International Journal of Machine Learning and Cybernetics
https://doi.org/10.1007/s13042-021-01393-7

ORIGINAL ARTICLE

Do gradient-based explanations tell anything about adversarial robustness to android malware?

Marco Melis[1] · Michele Scalas[2] · Ambra Demontis[1] · Davide Maiorca[1] · Battista Biggio[1,2] · Giorgio Giacinto[1] · Fabio Roli[1,2]

Received: 5 December 2020 / Accepted: 22 July 2021

Abstract

While machine-learning algorithms have demonstrated a strong ability in detecting Android malware, they can be evaded by *sparse* evasion attacks crafted by injecting a small set of fake components, e.g., permissions and system calls, without compromising intrusive functionality. Previous work has shown that, to improve robustness against such attacks, learning algorithms should avoid overemphasizing few discriminant features, providing instead decisions that rely upon a large subset of components. In this work, we investigate whether gradient-based attribution methods, used to explain classifiers' decisions by identifying the most relevant features, can be used to help identify and select more robust algorithms. To this end, we propose to exploit two different metrics that represent the *evenness of explanations*, and a new compact security measure called *Adversarial Robustness Metric*. Our experiments conducted on two different datasets and five classification algorithms for Android malware detection show that a strong connection exists between the uniformity of explanations and adversarial robustness. In particular, we found that popular techniques like Gradient*Input and Integrated Gradients are strongly correlated to security when applied to both linear and nonlinear detectors, while more elementary explanation techniques like the simple Gradient do not provide reliable information about the robustness of such classifiers.

Keywords Adversarial machine learning · Adversarial robustness · Android malware · Explainable artificial intelligence · Interpretability

1 Introduction

Machine learning systems are nowadays being extensively adopted in computer security applications, such as network intrusion and malware detection, as they obtained remarkable performances even against the increasing complexity of modern attacks [1, 39, 53]. More recently, learning-based techniques based on static analysis proved to be especially effective at detecting Android malware, which constitutes one of the major threats in mobile security. In particular, these approaches showed great accuracy even when traditional code concealing techniques (such as static obfuscation) are employed [4, 20, 21, 23, 48, 57].

Despite the successful results reported by such approaches, the problem of detecting malware created to fool learning-based systems is still far from being solved. The robustness of machine-learning models is challenged by the creation of the so-called *adversarial examples*, i.e., malicious files that receive fine-grained modifications oriented to deceive the learning-based algorithms [9, 13, 29, 61]. In particular, recent work concerning Android malware demonstrated that specific changes to the contents of malicious Android applications might suffice to change their classification (e.g., from malicious to benign) [15, 23], even though the real-word feasibility of these operations should be carefully evaluated [16, 54]. The main characteristic of these attacks is their *sparsity*, meaning that they enforce only a few changes to the whole feature set to be effective. Such changes may be represented by, e.g., the injection of unused permissions or parts of unreachable/unused executable code. For example, adding a component that is loaded when the application is started (through a keyword called

✉ Marco Melis
is.marco@tiscali.it

[1] Department of Electrical and Electronic Engineering, University of Cagliari, Piazza d'Armi, 09123 Cagliari, Italy

[2] Pluribus One, Cagliari, Italy

Published online: 23 October 2021

LAUNCHER) can significantly influence the classifier's decision [51].

One of the many reasons why such attacks are so effective is that classifiers typically assign significant relevance to a limited amount of features (this phenomenon has also been demonstrated in other applications such as email spam filtering). As a possible countermeasure, research showed that classifiers that avoid overemphasizing specific features, weighting them more evenly, can be more robust against such attacks [10, 23, 37]. Simple metrics characterizing this behavior were proposed to identify and select more robust algorithms, especially in the context of linear classifiers, where feature weights can be used as a direct measure of a feature's relevance to each decision [23–25]. In parallel, the ability to understand the classifiers behavior by looking to the input gradient, i.e. the feature weights in the case of linear classifiers, was also explored by multiple works in the field of explainable machine learning [2, 6, 59, 60]. In particular, it became of interest to figure out if the information provided by these gradient-based methods can also be employed to understand (and improve) the robustness of learning-based systems against attacks [18].

In this paper, motivated by the intuition that the classifiers whose attributions are more evenly distributed should also be the more robust, as they rely on a broader set of features for the decision, we propose and empirically validate a few synthetic metrics that allow correlating the *evenness* of gradient-based explanations with the classifier robustness to adversarial attacks. In summary, we make the following contributions:

- We statistically investigate the possible correlations between gradient-based explanations, and the classifiers robustness to adversarial *sparse* evasion attacks;
- We propose a new measure called *adversarial robustness metric*, to represent the classifier robustness to adversarial attacks along with an increasing attack power in a compact way (Sect. 4);
- We assess our findings on the Drebin [4] feature space, a popular learning-based detection system for Android, using two different large datasets of applications, i.e., *Drebin* [4] and *Tesseract* [3, 52], and five different classification algorithms including linear and non-linear Support Vector Machines, logistic, ridge, and the *secured* linear SVM from [23] (Sect. 5).

The paper is structured as follows. We first provide a description of learning-based systems for Android malware detection (Sect. 2) and their adversarial vulnerabilities (Sect. 3). Then, we present the synthetic metrics we use to perform our correlation analysis between the *evenness* of gradient-based explanations and the *adversarial robustness* of classifiers (Sect. 4). In Sect. 5 we present the results of our investigation, which unveils that, under some circumstances, there is a clear relationship between the distribution of gradient-based explanations and the adversarial robustness of Android malware detectors, especially when exploiting more advanced explanation techniques such as Gradient*Input [51, 59] and Integrated Gradients [60]. After a brief description of many related works on Android malware detectors, adversarial attacks and explainable machine learning (Sect. 6), we conclude the paper with a discussion on how our findings can pave the way towards the development of more efficient mechanisms both to evaluate adversarial robustness and to defend against adversarial Android malware examples (Sect. 7).

2 Android malware detection

Here we provide some background on the structure of Android applications, and then we describe Drebin [4], the Android malware detection system used in our analysis.

2.1 Background on android

Android applications are compressed in `apk` files, i.e., archives that contain the following elements: (a) the `AndroidManifest.xml` file, (b) `classes.dex` files, (c) resource and asset files, such as native libraries or images, and (d) additional `xml` files that define the application layout. Since Drebin analyzes the `classes.dex` files and the `AndroidManifest.xml`, we briefly describe them below.

Android Manifest (*manifest*). The basic information about the Android application is included in the `AndroidManifest.xml`, including its package name or the supported API levels, together with the declaration of its *components*, i.e., parts of code that perform specific actions. For example, one component might be associated with a screen visualized by the user (*activity*) or to the execution of background tasks (*services*). Application components can also perform actions (through *receivers*) on the occurrence of specific events; for instance, a change in the device's connectivity status (CONNECTIVITY_CHANGE) or the opening of an application (LAUNCHER). The `manifest` also contains the list of *hardware components* and *permissions* requested by the app to work (e.g., Internet access).

Dex bytecode (*dexcode*). The `classes.dex` file embeds the compiled source code of the applications, including all the user-implemented methods and classes; the bytecode can be executed with the Dalvik Virtual Machine (until Android 4.4) or the Android runtime (ART). The `classes.dex` may contain specific API calls that can access sensitive resources such as personal contacts (*suspicious calls*). Additionally, it contains all system-related,

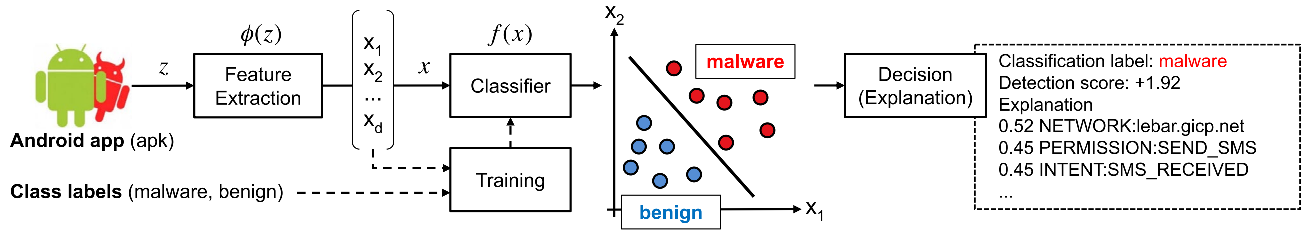


Fig. 1 A schematic representation ([23]) of Drebin. First, applications are represented as binary vectors in a d-dimensional feature space. A linear classifier is then trained on an available set of malware and benign applications, assigning a weight to each feature. During classification, unseen applications are scored by the classifier by summing up the weights of the present features: if $f(\boldsymbol{x}) \geq 0$, they are classified as malware. Drebin also explains each decision by reporting the most suspicious (or benign) features present in the app, along with the weight assigned to them by the linear classifier [4]

restricted API calls that require specific permissions (e.g., writing to the device's storage). Finally, this file can contain references to *network addresses* that might be contacted by the application.

2.2 Drebin

The majority of the approaches for Android malware detection employ static and dynamic analyses that extract information such as usage of permissions, communications through Inter-Component Communication (ICC), system- and user-implemented API calls, and so forth [4, 14, 20, 40, 57].

Drebin is among the most popular and used static detection approaches. It performs the detection of Android malware through static analysis of Android applications. In a first phase (training), it employs a set of benign and malicious apps provided by the user to determine the features that will be used for detection (meaning that the feature set will be strictly dependent on the training data). Such features are then embedded into a *sparse*, high-dimensional vector space. Then, after the training of a linear machine-learning model, the system is able to perform the classification of previously-unseen apps. An overview of the system architecture is given in Fig. 1, and discussed more in detail below.

Feature extraction. First, Drebin statically analyzes a set of n training Android applications to construct a suitable feature space. All features extracted by Drebin are presented as *strings* and organized in 8 different feature sets, as listed in Table 1.

Android applications are then mapped onto the feature space as follows. Let us assume that an app is represented as an object $z \in \mathcal{Z}$, being $\mathcal{Z}$ the abstract space of all `apk` files. We denote with $\Phi : \mathcal{Z} \mapsto \mathcal{X}$ a function that maps an `apk` file z to a d-dimensional feature vector $\boldsymbol{x} = \left(x^1, \ldots, x^{\mathsf{d}}\right)^\top \in \mathcal{X} = \{0, 1\}^{\mathsf{d}}$, where each feature is set to 1 (0) if the corresponding *string* is present (absent) in the `apk` file z. An application encoded in feature space may thus look like the following:

$$\boldsymbol{x} = \Phi(\boldsymbol{z}) \mapsto \begin{pmatrix} \cdots \\ 0 \\ 1 \\ \cdots \\ 1 \\ 0 \\ \cdots \end{pmatrix} \begin{array}{l} \cdots \\ \left.\begin{array}{l} \texttt{permission::SEND\_SMS} \\ \texttt{permission::READ\_SMS} \end{array}\right\} S_2 \\ \cdots \\ \left.\begin{array}{l} \texttt{api\_call::getDeviceId} \\ \texttt{api\_call::getSubscriberId} \end{array}\right\} S_5 \\ \cdots \end{array}$$

Learning and Classification. Drebin uses a linear Support Vector Machine (SVM) to perform detection. It can be expressed in terms of a linear function $f : \mathcal{X} \mapsto \mathbb{R}$, i.e., $f(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x} + b$, where $\boldsymbol{w} \in \mathbb{R}^{\mathsf{d}}$ denotes the vector of *feature weights*, and $b \in \mathbb{R}$ is the so-called *bias*. These parameters, optimized during training, identify a hyperplane that separates the two classes in the feature space. During classification, unseen apps are then classified as malware if $f(\boldsymbol{x}) \geq 0$, and as benign otherwise. In this work, we also consider other linear and nonlinear algorithms to learn the classification function $f(\boldsymbol{x})$.

Explanation. Drebin explains its decisions by reporting, for any given application, the most influential features, i.e., the ones that are present in the given application and are assigned the highest absolute weights by the classifier. The feature relevance values reported by Drebin correspond exactly to its feature weights, being Drebin a linear classifier. For instance, in Fig. 1 it is possible to see that Drebin correctly identifies the sample as malware since it connects to a suspicious URL and uses SMS as a side-channel for communication. In this work, we use different state-of-the-art explainability methods to measure feature relevance and evaluate whether and to which extent the distribution of

Table 1 Overview of Drebin feature sets

`manifest`		`dexcode`	
S_1	Hardware components	S_5	Restricted API calls
S_2	Requested permissions	S_6	Used permission
S_3	Application components	S_7	Suspicious API calls
S_4	Filtered intents	S_8	Network addresses

relevance values reveals any interesting insight on adversarial robustness.

3 Adversarial android malware

Machine learning algorithms are known to be vulnerable to adversarial examples. The ones used for Android malware detection do not constitute an exception. The vulnerability of those systems was demonstrated in [23, 24, 31], and a defense mechanism was proposed in [23]. In this section, we first explain how an attacker can construct Android malware able to fool a classifier, being recognized as benign. Then, considering the system called Sec-SVM [23] as a case-study, we explain how machine learning systems can be strengthened against this attack.

3.1 Attacking android malware detection

The goal of creating adversarial Android malware that evades detection can be formulated as an optimization problem, as detailed below. This optimization problem is constrained to ensure that the solution provides a functional and realizable malware sample, i.e., that the feature changes suggested by the attack algorithm are feasible and can be implemented as practical manipulations to the actual apk input file.

Problem Formulation. As explained in the previous section, Drebin is a binary classifier trained on Boolean features. To have a malware sample $\boldsymbol{z}$ misclassified as benign, the attacker should modify its feature vector $\boldsymbol{x}$ in order to decrease the classifier score $f(\boldsymbol{x})$. The number of features considered by Drebin is quite large (more than one million). However, the attacker can reasonably change only few of them (*sparse attack*) to preserve the malicious functionality of the application. The attacker has thus an ℓ_1-norm constraint on the number of features that can be modified. The feature vector of the adversarial application can be computed by solving the following optimization problem:

$$\arg\min_{\boldsymbol{x}'} \quad f(\boldsymbol{x}') \tag{1}$$

$$\text{s. t.} \quad \|\boldsymbol{x} - \boldsymbol{x}'\|_1 \le \varepsilon \tag{2}$$

$$\boldsymbol{x}_{\text{lb}} \le \boldsymbol{x}' \le \boldsymbol{x}_{\text{ub}} \tag{3}$$

$$\boldsymbol{x}' \in \{0, 1\}, \tag{4}$$

where Eq. (2) is the ℓ_1 distance constraint between the original $\boldsymbol{x}$ and the modified (adversarial) $\boldsymbol{x}'$ sample. Eq. (3) is a box constraint that enforces the features values of the adversarial malware to stay within some lower and upper bounds, while Eq. (4) enforces the attack to find a Boolean solution. The aforementioned problem can be solved with gradient-based optimization techniques, e.g., Projected Gradient Descent (PGD), as described in Alg. 1 [9, 24, 49]. At each step, this algorithm projects the feature values of the adversarial sample onto the constraints (Eqs. 2, 3), including binarization in $\{0, 1\}$.

Algorithm 1 PGD-based attack on Android malware

Input: $\boldsymbol{x}$, the input malware; ε, the number of features which can be modified; η, the step size; Π, a projection operator on the constraints (2) and (3); $t > 0$, a small number to ensure convergence.
Output: $\boldsymbol{x}'$, the adversarial (perturbed) malware.

```
$\boldsymbol{x}' \leftarrow \boldsymbol{x}$
repeat
    $\boldsymbol{x}^\star \leftarrow \boldsymbol{x}'$
    $\boldsymbol{x}' \leftarrow \Pi(\boldsymbol{x}^\star - \eta \cdot \nabla f(\boldsymbol{x}^\star))$
until $|f(\boldsymbol{x}') - f(\boldsymbol{x}^\star)| \le t$
return: $\boldsymbol{x}'$
```

Feature addition. To create malware able to fool the classifier, an attacker may, in theory, both adding and removing features from the original applications. However, in practice, removing features is a non-trivial operation that can easily compromise the malicious functionalities of the application. Feature addition is a safer operation, especially when the injected features belong to the `manifest`; for example, adding permissions does not influence any existing application functionality. When the features depend on the `dexcode`, it is possible to add them safely introducing information that is not actively executed, e.g., by adding code after `return` instructions (*dead code*) or methods that are never called by any `invoke` type instructions (i.e., the ones that indicate a method call). Therefore, in this work, we only consider feature addition. To find a solution that does not require removing features from the original application, the attacker can simply define $\boldsymbol{x}^{\text{lb}} = \boldsymbol{x}$ in Eq. (3). However, it is worth mentioning that this injection could be easily made ineffective, simply removing all the features extracted from code lines that are never executed. In this way, the attacker is forced to change the executed code, which is more difficult, as it requires considering the following additional and stricter constraints. Firstly, the attacker should avoid breaking the application functionalities. Secondly, they should avoid introducing possible artifacts or undesired functionalities, which may influence the semantics of the original program. Injecting a large number of features may be, therefore, difficult and not always feasible.

3.2 Sec-SVM: defending against adversarial android malware

In [23], the authors showed that the sparse evasion attack described above is able to fool Drebin, requiring the

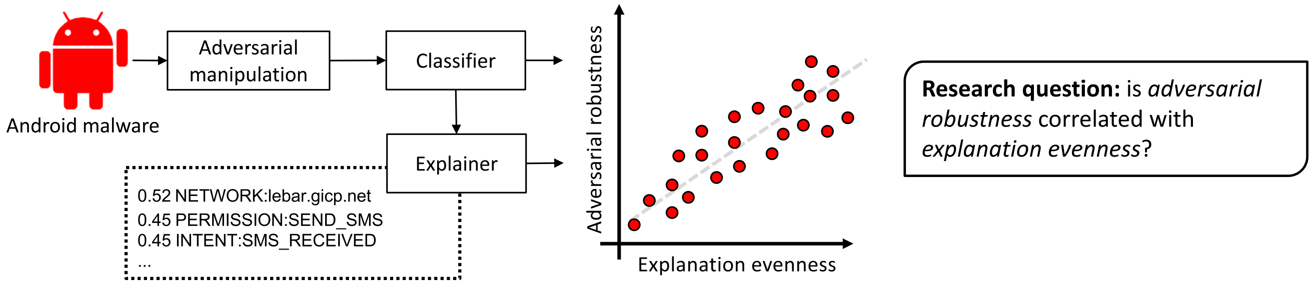


Fig. 2 Schematic representation of the analysis employed to verify the correlation between explanation evenness and adversarial robustness. First, for each malware in the test set, we create its adversarial counterpart. Then, for each of those adversarial applications, we evaluate: (1) a measure of the classifier robustness against it (*adversarial robustness metric*) (2) the evenness of the application attributions (*explanation evenness*). Finally, we asses the correlation between them

injection of a negligible number of features, and they propose a robust counterpart of that classifier. The underlying idea behind their countermeasure is to enforcing the classifier to learn more evenly distribute feature weights since this will require the attacker to manipulating more features to evade the classifier. To this end, they added a box constraint on the weights $\boldsymbol{w}$ of a linear SVM, obtaining the following learning algorithm (Sec-SVM):

$$\begin{aligned} &\min_{\boldsymbol{w},b} \frac{1}{2}\boldsymbol{w}^{\top}\boldsymbol{w} + C\sum_{i=1}^{\mathrm{n}} \max\left(0, 1 - y_i f(\boldsymbol{x}_i)\right) \\ &\text{s. t. } w_k^{\mathrm{lb}} \le w_k \le w_k^{\mathrm{ub}},\ k = 1, \dots, \mathrm{d}, \end{aligned} \tag{5}$$

where the lower and upper bounds on $\boldsymbol{w}$ are defined, respectively, by the vectors $\boldsymbol{w}^{\mathrm{lb}} = (w_1^{\mathrm{lb}}, \dots, w_{\mathrm{d}}^{\mathrm{lb}})$ and $\boldsymbol{w}^{\mathrm{ub}} = (w_1^{\mathrm{ub}}, \dots, w_{\mathrm{d}}^{\mathrm{ub}})$, which are application-dependent. Equation (5) can be easily optimized using a constrained variant of the Stochastic Gradient Descent (SGD) technique, as described in [23].

4 Do gradient-based explanations help to understand adversarial robustness?

In this work, we investigate whether gradient-based attribution methods used to explain classifiers' decisions provide useful information about the robustness of Android malware detectors against sparse attacks. Our intuition is that the classifiers whose attributions are usually evenly-distributed rely upon a broad set of features instead of overemphasizing only a few of them. Therefore, they are more robust against sparse attacks, where the attacker can change only a few features, having a negligible impact on the classifier decision function. To verify our intuition, we present an empirical analysis whose procedure is illustrated in Fig. 2 and described below. Firstly, we perform a security evaluation on the tested classifier, obtaining a compact measure we call *Adversarial Robustness Metric* (see Sect. 4.1), representing its robustness to the adversarial attacks along with an increasing number of added features ϵ. Then, we compute the attributions for each benign and manipulated malware sample $\boldsymbol{x}$ using a chosen gradient-based explanation technique (see Sect. 4.2) obtaining the relevance vectors $\boldsymbol{r}$. For each of those, we propose to look for a compact metric that encapsulates the degree of *Evenness* of the attributions (see Sect. 4.3). Finally, comparing this value with the adversarial robustness metric, we asses the connections between attributions' evenness and the robustness to adversarial evasion attacks. In Sect. 5, we present the results of our analysis on five different learning algorithms trained on the feature space extracted by Drebin, providing the empirical evidence of our intuition.

4.1 Adversarial robustness metric

We define the robustness to the evasion samples crafted injecting a fixed number of features ϵ as:

$$R(\mathcal{D}_\epsilon, f) = \frac{1}{n}\sum_{i=1}^{n} e^{-\ell_i}, \tag{6}$$

where $\ell_i = \ell(y_i, f(\boldsymbol{x}_i))$ is the adversarial loss attained by the classifier f on the data points in $\mathcal{D}_\varepsilon = \{\boldsymbol{x}_i, y_i\}_{i=1}^n$, containing the ε-sized adversarial samples optimized with Algorithm 1.

Finally, the adversarial robustness metric $\mathcal{R}$ of a classifier f is defined as the average of $R(\mathcal{D}_\varepsilon, f)$ on different ε:

$$\mathcal{R} = \mathbb{E}_\varepsilon\{R(\mathcal{D}_\varepsilon, f)\}. \tag{7}$$

4.2 Gradient-based explanation methods

In our analysis, we consider gradient-based attribution methods, where *attribution* means the contribution of each input feature to the prediction of a specific sample. The positive (negative) value of an attribution indicates that the classifier considers the corresponding feature as peculiar of the malicious (benign) samples. In the following, we review the three gradient-based techniques considered in this work.

Gradient. The simplest method to obtain the attributions is to compute the gradient of the discriminant function f with respect to the input sample $\boldsymbol{x}$. For image recognition models, it corresponds to the saliency map of the image [6]. The attribution of the ith feature is computed as:

$$\text{Gradient}_i(\boldsymbol{x}) := \frac{\partial f(\boldsymbol{x})}{\partial x_i}. \tag{8}$$

*Gradient*Input*. This technique has been proposed in [59] and utilized in one of our previous work [51], to identify the most influential features for an Android malware detector trained on sparse data. As we have shown in that paper, this approach is more suitable than the previously proposed ones when the feature vectors are sparse. The previously proposed approaches [6, 55] tended to assign relevance to features whose corresponding components are *not* present in the considered application, thus making the corresponding predictions challenging to interpret. To overcome this issue, this technique leverages the notion of *directional derivative*. Given the input point $\boldsymbol{x}$, it projects the gradient $\nabla f(\boldsymbol{x})$ onto $\boldsymbol{x}$, to ensure that only the non-null features are considered as relevant for the decision. More formally, the ith attribution is computed as:

$$\text{Gradient*Input}_i(\boldsymbol{x}) := \frac{\partial f(\boldsymbol{x})}{\partial x_i} * x_i. \tag{9}$$

Integrated Gradients. Sundararajan et al. [60] identified two axioms that attribution methods should satisfy: *implementation invariance* and *sensitivity*. Accordingly to the first, the attributions should always be identical for two functionally equivalent networks, e.g. they should be invariant to the differences in the training hyperparameters, which lead the network to learn the same function. The second axiom is satisfied if, for every input predicted differently from a baseline (a reference vector that models the neutral input, e.g. a black image) and that differs from the baseline in only one feature, has, for that feature, a non-zero attribution. In the same paper, they proposed a gradient-based explanation called Integrated Gradient that satisfies the axioms explained above. This method, firstly, considers the straight-line path from the baseline to the input sample and computes the gradients at all points along the path. Then, it obtains the attribution cumulating those gradients. The attribution along the ith dimension for an input $\boldsymbol{x}$ and baseline $\boldsymbol{x}'$ is defined as:

$$\text{IntegratedGrads}_i(\boldsymbol{x}) := \\ \left(x_i - x_i'\right) \cdot \int_{\alpha=0}^{1} \frac{\partial f\left(\boldsymbol{x}' + \alpha \cdot \left(\boldsymbol{x} - \boldsymbol{x}'\right)\right)}{\partial x_i} d\alpha. \tag{10}$$

To efficiently approximate the previous integral, one can sum the gradients computed at p fixed intervals along the joining path from $\boldsymbol{x}'$ to the input $\boldsymbol{x}$:

$$\text{IntegratedGrads}_i^{\text{approx}}(\boldsymbol{x}) := \\ \left(x_i - x_i'\right) \cdot \sum_{k=1}^{p} \frac{\partial f\left(\boldsymbol{x}' + \frac{k}{p} \cdot \left(\boldsymbol{x} - \boldsymbol{x}'\right)\right)}{\partial x_i} \cdot \frac{1}{p}. \tag{11}$$

For linear classifiers, where $\partial f / \partial x_i = w_i$, this method is equivalent to Gradient*Input if $\boldsymbol{x}' = \boldsymbol{0}$ is used as a baseline, which is a well-suited choice in many applications [60]. Therefore, in this particular case, also the Gradient*Input method satisfies the abovementioned axioms.

4.3 Explanation evenness metrics

To compute the evenness of the attributions, we consider the two metrics, described below. The first is the one proposed in [10, 37]. To compute the evenness metric, they firstly defined a function $F(\boldsymbol{r}, k)$ which, given a relevance vector $\boldsymbol{r}$, computes the ratio of the sum of the k highest relevance values to the sum of all absolute relevance values, for $k = 1, 2, \dots, m$:

$$F(\boldsymbol{r}, k) = \frac{\sum_{i=1}^{k} |r_{(i)}|}{\sum_{j=1}^{m} |r_{(j)}|},$$

where $r_1, r_2, \dots, r_m$ denote the relevance values, sorted in descending order of their absolute values, i.e., $|r_1| \geq |r_2| \geq \dots \geq |r_m|$ and m is the number of considered relevance values ($m \leq d$). This function essentially computes the evenness of the distribution of the relevance among the features. The evenest relevance distribution (the one where they are all equal), corresponds to $F(\boldsymbol{r}, k) = k/n$. Whereas the most uneven is attained when only one relevance differs from zero, and in this case, $F(\boldsymbol{r}, k) = 1$ for each k value. To avoid the dependence on k and to obtain a single scalar value, they compute the evenness as:

$$\mathcal{E}_1(\boldsymbol{r}) = \frac{2}{m-1}\left[m - \sum_{k=1}^{m} F(\boldsymbol{r}, k)\right]. \tag{12}$$

The range of $\mathcal{E}_1$ is [0, 1], $\mathcal{E}_1 = 0$ and $\mathcal{E}_1 = 1$ indicates respectively to the most uneven and to the most even relevance vector.

The second metric we consider is the one proposed in [25], based on the ratio between the ℓ_1 and ℓ_∞ norm:

$$\mathcal{E}_2(\boldsymbol{r}) = \frac{1}{m} \cdot \frac{\|\boldsymbol{r}\|_1}{\|\boldsymbol{r}\|_\infty}. \tag{13}$$

To have a broader perspective of the attributions' evenness, we compute the metrics on multiple samples, and we average the results. More formally, we define the *explanation evenness* as:

$$E = \frac{1}{n}\sum_{i=1}^{n}\mathcal{E}(\boldsymbol{r}^i), \tag{14}$$

where $\boldsymbol{r}^i$ with $i = 1, 2, \dots, n$ is the attribution vector computed on each sample of a test dataset $\mathcal{D} = \{\boldsymbol{x}_i, y_i\}_{i=1}^{n}$, and $\mathcal{E}$ can be equal either to $\mathcal{E}_1$ or $\mathcal{E}_2$. In the following, we represent the averaged evenness computed considering the per-sample metric $\mathcal{E}_1$ ($\mathcal{E}_2$) with E_1 (E_2).

4.4 Computational complexity

An important aspect to consider when performing the adversarial robustness evaluation of a classifier is the complexity of the process. Especially in the case of non-linear classifiers, computing a full security evaluation curve for this purpose may require hundred thousands iterations [13]. Because, for each test sample, the corresponding adversarial example should be computed, this process involves, for every single point, thousand of iterations (and thus of gradient and function evaluations) with the chosen optimization algorithm [63]. In this sense, exploiting the gradient-based attributions methods provides instead a massive computational advantage. In fact, for both Gradient and Gradient*Input methods, only a single gradient evaluation is required to obtain the attributions of an input sample, and this gradient is even identical for all samples in the case of linear classifiers (allowing to save even more evaluations). For the Integrated Gradients technique, the number of gradient evaluations depends on the chosen value of the p parameter in Eq. (11) which, however, is usually set to a small number. In our correlation analysis, we assume that the attack samples required to compute the adversarial robustness metric $\mathcal{R}$ and the attributions for each of the test samples are already acquired as part of the security and explainability evaluation of the classifiers. The complexity of the process is then given by the computation of the explanation evenness and the adversarial robustness metric.

5 Experimental analysis

In this section, we practically evaluate whether the measures introduced in Sect. 4 can be used to estimate the robustness of classifiers against sparse evasion attacks. After detailing our experimental setup (Sect. 5.1), we show the classifiers' detection performances, both in normal conditions and under attack (Sect. 5.2). In our evaluations, we focus on the feature addition attack setting (see Sect. 3), as they are typically the easiest to accomplish for the adversary. We use `secml` as a framework to implement classification systems, explanation techniques, and attack algorithms [50]. Finally, we assess the relationship of the proposed evenness metrics with our new adversarial robustness metric and the detection rate (Sect. 5.3).

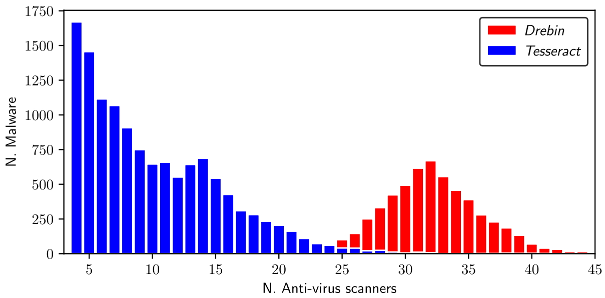


Fig. 3 Distribution of malware in each dataset with respect to the number of anti-virus scanners that flagged the applications as positive. Data extracted from `VirusTotal`

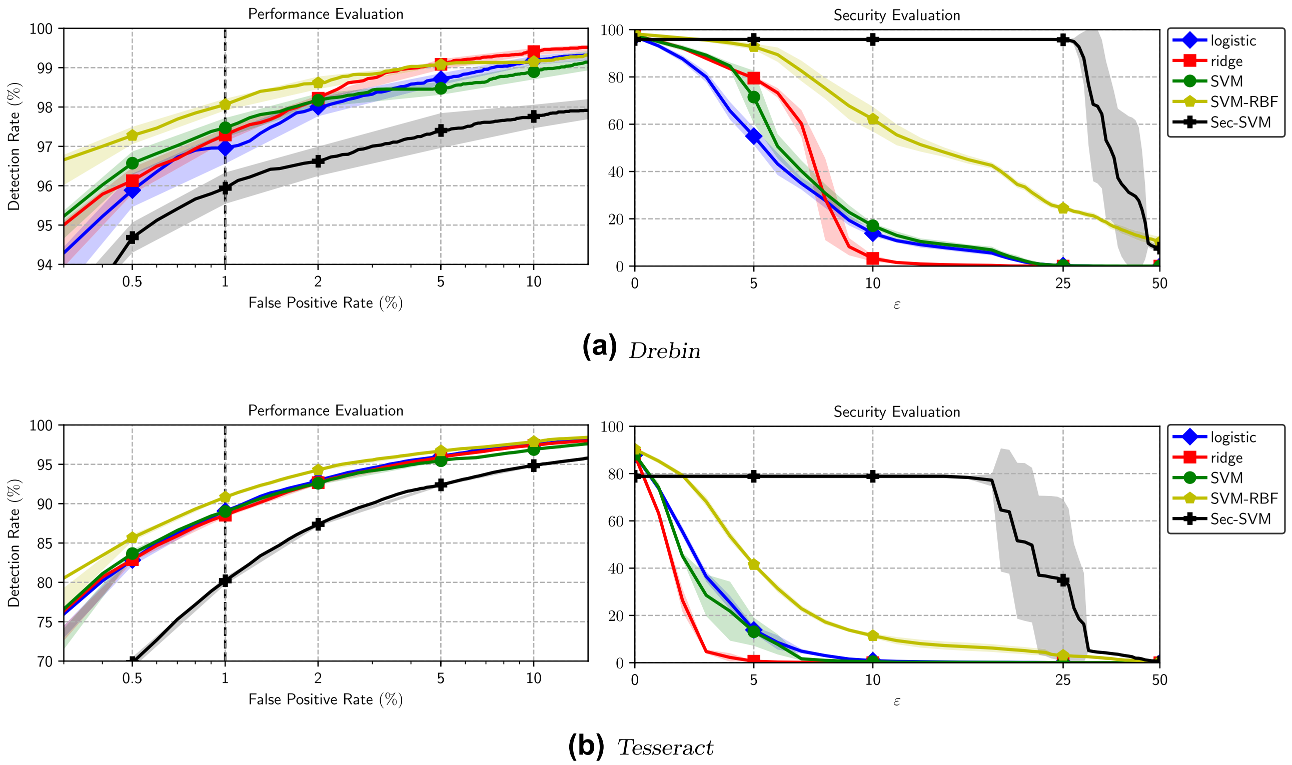
5.1 Experimental setup

Datasets. We use two different datasets of real-world Android applications. The first is the *Drebin* dataset [4], consisting of 121, 329 benign applications and 5, 615 malicious samples, labeled with `VirusTotal` and collected between August 2010 and October 2012. A sample is labeled as malicious if it is flagged by at least five anti-virus scanners, whereas it is labeled as benign otherwise. The second is the *Tesseract* dataset [52], consisting of 116, 993 benign applications and 12, 735 malicious samples, collected from `AndroZoo` [3] between January 2014 and December 2016. In Fig. 3 we report the distribution of malware in each dataset with respect to the number of anti-virus scanners that flagged the applications as positive. We can observe how for *Drebin* most samples are flagged by 30–35 scanners, while for *Tesseract* 4–10 scanners detect most of the positives. This shows how recognizing the samples of the latter, newer dataset, still represents a significant challenge for many scanners.

Training-validation-test splits. We average our results on 5 runs. In each run, we randomly selected 60,000 apps from both datasets to train the learning algorithms, and we used the remaining apps for testing.

Classifiers. We compare the standard *Drebin* implementation based on a linear Support Vector Machine (SVM) against the *secured* linear SVM from [23] (Sec-SVM), an SVM with the RBF kernel (SVM-RBF), a logistic regression (logistic) and a ridge regression (ridge).

Parameter setting. Using a 10-fold cross-validation procedure, we optimize the parameters of each classifier to maximize the detection rate (i.e., the fraction of detected malware) at 1% false-positive rate (i.e., the fraction of legitimate applications misclassified as



(a) *Drebin*

(b) *Tesseract*

Fig. 4 (left) Mean ROC curves for the tested classifiers. (right) White-box evasion attacks. Detection Rate at 1% False Positive Rate against an increasing number of added features ε. We can see how the Sec-SVM, despite providing a slightly lower detection rate compared to the other tested classifiers on both datasets, requires on average more than 20 different new feature additions to the original applications to be fooled by the attacker

malware). In particular, we optimize the parameters $C \in \{10^{-2}, 10^{-1}, \dots, 10^{2}\}$ for logistic and both linear and non-linear SVMs, the kernel $\gamma \in \{10^{-4}, 10^{-3}, \dots, 10^{2}\}$ for the SVM-RBF, and $\alpha \in \{10^{-2}, 10^{-1}, \dots, 10^{2}\}$ for ridge. For Sec-SVM, we optimized the $-\boldsymbol{w}^{\mathrm{lb}} = \boldsymbol{w}^{\mathrm{ub}} \in \{0.1, 0.25, 0.5\}$ and $C \in \{10^{-2}, 10^{-1}, \dots, 10^{2}\}$. When similar detection rates ($\pm 1\%$) are obtained for different hyperparameter configurations, we select the configuration corresponding to a more regularized classifier, as more regularized classifiers are expected to be more robust under attack [24]. The typical values of the aforementioned hyperparameters found for both datasets after cross-validation are $C = 0.1$ for SVM, $\alpha = 10$ for ridge, $C = 1$ for logistic, $C = 1$ and $w = 0.25$ for Sec-SVM, $C = 10$ and $\gamma = 0.01$ for SVM-RBF.

Attribution computation. For each dataset, we compute the attributions on 1000 malware samples randomly chosen from the test set. We took $\boldsymbol{x}' = 0$ as the baseline for Integrated Gradients, and we compute the attributions with respect to the malware class. As a result, positive (negative) relevance values in our analysis denote malicious (benign) behavior. Given the high sparsity ratio of the feature space, we use $m = 1000$ to compute the explanation evenness metrics.

5.2 Experimental results

We first perform an evaluation of the performances under normal conditions; the resulting Receiver Operating Characteristic (ROC) curves with the Detection Rate for each classifier, averaged over the 5 repetitions, is reported in the left side of Fig. 4a and b. We then perform a white-box evasive attack against each classifier, aiming to have 1000 malware samples randomly chosen from the test sets misclassified as benign. The results are shown on the right side of Fig. 4a and b, which report the variation of the detection rate as the number of modified features ε increases. On both datasets, we can notice how the Sec-SVM classifier (described in Sect. 3.2) provides a slightly worse detection rate compared to the other classifiers, but is particularly robust against adversarial evasion attacks.

Table 2 Top-10 influential features and corresponding Gradient*Input relevance (%) for a malware of the `FakeInstaller` family (top) and a malware of the `Plankton` (bottom). Notice that the minimum number of features to add $\varepsilon_{\min}$ to evade the classifiers with the evenness metric $\mathcal{E}_1$ and $\mathcal{E}_2$

SVM-RBF ($\mathcal{E}_1 = 46.24\%$, $\mathcal{E}_2 = 22.47\%$, $\varepsilon_{\min} = 6$)			Sec-SVM ($\mathcal{E}_1 = 73.04\%$, $\mathcal{E}_2 = 66.24\%$, $\varepsilon_{\min} = 31$)		
Set	**Feature Name**	r (%)	**Set**	**Feature Name**	r (%)
S2	SEND_SMS	10.35	S2	READ_PHONE_STATE	3.51
S7	android/telephony/TelephonyManager ;->getNetworkOperator	10.05	S7	android/telephony/TelephonyManager ;->getNetworkOperator	3.51
S4	LAUNCHER	-8.89	S2	SEND_SMS	3.51
S5	android/os/PowerManager$WakeLock ;->release	-8.01	S3	c2dm.C2DMBroadcastReceiver	3.51
S2	READ_PHONE_STATE	5.03	S2	INTERNET	3.44
S2	RECEIVE_SMS	-5.00	S3	com.software.application.ShowLink	3.39
S3	c2dm.C2DMBroadcastReceiver	4.56	S3	com.software.application.Main	3.39
S2	READ_SMS	3.52	S3	com.software.application.Notificator	3.39
S4	DATA_SMS_RECEIVED	3.50	S3	com.software.application.Checker	3.39
S5	android/app/NotificationManager ;->notify	-3.49	S3	com.software.application.OffertActivity	3.39

SVM-RBF ($\mathcal{E}_1 = 60.74\%$, $\mathcal{E}_2 = 25.84\%$, $\varepsilon_{\min} = 31$)			Sec-SVM ($\mathcal{E}_1 = 63.14\%$, $\mathcal{E}_2 = 52.70\%$, $\varepsilon_{\min} = 39$)		
Set	**Feature Name**	r (%)	**Set**	**Feature Name**	r (%)
S4	LAUNCHER	-1.89	S2	ACCESS_NETWORK_STATE	0.93
S7	android/net/Uri;->fromFile	1.34	S2	READ_PHONE_STATE	0.93
S5	android/os/PowerManager$WakeLock ;->release	-1.25	S6	READ_HISTORY_BOOKMARKS	0.93
S2	INSTALL_SHORTCUT	1.23	S7	android/telephony/TelephonyManager ;->getNetworkOperatorName	-0.93
S7	android/telephony/SmsMessage ;->getDisplayMessageBody	-1.21	S6	ACCESS_NETWORK_STATE	-0.93
S7	android/telephony/SmsMessage ;->getTimestampMillis	-1.20	S7	android/telephony/SmsMessage;- >getDisplayOriginatingAddress	0.93
S2	SET_ORIENTATION	-1.20	S7	android/telephony/TelephonyManager ;->getNetworkOperator	0.93
S2	ACCESS_WIFI_STATE	1.15	S7	android/net/Uri;->getEncodedPath	-0.93
S4	BOOT_COMPLETED	1.08	S2	SET_ORIENTATION	-0.93
S5	android/media/MediaPlayer;->start	-1.06	S7	java/lang/reflect/Method;->invoke	0.93

5.3 Is adversarial robustness correlated with explanation evenness?

We now investigate the connection between adversarial robustness and evenness of gradient-based explanations. We start with two illustrative examples. Table 2 shows the top-10 influential features for two malware samples from the *Drebin* dataset, one of `FakeInstaller`[1] and one of `Plankton`[2] family, reported for the SVM-RBF and Sec-SVM algorithms, and obtained through the Gradient*Input technique. All the classifiers correctly label the samples as malware.

Looking at the features of the `FakeInstaller` malware, we can observe how both the classifiers identify the cellular- and SMS-related features, e.g., the `GetNetworkOperator()` method or the `SEND_SMS` permission, as highly relevant. This is coherent with the actual behavior of the malware sample since its goal is to send SMS messages to premium-rate numbers. With respect to the relevance values, the first aspect to point out comes from their relative magnitude, expressed as a percentage in Table 2. In particular, we can observe that the top-10 relevance values for SVM-RBF vary, regardless of their signs, from 3.49 to 10.35%, while for Sec-SVM the top values lie in the 3.39–3.51% range. This suggests that SVM-RBF assigned high prominence to few features; conversely, Sec-SVM distributed the relevance values more evenly. It is possible to catch this behavior more easily through the synthetic evenness measures $\mathcal{E}_1$ (Eq. 12) and $\mathcal{E}_2$ (Eq. 13) reported in Table 2, which show higher values for Sec-SVM. Table 2 also shows the $\varepsilon_{\min}$ value, i.e., the minimum number of features to add to the malware to evade the classifier. We can notice how the $\varepsilon_{\min}$ parameter is strictly related to the evenness distribution, since higher values of $\mathcal{E}_1$ and $\mathcal{E}_2$ correspond to higher values of $\varepsilon_{\min}$, i.e., a higher effort for the attacker to accomplish her goal. In particular, it is possible to identify a clear difference between the behavior of SVM-RBF and Sec-SVM: the diversity of their evenness metrics, which cause the $\varepsilon_{\min}$ values to be quite different as well, indicates that, for this prediction, SVM-RBF is quite susceptible to a possible attack compared to Sec-SVM.

Conversely, considering the second sample, the attributions (regardless of the sign) and the evenness metrics present similar values. Such behavior is also reflected in the associated $\varepsilon_{\min}$ values. In this case, the relevance values are more evenly distributed, which indicates that the evasion is more difficult.

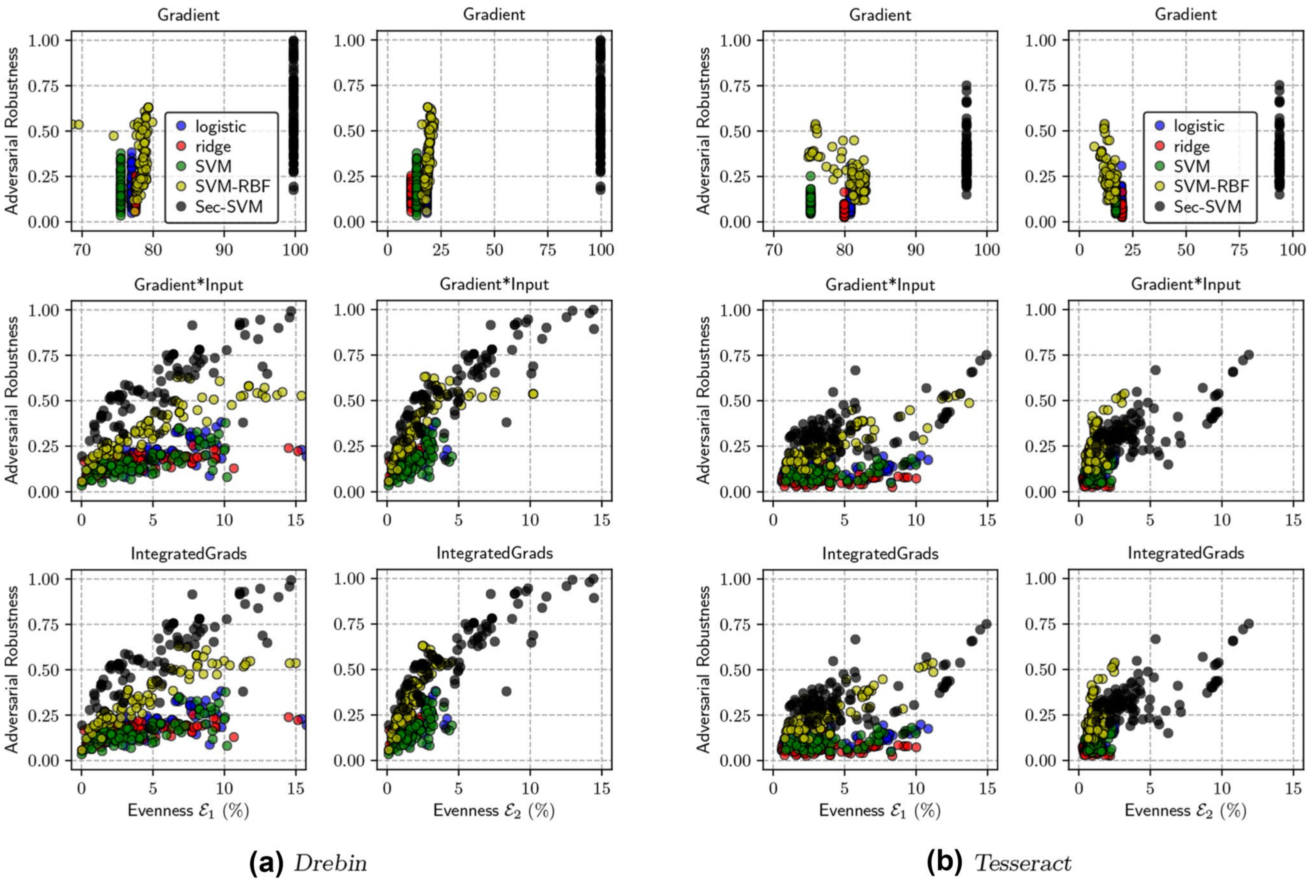
We now correlate the evenness metrics with the *adversarial robustness metric* $\mathcal{R}$, introduced in Sect. 4.1. Figure 5 shows the relationship between this value and the evenness metrics for 100 samples chosen from the test set of *Drebin* (Fig. 5a) and *Tesseract* (Fig. 5b), reported for each explainability technique. From this broader view, we can see how the evenness values calculated on top of the Gradient*Input and Integrated Gradients explanations present a significant connection to the adversarial robustness metric for both datasets. This seems not applicable to the Gradient technique, which appears to be weakly correlated with explanation evenness. Specifically, we observe in Fig. 5 that the dots of the linear classifiers are perfectly vertical-aligned. This fact is caused by the constant value of the gradient across all the samples, which implies constant values for the evenness metrics as well. The reliability of this technique appears to be low even in the case of SVM-RBF, especially for the *Tesseract* dataset where we observe a negative correlation with the explanation evenness.

In order to assess the statistical significance of these plots, we also compute the associated correlation values with three different metrics: Pearson (P), Spearman Rank (S), Kendall's Tau (K). The results are shown in Table 3a and b. In the case of *Drebin* data, we obtain a strong p-val $\ll 0.05$ for all the tested classifiers using both Gradient*Input and Integrated Gradients, confirming the validity of our findings from Fig. 5. The same is valid in the case of *Tesseract* data, with p-val < 0.01 for the two explanation techniques and both evenness metrics in all cases.

We also inquire whether the connection between the evenness metrics and the detection performance of a classifier can provide a global assessment of its robustness. Figure 6a and b show the correlation between the explanation evenness and the mean detection rate under attack, calculated

[1] MD5: f8bcbd48f44ce973036fac0bce68a5d5.

[2] MD5: eb1f454ea622a8d2713918b590241a7e.



(a) *Drebin* **(b)** *Tesseract*

Fig. 5 Evaluation of the adversarial robustness metric $\mathcal{R}$ against the evenness $\mathcal{E}_1$, $\mathcal{E}_2$ metrics for the different gradient-based explanation techniques computed on 1000 samples of the test set (only 100 samples are shown)

for ε in the range [1, 50]. Similarly to the previous tests, the uniformity metrics computed on the explanations from Gradient*Input and Integrated Gradients techniques present a significant connection to the detection rate, also witnessed by the p-values mostly under 0.01 for both datasets. Finally, the correlation with the Gradient is again scarce, showing how this technique is not reliable to obtain information about the adversarial robustness of the tested classifiers to sparse evasion attacks.

6 Related work

In this section, we provide an overview of the literature on Android Malware Detection systems (Sect. 6.1), on the techniques to craft powerful adversarial attacks against them (Sect. 6.2), and, finally, on the approaches to explain their decisions (Sect. 6.3).

6.1 Android malware detection

The detection of Android malware attacks has been addressed over the years through works leveraging static, dynamic, or hybrid analyses.

Arzt et al. [5] proposed FlowDroid, a security tool that performs static taint analysis within the single components of Android applications. Feng et al. [27] proposed Apposcopy, a detection tool that combines static taint analysis and intent flow monitoring to produce a signature for applications. Tam et al. [62] proposed CopperDroid, a dynamic analyzer that aims to identify suspicious high-level behaviors of malicious Android applications. More recently, MaMaDroid by Mariconti et al. [48] employs Markov chains to model sequences of API calls. Chen et al. [21] converted app opcodes to an image-like structure in order to perform data augmentation through a Generative Adversarial Network (GAN), while the works by Mahindru et al. focused on assessing effective feature selection, mainly considering the usage of APIs and permissions as features [44, 45]. Moreover, different works in the literature target specific types of

Table 3 Correlation between the adversarial robustness metric $\mathcal{R}$ and the evenness metrics $\mathcal{E}_1$ and $\mathcal{E}_2$. Pearson (P), Spearman Rank (S), Kendall's Tau (K) coefficients along with corresponding p-values. The linear classifiers lack a correlation value since the evenness is constant (being the gradient constant as well), thus resulting in a not defined correlation

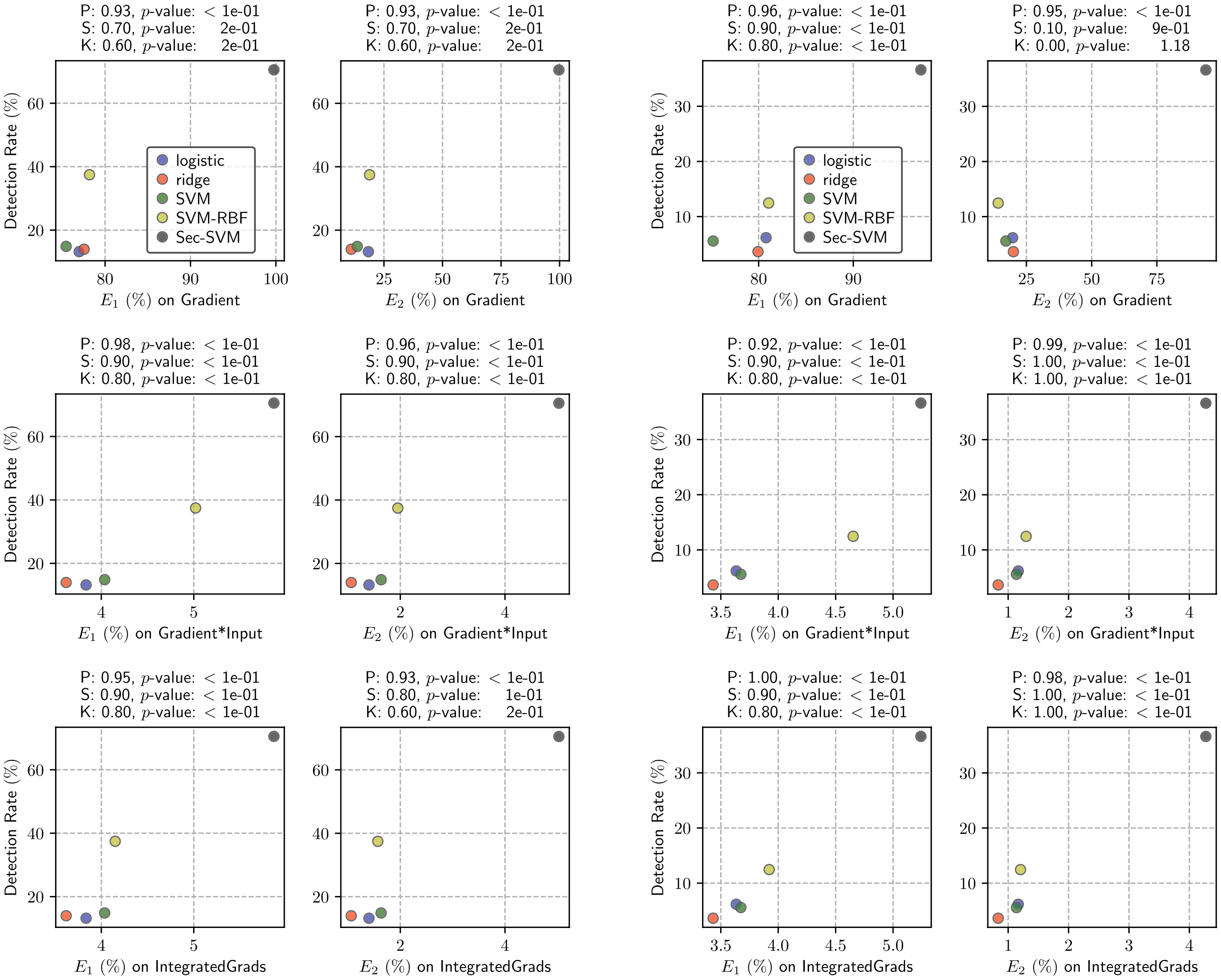
		Gradient		Gradient*Input		Int. Gradients	
		$\mathcal{E}_1$	$\mathcal{E}_2$	$\mathcal{E}_1$	$\mathcal{E}_2$	$\mathcal{E}_1$	$\mathcal{E}_2$
(a) *Drebin*							
logistic	P			0.63, < 1e – 5	0.71, < 1e – 5	0.63, < 1e – 5	0.71, < 1e – 5
	S			0.66, < 1e – 5	0.69, < 1e – 5	0.66, < 1e – 5	0.69, < 1e – 5
	K			0.48, < 1e – 5	0.51, < 1e – 5	0.48, < 1e – 5	0.51, < 1e – 5
ridge	P			0.47, < 1e – 5	0.59, < 1e – 5	0.47, < 1e – 5	0.59, < 1e – 5
	S			0.47, < 1e – 5	0.59, < 1e – 5	0.47, < 1e – 5	0.59, < 1e – 5
	K			0.33, < 1e – 5	0.43, < 1e – 5	0.33, < 1e – 5	0.43, < 1e – 5
SVM	P			0.62, < 1e – 5	0.67, < 1e – 5	0.62, < 1e – 5	0.67, < 1e – 5
	S			0.65, < 1e – 5	0.71, < 1e – 5	0.65, < 1e – 5	0.71, < 1e – 5
	K			0.48, < 1e – 5	0.54, < 1e – 5	0.48, < 1e – 5	0.54, < 1e – 5
SVM-RBF	P	0.04, 0.709	0.68, < 1e – 5	0.80, < 1e – 5	0.77, < 1e – 5	0.89, < 1e – 5	0.91, < 1e – 5
	S	0.41, < 1e – 4	0.72, < 1e – 5	0.94, < 1e – 5	0.94, < 1e – 5	0.94, < 1e – 5	0.93, < 1e – 5
	K	0.32, < 1e – 5	0.54, < 1e – 5	0.79, < 1e – 5	0.80, < 1e – 5	0.77, < 1e – 5	0.77, < 1e – 5
Sec-SVM	P			0.77, < 1e – 5	0.81, < 1e – 5	0.77, < 1e – 5	0.81, < 1e – 5
	S			0.84, < 1e – 5	0.87, < 1e – 5	0.84, < 1e – 5	0.87, < 1e – 5
	K			0.66, < 1e – 5	0.79, < 1e – 5	0.66, < 1e – 5	0.79, < 1e – 5
(b) *Tesseract*							
logistic	P			0.40, < 1e – 4	0.49, < 1e – 5	0.40, < 1e – 4	0.49, < 1e – 5
	S			0.36, < 1e – 3	0.41, < 1e – 4	0.36, < 1e – 3	0.41, < 1e – 4
	K			0.25, < 1e – 3	0.31, < 1e – 5	0.25, < 1e – 3	0.31, < 1e – 5
ridge	P			0.18, < 1e – 1	0.10, < 1e – 1	0.18, < 1e – 1	0.10, < 1e – 1
	S			0.26, < 1e – 2	0.08, < 1e – 1	0.26, < 1e – 2	0.08, < 1e – 1
	K			0.17, < 1e – 1	0.07, < 1e – 1	0.17, < 1e – 1	0.07, < 1e – 1
SVM	P			0.46, < 1e – 5	0.31, < 1e – 2	0.46, < 1e – 5	0.31, < 1e – 2
	S			0.37, < 1e – 3	0.24, < 1e – 1	0.37, < 1e – 3	0.24, < 1e – 1
	K			0.26, < 1e – 4	0.17, < 1e – 1	0.26, < 1e – 4	0.17, < 1e – 1
SVM-RBF	P	−0.78, < 1e – 5	−0.58, < 1e – 5	0.88, < 1e – 5	0.66, < 1e – 5	0.88, < 1e – 5	0.54, < 1e – 5
	S	−0.64, < 1e – 5	−0.52, < 1e – 5	0.85, < 1e – 5	0.56, < 1e – 5	0.79, < 1e – 5	0.45, < 1e – 5
	K	−0.45, < 1e – 5	−0.35, < 1e – 5	0.67, < 1e – 5	0.41, < 1e – 5	0.61, < 1e – 5	0.31, < 1e – 5
Sec-SVM	P			0.61, < 1e – 5	0.66, < 1e – 5	0.61, < 1e – 5	0.66, < 1e – 5
	S			0.42, < 1e – 4	0.48, < 1e – 5	0.42, < 1e – 4	0.48, < 1e – 5
	K			0.30, < 1e – 4	0.35, < 1e – 5	0.30, < 1e – 4	0.35, < 1e – 5

attacks, such as botnets [33] or ransomware samples [17, 47, 57].

An interesting aspect to underline is that most of the feature sets used in previous work—the earliest as well as the newest ones—include information from Android APIs [1, 4, 19, 38, 45, 47, 48, 57]. According to Zhang et al. [67], although Android malware evolves over time, many semantics are still the same or similar, and can be caught by identifying the relations between the different APIs. In particular, several works other than Drebin [4] inspect the usage of certain APIs [1, 45] or the number of API calls [19, 57], which typically implies the design of *sparse* feature vectors for ML-based detectors. This approach is often valid for other Android components (e.g., the usage of permissions). Hence, suggesting that our analysis is likely to be relevant and applicable to many other detectors in the literature.

6.2 Adversarial attacks

According to a recent survey by Biggio et al. [13], several works questioned the security of machine learning since 2004. Two pioneering works were proposed by Dalvi et al. [22] in 2004 and by Lowd and Meek [41] in 2005. Those works, considering linear classifiers employed to perform spam filtering, demonstrated that an attacker could easily deceive the classifier at test time (*evasion* attacks) by performing a limited amount of carefully-crafted changes to an email. Subsequent works [7, 8, 11] proposed attacker models and frameworks that are still used to study the security of learning-based systems also against training-time (*poisoning*) attacks. The first gradient-based poisoning [12] and evasion [9] attacks were proposed by Biggio et al. respectively in 2012 and 2013. Notably, in [9] the authors also



(a) *Drebin* **(b)** *Tesseract*

Fig. 6 Evaluation of the evenness metrics $\mathcal{E}_1$ (left) and $\mathcal{E}_2$ (right) against the Detection Rate (FPR 1%) for the different gradient-based explanation techniques

introduced two important concepts that are still heavily used in the adversarial field, namely *high-confidence* adversarial examples and the use of a *surrogate* model. This work anticipated the discovery of the so-called *adversarial examples* against deep networks [29, 61].

The vulnerability to evasion attacks was then studied especially on learning systems designed to detect malware samples (for example, on PDF files [46, 64]), thus raising serious concerns about their usability under adversarial environments. In particular, for Android malware detectors, Demontis et al. [23] demonstrated that linear models trained on the (static) features extracted by Drebin can be easily evaded by performing a fine-grained injection of information (a more advanced injection approach that directly operates on the Dalvik bytecode has been proposed by Yang et al. [66]) by employing gradient descent-based approaches. Grosse et al. [31] have also attained a significant evasion rate on a neural network trained with the Drebin feature set. Although the adversarial robustness of other Android detectors aside from [4] was not fully explored, it is evident that employing information that can be easily injected or modified may increase the probability of the attacker to attain successful evasion.

However, as discussed in Sect. 3, adding or removing (modifying) parts of a sample to create adversarial attacks is an apparently-straightforward operation. In practise, the real-world feasibility and the constraints of these operations should be carefully evaluated. Only recently, research efforts were spent into investigating *problem-space attacks*, focusing on the generation of real evasive samples [16, 54]. Given that in the software domain there is no clear inverse mapping to the feature space, unlike in computer vision for example (so that the app's semantics are correctly preserved), this research direction remained underexplored for many years.

6.3 Explainability

Consequently to the rise of black-box models in the last decade, explainability became a hot research topic. It can be leveraged to achieve multiple goals, from justifying each prediction (the *right of explanation* required by the European General Data Protection Regulation (GDPR) [30]) to discovering new knowledge and causal relations. Explainability became increasingly popular in security as well, as providing a proper explanation of predictions can help to secure the systems against adversarial attacks.

Several approaches for interpretability have been proposed, with a particular attention to *post-hoc* explanations for black-box models. In 2016, Ribeiro et al. [55] proposed LIME, a model-agnostic technique that provides local explanations by generating small perturbations of the input sample, thus obtaining the explanations from a linear model fitted on the perturbed space. Lundberg and Lee [43] unified different techniques, including LIME, under the name of SHAP, by leveraging cooperative game theory results to identify theoretically-sound explanation methods and provide feature importance for each prediction. More recently, Lundberg et al. [42] improved this method for tree-based models, including those based on multiple trees, by mantaining the desidered properties for local explanations and enabling faithful *global* understanding of the models as well. The work by Koh and Liang [35] showed that using a gradient-based technique called *influence functions*, which is well known in the field of robust statistics, it is possible to associate each input sample to the training samples (*prototypes*) that are most responsible for its prediction. The theory behind the techniques proposed by the authors holds only for classifiers with differentiable loss functions. However, the authors empirically showed that their technique provides sensible prototypes also for classifiers with not-differentiable losses if computed on a smoothed counterpart.

Another interesting venue is the generation of high-level *concepts* rather than feature attributions. In this sense, Kim et al. [34] proposed a technique that introduces the notion of *Concept Activation Vectors* (CAVs), which evaluate the sensitivity of the models to user-defined examples defining particular concepts. Koh et al. [36] focused instead on guiding models to learn concepts at training time; such concepts are then used to predict the target samples.

Notably, as recent work started explaining malware detectors through some of the above-described techniques [51, 58], Warnecke et al. [65] discussed general and security-specific criteria to evaluate explanation methods in different security domains. Moreover, Guo et al. [32] proposed LEMNA, a method specifically designed for security tasks, i.e., that is optimized for RNN and MLP networks, and that highlights the feature dependence (e.g., for binary code analysis).

Finally, a few recent works proposed to leverage explanations for both generating and detecting attack samples. Rosenberg et al. [56] obtained from explainability algorithms the most relevant features for a malware classification task, so that those can be the first to be modified in order to generate an effective adversarial attack. Starting from the same idea, Fidel et al. [28] proposed a highly accurate detector of adversarial examples, based on the SHAP values computed for the internal layers of a DNN classifier. Also, Dombrowski et al. showed how saliency maps can be manipulated arbitrarily by applying perturbations to the input, while keeping the model's output approximately constant [26]. This is a worst case scenario where not only the prediction of the system is wrong (the perturbed malicious point evades detection), but also the explanation that might have been used to identify the vulnerability is compromised.

7 Conclusions and future work

In this paper, we empirically evaluate the correlation between multiple gradient-based explanation techniques and the *adversarial robustness* of different linear and nonlinear classifiers, trained on two popular Android applications datasets (*Drebin* and *Tesseract*), against sparse evasion attacks. To this end, we leverage two synthetic measures of the *explanation evenness*, which main advantage is not requiring any computationally-expensive attack simulations. Thus, they may be used by system designers and engineers to choose, among a plethora of different models, the one that is most resilient against sparse attacks. Our experiments show that a strong connection exists between the evenness of explanations and the adversarial robustness. This correlation is stronger when advanced explanation techniques such as Gradient*Input and Integrated Gradients are used, while considering the simple Gradient does not provide reliable information about the robustness of such classifiers.

In the future, we plan to extend our study to other malware detectors as well as other application domains. Moreover, as the proposed metrics may be used to estimate the robustness only against sparse evasion attacks in a boolean feature space, such as the one of Drebin, an interesting research direction would be to devise a similar measure that can be used to estimate the robustness of detectors working in continuous, dense feature spaces, and when the attack is subjected to different application constraints. Also, it could be interesting to assess if our vulnerability measures can be successfully applied when the attacker does not know the classifier parameters or when the model is not differentiable; in that case, a surrogate classifier would be used to explain the original unknown model function.

Finally, another interesting research avenue is to modify the objective functions used to train the considered machine learning models by adding to them a penalty which is inversely proportional to the proposed evenness metrics, in order to enforce the classifier to learn more evenly distributed relevance scores and, consequently, the model robustness.

Acknowledgements This work has been partly supported by the PRIN 2017 project RexLearn (grant no. 2017TWNMH2), and by the project PON AIM Research and Innovation 2014–2020 - Attraction and International Mobility, both funded by the Italian Ministry of Education, University and Research, and by BMK, BMDW, and the Province of Upper Austria in the frame of the COMET Programme managed by FFG in the COMET Module S3AI.

Data availability statement The datasets generated during and/or analysed during the current study are available in the `Androzoo` repository, https://androzoo.uni.lu/, and upon request at https://www.sec.tu-bs.de/~danarp/drebin/.

Declarations

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

References

1. Aafer Y, Du W, Yin H (2013) DroidAPIMiner: mining API-level features for robust malware detection in android. In: Proc. of international conference on security and privacy in communication networks (SecureComm). https://doi.org/10.1007/978-3-319-04283-1_6
2. Adadi A, Berrada M (2018) Peeking inside the black-box: a survey on explainable artificial intelligence (xai). IEEE Access 6:52138–52160
3. Allix K, Bissyandé TF, Klein J, Le Traon Y (2016) Androzoo: collecting millions of android apps for the research community. In: 2016 IEEE/ACM 13th working conference on mining software repositories (MSR), pp 468–471, IEEE
4. Arp D, Spreitzenbarth M, Hübner M, Gascon H, Rieck K (2014) Drebin: efficient and explainable detection of android malware in your pocket. In: Proc. 21st annual network & distributed system security symposium (NDSS). The Internet Society
5. Arzt S, Rasthofer S, Fritz C, Bodden E, Bartel A, Klein J, Le Traon Y, Octeau D, McDaniel P (2013) FlowDroid: precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps. In: Proceedings of the 35th ACM SIGPLAN conference on programming language design and implementation—PLDI '14, pp 259–269. ACM Press. https://doi.org/10.1145/2594291.2594299, http://dl.acm.org/citation.cfm?doid=2594291.2594299
6. Baehrens D, Schroeter T, Harmeling S, Kawanabe M, Hansen K, Müller KR (2010) How to explain individual classification decisions. J Mach Learn Res 11:1803–1831
7. Barreno M, Nelson B, Joseph A, Tygar J (2010) The security of machine learning. Mach Learn 81:121–148
8. Barreno M, Nelson B, Sears R, Joseph AD, Tygar JD (2006) Can machine learning be secure? In: Proc. ACM Symp. information, computer and comm. Sec., ASIACCS '06, pp 16–25. ACM, New York
9. Biggio B, Corona I, Maiorca D, Nelson B, Šrndić N, Laskov P, Giacinto G, Roli F (2013) Evasion attacks against machine learning at test time. In: Blockeel H, Kersting K, Nijssen S, Železný F (eds) Machine learning and knowledge discovery in databases (ECML PKDD), Part III, LNCS, vol 8190. Springer, Berlin, Heidelberg, pp 387–402
10. Biggio B, Fumera G, Roli F (2010) Multiple classifier systems for robust classifier design in adversarial environments. Int J Mach Learn Cybern 1(1):27–41
11. Biggio B, Fumera G, Roli F (2014) Security evaluation of pattern classifiers under attack. IEEE Trans Knowl Data Eng 26(4):984–996
12. Biggio B, Nelson B, Laskov P (2012) Poisoning attacks against support vector machines. In: Langford J, Pineau J (eds) 29th Int'l Conf. on Machine Learning, pp 1807–1814, Omnipress
13. Biggio B, Roli F (2018) Wild patterns: ten years after the rise of adversarial machine learning. Pattern Recogn 84:317–331
14. Cai H, Meng N, Ryder B, Yao D (2018) Droidcat: effective android malware detection and categorization via app-level profiling. IEEE Trans Inf Forensics Secur 14(6):1455–1470
15. Calleja A, Martin A, Menendez HD, Tapiador J, Clark D (2018) Picking on the family: disrupting android malware triage by forcing misclassification. Expert Syst Appl 95:113–126
16. Cara F, Scalas M, Giacinto G, Maiorca D (2020) On the feasibility of adversarial sample creation using the android system api. Information 11(9):433
17. Chen J, Wang C, Zhao Z, Chen K, Du R, Ahn GJ (2018) Uncovering the Face of Android Ransomware: characterization and real-time detection. IEEE Trans Inf Forensics Secur 13(5):1286–1300. https://doi.org/10.1109/TIFS.2017.2787905, http://ieeexplore.ieee.org/document/8241433/
18. Chen J, Wu X, Rastogi V, Liang Y, Jha S (2019) Robust attribution regularization. Adv Neural Inf Process Syst 2019:14300–14310
19. Chen L, Hou S, Ye Y, Xu S (2018) Droideye: fortifying security of learning-based classifier against adversarial android malware attacks. In: Proceedings of the 2018 IEEE/ACM international conference on advances in social networks analysis and mining, ASONAM 2018, pp. 782–789. Institute of Electrical and Electronics Engineers Inc. https://doi.org/10.1109/ASONAM.2018.8508284
20. Chen S, Xue M, Tang Z, Xu L, Zhu H (2016) Stormdroid: a streaminglized machine learning-based system for detecting android malware. In: Proceedings of the 11th ACM on Asia conference on computer and communications security, pp 377–388
21. Chen YM, Yang CH, Chen GC (2021) Using generative adversarial networks for data augmentation in android malware detection. In: 2021 IEEE conference on dependable and secure computing (DSC), pp 1–8, IEEE. https://doi.org/10.1109/DSC49826.2021.9346277, https://ieeexplore.ieee.org/document/9346277/
22. Dalvi N, Domingos P, Mausam G, Sanghai S, Verma D (2004) Adversarial classification. In: Tenth ACM SIGKDD international conference on knowledge discovery and data mining (KDD), pp 99–108. Seattle
23. 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. In: IEEE transactions on dependable and secure computing, pp 1–1. https://doi.org/10.1109/TDSC.2017.2700270
24. Demontis A, Melis, M., Pintor M, Jagielski M, Biggio B, Oprea A, Nita-Rotaru C, Roli F (2019) Why do adversarial attacks transfer?

Explaining transferability of evasion and poisoning attacks. In: 28th USENIX Security Symposium (USENIX Security 19), pp 321–338. USENIX Association, Santa Clara

25. Demontis A, Russu P, Biggio B, Fumera G, Roli F (2016) On security and sparsity of linear classifiers for adversarial settings. In: Robles-Kelly A, Loog M, Biggio B, Escolano F, Wilson R (eds) Joint IAPR Int'l workshop on structural, syntactic, and statistical pattern recognition, LNCS, vol 10029. Springer International Publishing, Cham, pp 322–332
26. Dombrowski AK, Alber M, Anders CJ, Ackermann M, Müller KR, Kessel P (2019) Explanations can be manipulated and geometry is to blame. arXiv:1906.07983
27. Feng Y, Anand S, Dillig I, Aiken A (2014) Apposcopy: semantics-based detection of Android malware through static analysis. In: Proceedings of the 22nd ACM SIGSOFT international symposium on foundations of software engineering—FSE 2014, pp 576–587. ACM Press. https://doi.org/10.1145/2635868.2635869, http://dl.acm.org/citation.cfm?doid=2635868.2635869
28. Fidel G, Bitton R, Shabtai A (2020) When explainability meets adversarial learning: Detecting adversarial examples using shap signatures. In: 2020 international joint conference on neural networks (IJCNN), pp 1–8, IEEE
29. Goodfellow IJ, Shlens J, Szegedy C (2015) Explaining and harnessing adversarial examples. In: International conference on learning representations
30. Goodman B, Flaxman S (2016) European Union regulations on algorithmic decision-making and a "right to explanation". In: AI magazine, vol 38, pp 50–57
31. Grosse K, Papernot N, Manoharan P, Backes M, McDaniel PD (2017) Adversarial examples for malware detection. In: ESORICS (2), *LNCS*, vol 10493, pp 62–79. Springer
32. Guo W, Mu D, Xu J, Su P, Wang G, Xing X (2018) Lemna: explaining deep learning based security applications. In: Proceedings of the 2018 ACM SIGSAC conference on computer and communications security, pp 364–379
33. Hijawi W, Alqatawna J, Al-Zoubi AM, Hassonah MA, Faris H (2021) Android botnet detection using machine learning models based on a comprehensive static analysis approach. J Inf Secur Appl 58:102735. https://doi.org/10.1016/j.jisa.2020.102735, https://linkinghub.elsevier.com/retrieve/pii/S2214212620308711
34. Kim B, Wattenberg M, Gilmer J, Cai C, Wexler J, Viegas F, Sayres R (2018) Interpretability beyond feature attribution: quantitative testing with concept activation vectors (TCAV). In: 35th international conference on machine learning (ICML 2018), vol 80, pp 2668–2677, Stockholm
35. Koh PW, Liang P (2017) Understanding black-box predictions via influence functions. In: International conference on machine learning (ICML)
36. Koh PW, Nguyen T, Tang YS, Mussmann S, Pierson E, Kim B, Liang P (2020) Concept bottleneck models. In: III HD, Singh A (eds) Proceedings of the 37th international conference on machine learning, *Proceedings of Machine Learning Research*, vol 119, pp 5338–5348, PMLR. http://proceedings.mlr.press/v119/koh20a.html
37. Kolcz A, Teo CH (2009) Feature weighting for improved classifier robustness. In: Sixth conference on email and anti-spam (CEAS). Mountain View
38. Li Q, Hu Q, Qi Y, Qi S, Liu X, Gao P. (2021)Semi-supervised two-phase familial analysis of Android malware with normalized graph embedding. Knowl Based Syst 218:106802. https://doi.org/10.1016/j.knosys.2021.106802, https://linkinghub.elsevier.com/retrieve/pii/S0950705121000654
39. Lindorfer M, Neugschwandtner M, Platzer C (2015) Marvin: efficient and comprehensive mobile app classification through static and dynamic analysis. In: Proceedings of the 39th annual international computers, software & applications conference (COMPSAC)
40. Lindorfer M, Neugschwandtner M, Platzer C (2015) MARVIN: efficient and comprehensive mobile app classification through static and dynamic analysis. In: 2015 IEEE 39th annual computer software and applications conference, vol 2, pp 422–433
41. Lowd D, Meek C (2005) Adversarial learning. In: Proc. 11th ACM sigkdd international conference on knowledge discovery and data mining (KDD), pp 641–647. ACM Press, Chicago
42. Lundberg SM, Erion G, Chen H, DeGrave A, Prutkin JM, Nair B, Katz R, Himmelfarb J, Bansal N, Lee SI (2020) From local explanations to global understanding with explainable AI for trees. Nature Mach Intell 2(1): 56–67. https://doi.org/10.1038/s42256-019-0138-9, http://www.nature.com/articles/s42256-019-0138-9
43. Lundberg SM, Lee SI (2017) A unified approach to interpreting model predictions. In: Advances in neural information processing systems, pp 4765–4774
44. Mahindru A, Sangal AL (2021) MLDroid-framework for Android malware detection using machine learning techniques. Neural Comput Appl 33(10):5183–5240. https://doi.org/10.1007/s00521-020-05309-4
45. Mahindru A, Sangal AL (2021) SemiDroid: a behavioral malware detector based on unsupervised machine learning techniques using feature selection approaches. Int J Mach Learn Cybern 12(5):1369–1411. https://doi.org/10.1007/s13042-020-01238-9
46. Maiorca D, Biggio B, Giacinto G (2019) Towards adversarial malware detection: lessons learned from pdf-based attacks. ACM Comput Surv (CSUR) 52(4):1–36
47. Maiorca D, Mercaldo F, Giacinto G, Visaggio CA, Martinelli F (2017) R-packdroid: Api package-based characterization and detection of mobile ransomware. In: Proceedings of the symposium on applied computing, SAC '17, pp 1718–1723. ACM, New York. https://doi.org/10.1145/3019612.3019793
48. Mariconti E, Onwuzurike L, Andriotis P, Cristofaro ED, Ross GJ, Stringhini G (2017) Mamadroid: Detecting android malware by building markov chains of behavioral models. In: NDSS. The Internet Society
49. Melis M, Demontis A, Biggio B, Brown G, Fumera G, Roli F (2017) Is deep learning safe for robot vision? Adversarial examples against the icub humanoid. In: ICCV workshop on vision in practice on autonomous robots (ViPAR)
50. Melis M, Demontis A, Pintor M, Sotgiu A, Biggio B (2019) secml: a python library for secure and explainable machine learning. arXiv:1912.10013
51. Melis M, Maiorca D, Biggio B, Giacinto G, Roli F (2018) Explaining black-box android malware detection. In: 2018 26th european signal processing conference (EUSIPCO), pp 524–528, IEEE
52. Pendlebury F, Pierazzi F, Jordaney R, Kinder J, Cavallaro L (2019) {TESSERACT}: Eliminating experimental bias in malware classification across space and time. In: 28th {USENIX} Security Symposium ({USENIX} Security 19), pp 729–746
53. Peng H, Gates C, Sarma B, Li N, Qi Y, Potharaju R, Nita-Rotaru C, Molloy I (2012) Using probabilistic generative models for ranking risks of android apps. In: Proceedings of the 2012 ACM conference on computer and communications security
54. Pierazzi F, Pendlebury F, Cortellazzi J, Cavallaro L (2020) Intriguing properties of adversarial ml attacks in the problem space. In: 2020 IEEE symposium on security and privacy (SP), pp 1332–1349, IEEE
55. Ribeiro MT, Singh S, Guestrin C (2016) "why should i trust you?": explaining the predictions of any classifier. In: 22nd ACM SIGKDD Int'l Conf. Knowl. Disc. Data Mining, KDD '16, pp 1135–1144. ACM, New York
56. Rosenberg I, Meir S, Berrebi J, Gordon I, Sicard G, David EO (2020) Generating end-to-end adversarial examples for malware

classifiers using explainability. In: 2020 international joint conference on neural networks (IJCNN), pp 1–10, IEEE

57. Scalas M, Maiorca D, Mercaldo F, Visaggio CA, Martinelli F, Giacinto G (2019) On the effectiveness of system api-related information for android ransomware detection. Comput Secur 86:168–182
58. Scalas M, Rieck K, Giacinto G (2021) Explanation-driven characterization of android ransomware. In: ICPR'2020 workshop on explainable deep learning—AI, pp 228–242. Springer, Cham. https://doi.org/10.1007/978-3-030-68796-0_17
59. Shrikumar A, Greenside P, Shcherbina A, Kundaje A (2016) Not just a black box: learning important features through propagating activation differences
60. Sundararajan M, Taly A, Yan Q (2017) Axiomatic attribution for deep networks. In: Proceedings of the 34th international conference on machine learning-vol 70, pp 3319–3328. JMLR. org
61. Szegedy C, Zaremba W, Sutskever I, Bruna J, Erhan D, Goodfellow I, Fergus R (2014) Intriguing properties of neural networks. In: International conference on learning representations. arxiv:1312.6199
62. Tam K, Khan SJ, Fattori A, Cavallaro L (2015) CopperDroid: automatic reconstruction of android malware behaviors. In: Proc. 22nd annual network & distributed system security symposium (NDSS). The Internet Society
63. Tramer F, Carlini N, Brendel W, Madry A (2020) On adaptive attacks to adversarial example defenses. In: Larochelle H, Ranzato M, Hadsell R, Balcan MF, Lin H (eds) Advances in neural information processing systems, vol 33, pp 1633–1645. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2020/file/11f38f8ecd71867b42433548d1078e38-Paper.pdf
64. Šrndic N, Laskov P (2014) Practical evasion of a learning-based classifier: a case study. In: Proc. 2014 IEEE symp. security and privacy, SP '14, pp 197–211. IEEE CS, Washington, DC
65. Warnecke A, Arp D, Wressnegger C, Rieck K (2020) Evaluating explanation methods for deep learning in security. In: 2020 IEEE european symposium on security and privacy (EuroS&P), pp 158–174. IEEE, Genova. https://doi.org/10.1109/EuroSP48549.2020.00018
66. Yang W, Kong D, Xie T, Gunter CA (2017) Malware detection in adversarial settings: exploiting feature evolutions and confusions in android apps. In: ACSAC, pp 288–302. ACM
67. Zhang X, Zhang Y, Zhong M, Ding D, Cao Y, Zhang Y, Zhang M, Yang M (2020) Enhancing State-of-the-art Classifiers with API Semantics to Detect Evolved Android Malware. In: Proceedings of the 2020 ACM SIGSAC conference on computer and communications security, pp 757–770. ACM, New York. https://doi.org/10.1145/3372297.3417291

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.