M., Gascon, H., Rieck, K. (2014). DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket. http://dx.doi.org/doi-info-to-be-provided-later
27. Cheng, J., Tang, X., Liu, X. (n.d.). Proceedings. http://www.springer.com/series/7410
28. Maiorca, D., Demontis, A., Biggio, B., Roli, F., Giacinto, G. (2017). Adversarial Detection of Flash Malware: Limitations and Open Issues. https://doi.org/10.1016/j.cose.2020.101901
29. Li, D., Li, Q., Ye, Y., Xu, S. (2018). Enhancing Robustness of Deep Neural Networks Against Adversarial Malware Samples: Principles, Framework, and AICS'2019 Challenge. http://arxiv.org/abs/1812.08108
30. Pierazzi, F., Pendlebury, F., Cortellazzi, J., Cavallaro, L. (2019). Intriguing Properties of Adversarial ML Attacks in the Problem Space. http://arxiv.org/abs/1911.02142
31. Maiorca, D., Biggio, B., Giacinto, G. (2019). Towards adversarial malware detection: Lessons learned from PDF-based attacks. ACM Computing Surveys, 52(4). https://doi.org/10.1145/3332184
32. Chernikova, A., Oprea, A. (2019). FENCE: Feasible Evasion Attacks on Neural Networks in Constrained Environments. http://arxiv.org/abs/1909.10480
33. Mardziel, P., Vazou, N., Association for Computing Machinery. Special Interest Group on Security, A., ACM Conference on Computer and Communications Security (26th: 2019: London, E. (n.d.). PLAS'19: proceedings of the 14th Workshop on Programming Languages and Analysis for Security: November 15, 2019, London, United Kingdom.
34. Li, D., Li, Q. (2020). Adversarial Deep Ensemble: Evasion Attacks and Defenses for Malware Detection. IEEE Transactions on Information Forensics and Security, 15, 3886–3900. https://doi.org/10.1109/TIFS.2020.3003571
35. Xu, H., Ma, Y., Liu, H. C., Deb, D., Liu, H., Tang, J. L., Jain, A. K. (2020). Adversarial Attacks and Defenses in Images, Graphs and Text: A Review. In International Journal of Automation and Computing (Vol. 17, Issue 2, pp. 151–178). Chinese Academy of Sciences. https://doi.org/10.1007/s11633-019-1211-x
36. Rosenberg, I., Shabtai, A., Elovici, Y., Rokach, L. (2020). Adversarial Machine Learning Attacks and Defense Methods in the Cyber Security Domain. http://arxiv.org/abs/2007.02407
37. Erba, A., Taormina, R., Galelli, S., Pogliani, M., Carminati, M., Zanero, S., Tippenhauer, N. O. (2020). Constrained Concealment Attacks against Reconstruction-based Anomaly Detectors in Industrial Control Systems. ACM International Conference Proceeding Series, 480–495. https://doi.org/10.1145/3427228.3427660
38. Li, D., Li, Q., Ye, Y., Xu, S. (2020). Arms Race in Adversarial Malware Detection: A Survey. http://arxiv.org/abs/2005.11671
39. Verwer, S., Nadeem, A., Hammerschmidt, C., Bliek, L., Al-Dujaili, A., O'Reilly, U. M. (2020). The Robust Malware Detection Challenge and Greedy Random Accelerated Multi-Bit Search. AISec 2020 - Proceedings of the 13th ACM Workshop on Artificial Intelligence and Security, 61–70. https://doi.org/10.1145/3411508.3421374
40. Xu, W., Qi, Y., & Evans, D. (2017, May 13). Automatically Evading Classifiers: A Case Study on PDF Malware Classifiers. https://doi.org/10.14722/ndss.2016.23115
41. Li, D., Li, Q., Ye, Y., Xu, S. (2020). A Framework for Enhancing Deep Neural Networks Against Adversarial Malware. https://doi.org/10.1109/TNSE.2021.3051354
42. Shu, R., Xia, T., Williams, L., Menzies, T. (2020). Omni: Automated Ensemble with Unexpected Models against Adversarial Evasion Attack. http://arxiv.org/abs/2011.12720
43. Demontis, A., Melis, M., Biggio, B., Maiorca, D., Arp, D., Rieck, K., Corona, I., Giacinto, G., & Roli, F. (2017). Yes, Machine Learning Can Be More Secure! A Case Study on Android Malware Detection. http://arxiv.org/abs/1704.08996
44. Rouseev, V., Thuraisingham, B., Association for Computing Machinery. Special Interest Group on Security, A., Association for Computing Machinery. (n.d.). CODASPY 20: proceedings of the Tenth ACM Conference on Data and Application Security and Privacy: March 16-18, 2020, New Orleans, LA, USA.
45. Calzavara, S., Lucchese, C., Marcuzzi, F., Orlando, S. (2020). Feature Partitioning for Robust Tree Ensembles and their Certification in Adversarial Scenarios. http://arxiv.org/abs/2004.03295
46. Sotgiu, A., Demontis, A., Melis, M., Biggio, B., Fumera, G., Feng, X., Roli, F. (2020). Deep neural rejection against adversarial examples. Eurasip Journal on Information Security, 2020(1). https://doi.org/10.1186/s13635-020-00105-y
47. Demetrio, L., Biggio, B., Lagorio, G., Roli, F., Armando, A. (2020).

Functionality-preserving Black-box Optimization of Adversarial Windows Malware. https://doi.org/10.1109/TIFS.2021.3082330
48. Labaca-Castro, R., Muñoz-González, L., Pendlebury, F., Rodosek, G. D., Pierazzi, F., Cavallaro, L. (2021). Universal Adversarial Perturbations for Malware. http://arxiv.org/abs/2102.06747
49. Wang, C., Zhang, L., Zhao, K., Ding, X., Wang, X. (2021). Advandmal: Adversarial training for android malware detection and family classification. Symmetry, 13(6). https://doi.org/10.3390/sym13061081
50. fan, W., Zhao, L., Wang, J., Chen, Y., Wu, F., Liu, Y. (2021). FamDroid: Learning-Based Android Malware Family Classification Using Static Analysis. http://arxiv.org/abs/2101.03965
51. Friee, T.-T., Cristianini, N., Campbell, C. (n.d.). The Kernel-Adatron Algorithm: A Fast and Simple Learning Procedure for Support Vector Machines.
52. Mangasarian, O. L., Musicant, D. R. (1999). Successive Overrelaxation for Support Vector Machines. In IEEE TRANSACTIONS ON NEURAL NETWORKS (Vol. 10, Issue 5).
53. Hsu, C.-W. (2002). A Simple Decomposition Method for Support Vector Machines (Vol. 46).
54. Osuna, E., Girosi, F. (n.d.). Reducing the run-time complexity of Support Vector Machines.
55. Proceedings of COMPSTAT'2010. (2010). In Proceedings of COMPSTAT'2010. Physica-Verlag HD. https://doi.org/10.1007/978-3-7908-2604-3
56. Zhang, T. (2004). Solving large scale linear prediction problems using stochastic gradient descent algorithms. Twenty-First International Conference on Machine Learning - ICML '04. doi:10.1145/1015330.1015332
57. Cortes, C., Vapnik, V., Saitta, L. (1995). Support-Vector Networks Editor. In Machine Leaming (Vol. 20). Kluwer Academic Publishers.
58. S. Boyd and L. Vandenberghe, Convex Optimization. Cambridge, U.K.: Cambridge Univ. Press, 2004.
59. Malware Statistics and Trends Report, AV-TEST, https://www.av-test.org/en/statistics/ malware. Accessed Jan 2020.
60. Report: Average of 82,000 new malware threats per day in 2013. https://www.pcworld.com/article/2109210/report-average-of-82-000-new-malware-threats-per-day-in-2013.ht ml. Accessed March 2014.
61. Symantec Intelligence for February 2017, https://www.symantec.com/connect/blogs/latestintelligence-february-2017. Accessed Oct 2017 New Malware Variants Near Record-Highs: Symantec, http://www.securityweek.com/newmalware-variants-near-record-highs-symantec . Accessed March 2017
62. Akhtar, N., Mian, A. (2018). Threat of Adversarial Attacks on Deep Learning in Computer Vision: A Survey. In IEEE Access (Vol. 6, pp. 14410–14430). Institute of Electrical and Electronics Engineers Inc. https://doi.org/10.1109/ACCESS.2018.2807385
63. Barreno, M., Nelson, B., Joseph, A. D., and Tygar, J. D. (2010). The security of machine learning. Machine Learning, 81(2), 121–148. https://doi.org/10.1007/s10994-010-5188-5
64. Barreno, M., Nelson, B., Sears, R., Joseph, A. D., and Tygar, J. D. (2006). Can Machine Learning Be Secure?
65. Corona, I., Giacinto, G., and Roli, F. (n.d.). Adversarial Attacks against Intrusion Detection Systems: Taxonomy, Solutions and Open Issues. http://faculty.cs.tamu.edu/guofei/sec

Figure1

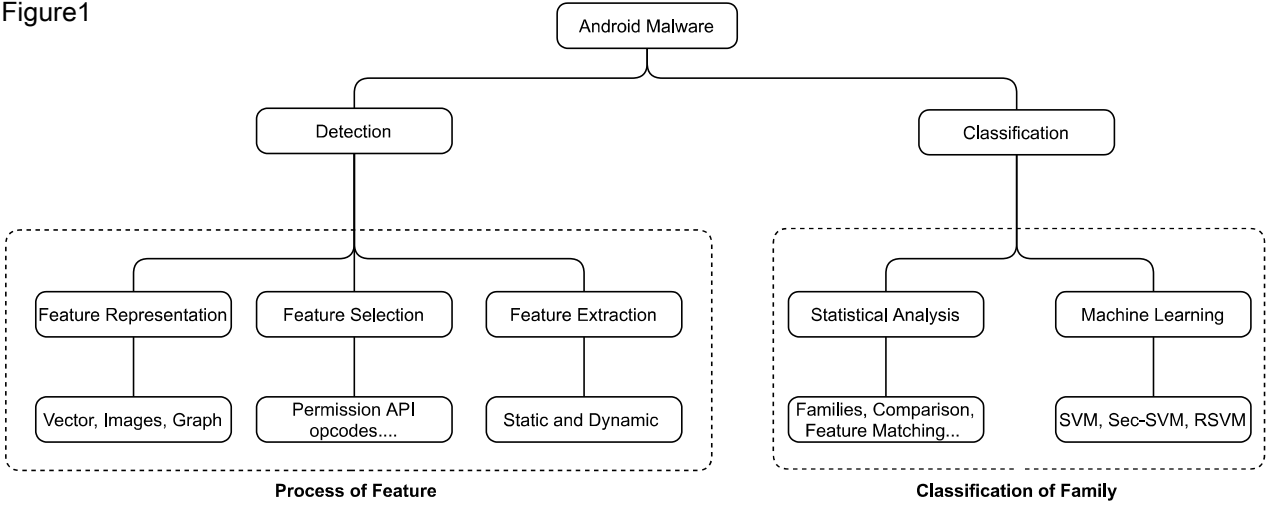

Android Malware
Detection
Classification
Feature Representation
Feature Selection
Feature Extraction
Statistical Analysis
Machine Learning
Vector, Images, Graph
Permission API opcodes....
Static and Dynamic
Families, Comparison, Feature Matching...
SVM, Sec-SVM, RSVM
Process of Feature
Classification of Family

Figure2

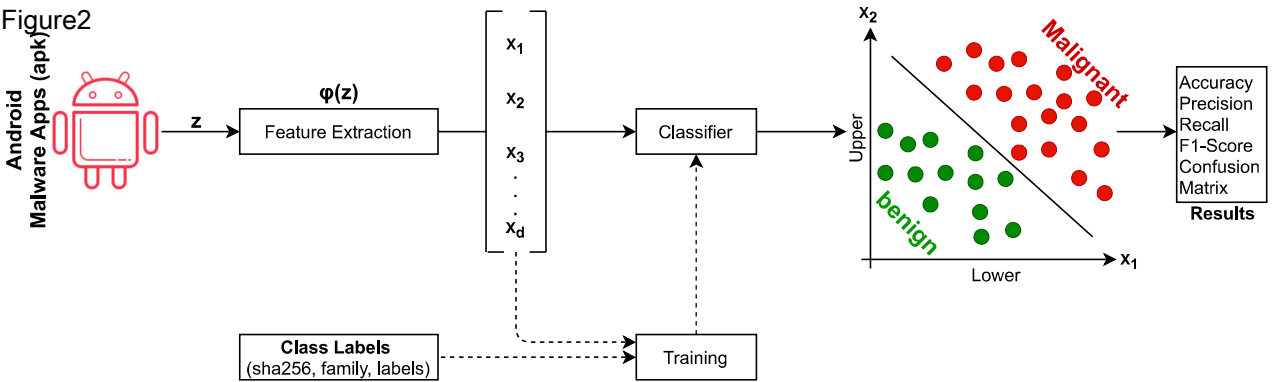

Android
Malware Apps (apk)
z
φ(z)
Feature Extraction
x1
x2
x3
xd
Classifier
x2
Upper
Lower
x1
Malignant
benign
Accuracy
Precision
Recall
F1-Score
Confusion
Matrix
Results
Class Labels
(sha256, family, labels)
Training

Figure3

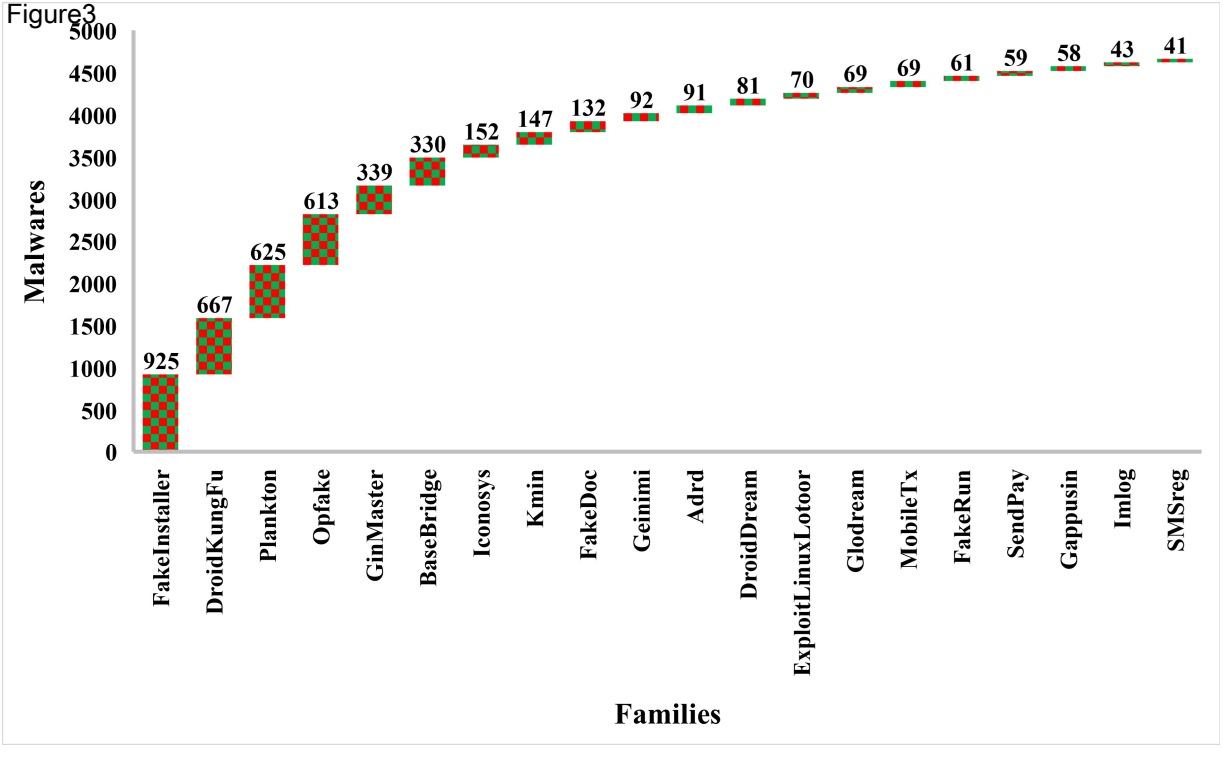

Malwares
5000
4500
4000
3500
3000
2500
2000
1500
1000
500
0
925
667
625
613
339
330
152
147
132
92
91
81
70
69
69
61
59
58
43
41
FakeInstaller
DroidKungFu
Plankton
Opfake
GinMaster
BaseBridge
Iconosys
Kmin
FakeDoc
Geinimi
Adrd
DroidDream
ExploitLinuxLotoor
Glodream
MobileTx
FakeRun
SendPay
Gappusin
Imlog
SMSreg
Families

Figure4

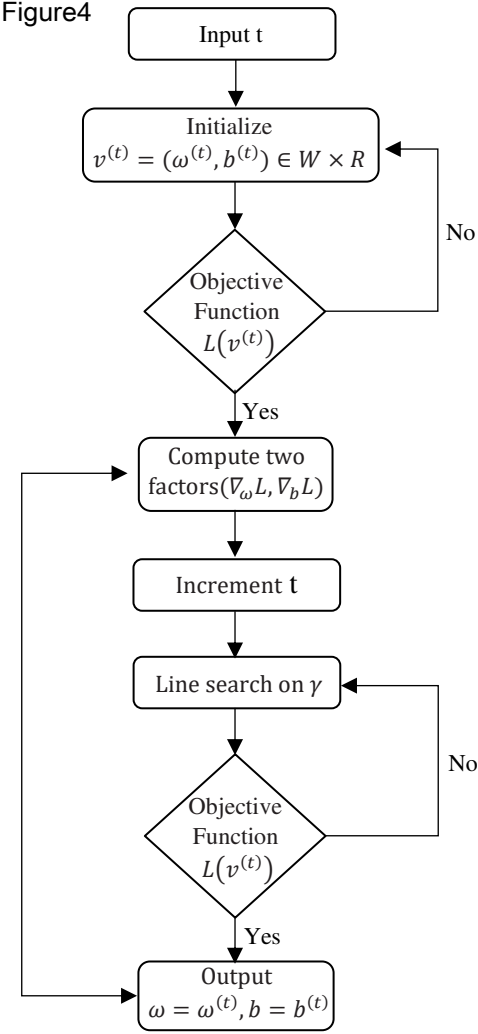

Input t
Initialize
$v^{(t)} = (\omega^{(t)}, b^{(t)}) \in W \times R$
No
Objective Function
$L(v^{(t)})$
Yes
Compute two factors$(\nabla_\omega L, \nabla_b L)$
Increment t
Line search on γ
No
Objective Function
$L(v^{(t)})$
Yes
Output
$\omega = \omega^{(t)}, b = b^{(t)}$

Figure5

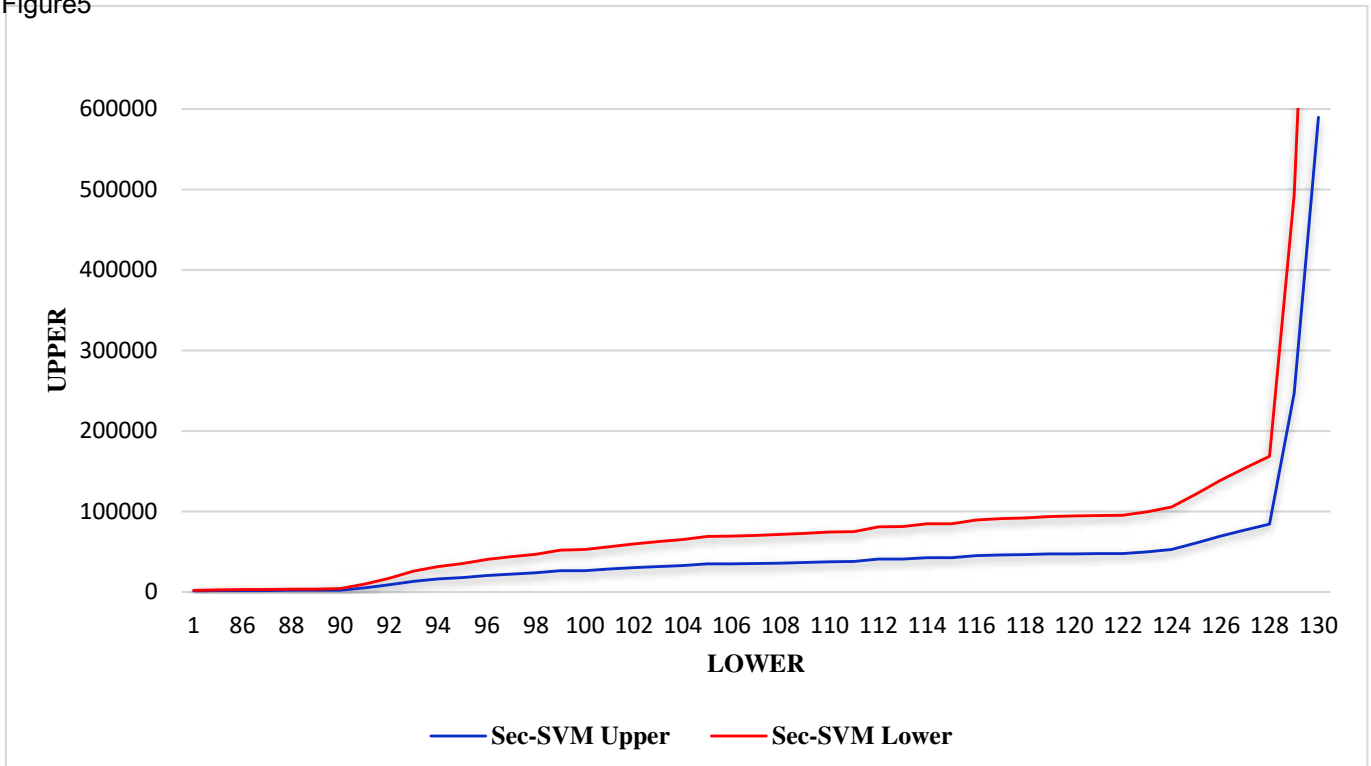

600000
500000
400000
300000
200000
100000
0
UPPER
1 86 88 90 92 94 96 98 100 102 104 106 108 110 112 114 116 118 120 122 124 126 128 130
LOWER
Sec-SVM Upper
Sec-SVM Lower

Figure6

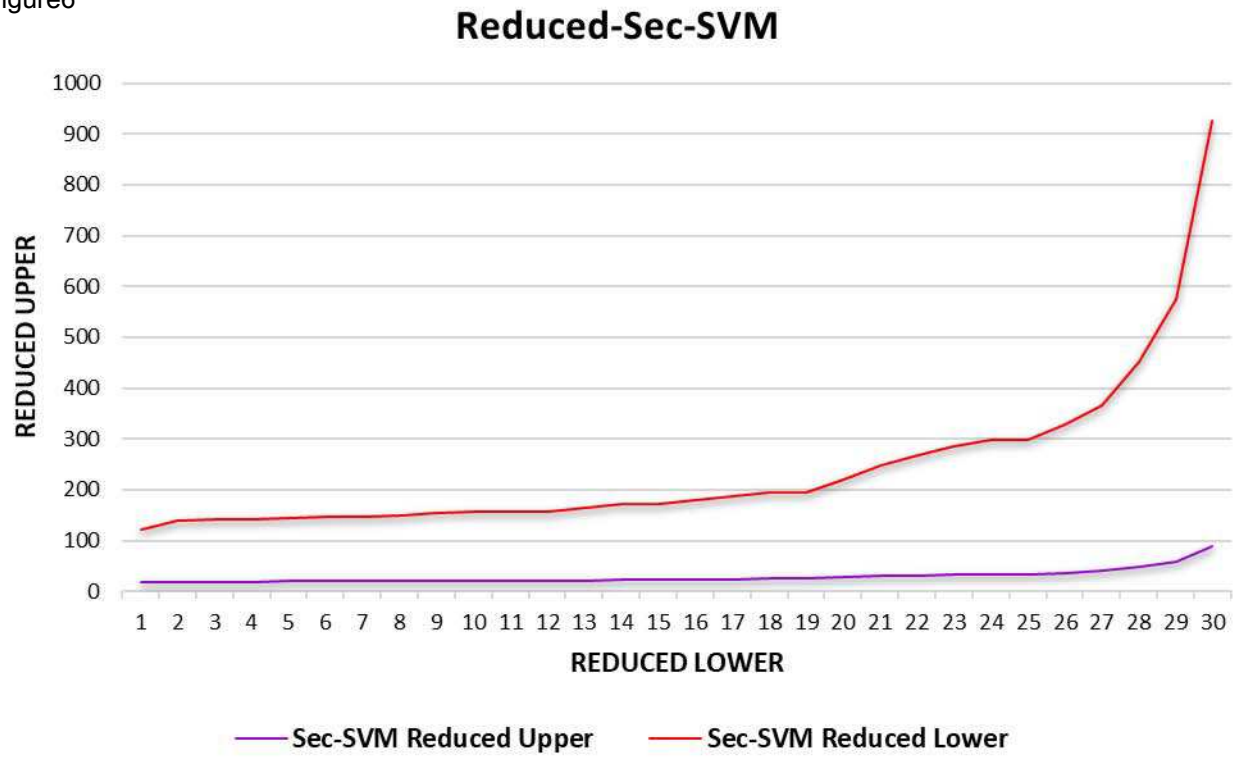

Reduced-Sec-SVM
1000
900
800
700
600
500
400
300
200
100
0
REDUCED UPPER
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
REDUCED LOWER
Sec-SVM Reduced Upper
Sec-SVM Reduced Lower

Figure7

Learned Function Output

GROUND		Positive	Negative
	Positive	**1717**	**109**
	Negative	**176**	**41863**

Figure8

5560 Malwares	Adrd	Base Bridge	Droid Dream	Droid KungFu	ExploitLinux Lotoor	FakeDoc	FakeInstaller	FakeRun	Gappusin	Geinimi	GinMaster	Glodream	Iconosys	Kmin	MobileTx	Opfake	Plankton	SendPay	Unknown Family
Adrd	**21**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4
Base Bridge	0	**92**	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	11
Droid Dream	0	0	**26**	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	3
Droid Kung Fu	0	0	0	**228**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6
Exploit Linux Lotoor	2	0	0	0	**10**	0	0	0	0	0	4	0	0	0	0	0	0	0	7
Fake Doc	0	0	0	0	0	**42**	0	0	0	0	0	0	0	0	0	0	0	0	1
Fake Installer	0	0	0	0	0	5	**256**	0	0	0	0	14	0	0	0	35	0	0	3
Fake Run	0	0	0	0	0	0	0	**21**	0	0	0	0	0	0	0	0	0	0	0
Gappusin	0	0	0	1	0	0	0	0	**10**	0	0	0	0	0	0	0	0	0	5
Geinimi	0	0	0	0	0	0	0	0	0	**27**	0	0	0	0	0	0	0	0	2
Gin Master	0	0	0	1	0	0	0	0	0	0	109	0	0	0	0	0	0	0	11
Glodream	0	0	0	1	0	0	0	0	0	0	0	**20**	0	0	0	0	0	0	5
Iconosys	0	0	0	0	0	0	0	0	0	0	0	0	**49**	0	0	0	0	0	7
Kmin	0	0	0	0	0	0	0	0	0	0	0	0	0	**59**	0	0	0	0	0
Mobile Tx	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**25**	0	0	0	0
Opfake	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	**189**	0	0	1
Plankton	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**211**	0	7
Send Pay	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**21**	0
Unknown Family	9	7	0	1	2	4	16	2	0	1	0	12	0	1	0	0	10	0	**261**

Figure9

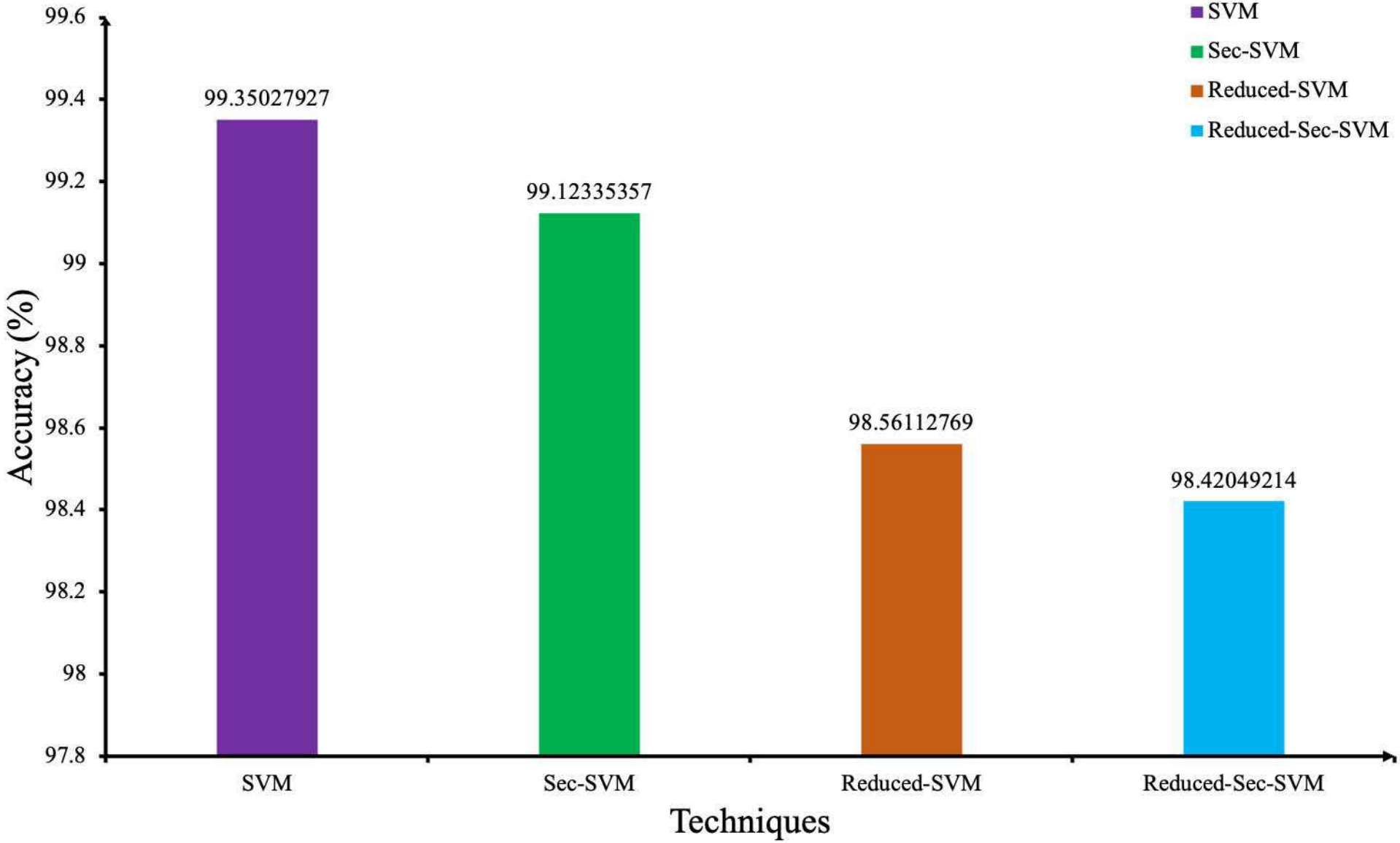
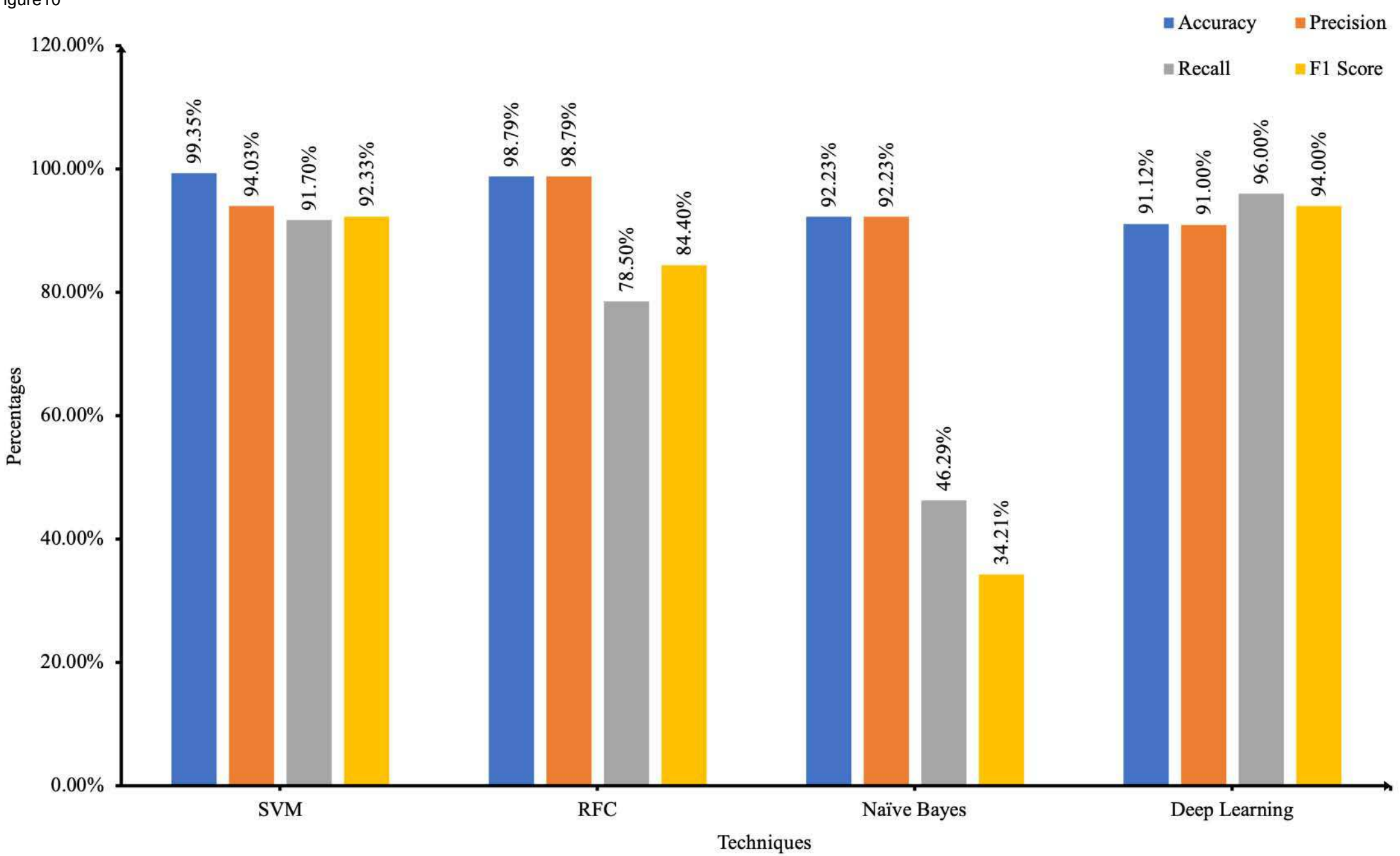

SVM
Sec-SVM
Reduced-SVM
Reduced-Sec-SVM
99.6
99.4
99.2
99
98.8
98.6
98.4
98.2
98
97.8
Accuracy (%)
99.35027927
99.12335357
98.56112769
98.42049214
SVM
Sec-SVM
Reduced-SVM
Reduced-Sec-SVM
Techniques

Figure10


Accuracy
Precision
Recall
F1 Score
120.00%
100.00%
80.00%
60.00%
40.00%
20.00%
0.00%
Percentages
99.35%
94.03%
91.70%
92.33%
98.79%
98.79%
78.50%
84.40%
92.23%
92.23%
46.29%
34.21%
91.12%
91.00%
96.00%
94.00%
SVM
RFC
Naïve Bayes
Deep Learning
Techniques