Debiasing Android Malware Datasets: How Can I Trust Your Results If Your Dataset Is Biased?

Tomás Concepción Miranda, Pierre-Francois Gimenez, Jean-François Lalande, Valérie Viet Triem Tong, and Pierre Wilke

***Abstract*— Android security has received a lot of attention over the last decade, especially malware investigation. Researchers attempt to highlight applications' security-relevant characteristics to better understand malware and effectively distinguish malware from benign applications. The accuracy and the completeness of their proposals are evaluated experimentally on malware and goodware datasets. Thus, the quality of these datasets is of critical importance: if the datasets are outdated or not representative of the studied population, the conclusions may be flawed. We specify different types of experimental scenarios. Some of them require unlabeled but representative datasets of the entire population. Others require datasets labeled with valuable characteristics that may be difficult to compute, such as malware datasets. We discuss the irregularities of datasets used in experiments, questioning the validity of the performances reported in the literature. This article focuses on providing guidelines for designing debiased datasets. First, we propose guidelines for building representative datasets from unlabeled ones. Second, we propose and experiment a debiasing algorithm that, given a biased labeled dataset and a target representative dataset, builds a representative and labeled dataset. Finally, from the previous debiased datasets, we produce datasets for experiments on Android malware detection or classification with machine learning algorithms. Experiments show that debiased datasets perform better when classifying with machine learning algorithms.**

***Index Terms*— Datasets, malware, experiments.**

I. Introduction

Due to the growing number of devices equipped with the Android operating system, attackers have developed a large number of malicious applications that may steal personal information, demand ransoms, or make smartphones unavailable. In response to that phenomenon, researchers have tackled the question of determining whether an application is malicious or benign and the question of classifying an application inside malware families. Most of the work in this area follows an experimental approach, and the proposed solutions must be evaluated on datasets mirroring the possible set of Android applications.

Over time, some datasets become extensively used and at some point, become outdated even if they are the latest to provide good information [1]–[3]. Results are often impressive, with detection ratios greater than 99% for some articles [4]–[6]. However, for these results to have any practical significance, the evaluation methodology and the nature of the datasets should be thoroughly inspected. In particular, special care should be taken to ensure that the experimental datasets are *representative* of the overall population of Android applications *in the wild*. Pendlebury *et al.* [7] identified the problem of bias in test datasets used to evaluate machine learning classifiers, which leads to overestimating the performance of an algorithm compared to its use in real-life scenarios. To evaluate an algorithm's performance on a population of applications, we need a dataset that is representative of the population and that is labeled with the sought answers. Such answers require manual analysis of each application which is often a non-trivial task.

This article presents four main contributions:

- A methodology for building, by random sampling, a non-labeled dataset that is representative of the considered population of Android applications;
- A debiasing algorithm that, given an initial labeled and non-representative dataset and a representative target dataset, outputs a labeled and representative dataset;
- An experimental study on different datasets that have been debiased.
- An evaluation of the usefulness of debiased datasets when classifying malware/goodware samples using machine learning algorithms.

The article is structured as follows. Section II provides background information on the various datasets available to the research community and the type of experiments performed with them. Section III describes related work on the mitigation of irregularities that affect datasets. Section IV introduces the notion of representativity we will be using in the rest of this article. Then, Section V gives a methodology to extract a representative dataset from a population, and Section VI proposes a method to evaluate whether a dataset is representative of a population. Section VII introduces an algorithm that starts from a labeled dataset and builds a representative one. Section VIII experiments this algorithm on benign and malware datasets.

II. Datasets for Experiments

To perform analyses on Android applications, researchers use published datasets as well as online repositories. To clarify

Manuscript received June 23, 2021; revised November 22, 2021 and March 15, 2022; accepted May 18, 2022. Date of publication June 3, 2022; date of current version June 15, 2022. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Chunyi Peng. *(Corresponding author: Jean-François Lalande.)*

The authors are with the CentraleSupélec, Inria, CNRS, University of Rennes 1, IRISA, 35000 Rennes, France (e-mail: jean-francois.lalande@irisa.fr).

Digital Object Identifier 10.1109/TIFS.2022.3180184

this landscape, we discuss the different datasets and categorize their corresponding experimental scenarios.

A. Sources, Archives and Specific Datasets

Websites called "markets" are the different Android application distribution platforms available to the public, e.g., Google Play, Aptoide, Cafe Bazaar, F-Droid, MiKandi, Appland. Google Play, the first and largest of them, is supposed to be a source of new benign applications (goodware). Indeed, Google uses services such as the `bouncer`[1] and Google Play Protect,[2] which analyze applications before publishing them. There are also country-specific markets, particularly several Chinese markets (e.g., Baidu, Tencent, 360, 25PP, HiAPK, Wandoujia), and vendor-specific markets (e.g., Huawei, OPPO, Xiaomi, Lenovo). These markets evolve through time, as new applications get published and old ones are withdrawn.

There are also archives datasets where applications are never removed. AndroZoo [8], APKMirror[3] and APKPure,[4] are examples of archives.

Lastly, specific datasets provide a variety of applications that share a common characteristic, for example, being a malware, being repackaged applications, or being a specific type of malware (ransomware, dropper, etc.). Genome [2], Drebin [1], Andrubis [9], Kharon [10], AMD [3], CICAndMal2017 [11], VirusShare [12], UpDroid [13], Koodous [14], and VirusTotal [15] are examples of specific datasets (and mainly malware datasets). Some datasets are mostly *static*, meaning they are not regularly updated and therefore become rapidly obsolete.

B. Experimental Scenarios

We have identified three types of scenarios typically experienced by security researchers. Each of these scenarios has different issues and therefore requires the use of datasets with different expected properties.

1) Markets Profiling: Researchers may want to estimate a statistical characteristic of a market [8], [16], [17], e.g., "What proportion of Google Play applications have an API version lower than some threshold?." For this case, one needs to obtain a dataset which is *representative* of the market, where results of analysis on this dataset can be extrapolated to the whole market. Section V gives more details on how such a representative dataset should be extracted.

2) Algorithm Evaluation: Some experiments aim to compute a characteristic of interest from an application. Examples of interesting characteristics include detecting malicious behavior in an application [18], whether the sample is repackaged [19], identifying its malware family [20], detecting similar applications [21], whether it leaks personal information to the network [22]. To answer these questions, researchers implement algorithms using various analysis techniques, and run them on a dataset to estimate their precision. This dataset should be representative and *labeled*, i.e., the characteristic of interest for each application should be known beforehand.

3) Machine Learning: In supervised learning methods, researchers learn a model that classifies inputs into a set of classes. Two datasets are used in this setting: a training dataset and a test dataset. The training set is used during the *learning* phase, and the test set is used to evaluate the learned model on new data. Both datasets should be labeled. The training set needs to be selected carefully to avoid biases, as discussed in more detail in Section III. The test set should be representative, as for the *Algorithm evaluation* scenario.

For any of the three kinds of experiments above, building an appropriate dataset is a non-trivial task. In the rest of the paper we take as hypothesis that a researcher needs a dataset for evaluating a detection algorithm.

III. Related Work

The main irregularities in experimental datasets have already been studied [7], [23]–[25] and are listed below. We reviewed the recent literature related to Android malware experiments and confirm that these irregularities are still present. Detailed results are reported in Appendix C.

A. Class Imbalance

Class imbalance occurs when a dataset contains significantly more units of one class than of others [23]. An overrepresented class may push a learning algorithm towards preferring this class. For malware detection, if there are more samples of goodware, this has the effect of decreasing the recall but improving the precision [7].

This problem has been addressed in the machine learning community [24]: the overrepresented class is undersampled, i.e., some elements are removed while the underrepresented class is oversampled. The undersampling can be random or rely on a more sophisticated method, such as removing elements in dense clusters [26]. The most popular oversampling technique is SMOTE [27] that creates new instances from the interpolation of close samples. This technique cannot be used for malware datasets, as one cannot forge a new malware sample with specific characteristics automatically. Besides, class balancing is mostly restricted to binary classes [24], while we are interested in balancing more diverse classes.

B. Presence of Clones

Another irregularity is the presence of clones, where the abundance of several very similar apps in the training set prevents the algorithm from learning unique units [25]. Because experiments do not usually share the list of used applications, it is not possible to check the presence of clones in their datasets. Pan *et al.* [28] show a method to reduce redundancy in a dataset by extracting *representatives* of cloned units using mutual information and relative entropy.

C. Time Incoherence

This irregularity concerns the dates of the samples in the test and learning sets used in machine learning algorithms. It was shown that, when evaluating the accuracy of state-of-the-art algorithms, using a test set of applications newer than

[1] http://googlemobile.blogspot.com/2012/02/android-and-security.html
[2] https://www.blog.google/products/android/google-play-protect/
[3] https://www.apkmirror.com/
[4] https://apkpure.com

those in the training set, the performance of these algorithms decreases [7], [29]. In [30], [31] and [32], Cai states that the main reason that existing Android malware detection techniques do not sustain well lies in their failure to account for the evolutionary dynamics of the Android ecosystem. Android malware classifiers may not be sustainable – they would need to be retrained constantly for later use, or their performance could downgrade over time. A regular re-training can mitigate this problem, however, it is not always a solution, as new samples are not always available as quickly as needed or in sufficient diversity. In [33], Zhang *et al.* observes that malware samples often implement the same functionalities over time but change their implementation because of the evolution of APIs or changes of third-party libraries. These changes allow such malware to avoid detection by older classifiers. One way to limit this problem is to choose more semantic than syntactic features, as stated by Zhang *et al.* in [33]. Finally, in [34], Xu *et al.* propose to enhance malware detection over time by making necessary updates to their detection models with evolving feature sets. For that purpose they maintain a pool of different detection models initialized with a set of labeled applications using various online learning algorithms.

The previously cited works cannot solve the problem of researchers having to evaluate their algorithm on an up-to-date dataset. Indeed, this problem is different than adapting a classifier to an evolving market – even if this problem is of independent interest. The researcher would primarily focus on getting a dataset that mimics the whole set of applications that he targets. Additionally, the dataset should be labeled, for example with a malware/goodware label if the study concerns a detection algorithm. These labels can be missing in the source from which applications are picked. As a consequence, we propose in the following to define the notion of representativity of a dataset against the population it represents. A representative dataset has to be used to validate an experimental results at the time of the experiment. The representativity property is linked to a population at a period of time. A dataset that is representative at a date will not necessarily be so in the future because the population may change significantly. Such a representative dataset will help the researcher to assess results for real-world scenarios [35].

IV. Representativity

In this section, we describe the notion of *representativity* of a population that we will be aiming for in the rest of this article.

The notion of population refers to a set of applications that a user would want to consider for his use case. As an example, a user who wants to detect malware in the Google Play store would want to take the Google Play store as population. Nevertheless, it is very difficult to get a snapshot of such a market. Other markets of applications could be considered [17] but the same problem remains. As a consequence we chose in this paper AndroZoo as population. It is *not* an hypothesis that reduces the contributions of this paper: the proposed techniques work more generally beyond a particular reference dataset.

From a general perspective, we consider a population $\mathcal{P}$ of N applications. Intuitively, a dataset $\mathbf{D}$ of n elements is representative of $\mathcal{P}$ if n is significantly smaller than N and if $\mathbf{D}$ *looks like* $\mathcal{P}$ for the following purposes: 1) evaluating the proportion $\hat{p}$ of applications exhibiting a specific characteristic on $\mathbf{D}$ should yield similar results as evaluating the same proportion p on $\mathcal{P}$; 2) evaluating an algorithm on $\mathbf{D}$ should yield similar results as evaluating the same algorithm on $\mathcal{P}$.

Intuitively, we could consider that a dataset $\mathbf{D}$ is representative of $\mathcal{P}$ according to a single characteristic if the proportion of this characteristic is equal in $\mathcal{P}$ and $\mathbf{D}$. However, the exact equality of the proportions in $\mathcal{P}$ and $\mathbf{D}$ (i.e., $p = \hat{p}$) cannot always be achieved in practice. For example, if $p = 1/N$ (i.e., only one element has this characteristic in $\mathcal{P}$), then the only way to have exactly the same proportion in $\mathbf{D}$ is by selecting all N elements from $\mathcal{P}$. This is in contradiction with the first desired property of a representative dataset. Hence a margin of error δ must be taken into account, which is the maximum difference between p and $\hat{p}$ that we allow considering that $\mathbf{D}$ is representative of $\mathcal{P}$. Intuitively, a dataset is representative of a population if it has the same proportions, up to a maximum difference of δ.

This notion can be easily extended to multiple characteristics. Consider a set of d characteristics $\{c_1, \ldots, c_d\}$ that can take respectively $v_1, \ldots, v_d$ values. This generates $v_1 \times \ldots \times v_d$ different combinations. In the following, we call these combinations *classes*, and we denote $\mathcal{K}$ this set of classes. Intuitively again, we could consider that a dataset $\mathbf{D}$ is representative of $\mathcal{P}$ according to a set of classes $\mathcal{K}$ if the proportion of each class $k \in \mathcal{K}$ is equal in $\mathcal{P}$ and $\mathbf{D}$.

Definition 1: Let $0 \leq \delta \leq 1$. *A dataset* $\mathbf{D}$ *is* δ*-representative of a population* $\mathcal{P}$ *for a finite set of classes* $\mathcal{K}$ *if* $\forall k \in \mathcal{K}, |p_{\mathbf{D},k} - p_{\mathcal{P},k}| \leq \delta$, *where* $p_{\mathbf{D},k}$ *(resp.* $p_{\mathcal{P},k}$*) is the proportion of class* k *in* $\mathbf{D}$ *(resp.* $\mathcal{P}$*).*

Characteristics can represent various properties of an application. For example, file size, number of classes in the code or whether the application reads SMS messages are examples of characteristics. Note that a class is simply a set of applications sharing the same set of low-level characteristics, and is not related with the notion of malware families.

V. Sampling Representative Datasets

In this article, we consider the population of Android applications to be AndroZoo, because this is one of the largest datasets available, it is regularly updated from other markets and malware databases, and no application is removed from it. At the end of 2020, AndroZoo contained 13 696 936 different APK. In this section, we focus on the construction of a non-labeled δ-representative dataset of AndroZoo by sampling, i.e., random drawing.

The standard way to construct a representative dataset consists of drawing n random samples without replacement [36]. Because elements are drawn randomly, a very large fraction of the population would be necessary to ensure, with 100% confidence, an error smaller than δ. In statistics, one is generally more interested in probabilistic guarantees such as "with 95% chance, the error is smaller than δ." Intuitively, the more certainty one seeks, the more elements should be drawn from $\mathcal{P}$. This confidence level is usually denoted C.

We recall standard statistical equations to estimate the minimal size n to ensure the randomly drawn dataset $\mathbf{D}$ is representative of the population. We show that the minimum size of a representative dataset depends primarily on the number of characteristics for which the dataset must be representative.

A. Dataset Size for a Single Boolean Characteristic

Once the expected margin of error δ and the expected confidence level C are defined (according to the requirements of the experiment), a lower bound on the size n of the dataset $\mathbf{D}$ can be calculated by using Equation (1). This equation is obtained by rearranging the margin of the classical error formula [37] using the finite population correction [36], where $z(C)$ is the standard deviation value at C confidence level for the standard normal distribution with mean 0 and standard deviation of 1 (the derivation of this equation is detailed in the appendix).

$$n \geq N\left(\frac{4(N-1)\delta^2}{z(C)^2}+1\right)^{-1} \tag{1}$$

This computation is based on some light assumptions, notably that n is not too small (at least 30 [38]) and that the proportion p is not very close to 0 or 1. In such cases, other confidence interval formulas can be used, like the Hoeffding inequality [39].

The effect of N is limited when it is large. The minimum size of a random sample for 3 014 560 applications in Google Play with the values $\delta = 0.01$ and $C = 99\%$ is $n = 16\,497$ while it is $n = 16\,568$ for AndroZoo, a dataset four times larger. For arbitrarily large datasets, $n = 16\,588$ examples are sufficient. The influence of N is more significant for smaller populations. For example, with a population of $N = 15\,000$ elements, one only needs $n = 7163$ examples.

B. Dataset Size for one or Multiple Characteristics

The calculation of the size of the dataset given in the previous subsection only holds for a single boolean characteristic, so when there are only two classes. If one is interested in several characteristics or in a non-boolean characteristic, then there are more classes for the dataset to be representative of, and intuitively it means that the minimal size of a dataset representative of a population is larger.

As with a single boolean characteristics, the margin of error can be used to get a sample with an error up to δ. The Bonferroni correction [40] states that correctly representing a set of classes $\mathcal{K}$ with a confidence level C using random sampling is more likely than correctly representing one boolean characteristic with a confidence level $1-(1-C)/(|\mathcal{K}|-1)$. Therefore, we can modify Equation (1) by replacing $z(C)$ by $z(1-\frac{1-C}{|\mathcal{K}|-1})$. Hence, the dataset $\mathbf{D}$ will correctly represent $|\mathcal{K}|$ classes with a margin error δ with a probability higher than C if n satisfies the following inequation:

$$n \geq N\left(\frac{4(N-1)\delta^2}{z\left(1-\frac{1-C}{|\mathcal{K}|-1}\right)^2}+1\right)^{-1} \tag{2}$$

Remark that, in the case of a single boolean characteristic, $|\mathcal{K}| = 2$ and equations (1) and (2) are equal.

In the experiments of Section VI, we will observe 2411 non-empty classes in $\mathrm{AZ20}_{30k}$, the reference dataset. With the values $\delta = 0.015$, $C = 99\%$ and $|\mathcal{K}| = 36\,864$, the minimum size of a random sample is $n = 29\,099$ for Google Play, $n = 29\,320$ for AndroZoo and $n = 29\,383$ for datasets with arbitrarily large N.

VI. Evaluation of Existing Datasets

We now assume that an algorithm or a machine learning model evaluation has to be performed, as described in Section II-B. Such evaluation needs a labeled dataset, i.e., one whose ground truth is known. It should also be representative of the entire population. Otherwise, results could be erroneous due to biases in the dataset. In the case of Android applications, some characteristics like "*being malware*" are not easily computable, and only small datasets or old datasets contain such information.

For a given hard-to-compute characteristic, we separate the infinite number of other characteristics into two groups. In the first group, the characteristics that are likely to be related to the hard-to-compute characteristic. For example, using a cryptographic library could be related to being a ransomware or not. In the second group, the characteristics are not related to this hard-to-compute characteristic. We would like the datasets used to be homogeneous to the population for the characteristics of this second group.

We now propose a homogeneity test to assess if a dataset is homogeneous to a population for a finite set of characteristics. To determine this finite set, we consider the hard-to-compute characteristic "being a malware" and we choose a set of characteristics that *are not* related to it. Then, we test the homogeneity of existing datasets with the population for this set.

A. Choosing Characteristics

The more characteristics are selected, the more combinations there are. Since this number of combinations is exponential in the number of characteristics, it can quickly grow bigger than any dataset size as the number of selected characteristics increases. Therefore, it is necessary to limit the number of characteristics. Besides, numerical characteristics should be reduced to a few values to reduce this combination's number. For example, the APK size can be projected to a few intervals of size to distinguish small, medium, and large applications.

As a consequence, in the following experiments, we choose to rely on the following set of characteristics that **are not** related to security:

- APK size (4 values)
  0) 1 MB $<$ size
  1) 1 MB $\leq$ size $\leq$ 5 MB
  2) 5 MB $\leq$ size $\leq$ 20 MB
  3) size $>$ 20 MB
- year (9 values): {$<$ 2011, 2011-2012, 2012-2013, 2013-2014, 2014-2015, 2015-2016, 2016-2017, 2017-2018, $>$ 2018}

- Internet permission (bool)
- External or internal storage access (bool)
- Uses Google play services (bool)
- Generates UUIDs (bool)
- Vibrate phone permission (bool)
- NFC permission (bool)
- Bluetooth permission (bool)
- Performs HTTP request (bool)
- Uses JSON objects (bool)
- Specifies User-Agent (bool)

These characteristics can be easily extracted using Droidlysis [41],[5] a property extraction tool. With such characteristics, 36 864 ($4 \times 9 \times 2^{10}$) classes would be possible, but in practice, we observe 2874 non-empty classes in the three studied datasets (AZ20$_{30k}$, $\mathbf{D}_{mix}$ and AMD) presented later in Figure 1.

These characteristics are diversified and easy to evaluate statically, limiting the computational cost of our experiment. It would be possible to add some dynamic characteristics, such as the number of executed activities or methods, but it would require executing each application. These characteristics will be used to study the homogeneity of the datasets of the literature against an extract of the population, AndroZoo in 2020.

B. Statistical Test of Homogeneity for Datasets

To evaluate whether a dataset $\mathbf{D}$ can be considered to be not statistically homogeneous to a population $\mathcal{P}$ for a finite number of characteristics, we propose to use the χ^2 statistical test of homogeneity [42] and compare $\mathbf{D}$ with a second dataset $\mathbf{D}_r$, a large dataset randomly drawn from $\mathcal{P}$ (see Section V). The null hypothesis of the test is H_0 : $\mathbf{D}$ *and* $\mathbf{D}_r$ *are drawn from the same population* $\mathcal{P}$ *(with replacement)*. As other statistical tests, the χ^2 test can only reject this null hypothesis but not confirm it. If the null hypothesis is rejected, we will admit that the dataset $\mathbf{D}$ is not homogeneous to $\mathbf{D}_r$, and therefore dissimilar from the population $\mathcal{P}$ for the characteristics under consideration.

To apply this test, we consider a finite set of classes $\mathcal{K}$. We denote $m_{\mathbf{D},k}$ (respectively $m_{\mathbf{D}_r,k}$) the number of samples in $\mathbf{D}$ (respectively in $\mathbf{D}_r$) of class $k \in \mathcal{K}$ and $p_{\mathcal{P},k}$ (respectively $p_{\mathbf{D},k}$ and $p_{\mathbf{D}_r,k}$) the proportion of class k in $\mathcal{P}$ (respectively in $\mathbf{D}$, in $\mathbf{D}_r$).

If the null hypothesis holds true, then the three proportions $p_{\mathcal{P},k}$, $p_{\mathbf{D},k}$ and $p_{\mathbf{D}_r,k}$ of class $k \in \mathcal{K}$ should be close. The proportion $p_{\mathcal{P},k}$ can thus be estimated by $\frac{m_{\mathbf{D}_r,k}+m_{\mathbf{D},k}}{|\mathbf{D}_r|+|\mathbf{D}|}$ and the expected number of occurrences of class k in $\mathbf{D}$ can be estimated by $E_{\mathbf{D},k,\mathbf{D}_r} = |\mathbf{D}| \times \frac{m_{\mathbf{D}_r,k}+m_{\mathbf{D},k}}{|\mathbf{D}_r|+|\mathbf{D}|}$ and in $\mathbf{D}_r$ by $E_{\mathbf{D}_r,k,\mathbf{D}} = |\mathbf{D}_r| \times \frac{m_{\mathbf{D}_r,k}+m_{\mathbf{D},k}}{|\mathbf{D}_r|+|\mathbf{D}|}$. The χ^2 value of two datasets $\mathbf{D}$ and $\mathbf{D}_r$ is defined as:

$$\chi^2 = \sum_{k \in \mathcal{K}} \frac{\left(m_{\mathbf{D},k} - E_{\mathbf{D},k,\mathbf{D}_r}\right)^2}{E_{\mathbf{D},k,\mathbf{D}_r}} + \frac{\left(m_{\mathbf{D}_r,k} - E_{\mathbf{D}_r,k,\mathbf{D}}\right)^2}{E_{\mathbf{D}_r,k,\mathbf{D}}}$$

A low value for a χ^2 test means the two sets of data $\mathbf{D}$ and $\mathbf{D}_r$ have very similar class proportions. On the contrary, a large value (larger than a critical value that depends on the number of classes and the confidence level) indicates a statistically significant difference. A χ^2 test is associated with a p-value. A low p-value (usually under 0.05) indicates that one can reject the null hypothesis with a certain confidence. A high p-value, however, does not confirm the null hypothesis, but rather produces an inconclusive result.

In the rest of this section, we investigate whether labeled datasets commonly used in the literature can be considered homogeneous to all Android applications according to the χ^2 test and whether they can be considered δ-representative for some low δ.

C. Studied Android Datasets

In this section, we focus on the following datasets:

- 6 **malware datasets: Drebin** [1], **AMD** [3], as well as **VS 2015**, **2016**, **2017** and **2018**, containing all the malware collected by VirusShare for each of these years;
- 2 **groups of datasets extracted from AndroZoo [8]:**
  - **AZ19$_{\mathbf{100k}}$**, 100 000 applications randomly drawn in 2019, and the subsets **AZ19$_{\mathbf{100k}}$ 2015**, **2016**, **2017**, **2018** restricted to applications from each year;
  - **AZ20$_{\mathbf{10k}}$**, **AZ20$_{\mathbf{20k}}$**, and **AZ20$_{\mathbf{30k}}$** three datasets randomly drawn from AndroZoo in 2020, containing respectively 10 000, 20 000 and 30 000 applications.
- 1 **mix dataset: $\mathbf{D}_{\mathbf{mix}}$** composed of 5560 malware from Drebin and 11 120 goodware from 2018 extracted from AndroZoo. $\mathbf{D}_{\mathbf{mix}}$ mimics the datasets used in previous machine learning papers [43], [44].

In this article, we use AndroZoo as the population, because it is the easiest source to draw a large number of applications. Our goal is to determine which datasets commonly used in the literature can be considered as representative of the population. AndroZoo provides 13 696 936 different APKs applications. Due to its size, it is not reasonable to calculate exactly the distribution of AndroZoo. Thus, we represent here AndroZoo through a representative dataset AZ20$_{30k}$ of 30 000 randomly chosen from it (corresponding to $\delta = 0.015$ and $C = 99\%$ for all 36 864 theoretical classes, cf. Section V-B).

We performed the comparison of AZ20$_{30k}$ with the datasets of the literature, presented in Section II. As an illustration, Figure 1 shows the proportion of applications in each of the 2874 non-empty classes of three datasets: the reference AZ20$_{30k}$ (in blue), the mix dataset $\mathbf{D}_{mix}$ (in orange) and AMD (in green). The classes are sorted in descending order of size for AZ20$_{30k}$. This figure already suggests that the three datasets are very different. We confirm this intuition by performing a χ^2 test with the AZ20$_{30k}$ dataset. Table I presents the dataset size, the number of classes represented only in AZ20$_{30k}$ (and not in the studied dataset), the number of classes common to AZ20$_{30k}$ and the studied dataset, the number of classes represented only in the studied dataset (and not in AZ20$_{30k}$), the maximum and average δ of class proportions, the value of the χ^2 test, the associated p-value, and the conclusion (or lack of) of the test between the studied dataset and AZ20$_{30k}$. All datasets, except the last extract of AndroZoo, can be considered as not homogeneous to AZ20$_{30k}$,

[5] https://github.com/cryptax/droidlysis

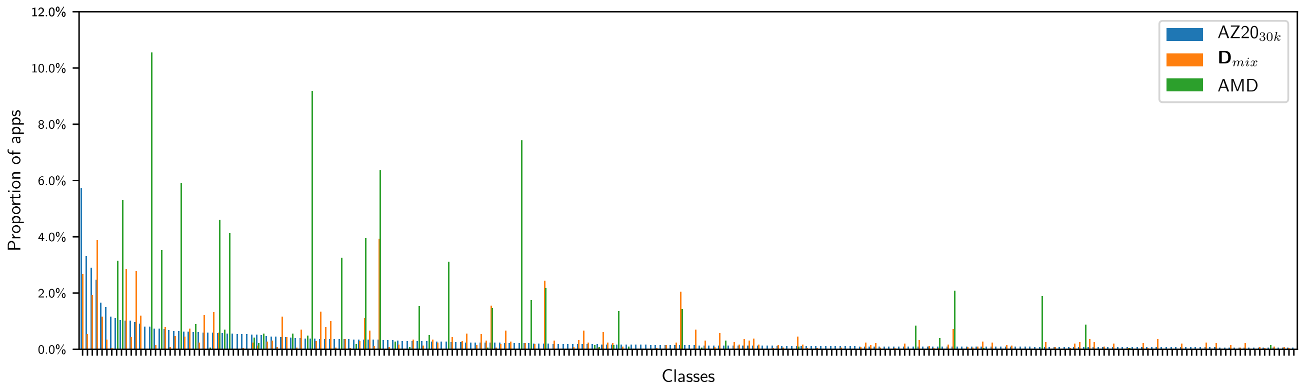

Fig. 1. Distribution of application classes for the top most 200 over 2874 classes of AZ20$_{30k}$ and compared with $\mathbf{D}_{mix}$ and AMD.

TABLE I

χ^2 Test for Homogeneity Over AZ20$_{30k}$ for a Finite Set of Non-Security Characteristics

Datasets	Dataset size	# of classes in common	# of classes only in the dataset	# of classes only in AZ20$_{30k}$	Max δ	Avg δ	χ^2 value	p-value	Conclusion
AZ20$_{30k}$	29 974	2 411	–	–	0	0	0	1	
Drebin	5 304	181	141	2 230	0.1184	0.0007	27 902	0	Reject (C=99%)
AMD	23 258	128	16	2 283	0.0974	0.0007	39 771	0	Reject (C=99%)
VirusShare 2015	28 896	250	65	2 161	0.1683	0.0007	44 537	0	Reject (C=99%)
VirusShare 2016	12 651	172	28	2 239	0.1595	0.0007	30 312	0	Reject (C=99%)
VirusShare 2017	9 945	234	39	2 177	0.0898	0.0006	22 050	0	Reject (C=99%)
VirusShare 2018	28 543	728	328	1 683	0.5057	0.0006	42 617	0	Reject (C=99%)
$\mathbf{D}_{mix}$	16 424	982	447	1 429	0.0359	0.0004	17 852	0	Reject (C=99%)
AZ19$_{100k}$	99 999	2 086	2 062	325	0.0320	0.0001	12 662	0	Reject (C=99%)
DroidBench	119	6	0	2 405	0.4773	0.0008	14 121	0	Reject (C=99%)
AZ19$_{100k}$ 2015	5 794	648	291	1 763	0.0574	0.0005	15 390	0	Reject (C=99%)
AZ19$_{100k}$ 2016	27 516	1 183	921	1 228	0.0573	0.0004	30 725	0	Reject (C=99%)
AZ19$_{100k}$ 2017	6 524	688	274	1 723	0.0381	0.0004	12 378	0	Reject (C=99%)
AZ19$_{100k}$ 2018	24 549	1 216	547	1 195	0.0285	0.0003	14 754	0	Reject (C=99%)
AZ20$_{10k}$	9 971	1 370	423	1 041	0.0443	0.0003	6 296	0	Reject (C=99%)
AZ20$_{20k}$	19 927	1 361	397	1 050	0.0245	0.0001	3 560	1	*Non-conclusive*

rejecting the null hypothesis with a p-value < 0.05. We can also remark that extracts of AndroZoo have a low average δ value, smaller that 0.001, while non-homogeneous datasets have significantly larger average δ values. This observation is consistent with our definition of representativity, as we can see a link between a low average δ value and a high p-value where the null hypothesis is not rejected.

VII. Dataset Debiasing Algorithm

The previous section strongly suggests that the datasets regularly used in Android malware scanning experiments are not representative of the entire current population of Android applications. Definitely, they are not for the common characteristics we have studied. Nevertheless, these datasets are valuable because they contain samples for which the characteristic of interest is already computed. We are now interested in producing a representative dataset of a population for a finite set of classes $\mathcal{K}$ from an initial dataset $\mathbf{B}$. Our motivation is to not have to recompute computationally expensive characteristics on a randomly drawn dataset but to take maximum advantage of $\mathbf{B}$'s labels.

As seen in the previous experiments, the χ^2 test can show that datasets deviate significantly from a population. However, this test cannot conclude that a dataset is representative of a population, only that it is not homogeneous. Besides, the χ^2 value has no intuitive meaning, and the p-value is notoriously prone to cause interpretation error while the maximal error δ is easier to interpret. For these reasons, we will use the δ-representativity when debiasing datasets.

We propose here two algorithms to limit the bias of an initial dataset $\mathbf{B}$, while minimizing the number of modifications. Algorithm 1 constructs a dataset of the same size as $\mathbf{B}$. This can lead to suboptimal solutions because the size of the optimal solution and $\mathbf{B}$ may differ. Algorithm 2 allows the size of the generated dataset to vary. Besides, we prove that this algorithm is optimal in the number of modifications to $\mathbf{B}$. These two algorithms take as input an initial base dataset $\mathbf{B}$, a target dataset $\mathbf{T}$ which we would like $\mathbf{B}$ to resemble, a source labeled dataset $\mathbf{S}$ from which we will draw applications, and a positive ratio of tolerated difference δ. The objective of both algorithms is to produce a generated dataset $\mathbf{G}$ which is δ-representative of $\mathbf{T}$. We say that a class k is *overrepresented* in $\mathbf{G}$ if $p_{\mathbf{G},k} - p_{\mathbf{T},k} > \delta$; and a class k is *underrepresented* in $\mathbf{G}$ if $p_{\mathbf{G},k} - p_{\mathbf{T},k} < -\delta$.

A. Constant-Size Debiasing Algorithm

Algorithm 1 produces a dataset $\mathbf{G}$ whose size is the same as its input $\mathbf{B}$. Intuitively, this algorithm consists in adding elements from the source **S** to the most underrepresented classes of **B** and to remove elements from **B** for the most overrepresented class. Each time an element is added, another element is removed in order to keep a constant size. With the constraint of constant size, the debiasing may be impossible for one of two reasons: either no element can be added to an underrepresented class (because the source does not contain elements of this class); or it is not possible to obtain for some class k a proportion that lies in $[p_{\mathbf{T},k} - \delta; p_{\mathbf{T},k} + \delta]$ because δ is too small.

Algorithm 1 Constant Size Debiasing Algorithm.

Input: $\mathbf{B}, \mathbf{T}, \mathbf{S}, \delta$
Output: Generated dataset $\mathbf{G}$
$\mathbf{S}' \leftarrow \mathbf{S}$; $\mathbf{G} \leftarrow \mathbf{B}$
while $max_{k \in \mathcal{K}} |p_{\boldsymbol{G},k} - p_{\boldsymbol{T},k}| > \delta$ **do**
  $k_h \leftarrow \arg\max_{k \in \mathcal{K}}(p_{\mathbf{G},k} - p_{\mathbf{T},k})$ ▷ most overrepresented class
  $k_l \leftarrow \arg\min_{k \in \mathcal{K}}(p_{\mathbf{G},k} - p_{\mathbf{T},k})$ ▷ most underrepresented class
  if $|\boldsymbol{S}'(k_l)| = 0$ **then**
    $\mathcal{K}_{\mathbf{S}} = \{k \in \mathcal{K} \mid |\mathbf{S}'(k)| > 0\}$
    if $p_{\boldsymbol{G},k_l} \geq p_{\boldsymbol{T},k_l} - \delta$ *and* $\mathcal{K}_{\boldsymbol{S}} \neq \varnothing$ **then**
      $k_l \leftarrow \arg\min_{k \in \mathcal{K}_{\mathbf{S}}} p_{\mathbf{G},k} - p_{\mathbf{T},k}$
    else return *"Impossible"*;
  $r \leftarrow$ one element of $\mathbf{S}'(k_l)$ chosen at random
  Remove r from $\mathbf{S}'$ and add it to $\mathbf{G}$
  Remove one element of $\mathbf{G}(k_h)$ chosen at random
  if $p_{\boldsymbol{G},k_h} < p_{\boldsymbol{T},k_h} - \delta$ *or* $p_{\boldsymbol{G},k_l} > p_{\boldsymbol{T},k_l} + \delta$ **then**
    return *"Impossible"*
return $\boldsymbol{G}$

Algorithm 1 is optimal in terms of number of modifications.[6] If a class is underrepresented in **B**, we need to add elements to increase its proportion and make it correctly represented. If a_k elements are added to this class while the size of **D** remains constant (i.e. $|\mathbf{D}| = |\mathbf{B}|$), the proportion of class k in **D** is $p_{\mathbf{B},k} + \frac{a_k}{|\mathbf{B}|}$. A class is not underrepresented if its proportion is greater than $p_{\mathbf{T},k} - \delta$, so we are looking for the smallest a_k such that $p_{\mathbf{B},k} + \frac{a_k}{|\mathbf{B}|} \geq p_{\mathbf{T},k} - \delta$, hence $a_k \geq |\mathbf{B}|(p_{\mathbf{T},k} - \delta - p_{\mathbf{B},k})$. By definition of the ceil function $\lceil \cdot \rceil$, $\lceil |\mathbf{B}|(p_{\mathbf{T},k} - \delta - p_{\mathbf{B},k}) \rceil$ is the lowest integer that verifies this inequality. So the minimal number of addition for an underrepresented class k is $\lceil |\mathbf{B}|(p_{\mathbf{T},k} - \delta - p_{\mathbf{B},k}) \rceil$. If this number is negative, it means that the class is not underrepresented: in that case, no addition should be made. We denote, for any class k, the minimal number of addition as $n^-_{\mathbf{B}}(k) = \max(0, \lceil |\mathbf{B}|(p_{\mathbf{T},k} - \delta - p_{\mathbf{B},k}) \rceil)$. By the same reasoning, we define the minimal number of deletions as $n^+_{\mathbf{B}}(k) = \max(0, \lceil |\mathbf{B}|(p_{\mathbf{B},k} - p_{\mathbf{T},k} - \delta) \rceil)$. In fact, we prove that Algorithm 1 makes exactly $2\max(\sum_k n^+_{\mathbf{B}}(k), \sum_k n^-_{\mathbf{B}}(k))$ modifications between its input **B** and its output **G**.

[6]Proofs are available in the appendix.

B. Optimal Debiasing Algorithm

Algorithm 1 can lead to suboptimal solutions. Algorithm 2 produces a dataset **G** without size constraint which is representative of **T**, with minimal modifications to **B**. This algorithm calls Algorithm 1 multiple times with different dataset sizes and maintains two variables: $\mathbf{G}_{best}$ is the best dataset found so far, i.e., the one that minimizes the number of modifications to **B**; and d_{min} is this distance between $\mathbf{G}_{best}$ and **B**. The algorithm starts with $\mathbf{G}_{best} = \text{"}Impossible''$ and $d_{min} = \infty$ (line 1). First, it calls Algorithm 1 (line 2) with the initial dataset **B**. If this succeeds, d_{min} is updated with the appropriate number of additions and deletions, and $\mathbf{G}_{best}$ becomes the result of Algorithm 1. Then, in lines 6 to 11, it tries to remove applications from the most overrepresented classes from **B**. Finally, in lines 14 to 14, it tries to add applications to the most underrepresented classes to **B**. For each modification to **B**, Algorithm 1 is called to update d_{min} and $\mathbf{G}_{best}$.

Algorithm 2 Optimal debiasing Algorithm

Input: **B**, **T**, **S**, δ
Output: Generated dataset
1 $d_{min} \leftarrow \infty$; $\mathbf{G}_{best} \leftarrow$ "Impossible"
2 **if** $Algo1(\boldsymbol{B}, \boldsymbol{T}, \boldsymbol{S}, \delta) \neq$ *"Impossible"* **then**
3   $d_{min} \leftarrow 2\max(\sum_k n^+_{\mathbf{B}}(k), \sum_k n^-_{\mathbf{B}}(k))$
4   $\mathbf{G}_{best} \leftarrow Algo1(\mathbf{B}, \mathbf{T}, \mathbf{S}, \delta)$
5 $\mathbf{G} \leftarrow \mathbf{B}$
6 **for** *i from* 1 *to* $d_{min} - 1$ **do** ▷ Search smaller datasets
7   **if** $\boldsymbol{G} = \varnothing$ **then break**;
8   $k_h \leftarrow \arg\max_{k \in \mathcal{K}}(p_{\mathbf{G},k} - p_{\mathbf{T},k})$
9   Remove one random element of class k_h from **G**
10   $d \leftarrow |\, |\mathbf{B}| - |\mathbf{G}|\, | + 2\max(\sum_k n^+_{\mathbf{G}}(k), \sum_k n^-_{\mathbf{G}}(k))$
11   **if** $d < d_{min}$ *and* $Algo1(\boldsymbol{G}, \boldsymbol{T}, \boldsymbol{S}, \delta) \neq$ *"Impossible"* **then**
12     $d_{min} \leftarrow d$; $\mathbf{G}_{best} \leftarrow Algo1(\mathbf{G}, \mathbf{T}, \mathbf{S}, \delta)$
13 $\mathbf{G} \leftarrow \mathbf{B}$
14 **for** *i from* 1 *to* $d_{min} - 1$ **do** ▷ Search larger datasets
15   $k_l \leftarrow \arg\min_{k \in \mathcal{K}}(p_{\mathbf{G},k} - p_{\mathbf{B},k})$
16   **if** $|\boldsymbol{S}(k_l)| = 0$ **then**
17     $\mathcal{K}_{\mathbf{S}} = \{k \in \mathcal{K} \mid |\mathbf{S}(k)| > 0\}$
18   **if** $p_{\boldsymbol{G},k_l} \geq p_{\boldsymbol{T},k} - \delta$ *and* $\mathcal{K}_{\boldsymbol{S}} \neq \varnothing$ **then**
19       $k_l \leftarrow \arg\min_{k \in \mathcal{K}_{\mathbf{S}}} p_{\mathbf{G},k} - p_{\mathbf{T},k}$
20   **else break**;
21   $r \leftarrow$ one element of $\mathbf{S}(k_l)$ chosen at random
22   Remove r from **S** and add it to **G**
23   $d \leftarrow |\, |\mathbf{B}| - |\mathbf{G}|\, | + 2\max(\sum_k n^+_{\mathbf{G}}(k), \sum_k n^-_{\mathbf{G}}(k))$
24   **if** $d < d_{min}$ *and* $Algo1(\boldsymbol{G}, \boldsymbol{T}, \boldsymbol{S}, \delta) \neq$ *"Impossible"* **then**
25     $d_{min} \leftarrow d$; $\mathbf{G}_{best} \leftarrow Algo1(\mathbf{G}, \mathbf{T}, \mathbf{S}, \delta)$
26 **return** $\boldsymbol{G}_{best}$

Theorem 1: Let $\boldsymbol{B}$, $\boldsymbol{S}$, $\boldsymbol{T}$ *be three datasets and* $\delta > 0$. *If there exists at least one* δ*-representative dataset of* $\boldsymbol{T}$ *that is composed of elements of* $\boldsymbol{B}$ *and* $\mathbf{S}$*, then Algorithm 2 produces such a dataset* $\mathbf{G}$ *such that the number of additions and* *deletions from the initial dataset* $\mathbf{B}$ *is minimal. If such a dataset doesn't exist, it returns "Impossible."*

Its worst-case temporal complexity is $O\left((\frac{1}{\delta}+|\mathbf{B}|)^2|\mathcal{K}|\right)$.

While Algorithm 2 computes the closest dataset to the original one, it may happen that the solution found contains only a few applications from the original dataset. This is an indication that the original dataset was very biased.

If $\mathbf{B}$ has been debiased towards a target dataset $\mathbf{T}$ randomly drawn from a population $\mathcal{P}$, with a tolerated difference of $\delta_{debiasing}$, we can estimate its error with the population $\mathcal{P}$ by taking into account the error $\delta_{sampling}$ and confidence interval C used to derive the size of $\mathbf{T}$ with Eq. (2): with confidence level C, the error between the debiased dataset and $\mathcal{P}$ is less than $\delta_{sampling} + \delta_{debiasing}$.

VIII. Experiments

In this last section, we are interested in producing datasets for experiments concerning Android malware detection or classification.[7] As identified in Section II-B three types of experimental scenarios can occur. The first scenario (market profiling) requires only unlabeled representative data sets that can be drawn randomly without requiring debiasing (Section V). The two other cases, Algorithm evaluation and Machine learning, require labeled datasets with different proportions of malware/goodware samples. Therefore, we propose in Section VIII-A to debias datasets containing only malware or only goodware and mix them, in Section VIII-B, to obtain mixed datasets directly usable for machine learning algorithms.

A. Debiasing Datasets

The following details the use of our algorithm to produce representative datasets and in particular the choice of the base, reference and source datasets. Through these objectives, we discuss three main questions:

- Is it always possible to debias a strongly biased dataset such as Drebin or VirusShare?
- How many modifications (del/add) are required?
- Is a debiased malware dataset stable over time?

1) Debiasing 100% *Goodware Datasets:* We have randomly drawn a New AndrooZoo Extract (NAZE-18-G) and filtered it by relying on Virus Total to keep only goodware with a date no greater than 2018. This dataset should be close to a representative dataset of the Android app population in 2020. Indeed, by taking as reference dataset $\mathrm{AZ20}_{30k}$ and as source dataset $\mathrm{AZ19}_{100k}\mathrm{G}$, we successfully build a representative goodware dataset when δ reaches 0.001, as shown in Table II. Indeed, for $\delta = 0.04$ the dataset is already homogeneous: no addition/deletion is required, but the p-value is 0. Then, we decrease δ until the p-value becomes close to 1. For $\delta = 0.001$, 19.53% of the resulting dataset are new applications and more than 50% of applications have been removed from the original one. Remark that these values of δ are far lower than the maximum δ observed in Table I.

[7] All materials used in the experiments (SHA256 hashes of samples composing the datasets and the code) are published in the Dada dataset [45].

2) Debiasing 100% *Malware Datasets:* We were first interested in investigating whether it is possible to debias a small and old dataset (Drebin) on the one hand and a more recent and larger dataset on the other hand (malware from VirusShare between 2015 and 2018). We want the produced datasets to resemble the population of AndroZoo, extracted in 2020.

The debiasing results show us that Drebin is strongly biased as we need to add from 1.8% to 12.7% of new samples, because a lot of classes of the population are underrepresented or missing in Drebin. Even with these modifications, the resulting dataset is not homogeneous to the population. On the contrary, VirusShare can be debiased with few additions (less than 2.9%) but requires removing more than 90% of the dataset. In that case, the resulting dataset obtains a p-value of 1. We conclude that the VirusShare dataset already contains enough material to be representative of AndroZoo for certain classes but that a lot of these classes are overrepresented.

3) Discussion: As seen in the results, Algorithm 2 can generate a new dataset statistically indistinguishable from the target dataset with a margin of error δ. However, the algorithm may fail (cf. grey cells in Table II) when some classes are underrepresented in the source, as discussed in Section VII-A. This limitation can be overcome by providing a large and diverse enough source dataset.

On the other hand, when the algorithm succeeds, the generated datasets' size tends to be much smaller than the base dataset. This is due to a lot of overrepresented classes, from which the algorithm removes elements at line 9. This is especially the case for historical malware datasets.

The number of classes to consider may also limit the debiasing: intuitively, more characteristics would help to resemble the target dataset better, but it spreads the elements into more classes and makes it more difficult to find elements for certain classes.

B. Building Mixed Datasets for Machine Learning

We finally produced two mixed datasets (goodware/malware) usable for training and testing machine learning algorithms. These datasets are built on top of the debiased malware datasets previously discussed. The goodware are taken from $\mathrm{NAZE}_{\mathrm{Debiased}}$18-G dataset. These two datasets are:

- **DR-AG**$_{Deb}$ (standing for Drebin+AndroZoo Goodware), that mixes the debiased version of Drebin with $\delta = 0.01$ and the goodware.
- **VS-AG**$_{Deb}$ (standing for VirusShare+AndroZoo Goodware), that mixes the debiased version of $\mathrm{VS}_{\mathrm{Debiased}}$15-18 when $\delta = 0.1$ and the goodware.

Because the goal is to use mixed datasets for machine learning algorithms, we have to follow additional recommendations from Pendlebury *et al.* [7]. We build our mixed sets by extracting random samples from goodware and malware and by following the recommendations C1 (APK of the training set should be older than the ones of the test set) and C3 (realistic malware-to-goodware ratio i.e. 5% or 10%). In Table III we call "Pivot year" the year that delimitates the train and test sets. Note also that a consequence of C1 is that applications

TABLE II
RESULTS AFTER DEBIASING DATASETS

Base Dataset |B|		**Reference** |T|	**Source** |S|	**Diff** δ	**Del**	**Add**	**Debiased dataset G** |G|	Add ratio	χ^2	p-value	Duration validity
Debiasing Goodware datasets											
NAZE-18-G	$\rightarrow$	AZ20$_{30k}$	AZ19$_{100k}$G				NAZE$_{\text{Debiased}}$18-G				
11 120		29 974	72 131	0.04	0	0	11 120	0.00	12 589	0	
				0.02	438	50	10 732	0.47	12 058	0	
				0.01	1 026	223	10 317	2.16	11 291	0	
				0.005	1 885	632	9 867	6.41	9 726	0	
				0.002 5	2 685	1 280	9 715	13.18	7 998	0	
				0.001	5 808	1 289	6 601	19.53	5 104	0.999 44	
Debiasing Malware datasets											
Drebin	$\rightarrow$	AZ20$_{30k}$	VS 15-18, AMD				Drebin$_{Debiased}$				
5 304		29 974	103 293	0.04	892	81	4 493	1.80	25 781	0	
				0.02	3 634	116	1 786	6.49	20 178	0	
				0.01	4 596	103	811	12.70	15 833	0	
				0.005							
VS 15-18	$\rightarrow$	AZ20$_{30k}$	Drebin, AMD, AZ-17-18-M				VS$_{\text{Debiased}}$15-18				
80 035		29 974	32 562	0.04	66 475	21	13 581	0.15	16 709	0	
				0.02	75 825	31	4 241	0.73	7 406	0	
				0.01	78 074	43	2 004	2.15	4 349	1	
				0.005	79 063	29	1 001	2.90	915	1	
				0.002 5							
Debiasing the VirusShare dataset over time											
VS 15	$\rightarrow$	AZ19$_{100k}$ 15	Drebin				VS$_{Debiased}$15				
28 896		5 792	5 304	0.04	13 877	4	15 023	0.03	29 475	0	0
				0.02	19 832	90	9 154	0.98	21 828	0	0
				0.01							
VS 16	$\rightarrow$	AZ19$_{100k}$ 16	Drebin, VS 15				$VS_{Debiased}$16				
12 651		27 516	34 200	0.04	4 549	1	8 103	0.01	23 204	0	0
				0.02							
VS 17	$\rightarrow$	AZ19$_{100k}$ 17	Drebin, VS 15-16, AMD				$VS_{Debiased}$17				
9 945		6 524	70 109	0.04	580	0	9 365	0.00	21 073	0	0
				0.02							
VS 18	$\rightarrow$	AZ19$_{100k}$ 18	Drebin, VS 15-17, AMD				$VS_{Debiased}$18				
28 543		24 549	80 054	0.04	27 644	7	906	0.77	4 997	0.356 52	1
				0.02	28 189	7	361	1.94	1 809	1	1
				0.01							

in the training and test sets are disjoint (which correspond to an added C4*: Train $\cap$ Test $= \varnothing$). For C2 (temporal window consistency) we do not enforce the constraint as we already balanced the classes on the year when performing the debiasing algorithms. Nevertheless, we can enforce them if needed, at the cost of reducing the output size of the training/tests sets. For example, the set VS-AG-C2$_{Deb}$ has fewer samples than VS-AG$_{Deb}$. In the rest of the paper we only use the set that does not enforce C2 to decrease the quantity of results to discuss.

We recomputed the average δ of these mixed datasets: it is still quite low (worst case of 0.0012 for VS-AG$_{Deb}$). The p-value falls to zero for some sets because applying the constraints of Pendlebury *et al.* drives the datasets away from the population. Nevertheless, we believe that respecting the time consistency for machine learning algorithms is of higher importance.

Finally, for evaluating these datasets when used for machine learning purposes, Table III includes additional datasets:

- **VS-AG**$_{Deb,\delta=0.4}$ a relaxed version of the debiasing of VS$_{\text{Debiased}}$15-18 with $\delta = 0.4$. This dataset as a p-value of 0 but contains more samples in the training set.
- **DR-AG**: a mixed version of Drebin and goodware.
- **VS-AG**: a mixed version of VS 15-18 and goodware.
- **ACT14** and **ACT17**: a mixed version of AndroCT [46] with pivot year 2014 and 2017 respectively.
- **AZL14** and **AZL17**: a mixed version of random samples extracted from AZ20$_{30k}$ for which we took the goodware/malware label of AndroZoo as an oracle.

All these datasets will be used to compare the efficiency of machine learning algorithms when using our debiased datasets. In particular, AZL14 and AZL17 are the biggest extract of the population. AndroCT [46] is a new recent datasets that cover ten years of applications: it will be used to observe its performances when learning with it and testing on AZL14/17.

C. Classifying With Machine Learning

We evaluate the performance of various machine learning classifiers depending on their training datasets to assess the contribution of debiased datasets. The datasets used in this experiments are those computed in the previous section, cf. Table III. The machine learning classifiers rely on a total of 262 characteristics. The first 12 characteristics are those used to debiased the datasets that are not related to security. Additionally, we included the characteristics related to security computed from two sources:

TABLE III
MIXED DATASETS FOR MACHINE LEARNING TRAINING AND EVALUATION

Malware	**Goodware**	**Pivot** year	**Set**	**Mal** prop.	**Good** prop.	**Constraints [7]** C1	C2	C3	C4*	**Mixed dataset** Name	Size	Avg δ	p-value
$\text{Drebin}_{\text{Debiased}}$ ($\delta = 0.01$)	$\text{NAZE}_{\text{Debiased}}$18-G ($\delta = 0.001$)	2014	Training	50%	50%	✓	×	✓	✓	DR-AG$_{Deb}$	1614	0.017	0
						✓	✓	✓	✓	DR-AG-C2$_{Deb}$	572	0.054	0
			Test	10%	90%	✓	×	✓	✓	DR-AG$_{Deb}$	40	0.071	1
						✓	✓	✓	✓	DR-AG-C2$_{Deb}$	40	0.089	1
$\text{VS}_{\text{Debiased}}$15-18 ($\delta = 0.01$)	$\text{NAZE}_{\text{Debiased}}$18-G ($\delta = 0.001$)	2017	Training	50%	50%	✓	×	✓	✓	VS-AG$_{Deb}$	3492	0.012	1
						✓	✓	✓	✓	VS-AG-C2$_{Deb}$	2858	0.013	1
			Test	10%	90%	✓	×	✓	✓	VS-AG$_{Deb}$	1333	0.057	0
						✓	✓	✓	✓	VS-AG-C2$_{Deb}$	1333	0.057	0
$\text{VS}_{\text{Debiased}}$15-18 ($\delta = 0.04$)	$\text{NAZE}_{\text{Debiased}}$18-G ($\delta = 0.01$)	2017	Training	50%	50%	✓	×	✓	✓	VS-AG$_{Deb,\delta=0.4}$	16862	0.018	0
			Test	10%	90%	✓	×	✓	✓	VS-AG$_{Deb,\delta=0.4}$	2095	0.057	0
Drebin	AZ19$_{100k}$G	2014	Training	50%	50%	✓	×	✓	✓	DR-AG	10608	-	-
VS 15-18	AZ19$_{100k}$G	2017	Training	50%	50%	✓	×	✓	✓	VS-AG	133244	-	-
			Test	10%	90%	✓	×	✓	✓	VS-AG	6113	-	-
AndroCT [46]		2014	Training	50%	50%	✓	×	✓	✓	ACT14	19351	0.065	0
			Test	10%	90%	✓	×	✓	✓	ACT14	12241	0.057	0
		2017	Training	50%	50%	✓	×	✓	✓	ACT17	26389	0.054	0
			Test	10%	90%	✓	×	✓	✓	ACT17	4580	0.093	0
AZ20$_{30k}$ with labels		2014	Training	50%	50%	✓	×	✓	✓	AZL14	4890	0.015	0
			Test	10%	90%	✓	×	✓	✓	AZL14	8785	0.057	0
		2017	Training	50%	50%	✓	×	✓	✓	AZL17	9474	0.019	0.894
			Test	10%	90%	✓	×	✓	✓	AZL17	2033	0.059	0

TABLE IV
MEAN AND MAX AUC OF MACHINE LEARNING CLASSIFIERS DEPENDING ON THEIR TRAINING SET AND THEIR TEST SET WHEN THE PIVOT YEAR IS 2017. IN BOLD FACE: BEST AUC ON A TEST SET

Test / Train	**ACT17** Mean	Max	**AZL17** Mean	Max	**VS-AG** Mean	Max	**VS-AG$_{Deb}$** Mean	Max
ACT17	0.62	0.65	0.67	0.71	0.76	0.85	0.74	0.78
AZL17	**0.72**	**0.81**	**0.78**	**0.86**	0.71	0.89	0.75	0.80
VS-AG	0.62	0.68	0.66	0.73	**0.96**	**0.97**	0.74	0.84
VS-AG$_{Deb}$	0.70	0.74	0.71	0.75	0.86	0.89	**0.83**	**0.87**
DR-AG	0.54	0.57	0.53	0.60	0.57	0.63	0.54	0.58

TABLE V
MEAN AND MAX AUC OF MACHINE LEARNING CLASSIFIERS DEPENDING ON THEIR TRAINING SET AND THEIR TEST SET WHEN THE PIVOT YEAR IS 2014. IN BOLD FACE: BEST AUC ON A TEST SET

Test / Train	**ACT14** Mean	Max	**AZL14** Mean	Max	**DR-AG$_{Deb}$** Mean	Max
ACT14	0.69	0.73	0.71	0.74	0.69*	0.83*
AZL14	0.75	0.81	0.76	0.82	0.72*	0.99*
DR-AG	0.57	0.58	0.56	0.58	0.48*	0.50*
DR-AG$_{Deb}$	0.62	0.66	0.61	0.66	0.72*	0.80*

* due to the very limited size of the test set of DR-AG$_{Deb}$, these results are unreliable

- Droidlysis that computes 214 Booleans about the use of some APIs or behaviors (loading a DEX file, using cryptography, etc.);
- FalDroid [47] that computes 48 characteristics representing score values related to graph-based features (with parameters $\epsilon = 0.8, \theta = 0.1$ [47]). As FalDroid is intended to classify families, we configured the software with only two families (goodware/malware) and we extracted features related to control flow graph that manipulates sensitive APIs.

We base this evaluation on the area under the receiver operating characteristic curve (AUC), a classical metric in machine learning. This metric can be interpreted as the probability that a classifier considers a randomly chosen goodware to be more probably benign than a randomly chosen malware. A perfect classifier has an AUC of 1 (every goodware is ranked higher than all malware), meaning that there exists a threshold to separate exactly goodware and malware. A random classifier has an AUC of 0.5. Besides, this metric does not depend on some threshold selection.

We evaluate various machine learning techniques: k-nearest neighbors [48] (neighborhood-based technique), decision trees [49], random forest [50] (bagging ensemble technique with decision trees), Gaussian naive Bayes [51] (statistical model), and AdaBoost [52] (boosting ensemble models with decision trees). We used the scikit-learn [53] implementation with default parameters. Since the decision trees, random forest and AdaBoost learning algorithms are stochastic, we took the average AUC over 25 seeds.

Average and max AUC are presented in Tables IV (pivot year of 2017) and V (pivot year of 2014) where each line corresponds to a different training set and each column to a different test set. The mean AUC over the various classifiers helps to estimate the global quality of a training dataset for machine learning. The max AUC reflects what performances could be obtained with the best classifiers.

The datasets go by pair: there is one training set and one test set (cf. Table III). In Table IV, we can see that generally learning on one training set yields the best results on the associated test set (i.e., the diagonal is in bold). The unique exception is that learning on AZL17 yields better results than learning on ACT17 when testing with ACT17. In fact, we can see that the AZL17 training set allows learning the best classifiers on ACT17 (max AUC: 0.77) and AZL17 (max AUC: 0.84), probably because it is a real randomly sampled dataset, and therefore it is less prone to bias. However, this training set was built using VirusTotal as an oracle, so it is, in general, not available for training.

Moreover, the most interesting results are the comparison of the performances of VS-AG and VS-AG$_{Deb}$ on ACT17 and AZL17 test sets. We see that **the debiased dataset always performs better than the original one**. Another important result is that **it is notably better to train on the debiased dataset VS-AG$_{Deb}$ than on ACT17 (AndroCT [46])**, which is the most recent dataset we have. Finally, **the debiased test dataset VS-AG$_{Deb}$ is more difficult to predict than the test set VS-AG**. It is a sign that the bias in the test set of VS-AG makes it easier for a model to correctly classify it, probably because some difficult cases are absent.

Finally, we can remark that the DR-AG training set yields poor results on all test sets, which is an expected result, as no applications in this dataset after 2015. We tried to confirm this result with an additional experiment with the pivot year of 2014, in Table V. In this experiment, we evaluate if the debiased version of Drebin can obtain good results, even if it is old and quite small. The row in Table V concerning DR-AG$_{Deb}$ shows that this dataset is far from being effective compared to ACT14. This is due to its size and the failing of debiasing when δ reaches 0.05 (cf. Table II).

D. Discussion

This section discusses the validity of the datasets through time. We also compare these results with related works [32], [54], [55] that have studied this question and the relation with Android history.

First, Cai *et al.* [54] focus on the evolution of usage of the system's API over time, using a dedicated analysis framework [32]. They conclude three main points: the decrease of usage of callbacks for methods of the app activities' lifecycle, a stable and small usage of inter-component communication, and a stable distribution of source/sink categories of callbacks. Second, Cai *et al.* [55] characterize this evolution for goodware and malware applications. Newer malware tend to access system APIs more often through third-party libraries than older malware. The use of activities and services has increased over time, while the use of ICC components has decreased. The diversity of callback categories has also increased.

Our experiments confirm this evolution at a higher level. We study the evolution of representativeness of our debiased dataset over the years. More precisely, we debiased several extracts of VirusShare 2015, 2016, 2017 and 2018, and checked whether these debiased datasets stay representative of an evolving population over time. The target datasets are extracts of applications from AndroZoo 2019 ($AZ19_{100k}$) filtered on the corresponding period of time ($\mathbf{T} = AZ19_{100k}\ i$, with $i = 15, 16, 17, 18$). The source datasets are known datasets of malware such as Drebin and AMD, available at date i.

When the p-value confirms that the dataset is not statistically different, we tested the produced debiased version $VS_{Debiased}\ i$ with the next years' population, i.e., $AZ19_{100k}\ i+1, i+2, \ldots$. The last column indicates contains 0 if the p-value is 0: no discussion can be done for such debiased datasets. The only debiased dataset of the parts of VirusShare is VS 18. As it is the last set of applications we have, we only conclude that this set can be used during 1 year.

IX. Conclusion

In this article, we presented the different types of Android datasets used by researchers to evaluate their experiments. We argued that these datasets have problems with their construction depending on the type of experiment. For surveys on bigger datasets, we proposed the use of a margin of error to obtain a lower bound on the size of the sample to be drawn. We also proposed a methodology based on the χ^2 test to evaluate the homogeneity of a dataset with respect to a given population for a finite set of characteristics. When starting from a dataset that does not resemble the population, we introduced a debiasing algorithm to push this biased dataset towards the population's characteristics.

The choice of characteristics for which the dataset is debiased is of high importance. We propose here to debias datasets for a finite set of computable characteristics unrelated to the characteristic under study in order to keep only the relevant differences with the population.

Finally, using this algorithm, we proposed directly usable datasets for the evaluation of machine learning algorithms. We evaluated these debiased datasets using machine learning algorithms and we show that they outperform biased ones and in particular one recent dataset of the literature. Additionally, our debiased dataset, VS-AG$_{Deb}$, would be usable for further works as a difficult test set for classification experiments.

Future works concern the study of the influence of the considered characteristics over the performances of detection algorithms. In particular, limiting the study to a small number of characteristics when debiasing, prevents combinations from exploding. As any characteristic could introduce a bias in the dataset, it would be unrealistic to enumerate and balance them when building a dataset. An additional study about the influence of the characteristic over the bias is needed. It would help to understand what are the characteristics that are important when evaluating an algorithm. Additionally, studying more characteristics, especially the ones that can be extracted from dynamic analysis, is of independant interest.

Appendix

A. Equation (1) Proof

The margin of error with finite population correction [37] for a boolean characteristic of estimated probability $\hat{p}$ is:

$$\delta \leq z(C)\sqrt{\frac{\hat{p}(1-\hat{p})}{n}}\sqrt{\frac{N-n}{N-1}}$$

The term $\hat{p}(1-\hat{p})$ is maximized when $\hat{p}=0.5$, so we can bound this value by $0.5(1-0.5)=\frac{1}{4}$, i.e.:

$$\delta \leq z(C)\sqrt{\frac{\hat{p}(1-\hat{p})}{n}}\sqrt{\frac{N-n}{N-1}} \leq z(C)\sqrt{\frac{N-n}{4n(N-1)}}$$

Rearranging this inequality brings Equation (1). □

B. Algorithm Proofs

For the sake of brevity and clarity, we introduce the notation $\Delta_{\mathbf{T}} p_{\mathbf{G},k} = p_{\mathbf{G},k} - p_{\mathbf{T},k}$. So a class k is overrepresented w.r.t. $\mathbf{T}$ if $\Delta_{\mathbf{T}} p_{\mathbf{G},k} > \delta$ and underrepresented if $\Delta_{\mathbf{T}} p_{\mathbf{G},k} < -\delta$.

Theorem 2: Algorithm 1 always halts.

Proof: First, remark that if a class k is correctly represented at some iteration, i.e., if $|\Delta_{\mathbf{T}} p_{\mathbf{G},k}| \leq \delta$, then it will be correctly represented in the next iteration. Besides, a class that is overrepresented in $\mathbf{G}$ cannot become underrepresented in the following iteration (and vice versa) without the algorithm halting. Both remarks are ensured by the "If" block on line 13 that halts the program by verifying whether a class with an added element is overrepresented or a class with a removed element is underrepresented.

Let's assume that this case never happens. At each iteration of the algorithm, there is at least one class k such that $|\Delta_{\mathbf{T}} p_{\mathbf{G},k}| > \delta$ that is modified in $\mathbf{G}$: one element is added if k is underrepresented and one element is removed if k is overrepresented in $\mathbf{G}$. So $\sum_k |\Delta_{\mathbf{T}} p_{\mathbf{G},k}|$ is reduced by at least $\frac{1}{|\mathbf{B}|}$ in the updated value of $\mathbf{G}$. With enough iterations, $\sum_k |\Delta_{\mathbf{T}} p_{\mathbf{G},k}|$ will so low that every class will be correctly represented. At that point, the algorithm halts. □

Besides, a representative dataset always exists. This is necessary to prove the termination and the complexity of Algorithm 2.

Theorem 3: Let $\mathbf{T}$ be a dataset, $\delta > 0$ and $n > 0$ such as $n \geq \frac{1}{\delta}$. There exists a dataset $\mathbf{D}$ of size n such as $\mathbf{D}$ is δ-representative of $\mathbf{T}$.

Proof: To prove the existence of such dataset of size n, we construct it. More precisely, we associate to each class k a number of element b_k such as this class is δ-representative of $\mathbf{T}$, i.e., $\frac{b_k}{n} \in [p_{\mathbf{T},k} - \delta; p_{\mathbf{T},k} + \delta]$. For each class k, define a_k such as $\frac{a_k}{n} < p_{\mathbf{T},k} \leq \frac{a_k+1}{n}$. Remark that $p_{\mathbf{T},k} - \delta \leq \frac{a_k}{n} < p_{\mathbf{T},k} \leq \frac{a_k+1}{n} \leq p_{\mathbf{T},k} + \delta$ since $\delta \geq \frac{1}{n}$. As $\sum_k p_{\mathbf{T},k} = 1$, $\sum_k a_k < n$ and $\sum_k (a_k+1) \geq n$. So there exist b_k such as for all class k either $b_k = a_k$ or $b_k = a_k + 1$ and that verify $\sum_k b_k = n$.

Denote the dataset $\mathbf{D}$ such as $p_{\mathbf{D},k} = \frac{b_k}{n}$. The size of $\mathbf{D}$ is $\sum_k b_k = n$ and $\mathbf{D}$ is δ-representative of $\mathbf{T}$ since $\frac{b_k}{n} \in [p_{\mathbf{T},k} - \delta; p_{\mathbf{T},k} + \delta]$ for every class k. It proves the existence of such a dataset. □

Theorem 4: If there exists a dataset $\mathbf{D}$ consisting of elements of $\mathbf{B}$ and $\mathbf{S}$ (i.e., $\mathbf{D} \subseteq \boldsymbol{B} \cup \boldsymbol{S}$) such as $\mathbf{D}$ is representative of $\mathbf{T}$ and $|\boldsymbol{D}| = |\boldsymbol{B}|$, then Algorithm 1 returns a representative dataset. If no such dataset exists, the algorithm returns "Impossible."

Proof: Assume such dataset $\mathbf{D}$ exists. Let us first prove that the **return** on line 9 cannot be reached. This line is reached if $\mathbf{S}$ is empty or if $\Delta_{\mathbf{T}} p_{\mathbf{G},k_l} < -\delta$.

Consider the case where $\Delta_{\mathbf{T}} p_{\mathbf{G},k_l} < -\delta$: we will prove by contradiction that this case is impossible when $\mathbf{D}$ exists. Since the algorithm never removes an element of an underrepresented class, at each iteration $p_{\mathbf{G},k_l} = \frac{\mathbf{B}(k_l)+m}{n}$ where m is the number of elements added to the class from $\mathbf{S}$. If $\mathbf{S}(k_l) = 0$, then $p_{\mathbf{G},k_l}$ is maximized, i.e., $p_{\mathbf{G},k_l} \geq p_{\mathbf{D},k_l}$. So $\Delta_{\mathbf{T}} p_{\mathbf{D},k_l} \leq \Delta_{\mathbf{T}} p_{\mathbf{G},k_l} < -\delta$ which means that $\mathbf{D}$ is not representative of $\mathbf{T}$. This is a contradiction: therefore this case is not possible.

Consider the case where $\mathbf{S}$ is empty and $\Delta_{\mathbf{T}} p_{\mathbf{G},k_l} \geq -\delta$. Once again, we will prove by contradiction that this case is impossible when $\mathbf{D}$ exists. The fact that $\Delta_{\mathbf{T}} p_{\mathbf{G},k_l} \geq -\delta$ implies that $\Delta_{\mathbf{T}} p_{\mathbf{G},k_h} \geq \delta$, otherwise the "While" condition would not have been true. Denote m the number of completed iterations so far. Since one element of $\mathbf{S}$ is removed at each iteration, we conclude that initially $|\mathbf{S}| = m$. Denote $\mathbf{B_D} \subseteq \mathbf{B}$ and $\mathbf{S_D} \subseteq \mathbf{S}$ such as $\mathbf{D} = \mathbf{B_D} \cup \mathbf{S_D}$. Due to the definition of k_h and the fact that $\Delta_{\mathbf{T}} p_{\mathbf{G},k_h} \geq \delta$, it means that for each previous iterations one element has been removed from an overrepresented class. So there are at least $m+1$ elements of $\mathbf{B}$ that are absent in $\mathbf{B_D}$, i.e., $|\mathbf{B_D}| \leq |\mathbf{B}| - m$. However, since $|\mathbf{B}| = |\mathbf{D}| = |\mathbf{B_D}| + |\mathbf{S_D}|$, it means that $|\mathbf{S}| \geq m + 1$. This is in contradiction with the fact that $|\mathbf{S}| = m$. So $\mathbf{S}$ cannot be empty. Finally, the **return** on line 9 cannot be reached when $\mathbf{D}$ exists.

We now prove that the "If" condition on line 13 cannot be true in Algorithm 1 when $\mathbf{D}$ exists. For algorithm 1 to return an error on line 14, the condition $max_{k \in \mathcal{K}} |\Delta_{\mathbf{T}} p_{\mathbf{G},k}| > \delta$ must be true. Let us assume (without loss of generality) that there exists k' such as $\Delta_{\mathbf{T}} p_{\mathbf{G},k'} > \delta$ (the case $\Delta_{\mathbf{T}} p_{\mathbf{G},k'} < -\delta$ is similar).

Consider now the value of k_h. Since $k_h \leftarrow \arg\max_{k \in \mathcal{K}} \Delta_{\mathbf{T}} p_{\mathbf{G},k}$, we can conclude that $\Delta_{\mathbf{T}} p_{\mathbf{G},k_h} \geq \Delta_{\mathbf{T}} p_{\mathbf{G},k'} > \delta$. Denote $n = |\mathbf{B}|$ and $\mathbf{G}'$ the dataset obtained from $\mathbf{G}$ by removing one element from k_h and adding one element to k_l. The proportions of the classes of $\mathbf{G}'$ are:

$$p_{\mathbf{G}',k} = \begin{cases} p_{\mathbf{G},k} - \frac{1}{n} \text{ if } k = k_h \\ p_{\mathbf{G},k} + \frac{1}{n} \text{ if } k = k_l \\ p_{\mathbf{G},k} \text{ otherwise} \end{cases}$$

Our goal is to prove that the "If" condition will never be true, i.e., that $\Delta_{\mathbf{T}} p_{\mathbf{G}',k_h} \geq -\delta$ and $\Delta_{\mathbf{T}} p_{\mathbf{G}',k_l} \leq \delta$.

Let a_1 and a_2 be two naturals such as $p_{\mathbf{G},k_h} = \frac{a_1}{n}$ and $p_{\mathbf{D},k_h} = \frac{a_2}{n}$. Since $\Delta_{\mathbf{T}} p_{\mathbf{G},k_h} > \delta$ and $\Delta_{\mathbf{T}} p_{\mathbf{D},k_h} \leq \delta$ (due to $\mathbf{D}$ being representative), we can conclude that $p_{\mathbf{G},k_h} > p_{\mathbf{D},k_h}$, i.e., $a_1 > a_2$. Since $p_{\mathbf{G}',k_h} = p_{\mathbf{G},k_h} - \frac{1}{n} = \frac{a_1 - 1}{n}$ and $a_1 - 1 \geq a_2$, we conclude that $p_{\mathbf{G}',k_h} \geq p_{\mathbf{D},k_h}$ and $\Delta_{\mathbf{T}} p_{\mathbf{G}',k_h} \geq \Delta_{\mathbf{T}} p_{\mathbf{D},k_h}$. Since $\mathbf{D}$ is itself representative of $\mathbf{T}$, we know that $\Delta_{\mathbf{T}} p_{\mathbf{D},k_h} \geq -\delta$ and therefore that $\Delta_{\mathbf{T}} p_{\mathbf{G}',k_h} \geq -\delta$.

Let us show that there exists k'_l such as $\Delta_{\mathbf{T}} p_{\mathbf{G},k_l} < \Delta_{\mathbf{T}} p_{\mathbf{D},k'_l} \leq \delta$. Since $\Delta_{\mathbf{T}} p_{\mathbf{G},k_h} > \Delta_{\mathbf{T}} p_{\mathbf{D},k_h}$ and $\sum_k \Delta_{\mathbf{T}} p_{\mathbf{G},k} = \sum_k \Delta_{\mathbf{T}} p_{\mathbf{D},k}$, we can conclude that there exists k'_l such as $\Delta_{\mathbf{T}} p_{\mathbf{G},k'_l} < \Delta_{\mathbf{T}} p_{\mathbf{D},k'_l}$. Furthermore, $k'_l \in \mathcal{K}_S$ because if $\mathbf{S}$ were empty it would not be possible for $\mathbf{D}$ to have more elements than $\mathbf{G}$ for this class.

Due to the definition of k_l, $\Delta_{\mathbf{T}} p_{\mathbf{G},k_l} \leq \Delta_{\mathbf{T}} p_{\mathbf{G},k'_l} < \Delta_{\mathbf{T}} p_{\mathbf{D},k'_l} \leq \delta$. Let b_1, b_2, b_3, b_4 four naturals such as $p_{\mathbf{G},k_l} = \frac{b_1}{n}$, $p_{\mathbf{T},k_l} = \frac{b_2}{n}$, $p_{\mathbf{D},k'_l} = \frac{b_3}{n}$ and $p_{\mathbf{T},k'_l} = \frac{b_4}{n}$. We can conclude of the previous inequalities that $b_1 - b_2 < b_3 - b_4$. Therefore $b_1 - b_2 + 1 \leq b_3 - b_4$ and $\Delta_{\mathbf{T}} p_{\mathbf{G}',k_l} = p_{\mathbf{G},k_l} + \frac{1}{n} - p_{\mathbf{D},k_l} \leq \delta$. This proves that the "If" condition is never true if $\mathbf{D}$ exists.

Finally, if $\mathbf{D}$ exists, then Algorithm 1 returns a dataset. This dataset is representative because the "While" condition is true for the last iteration. When the algorithm outputs a dataset, it is representative and its size is the same as $\mathbf{B}$. If such dataset does not exist, the algorithm cannot output one. Since the algorithm always halts, it must return "Impossible." $\square$

We denote $d_H(\mathbf{A}, \mathbf{B})$ the number of modifications between $\mathbf{A}$ and $\mathbf{B}$. More precisely, we extend the Hamming distance to a distance between two sets by defining $d_H(\mathbf{A}, \mathbf{B})$ as $d_H(\mathbf{A}, \mathbf{B}) = |(\mathbf{A} \setminus \mathbf{B}) \cup (\mathbf{B} \setminus \mathbf{A})|$. $d_H(\mathbf{A}, \mathbf{B})$ is the number of elements present in only one dataset, $\mathbf{A}$ or $\mathbf{B}$. In other words, it is the minimal number of addition and deletions to transform $\mathbf{A}$ into $\mathbf{B}$ (and vice-versa).

Theorem 5: Let $\mathbf{B}$ *be a biased dataset,* $\mathbf{D}$ *a dataset representative of* P *such as* $\mathbf{D} \subseteq \boldsymbol{B} \cup \boldsymbol{S}$ *and* $\mathbf{G} = Algo1(\boldsymbol{B})$. *If* $|\boldsymbol{B}| = |\boldsymbol{D}|$, *then* $d_H(\boldsymbol{B}, \boldsymbol{D}) \geq d_H(\boldsymbol{B}, \boldsymbol{G}) = 2\max(\sum_k n^+_{\boldsymbol{B}}(k), \sum_k n^-_{\boldsymbol{B}}(k))$. *So Algorithm 1 outputs a representative dataset with minimal distance to* $\mathbf{B}$.

Proof: Let $\mathbf{A} = \mathbf{D} \setminus \mathbf{B}$ be the set of elements added to $\mathbf{B}$ and $\mathbf{R} = \mathbf{B} \setminus \mathbf{D}$ be the set of elements removed from $\mathbf{B}$. So $\mathbf{D} = (\mathbf{B} \setminus \mathbf{R}) \cup \mathbf{A}$ and $d_H(\mathbf{B}, \mathbf{D}) = |\mathbf{A}| + |\mathbf{R}|$. Due to the hypothesis $|\mathbf{B}| = |\mathbf{D}|$, we know that $|\mathbf{A}| = |\mathbf{R}|$ (there are as many additions as removals).

For any class k that is overrepresented in $\mathbf{B}$, $n^+_{\mathbf{B}}(k)$ is the minimal number of removal of elements of class k such as k is not overrepresented anymore. $n^-_{\mathbf{B}}(k) = \max(0, \lceil -|\mathbf{B}|(\Delta_{\mathbf{T}} p_{\mathbf{B},k} + \delta)\rceil)$ is similar for underrepresented classes.

To make a representative dataset from $\mathbf{B}$, one must remove at least $\sum_k n^+_{\mathbf{B}}(k)$ elements and add $\sum_k n^-_{\mathbf{B}}(k))$, i.e., $|\mathbf{A}| \geq \sum_k n^+_{\mathbf{B}}(k)$ and $|\mathbf{R}| \geq \sum_k n^-_{\mathbf{B}}(k))$. However, since $|\mathbf{A}| = |\mathbf{R}|$, we conclude that $|\mathbf{A}| = |\mathbf{R}| \geq \max(\sum_k n^+_{\mathbf{B}}(k), \sum_k n^-_{\mathbf{B}}(k))$. It means that $d_H(\mathbf{B}, \mathbf{D}) \geq 2\max(\sum_k n^+_{\mathbf{B}}(k), \sum_k n^-_{\mathbf{B}}(k))$.

Let now show that $d_H(\mathbf{B}, \mathbf{G}) = 2\max(\sum_k n^+_{\mathbf{B}}(k), \sum_k n^-_{\mathbf{B}}(k))$. At each iteration of the algorithm, if $\sum_k n^+_{\mathbf{G}}(k) > 0$, then an overrepresented class has one of its elements removed, so $\sum_k n^+_{\mathbf{G}'}(k) < \sum_k n^+_{\mathbf{G}}(k)$. For the same reason, at each iteration, $\sum_k n^-_{\mathbf{G}'}(k) < \sum_k n^-_{\mathbf{G}}(k)$. When $\sum_k n^+_{\mathbf{G}}(k) = \sum_k n^-_{\mathbf{G}}(k) = 0$, then the algorithm halts. So there are $\max(\sum_k n^+_{\mathbf{G}}(k) = \sum_k n^-_{\mathbf{G}}(k))$ iterations. Since each iteration makes two modifications of the dataset, we can conclude that $d_H(\mathbf{B}, \mathbf{G}) = 2\max(\sum_k n^+_{\mathbf{B}}(k), \sum_k n^-_{\mathbf{B}}(k))$.

So $d_H(\mathbf{B}, \mathbf{G}) \leq d_H(\mathbf{B}, \mathbf{D})$. $\square$

Proof of theorem 1: Let us first prove that at each step of the algorithm, d is equal to the number of additions and deletions between $\mathbf{B}$ and $Algo1(\mathbf{G})$. This number can be decomposed as the sum of the number of additions and deletions between $\mathbf{B}$ and $\mathbf{G}$ and between $\mathbf{G}$ and $Algo1(\mathbf{G})$. The number of modifications between $\mathbf{G}$ and $\mathbf{B}$ is simply the difference of their size, i.e., $|\,|\mathbf{B}| - |\mathbf{G}|\,|$. The number of modification between $\mathbf{G}$ and $Algo1(\mathbf{G})$ is $2\max(\sum_k n^+_{\mathbf{G}}(k), \sum_k n^-_{\mathbf{G}}(k))$ (cf. the previous proof algorithm). So the number of modifications between $\mathbf{B}$ and $Algo1(\mathbf{G})$ is $|\,|\mathbf{B}| - |\mathbf{G}|\,| + 2\max(\sum_k n^+_{\mathbf{G}}(k), \sum_k n^-_{\mathbf{G}}(k))$.

First, let us show that the algorithm always halts. This algorithm halts either if $d_{min} \neq \infty$ (so both loops have a finite number of iterations) or if the source dataset lacks some element (lines 16 to 16). If a solution $\mathbf{G}$ exists, then at some point Algorithm 2 will call Algorithm 1 with a dataset whose size is the same as $\mathbf{G}$ (either at line 2 if $|\mathbf{G}| = |\mathbf{B}|$, at line 10 if $|\mathbf{G}| < |\mathbf{B}|$ or at line 26 if $|\mathbf{G}| > |\mathbf{B}|$) and Algorithm 1 will find a solution (due to Theorem 5). If at some point it is not possible to add an element to a underrepresented class because there is no such element in the source dataset, then the algorithm halts (lines 5 to 9).

Second, we show that doing $d_{min} - 1$ iterations per loop is sufficient (lines 6 and 15). Let $\mathbf{G}_{best}$ be the best dataset found so far, associated to a distance d_{min}, and a better solution $\mathbf{G}'$ associated to a distance d' such as $d' < d_{min}$. Remark that at each step, $i = |\,|\mathbf{B}| - |\mathbf{G}|\,|$ as one element is modified per iteration. Denote j the iteration at which $\mathbf{G}'$ could be found. Since $d' = |\,|\mathbf{B}| - |\mathbf{G}|\,| + 2\max(\sum_k n^+_{\mathbf{G}}(k), \sum_k n^-_{\mathbf{G}}(k)) \geq j$ and $d' < d_{min}$, we conclude that $j \leq d_{min}$ and therefore that $\mathbf{G}'$ will be found by the algorithm. $\square$

Theorem 6: The worst-case temporal complexity of Algorithm 2 is $O\left((\frac{1}{\delta} + |\boldsymbol{B}|)^2 |\mathcal{K}|\right)$.

Proof: Remark that for all datasets $\mathbf{A}$ and $\mathbf{B}$, the number of additions and deletions required to obtain $\mathbf{B}$ from $\mathbf{A}$ is lower than or equal to $|\mathbf{A}| + |\mathbf{B}|$. Theorem 3 shows that there exists a representative dataset $\mathbf{D}$ of size $\lceil \frac{1}{\delta} \rceil$. So the minimal distance d_{min} to reach a representative dataset $\mathbf{D}$ from the initial base dataset $\mathbf{B}$ is bounded by $\lceil \frac{1}{\delta} \rceil + |\mathbf{B}|$. So Algo. 2 will do at most $2(\lceil \frac{1}{\delta} \rceil + |\mathbf{B}|)$ iterations. Each iteration calls Algo. 1, whose temporal complexity is $O(|\mathbf{G}| \cdot |\mathcal{K}|)$ where the size of $\mathbf{G}$ is at most $|\mathbf{B}| + d_{min} \leq \lceil \frac{1}{\delta} \rceil + 2|\mathbf{B}|$. So the worst-case temporal complexity of Algo. 2 is $O\left((\frac{1}{\delta} + |\mathbf{B}|)^2 |\mathcal{K}|\right)$. $\square$

C. Datasets Used in the Literature

Table VI evaluates the dataset usages of 28 articles on Android malware detection using machine learning techniques. Column "Paper" gives the name of the tool or the authors' names. Next, column "Goodware source" contains the datasets used in the experiment. Sometimes, several sets are used in a paper: we note these subsets $\mathbf{D}_i$ and we give their size in other columns accordingly. Next, columns "GS year" and "Nb Goodware" are the year range and number of goodware according to the article, if available. The next three colmuns give the same information for the used malware.

TABLE VI

DETAILS ON THE DATASET USAGE IN DETECTION EXPERIMENTS. GP = GOOGLE PLAY, VT = VIRUSTOTAL, GM = GM, DB = DREBIN, AZ = ANDROZOO, CT = CONTAGIO, VS = VIRUSSHARE, PD = PLAYDRONE, MSL = MOBISEC LAB, TZ = THEZOO, MS = MALSHARE, N/A = NOT APPLICABLE

Paper	Goodware source	Goodware source year	Goodware size	Malware source	Malware source year	Malware Size	Training set (M/G)	Test set (M/G)	Class Imbalance	Time Incoherence
MADAME [56]				GM CT VS	2010-2011 2016 2012-2015	1 242 1 923	2,784/0 (Total)		X	
DroidDetector [57]	GP	2016	20 000	CT GM	2016 2010-2011	500 1 260	880/800	880/880		X
ICCDetector [44]	GP	2014	12 026	DB	2010-2012	5 264	15 561 (mixed)	1 203/526	X	X
Androdialysis [43]	GP	2016	1 846	DB	2010-2012	5 560	10-fold cross-validation	5 560/1 846	X	X
MaMaDroid [58]	PD GP	2013-2014 2016	8 447 2 843	DB VS " " "	2010-2012 2013 2014 2015 2016	5 560 6 228 15 417 5 314 2 974	10-fold cross-validation	DB/PD 2013/PD 2014/PD 2014/GP 2015/GP 2016/GP	X	
StormDroid [59]	GP	2015	4 350	MSL GM CT	2015 2010-2011 2012-2015	2 000 1 260 360	$\frac{(900+500+100)}{1500}$	$\frac{(600+300+100)}{1000}$		X
PIndroid [4]	GP AppBrain F-Droid Getjar Aptoid Mobango	up to 2016	445	CT DB GM VT TZ[60] MS[61] VS	2012-2015 2010-2012 2010-2011 ? 2016 ? 2016	60 100 1 000 70 70 25 25	80% (from 1300/445 total)	20% (from 1300/445 total)	X	X
Vinod et al. [5]	GP ($\mathbf{D}_1, \mathbf{D}_2, \mathbf{D}_3, \mathbf{D}_4, \mathbf{D}_5$) Baidu ($\mathbf{D}_1, \mathbf{D}_2$) Koodous($\mathbf{D}_1, \mathbf{D}_2$)(1,514) 1Mobile ($\mathbf{D}_1, \mathbf{D}_2$) 9apps ($\mathbf{D}_1, \mathbf{D}_2$)	2016	3 130	DB ($\mathbf{D}_1, \mathbf{D}_2$) Koodous ($\mathbf{D}_2$) [62] ($\mathbf{D}_4$) GM ($\mathbf{D}_5$)	2010-2012 ? 2011-2015 2010-2011	 1 514 1 000 1 206	10-fold cross-validation	$\mathbf{D}_1$ 2 520/3 130 $\mathbf{D}_2$ 1 514/3 130 $\mathbf{D}_3$ 2 474/2 474 $\mathbf{D}_4$ 1 000/14 000 $\mathbf{D}_5$ 1 206/1 206	X	X
MalDozer [6]	GP	2016	38 000	GM DB *Own* "Merged"	2010-2011 2010-2012	1 000 5 500 20 000 33 000	Each malware dataset individually/37 627		X	X
IntelliAV [63]	VT "	2011-2016 2017	10 058 2 898	VT "	2011-2016 2017	9 664 2 311	9 664/10 058 2 311/2 898		X	
Martín et al. [64]	GP	2015	$\sim$ 49 000	GP	2015	69 000	10-fold cross validation with 9 subsets of 50 000 samples, varying malware concentration from 2%, 25%, 50% of the whole subset, and the number of antivirus alerts: 1-AV, 2-AV, 4-AV (4-AV, 50% subset only contains 36 000)		X	
HinDroid [65]	Comodo Cloud Security Center	2017	920 ($\mathbf{D}_1$ Tr) 198 ($\mathbf{D}_1$ Te) 15 000 ($\mathbf{D}_2$)	Comodo Cloud Security Center	2017	914 ($\mathbf{D}_1$ Tr) 302 ($\mathbf{D}_1$ Te) 15 000 ($\mathbf{D}_2$)	914/920 ($\mathbf{D}_1$) 15 000/15 000 ($\mathbf{D}_2$)	500 ($\mathbf{D}_1$) 10-fold CV ($\mathbf{D}_2$)		
RevealDroid [66]	GP (AZ)	2016	24 600	VS DB	2016 2010-2012	22 592 5 538	10-fold CV Training-Test with date separation (Training date < Test date)		X	X
Demontis et al. [67]	DB	2010-2012	121 329	DB Mod DB and CT	2010-2012 2010-2012, 2016	5 615 10 500	10 runs, where 30 000 for training, and the rest for test		X	*Too Old*
McLaughlin et al. [68]	GM (GP) McAfee Labs "	2010-2011 2016 ? 2016 ?	863 3 627 9 268	GM McAfee Labs "	2010-2011 2016 ? 2016 ?	1 260 2 475 9 902	10-fold CV, with same GW/MW ratio In training and test			X
Alzaylaee et al. [69]	Intel Security (McAfee Labs)	2016 ?	1 222	GM	2010-2011	1 222	2/3 of GW/MW	1/3 of GW/MW		X
Melis et al. [70]	DB	2010-2012	121 329	DB	2010-2012	5 615	5 runs, where 60 000 units are randomly selected for training, And the rest for testing		X	*Too Old*
MKLDroid [71]	GP1 GP2 AndroidDrawer (AD) AnZhi (AZ) AppsApk (AA) F-Droid (FD) SlideMe (SM)	2012-2014 2012-2014 2013-2016 2013-2016 2013-2016 2013-2016 2013-2016	5 000 10 000 2 399 3 027 2 481 1 007 5 770	DR VS MYST	2010-2012 2013-2014 2015	5 560 24 317 3 000	CE1: 70% of DR + GP1 CE2: 70% of VS + GP2 CE3: DR + GP1 WEx: DR + VS + GP1 + GP2 MCLEx: ⅔ of MYST + GP1	CE1: 30% of DR + GP1 CE2: 30% of VS + GP2 CE3: VS + GP2 WEx: AD + AZ + AA + FD + SM MCLEx: ⅓ of MYST + GP1	X	X
Li et al. [72]	apkpure 360 HKUST	2019	16 753	DB AMD	2010-2011 2010-2016	5 560 16 753	5-fold CV		X	X
SMART [73]	GP	2015	5 600	DB	2010-2012	5 560	10-fold CV			X
Xiao et al. [74]	GP	2016 ?	3 536	DB	2010-2013	3 567	10-fold CV			X
DroidCat [75]	GP (AZ) " " "	2016-2017 2014-2015 2012-2013 2009-2011	5 346 6 545 5 035 439	VS, AZ VS, AZ VS, AZ, GM, DB VS, AZ, GM, DB	2016-2017 2014-2015 2012-2013 2009-2011	3 450 3 190 9 084 1 254	70% of older apps	30% of newer apps	X	
Rana et al. [76]	DB	2010-2012	5 560	DB	2010-2012	5 560	80% training and 20% testing			*Too Old*
Ma et al. [77]	AZ	?	10 010	AMD	2010-2016	10 683	Everything divided in 10 parts, Then 10-CV for each part		?	?
Huang et al. [78]	GP	2018 ?	3 312	VS	?	3 312	10-fold CV			*N/A*
Liu et al. [79]	DB	2010-2012	123 453	DB	2010-2012	5 560			X	*Too Old*
HG-Learning [80]	Tencent Security Lab (Tr) " (Te)	2018 ? 2018 ?	83 784 13 313	Tencent Security Lab (Tr) " (Te)	2018 ? 2018 ?	106 912 4 433	83 784/106 912	13 313/4 433	X	
BRIDEMAID [81]	GP	2016	9 804	DB GM CT	2010-2012 2010-2011 2016	2 794	2 974 applications tested		X	X

Next, columns "Training set" and "Test set" correspond to the number of applications used in the respective sets. The notation "(M/G)" reports the Malware/Goodware for each experiment. The use of "k-fold cross validation" will be specified in these columns. Any other special cases are reported in these two columns. Finally, columns "Class imbalance" and "Time Incoherence" indicate whether the datasets used in the experiments suffer from these irregularities, introduced in Section III.

REFERENCES

[1] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck, "DREBIN: Effective and explainable detection of Android malware in your pocket," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, Reston, VA, USA, 2014, pp. 720–731.

[2] Y. Zhou and X. Jiang, "Dissecting Android malware: Characterization and evolution," in *Proc. IEEE Symp. Secur. Privacy*, New York, NY, USA, May 2012, pp. 95–109.

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

[4] F. Idrees, M. Rajarajan, M. Conti, T. M. Chen, and Y. Rahulamathavan, "Pindroid: A novel Android malware detection system using ensemble learning methods," *Comput. Secur.*, vol. 68, pp. 36–46, Jul. 2017.

[5] P. Vinod, A. Zemmari, and M. Conti, "A machine learning based approach to detect malicious Android apps using discriminant system calls," *Future Gener. Comput. Syst.*, vol. 94, pp. 333–350, May 2019.

[6] E. B. Karbab, M. Debbabi, A. Derhab, and D. Mouheb, "MalDozer: Automatic framework for Android malware detection using deep learning," *Digit. Invest.*, vol. 24, pp. S48–S59, Mar. 2018.

[7] F. Pendlebury, F. Pierazzi, R. Jordaney, J. Kinder, and L. Cavallaro, "TESSERACT: Eliminating experimental bias in malware classification across space and time," in *Proc. 28th USENIX Secur. Symp.* Berkeley, CA, USA: USENIX Association, 2019, pp. 729–746.

[8] K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon, "AndroZoo: Collecting millions of Android apps for the research community," in *Proc. 13th Int. Conf. Mining Softw. Repositories*, New York, NY, USA, May 2016, pp. 468–471.

[9] M. Lindorfer, M. Neugschwandtner, L. Weichselbaum, Y. Fratantonio, V. V. D. Veen, and C. Platzer, "ANDRUBIS—1,000,000 apps later: A view on current Android malware behaviors," in *Proc. 3rd Int. Workshop Building Anal. Datasets Gathering Exper. Returns Secur. (BADGERS)*, New York, NY, USA, Sep. 2014, pp. 3–17.

[10] N. Kiss, J.-F. Lalande, M. Leslous, and V. V. T. Tong, "Kharon dataset: Android malware under a microscope," in *Proc. Workshop, Learn. Authoritative Secur. Exp. Results*. Berkeley, CA, USA: USENIX Association, 2016, pp. 1–12.

[11] A. H. Lashkari, A. A. F. Kadir, L. Taheri, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark Android malware datasets and classification," in *Proc. 52nd Int. Carnahan Conf. Secur. Technol. (ICCST)*, New York, NY, USA, Oct. 2018, pp. 1–7.

[12] VirusShare. (2021). *VirusShare.com—Because Sharing is Caring*. [Online]. Available: https://virusshare.com/

[13] K. Aktas and S. Sen, "Updroid: Updated Android malware and its familial classification," in *Proc. Nordic Conf. Secure IT Syst.* Cham, Switzerland: Springer, 2018, pp. 352–368.

[14] Koodous Team. (2021). *Koodous Project*. [Online]. Available: https://koodous.com/about

[15] Hispasec Sistemas. (2021). *Virus Total*. [Online]. Available: https://www.virustotal.com

[16] S. Dong *et al.*, "Understanding Android obfuscation techniques: A large-scale investigation in the wild," in *Security and Privacy in Communication Networks*. Cham, Switzerland: Springer, 2018, pp. 172–192.

[17] H. Wang *et al.*, "Beyond Google play: A large-scale comparative study of Chinese Android app markets," in *Proc. Internet Meas. Conf.*, New York, NY, USA, Oct. 2018, pp. 293–307.

[18] M. Grace, Y. Zhou, Q. Zhang, S. Zou, and X. Jiang, "RiskRanker: Scalable and accurate zero-day Android malware detection," in *Proc. 10th Int. Conf. Mobile Syst., Appl., Services (MobiSys)*, New York, NY, USA, 2012, pp. 281–294.

[19] H. Gonzalez, A. A. Kadir, N. Stakhanova, A. J. Alzahrani, and A. A. Ghorbani, "Exploring reverse engineering symptoms in Android apps," in *Proc. 8th Eur. Workshop Syst. Secur.*, New York, NY, USA, Apr. 2015, pp. 1–7.

[20] P. Battista, F. Mercaldo, V. Nardone, A. Santone, and C. A. Visaggio, "Identification of Android malware families with model checking," in *Proc. 2nd Int. Conf. Inf. Syst. Secur. Privacy*, Rome, Italy, 2016, pp. 542–547.

[21] J. Crussell, C. Gibler, and H. Chen, "AnDarwin: Scalable detection of semantically similar Android applications," in *Proc. Comput. Secur. (ESORICS)*. Berlin, Germany: Springer, 2013, pp. 182–199.

[22] L. Li *et al.*, "IccTA: Detecting inter-component privacy leaks in Android apps," in *Proc. IEEE/ACM 37th IEEE Int. Conf. Softw. Eng.*, New York, NY, USA, May 2015, pp. 280–291.

[23] G. M. Weiss and F. Provost, "The effect of class distribution on classifier learning: An empirical study," Dept. Comput. Sci., Rutgers Univ., Fredericton, NM, Canada, Tech. Rep. ML-TR-44, 2001.

[24] S. Susan and A. Kumar, "The balancing trick: Optimized sampling of imbalanced datasets—A brief survey of the recent state of the art," *Eng. Rep.*, vol. 3, no. 4, 2020, Art. no. e12298.

[25] D. Pfaff, M.-T. Walter, and M. Backes, "Code clones considered harmful? Quantifying and exploiting the effects of code clones in static malware classifiers for Javascript," in *Proc. 14th Int. Conf. Malicious Unwanted Softw.*, 2019, pp. 111–118.

[26] I. Tomek, "Two modifications of CNN," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-6, pp. 769–772, 1976.

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

[28] F. Pan, W. Wang, A. K. H. Tung, and J. Yang, "Finding representative set from massive data," in *Proc. 5th IEEE Int. Conf. Data Mining (ICDM)*, New York, YN, USA, Jun. 2005, pp. 338–345.

[29] X. Fu and H. Cai, "On the deterioration of learning-based malware detectors for android," in *Proc. IEEE/ACM 41st Int. Conf. Softw. Eng., Companion (ICSE-Companion)*, May 2019, pp. 272–273.

[30] H. Cai and J. Jenkins, "Towards sustainable Android malware detection," in *Proc. 40th Int. Conf. Softw. Eng., Companion*, New York, NY, USA, May 2018, pp. 350–351.

[31] H. Cai, "Assessing and improving malware detection sustainability through app evolution studies," *ACM Trans. Softw. Eng. Methodol.*, vol. 29, no. 2, pp. 1–28, Apr. 2020.

[32] H. Cai, "Embracing mobile app evolution via continuous ecosystem mining and characterization," in *Proc. IEEE/ACM 7th Int. Conf. Mobile Softw. Eng. Syst.*, New York, NY, USA, Jul. 2020, pp. 31–35.

[33] X. Zhang *et al.*, "Enhancing state-of-the-art classifiers with API semantics to detect evolved Android malware," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, New York, NY, USA, Oct. 2020, pp. 757–770.

[34] K. Xu, Y. Li, R. Deng, K. Chen, and J. Xu, "DroidEvolver: Self-evolving Android malware detection system," in *Proc. IEEE Eur. Symp. Secur. Privacy (EuroSP)*, Jun. 2019, pp. 47–62.

[35] K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon, "Are your training datasets yet relevant?" in *Engineering Secure Software and Systems*. Cham, Switzerland: Springer, 2015, pp. 51–67.

[36] W. G. Cochran, *Sampling Techniques*, 3rd ed. New York, NY, USA: Wiley, 1977.

[37] R. D. De Veaux, P. F. Velleman, and D. E. Bock, *Stats Data and Models*. London, U.K.: Pearson, 2016.

[38] R. V. Hogg, E. A. Tanis, and D. L. Zimmerman, *Probability and Statistical Inference*. London, U.K.: Pearson, 2015.

[39] R. Bardenet and O.-A. Maillard, "Concentration inequalities for sampling without replacement," *Bernoulli*, vol. 21, no. 3, pp. 1361–1385, 2015.

[40] O. J. Dunn, "Multiple comparisons among means," *J. Amer. Statist. Assoc.*, vol. 56, no. 293, pp. 52–64, 1961.

[41] L. Apvrille and A. Apvrille, "Pre-filtering mobile malware with heuristic techniques," in *Proc. 2nd Int. Symp. Res. Grey-Hat Hacking (GreHack)*, 2013, pp. 44–59.

[42] K. Pearson, "X. On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling," *Philos. Mag.*, vol. 50, no. 320, pp. 157–175, 1900.

[43] A. Feizollah, N. B. Anuar, R. Salleh, G. Suarez-Tangil, and S. Furnell, "AndroDialysis: Analysis of Android intent effectiveness in malware detection," *Comput. Secur.*, vol. 65, pp. 121–134, Mar. 2017.

[44] K. Xu, Y. Li, and R. H. Deng, "ICCDetector: ICC-based malware detection on Android," *IEEE Trans. Inf. Forensics Security*, vol. 11, no. 6, pp. 1252–1264, Jun. 2016.

[45] T. C. Miranda, P.-F. Gimenez, and J.-F. Lalande. (2021). *Dada: Debiased Android DAtasets*. [Online]. Available: https://dx.doi.org/10.21227/0frv-zb46

[46] W. Li, X. Fu, and H. Cai, "AndroCT: Ten years of app call traces in Android," in *Proc. IEEE/ACM 18th Int. Conf. Mining Softw. Repositories (MSR)*, May 2021, pp. 570–574.

[47] M. Fan *et al.*, "Android malware familial classification and representative sample selection via frequent subgraph analysis," *IEEE Trans. Inf. Forensics Security*, vol. 13, no. 8, pp. 1890–1905, Aug. 2018.

[48] E. Fix and J. L. Hodges, Jr., "Discriminatory analysis. Nonparametric discrimination: Consistency properties," *Int. Stat. Rev.*, vol. 57, no. 3, pp. 238–247, 1989.

[49] W.-Y. Loh, "Classification and regression trees," *Wiley Interdiscipl. Rev., Data Mining Knowl. Discovery*, vol. 1, no. 1, pp. 14–23, 2011.

[50] T. K. Ho, "Random decision forests," in *Proc. 3rd Int. Conf. Document Anal. Recognit.*, vol. 1, Aug. 1995, pp. 278–282.

[51] D. J. Hand and K. Yu, "Idiot's Bayes—Not so stupid after all?" *International Stat. Rev.*, vol. 69, no. 3, pp. 385–398, 2001.

[52] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *J. Comput. Syst. Sci.*, vol. 55, no. 1, pp. 119–139, Aug. 1997.

[53] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, Oct. 2011.

[54] H. Cai and B. Ryder, "A longitudinal study of application structure and behaviors in android," *IEEE Trans. Softw. Eng.*, vol. 47, no. 12, pp. 2934–2955, Dec. 2021.

[55] H. Cai, X. Fu, and A. Hamou-Lhadj, "A study of run-time behavioral evolution of benign versus malicious apps in android," *Inf. Softw. Technol.*, vol. 122, Jun. 2020, Art. no. 106291.

[56] A. Saracino, D. Sgandurra, G. Dini, and F. Martinelli, "MADAM: Effective and efficient behavior-based Android malware detection and prevention," *IEEE Trans. Dependable Secure Comput.*, vol. 15, no. 1, pp. 83–97, Jan. 2018.

[57] Z. Yuan, Y. Lu, and Y. Xue, "Droiddetector: Android malware characterization and detection using deep learning," *Tsinghua Sci. Technol.*, vol. 21, no. 1, pp. 114–123, Feb. 2016.

[58] E. Mariconti, L. Onwuzurike, P. Andriotis, E. De Cristofaro, G. Ross, and G. Stringhini, "MaMaDroid: Detecting Android malware by building Markov chains of behavioral models," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, San Diego, CA, USA, 2017, pp. 297–311.

[59] S. Chen, M. Xue, Z. Tang, L. Xu, and H. Zhu, "StormDroid: A streaminglized machine learning-based system for detecting Android malware," in *Proc. 11th ACM Asia Conf. Comput. Commun. Secur.*, May 2016, pp. 377–388.

[60] Yuval Nativ YTISF. *TheZoo—A Live Malware Repository Web Site*. Accessed: Mar. 15, 2022. [Online]. Available: http://ytisf.github.io/theZoo/

[61] Silas Cutler. *MalShare*. Accessed: Mar. 15, 2022. [Online]. Available: https://malshare.com/about.php

[62] S. Bhandari, R. Panihar, S. Naval, V. Laxmi, A. Zemmari, and M. S. Gaur, "SWORD: Semantic aWare Android malwaRe detector," *J. Inf. Secur. Appl.*, vol. 42, pp. 46–56, Oct. 2018.

[63] M. Ahmadi, A. Sotgiu, and G. Giacinto, "IntelliAV: Toward the feasibility of building intelligent anti-malware on Android devices," in *Machine Learning and Knowledge Extraction*. Cham, Switzerland: Springer, 2017, pp. 137–154.

[64] I. Martín, J. A. Hernández, A. Mu noz, and A. Guzmán, "Android malware characterization using metadata and machine learning techniques," *Secur. Commun. Netw.*, vol. 2018, pp. 5749481:1–5749481:11, Jul. 2018.

[65] S. Hou, Y. Ye, Y. Song, and M. Abdulhayoglu, "HinDroid: An intelligent Android malware detection system based on structured heterogeneous information network," in *Proc. 23rd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, New York, NY, USA, Aug. 2017, pp. 1507–1515.

[66] J. Garcia, M. Hammad, and S. Malek, "Lightweight, obfuscation-resilient detection and family identification of Android malware," *ACM Trans. Softw. Eng. Methodol.*, vol. 26, no. 3, pp. 1–29, Jan. 2018.

[67] A. Demontis *et al.*, "Yes, machine learning can be more secure! A case study on Android malware detection," *IEEE Trans. Dependable Secure Comput.*, vol. 16, no. 4, pp. 711–724, Jul. 2019.

[68] N. McLaughlin *et al.*, "Deep Android malware detection," in *Proc. 7th ACM Conf. Data Appl. Secur. Privacy*, New York, NY, USA, 2017, pp. 301–308.

[69] M. K. Alzaylaee, S. Y. Yerima, and S. Sezer, "EMULATOR vs REAL PHONE: Android malware detection using machine learning," in *Proc. 3rd ACM Int. Workshop Secur. Privacy Anal.*, Scottsdale, AZ, USA, Mar. 2017, pp. 65–72.

[70] M. Melis, D. Maiorca, B. Biggio, G. Giacinto, and F. Roli, "Explaining black-box Android malware detection," in *Proc. 26th Eur. Signal Process. Conf. (EUSIPCO)*, Sep. 2018, pp. 524–528.

[71] A. Narayanan, M. Chandramohan, L. Chen, and Y. Liu, "A multi-view context-aware approach to Android malware detection and malicious code localization," *Empirical Softw. Eng.*, vol. 23, no. 3, pp. 1222–1274, 2018.

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

[73] G. Meng, Y. Xue, Z. Xu, Y. Liu, J. Zhang, and A. Narayanan, "Semantic modelling of Android malware for effective malware comprehension, detection, and classification," in *Proc. 25th Int. Symp. Softw. Test. Anal.*, New York, NY, USA, Jul. 2016, pp. 306–317.

[74] X. Xiao, S. Zhang, F. Mercaldo, G. Hu, and A. K. Sangaiah, "Android malware detection based on system call sequences and LSTM," *Multimedia Tools Appl.*, vol. 78, no. 4, pp. 3979–3999, 2019.

[75] H. Cai, N. Meng, B. G. Ryder, and D. Yao, "DroidCat: Effective Android malware detection and categorization via app-level profiling," *IEEE Trans. Inf. Forensics Security*, vol. 14, no. 6, pp. 1455–1470, Nov. 2019.

[76] M. S. Rana, C. Gudla, and A. H. Sung, "Android malware detection using stacked generalization," in *Proc. 27th Int. Conf. Softw. Eng. Data Eng.*, 2018, pp. 15–19.

[77] Z. Ma, H. Ge, Y. Liu, M. Zhao, and J. Ma, "A combination method for Android malware detection based on control flow graphs and machine learning algorithms," *IEEE Access*, vol. 7, pp. 21235–21245, 2019.

[78] N. Huang, M. Xu, N. Zheng, T. Qiao, and K.-K.-R. Choo, "Deep Android malware classification with API-based feature graph," in *Proc. 18th IEEE Int. Conf. Trust, Secur. Privacy Comput. Commun., 13th IEEE Int. Conf. Big Data Sci. Eng. (TrustCom/BigDataSE)*, Aug. 2019, pp. 296–303.

[79] X. Liu, X. Du, X. Zhang, Q. Zhu, H. Wang, and M. Guizani, "Adversarial samples on Android malware detection systems for IoT systems," *Sensors*, vol. 19, no. 4, p. 974, Feb. 2019.

[80] Y. Ye *et al.*, "Out-of-sample node representation learning for heterogeneous graph in real-time Android malware detection," in *Proc. 28th Int. Joint Conf. Artif. Intell.*, Aug. 2019, pp. 4150–4156.

[81] F. Martinelli, F. Mercaldo, and A. Saracino, "BRIDEMAID: An hybrid tool for accurate detection of Android malware," in *Proc. ACM Asia Conf. Comput. Commun. Secur.*, New York, NY, USA, Apr. 2017, pp. 899–901.