# Misleading Metrics: On Evaluating Machine Learning for Malware with Confidence

Roberto Jordaney, Zhi Wang, Davide Papini, Ilia Nouretdinov, and Lorenzo Cavallaro
Royal Holloway, University of London

***Abstract*—Malware pose a serious and challenging threat across the Internet and the need for automated learning-based approaches has become rapidly clear. Machine learning has long been acknowledged as a promising technique to identify and classify malware threats; such a powerful technique is unfortunately often seen as a black-box panacea, where little is understood and the results—especially with high accuracy—are taken without questioning their quality. For such reasons, results are often biased by the choice of empirical thresholds or dataset-specific artifacts, hindering the ability to set easy-to-understand error metrics and thus compare different approaches. This setting, calls for new metrics that look beyond quantitative measurements (e.g., precision and recall), and help in scientifically assessing the soundness of the underlying machine learning tasks. To this end, we propose *conformal evaluator*, a framework designed at evaluating the quality of a result in terms of statistical metrics such as credibility and confidence. Credibility tells you how much a sample is credited with one given prediction (e.g., a label), whereas confidence focuses on pointing out how much a given sample is distinguished from other predictions. Such evaluation metrics give useful insights, providing a quantifiable per-choice level of assurance and reliability. Core of conformal evaluator is a non-conformity measure, which, in essence, allows for measuring the difference between a sample and a set of samples. For this reason, our framework is general enough to be immediately applied by a large class of algorithms that rely on distances to identify and classify malware, allowing to better understand and compare machine learning results. To further support our claim, we present case studies where the outcome of three different algorithms are evaluated under conformal evaluator settings. We show how traditional metrics mislead about the performance of different algorithms. Instead, conformal evaluator's metrics enable to understand the reasons behind the performance of a given algorithm, and reveal shortcomings of apparently highly accurate methods.**

## I. Introduction

The analysis and prompt detection of malicious software represent one of the most pressing and important issues that plague the security of the Internet and its users, nowadays. With more than 500,000 unique malware samples per day reported in Q2 2015 [17], it is clear that manual analysis does not scale up and the shift is therefore on automatic and adaptive techniques able to identify unknown and previously-unseen threats. To this end, machine learning, with a particular emphasis on clustering and classification, has long been acknowledged as a promising technique to address such a fundamental need; botnet detection [8], [30], mobile malware [3], and malware detection and classification [15], [5], [19], [18], [22] are just a few explanatory examples.

Looking at the advances in the field, it would seem that the problem is almost solved. However, assessing the results of a given algorithm is problematic. With fewer exceptions, e.g., [20], [3], [31], the lack of publicly-available datasets hinders the ability to reproduce results. Furthermore the usage of traditional metrics (e.g., accuracy, precision, recall and confusion matrices) to assess the performance of a machine learning algorithm, might produce misleading results. As a matter of fact such metrics report statistics on correct and incorrect decisions, but do not capture their quality and are hence ill-suited to evaluate a given task.

Quite recently, Li et al. considered this problem in [16], empirically showing that traditional metrics with high accuracy do not necessarily imply that the underlying machine learning is good. They show how the dataset is often chosen to support the claim of the author. Their work focused primarily on methods specifically built on the available datasets and that suffered from data over-fitting issues. Conversely, in our work, we aim at tackling the problem under a broader scope, providing a way of assessing the quality of a given algorithm in a scientific and rigorous manner. We are not saying that traditional metrics do not provide important insights, however they are just performance indicators and cannot be used to assess the quality of a given algorithm.

Another factor that influences the outcome of a machine learning based algorithm is the process of feature selection. As a matter of fact the algorithm in itself, although designed appropriately, might not work if the selected features do not correctly capture and separate the desired qualities of the samples. One might say that any algorithm works as long as features are properly selected, and hence it is important to focus on that process only. However in our work we show that focusing on features is not enough (see § VI), there might be algorithms that perform very well even with bad separated features or algorithms that do not work even with perfect features.

To address the problem, we propose *conformal evaluator*, an evaluation framework that makes use of statistical metrics to provide a quality evaluation of the algorithm. It is built around conformal predictor [28], a machine learning algorithm tailored at classification tasks. It relies on statistics to select the best result and provide quality guarantees. In this paper, we rethink conformal predictor as a novel approach to the problem of evaluating malware clustering and classification techniques.

In particular, we make the following contributions:

- We extend the quality metrics of conformal predictor and propose two novel evaluation metrics to capture

the quality of an algorithm results: algorithm credibility (§ III-B) and algorithm confidence (§ III-C).
- We propose two novel analyses to produce qualitative and quantitative metrics to the correctness of a machine learning process, which can be exploited to evaluate its outputs (§ IV-A), and how these are influenced by the dataset composition (e.g. malware families) (§ IV-B). In practical settings, our technique can be plugged on top of any machine learning algorithm that relies on a similarity function to compute distances between a sample and a set of samples.
- We rely on our analyses to evaluate four different algorithms on a number of different datasets and in different scenarios. Our experiments clearly show whether we can trust the results of such algorithms or not, providing useful qualitative insights.
- We further show how methods (e.g., t-SNE [29] and PCA [13]) to assess the quality of selected features do not produce sound results, as shown in § VI, as we do with conformal evaluator. Particularly, with our experiments, we show examples where algorithms produce very high quality results even with very poor sample separation.

We believe that conformal evaluator can help enormously in building better machine learning based techniques, allowing to assess both the quality of the algorithm and features altogether. The need for metrics that takes into account the quality of a classification/clustering scheme is pressing and we provide a solution that can be applied to existing algorithms.

The rest of the paper is organized as follows: § II introduces the problem of clustering and classification of malware, § III describes conformal evaluator in details, § IV presents our novel conformal predictor-driven analyses, § V describes the experiments and discusses their findings, and finally § VII and § VIII discuss limitations, future works and conclusions.

## II. Machine Learning against Malware

Malware is a malicious piece of software that is engineered to steal precious information (e.g., personal data) or to perform evil actions (e.g., attack a system). As such, malware are designed to be difficult to detect, so that the threat is as most persistent as possible. As a matter of fact, new malware are deployed every day so that security appliances have to continuously adapt to detect them.

However stealthy, at some point malware has to perform the malicious activity for which it was designed, hence industry and research have been focusing on identifying and detecting such activities. A valid approach consists in finding common behaviors and group them together in order to find the underlying characteristics that malware belonging to the same family share. This is usually done using machine learning techniques that rely on *clustering* (e.g., [9],[8] and [30]) or *classification* (e.g., [22] and [3]).

However, assessing the validity of machine learning techniques is a problem that has not been completely solved. Particularly, Li et al. in [16] have started to reason about the problem of assessing the effective validity of traditional measures, e.g., false positive rate, precision, accuracy, recall and they have found out that such measures are strongly influenced by the underlying dataset. Their work suggests that there is the need for a more scientifically robust approach for evaluating malware identification methods.

Similar concerns have also been raised in other works e.g., [4], [23] and [25]. Specifically [25] says:

> *“The community does not benefit any further from yet another study measuring the performance of some previously untried combination of a machine learning scheme with a particular feature set. The nature of our domain is such that one can always find a variation that works slightly better than anything else a particular setting. Unfortunately, while obvious for those working in the domain for some time, this fact can be easily lost on newcomers. Intuitively, when achieving better results on the same data than anybody else, one would expect this to be a definite contribution to the progress of the field. The point we wish to convey however is that we are working in an area where insight matters much more than just numerical results”*.

The authors here were addressing the intrusion detection community, however this statement is still valid in any setting where machine learning is applied to solve security related problems.

Another relevant work is done by Allix et al. [14], where the authors show how incorrectly handling a dataset could potentially lead to biased results. Particularly they historically consider malware and show that most of the time future knowledge, i.e. malware discovered later, is used to classify old malware and not vice-versa, leading to non realistic scenarios. This suggests that there are a lot of common practices within the machine learning security community that researchers usually adhere to, but which are not completely understood.

Moreover, García et al. [7] highlight another issue concerning the development of new methods. In their survey, the authors review fourteen network-based botnet detection methods and notice that only one of them makes an actual comparison with previous works. This is due mainly to practical reasons such as missing or incomplete public dataset, and algorithm unavailability for comparison.

As anticipated in the introduction, to evaluate their methods researchers usually relies on measures that, given a labeled dataset, analyze the success rate of classification or detection of malware. In our work we argue that traditional error metrics, e.g., confusion matrix, accuracy, precision, recall and ROC curve, suffer from a common flaw (as shown in our experiments). Specifically they do not investigate the quality of the single decision, i.e. they don’t take into account how much good or bad decision is, compared to alternative ones. Under these premises, traditional metrics potentially base on weak decisions, i.e., correct choices that are close to wrong ones and the opposite. In these circumstances, a small variation in the data can dramatically change the results of the overall algorithm.

## III. Conformal Evaluator

Conformal evaluator (CE) is our evaluation framework built on top of conformal predictor [28]. Conformal predictor is a machine learning algorithm, usually applied to classification problems, that gives a prediction (i.e., a label) with precise levels of trust on the prediction itself. Specifically, given a sequence of objects $z_1, z_2 \dots z_k \in C$, and a new object $z^*$, conformal predictor enables us to decide whether $z^* \in C$ or $z^* \notin C$.

In our framework, we dissect this algorithm to provide two statistical metrics that measure the quality of the results: *algorithm confidence* and *algorithm credibility*. Algorithm credibility gives valuable insights by telling how much the new object (e.g., a new malware sample) is credited with a given set (e.g., a malware family), that reflects the quality of the choice taken by the algorithm. If we define, in a classification setting, $C$ as a class of objects with $k$ elements and $\mathbb{D}$ as the set of all the classes of objects (i.e. the whole dataset), by iterating the process through every class in $\mathbb{D}$, the algorithm confidence measures how distinguished $z^*$ is with respect to the other classes. Algorithm credibility and algorithm confidence are further explained in § III-B and § III-C.

Core of conformal evaluator, is a non-conformity measure, a real-valued function $A(C, z)$, which tells how different an object $z$ is from a set of objects $C$. Thanks to the real-valued range of non-conformity measure, conformal evaluator can be immediately used with well known machine learning methods such as support-vector machines, neural networks, decision trees and Bayesian prediction (see [24]) and in general with any method that makes use of real-valued numbers (i.e., a similarity function) to distinguish objects. Such flexibility enables our framework to assess a wide range of algorithms.

Once a non-conformity measure is selected, conformal evaluator computes a p-value $p_{z^*}$ which, in essence for a new object $z^*$, represents the percentage of objects in $\{x \in C, \forall C \in \mathbb{D}\}$ (i.e. the whole dataset) that are equally or more estranged to $C$ as $z^*$. The algorithm is shown in Listing 1.

```
**Data**: Dataset $D = \{z_1, ..., z_n\}$, Sequence of objects
        $C \subset D$, non-conformity measure $A$, new object $z^*$
**Result**: p-value $p_{z^*}$

Set provisionally $C = C \cup \{z^*\}$
**for** $i \leftarrow 1$ ***to*** $n$ **do**
    $\alpha_i \leftarrow \begin{cases} A(C \setminus z_i, z_i) & \text{if } z_i \in C \\ A(C, z_i) & \text{if } z_i \notin C \end{cases}$
**end**

$p_{z^*} = \frac{|\{j : \alpha_j \geq \alpha_{z^*}\}|}{n}$
```

**Listing 1:** P-value calculation used in Conformal Evaluator.

P-values are directly involved in the algorithm credibility and confidence.

### *A. Non-conformity Measure*

Many machine-learning algorithms for classification are based on a *scoring algorithm* that given a training set of examples $C$ and a test object $z^*$ will output a *prediction score* $A(C, z^*)$. The *non-conformity measure* is elicited directly from the scoring function of the algorithm and is one of the basic blocks of conformal evaluator.

It is used to measure the difference between a group of objects belonging to the same class (e.g., malware belonging to the same family) and a new object (i.e., a sample). The higher the measure, the more different is the object with respect to the group of objects. Hence, if we take for example two objects, $z_1$ and $z_2$, and a group of objects $C$, $z_1$ is more dissimilar than $z_2$ to $C$ if $A(C, z_1) > A(C, z_2)$. The real-valued nature of the non-conformity measure allows for negative values. Whenever we find a measure that increases as the new object $z$ is similar to $C$ (i.e., a similarity function), we can easily convert this to a non-conformity measure by changing its sign. It should be obvious at this point that conformal evaluator can be applied on top of almost any classification (or clustering) algorithm that uses a score to classify a new object (or to assign it to a cluster).

To give an example, Rieck et al. in [22] use Support Vector Machine (SVM) as the core of their classification approach. Whenever a new sample comes in, a score is computed and the label with the highest score is assigned to the sample. In this case the score can be converted to a non-conformity measure, allowing for evaluation of choices through conformal evaluator.

### *B. Algorithm Credibility*

The first evaluation metric we explore is *algorithm credibility*. It is defined as the p-value corresponding to the label chosen by the algorithm under analysis.

It is calculated iterating Algorithm 1 through every class $c_1, c_2 \dots c_m \in \mathbb{D}$, the output of the iteration is a series of p-values $p_{z^*}^{c_1}, p_{z^*}^{c_2} \dots p_{z^*}^{c_m}$ which tells us how likely it is that $z^* \in c_i, \forall i \in [1 \dots m]$.

From Algorithm 1, the p-value is defined as:

$$p_{z^*}^{c} = \frac{|\{j : \alpha_j \geq \alpha_{z^*}\}|}{n} \quad (1)$$

where $c$ is the class the p-value is computed for.

As stated earlier, the p-value measures the fraction of objects within $\mathbb{D}$, that are at least as different from a class $C$ as the new object $z^*$. Therefore a high credibility value means that $z^*$ is very similar to the objects in $C$. This observation alone can be considered already a good achievement, however high credibility alone tells us very little with respect to the quality of the choice, as there may be multiple p-values that are close to the maximum.

Considering a low credibility value instead, this usually shows that $z^*$ is very different from $C$, yet this could also reveal that the object is poorly identified. These two observations show that credibility alone is somewhat insufficient, hence we introduce another measure: algorithm confidence.

### C. Algorithm Confidence

Algorithm confidence is the second evaluation metric we explore. For a given choice (e.g., assigning $z$ to a class $c_i$), confidence tells how certain or how committed the evaluated algorithm is to the choice. Looking at it from a more formal perspective, it measures how much the new object $z^* \in c_i$ is distinguishable from other classes $c_j$ with $j \neq i$.

We define the algorithm confidence to be one minus the maximum p-value among all p-values except the *p-value* chosen by the algorithm (i.e., algorithm credibility):

$$AConf = 1 - max(\mathbb{P} \setminus ACred), \mathbb{P} = \{p^c : c \in \mathbb{D}\} \quad (2)$$

where $ACred$ and $AConf$ are respectively algorithm credibility and algorithm confidence and $\mathbb{P}$ is the set of p-values associated to the possible choices (i.e. classes) for the new object $z^*$.

Given an object and a set of possible choices, the highest possible value of confidence is the one associated with the highest p-value. In a conformal predictor setting this is considered to be the best choice, thus resulting also in the highest confidence possible, however in our setting, where we rethink conformal predictor for evaluation purposes, it may happen that the choice made by the algorithm is not the best one, reflecting also on the confidence being not optimal. We will see through the experiments section that this sometimes brings to valuable insights, especially when the methods under assessment take choices with low values of confidence and credibility.

A low value for algorithm confidence indicates that the given object is similar to other classes as well. Depending on the algorithm credibility value, this indication may imply that the decision algorithm is not able to uniquely identify the classes or, that the new object has features common to two or more classes. On the other hand a high confidence in general is a sign that the identification method is good in distinguishing a class from the others.

As a final remark, before explaining how to use conformal evaluator for evaluating malware clustering and classification methods, we would like to highlight that confidence and credibility are not biased from the number of classes within a dataset as common measures such as precision and recall are, as Li et al. have shown in [16]. This means that conformal evaluator findings are more robust to dataset changes than other evaluation methods.

## IV. Framework Description

In the previous section we have introduced conformal evaluator along with its measures, algorithm confidence and algorithm credibility. In order to fully leverage their benefits, we have built a framework around them that, given a dataset and an algorithm, evaluates the quality of the algorithm by producing two assessments.

The framework is shown in Figure 1. From the similarity-based classification/clustering algorithm we elicit a non-conformity measure which is then used by conformal evaluator. Whether intended at classification or clustering, such algorithms sometimes use methods as an intermediate step to score the similarity to previous trained malware profiles. In these instances a non-conformity measure might be elicited from the intermediate step.

Our framework introduces two novel analyses, which we briefly outline here below and explain in detail afterwards.

**Decision assessment**, which helps in understanding how robust are the choices taken by the evaluated algorithm. It directly relates to the results given by the similarity-based classification/clustering algorithm. **Alpha assessment**, which gives indication of how good the non-conformity measure is with respect to the dataset. It provides more profound and trustworthy insights on how much the algorithm is good (or bad) with respect to the data at hand. It works by assessing how well the non-conformity measure, hence the measuring method of the similarity-based classification/clustering algorithm itself, works with the dataset. We call this *alpha assessment*, as we often refer to the non-conformity score for object $z_j$ as $\alpha_j$.

### A. Decision Assessment

The goal of this analysis is to assess the algorithm decisions in qualitative manner. To do so, for each new object $z^*$ (e.g., a malware) conformal evaluator takes the decision $c_i \in \mathbb{D}$ made by the algorithm (i.e., the assigned label), and computes its algorithm credibility and algorithm confidence.

At this point we can evaluate, for each choice, what is the algorithm credibility and the algorithm confidence behind any right and wrong choice. Hence, four possible scenarios unfold:

- High algorithm confidence, high algorithm credibility: the best situation, the algorithm is able to correctly identify a sample towards one class and one class only.
- High algorithm confidence, low algorithm credibility: the algorithm is not able to correctly associate the sample to any of the class present in the dataset
- Low algorithm confidence, low algorithm credibility: the algorithm gives a label to the sample but it seems to be more similar to another label
- Low algorithm confidence, high algorithm credibility: according to the algorithm, it seems that the sample is similar to two or more classes.

The values of the two measures are then grouped into two separate sets, correct or wrong, which represents values for correctly and wrongly classified objects respectively. Values are then averaged and their standard deviation is also computed, this is done for every class $c \in \mathbb{D}$, so that we may highlight whether the algorithm works consistently against all classes or if there are difficult classes that the algorithm has trouble dealing with.

Now, it is interesting to compare the results obtained for correct and wrong choices. For correct choice it would be desirable to have high credibility and high confidence. Conversely, for wrong choice, it would be desirable to have low credibility and high confidence. The divergence from these situations will help us to understand whether the algorithm takes strong decisions, meaning that there is a strong statistical

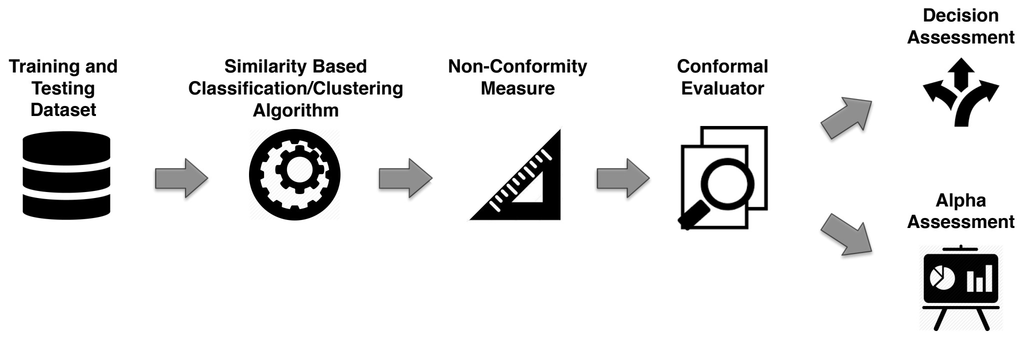
Training and Testing Dataset
Similarity Based Classification/Clustering Algorithm
Non-Conformity Measure
Conformal Evaluator
Decision Assessment
Alpha Assessment

Fig. 1: Evaluating malware classification/clustering with *conformal evaluator*. Decision assessment is derived from the algorithm decision alone and it is used to evaluate the quality of correct and incorrect algorithm chooses separately. Alpha assessment is used to assess how good is the similarity function with respect to the underlying dataset.

evidence to confirm its decisions, or, in contrast, if the decisions taken are easily modified with a minimal modification to the underlying data.

Looking at the outcome of decision assessment, it is already possible to understand whether the algorithm works well or not. Otherwise, it is possible to have an indication where to look for possible errors or improvements, i.e., which classes are troublesome, and whether further analysis is needed, e.g. by resorting to the alpha assessment.

### B. Alpha Assessment

Additionally to the decision assessment, which to evaluates the output of a similarity-based classification/clustering algorithm , another important step in understanding how the algorithm works and possibly its subtleties, consists in assessing how its inner similarity-based mechanism works with respect to the available datasets. As a matter of fact, due mainly to practical reasons, malware similarity-based algorithms are developed around a specific dataset, insomuch as it is often a question whether the newly developed algorithm is overfitting the dataset. Unfortunately the only way to answer this question is to try the algorithm against as many datasets as possible. In our research we found out that conformal evaluator can help with respect to this aspect, when no more than one dataset is accessible.

The *alpha assessment* is an analysis that takes into account how appropriate is the similarity-based algorithm with respect to a dataset. If the results are too good to be true (i.e., perfect), then probably the method is most likely overfitting the dataset, at the same time if the results are poor, then the method is already bad with the dataset at hand, so it is very likely that with new datasets it will not work as well as with the base one.

Furthermore the assessment enables to get insights on classes or groups of them (e.g., malware families), highlighting how the similarity based method works against them, and helping the researcher to gather new intelligence regarding the peculiarities of each class, hence to possibly narrow down the places to look into in order to improve the algorithm.

As a first step, for each object $z_j \in D$, where $c_j$ is $z_j$ true class, we compute its p-values against every possible decision in the dataset.

We then build a figure in a boxplot fashion [10], containing the p-values for each decision (e.g., malware families). By aligning together these boxplots and grouping them by class/cluster, we can see how much an element of class/cluster $j$ resembles that of another one according to the non-conformity measure, which is derived from the evaluated algorithm.

Apart from the obvious insights that we can gather from this figure, it also enable for identifying the more problematic cases (e.g., malware families) within subgroups of two or more, allowing for reasoning about the similarity-based algorithm itself.

In the next section we are going to present three case studies where we evaluate the performance of three algorithms within the conformal evaluator framework.

## V. Experiments Set Up and Analysis

To demonstrate conformal evaluator, in this section we evaluate three malware classification algorithms, leveraging the assessments defined in § IV:

- **Algorithm 1**: *Drebin* [3], a detection algorithm for malicious android applications.
- **Algorithm 2**: *BotFinder* [27], an algorithm for botnet classification and detection.
- **Algorithm 3**: algorithm used in the Kaggle's Microsoft Malware Classification Challenge [11] achieving position 49th over 377.

The first one, Drebin, detects malicious android applications based on static analysis of the application's APK (android package file). The method consists in extracting various information from the application manifest and the decompiled code. This information consists of the requested permissions, the requested hardware components, the application intents, the type of application components, various types of API calls, the effective used permissions (from the decompiled code) and the network addresses. These features are then used to build

| Drebin Dataset | | | |
|---|---|---|---|
| Original Dataset | | Perturbed Dataset | |
| Type | Samples | Type | Samples |
| Malicious app | 5,560 | Malicious app | 2,702 |
| Benign app | 123,453 | Benign app | 60,000 |

TABLE I: Dataset composition used by [3]

| Microsoft Malware Classification Challenge Dataset | | | |
|---|---|---|---|
| Malware | Samples | Malware | Samples |
| Ramnit | 1541 | Tracur | 751 |
| Lollipop | 2478 | Obfuscator.ACY | 1228 |
| Kelihos˙ver3 | 2942 | Gatak | 1013 |
| Vundo | 475 | Kelihos˙ver1 | 398 |

TABLE II: Number of samples for each family used in the Microsoft challenge

a model to decide whether new applications are malicious or not. The detection model is built using a well know machine learning algorithm, Support Vector Machine (SVM). In our conformal evaluator we use the distance to the hyperplane identified by SVM as non-conformity measure, to assess the results of the detection process. The dataset used in [3] is a public dataset composed of 123,435 benign applications and 5,560 malicious applications (see Table I).

We chose this algorithm for several reasons: the large dataset which is publicly available, Drebin reported performance is very good and the method is described well enough in the paper to give us the possibility to replicate it. For more details, please refer to [3].

With respect to *BotFinder* [27], the algorithm processes network traces to build family-based malware behavior profiles. These are later used for classification of new samples. We chose BotFinder as second use case, as it is a fairly recent work and it has been tested in the field with interesting results, moreover extracting a non-conformity measure was relatively straightforward and, finally, because the authors were very kind to provide us with the same dataset as the one they used for their own experiments (we refer to this dataset as Dataset-B on Table III). Having the same dataset is indeed crucial in order to have a meaningful comparison. As for BotFinder algorithm itself, we had to re-implement it from scratch, but [27] was detailed enough to let us build it, achieving similar performances.

The algorithm leverages five features, which are extracted from network flows in the captured botnet communications. These are the average value of the time intervals between two subsequent flows, the average duration of connections, average number of source bytes and destination bytes per flow and, finally, the Fast Fourier Transform to highlight periodic communications. These features are then combined together to obtain, once a new sample comes in, a score associated to each family in the dataset. The score, referred to as $\gamma_M$ in [27], is the product of the quality of the matched cluster of a malware family and the quality of the new sample. The quality is given by a quality rating function based on the mean and standard deviation of features.

The malware sample is labeled as belonging to the family with the highest score. This score naturally serves as non-conformity measure by inverting its sign (see § III-A). Before labeling the sample, BotFinder implements a filtering step that relies on an empirical threshold defined through iterative experiments. In our assessment we decided to omit this step as we are more interested in evaluating the scoring function itself.

We are going to apply the selected algorithms to the dataset used in [27] and a perturbed dataset. BotFinder's dataset is composed of 5 malware families, each one having a different number of network traces (see Table III). In addition, we introduce 5 new malware families (referred as Dataset-C in Table III) with wide range of numbers of network traces from 5 to 555, and combine them with BotFinder's dataset to generate the perturbed dataset totaling 10 families (see Table III).

The third algorithm we are going to evaluate comes from the Microsoft Classification Challenge [11] on the Kaggle platform [1]. The website is a well known platform hosting a wide range of data science related competitions, from image processing to medical related topics to security challenges. The Microsoft Malware Classification Challenge involves the classification of 9 malware families by leveraging statistical analysis of the disassembled binaries. In this study we decided to include only 8 families in the evaluation, as the excluded family has noticeably less samples than the others (10 to 100 times less).

The algorithm is described in [2]. The authors use the eXtreme Gradient Boosting (XGBoost) as their machine learning classification algorithm [26]. It's based on gradient boosting [21] and, like other boosting techniques, it combines different weak prediction models to create a stronger one. Particularly in their work, the authors use XGBoost with decision trees.

As non-conformity measure, we select the probability of one sample belonging to one class, with its sign inverted (since probabilities are conformity scores).

All the datasets, Drebin's and BotFinder's and Kaggle's challenge, consist of labeled malware samples whose ground

[1]https://www.kaggle.com

| Dataset-B | | Dataset-C | |
|---|---|---|---|
| Original Dataset | | Perturbed Dataset | |
| Malware | Samples | Malware | Samples |
| Bifrose | 51 | Gammima | 34 |
| Sasfis | 55 | Hupigon | 14 |
| Blackenergy | 62 | Swizzor | 117 |
| Banbra | 98 | Tibs | 555 |
| Pushdo | 46 | Windefender | 5 |

TABLE III: Number of samples for each family. Datasets used by [27]

| Sample | Assigned label Malicious | Benign | Recall |
|---|---|---|---|
| Malicious | 5132 | 428 | 0.92 |
| Benign | 297 | 123156 | 0.99 |
| **Precision** | 0.95 | 0.99 | |

TABLE IV: Confusion matrix for [3] with original dataset

| Sample | Assigned label Malicious | Benign | Recall |
|---|---|---|---|
| Malicious | 2678 | 24 | 0.99 |
| Benign | 1 | 59999 | 0.99 |
| **Precision** | 0.99 | 0.99 | |

TABLE V: Confusion matrix for [3] with perturbed dataset

truth has been verified.

We would like to remark that the evaluations presented here are just to demonstrate the validity of conformal evaluator and not to criticize the cited works in any way.

## *A. Algorithm 1: Drebin*

In this section we evaluate Drebin, described in [3]. The algorithm achieves very good performance as shown on the confusion matrix on Table IV.

Figure 2 shows the decision assessment (described in § IV-A) for [3]. Looking at correct choices, we can see that the average algorithm credibility is around 0.5 and the average algorithm confidence is over 0.9. This is considered a very good result because when the average confidence for the correct choices is high, it means that the samples are usually very different from the wrong label, so we can see that the algorithm takes the right decision with high statistical evidence of correctness. A value around 0.5 for average algorithm credibility is to be expected if most of the samples are correctly labeled (due to mathematical properties of conformal evaluator). For incorrect results, we can see the average algorithm credibility is less than 0.2 and the average algorithm confidence is very high, greater than 0.9. This again, is a good result because even when the algorithm chooses a wrong label for one sample, it's poorly associated with that label meaning that the algorithm has poor statistical evidence for his choice, hence it is not completely able to tell apart the right label from the wrong one, meaning that its error margin is very small (see § IV-A).

A good separation between correctly identified and incorrectly identified samples, shows that the decision made by the algorithm is most of the time far from the decision border, where by "decision border" we mean the ideal (or real) border where decisions switch from "A" to "B".

Looking thoroughly at [3], the reasons for this good result are threefold: the authors have a rather large dataset as ground truth, binary classification problem is usually simpler than multiclass classification and the method has strong scientific foundations and it is well designed.

As you have gathered, Figure 2 gives us more information than pure performance metrics. It is telling us that, based on the quality of the results obtained, we are confident the method is not strongly dependent on hidden choices or dataset specific tweaks. Later on, we are going to perturb the dataset to empirically confirm this statement.

Figure 3 reports the alpha assessment for Drebin. We plot this assessment in a boxplot fashion to show details of the p-value distribution. On the left side, we can see that some parts of the p-value distribution overlap, implying that some malicious samples are not perfectly separated. From the confusion matrix IV we can see that few malicious samples are misclassified. From the pure knowledge of the confusion matrix it is not possible to appreciate the quality of the decisions as it is by looking at the plot of the alpha assessment.

Moreover, based on results of the alpha assessment for benign samples, we do not observe any overfitting issues. Particularly, for the benign samples, the p-values for the two classes are quite different and the p-values to the malicious class are condensed in the lower part of the scale. In this case, the algorithm decision is strong and difficult to perturb even by changing the underling dataset.

*Results with perturbed dataset:* To give credit to the results obtained with the decision assessment, we rerun conformal evaluator with a perturbed the dataset to check results consistency against potential customized threshold or previous dataset specific tweaks. The perturbed dataset is described on Table I. Half of the malicious applications come from [31], while the other half were kindly provided by McAfee. The benign applications are a subset of the original Drebin's dataset.

In Figure 4, we can see the results of the decision assessment for the perturbed Drebin's dataset. The assessment shows a similar behavior as the original approach, meaning that

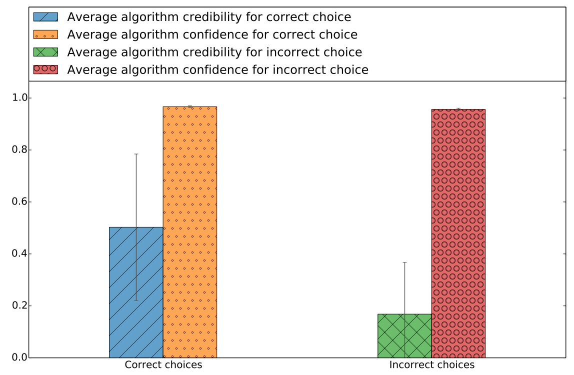

Fig. 2: Decision assessment for [3] with the original dataset. Very good results: correct classifications have a high confidence, while incorrect classifications have low credibility and high confidence. An algorithm credibility value of 0.5 for correct choice is to be expected if most of the samples are correctly labeled.

the results do not change when the underlying dataset does. This fact experiment empirically confirms that the decisions made by the algorithm are consistent across different datasets and hence not influenced by dataset customization and, more importantly, that the algorithm does not over-fit data. Even without an additional dataset, the information provided by the decision assessment alone (Figure 2) raises the reliability of the drawn conclusions (i.e., Drebin is well designed algorithm) to an higher level of confidence. The alpha assessment for the perturbed dataset 5 shows that the benign samples are again well distinguished from the malicious ones, with similar considerations as before.

### *B. Algorithm 2: BotFinder*

In this section we evaluate BotFinder, described in [27]. On Table VI, we can see how BotFinder performs according to traditional error metrics such as recall and precision. Looking at these performance metrics we can see that for Banbra and Bifrose, the algorithm works quite well, while we cannot say the same for Blackenergy and Sasfis, which have the lowest recall values.

The decision assessment (Figure 6) helps in understanding whether the algorithm would work with similar performances with other datasets or not.

The intuition is that if the average algorithm confidence and credibility from correct and wrong choice are well separated (i.e., different enough, as with Drebin, § V-A) then the algorithm performance is most likely to be consistent. Without a good separation, in a situation where the algorithm has a good performance but correct and incorrect classifications are not well separated, results will most likely change if another dataset is used.

The reason behind this is that traditional metrics merely measure the outcome of a given algorithm without taking into consideration the quality of said outcome. They do not take into account how good the classification is. Even with high precision and recall, we could have "lucky" decisions, meaning that the decisions taken are very close to the wrong ones even though still right. With a slight variation in the ground truth, the algorithm decision might be biased towards a wrong category. This means the method is not strong against variations.

Looking at Bifrose family (see Figure 6), the average algorithm credibility and confidence for correct decision indicates that when the algorithm chooses the correct label, the sample is very similar to the family (high algorithm credibility) but it is also similar to other families (low algorithm confidence). This is indeed not a good sign as a slight change in the data

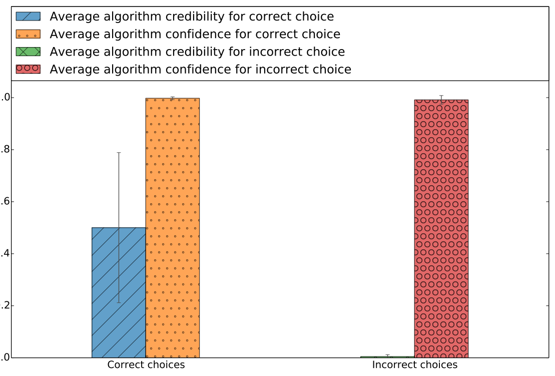

Fig. 4: Decision assessment for [3] with a perturbed dataset. We have again very good results: correct classifications have a high confidence, while incorrect classifications have low credibility and high confidence.

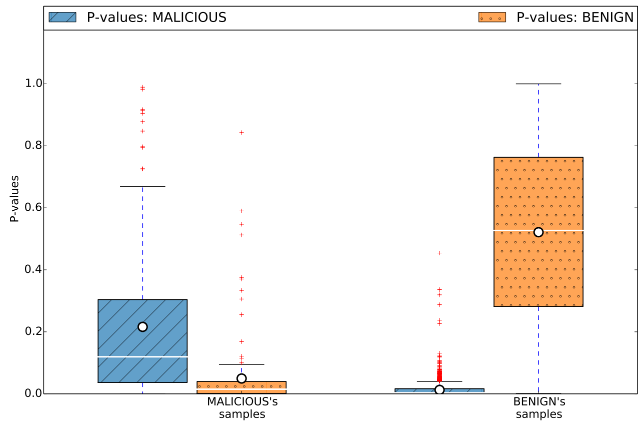

Fig. 3: Alpha assessment for [3] with the original dataset. Very good results achieved by the algorithm: the benign samples are well separated from the malicious ones especially for the benign samples. White dots represent the average values, white lines represent the median values, red crosses represent outliers.

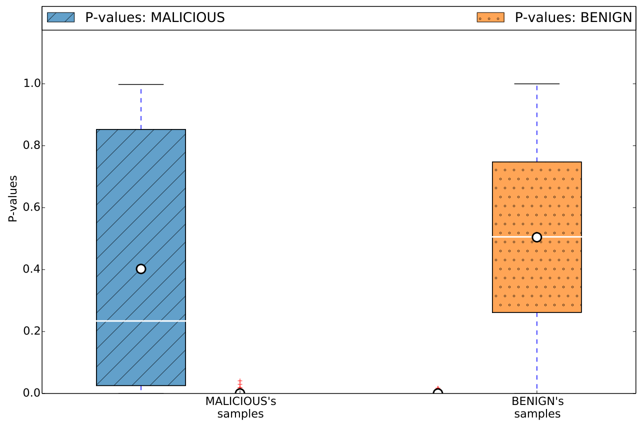

Fig. 5: Alpha assessment for [3] with the perturbed dataset. Very good results achieved by the algorithm: the p-values for benign samples are well separated. The p-values for malicious are once again not really well separated but the p-values for the benign class are very concentrated in the low part of the scale. White dots represent the average values, white lines represent the median values, red crosses represent outliers.

would most likely change the results of the algorithm as well.

For the same family, the average credibility and confidence for incorrect decision, shows that algorithm is quite sure about this “wrong” decision (high credibility and very high confidence). For an incorrect decision, these two facts indicate that we are in a situation where families overlap, hence the line distinguishing them is very thin. To improve the algorithm we therefore need to analyze those overlapping families to understand what to change in order to mark a more sharp border (if this is even possible). This analysis can be done with the alpha assessment which is discussed later on.

With respect to Banbra family, the decision assessment shows that the samples are very similar to their own family (high credibility) but they are also similar to others families (low confidence). Even in this case, changing the underlying ground truth samples, will lead to high variation in the results. However the good recall achieved by the algorithm, is due to the fact that for incorrect results the samples have a really low credibility meaning that it is very unlikely for the algorithm to mistake a Banbra’s sample for another family. It should be clear how recall is related to the average algorithm credibility and confidence for incorrect decision. The more their values deviates from good ones, like the one observed in [3], the more likely we will observe a poor recall.

With the sole knowledge of Table VI, it is difficult to draw conclusions or to reason about where to look to improve the classification. This is because traditional evaluation metrics only reports performances and do not try to explain what is happening under the hood.

Looking at the Alpha assessment, Figure 7, we can understand why some families have good performance while others do not, and if the algorithm is overfitting the data. For example, from the average p-values for Banbra, we can see that even if there is no misclassification with the Bifrose family (from the confusion matrix), the p-values of Banbra’s samples for the Bifrose hypothesis (Figure 7, Banbra’s samples, first column) are high and close to the p-values of Banbra. This fact indicates that it is unlikely that by perturbing the dataset, the results will be comparable.

| Sample | Assigned label | | | | | **Recall** |
|---|---|---|---|---|---|---|
| | Bifrose | Sasfis | Blackenergy | Banbra | Pushdo | |
| Bifrose | 41 | 6 | 0 | 4 | 0 | 0.80 |
| Sasfis | 1 | 18 | 32 | 1 | 3 | 0.33 |
| Blackenergy | 1 | 21 | 30 | 0 | 10 | 0.48 |
| Banbra | 0 | 3 | 8 | 87 | 0 | 0.89 |
| Pushdo | 2 | 1 | 13 | 0 | 30 | 0.65 |
| **Precision** | 0.91 | 0.37 | 0.36 | 0.95 | 0.7 | |

TABLE VI: Confusion matrix for [27] with Dataset-B

Looking at the confusion matrix, the situation of Pushdo’s samples is similar, i.e., the misclassification of this family to Banbra is null. If we take a look at the alpha assessment, we can see that Pushdo’s samples p-values with respect to the Banbra family are different from the one of Pushdo (implying that it is very difficult to mistake Pushdo for Banbra). With the last two examples, we started from the same situation in the confusion matrix (i.e., same no misclassification for a particular family) and we ended up having very different quality observations as we can see from the alpha assessment. This shows how traditional metrics are ill-suited for understanding the quality of a given machine learning task and may therefore be misleading in deploying it in real-world settings.

Regarding recall we can see that when one or more families starts to interfere with one another (look at the samples by family, e.g. first 5 columns and second five columns) quite heavily, the recall of this family drops. From Figure 7 we can see that Sasfis, Bifrose and Pushdo, are subject to heavy interference, making it difficult to identify them. Singling out interfering families can help to focus the attention into the

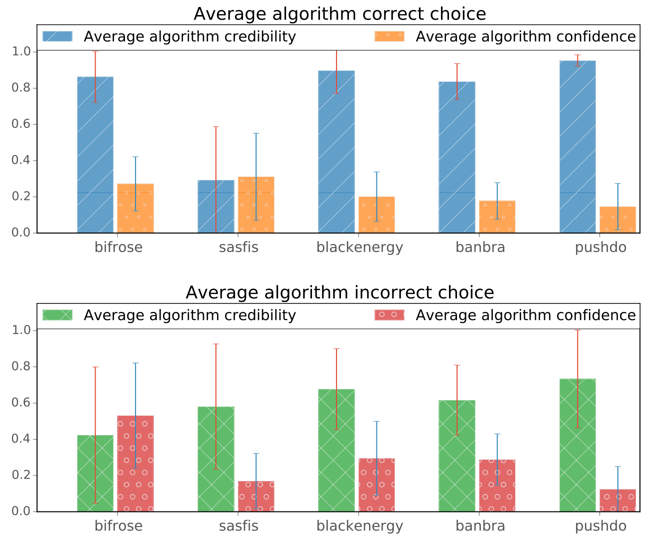

Fig. 6: Decision assessment for [27] with dataset-B. All the families show a poor separation between correct and incorrect results. This is symptom of weaknesses against dataset modification.

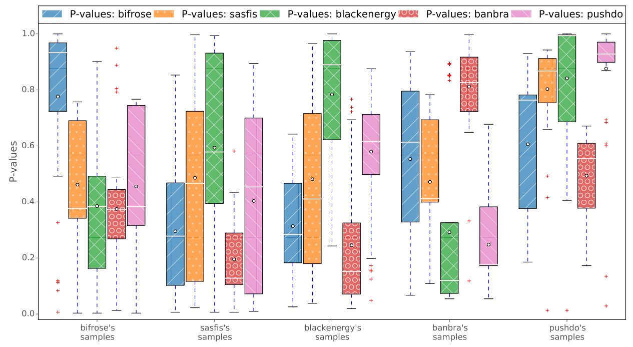

Fig. 7: Alpha assessment for [27] with dataset-B representing interfering families: white dots represent the average values, white lines represent the median values, red crosses represent outliers.

| | Assigned label | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | Bifrose | Sasfis | Blackenergy | Banbra | Pushdo | Gammima | Hupigon | Swizzor | Tibs | Windefender | **Recall** |
| Bifrose | 26 | 1 | 0 | 0 | 0 | 2 | 2 | 3 | 17 | 0 | 0.51 |
| Sasfis | 1 | 11 | 27 | 0 | 1 | 2 | 1 | 5 | 6 | 1 | 0.20 |
| Blackenergy | 0 | 9 | 27 | 0 | 7 | 3 | 0 | 6 | 10 | 0 | 0.44 |
| Banbra | 0 | 0 | 2 | 30 | 0 | 15 | 39 | 3 | 1 | 8 | 0.31 |
| Pushdo | 2 | 0 | 3 | 0 | 29 | 3 | 0 | 0 | 9 | 0 | 0.63 |
| Gammima | 1 | 3 | 5 | 2 | 0 | 9 | 0 | 8 | 6 | 0 | 0.26 |
| Hupigon | 1 | 0 | 0 | 1 | 0 | 0 | 8 | 1 | 0 | 3 | 0.57 |
| Swizzor | 1 | 1 | 36 | 0 | 0 | 0 | 37 | 35 | 0 | 7 | 0.30 |
| Tibs | 22 | 66 | 109 | 0 | 38 | 59 | 4 | 34 | 218 | 5 | 0.39 |
| Windefender | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 2 | 0 | 0 | 0 |
| **Precision** | 0.48 | 0.12 | 0.13 | 0.91 | 0.39 | 0.10 | 0.09 | 0.36 | 0.82 | 0 | |

TABLE VII: Confusion matrix for BotFinder with Dataset-B and Dataset-C together

| | Assigned label | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Sample | Ramnit | Lollipop | Kelihos˙ver3 | Vundo | Tracur | Kelihos˙ver1 | Obfuscator.ACY | Gatak | **Recall** |
| Ramnit | 768 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0.99 |
| Lollipop | 1 | 1236 | 0 | 0 | 0 | 1 | 1 | 0 | 0.99 |
| Kelihos˙ver3 | 0 | 0 | 1471 | 0 | 0 | 0 | 0 | 0 | 1 |
| Vundo | 0 | 0 | 0 | 236 | 0 | 0 | 1 | 0 | 0.99 |
| Tracur | 1 | 0 | 0 | 0 | 369 | 1 | 3 | 1 | 0.99 |
| Kelihos˙ver1 | 1 | 0 | 0 | 0 | 1 | 196 | 1 | 0 | 0.99 |
| Obfuscator.ACY | 4 | 0 | 0 | 1 | 0 | 0 | 607 | 2 | 0.99 |
| Gatak | 0 | 0 | 0 | 0 | 1 | 1 | 2 | 502 | 0.99 |
| **Precision** | 0.99 | 1 | 1 | 0.99 | 0.99 | 0.99 | 0.99 | 0.99 | |

TABLE VIII: Confusion matrix for the Microsoft Challenge

most problematic ones, sparing time that would otherwise be spent in a full analysis.

*1) Results with perturbed dataset:* We are going now to perturb the dataset of [27] using as ground truth the families in Table III. Table VII shows the confusion matrix and the performance matrix respectively. We can clearly see that the results obtained with the new dataset are very bad especially regarding Swizzor and Tibs families. Looking these results, we can reach the same conclusion that the decision assessment in Figure 6 suggested already with only the original dataset, i.e., without a good separation between correct and incorrect classifications, the results with a new dataset will dramatically change.

For completeness, we show the alpha assessment in Figure 16 included in Appendix A to show the heavy family interference.

## *C. Algorithm 3: Microsoft Malware Classification Challenge*

In this section we are going to evaluate one of the algorithms proposed as a solution for the Kaggle's Microsoft Malware Classification Challenge described in [2]. Table VIII reports the confusion matrix, precision and recall of the algorithm while Figure 8 shows the decision assessment. For the few misclassifications reported, we can see that the confidence is very low meaning that there was at least another family very close to the chosen one. It is interesting to note that even if the average credibility for correct choices is high (close to 1), the confidence on that choices is on average close to 0.4. This indicates that in general there are other families are not that much dissimilar to the correct one. However the disparity is still high enough not to pose serious classification problems (i.e. one minus the lowest confidence is still lower than the lowest credibility).

Looking at the confusion matrix, we can see that the performance of the algorithm regarding Kelihos˙ver3 and Obfuscator.ACY are similar. From a quality perspective instead (see Figure 10), the two families have different performances. Kelihos˙ver3 family has p-values much higher than the interfering families, which are very low with some outliers (see Kelihos˙ver3's samples boxplots). For Vundo instead, the p-values of interfering families are closer to the correct one. These facts, spotted by the alpha assessment only, indicates that the identification of the Kelihos˙ver3 samples will be similar when the underling ground truth change.

## VI. Discussion

It is widely known among the machine learning and security community, that assessing the effectiveness of an approach is not a solved problem (see § II). Throughout this paper,

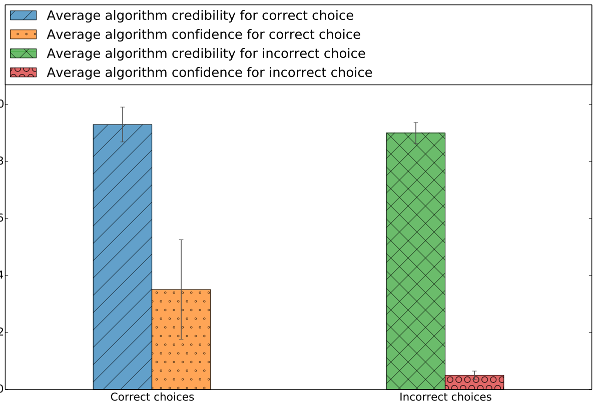

Fig. 8: Decision assessment for Microsoft Challenge algorithm: poor confidence for correct choices indicates the samples are similar to other families, poor confidence for incorrect choices is desirable but high credibility for incorrect choices it is not.

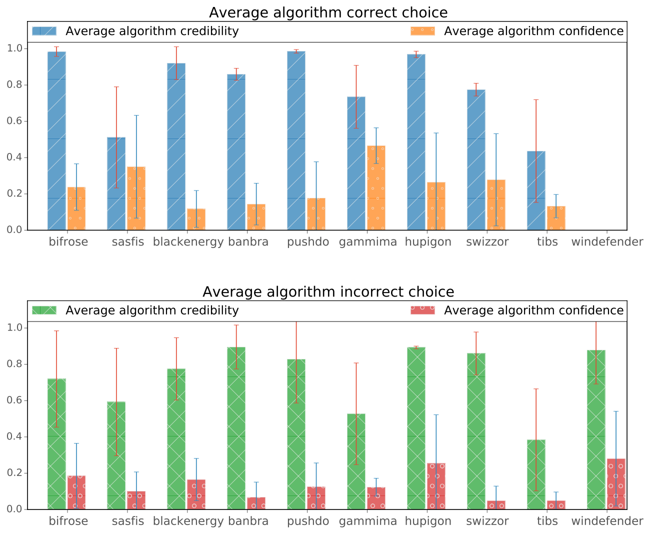

Fig. 9: Decision assessment for [27] with dataset-B and dataset-C: perturbing the dataset the results dramatically change.

we have shown how conformal evaluator new metrics can be used to assess consistency across different datasets, when such results are good as for Drebin in § V-A. Conversely we have also shown how apparently good results, according to traditional metrics, can be identified as troublesome (see § V-B). Moreover, as shown in V, traditional metrics can hide critical situations, (e.g., samples classified correctly by chance) which are otherwise unearthed by CE metrics. Specifically, conformal evaluator can help in identifying possible areas of improvement by narrowing down the problem to mutual interfering classes.

There are alternative methods that might unhortodoxely be used to look for inter-class interference, these are dimensionality reduction algorithms e.g. PCA [13] (Principal Component Analysis), t-SNE [29] (t-distributed stochastic neighbor embedding), LDA [12] (Linear discriminant analysis) or SOM [1] (self organizing maps). They leverage sample distribution to compute a new space that tries to maximize the distance (with different techniques) between the samples, or to project them onto an orthogonal feature space.

Although, these techniques might seem quite effective, they suffer from the following drawbacks:

- Space complexity grows exponentially with the number of features and samples (while CE on the other hand, is not significantly affected by it).
- The decision step made by the algorithm is not taken into account during the evaluation hence conclusions based on these methods might be misleading, as shown later on with an example.
- These techniques can operate on different space transformations with respect the ones used by the evaluated algorithm.
- The evaluation is limited to subjective visual inspection, while CE provides precise objective qualitative metrics.

Moreover, dimensionality reduction techniques are often used at early stages of the algorithm development process and dropped later and hence not even used in the final decision process. CE on the other hand, evaluates the final decision of the algorithm, taking into account all the operations happening within it.

We would like to remark that CE does not replace these methods, as they are usually part of the algorithm development process, however in order to have a comprehensive evaluation, the role of the algorithm must be taken into consideration.

We decided to provide evidence of the aforementioned limitations by applying PCA and t-SNE (as they are most spread among the security community) to Algorithm 1 (§ V-A) and Algorithm 2 (§V-B).

### A. *Algorithm 1: PCA and t-SNE Limitations*

Applying PCA to Drebin to represent its features in a 2D or 3D plot is of no use; the original features set is composed by more than 200K features and, after PCA, the variance expressed by keeping the 2 most variance-preserving features is less than 10% and less then 14% when considering 3 features. It is clear that every possible conclusions based on such plots would have no relevance.

With respect to the t-SNE analysis, this is still computationally intensive. In order to execute it we had to reduce the number of features to 1K. To choose meaningful features we performed t-SNE on the 1K most variance-preserving features output by PCA, this fact alone already shows a serious limitation in using t-SNE. Figure 11 shows t-SNE plot performed on a subset of 73.5K samples of the original 120K Drebin dataset samples. Particularly, the subset is composed of 70k benign apps and 3.5k malicious ones, with a ratio of 1:20 (original ratio 1:25). We decided to plot only a part of the

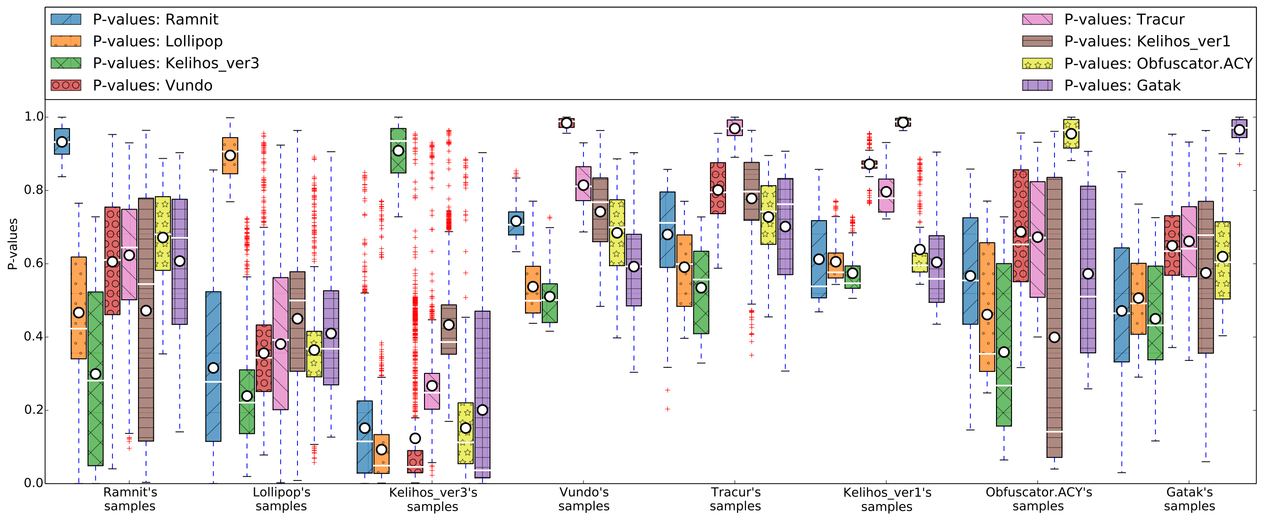

Fig. 10: Alpha assessment for the Microsoft classification challenge: from this picture we can see that quality of the decision taken by the algorithm. Behind very good results seen on the confusion matrix, the quality of those results is in general not optimal.

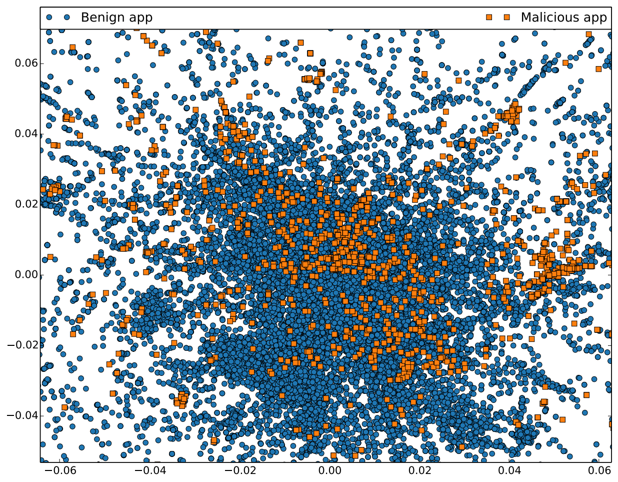

Fig. 11: t-SNE representation of Drebin dataset on 70k benign samples and 3.5k malicious samples with 1k features reduced with PCA. The malicious samples are superimposed over the benign ones, malicious apps are mixed with benign apps.

original dataset since we were interested in looking at how much benign and malicious apps are separated, furthermore the computational complexity of t-SNE increases with the square of the number of samples, making the analysis of large datasets computationally expensive. From Figure 11 we can clearly see that malicious applications are indistinguishable from benign ones[2] and zooming in the plot highlights even more how each malicious application is surrounded by many benign ones. We might even come to the wrong conclusion that the features used in the algorithm are not good enough to distinguish between malicious and benign applications since the interference between the two is very high.

Without the limitations imposed by these techniques, as pointed in § V-A, conformal evaluator shows instead that the synergy between features and the algorithm produces very limited interference between benign and malicious samples.

## B. Algorithm 2: PCA and t-SNE Limitations

Differently from Drebin, BotFinder has a very low number of features, hence PCA and t-SNE can be performed over the whole original dataset. Figures 12 and 13 shows the 2D plots for PCA and t-SNE respectively. Figure 12 shows that the use of PCA in the analysis of BotFinder features is ineffective. Most of the families are close to each other, concentrated in a small portion of the space and mixed together so that it seems difficult to tell families apart.

The t-SNE projection is shown in Figure 13. From the plot it seems that Banbra is isolated in one cluster. For the other families, some small clusters can be identified even if most of them are mixed together. Following the t-SNE projection, seems that the chosen features might be a good starting point to

[2]in the plot, malicious apps ore superimposed over benign apps, otherwise they would not be visible.

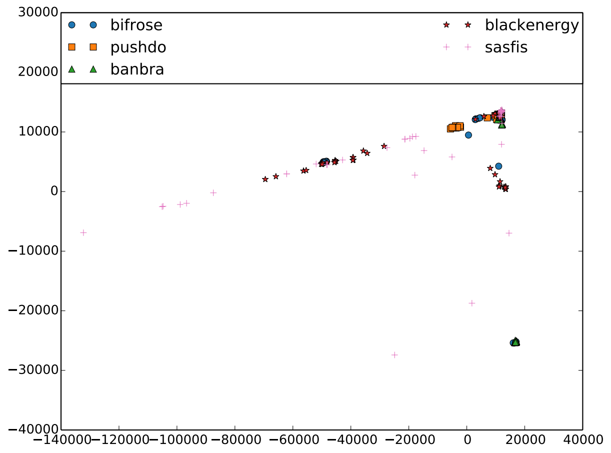

Fig. 12: Zoomed part of PCA representation of BotFinder dataset. From this view we can see that are few clusters of samples are isolated from the others but the vast majority is not well separated.

separate the families. On the other hand, our analysis on § V-B shows that families are very much interfering each others.

Quantifying the amount of family interference by looking at PCA and t-SNE only is quite difficult. This is also due to the fact that to perform an analytic comparison using t-SNE and PCA, one needs to choose an algorithm that is able to correctly group together the samples of a family (e.g., a clustering algorithm). Clearly the choice of the algorithm already influences the outcome of the analysis, furthermore we feel like we are going around in circles (to evaluate a classification/clustering algorithm we have to choose a classification/clustering algorithm). This is why only a visual evaluation of the figures is wort to discuss. Of course visual

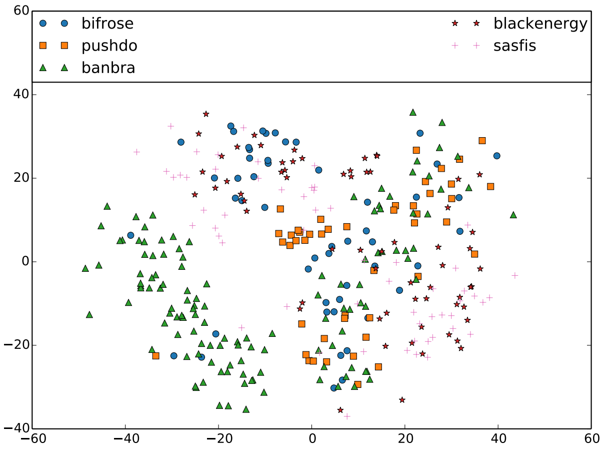

Fig. 13: t-SNE representation of BotFinder dataset- B. Banbra seems isolated in one cluster while for the other families some small clusters can be identified, nevertheless remaining samples are mixed together.

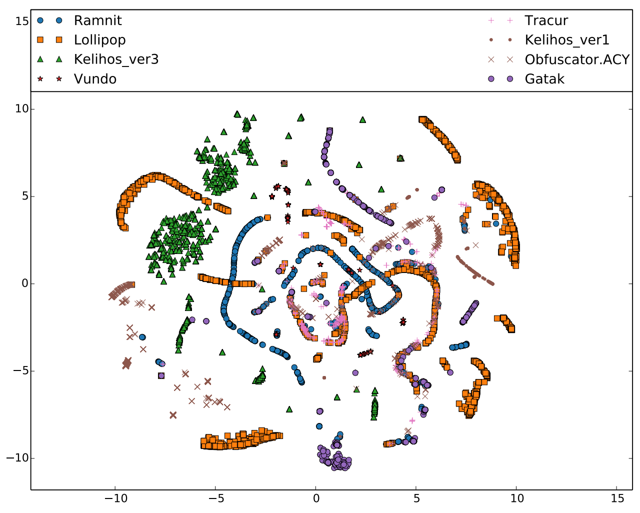

Fig. 14: t-SNE projection of the features used in the Microsoft challenge: families appear to be well separated from the others in the outer parts of the figure while in the middle they seems mixed together.

evaluation is subject to personal interpretation and could not be conveyed uniformly to the community.

With CE instead, we enable the evaluation of the actual algorithm under quantifiable and objective measures.

### C. *Algorithm 3: PCA and t-SNE Limitations*

From the t-SNE projection of the features extracted for the Microsoft challenge (Figure 14), we can see that some of the classes are well separated from the others, while other classes seems to be mixed together. From this picture it's difficult to imagine a confusion matrix with so few misclassification as the one that we get in Table VIII. Even in this case, the algorithm plays an important role in the classification, and hence excluding it from the evaluation is not ideal.

As for PCA, the features extracted and plotted in Figure 15 retain in total 99.99% of the variance (i.e., are reliable as much as the original features). As we can see, the family Ramnit has the most of the variance in the features. Not surprisingly, even PCA is not ideal to fully evaluate the features.

### D. *Final Remarks*

Following the discussion on the limitations of dimensionality reduction techniques, we have shown how these can lead to wrong conclusions if you don't consider the algorithm altogether with the features. Particularly, Algorithm 1 bad results shown on t-SNE are overturned when considering the algorithm. Conversely, Algorithm 2 apparently good results shown by t-SNE leads to very poor results when bringing the algorithm into the picture. As for Algorithm 3, the analyses are more promising, however drawing a precise and quantifiable conclusion is not as direct as we might think. For these reasons, the algorithm plays an important role and hence cannot be dismissed during the evaluation.

## VII. Limitations

Throughout the paper we have explained the benefits of using conformal evaluator by showing how it can be used to assess the quality of malware classification and clustering algorithms. However conformal evaluator has also its drawbacks, which we outline next along with possible ways to address them.

The main limitation consists in the fact that in order to apply conformal evaluator to any machine learning technique, the latter needs to have a similarity function, or a similar concept, that can be shaped as a non-conformity measure. However limiting this might seem, conformal evaluator can still help in evaluating stages of a larger process, that relies on methods based on a similarity function. For instance, FIRMA [20] relies on token-set payload signatures to identify malware. This is a found/not-found classification approach that cannot directly be translated into a non-conformity measure. Nevertheless, FIRMA internally relies on clustering techniques whose quality could be assessed through conformal evaluator. There are also other approaches where the application of conformal evaluator seems not possible or at least very complex. In [15], for example, the authors use a locality sensitive hashing algorithm to tell whether two malware samples are similar. This algorithm cannot be directly translated into a non-conformity measure, because its basic theory relies on some specific distance functions between pairwise samples, that are translated into hash functions. Still, with some effort, the algorithm could be converted into a non-conformity measure (i.e., distance from one group to one element). This is indeed an interesting area to explore further in the future.

Another area of concern refers to the computational complexity of conformal evaluator. The underling machine learning algorithm, conformal predictor, used by conformal evaluator is computationally expensive. For each sample $z$ in a

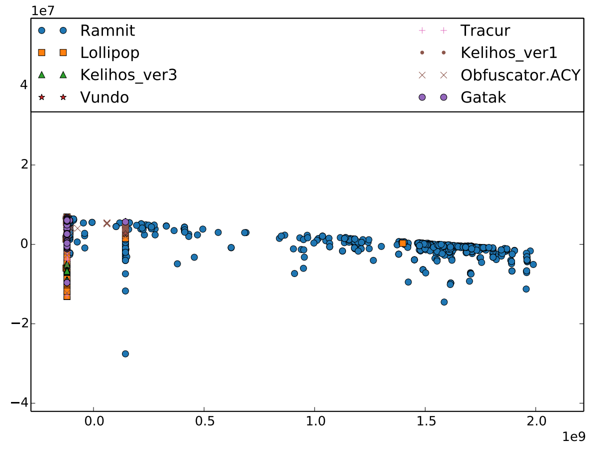

Fig. 15: 2D PCA projection of the features used in the Microsoft challenge: Ramnit family retains most of the variance while the other families are concentrated on small portion of the graph.

class $c \in C$, the production of a p-value requires to compute a non-conformity measure for every element in the whole dataset. This can further be exacerbated by non-conformity measures that rely on distances that are complex to compute. For instance, BotFinder [27] builds one or more models to profile malware behaviors, and then uses each model as a part in the computation non-conformity measure.

The computational complexity in relation to the number of the times that the non-conformity measure needs to be computed is expressed in Equation 3:

$$\mathcal{O}(CN^2) \tag{3}$$

Here $N$ represents the total number of samples and $C$ represent the number of classes.

To speedup the operations, most of the time we can compute a whole set of non-conformity scores in one single algorithm run. For example SVM used in [3] can directly supply the total non-conformity scores for the calculation of one p-value in only one run of the algorithm, thus reducing Equation 3 into Equation 4:

$$\mathcal{O}(CN) \tag{4}$$

To further speed up the process, some algorithms treat each class separately. For example [27] uses a separate model for each class. In this case, it's useless to re-run the algorithm for all the classes and we can make it run just for the class that is currently under analysis.

To have a concrete example, let's focus on a dataset composed by 2000 samples and 10 classes. One single run of the algorithm will require 10 minutes. The overall time needed for the evaluation will be then (by Equation 4):

$$2000 * 10 = 20000 \textit{ minutes} \approx 13 \textit{ days} + 22 \textit{ hours}$$

Waiting this much might not be optimal for some situations where one does not simply have the luxury to wait. For this reason we take advantage of the fully parallelizable calculation process speeding up the operation.

If we distribute the operation over for example just 8 processes, (e.g., a standard hyper-threaded CPU), the estimated time already drops to:

$$20000 \textit{ minutes}/8 \approx 2 \textit{ days}$$

Throughout our experiments with [3], which has a rather large dataset ($\approx 129K$ samples over $\approx 200K$ features for each sample) the analysis took 18 days and 2 hours with the optimized complexity with a standard i7 CPU with 4 cores/8 threads dedicated to the evaluation. The analysis was also run on a high-end Xeon CPU, with 23cores/46 threads, and took approximately 3 days.

Regarding memory complexity and consumption, we have not noticed any difficulties to handle the workload by a standard desktop workstation with 16 GB of RAM, as the evaluation process took around 8 GB of RAM.

To further speedup the evaluation, a potential solution to the complexity requirements of conformal predictor has recently been addressed by the machine learning community. They propose an alternative to the traditional conformal predictor which is known as *inductive conformal predictor* (ICP) [6]. ICP divides the training set into proper training set and calibration set. Only the calibration set is then used to compute p-values during the clustering or classification steps, which relaxes considerably the computational resources required by traditional conformal predictor.

Even if optimizations can be put in place to reduce the computational complexity of conformal predictor, we want to stress the fact that our framework is meant for evaluation purposes only, hence for an off-line scenario, when the algorithm is not yet deployed in the field. For this reason performance optimizations are not a primary issue for us and are not in the scope of this work.

## VIII. Conclusions

Assessing the validity of malware clustering and classification approaches has always proven difficult. Researchers have empirically shown that traditional metrics fall short of assessing the actual quality of a classification/clustering methodology. To address such shortcomings, we have presented conformal evaluator, an evaluation framework built around conformal predictor, a machine learning algorithm originally designed for tailoring classification tasks. Within conformal evaluator, we proposed two novel analyses, which are able to evaluate similarity-based classification and clustering algorithms with respect to their own decisions and against the data set itself. To this end, such analyses are built on top of algorithm confidence and credibility jointly which ultimately enable to assess whether the similarity function underpinning such machine learning approaches is poorly designed or badly handled.

We have demonstrated the usage of conformal evaluator through three malware detection and classification algorithms belonging to three different areas (i.e. botnet network communication, windows and android malware), and we have shown how apparently good results, according to traditional metrics, can result in inconsistent performances, according to conformal evaluator metrics. Additionally, we have demonstrated the robustness of the conformal evaluator results by running the same tests on different datasets.

By building tools that allow us to study and investigate the problem in a more comprehensive and scientific manner, we realize that we are merely opening a whole new Pandora's Box; our hope is that we might stimulate researchers to increase their efforts in the same direction hence bringing more insights than merely numerical results.

Our ultimate desire is to marry the machine learning and the systems security community to provide a toolkit to the latter to understand the subtle implications of machine learning-based techniques (e.g., choosing a similarity measure over another), and better support their claims.

## IX. Availability

Our library implementation of conformal evaluator as a technique to understand clustering and classification results is available at www.example.com[3].

## References

[1] H. S. A. Ultsch. Kohonen's self organizing feature maps for exploratory data analysis. In *International Neural Network Conference*, volume 2, page 305308, 1990.

[2] M. Ahmadi, G. Giacinto, D. Ulyanov, S. Semenov, and M. Trofimov. Novel feature extraction, selection and fusion for effective malware family classification. Technical Report, arXiv:submit/1402914, 2015.

[3] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, K. Rieck, and C. Siemens. Drebin: Effective and explainable detection of android malware in your pocket. In *Proc. of NDSS*, 2014.

[4] A. J. Aviv and A. Haeberlen. Challenges in experimenting with botnet detection systems. In *Proceedings of the 4th USENIX Workshop on Cyber Security Experimentation and Test (CSET '11)*, 2011.

[5] M. Christodorescu, S. Jha, and C. Kruegel. Mining specifications of malicious behavior. In *Proceedings of the 1st India Software Engineering Conference*, ISEC '08, pages 5–14, New York, NY, USA, 2008. ACM.

[6] A. Gammerman and V. Vovk. Hedging predictions in machine learning. *Computer Journal 50*, pages 151–177, 2007.

[7] S. García, A. Zunino, and M. Campo. Survey on network-based botnet detection methods. *Security and Communication Networks*, 7(5):878903, May 2014.

[8] G. Gu, R. Perdisci, J. Zhang, and W. Lee. Botminer: Clustering analysis of network traffic for protocol- and structure-independent botnet detection. In *Proceedings of the 17th Conference on Security Symposium*, SS'08, pages 139–154, Berkeley, CA, USA, 2008. USENIX Association.

[9] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee. Bothunter: Detecting malware infection through ids-driven dialog correlation. In *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*, SS'07, pages 12:1–12:16, Berkeley, CA, USA, 2007. USENIX Association.

[10] M. Hubert and E. Vandervieren. An adjusted boxplot for skewed distributions. *Computational Statistics and Data Analysis*, 52(12):5186 – 5201, 2008.

[11] K. Inc. Microsoft malware classification challenge (big 2015), 2015.

[12] A. J. Izenman. Linear discriminant analysis. In *Modern Multivariate Statistical Techniques*, pages 237–280. Springer, 2008.

[13] I. Jolliffe. *Principal component analysis*. Wiley Online Library, 2002.

[14] J. K. Kevin Allix, Tegawendé F. Bissyandé and Y. L. Traon. Machine learning-based malware detection for android applications: History matters! 2014.

[15] C. Kruegel, E. Kirda, P. M. Comparetti, U. Bayer, and C. Hlauschek. Scalable, behavior-based malware clustering. In *Proceedings of the 16th Annual Network and Distributed System Security Symposium (NDSS 2009)*, 1 2009.

[16] P. Li, L. Liu, D. Gao, and M. K. Reiter. On challenges in evaluating malware clustering. In *Recent Advances in Intrusion Detection*, pages 238–255. Springer, 2010.

[17] McAfee Labs. Threats Report. http://www.mcafee.com/uk/resources/reports/rp-quarterly-threats-aug-2015.pdf, Aug 2015.

[18] R. Perdisci, D. Ariu, and G. Giacinto. Scalable fine-grained behavioral clustering of http-based malware. *Computer Networks*, 57(2):487–500, 2013.

[19] R. Perdisci, W. Lee, and N. Feamster. Behavioral clustering of http-based malware and signature generation using malicious network traces. In *NSDI*, pages 391–404, 2010.

[20] M. Z. Rafique and J. Caballero. Firma: Malware clustering and network signature generation with mixed network behaviors. *Proceedings of the 16th International Symposium on Research in Attacks, Intrusions and Defenses*, 2013.

[21] G. Ridgeway. The state of boosting. *Computing Science and Statistics*, pages 172–181, 1999.

[22] K. Rieck, T. Holz, C. Willems, P. Dssel, and P. Laskov. Learning and classification of malware behavior. In *Detection of Intrusions and Malware, and Vulnerability Assessment, Lecture Notes in Computer Science*, pages 108–125, 2008.

[23] L. Salgarelli, F. Gringoli, and T. Karagiannis. Comparing traffic classifiers. *ACM SIGCOMM Computer Communication Review*, 37(3):65–68, July 2007.

[24] G. Shafer and V. Vovk. A tutorial on conformal prediction. *The Journal of Machine Learning Research*, 9:371–421, 2008.

[25] R. Sommer and V. Paxson. Outside the Closed World: On Using Machine Learning for Network Intrusion Detection. In *IEEE Symposium on Security and Privacy*, pages 305–316, 2010.

[26] Y. Tang. extreme gradient boosting.

[27] F. Tegeler, X. Fu, G. Vigna, and C. Kruegel. Botfinder: Finding bots in network traffic without deep packet inspection. In *In Proc. Co-NEXT 12*, pages 349–360, 2012.

[28] A. G. V. Vovk and G. Shafer. *Algorithmic learning in a random world*. Springer-Verlag New York, Inc., 2005.

[29] L. Van der Maaten and G. Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(2579-2605):85, 2008.

[30] P. Wurzinger, L. Bilge, T. H. J. Goebel, C. Kruegel, and E. Kirda. Automatically generating models for botnet detection. In *Proceedings of the 14th European conference on Research in computer security (ESORICS2009)*, pages 232–249. Springer-Verlag Berlin, Heidelberg, 2009.

[31] Z. Yajin and J. Xuxian. Dissecting android malware: Characterization and evolution. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, pages 95–109, 2012.

## Appendix A
## Alpha Assessment for [27] with Dataset-B and Dataset-C

[3]To comply with submission requirements, the actual URL has only been provided to the PC Chairs.

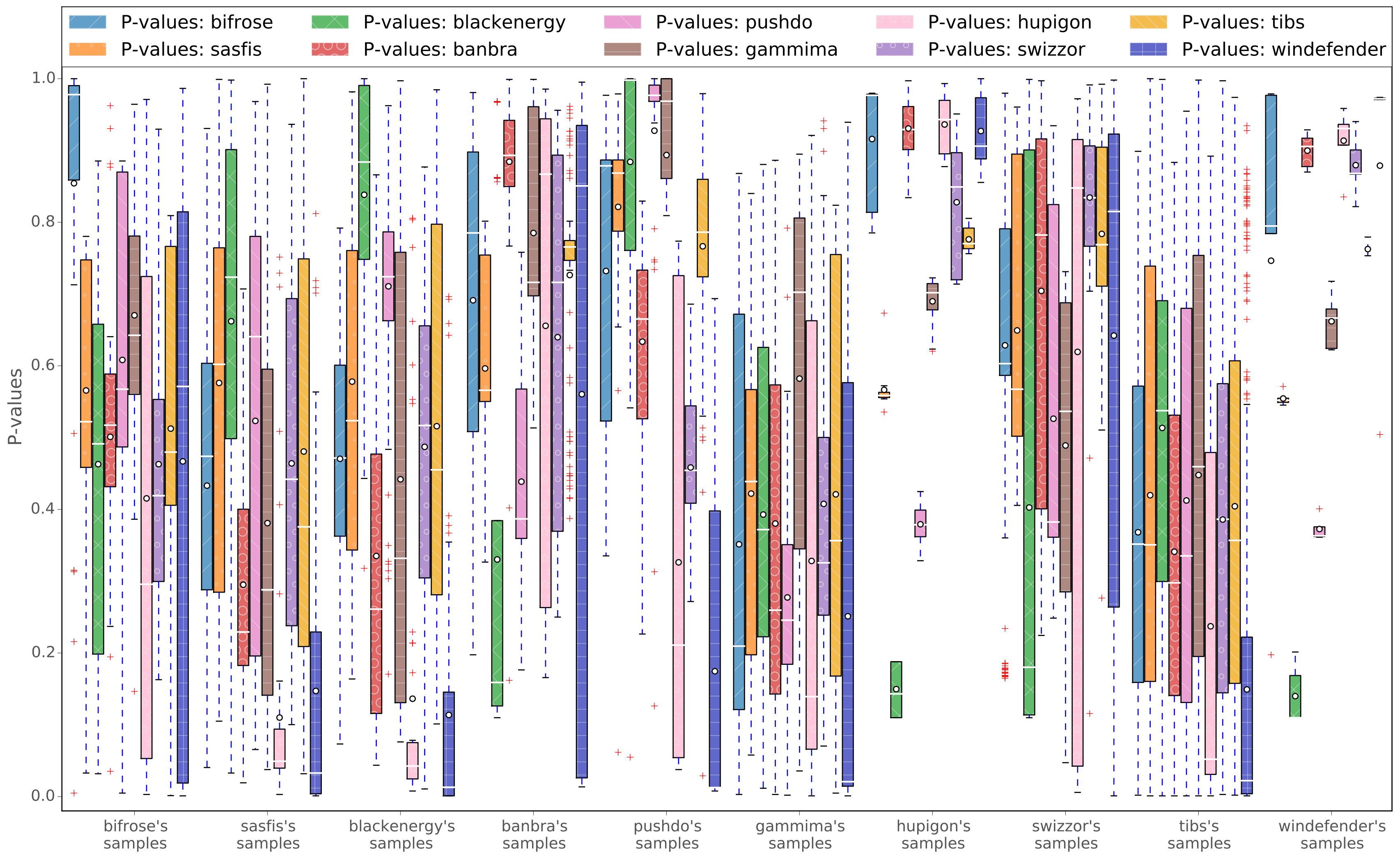

Fig. 16: Aplha assessment for [27] with dataset-C and dataset-B representing families interference