ROYAL HOLLOWAY AND BEDFORD NEW COLLEGE

UNIVERSITY OF LONDON

INFORMATION SECURITY GROUP

PhD Dissertation

Machine Learning Techniques for Evolving Threats

Supervisor
Prof Lorenzo Cavallaro

Candidate
Roberto Jordaney

March 2019

Summary

Malicious threats pose a serious problem for everyday activities. The number of attacks are always increasing and automatic means of analysis have been employed to deal with this growth. In this scenario, it is difficult to trust the automatic decision making-criteria because they are often based on the assumption that the objects used to learn the malicious patterns are similar to the ones in need of an assessment. This is usually not the case because the strategy perpetrated by malicious actors is ever-evolving to defeat new defence mechanisms.

For this reason, I have developed Conformal Evaluator, a statistical assessment framework that provides quality measures to the decisions of an existing classifier. The goal of the framework is to enrich the information provided by the classifier with quality measures. With this framework, I assessed the quality of the decision-making process of 3 algorithms and provided interesting insights.

Using conformal evaluator framework, I developed Transcend, a technique that aims to identify the start of the performance degradation due to a change in the testing distribution. Transcend is then successfully applied to 2 algorithms on an Android binary and a Windows malware multiclass classification settings. This technique shows that it is possible to identify thresholds below which it is not wise to trust the outcome of a classification.

To further investigate the link between malicious actors and evolving malicious strategies I looked into the beginning of an infection. The first step is often that a malicious software is downloaded and installed by an unaware user. Cyber criminals often target users with online malicious campaigns inducing them to install malicious software. To tackle this problem, I developed an algorithm that aims to identify malicious downloads before the actual executable is downloaded. The system was tested with the traffic generated from a major US university producing interesting results.

Acknowledgement

I would like to thank all the people that during my PhD years supported me in various ways. First of all, I would like to thank my supervisor Professor Lorenzo Cavallaro for believing in me and pushing me to overcome my limits. I also would like to thank Davide and Zhi for their guidance and all the other colleagues for making the lab an enjoyable and learning workspace. Finally, I want to thank Giulia and my family for their endless patience and support during the years.

Index

List of Figures

List of Tables

Part I

Introduction

Chapter 1

Introduction

Malicious software is one of the most pressing problems in the Internet nowadays. According to different anti-virus firms [66, 99, 48], the total amount of malware continued to rise in 2018. McAfee has reported that it needs to analyse 1.8M URLs and 1M files every day. With such numbers, it is clear that new challenges need to be faced in order to scale up the analyses.

Automatic means of analysis such as machine learning (ML) have become the preferred tool to deal with this problem as it has proven to be successful in many applications including malware analysis and malicious URL detection. ML techniques usually build classification models to be able to distinguish between benign phenomena and malicious ones. The main shortcoming of these applications is that they implicitly rely on the assumption that the threats do not change over time, (i.e., the population is static), but it is not the case.

Once a malware has been identified and the defence mechanism updated, its author typically tries to modify it in order to avoid the detection, thus creating a new variant. The amount of variation introduced by the new variant depends on the complexity of the detection mechanism that was able to identify the malware itself. If the detection mechanism hits the core behaviours of the malware, then the modification needed by its author to avoid detection will need to be more substantial, otherwise a shallow modification can trick the defences as well. When a new variant of the malware is released it is possible to think of it as an evolution of its previous version.

To deploy malware to a user's machine, attackers typically rely on different infection mechanisms. A common technique involves tricking the user to download and install programs that are thought to be benign. The attackers organise these attacks in malicious campaign that usually share some common roots. The goal of the defence mechanisms is to identify the unique traits of the campaign to distinguish its *modus operandi*. Nowadays, typical campaigns can run between two weeks to up to a few months before being identified. In such evolving scenarios, it is clear that the attacker needs a way to change the offending strategy very often.

In this ever-evolving scenario, statistical learning tools such as machine learning need to be carefully employed to maximise their efficiency as they will be affected by concept drift sooner or later. Therefore, it is important to understand when it is time to change the model or modify the employed techniques.

Concept drift is the change in the statistical properties of a certain phenomenon over a period of time. It affects malware classification tasks: on one hand, malware evolves over time to bypass new defence mechanisms, while, on the other hand, malware campaigns evolve likewise as new actors and new forms to exploit technical and social flaws are identified by the attackers. Regardless the degree of the drift, it is important to be able to identify this drift to promptly apply remedial actions. This is because concept drift will cause the performance of a classification model to degrade as a result of the model built to deal with a specific target population that eventually changes.

The classification degradation is only observed once the classified threats are manually verified. At that point, it is too late to apply the correct remedial action and to stop the threat.

1.1 Research Contributions

These problems are tackled in the literature typically by two approaches. The first one is machine learning oriented and focuses on statistical techniques. The second approach focuses on algorithmic techniques specific to the domain of interest in order to delay the problem, as much as possible. In my research work, I have tackled the problem using both ways.

Firstly, I proposed a new framework to measure the quality of existing classifiers named Conformal Evaluator (§ 3). This method is based on two novel quality metrics *algorithm credibility* and *algorithm confidence* (§ 3.2) that quantify the quality of the predicted outcome.

Secondly, on top of this framework, I developed and proposed the use of a new algorithm named Transcend (§ 4). The algorithm uses the quality metrics of conformal evaluator to try to find per class threshold to distinguish between drifted and non-drifted samples by maximising an objective performance function subject to constraints.

These two methods, even if used together, can be used separately from each other. Conformal Evaluator can be used to infer the quality of a classifier alone or in conjunction with other algorithms to identify concept drift. Likewise, Transcend can be used to find per class thresholds using different quality metrics.

It must be pointed out that even if the two methods will be demonstrated in the scope of the systems security, they can be applied to other disciplines, making them very versatile.

Finally, I proposed a new algorithm to deal with concept drift in the malicious

URL detection domain (§ 5). This algorithm exploits the history of HTTP requests generated from a host in a new fashion. This technique was adopted to identify the drift of the URL used in malware campaign to distribute malware.

Chapter 2

Machine Learning and Security

This chapter provides the necessary machine learning (ML) information to understand the research part of my dissertation (Part II).

ML algorithms are generally used to identify patterns across data.

Data is usually represented as a list of examples:

$$(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n) \tag{2.1}$$

Here, $x_1, \ldots, x_n$ are n observations and $y_1, \ldots, y_n$ are the responses to those observations and x_i is a vector of size p. In a systems security domain for example, the x_1 can represent some measurements of a given binary while y_1 can represent the label *malicious* or *benign*.

The matrix X represents the ensemble of the observation $x_{1\ldots n}$:

$$X = \begin{pmatrix} x_{11} & x_{12} & \ldots & x_{1p} \\ x_{21} & x_{12} & \ldots & x_{1p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \ldots & x_{np} \end{pmatrix} \tag{2.2}$$

and Y is the ensemble of the the responses:

$$Y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} \tag{2.3}$$

Eq. 2.3 implies that the response for each observation is a single variable, nevertheless the equation can be generalised to accommodate for a multi-variable response.

We assume that there is a relation between the data defined as:

$$Y = f(X) + \epsilon \tag{2.4}$$

Where ϵ represents an error term. Therefore, the goal of machine learning is to find:

$$\hat{Y} = \hat{f}(X) \tag{2.5}$$

Where $\hat{f}$ is the best estimation for f, and $\hat{Y}$ are the estimations for the true Y.

Many algorithms were created for most varied purposes, therefore, in § 2.2.1, I will explore the ones that are needed to understand Part II of this thesis.

In § 2.1 I explain how to manage the available dataset to run the experiments properly.

2.1 Training, Validation and Testing Set

Machine learning algorithms are usually used to make future predictions. These algorithms work by building a prediction model and test it to make sure it is as accurate as possible once used in operational settings.

The dataset available during the construction of one prediction model is usually divided into 3 sets with different purposes: *training set*, *validation set* and *testing set*. The usual separation of the available dataset is: 60% for the train set, 20% for the validation set and 20% for the test set. This separation is usually accepted as common criteria but different splits are possible.

One important thing to note is that with more training and validation data, the variance of the generated model will be reduced while increasing the size of the test set will decrease the variance on the results. This is a trade-off that needs to be considered.

Common techniques that divide the data into the different sets are random sampling and stratified sampling (see [75] and [44] for further details). I will not go deeper into these techniques because they are beyond the scope of this thesis.

Training Set. The training set is the set of examples used to fit the model chosen to be the prediction model. It must be chosen carefully to cover the entire population such that the training algorithm can learn the difference between the categories. It is crucial because it might bias the decisions made by the algorithm (as shown in [76]).

Validation Set. The validation set is the set of examples used to predict the responses with the model created with the training set. The purpose of this set is to evaluate the performance and tune the algorithm accordingly, i.e., to optimise the parameter of the prediction model in order to get better performance. Another use of the validation set is to avoid overfitting of the model to the training data. Overfitting is a problem caused by ML algorithms that learn the patterns on the training data too precisely and are not able to generalise on new and unseen instances. Overfitting causes the error obtained on the test data to increase, instead of decreasing, when more training data is added to the training set.

Testing Set. The testing set is a set of examples used to report the performance obtained once the ML algorithm is optimised and ready to be deployed. The training and validation sets need to be strictly separated from the test set. This separation is crucial to avoid biased results. On the other hand, building a prediction model on the same data that is later on used to assess its performance will generate unrealistic scenarios.

Throughout this thesis, I will refer to training phase to describe the creation and optimisation of the model using both train and validation sets.

2.1.1 Cross-Validation

Cross-validation (CV) is a re-sampling technique that is used for model validation. It is used to estimate the test error of a model. It involves *holding out* a subset of training samples from the training process and then testing the model created with this subset. The error of the testing is given by:

$$err = \frac{1}{n}\sum_{i=1}^{n}(y-\hat{y})^2 \tag{2.6}$$

where, n is the number of testing samples in that subset, y is the true value for a sample and $\hat{y}$ is the corresponding estimation for that sample.

Cross-validation is often used with its *k-fold* variant. K-fold cross validation consists of dividing the dataset in k non-overlapping folds. The model will be trained with the $k-1$ folds and the remaining one will be used to test the model. This process will be repeated k times; every time choosing a different testing fold. Finally, the error of the model will be calculated as:

$$err = \frac{1}{k}\sum_{i=1}^{k} err_k \tag{2.7}$$

where err_k is the error of one fold calculated following Eq. 2.6. When k is equal to N (the number of the examples in the dataset) this technique is called *leave one out* (LOO or LOOCV).

CV technique is often used to show that the model created is performing well not only on a specific configuration of the train/test set. Nevertheless, it should be carefully used to avoid any unwanted biases (see § 2.4).

2.2 Algorithms

Traditional machine learning algorithms are based on established techniques studied by the ML community over the years. This section will provide a brief overview of different ML algorithms available in the literature. Depending on the characterisation of Y and f we may use different algorithms.

2.2.1 Taxonomy

Nowadays, there is a great variety of machine learning algorithms for many different data scenarios. Most of them perform very well in particular situations while do not at all in others. The following categories will describe some of the most known algorithms based on the goal of the analysis and on the data available.

Regression vs Classification. Depending on the type of the response that we would like to obtain from a ML algorithm, Y (in Eq. 2.4) can be *quantitative* or *qualitative* (or *categorical*). Quantitative variables take numerical values such as 1.43, 302, ...and so on. For example, predicting the height of children based on the height of their parents is a problem that require a quantitative response. In such cases, we are talking about *regression* problems.

On the other hand, when the responses of our ML algorithm can take k different classes or categories, the problems are referred as *classification* problems. For example, predicting if a binary artefact is a malware or a benignware is a classification task as the output of the prediction can take only 2 values (in this case is a binary classification problem opposed to multiclass classification problem where the output variable can take 3 or more values).

In this thesis, I will focus my attention on classification problems only.

Supervised vs Unsupervised. Depending on the availability of the response Y in Eq. 2.4, we can have *supervised* or *unsupervised* (or semi-supervised) algorithms. Of course, it is possible to ignore the responses in a given dataset.

The algorithms that try to associate a response Y to predictors X are called *supervised* algorithms.

If the response Y is missing from the dataset (or if we are trying to solve a different problem) then we are dealing with *unsupervised* algorithm. In this type of problems, the goal is to understand the relation between the predictors of the available instances rather than the relation between the predictors X and the responses Y. Problems like cluster analysis and outlier detection fall in this category.

Parametric vs Non-parametric. Depending on the type of assumption that we are making on the data we can have *parametric* or *non-parametric* algorithms.

Parametric algorithms are those that make an assumption on the form of f in Eq. 2.4. Once the assumption is made, the goal of the algorithm is to find the parameters of the function f that minimise the prediction error. Examples of these algorithms are SVM (§ 2.2.2) and linear regression [15].

Non-parametric algorithms do not assume a specific form for the function f. Instead, they try to estimate f to be as close as possible to the data points. Example of these algorithms are random forest (§ 2.2.3) and K-Nearest neighbour [18].

2.2.2 Support Vector Machine

The Support Vector Machine (SVM) is a supervised parametric machine learning algorithm used for classification. It was originally designed to be a binary classification algorithm but it was extended for multiclass purposes [108]. It has proven to be successful in many different applications [74, 25, 73] including systems security [49, 55] and it has been used in Drebin [9], one of the algorithm used for my case studies (§ 3.4). SVM is a generalisation of the Support Vector Classifier that by itself is a generalisation of the maximal margin classifier. Therefore, I will first describe the maximal margin classifier, then the support vector classifier and finally the support vector machine. I will give an intuition for the algorithm then I will go through these algorithms to finally end up explaining the SVM.

Intuition. The SVM algorithm divides the feature space with an *hyperplane* in 2 subspaces. The hyperplane is a $p-1$ dimensional space where p is the number of dimensions of the features space. During the construction of the model, the algorithm maximises the distance from each instance to the hyperplane. During the test phase, it will classify each instance depending on which side of the hyperplane the instance will lie.

Different shapes of the hyperplane are possible, e.g., linear, polynomial and sigmoid. The following explanation starts to assume the hyperplane as linear to generalise the definition later on.

Maximal Margin Classifier. The definition of hyperplane is:

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p = 0 \tag{2.8}$$

Eq. 2.8 implies that each example $X = (x_1, x_2, \ldots, x_p)^T$ that satisfy the equation lies on the hyperplane itself.

If Eq. 2.8 is not satisfied, it follows that either:

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p > 0 \tag{2.9}$$

or:

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p < 0 \tag{2.10}$$

i.e., X is located on one side of the hyperplane.

Let's suppose that we have different training objects $x_1, x_2, \ldots, x_n$ each with its corresponding $y_1, y_2, \ldots, y_n \in -1, +1$ (i.e., the points lies on one side of the hyperplane). If the training data is in fact separable, then there will be infinite hyperplanes that can separate them. The maximal margin classifier selects the one that is the farthest from the training observations. The distance is calculated as the perpendicular distance from the training objects to the hyperplane and the smallest distance of these distances is called *margin*. In other words, the hyperplane to select is the one with the highest margin. In this setting we there will be only one hyperplane to satisfy this condition. It follows that there will be at least one sample for each class that lies on the margin on either sides of the hyperplane. These samples are called *support vectors* because all the other points of the training set can move freely without changing the hyperplane (if they don't cross the margin). Therefore, the hyperplane depends only on few points of the dataset but if these samples change, the hyperplane changes as well.

More formally, to find the separating hyperplane we want to maximise M in:

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + +\beta_p x_{ip}) \geq M \;\; \forall i = 1, \ldots, n \tag{2.11}$$

subject to $\sum_{j=1}^{p} \beta_j^2 = 1$.

Without going in much details, it turns out that the solution of Eq. 2.11 can be solved quite efficiently but the details are beyond the scope of this thesis.

The distance from a point to the hyperplane can be seen as a measure of *confidence* in the prediction when the model is used to classified test points.

Support Vector Classifier. The maximal margin classifier is a *lucky* case when it exists an hyperplane that can perfectly separate the training data. In other cases instead, we are not so lucky and there is no hyperplane that can separate the data, i.e., the Eq. 2.11 doesn't have a solution for $M > 0$. Sometimes instead, a separation hyperplane exists but we will become too sensitive to single observations.

To cope with these situations, the support vector classifier allows some observation to be on the wrong side of margin and even on the wrong side of the separation hyperplane.

More formally, to find the separating hyperplane in this scenario, we want to maximise M in:

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + +\beta_p x_{ip}) \geq M(1 - \epsilon_i) \;\; \forall i = 1, \ldots, n \tag{2.12}$$

subject to $\sum_{j=1}^{p} \beta_j^2 = 1$, where ϵ_i are variables that allow individual observations to be on the wrong side of the separating hyperplane. These variables are subject to:

$$\epsilon_i \geq 0 \quad \sum_{i=1}^{n} \epsilon_i \leq C \tag{2.13}$$

where C is tuning parameter aimed to bound the sum of ϵ_i. Greater is C greater will be the amount of violation allowed by individual instances x_i. The parameter C can be seen as a bias-variance trade-off for this technique.

Similarly as before, also in Eq. 2.12, only the samples that are violating the margin or the samples that are in the wrong side of the hyperplane affect the equation. This means that the other points can change position in the space without changing the position of the hyperplane.

Support Vector Machine. The maximal margin classifier and support vector classifier function properly when the separating hyperplane is linear. To deal with non-linear separating hyperplane the support vector machine (SVM) is needed. SVM enlarges the original feature space in specific ways using *kernels*. Enlarging the feature space is used to address non-linearity by adding quadratic, cubic or non-polynomial function of the predictors thus resulting in a non-linearity in the original features space.

In SVM the feature space is enlarged as function of the predictors. In this way, the decision boundary will still be linear in the enlarged feature space but it will result generally non-linear in the original feature space.

It can be shown that Eq. 2.11 and Eq. 2.12 can be rewritten as inner products of its observation:

$$K(x_i, x_{i'}) = \langle x_i, x_{i'} \rangle \tag{2.14}$$

where K is defining a kernel, specifically, Eq. 2.14 is defining a linear kernel. By modifying Eq. 2.14 it is possible to define other kernels. Polynomial kernels and radial kernels are the most common but custom ones are possible.

2.2.3 Gradient Boosting

Gradient boosting is a method to reduce the variance error of a model [35]. It is usually used with decision tree but it can be used with other methods as well.

It has been proven successful in many different applications [87, 4] including systems security [90, 67] and it has been used in one of the algorithms analysed in my case studies by Mansour et al. [3] (§ 3.4). The following section explains the classification tree algorithm and how it is used in conjunction with gradient boosting.

Classification Tree. The idea of this algorithm is to divide or split the predictor space in a number of simple regions. In each of those regions the same rule is applied, i.e., the same prediction will be given in output inside of each region. The simple regions are created during the creation of this model (training phase) where the best variable and split-point will be chosen in a recursive fashion until a certain criteria is met.

For example, if we consider a simple model with only one predictor, if X is divided at the point t then the regions $X \leq t$ and $X > t$ will output respectively the same prediction.

In a more general and formal way we can define this concept. Given the N examples $(x_1, y_1), (x_2, y_2), \dots, (x_i, y_i)$ with p predictors $x_i = (x_{i1}, x_{i2}, \dots, x_{ip})$ the tree algorithm will produce M regions of space $R_1, R_2, \dots, R_M$ where a the response is a constant c_m. The function is therefore defined as:

$$\hat{f}(x) = \sum_{m=1}^{M} c_m I(x \in R_m) \tag{2.15}$$

Here, in a region R_m with N_m observations, we have that:

$$p_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} I(y_i = k) \tag{2.16}$$

is the proportion of class k in a region R_m. Therefore we can estimate the classification $\hat{k}_m$ as the majority class as:

$$\hat{k}_m = \arg\ \max_m p_{mk} \tag{2.17}$$

Finding the best possible split in generally computationally infeasible, therefore greedy algorithms are usually used. Following a greedy algorithm, the best possible split is found for each predictor and recursively the algorithm continues with new regions created. This is computationally feasible but the details of such operations are beyond the scope of this thesis. At this point, it is important to define the stopping criteria for the algorithm, i.e., when the algorithm will stop to split the newly created regions. Popular choices for this criterion are:

- minimum misclassification error: a number from Eq. 2.16 below which if reached will stop the split
- minimum error decrease: to stop after the algorithm find *difficult* to split a region with meaningful decrease of misclassification rate
- minimum number of examples in the region
- maximum number of splits

Instead of the misclassification error, other measures of error are possible such as Gini Index [43] or Cross-entropy [35].

Once the stopping criteria is reached the model is created and it can be used to predict the label of a new observation.

The Algorithm. The idea is to create the model by sequentially combining previous weak learners as input and assigning a weight to each of them in order to obtain the final prediction model that will be used during the testing phase. Weak learners are prediction models that are only slightly correlated with the true outcome (while strong learners are well correlated with the true outcome).

This technique is similar to bagging (or bootstrap aggregation), and random forest because they are ensemble learning techniques that try to combine *weak* learners to build a *strong* learner in order to reduce the variance.

In bagging, different learners are created in parallel from k subsets of the training samples (using random sampling with replacement) and then averaged to create the final model.

In random forest, the model is created by combining together different decision tress. Each of them is created using a subset $\mathcal{B}$ of the examples in the dataset $\mathcal{D}$ and only a subset of m predictor from the total predictors p. The goal is to decorrelate the trees to obtain a strong predictor. A typical value for m is $\sqrt{p}$ but the best value depends on the data.

2.2.4 Conformal Predictor

Conformal Predictor (CP) [104] is a supervised algorithm used for classification that gives a prediction with additional information regarding the level of trust on the prediction itself. CP has been used as the base algorithm in my research to build Conformal Evaluator (see § 3).

Intuition. The classification choice is based on *credibility*, a quality measure which gives valuable insights by telling how much the new object is credited with each label. The algorithm will output the label associated with the highest *credibility*. Along with *credibility*, CP provides another metric called *confidence*. It represents how the chosen label is distinguishable from other labels.

At the core of conformal predictor there is a function that measures the similarity between a group of objects and a single target object called *non-conformity measure* (NCM). This is quite a generic definition that allow CP to use many NCMs such as Random Forest (§ 2.2.3), Support Vector Machine (§ 2.2.2), Neural Network [13], K-Nearest Neighbour [18] and many others.

In my research work (Part II), I have adapted CP and used it to assess the quality of malware identification approaches.

Conformal Predictor. I am now going to explain more formally the conformal predictor algorithm. Given the infinite examples

$$(x_1, y_1), (x_2, y_2), (x_3, y_3), \dots \tag{2.18}$$

The observations $x_1, x_2, x_3, \dots$ belongs to the *object space* X while the labels $y_1, y_2, y_3, \dots$ belongs to the *label space* Y. A pair (x_i, y_i) represents an example z_i. A sequence z_1, z_2, z_3 we assume to be drawn from an i.i.d. (independent and identically distributed) population and exchangeable. Being exchangeable refers to the probability distribution that does not change when the positions in the sequence change.

Together X and Y make the example space:

$$Z := X \times Y \tag{2.19}$$

To define the non-conformity measure, I need to introduce the concept of *bag*. A bag is a collection of $n \in \mathbb{N}$ elements where there can be identical elements and where the order is irrelevant. An extreme case is when the bag $(z_1, z_2, \dots, z_n)$ is made of elements that are all the same because the definition allows it. I now define Z^n as the set of all possible bags of size n and Z^* as the set of all bags of Z.

I can now formally introduce the definition of non-conformity measure as:

$$\text{NCM} : Z^* \times Z \to \mathbb{R} \tag{2.20}$$

Therefore, considering a bag with a finite number of element n such as $z_1, z_2, \dots, z_n$ a non-conformity measure it is defined as:

$$\text{NCM}_n : Z^n \times Z \to \mathbb{R} \tag{2.21}$$

If a particular NCM is a *good* measure of non-conformity is subject to the specific application, but the definition is general enough to allow many different non-conformity measures.

It is possible now to compute a non-conformity score between a bag and an element i of the bag as

$$\alpha_i := \text{NCM}_n((z_1, \dots, z_{i-1}, z_{i+1}, \dots, z_n), x_i) \tag{2.22}$$

Alone α_i is telling how much the chosen NCM find different x_i from the bag of objects.

Non-conformity Measure for Security. In this section, I discuss how the non-conformity measure concept applies to a security algorithm.

Many algorithms for classification are based on inner machine-learning techniques or *scoring algorithm* that given a training set of examples $\mathcal{Z}$ and a test object z^* will output a *prediction score* $A(\mathcal{Z}, z^*)$. 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 examples 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 examples. Hence, if we take for example two objects, z_1 and z_2, and a group of objects $\mathcal{Z}$, z_1 is more dissimilar than z_2 to $\mathcal{Z}$ if $A(\mathcal{Z}, z_1) > A(\mathcal{Z}, z_2)$. 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 $\mathcal{Z}$ (i.e., a similarity function), we can easily convert this to a non-conformity measure by changing its sign. Thus, conformal evaluation is agnostic to the algorithm, making it versatile and compatible with multiple ML algorithms; it can be applied on top of any classification or clustering algorithm that uses a score for prediction, even the one that are not directly based on a known ML algorithm.

To give an example, Rieck et al. in [82] 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. For example, in § 3.4.2, we will use the scoring mechanism of BotFinder [102] as a conformity measure. This algorithm is a good example of conformity measure because it gives as output a score that is higher when a new malware sample is more similar to one family. To convert this algorithm to be used into the Conformal Evaluator, we just need to change the sign to the score returned by BotFinder.

Some ML algorithms already have built-in quality measures (e.g., the distance of a sample from the hyperplane in SVM). However, these are algorithm specific and cannot be directly compared with other algorithms. On the other hand, Transcend unifies such quality measures through uniform treatment of non-conformity in an algorithm-agnostic manner. Moreover, algorithms that do not inherently have such quality metrics, (e.g., custom algorithms), benefit even more from conformal evaluator as it defines the quality measure for them.

P-value. The algorithm now computes $\alpha_j \ \forall \ i \neq j$, i.e., all the non-conformity measurements for the other elements in the bag (when the element is included in the bag). This operation wants to understand how well the element will fit with the bag, where it will be included. Once all the α_j are computed, it is possible to calculate a *p-value* for the observation x_i by comparing α_i with the other α_j with $j \neq i$ as:

$$\frac{|\{j = 1, \dots, n : \alpha_j \geq \alpha_i\}|}{n} \tag{2.23}$$

It can go ideally from 1 to $1/n$. It is small when x_i is very different from the other elements (e.g.: an outlier) or it very high when x_i is very similar to the other elements.

In the context of classification a bag of objects can represent a class $k \in K$ where K is the set of classes in the dataset. The calculation of a p-value for an observation $x_i \in \mathcal{D}$ respect to all the elements with label k is shown in Algorithm 1.

Algorithm 1: P-value computation for observation x_i and label k

```
Data:   $\mathcal{D} = \{z_1, \ldots, z_{i-1}, z_{i+1}, \ldots, z_n\}$
        non-conformity measure NCM
        observation $x_i$
Result: p-value of $x_i$ for label $k$

1 Set $z_i = (x_i, k)$
2 for $j \leftarrow 1$ to $n$ do
3     $\alpha_j \leftarrow \mathrm{NCM}(\mathcal{D} \setminus z_j, z_j)$
4 end
5 $\tau = U(0,1)$
6 $p_i^k = \frac{|\{j:\alpha_j>\alpha_i\}|+|\{i:\alpha_j=\alpha_i\}|\tau}{|\mathcal{D}|}$
7 return $p_i^k$
```

Calculating the p-value of an observation x_i of the training dataset $\mathcal{D}_{tr}$ can be interesting to evaluate the fit of that observation on its own class of k. Nevertheless it is known that the observation x_i has the label y_i because it was chosen from the training observations. It will be more interesting to evaluate the fit of an element x_i to a class different than its own y_i.

A variant of Eq. 2.23 can be defined as follow:

$$\frac{|\{j = 1, \ldots, n : \alpha_j \geq \alpha_j\}| + \tau|\{j = 1, \ldots, n : \alpha_j = \alpha_j\}|}{n} \tag{2.24}$$

Eq. 2.24 represents a *smoother p-value* [104]. In Eq. 2.23, the τ is a smoothing factor applied only to those objects that are equal to α_i. This is a mathematical trick used to make the equation differentiable and to smooth down the importance of objects that are on the decision border, i.e., the objects that have equal α_i. Pragmatically, Eq. 2.23 and Eq. 2.24 are very similar therefore I will refer to p-value even if a smoothed p-value is used.

The p-value terminology comes from the statistical hypothesis testing. Here, the null hypothesis H_0, represents the support that the an object belongs to a label k. The alternative hypothesis is that the object does not belong k. Nevertheless, differently to the hypothesis testing, the H_0 is not rejected according to different levels of confidence (typically 90%, 95%, 98%, 99%) but its value will be used by the CP algorithm.

Label Conditional vs Non Label Conditional. In Algorithm 1, line 6, the p-value is calculated over the size of $\mathcal{D}$. This means that all the element in the dataset will be considered hence there will be an $\alpha_i \quad \forall i = 1, \ldots, n$. This is called *non label-conditional* p-value because for all the elements of the dataset there will be one α calculated for it. There is another variant of p-values called *label conditional* where for the p-value for class k only the elements of class k are considered and only on those there will be a α computation. The label conditional and non label conditional algorithms are displayed respectively in Algorithm 3 and Algorithm 2.

Algorithm 2: Non label-conditional p-value

```
2 for $j \leftarrow 1$ to $n$ do
3     $\alpha_j \leftarrow \mathrm{NCM}(\mathcal{D} \setminus z_j, z_j)$
4 end
5 $\tau = U(0,1)$
6 $p_i^k = \frac{|\{j:\alpha_j > \alpha_i\}| + |\{i:\alpha_j = \alpha_i\}|\tau}{|\mathcal{D}|}$
```

Algorithm 3: Label-conditional p-value

```
2 for $j \leftarrow 1$ to $n_k$ do
3     $\alpha_j \leftarrow \mathrm{NCM}(\mathcal{D} \setminus z_j, z_j)$
4 end
5 $\tau = U(0,1)$
6 $p_i^k = \frac{|\{j:\alpha_j > \alpha_i\}| + |\{i:\alpha_j = \alpha_i\}|\tau}{|\mathcal{D}^k|}$
```

P-values vs. Probabilities One might question the utility of p-value over probability of a test object belonging to a particular class. Probabilities are computed by most learning algorithms as qualitative feedback for a decision. SVM uses Platt's scaling to compute probability scores of an object belonging to a class while a random forest averages the decision of individual trees to reach a final prediction [16]. In this section, we discuss the shortcomings of using probabilities for decision assessment as shown in § 4.3 and § 4.4. Additionally, we also provide empirical evidence in favour of p-values as a building block for decision assessment.

P-values offer a significant advantage over probabilities when used for decision assessment. Let us assume that the test object z^* has p-values of $p^1_{z^*}, p^2_{z^*} \cdots p^k_{z^*}$ and probability of $r^1_{z^*}, r^2_{z^*} \cdots r^k_{z^*}$ of belonging to classes $k_1, k_2 \cdots k_n$ (which is the set of all classes in $\mathcal{K}$). In the case of probabilities, $\Sigma_i r^i_{z^*}$ must sum to 1.0. Now, let's consider a 2-class problem. If z^* does not belong to either of the classes, and the algorithm computes a low probability score $r^1_{z^*} \sim 0.0$, then $r^2_{z^*}$ would artificially tend to 1.0. In other words, if we use probabilities for decision assessment it is likely that we might reach an incorrect conclusion for previously unseen samples. P-values on the other hand are not constrained by such limitations. It is possible for both $p^1_{z^*}$ and $p^2_{z^*}$ to be a low value for the case of a previously unseen sample. This is true also when p-values are built using probability as NCM. To calculate the probability of a test sample, only information belonging to the test samples are used (e.g., distance to the hyperplane in the case SVM or ratio of decisions for one class in the case of random forest). Instead, a p-value is computed comparing the scores of all the samples in a class.

I will further elaborate on this in Transcend, § 4, by training an SVM classifier with objects from a dataset and testing it using objects from a different dataset (see § 4.2). During the training process I will calculate a threshold that will applied during the testing phase to distinguish the classifier decisions with enough trust on the decision itself. Fig. 4.1 shows the average of F1-score for malicious and benign classes after the application of the threshold for the objects that fall above (Fig. 4.1b) and below it (Fig. 4.1d). Fig. 4.1 also shows the number of objects with enough trust (Fig. 4.1a) and without (Fig. 4.1c). Fig. 4.1b shows that the use of p-values produces better performance as it identifies more objects to reject than probabilities (Fig. 4.1a). By testing a different dataset it is correct to be unsure on the decision of a high number of samples if the dataset tested is drifter compared to the one used in the training, such as this one. Trusting too many decisions would degrade the performance of the algorithm (Fig. 4.1d and Fig. 4.1c). I will present the case studies in § 4, which show how to derive it from the training dataset.

Credibility. The process described in Algorithm 1 refers to an element that is in the training dataset $\mathcal{D}_{tr}$. Nevertheless, it is trivial to apply the algorithm to an observation in the testing dataset $\mathcal{D}_{te}$, i.e., an observation where the label is not available.

Conformal predictor iterate the Algorithm 1 over all possible $k \in K$. It will result in K different p-value p_i^k for an element x_i. The output label for the observation x_i is the one corresponding to the highest p-value. In the same way the *credibility* is defined as:

$$\text{credibility} = \max p_i^k \tag{2.25}$$

The *credibility* represents the amount of evidence that the element x_i is associated with the class k. In other words, it measure how well the observation x_i fit in the bag of objects with label k.

Confidence. Another important metric that credibility. It is defined as:

$$\text{credibility} = 1 - \text{confidence} \tag{2.26}$$

It represents the amount of evidence that the element x_i is associated with classes different than k. In other words, it represents how well the observation x_i is distinguishable from the other classes.

Credibility and confidence together can create 4 different situations for an observation x_i:

- Low credibility, low confidence: the *worst* situation, i.e., the observation is poorly associated with the predicted label. There are other labels that can be similarly associated with that observation.
- Low credibility, high confidence: the predicted label is poorly associated with the observation but there are no other candidate labels that seems to be a better choice.
- High credibility, low confidence: the observation is highly associated with the predicted label but there are other candidate labels that seem to be a good choice for that observation.
- High credibility, high confidence: the *best* situation, i.e., the observation is highly associated with the predicted label and there are no other label that can seem to be a good choice for that observation.

Prediction Set. Another important property of CP is the possibility to output multiple labels as prediction, i.e., prediction set. Given a significance level ϵ and an object x_i, CP will output all the labels y_i associated to a p-value (computed by Eq. 2.23 or Eq. 2.24) greater than ϵ. CP has the property of *validity* that means that the probability that the true label not being part of the output set is exactly ϵ. This property has been explored by the security community in [21] and it will be explored in the future works.

2.3 Evaluation Metrics

This section will describe the metrics used to evaluate the efficiency of a predictive system. There are many metrics that can be used to highlight different functionality of an algorithm. In this section, I will explain the ones that will be used in Part II of this thesis.

The metrics described are computed in relation to the class of interest, i.e., the positive class. In the malware security field, the positive class is usually associated with the malware class, but a different association is possible. This association is arbitrary but needs to be defined to avoid confusion during the analysis of the performance.

Metrics

The performance metrics are computed taking the binary classification scenario as reference. In this scenario, the positive class is associated with the malware class while the negative class is associated with the positive class. Therefore there are 4 possible outcomes for a single observation:

		PREDICTION	
		Prediction positive	Prediction negative
GROUND TRUTH	Condition positive	True Positive (tp)	False Negative (fn)
	Condition negative	False Positive (fp)	True Negative (tn)

Table 2.1: Possible outcomes for a binary classification.

- True Positive (tp): the ground truth label of the observation is the positive class and the predicted label is the positive class.
- False Negative (fn): the ground truth label of the observation is the positive class but the predicted label is the negative class (or one of the other classes in a multiclass scenario).
- False Positive (fp): the ground truth label of the observation is the negative class (or one of the other classes in a multiclass scenario) but the predicted label is the positive class.
- True Negative (tn): the ground truth label of the observation is the negative class (or one of the other classes in a multiclass scenario) and the predicted label is the negative class (or one of the other classes in a multiclass scenario, the same one of the ground truth).

The possible outcomes are summarised in Table 2.1.

From Table 2.1, it is possible to derive several performance metrics. All the performance metrics are referred to class of interest i.e., the malware class that is associated with the positive class.

Multiclass case. In a multiclass classification scenario, only the definitions of false positive and false negative change a little, while the metrics definitions remain the same. In fact, they change because the negative class will be associated to a group of classes, i.e., the classes that are not the ones of interest. For example, in a malware family classification problem, where the goal is to classify a binary in the correct malware family, to compute the performance of a class A; the positive class is associated with the class A while the negative class is associated with the classes B, C and D. Similarly, if we want to compute the performance of the class B, it is possible to associate the positive class with B and the negative class with the classes A, C and D.

In a scenario where we have more than one class of interest, it is possible to compute the performance of each class separately and then combine them together. To compute each class of interest separately it is possible to simply associate it with the positive class and compute the performance following the formula accordingly.

The ones that I am going to use throughout my thesis are precision, recall, false positive ratio, F_1 score, accuracy and ROC curve. This is not a comprehensive list. Many others are available and preferred in specific circumstances.

Precision. It is calculated as the ratio between the objects correctly identified (TP) and the total objects assigned with the positive label (prediction positive):

$$precision = \frac{tp}{tp + fp} \tag{2.27}$$

Recall. Also known as True Positive Ratio (TPR), it is the ratio between the objects correctly identified (tp) and the total objects that belong to that class ($tp + fn$):

$$recall = \frac{tp}{tp + fn} \tag{2.28}$$

False Positive Ratio. False Positive Ratio (FPR) represents the number of negative events wrongly categorized as positive:

$$fpr = \frac{fp}{fp + tn} \tag{2.29}$$

$\mathbf{F_1}$ score. It is a measure that combines the precision and recall in a single value:

$$F_1score = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{2.30}$$

Accuracy. It is the proportion of the true classified objects among the total number of objects:

$$Accuracy = \frac{tp + tn}{tp + fp + fn + tn} \tag{2.31}$$

ROC curve. The Receiver Operating Characteristic (ROC) curve is a graphical tool to represent the capability of a system to discriminate 2 classes by varying the thresholds of acceptance. On the x-axis, the FPR are shown while on the y-axis the TPR are displayed. In this way, in a binary system a sample does not belong to a class if the probability is greater than 50% but this threshold can vary. For each calculation of the threshold a new point (with coordinate FPR, TPR) is calculated. Varying the thresholds between 0% and 100% will produce a complete ROC curve.

Despite its popularity, the ROC curve have been shown to be a measure that can be tricked [22, 34]. For this reason a more suitable metric has been proposed:

precision-recall curve (PRC). PRC is similar to ROC where on the x-axis you will have the precision and on the y-axis the recall.

Aggregated Metrics

Often a single value that summarise the precision, recall or F_1 of the system is more effective to describe the system rather than having a metric for each class. To aggregate the values for metrics discussed there are two main ways: macro and micro average [93].

Macro Average. The first method consists in averaging the single values for each class of objects that the system is considering. For example, having a system with n classes with given precision $precision_1, precision_2, \ldots, precision_n$, the overall precision can be computed as:

$$precision = \sum_{i}^{n} \frac{precision_i}{n} \tag{2.32}$$

The aggregated performance for the other metrics can be computed with the same principle.

Micro Average. The second method consists of calculating the metric by recalculating the formulas. For example, in a system with n classes, the aggregated overall precision can be computed as:

$$precision = \frac{\sum_i^n tp_i}{\sum_i^n tp_i + fp_i} \tag{2.33}$$

These two methods can be used arbitrarily depending on the goal of the analysis but they need to be made explicit to avoid confusion. Both of the methods have drawbacks that need to be considered. The macro-average treats each class equally regardless of the number of samples in each class. Usually, this average is used when small classes are equally important as the bigger ones. In this case, if a small class is performing particularly badly, it will influence the overall performance significantly. The micro-average instead takes into account the number of samples in each class therefore bigger classes will influence the overall performance substantially. The choice of the type of average is dependent on the context and both of them can be preferred in a particular scenario.

2.4 Security Considerations

Machine learning has become a standard tool for malware research in the academic security community. It has been used in a wide range of domains including Windows malware [20, 100, 65], PDF malware [52, 63], malicious URLs [96, 54], Javascript malicious code [84, 19], and Android malware [9, 98, 64, 29, 113, 21, 111]. However, there are several domain-specific issues that have not been addressed properly leading to show tantalizingly high performances like if malware would be a problem of the past [76]. Among the most common problems are temporally inconsistent train and test splits, e.g., k-fold cross-validation. Malware classifiers are strongly affected by *concept drift*: as new malware variants and families appear, their performances decays over time [46]. Therefore, when temporally inconsistent experimental settings allow the classifier to train on what is effectively future malware, it will artificially inflate the test results [5, 69].

In this section, I highlight specific constraints and considerations that must be taken into consideration when planning experiments in a systems security domain, as shown in [76].

Domain-specific in-the-wild malware percentage. Estimating testing dataset distribution is fundamental for enabling unbiased evaluations of malware classifiers. This distribution needs to rely on measurement studies performed by industry players, measurement papers, and tech reports, to provide a realistic estimate for the correct setting of the targeted scenario. It is not always easy to find such studies but it is important to foster such studies to promote sound evaluations.

In the Android landscape, malware represents 6%–18.8% of all the apps, according to different sources. A key industrial player in the Android landscape reported the ratio as approximately 6%, whereas the AndRadar measurement study [59] reports around 8% of Android malware in the wild. The 2017 Google's Android security report [33] suggests 6–10% malware, whereas an analysis of the metadata of the AndroZoo dataset [6] counting almost 6M Android apps regularly updated, reveals an incidence of 18.8% of malicious apps. Correctly estimating the malware percentage in the testing dataset is a challenging task and need to be encouraged further as representative measurement studies [59, 105] and data sharing to obtain realistic experimental settings. This will promote the unbiased evaluation of machine learning-based malware classification techniques.

Performance bias increases when it increases the discrepancy between the percentage of malware in the wild and the percentage of malware tested during the study. Therefore, it is important that the value of this performance remains in the same ballpark of the real one. In case the dataset used in the study is not able to represent correctly the proportion seen in the wild, there are many techniques that can be adopted for this purpose (e.g., under-sampling, over-sampling) each of them has its own advantages and drawbacks.

As discussed in [76], the percentage of malware in the testing distribution needs to be estimated and *cannot* be changed, if one wants results to be representative of in-the-wild deployment of the malware classifier.

Temporal Consistency. It refers to temporally inconsistent evaluations, which integrates future knowledge about the testing objects into the training phase [5, 69]. This problem is exacerbated by families of closely related malware, where including even one variant in the training set may allow the algorithm to identify many variants in the testing.

The problem of temporal consistency between training and testing is known but so far has not been addressed conclusively in previous studies [5, 69].

The authors of [76] address this problem by enforcing a constraint. All the objects in the training must be *strictly* temporally precedent to the testing ones:

$$\mathit{time}(x_i) < \mathit{time}(x_j), \forall x_i \in \mathcal{D}_{tr}, \forall x_j \in \mathcal{D}_{te} \tag{2.34}$$

where x_i (resp. x_j) is an object in the train set $\mathcal{D}_{tr}$ (resp. testing set $\mathcal{D}_{te}$). Eq. 2.34 must hold; its violation inflates the results by including future knowledge in the classifier thus biasing the results.

The time information can have different meanings depending on the context of the application. In malware analysis scenario, it may represent the first time that a binary was observed in the wild. In URL analysis, it may represent the server's arrival time of the query.

Despite the variety of meaning the time can assume, the train-test temporal split must be enforced because in a real-world deployment of the algorithm because it is not possible to have a test set that temporarily proceed the train set.

Concept Drift. Cross-validation can cause positive bias in the model by artificially inflating the performance of malware classifiers [5, 69, 70]. The bias can be caused by concept drift in malware combined with the similarities among malware within the same family. This is because CV is likely to include in the train set at least one sample of each malware family in the dataset, whereas new families will be unknown at training time in a real-world deployment. I would like to stress that I am not claiming that 10-fold CV is wrong, and it is still very important to verify that the algorithm is not overfitting [12]. Nevertheless, in highly non-stationary contexts such as malware classification it is *not sufficient*—instead, CV results are misleading and positively inflated, because training samples are not temporally precedent to testing samples. Therefore, knowledge about the *future* malware (e.g., future families) are included in the model [5, 69] and hence the algorithm will able to recognize variants.

Despite the care that need to be adopted when using CV, its use is widespread in malware classification research [70, 80, 98, 61, 20, 100, 65, 52, 113, 19]; while a common mechanism to prevent overfitting [12], it is ill-suited for estimating the real-world performance of machine learning techniques with non-stationary data (e.g., malware) that are affected by concept drift and time decay.

Although this characteristic is important, a high performance is expected from a classifier in a *static* scenario where new malware fit in the training distribution.

In such a scenario, it is possible to estimate the performance of your classifier with CV, whether objects are old or new. Nevertheless, in the presence of concept drift, the testing distribution will be statistically different than the training distribution and therefore the CV is going to be misleading to evaluate the performance of the classifier.

We thus believe that experiments on the performance achieved on the detection of past malware can be misleading; the community should focus on building malware classifiers that are robust against time decay.

Part II

Research Work

Chapter 3

Conformal Evaluator

Malware pose a serious and challenging threat across the Internet. According to different antivirus firms [66, 99, 48], the number of new unique malware in 2018 was between 500M and 700M. With such numbers, manual analysis techniques cannot be employed to scale up and the need for automated approaches has become rapidly clear. Machine learning has long been acknowledged as a promising technique to identify and classify malware threats [86, 83, 88]; 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 artefacts, 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, *conformal evaluator* was developed: a framework designed for 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, the 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 this claim, the outcome of three different algorithms are evaluated under conformal evaluator settings (§ 3.4). My aim is to 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. Building on top of such metrics, *Transcend* was proposed: a framework to identify ageing classification models *in vivo* during deployment, much before the machine learning model's performance starts to degrade. This is a significant departure from conventional approaches that retrain ageing models retrospectively when poor performance is observed. The approach uses a statistical comparison of samples seen during deployment with those used to train the model, thereby building metrics for prediction quality. I will show how Transcend can be used to identify concept drift based on two separate case studies (§ 4.2) on Android and Windows malware, raising a red flag before the model starts making consistently poor decisions due to out-of-date training.

3.1 Methodology

Conformal evaluator (CE) is the evaluation framework built on top of conformal predictor [104]. 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 examples $z_1, z_2 \dots z_n \in Z^k$, and a new object z^*, conformal predictor enables us to decide whether $z^* \in k$ or $z^* \notin k$ (for $k \in \mathcal{K}$).

In this framework, CP algorithm is dissected 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, k as a class of objects with n_k elements and $\mathcal{K}$ as the set of all the classes, by iterating the process through every class in $\mathcal{K}$, the algorithm confidence measures how distinguished z^* is with respect to the other classes. Algorithm credibility and algorithm confidence are further explained in § 3.2.1 and § 3.2.2.

Core of conformal evaluator, is a non-conformity measure, a real-valued function $A(Z^k, z^*)$, which tells how different an object z^* is from a set of objects Z^k (i.e., objects with the same label k). 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 [91]) and in general with any method that makes use of real-valued numbers (i.e., a similarity function) to distinguish objects. Such flexibility enables CE framework to assess a wide range of algorithms.

Once a non-conformity measure is extracted from the algorithm under analysis, conformal evaluator computes a p-value p_{z^*} which, in essence for a new object z^*, represents the percentage of objects in the whole dataset that are equally or more estranged to Z^k as z^*. The basic algorithm is shown in Listing 1 where the p-value is calculated from an element in the same dataset $\mathcal{D}$, therefore to compute the p-value for a new element we need to provisionally include the element z^* in $\mathcal{D}$ i.e. $\mathcal{D}^* = \mathcal{D} \cup z^*$.

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

3.1.1 Relationship with Conformal Predictor

Although conformal evaluator is built on top of conformal predictor (CP), it does not share the same weaknesses as that of other solutions based on it [26, 36]. Fern and Dietterich[1] also showed how pragmatically CP is unsuited in an open world scenario (i.e., when new labels are present in the testing set). They suggest to apply an anomaly detector algorithm before the use of CP to feed it with normal data. We further highlight the differences between CP and CE that makes CE better-suited to the concept drift detection task.

They say that CP in unsuited in the Open World scenario (i.e., Unknown Unknowns, i.e., when there is an unknown label in the testing set) using traditional ML classifier, therefore they suggest to use an anomaly detector before the use of CP in order to feed CP with normal data (i.e., the anomaly are removed). They do not specifically talk about i.i.d. but it might be an inner cause.

CP relies on a non-conformity measure to compute p-values in a way similar to CE. For each classification task, CP builds on such p-values to introduce credibility—the class, in a classification problem, with the highest p-value and confidence—defined as one minus the class with the second highest p-value (these metrics are different from CE metrics, see § 3.2). The CP algorithm then outputs either a single class prediction with the identified credibility and confidence (or a prediction set).

CE dissects CP metrics and to extract its p-values calculation. The p-values are used together with the output labels provided by the algorithm under evaluation, to build CE metrics. CP ignores these labels as it tries to predict them. Conversely, CE uses this information to provide quality metrics to assess the quality of the encapsulated algorithm. This change is of paramount importance to derive the thresholds, computed by Transcend (§ 4), used to accept or reject a prediction and to assess an algorithm (§ 3.3).

This *posterior* use of the labels is a key feature that enables CE to detect concept drift. On the contrary, CP is designed as a predictive Transcend making only use of *prior* information. Since labels are important pieces of information, CE uses them to build its metrics and assessments (see, § 3.2 and § 3.3). The labels used by CE are the ones of the training samples and not the labels of the testing samples that

[1]A. Fern and T. Dietterich. "Toward Explainable Uncertainty". https://intelligence.org/files/csrbai/fern-slides-1.pdf

are unavailable at the time of classification.

CP exists in two modes: transductive and inductive [27]. The transductive mode is based a straightforward comparison of a new examples to the training ones by relative strangeness. In the inductive mode, a training set is divided into two parts —proper train and calibration set—, where the proper train set is used only to assess the strangeness of the remaining (calibration and testing) examples, but its examples are not assessed by NCM scores themselves. The inductive mode was originally invented to increase the computational velocity. But it also has an advantage in the context of "posterior" learning, simulating a division into train and test sets within the original train set, and therefore being more sensitive to over-fitted models. On the other hand, we would like to cover the case when the amount of examples may be relatively small, therefore we would not like to lose non-conformity score assigned to the examples from the proper training set. Therefore, we create CE that is a novel mix of transductive and inductive modes: the training set is divided into two sets, then the first of them gets NCM scores as in the transductive mode and the second follows the inductive scheme. This affects the validity of results to a very slight degree, which is negligible in the context of "posterior" evaluation, where the methods are compared to each other. Nevertheless, the mathematical property and the fundamental ground of CP ([91]) are intact. For these reasons, the degree of novelty introduced by CE makes it a different machine learning algorithm, even though, still based on CP.

3.2 Statistical Measures

In this section, I am going to explain the statistical tools that form the base of conformal evaluator: algorithm credibility and algorithm confidence.

At the heart of conformal evaluation is the non-conformity measure—a real-valued function $A(\mathcal{Z}, z)$, which tells how different an object z is from a set $\mathcal{Z}$. The set $\mathcal{Z}$ is a subset of the data space of objects $\mathcal{D}$ (with the same label k). Due to the real-valued range of non-conformity measure, conformal evaluator can be readily used with a variety of machine learning methods such as support-vector machines, neural networks, decision trees and Bayesian prediction [91] and others that use real-valued numbers (i.e., a similarity function) to distinguish objects. Such flexibility enables to assess a wide range of algorithms.

As explained in § 2.2.4, conformal evaluation computes a notion of similarity through p-values. For a set of objects $\mathcal{Z}^k$ (i.e., objects with same label k), the p-value $p^k_{z^*}$ for an object z^* is the proportion of objects in class k that are at least as dissimilar to other objects in $\mathcal{Z}^k$ as z^*. There are two standard techniques to compute the p-values: *Non-Label-Conditional* employed by *decision* and *alpha* assessments outlined in § 3.3.1 and § 3.3.2; and *Label-Conditional* employed by the concept drift detection described in § 2.2.4. The computation of the non-conformity measures non-label-conditional is listed in Algorithm 2 while the label-conditional is listed in Algorithm 3.

P-values compute an algorithm's credibility and confidence, crucial for decision assessments (§ 3.3).

3.2.1 Algorithm Credibility

The first evaluation metric to explore is *algorithm credibility.* It is defined as the p-value corresponding to the label chosen by the algorithm under analysis. This is an important difference with respect to CP. CE receives the label provided by the algorithm under evaluation to assess it. CP instead, it is a predictive algorithm and it provides to the sample the label associated to the maximum p-value. These two labels are generally different.

P-values are calculated according to Eq. 2.24. Therefore, algorithm credibility is defined as:

$$A_{cred} = p^k \ ; \ k \ \text{ is the label outcome of the algorithm} \tag{3.1}$$

As stated earlier, the p-value measures the fraction of objects within $\mathcal{D}$, that are at least as different from a class k as the new object z^*. Therefore a high credibility value means that z^* is very similar to the objects of class k.

There may potentially be high p-values for multiple labels indicating multiple matching labels for the test object which the classification algorithm has ignored. On the other hand, a low credibility value is an indicator of either z^* being very different from the objects in the class chosen by the classifier or the object being poorly identified. This observation alone can already be considered 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 k, yet this could also reveal that the object is poorly identified. These two observations show that credibility alone is not sufficient for reliable decision assessment. Hence, we introduce another measure to gauge the non-performance of the classification algorithm—*algorithm confidence.*

3.2.2 Algorithm Confidence

Algorithm confidence is the second evaluation metric to explore. For a given choice (e.g., assigning z to a class k_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 $z^* \in k_i$ is distinguishable from other classes k_j with $j \neq i$.

The definition of the algorithm confidence is one minus the maximum p-value among all p-values except the *p-value* chosen by the algorithm (i.e., algorithm credibility):

$$A_{conf} = 1 - max(\mathbb{P} \setminus A_{cred}) \tag{3.2}$$

where

$$\mathbb{P} = \{p^k : k \in \mathcal{K}\} \tag{3.3}$$

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 my setting, where CE rethinks 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 (as described in § 3.1.1). 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 [56]. This means that conformal evaluator findings are more robust to dataset changes than other evaluation methods.

CE can also provide quality evaluation that allows switching the underlying ML-based process to a more computationally intensive one on classes with poor confidence [21]. Our work details the CE metrics used by Dash et al. [21] and extends it to identify concept drift.

3.3 Assessments

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

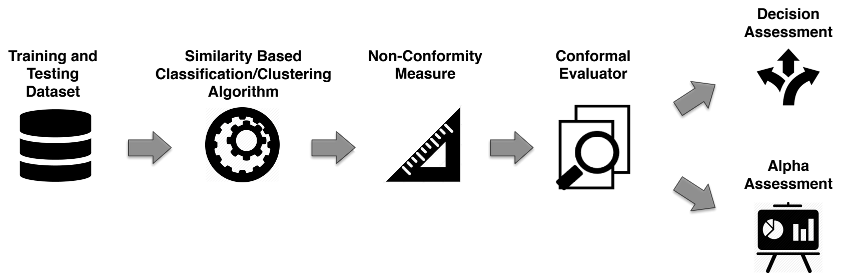

Training and Testing Dataset
Similarity Based Classification/Clustering Algorithm
Non-Conformity Measure
Conformal Evaluator
Decision Assessment
Alpha Assessment

Figure 3.1: 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 chosen separately. Alpha assessment is used to assess how good is the similarity function with respect to the underlying dataset.

The framework is shown in Fig. 3.1. From the similarity-based classification or 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.

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

Decision assessment 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* gives an indication of how good the non-conformity measure is in respect to the dataset. It provides more profound and trustworthy insights on how good (or bad) the algorithm is 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. I call this *alpha assessment*, as often it refers to the non-conformity score for object z_j as α_j.

3.3.1 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 $k \in \mathcal{K}$ 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 measures are then grouped into two sets —correct or wrong— which represents values for correctly and wrongly classified objects. Subsequently, values are averaged and their standard deviation is also computed, this is done for every class $k \in \mathcal{K}$, to study whether the algorithm works consistently for all classes or if there are difficult classes that the algorithm has trouble dealing with. This assessment, performed during the design phase of the algorithm, will also be used in Transcend (§ 4) to help to decide the cut-off threshold for a deployed scenario to separate the samples with enough statistical evidence of correctness.

Comparing the results obtained for correct and wrong choices produces interesting results. For correct choices it would be desirable to have high credibility and confidence. Conversely, for wrong choices it would be desirable to have low credibility and high confidence. The divergence from these scenarios helps understand whether the algorithm takes strong decisions, meaning that there is a strong statistical evidence to confirm its decisions, or, in contrast, if the decisions taken are easily modified with a minimal modification of the underlying data.

By looking at the outcome of decision assessment, it is possible to understand whether the choices made by an algorithm are supported with statistical evidence. Otherwise, it is possible to get 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.

3.3.2 Alpha Assessment

In addition to the decision assessment, which evaluates the output of a similarity-based classification/clustering algorithm, another important step in understanding the inner workings and subtleties of the algorithm includes analysing the data distribution of the algorithm under evaluation. Owing mainly to practical reasons, malware similarity-based algorithms are developed around a specific dataset. Hence there is often the possibility of the algorithm to over-fit its predictions to the dataset. Over-fitting results in poor performance when the algorithm analyses new or unknown datasets [56]. Despite employing techniques to avoid over-fitting, the best way to answer this question is to try the algorithm against as many datasets as possible. We show that conformal evaluator can help solve this problem, when no more than one dataset is available.

The alpha assessment analysis takes into account how appropriate is the similarity-based algorithm when applied to a dataset. It can detect if the final algorithm results still suffer from over-fitting issues despite the efforts of minimizing it using common and well-known techniques (e.g., cross validation).

Furthermore, the assessment enables us to get insights on classes (e.g., malware families), highlighting how the similarity-based method works against them. Researchers may gather new insights on the peculiarities of each class, which may eventually help to improve feature engineering and the algorithm's performance, overall.

First, for each object $z_j \in \mathcal{D}$, where k_j is z_j's true class, we compute its p-values against every possible $k \in \mathcal{K}$. We then plot the boxplot [39], containing the p-values for each decision. By aligning these boxplots and grouping them by class/cluster, we can see how much an element of class/cluster j resembles that of another one, allowing for reasoning about the similarity-based algorithm itself.

In § 3.4 three case studies are presented where I have statistically evaluated the quality behind performances of algorithms within the conformal evaluator framework.

3.4 Case studies

To demonstrate conformal evaluator, in this section I evaluate three malware classification algorithms, leveraging the assessments defined in § 3.3:

- **Algorithm 1**: *Drebin* [9], a detection algorithm for malicious android applications.
- **Algorithm 2**: *BotFinder* [102], an algorithm for botnet classification and detection.
- **Algorithm 3**: algorithm used in the Kaggle's Microsoft Malware Classification Challenge [40] 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 of extracting various information from the application manifest and decompiled code. This information consists of the requested permissions, the requested hardware components, the application intents, the type of application components, various

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 3.1: Dataset composition used by [9].

Microsoft Malware Classification Challenge Dataset			
Malware	Samples	Malware	Samples
Ramnit	1 541	Tracur	751
Lollipop	2 478	Obfuscator.ACY	1 228
Kelihos_ver3	2 942	Gatak	1 013
Vundo	475	Kelihos_ver1	398

Table 3.2: Number of samples for each family used in the Microsoft challenge.

types of API calls, the effective used permissions (from the decompiled code) and the network addresses. These features are then used to build a model to decide whether new applications are malicious or not. The detection model is built using a well-known 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 [9] is a public dataset composed of 123 435 benign applications and 5 560 malicious applications (see Table 3.1).

This algorithm was chosen 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 [9].

With respect to *BotFinder* [102], the algorithm processes network traces to build family-based malware behaviour profiles. These are later used for classification of new samples. BotFinder was chosen 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 (on Table 3.3 it is referred as original dataset). Having the same dataset is indeed crucial in order to have a meaningful comparison. As for BotFinder algorithm itself, it was re-implement from scratch according to the

Botfinder Dataset			
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 3.3: Number of samples for each family. Datasets used by [102].

description outlined in [102] that was detailed enough to allow us to achieve 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 γ_M in [102], 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 labelled as belonging to the family with the highest score. This score naturally serves as non-conformity measure by inverting its sign (see § 2.2.4). Before labelling the sample, BotFinder implements a filtering step that relies on an empirical threshold defined through iterative experiments. The assessment was performed omitting this step as it is more interesting to evaluate the scoring function itself.

I am going to apply the selected algorithms to the dataset used in [102] and a perturbed dataset. BotFinder's dataset is composed of 5 malware families, each one having a different number of network traces (see Table 3.3). In addition, 5 new malware families were introduced (referred as perturbed dataset in Table 3.3) 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 3.3).

The third evaluated algorithm comes from the Microsoft Classification Challenge [40] on the Kaggle platform [40]. 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 only 8 families were included in the evaluation, as the excluded one has noticeably less samples than the others (10 to 100 times less).

The algorithm is described in [3]. The authors use the eXtreme Gradient Boosting (XGBoost) as their machine learning classification algorithm [101]. It's based on gradient boosting [81] and, like other boosting techniques, it combines different weak prediction models to create a stronger one (see § 2.2.3). Particularly in their work, the authors use XGBoost with decision trees.

As non-conformity measure, I 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 labelled malware samples whose ground truth has been verified.

In a nutshell, our experiments aim to answer the following research question:

RQ: *What insights do CE statistical metrics provide?* Intuitively, such metrics provide a quantifiable level of quality of the predictions of a classifier.

3.4.1 Evaluation of Algorithm 1

In this section Drebin is evaluated, described in [9]. The algorithm achieves very good performance as shown on the confusion matrix on Table 3.4.

Fig. 3.2 shows the decision assessment (described in § 3.3.1) for [9]. 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 labelled (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 is 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 § 3.3.1).

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" I mean the ideal (or real) border where decisions switch from "A" to "B".

Sample	Assigned label Malicious	Benign	**Recall**
Malicious	5 132	428	0.92
Benign	297	123 156	0.99
Precision	0.95	0.99	

Table 3.4: Confusion matrix for [9] with original dataset.

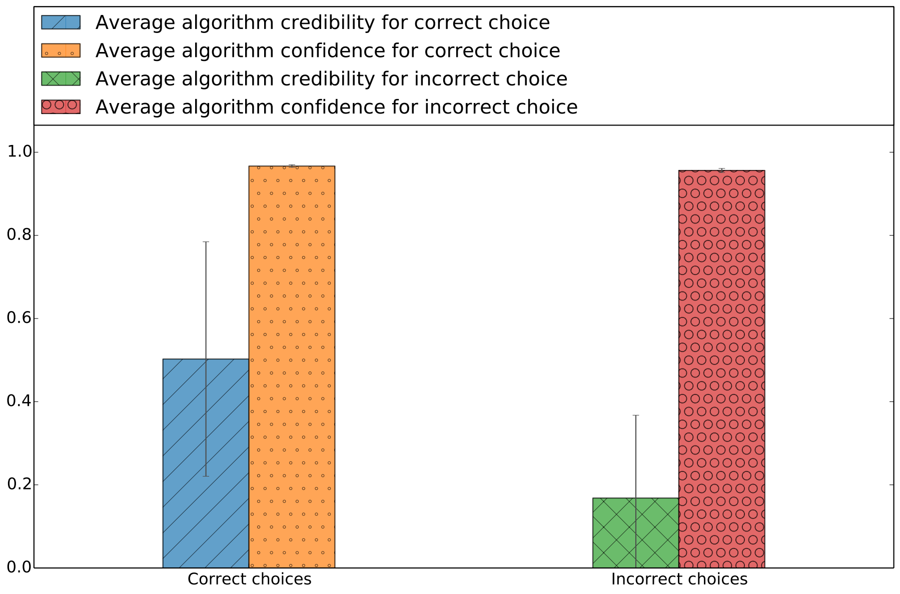


Figure 3.2: Decision assessment for the binary classification case study (Drebin [9]) with the original dataset. Correct predictions are supported by a high average algorithm credibility and confidence, while incorrect ones have a low and a high algorithm credibility and confidence, respectively. Overall, positive results supported by a strong statistical evidence.

Looking thoroughly at [9], 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, Fig. 3.2 gives us more information than pure performance metrics. It is telling us that, based on the quality of the results obtained, we can be 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

Sample	Assigned label		Recall
	Malicious	Benign	
Malicious	2 678	24	0.99
Benign	1	59 999	0.99
Precision	0.99	0.99	

Table 3.5: Confusion matrix for [9] with perturbed dataset.

statement.

Fig. 3.3 shows CE's alpha assessment of Drebin. We plot this assessment as a boxplot to show details of the p-value distribution. The plot shows that the p-value distribution for the wrong predictions (i.e., second and third column) is concentrated in the lower part of the scale (less than 0.1), with a few outliers; this means that, on average, the p-value of the class which is not the correct one, is much lower than the p-value of the correct predictions. Benign samples (third and fourth columns) seem more stable to data variation as the p-values for benign and malicious classes are well separated. Conversely, the p-value distribution of malicious samples (first and second columns) is skewed towards the bottom of the plot; this implies that the decision boundary is loosely defined, which may affect the classifier results in the presence of concept drift. A direct evaluation of the confusion matrix and associated metrics does not provide the ability to see decision boundaries nor predictions (statistical) quality.

Moreover, based on the 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 underlying dataset.

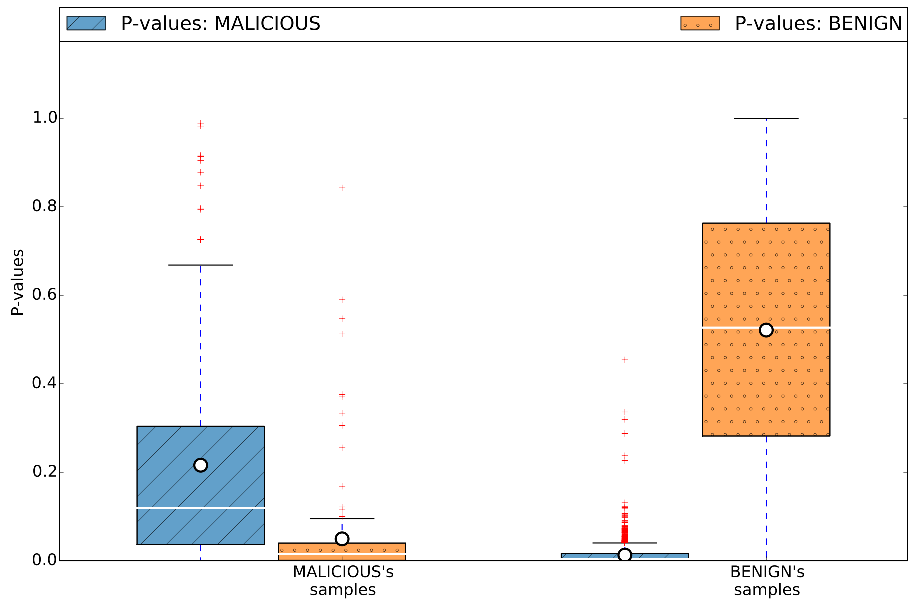


Figure 3.3: Alpha assessment for the binary classification case study (Drebin [9]) with the original dataset. Benign samples are well separated from malicious ones, especially when the assigned label is benign; this provides a clear statistical support that positively affects the quality of predictions.

Results with perturbed dataset

To give credit to the results obtained with the decision assessment, I am going to rerun conformal evaluator with a perturbed dataset to check results consistency against potential customized threshold or previous dataset specific tweaks. The perturbed dataset is described on Table 3.1. Half of the malicious applications come from [110], while the other half were kindly provided by McAfee. The benign applications are a subset of the original Drebin's dataset.

In Fig. 3.4, we can see the results of the decision assessment for the perturbed Drebin's dataset. The assessment shows a similar behaviour as the original approach, meaning that 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 (Fig. 3.2) 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 3.5 shows that the benign samples are again well distinguished from the malicious ones, with similar considerations as before.

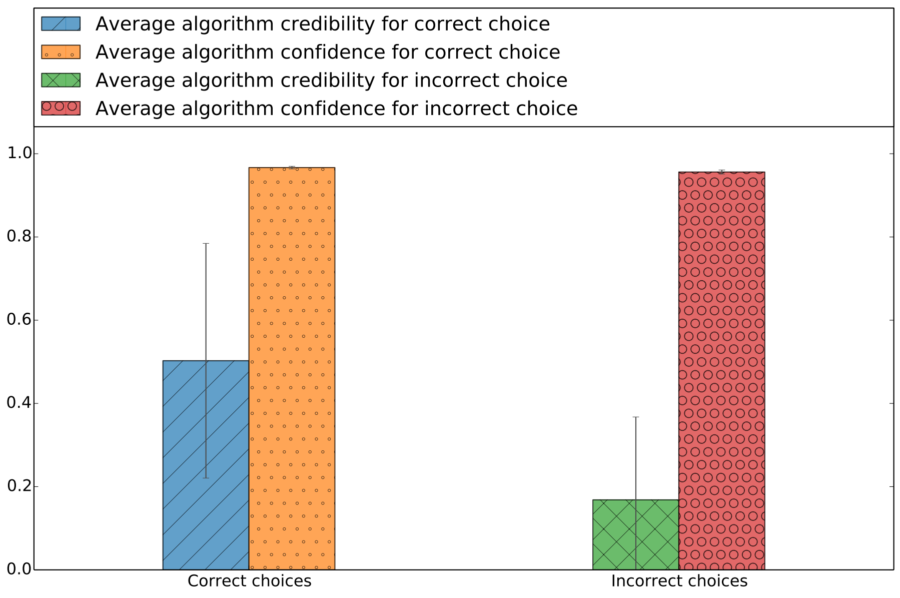


Figure 3.4: Decision assessment for [9] with a perturbed dataset. We have again very good results: correct classifications have a high confidence, while incorrect classifications have low credibility and high confidence.

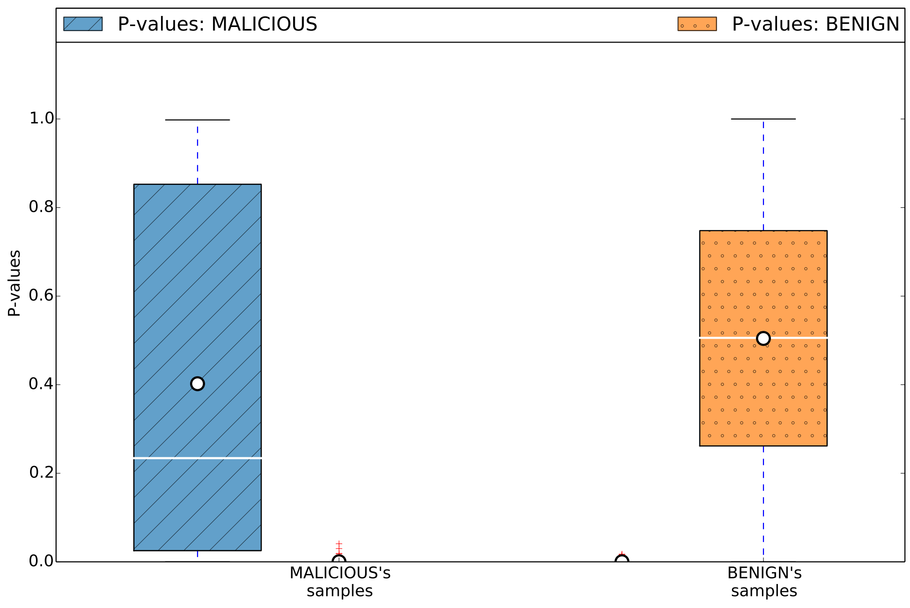


Figure 3.5: Alpha assessment for [9] with the perturbed dataset. Very good results achieved by the algorithm: the p-values for benign samples are well separated. The p-values for malicious are once again not really well separated but the p-values for the benign class are very concentrated in the low part of the scale. White dots represent the average values, white lines represent the median values, and red crosses represent outliers.

3.4.2 Evaluation of Algorithm 2

In this section I am going to evaluate BotFinder, described in [102]. On Table 3.6, 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 (Fig. 3.6) 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, § 3.4.1) 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 Fig. 3.6), 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 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 analyse 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

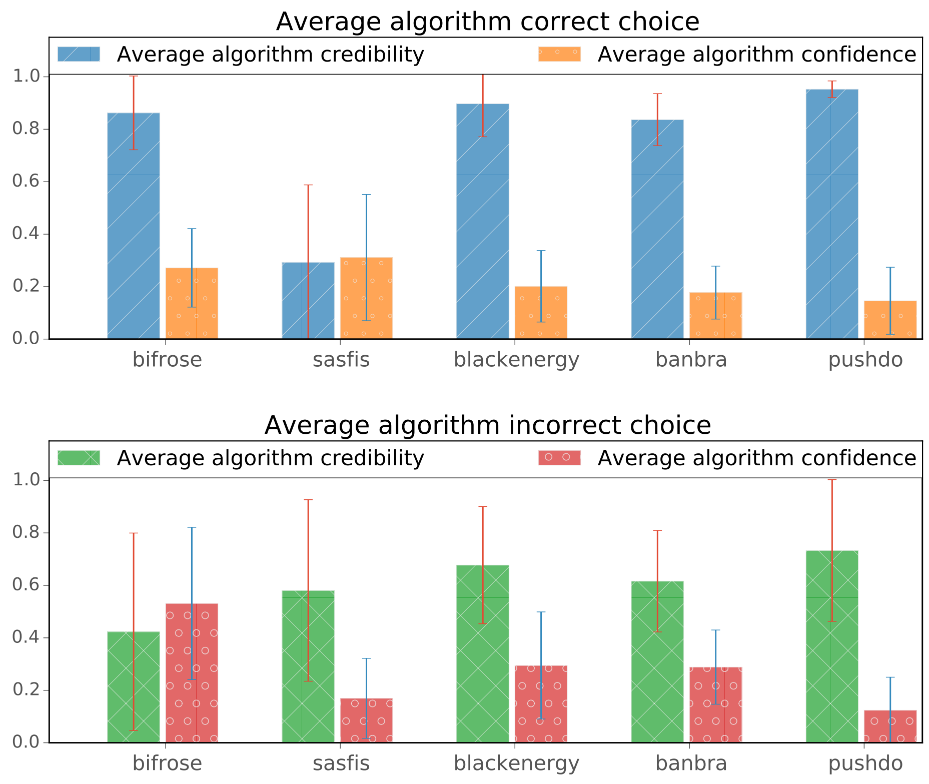


Figure 3.6: Decision assessment for [102] with the original dataset. All the families show a poor separation between correct and incorrect results. This is symptom of weaknesses against dataset modification.

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 [9], the more likely we will observe a poor recall.

With the sole knowledge of Table 3.6, 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, Fig. 3.7, 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 (Fig. 3.7, 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: Bifrose	Sasfis	Blackenergy	Banbra	Pushdo	**Recall**
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 3.6: Confusion matrix for [102] tested on the original dataset.

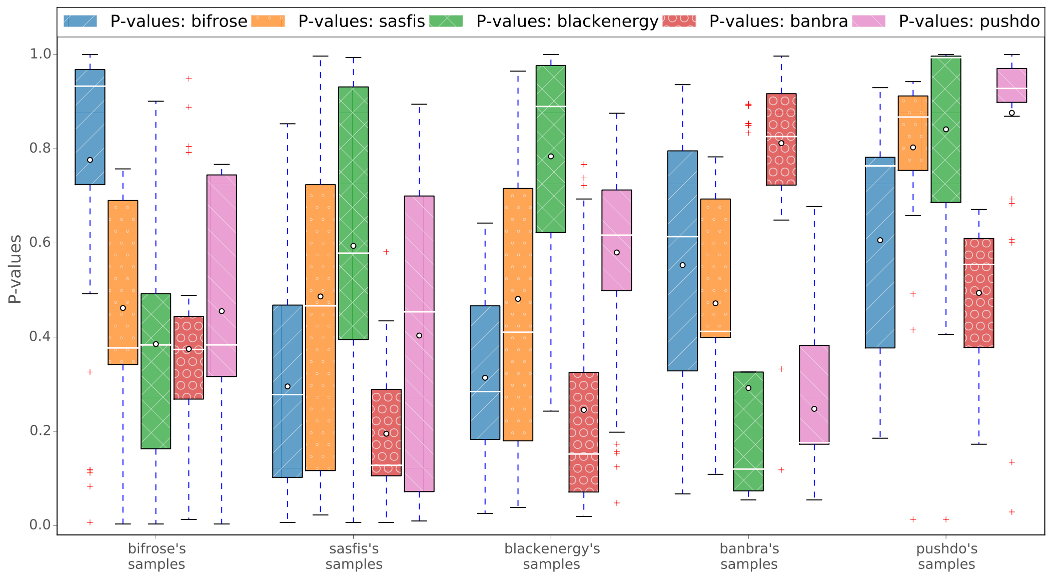


Figure 3.7: Alpha assessment for [102] with the original dataset representing interfering families: white dots represent the average values, white lines represent the median values, red crosses represent outliers.

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

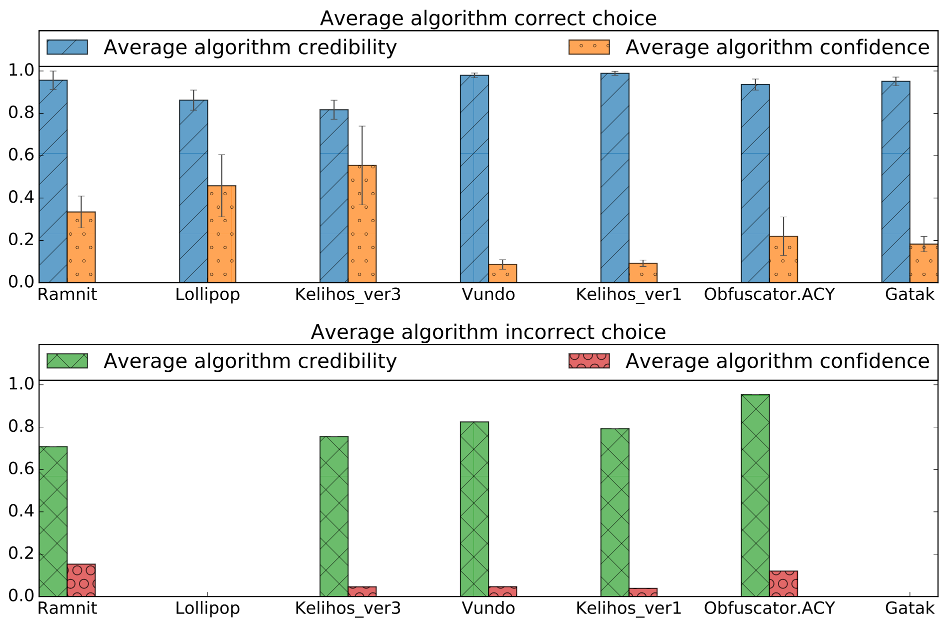


Figure 3.8: Decision assessment for Microsoft Challenge algorithm: poor confidence for correct choices indicates the samples are similar to other families, poor confidence for incorrect choices is desirable but high credibility for incorrect choices it is not.

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 start to interfere with one another (look at the samples by family, e.g., first group of 5 columns and second group of 5 columns) quite heavily, the recall of this family drops. From Fig. 3.7 we can see that Sasfis, Bifrose and Pushdo, are subjected to heavy interference, making it difficult to identify them. Singling out interfering families can help to focus the attention into the most problematic ones, sparing time that would otherwise be spent in a full analysis.

Results with perturbed dataset

Now, I am going to perturb the dataset of [102] using as ground truth the families in Table 3.3. Table 3.7 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 at these results, we can reach the same conclusion that the decision assessment in Fig. 3.6 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, in Fig. 3.9 you can see the alpha assessment to show the heavy family interference.

	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 3.7: Confusion matrix for BotFinder tested on the original and perturbed dataset together.

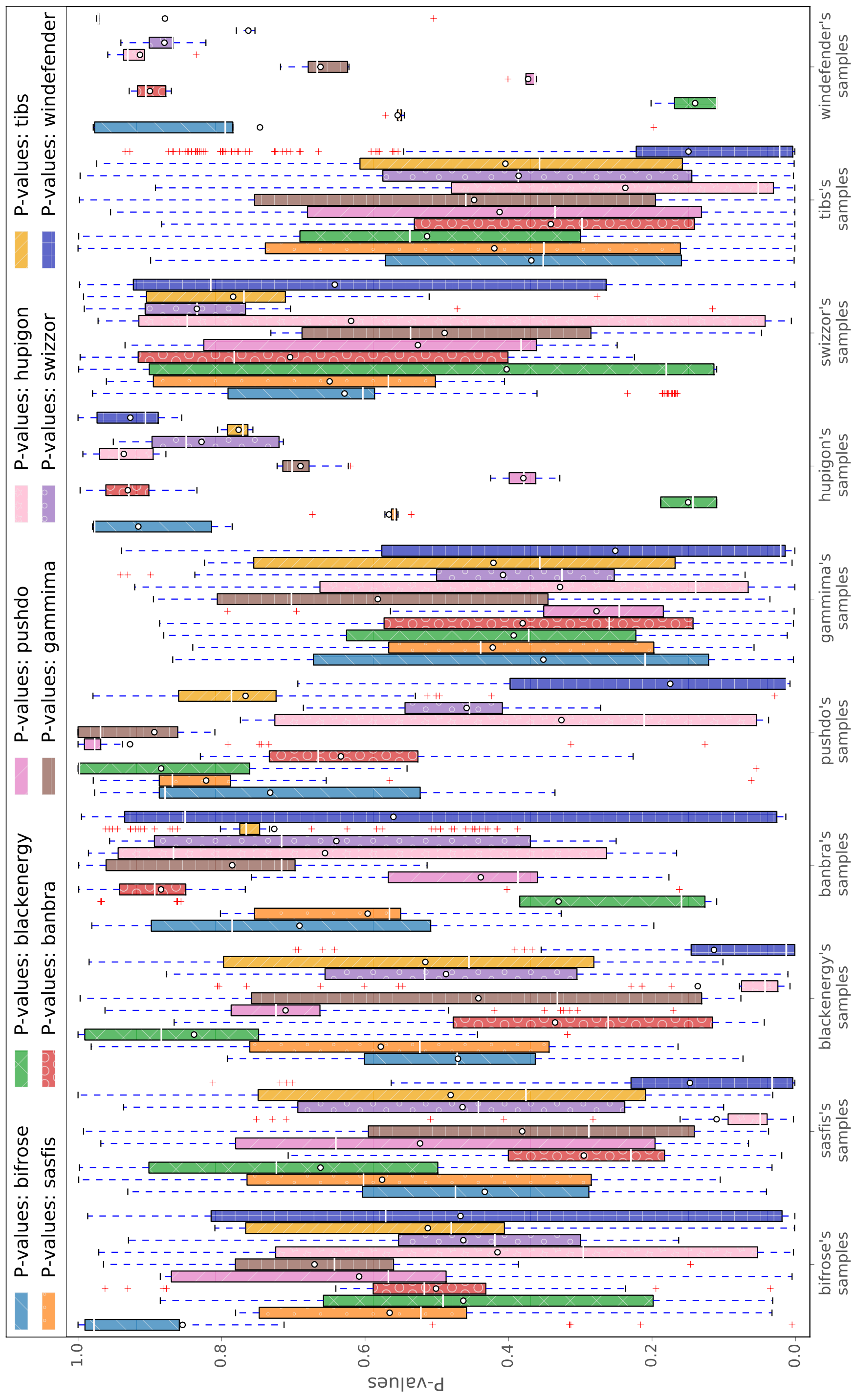


Figure 3.9: Aplha assessment for [102] with perturbed and original dataset representing families interference.

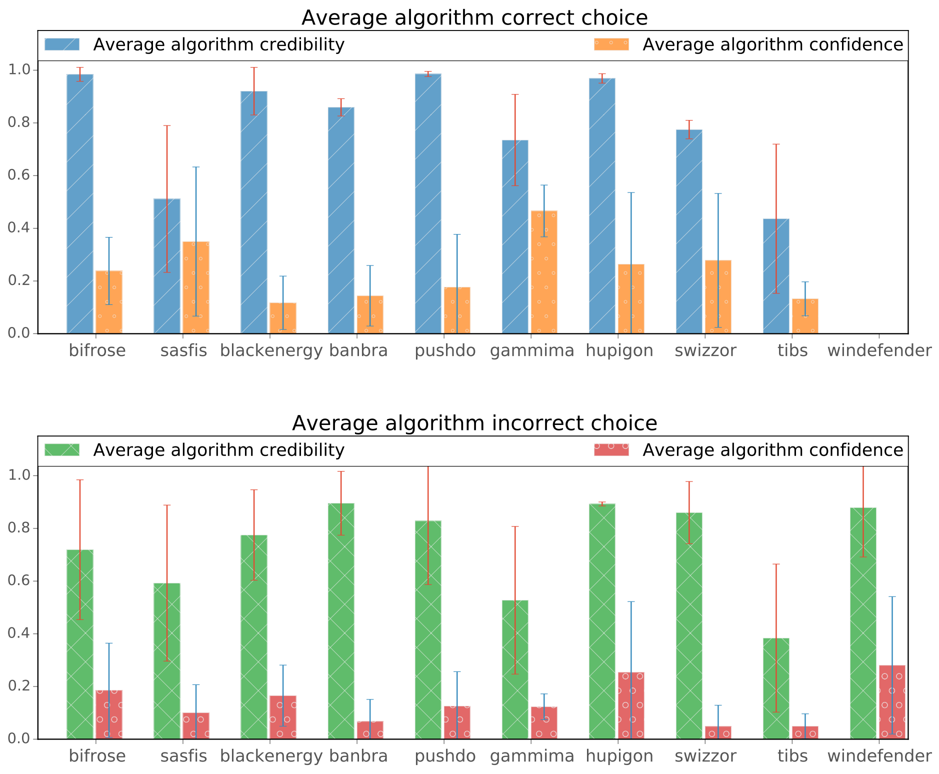


Figure 3.10: Decision assessment for [102] with original and perturbed dataset: perturbing the dataset the results dramatically change.

	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	1 236	0	0	0	1	1	0	0.99
Kelihos_ver3	0	0	1 471	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 3.8: Confusion matrix for the Microsoft Challenge.

3.4.3 Evaluation of Algorithm 3

In this section, I am going to evaluate one of the algorithms proposed as a solution for the Kaggle's Microsoft Malware Classification Challenge described in [3]. Table 3.8 reports the confusion matrix, precision and recall of the algorithm while Fig. 3.8 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 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).

From a quality perspective, drawing upon the alpha assessment of 3.11, two families, Vundo and Ramnit, have significant differences. The Ramnit family has p-values that are much higher than those of the interfering families. However, for Vundo the p-values of interfering families are closer to the correct ones. These details can be only be observed through the alpha assessment, suggesting that the identification of the Ramnit samples would be more robust when the data distribution changes.

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 Fig. 3.11), 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.

We note that confidence for incorrect choice is always lower than the confidence for correct choice indicating that the algorithm is less sure when it predicts a class that is incorrect. For the misclassifications of Kelihos_ver3 the credibility is high,

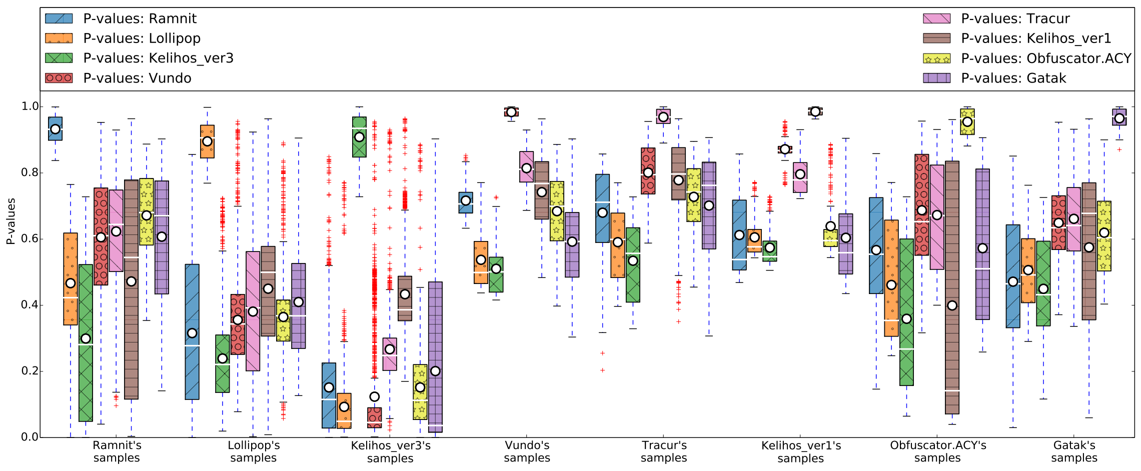


Figure 3.11: Alpha assessment for the Microsoft classification challenge: from this picture we can see that quality of the decision taken by the algorithm. Behind very good results seen on the confusion matrix, the quality of those results is in general vary a lot across different families.

to indicate a high degree of similarity towards that class but the confidence is really low, indicating that there are other families that very similar to these samples. For Vundo and Kelihos_ver1 the credibility and confidence for correct and incorrect choices are similar indicating that the decision boundaries are narrower than other families and prone to more misclassifications. This can be easily spotted when the true labels are available i.e. in the design phase of an algorithm. However, when the true labels are not available, as in the deployed scenario, one is left with only the algorithm choice. The quality metrics defined with the use of conformal evaluator allows us to trust a decision choice when high statistical evidence is available. The family Kelihos_ver3, has a high credibility and the highest confidence, not surprisingly it has the lowest misclassification rate.

3.4.4 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 § 3.6). In the assessment section § 3.4, 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 § 3.4.1. Conversely, we have also shown how apparently good results, according to traditional metrics, can be identified as troublesome (see § 3.4.2). Moreover, as shown in § 4.2, 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 unorthodoxly be used to look for inter-class interference, these are dimensionality reduction algorithms, e.g., PCA [45] (Principal Component Analysis), t-SNE [106] (t-distributed stochastic neighbor embedding), LDA [42] (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, when improperly used to evaluate class-interference, they suffer from the following drawbacks:

- The decision step made by the algorithm is not taken into account by these techniques because they operate on the feature space only. Hence, conclusions based on these methods might be misleading, as shown later on with an example.

- The evaluated algorithm might use a technique to transform the feature space that is different from the one used by these dimensionality reduction techniques. Hence, it is misleading to base conclusions on the results of these techniques.

- The evaluation is limited to subjective visual inspection, they do not provide a quantifiable value of interference. CE instead, 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 (§ 3.4.1), Algorithm 2 (§ 3.4.2) and Algorithm 3 (§ 3.4.3).

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 of 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.

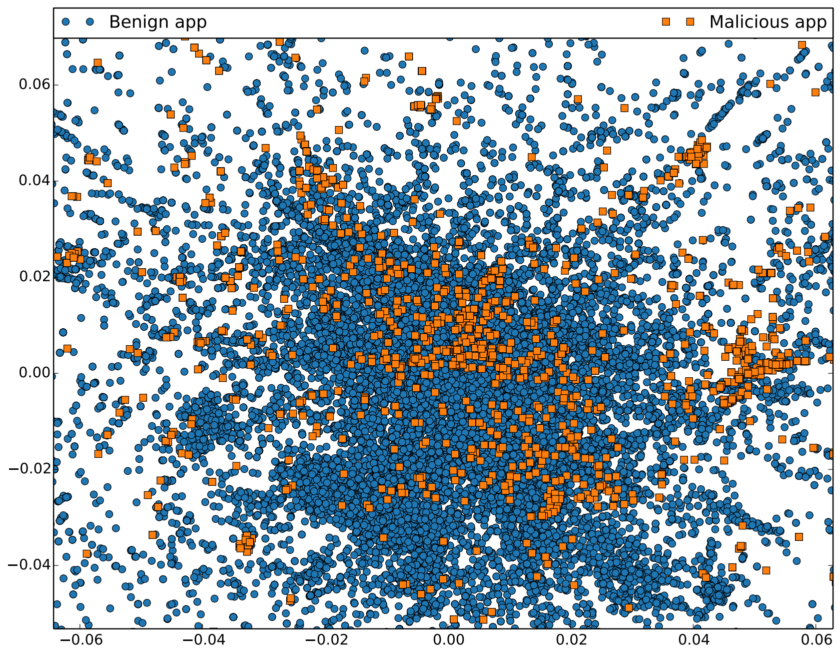


Figure 3.12: t-SNE representation of Drebin dataset on 70K benign samples and 3.5K malicious samples with 1K features reduced with PCA. The malicious samples are superimposed over the benign ones, malicious apps are mixed with benign apps.

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. Fig. 3.12 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 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 Fig. 3.12 we can clearly see that malicious applications are indistinguishable from benign ones(in the plot, malicious apps ore superimposed over benign apps, otherwise they would not be visible) 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.

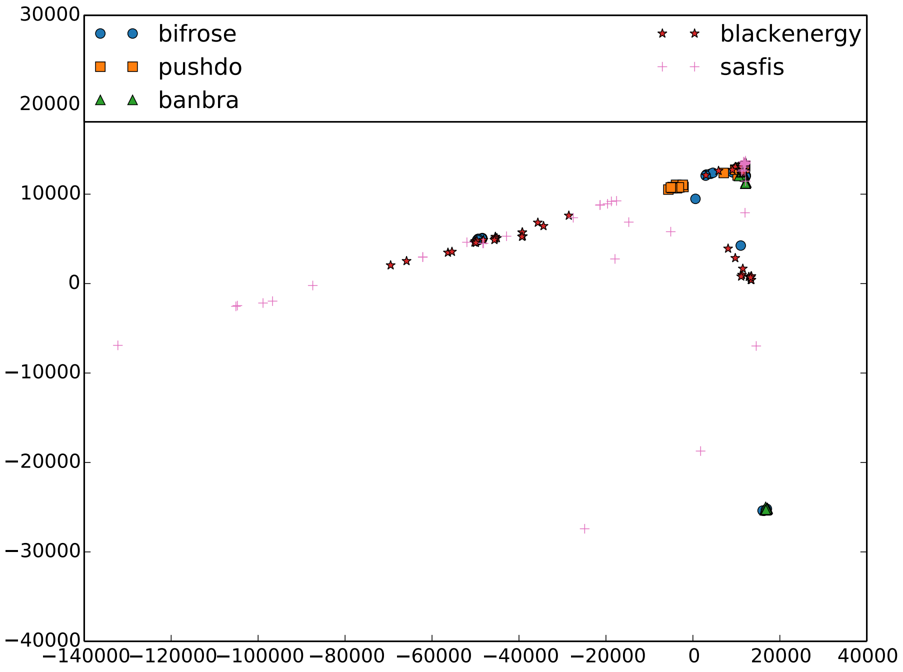


Figure 3.13: Zoomed part of PCA representation of BotFinder dataset. From this view we can see that are few clusters of samples are isolated from the others but the vast majority is not well separated.

Without the limitations imposed by these techniques, as pointed in § 3.4.1, conformal evaluator shows instead that the synergy between features and the algorithm produces very limited interference between benign and malicious samples.

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 3.13 and 3.14 show the 2D plots for PCA and t-SNE, respectively. Fig. 3.13 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 Fig. 3.14. 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, it seems that the chosen features might be a good starting point to separate the families. On the other hand, our analysis on § 3.4.2 shows that families are very much interfering with each others.

Quantifying the amount of family interference by looking at PCA and t-SNE

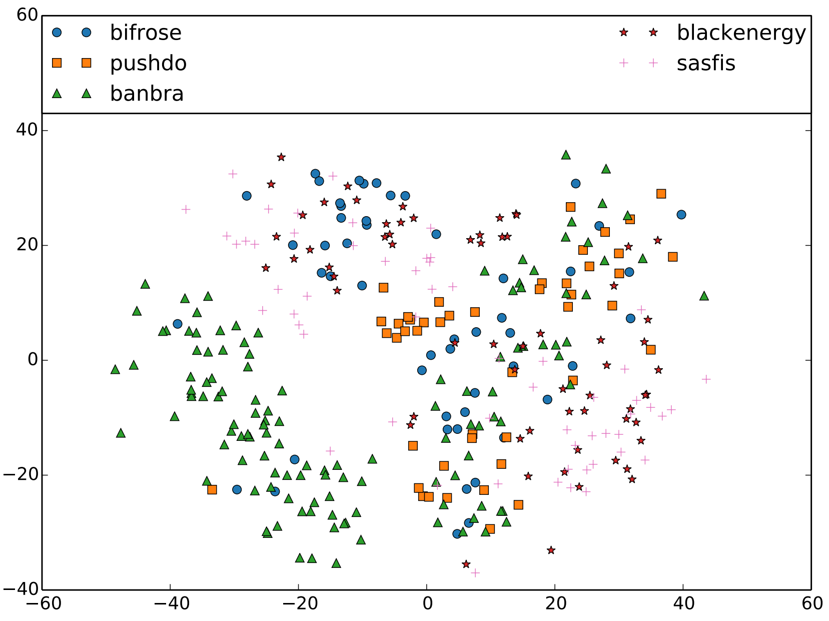


Figure 3.14: t-SNE representation of BotFinder dataset- B. Banbra seems isolated in one cluster while for the other families some small clusters can be identified, nevertheless remaining samples are mixed together.

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 worth discussing. Of course visual evaluation is subject to personal interpretation and cannot be conveyed uniformly to the community.

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

Algorithm 3: PCA and t-SNE Limitations

From the t-SNE projection of the features extracted for the Microsoft challenge (Fig. 3.15), 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 3.8. Even in this case, the algorithm plays an important role in the classification, and hence excluding it from the evaluation is not ideal.

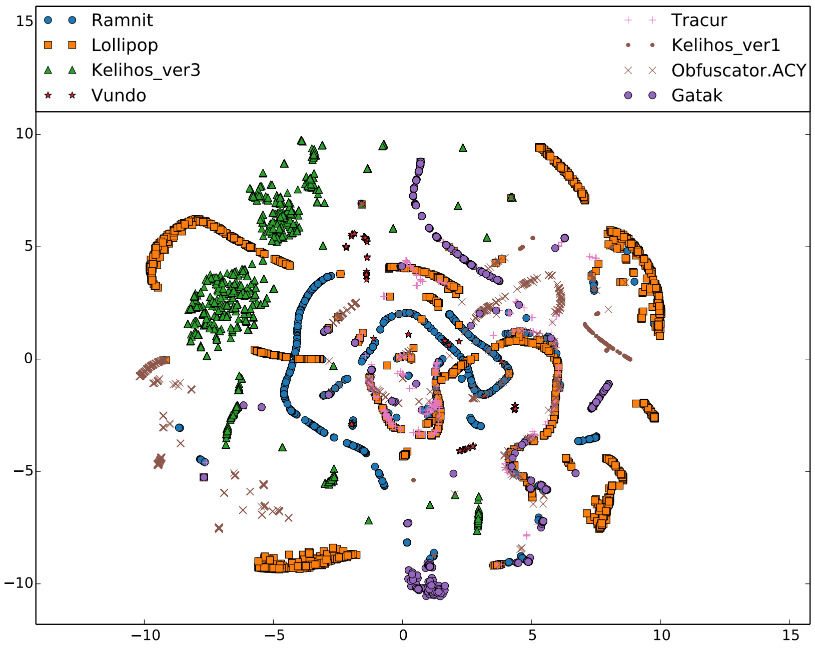


Figure 3.15: t-SNE projection of the features used in the Microsoft challenge: families appear to be well separated from the others in the outer parts of the figure while in the middle they seems mixed together.

As for PCA, the features extracted and plotted in Fig. 3.16 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.

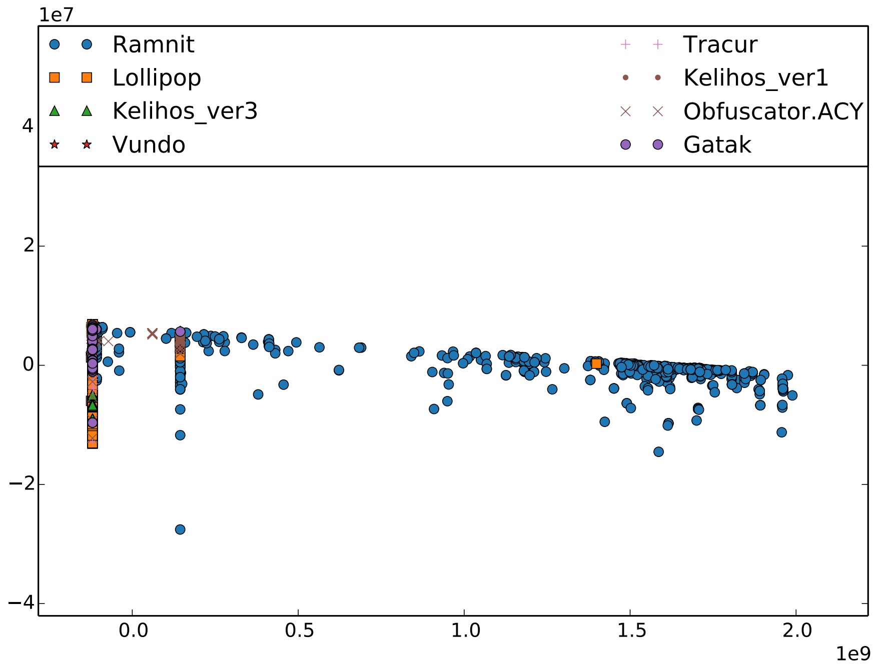


Figure 3.16: 2D PCA projection of the features used in the Microsoft challenge: Ramnit family retains most of the variance while the other families are concentrated on small portion of the graph.

Assessment 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.

3.5 Framework Limitations

In § 3, I have explained the benefits of using conformal evaluator by describing how it can be used to assess the quality of malware classification and clustering algorithms while in § 3.3 I will show empirical evidence of its benefits. However, conformal evaluator has also its drawbacks, which we outline next along with possible ways to address them.

The main limitation consists of 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 [79] 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 [50], 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 underlying machine learning algorithm, conformal predictor, used by conformal evaluator is computationally expensive. For each sample z in a class $k \in \mathcal{K}$, 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 [102] builds one or more models to profile malware behaviours, 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 Eq. 3.4:

$$\mathcal{O}(CN^2) \tag{3.4}$$

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

To speed-up 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 [9] can directly supply the total non-conformity scores for the calculation of one p-value in only one run of the algorithm, thus reducing Eq. 3.4 into Eq. 3.5:

$$\mathcal{O}(CN) \tag{3.5}$$

To further speed up the process, some algorithms treat each class separately. For example [102] 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 2 000 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 Eq. 3.5):

$$2\,000 * 10 = 20\,000 \ \ minutes \approx 13 \ \ days + 22 \ \ hours$$

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:

$$20\,000 \ \ minutes/8 \approx 2 \ \ days$$

Throughout our experiments with [9], 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 23 cores/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 speed up 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) [27]. 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 optimisations can be put in place to reduce the computational complexity of conformal predictor, I want to stress the fact that our framework is meant for evaluation purposes only, hence for an offline scenario, when the algorithm is not yet deployed in the field. For this reason, performance optimisations are not a primary issue and are not in the scope of this work.

3.6 Related Works

Assessing the validity of machine learning techniques is a problem that has not been completely solved. Particularly, Li et al. in [56] 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., [10], [89] and [94]. Specifically [94] 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 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. [7], 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. [28] 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 datasets, and algorithm unavailability for comparison.

As anticipated in the introduction, to evaluate their methods, researchers usually relies on measures that given a labelled dataset, analyse 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 good or bad a 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.

Solutions to detect concept drift, specific to security domains, have been proposed [103, 47, 62], in contrast our framework provides a generic solution which is algorithm agnostic. On the other hand, solutions [26, 36] developed by the ML community have constrains that are not suitable for security applications (e.g., retrospective detection of concept drift when the classification decision has already been made).

Thomas et al. [103] presented *Monarch* a real-time system that crawls URLs as they are submitted to web services and determines whether the URLs direct to spam. The system uses machine-learning to classify URLs as malicious or benign. The authors suggest training the model continuously to keep classification error low as the nature of malicious URLs keeps evolving. Kantchelian et al. [47] propose fusing human operators with the underlying machine-learning based security system to address concept drift in adversarial scenarios. Maggi et al. [62] present a machine-learning based system to classify malicious web applications. They use techniques specific to web application to detect concept drift and thus retrain their model to reduce false positives. Mariconti et al. [64] show how models decay over time and propose ways to resist longer. CE model unifies these techniques as it generalizes to both the area of application and machine-learning algorithm used. The presented model can not only accurately predict when to retrain a model but also provides a quality estimate of the decisions made. These results can reduce human intervention and make it more meaningful thus decreasing the cost of operation. Transcend can be plugged on top of any such approach to provide a clear separation between non-drifting and drifting objects.

Deo et al. [23] propose using Venn-Abers predictors for assessing the quality of binary classification tasks and identifying concept drift. The Venn-Abers predictors offer automatically well calibrated and probabilistic guidance to detect changes in distribution of underlying samples. Although useful, the approach has limitations and cannot draw concrete conclusions on sample clusters which are outliers. Also, Venn-Abers outputs multiple probabilities of which one is perfectly calibrated but it is not possible to know which one. Our approach provides a simple mechanism to compare predictions through p-values and does not suffer from the discussed shortcomings. CE also works on multi-class prediction tasks, while this is not currently supported by Venn-Abers predictors.

Other works try to detect change point detection when the underlying distribution of data samples changes significantly, e.g., in case of evolving malware which is observed as a disruption in exchangeability [107]. Martingales have often been used to detect drift of multidimensional data sequences using exchangeability [38, 37]. Prior works [26, 36] use conformal prediction to detect deviation of the data sequence from i.i.d. assumption which could be caused by concept drift. The drift is measured by creating a martingale function. If the data is not i.i.d., then the conformal predictor outputs an invalid result [26]. Some p-values assigned to the true hypotheses about data labels are too small (or have another deviation from uniformity), and this leads to high values of the martingale. However, this martingale approach does not use p-values assigned to *wrong* hypotheses, which is another cause of wrong classification, e.g., malicious samples being classified as benign. This information is important because in the case of malware evolution, malicious samples are often specially designed to be indistinguishable from benign samples, therefore they tend to get high p-values assigned to wrong hypotheses. Additionally, the martingale approach uses true labels to study the drift of data without making any predictions, in contrast our approach does not have access to true labels and analyses the predictions made by a given model.

Chapter 4

Transcend

In this chapter, I describe Transcend, the algorithm proposed to detect concept drift. I will demonstrate its efficacy on two systems security scenarios: binary and multiclass classification. Nevertheless, the algorithm can be employed in different research domains. It must be stressed here that we look at concept drift from the perspective of a malware analysis team. Consequently, the severity of the drift is a subjective issue. For critical applications, even a few misclassifications can cause major issues. Consequently, the malware analysis team would have a high standard for abandoning an ageing classification model. Transcend was developed to compute per-class threshold based on the quality metrics chosen. It is evaluated using p-value produced by conformal evaluator as base quality metric but it can be employed with other metrics interchangeably.

4.1 Methodology

We make the concept drift detection in Transcend parametric in two dimensions: the desired performance level (ω) and the proportion of samples in an epoch that the malware analysis team is willing to manually investigate (δ). The analyst selects ω and δ as degrees of freedom and Transcend will detect the corresponding concept drift point constrained by the chosen parameters. The goal is to find thresholds that best separate the correct decisions from the incorrect ones based on the quality metrics introduced by our analysis. These thresholds are computed on the training dataset but are enforced on predictions during deployment (for which we do not have labels). The rationale is very simple: predictions with p-values above such thresholds would identify objects that likely fit (from a statistical perspective) in the model; such classifications should be trusted. Conversely, objects out of predictions with p-values smaller than such thresholds should not be trusted as there is lack of statistical evidence to support their fit in the model.

What happens to untrustworthy predictions (and related test—likely drifted—objects) is beyond the scope of this work. It is reasonable to envision a pipeline that would eventually label drifted objects to retrain the machine learning model. While this raises several challenges (e.g., how many objects need to be labelled, how many resources can be invested in the process), we would like to remark on the fact that this is only possible once concept drift is detected: the goal of this research. Not only does Transcend plays a fundamental role in the identification of drifting objects and thus in the understanding of when a prediction should be trusted or not, but its metrics can also aid in selecting which drifted objects should be labelled first (e.g., those with low p-values as are the one that have drifted the most from the trained model).

The following discussion assumes two classes of data, malicious and benign, but it is straightforward to extend it to a multiclass scenario.

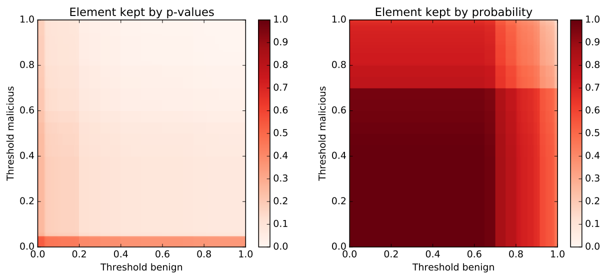
We define the function $f : \mathbb{B} \times \mathbb{M} \rightarrow \Omega \times \Delta$ that maps a pair of thresholds in the benign and malicious class and outputs the performance achieved and the number of decisions accepted. Here, the number of decisions accepted refers to the percentage of the algorithm outputs with a p-value (for benign or malicious classes, depending on the output itself) greater than the corresponding threshold; performance means the percentage of correct decisions amongst the accepted ones. $\mathbb{B}$, $\mathbb{M}$, Ω and Δ are the domains of the possible thresholds on benign samples, malicious samples, desired performance and classification decisions accepted, respectively. During training of our classifier, we iterate over all values of the benign threshold t'_b and the malicious threshold t'_m, at a pre-specified level of granularity, in the domain of $\mathbb{B}$ and $\mathbb{M}$, respectively. Let us assume f gives the output $f{:}f(t'_b, t'_m) = (\omega', \delta')$

To detect concept drift during deployment with a pre-specified threshold of either ω or δ, we need to define an inverse of f which we call $f^{-1} : \Lambda \rightarrow \mathbb{B} \times \mathbb{M}$ where $\Lambda = \Omega \cup \Delta$. When supplied with either ω or δ, f^{-1} would give us two thresholds t_b and t_m which would help Transcend decide when to accept the classifier's decision and when to ignore it. Notice that with a conjoined domain Λ, which only accepts either ω or δ, it is not trivial to reconstruct the values of t_b and t_m. For every value of ω, there could be multiple values for δ. Therefore, we adopt a simple heuristic to compute t_b and t_m whereby we maximize the second degree of freedom given the first. For example, given ω, we find t_b and t_m for every possible value of δ and pick the t_b and t_m that maximizes δ. The formulation is exactly the same when δ is used as an input. The formal equations for the inverse functions are:

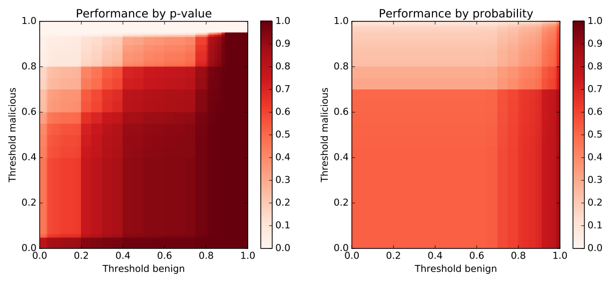
$$\begin{aligned}
\Gamma &= \{x : x \in \forall t'_b \forall t'_m . f(t'_b, t'_m))\} \\
f^{-1}(\omega) &= \{(t_b, t_m) : \delta \in f(t_b, t_m) = max(\forall \delta' \in \Gamma)\} \\
f^{-1}(\delta) &= \{(t_b, t_m) : \omega \in f(t_b, t_m) = max(\forall \omega' \in \Gamma)\}
\end{aligned}$$

Comparison with Probability. The algorithm used as inner non-conformity measure (NCM) in CE may have a pre-defined quality metric to support its own decision-making process (e.g., probability). Hence, we also compare the ability of detecting concept drift of the algorithm's internal metric with CE metrics. The thresholds are extracted from the true positive samples, because we expect the misclassified samples to have a lower value of the quality metric: it seems rather appropriate to select a higher threshold to highlight decisions the algorithm would likely make wrong. We compare our metrics with probability metrics derived from two different algorithms for our case studies. In the first case study (see, § 3.4.1), we compare our metrics with SVM probabilities derived from Platt's scaling [78]; on the other hand, the second case study (see, § 4.4) uses the probabilities extracted from a random forest [16] model. This comparison shows the general unsuitability of the probability metric to detect concept drift. For example, the threshold obtained from the first quartile of the true positive p-value distribution is compared with that of the first quartile of the true positive probability distribution, and so forth.

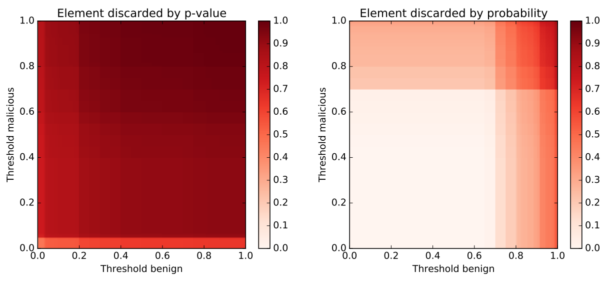
The reasoning outlined above still holds when a given algorithm, adapted to represent the non-conformity measure, uses raw score as its decision-making criteria. For instance, the transformation of a raw score to a probability value is often achieved through a monotonic transformation (e.g., Platt's scaling, for SVM) that does not affect the p-value calculation. Such algorithms do not provide a raw score for representing the likelihood of an alternative hypothesis (e.g., that the test object does not belong to any of the classes seen in the training). Moreover, a threshold built from a raw score lacks context and meaning; conversely, combining raw scores to compute p-values provides a clear statistical meaning, able of quantifying the observed drift in a normalized scale (from 0.0 to 1.0), even across different algorithms.



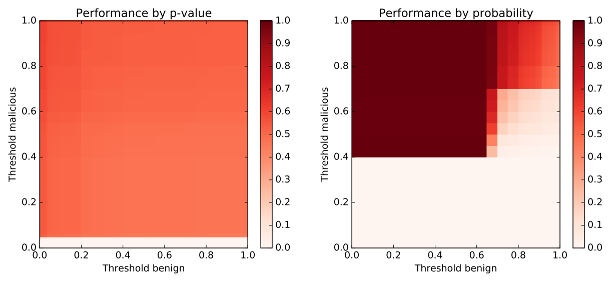
(a) Elements above the threshold.



(b) Performance of elements above the threshold.



(c) Elements below the threshold.



(d) Performance of elements below the threshold.

Figure 4.1: Performance comparison between *p-value* and *probability* for the objects above and below the threshold used to accept the algorithm's decision. The p-values are given by CE with SVM as non-conformity measure, the probabilities are given directly by SVM. As we can see from the graph, p-values tend to contribute to a higher performance of the classifier, identifying those (drifting) objects that would have been erroneously classified.

4.2 Case Studies

To evaluate the effectiveness of Transcend, we introduce two case studies: a binary classification to detect malicious Android apps [9], and a multi-class classification to classify malicious Windows binaries in their respective family [3]. The case studies were chosen to be representative of common supervised learning settings (i.e., binary and multi-class classification), easy to reproduce and of high quality.

Binary Classification Case Study. The algorithm evaluated is described in [9] and in more details in § 3.4. The Drebin dataset was collected from 2010 to 2012 and the authors released the feature set to foster research in the field. To properly evaluate a drifting scenario in such settings, we also use Marvin [58], a dataset that includes benign and malicious Android apps collected from 2010 and 2014. The rationale is to include samples drawn from a timeline that overlaps with Drebin as well as newer samples that are likely to drift from it (duplicated samples were

Drebin Dataset		Marvin Dataset	
Type	Samples	Type	Samples
Benign	123 435	Benign	9 592
Malware	5 560	Malware	9 179

Table 4.1: Binary classification case study datasets [9].

removed from the Marvin dataset to avoid biasing the results of the classifier). Table 4.1 provides details of the datasets.

§ 4.3 outlines this experiment in detail; however, without any loss of generality, we can say models are trained using the Drebin dataset and tested against the Marvin one. In addition, the non-conformity measure we instantiate CE with is the distance of testing objects from the SVM hyperplane, as further elaborated in § 4.3.

Multiclass Classification Case Study. The algorithm used for this experiment is described by Ahmadi et al. in [3] and in more details in § 3.4

Table 4.2 provides details of the Microsoft Windows Malware Classification Challenge dataset. To properly evaluate a drifting scenario, we omit the family Tracur from the training dataset, as further elaborated in § 4.4. In this setting, a reasonable conformity measure that captures the likelihood of a test object z^* to belong to a given family $k \in \mathcal{K}$ is represented by the probability p that z^* belongs to $k \in \mathcal{K}$, as provided by decision trees. We initialize conformal evaluator with $-p$ as non-conformity measure, because it captures the dissimilarities. Please note, we do not interpret $-p$ as a probability anymore (probability ranges from 0 to 1), but rather as a (non-conformity) score CE builds p-values from (see § 2.2.4).

We would like to remark that these case studies are chosen because they are general enough to show how concept drift affects the performance of the models. This is not a critique against the work presented in [9, 3]. Rather, we show that even models that perform well in closed world settings (e.g., k-fold cross validation),

Microsoft Malware Classification Challenge Dataset			
Malware	Samples	Malware	Samples
Ramnit	1 541	Obfuscator.ACY	1 228
Lollipop	2 478	Gatak	1 013
Kelihos_ver3	2 942	Kelihos_ver1	398
Vundo	475	Tracur	751

Table 4.2: Multiclass classification case study datasets [3].

eventually decay in the presence of non-stationary data (concept drift). Transcend identifies *when* this happens in operational settings, and provides indicators that allow to establish whether one should *trust* a classifier decision or not. In absence of retraining, which requires samples relabelling, the ideal net effect would then translate to having high performance on non-drifting objects (i.e., those that fit well into the trained model), and low performance on drifting ones.

In a nutshell, our experiments aim to answer the following research question:

RQ: *How can CE statistical metrics detect concept drift in binary and multiclass classification?* Intuitively, we can interpret quality metrics as thresholds: predictions of tested objects whose quality fall below such thresholds should be marked as untrustworthy, as they drift away from the trained model.

4.3 Binary Classification Case Study

This section assesses the quality of the predictions of Drebin (unless otherwise stated, I refer to Drebin as both the learning algorithm and the dataset outlined in [9]), the learning algorithm presented in [9]. We reimplemented Drebin and achieved results in line with those reported by Arp et al. in absence of concept drift (0.95 precision and 0.92 recall, and 0.99 precision and 0.99 recall for malicious and benign classes, respectively on hold out validation with 66-33% training-testing Drebin dataset split averaged on ten runs).

This section presents a number of experiments to show how Transcend identifies concept drift and correctly marks as untrustworthy the decisions the NCM-based classifier predicts erroneously.

We first show how the performance of the learning model introduced in [9] decays in the presence of concept drift. To this end, we train a model with the Drebin dataset [9] and we test it against 9,000 randomly selected malicious and benign Android apps (with equal split) drawn from the Marvin dataset [58]. The confusion matrix in Table 4.3a clearly shows how the model is affected by concept drift as it reports low precision and recall for the positive class representing malicious objects. Drebin spans the years 2010–2012 while Marvin covers from 2010 to 2014. Most of the Drebin's features capture information (e.g., string and IP addresses) that is likely to change over time, affecting the ability of the classifier to identify non-stationary data. This is further outlined in Fig. 4.2a, which shows how the p-value distribution of malicious objects is pushed towards low values (poor prediction quality).

Table 4.3b shows how enforcing cut-off quality thresholds affects —by improving— the performance of the same learning algorithm. We then asked Transcend to identify suitable quality thresholds with the aim to maximize the F1-score as derived by the calibration dataset, subject to a minimum F1-score of 0.99 and a minimum

	Assigned label		
Sample	Benign	Malicious	**Recall**
Benign	4 498	2	1
Malicious	2 890	1 610	0.36
Precision	0.61	1	

(a)

	Assigned label		
Sample	Benign	Malicious	**Recall**
Benign	4 257	2	1
Malicious	504	1 610	0.76
Precision	0.89	1	

(b)

	Assigned label		
Sample	Benign	Malicious	**Recall**
Benign	4 413	87	0.98
Malicious	255	4 245	0.94
Precision	0.96	0.98	

(c)

Table 4.3: Binary classification case study ([9]). Table 4.3a: confusion matrix when the model is trained on Drebin and tested on Marvin. Table 4.3b: confusion matrix when the model is trained on Drebin and tested on Marvin with p-value-driven threshold filtering. Table 4.3c: retraining simulation with training samples of Drebin as well as the filtered out element of Marvin of Table 4.3b (2 386 malicious samples and 241 benign) and testing samples coming from another batch of Marvin samples (4 500 malicious and 4 500 benign samples). The fate of the drifting objects is out of scope of this paper as that would require to solve a number of challenges that arise *once* concept drift is identified (e.g., randomly sampling untrustworthy samples according to their p-values, effort of relabelling depending on available resources, model retraining). We nonetheless report the result of a realistic scenario in which objects drifting from a given model, correctly identified by Transcend, represent important information to retrain the model and increase its performance (assuming a proper labelling as briefly sketched above).

	TPR of kept elements		FPR of kept elements		TPR of discarded elements		FPR of discarded elements		MALICIOUS kept elements		BENIGN kept elements	
	p-value	probability	p-value	probability	p-value	probability	p-value	probability	p-value	probability	p-value	probability
1st quartile	0.9045	0.6654	0.0007	0.0	0.0000	0.3176	0.0000	0.0013	0.3956	0.1156	0.6480	0.6673
Median	0.8737	0.8061	0.0000	0.0	0.3080	0.3300	0.0008	0.0008	0.0880	0.0584	0.4136	0.4304
Mean	0.8737	0.4352	0.0000	0.0	0.3080	0.3433	0.0008	0.0018	0.0880	0.1578	0.4136	0.7513
3rd quartile	0.8723	0.6327	0.0000	0.0	0.3411	0.3548	0.0005	0.0005	0.0313	0.0109	0.1573	0.1629

Table 4.4: Binary classification case study ([9]): examples of thresholds. From the results we can see that increasing the threshold will lead to keep only the sample where the algorithm is sure about. The number of discarded samples is very subjective to the severity of the shift in the dataset, together with the performance of those sample it is clear the advantage of the p-value metric compared to the probability one.

percentage of kept element of 0.76. In [9], Arp et al. report a TPR of 94% at a FPR of 1%. Such metrics do not rule out the possibility of having 0.99 as F1-score; if that is a plausible constraint, Transcend's parametric framework will find a suitable solution. It is worth noting that such thresholds are derived from the calibration dataset but are enforced to detect concept drift on a testing dataset. Results show how flagging predictions of testing objects with p-values below the cut-off thresholds as unreliable improves precision and recall for the positive (malicious) class, from 0.61 to 0.89 and from 0.36 to 0.76, respectively (in Table 4.3b).

We would like to remark that drifting objects are still given a label as the output of a classifier prediction; Transcend flags such predictions as untrustworthy, de-facto limiting the mistakes the classifier would likely make in the presence of concept drift. It is clear that one needs to deal with such objects, eventually. Ideally, they would represent an additional dataset that, once labelled properly, would help retraining the classifier to predict similar objects. This opens up a number of challenges that are out of the scope of this work; however, one could still rely on CE's metrics to prioritize objects that should be labelled (e.g., those with low p-values as they are the one the drift the most from the model). This might require to randomly sample drifting objects once enough data is available as well as understanding how much resources one can rely on for data labelling. It is important to note that Transcend plays a fundamental role in this pipeline: it identifies concept drift (and, thus, untrustworthy predictions), which gives the possibility of start reasoning on the open problems outlined above.

The previous paragraphs show the flexibility of the parametric framework we outlined in § 4, on an arbitrary yet meaningful example, where statistical cut-off thresholds are identified based on an objective function to optimize, subject to specific constraints. Such goals are however driven by business requirements (e.g., TPR vs FPR) and resource availability (e.g., malware analysts available vs number of likely drifting samples—either benign or malicious—for which we should not trust a classifier decision) thus providing numerical example might be challenging.

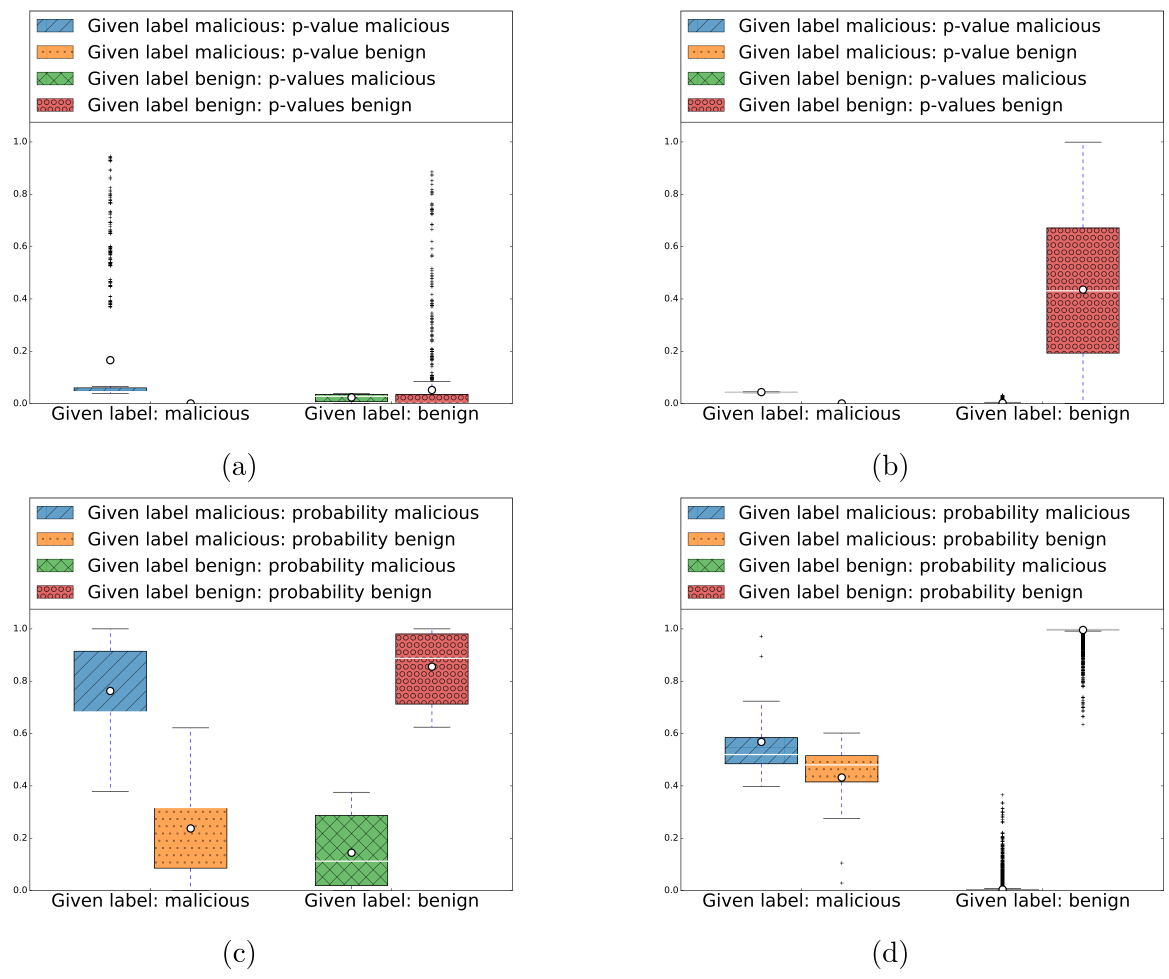


Figure 4.2: Binary Classification Case Study: p-value and probability distribution for true malicious and benign samples when the model is trained on Drebin dataset and tested on Marvin. Graph (a): p-value distribution for true malicious samples. Graph (b): p-value distribution of true benign samples. Graph (c): probability distribution of true malicious samples. Graph (d): probability distribution of true benign samples.

To better outline the suitability of CE's statistical metrics (p-values) in detecting concept drift, we provide a full comparison between p-values and probabilities as produced by Platt's scaling applied to SVM. We summarize a similar argument (with probabilities derived from decision trees) for multiclass classification tasks in § 4.4.

Comparison with Probability. In the following, we compare the distributions of p-values, as derived from CE, and probabilities, as derived from Platt's scaling for SVM, in the context of [9] under the presence of concept drift (i.e., training on Drebin, testing on Marvin as outlined). The goal of this comparison is to understand which metric is better-suited to identify concept drift.

Fig. 4.2a shows the alpha assessment of the classifications shown in Table 4.3a. The figure shows the distribution of p-values when the true label of the samples is malicious. Correct predictions (first and second columns), reports p-values (first column) that are are slightly higher than those corresponding to incorrect ones (second column), with a marginal yet well-marked separation as compared to the values they have for the incorrect class (third and fourth columns). Thus, when wrong predictions refer to the benign class, the p-values are low and show a poor fit to both classes. Regardless of the classifier outcome, the p-value for each sample is very low, a likely indication of concept drift.

Fig. 4.2b depicts the distribution of p-values when true label of the samples is benign. Wrong predictions (first and second columns) report p-values representing benign (second column) and malicious (first column) classes to be low. Conversely, correct predictions (third and fourth columns) represent correct decisions (fourth column) and have high p-values, much higher compared to the p-values of the incorrect class (third column). This is unsurprising as benign samples have data distributions that do not drift with respect to malicious ones.

A similar reasoning can be followed for Fig. 4.2c and Fig. 4.2d. Contrary to the distribution of p-values, probabilities are constrained to sum up to 1.0 across all the classes; what we observe is that probabilities tend to be skewed towards high values even when predictions are wrong. Intuitively, we expect to have poor quality on *all* the classes of predictions in the presence of a drifting scenario: while probabilities tend to be skewed, CE's statistical metrics (p-values) seem better-suited at this task.

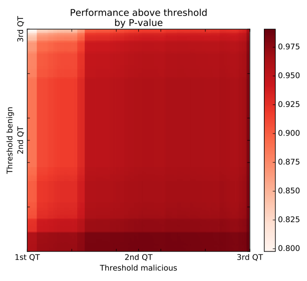
So far, we have seen how Transcend produces statistical thresholds to detect concept drift driven by predefined goals under specific constraints. In addition, the analysis of p-value and probability distributions highlighted how the former seem to be better-suited than probabilities to identify concept drift. In the following paragraphs, we show how CE's statistical metrics provide thresholds that *always outperform* probabilities in detecting concept drift. Fig. 4.3 provides a thorough comparison. For simplicity, here, we focus the attention on the 1st and 3rd quartile, the median and the average of the distribution of p-values and probabilities as potential cut-off thresholds, as shown in Table 4.4.

Intuitively speaking, a successful technique would not only achieve high performances on correct predictions, but it would also report poor performances on drifting objects. This is evident from Table 4.4, where a cut-off threshold at the 1st quartile reports a high performance for the objects that fit the trained model (0.9045 TPR at 0.0007 FPR), and a poor performance for those drifting away (0 TPR and 0 FPR); this means that at this threshold, CE's statistical metrics suggest to consider as untrustworthy only objects the classifier would have predicted incorrectly. Conversely, probabilities also tend to be skewed when predictions are wrong, affecting the ability to rely on such metrics to correctly identify concept drift. Table 4.4 shows 0.6654 TPR and 0 FPR for objects whose quality fall above the 1st quartile of the probability distribution, and 0.3176 TPR and 0.0013 FPR for those who fall below; this means that probabilities marked as unreliable also make predictions that would have been classified correctly.

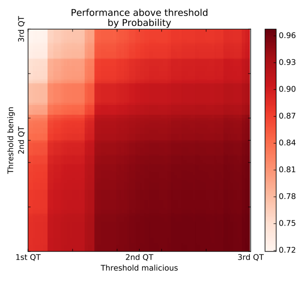
As we move up towards more conservative thresholds, CE's statistical metrics start discarding objects that would have been classified correctly. This is unsurprising as we have defined a threshold that is more selective of the desired quality. Regardless, at each point p-values still outperform probabilities (higher TPR and FPR of objects with a quality higher than the cut-off, and lower for those below the threshold). These results further show how relying on detecting concept drift is a challenging problem that cannot be easily addressed by relying on a prefixed 50% threshold [82].

Note that the number of untrustworthy predictions on the testing dataset is a function of the number of drifting objects. If the entire dataset drifts, we would expect Transcend to flag all (or most of) the predicted objects that do not fit the trained model as untrustworthy.

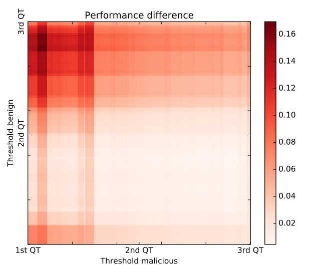
Adapting to Concept Drift. Once drifting objects are identified, the next step would require data relabelling and model retraining, as outlined throughout the thesis. Table 4.3c shows the results of these steps, which take precision for benign samples to 0.89 and recall for malicious ones to 0.76. We would like to remark that this work focuses on the construction of statistical metrics to *identify* concept drift as outlined so far. While relabelling is out of scope for this work, it is clear that an approach that identifies drifting objects is well-suited to address such a challenge in the pipeline as resources can be focused on analysing samples that do not fit in the trained model.



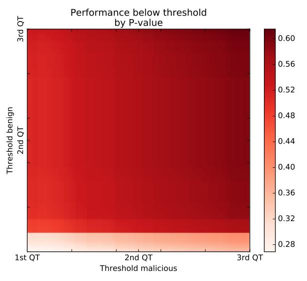
(a) Performance of p-value driven threshold for element above the threshold.



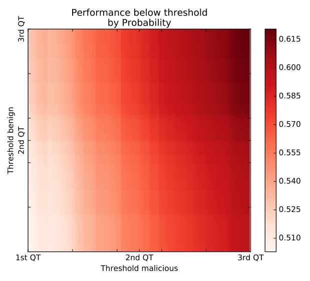
(b) Performance of probability driven threshold for element above the threshold.



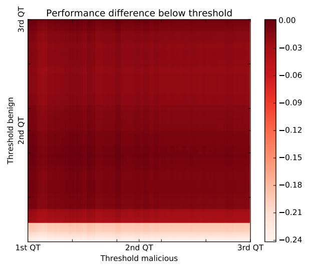
(c) Performance Δ between p-value and probability for element above the threshold.



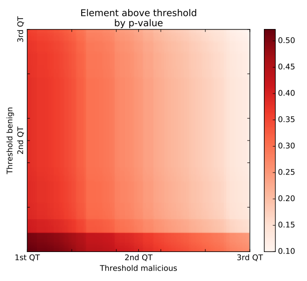
(d) Performance of p-value driven threshold for element below the threshold.



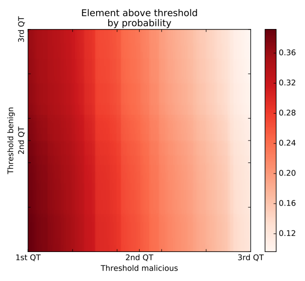
(e) Performance of probability driven threshold for element below the threshold.



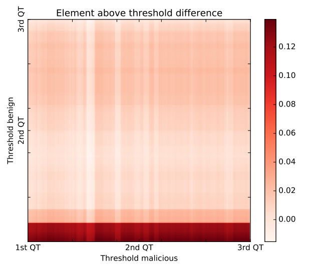
(f) Performance Δ between p-value and probability for element below the threshold.



(g) Number of element above the p-value threshold.



(h) Number of element above the probability threshold.



(i) Δ between the number of element above the threshold between p-value/probability.

Figure 4.3: Binary Classification Case Study [9]: complete comparison between p-value and probability metrics. Across all the threshold range we can see that the p-value based thresholding is providing better performance than the probability one, discarding the samples that would have been incorrectly classified if kept.

4.4 Multiclass Classification Case Study

In this section we evaluate the algorithm proposed by Ahmadi et al. [3] as a solution to Kaggle's Microsoft Malware Classification Challenge; the underlying rationale

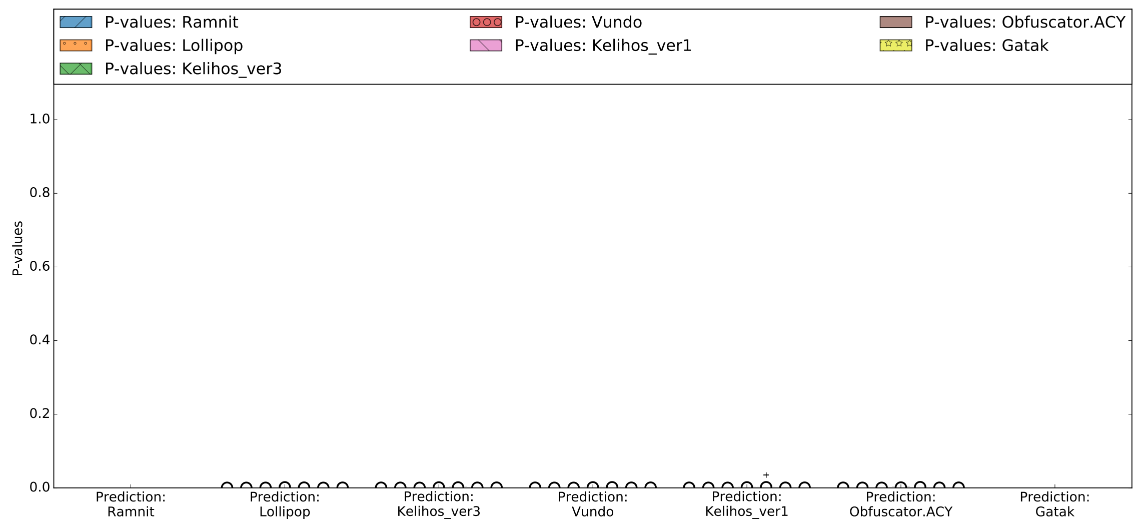


Figure 4.4: Multiclass Classification Case Study [3]: P-value distribution for samples of Tracur family omitted from the training dataset; as expected, the values are all close to zero.

is similar to that outlined in the previous section, thus, we only report insightful information and takeaways. In this evaluation, we train the classifier with seven out of eight available malware families; Trucur, the excluded family, represents our drifting testing dataset.

Family Discovery. Below, we show how we identify a new family based on CE's statistical metrics.

The testing samples coming from Tracur are classified as follows: 5 as Lollipop, 6 as Kelihos_ver3, 358 as Vundo and 140 as Kelihos_ver1. Looking at the distribution of probabilities (Fig. 4.5) and p-values (Fig. 4.4) it is easy to relate to the case of binary classification, i.e., for each family there is only one class with high p-values corresponding to the class of the true label. For the test objects of Tracur, we observe that the p-values for all the classes are close to 0. This is a clear pattern which shows that the samples are coming from an unknown distribution. In a scenario changing gradually, we will observe an initial concept drift (as shown in the binary classification case study in § 4.3), characterized by a gradual decrease of the p-values for all the classes, which ends up in a situation where we have p-values very close to 0 as observed here. These results clearly show that even in multiclass classification settings, CE provides metrics that are better-suited to identify concept drift than probabilities (the algorithm in [3] relies on probabilities because it is based on decision trees). The comparison between p-values and probabilities is reported in Fig. 4.6, 4.7, 4.4 and 4.5 and follow a reasoning similar to that of the binary

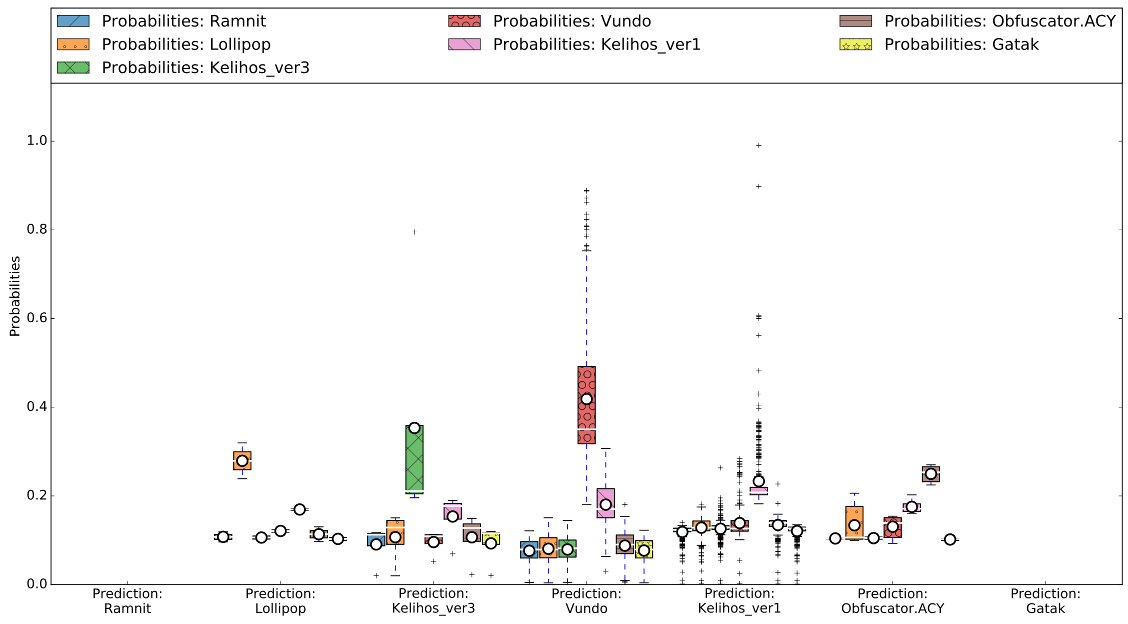


Figure 4.5: Multiclass Classification Case Study [3]: probability distribution for samples of Tracur family omitted from the training dataset. Probabilities are higher then zero and not equally distributed across all the families, making the classification difficult. It is worth noting some probabilities are skewed towards large values (i.e., greater than 0.5) further hindering a correct classification result.

classification case study.

4.5 Discussion

Security community has grappled with the challenge of concept drift for some time now [103, 47, 107]. The problem commonly manifests itself in most malware detection/classification algorithm tasks and models perform poorly as they become dated. Literature [64, 47, 62] recommends retraining the model periodically (see § 3.6) to get around this. However, retraining periodicity is loosely defined and is an expensive process that leads to sub-optimal results. Consequently, there are periods where the model performance cannot be trusted. The problem is further exacerbated as concept drift is hard to identify without manual intervention. If the model is retrained too frequently, there will be little novelty in information obtained through retraining to enrich the model. Regardless of the periodicity, the retraining process requires manual labelling of all the processed objects. Transcend selectively identifies the drifted objects with statistical significance thus is able to restrict the manual labelling process to the objects that are substantially different than the ones in the trained model (see § 4.4 and § 4.3). The p-value for an object z

with label k is the statistical support of the null hypothesis H_0, i.e., that z belongs to k. Transcend finds the significance level (the per-class threshold) to reject H_0 for the alternative hypothesis H_a, i.e., that z does not belong k (p-values for wrong hypotheses are smaller than those for correct ones, e.g., Fig. 3.3)

Comparison with Probability. Probabilities have been known to work well in some scenarios but as demonstrated in § 4.3 and § 4.4 they are not as effective as compared to p-values which are more versatile, especially in the presence of concept drift. When probabilities are reported to be low it is difficult to understand if the sample does not belong to any class or if the sample is actually just *difficult* to classify while still belonging to one of the known classes. In other words, the p-value metric offers a natural *null option* when the p-values calculated for all the classes are low. Instead, as shown in the case of SVM (see, § 4.3), the probability metric is bounded to one of the options in the model. It does not matter if the probabilities are well calibrated or not, the limitation is inherent to the metric. As discussed, the work by Rieck et al. [82] faces similar challenges when choosing the probability threshold. Moreover, the p-value metric provided by our framework, can be calculated from algorithms that do not provide probabilities, e.g., custom algorithms like [102], thus extending the range of algorithms that can benefit from a statistical evaluation.

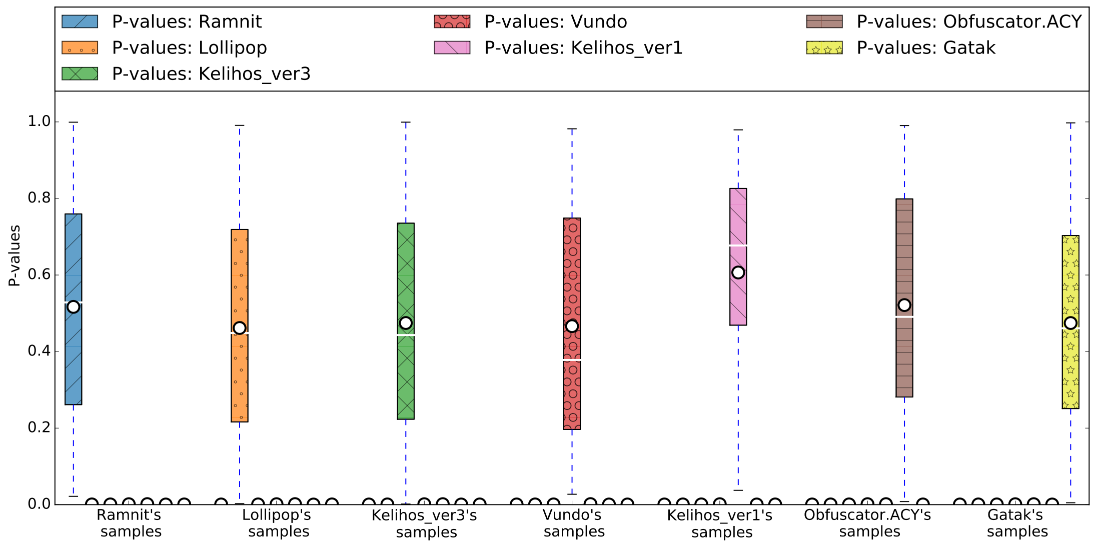


Figure 4.6: Multiclass Classification Case Study [3]: a new family is discovered by relying on the p-value distribution for known malware families. The figure shows the amount of conformity each sample has with its own family; for each sample, there is only one family with high p-value.

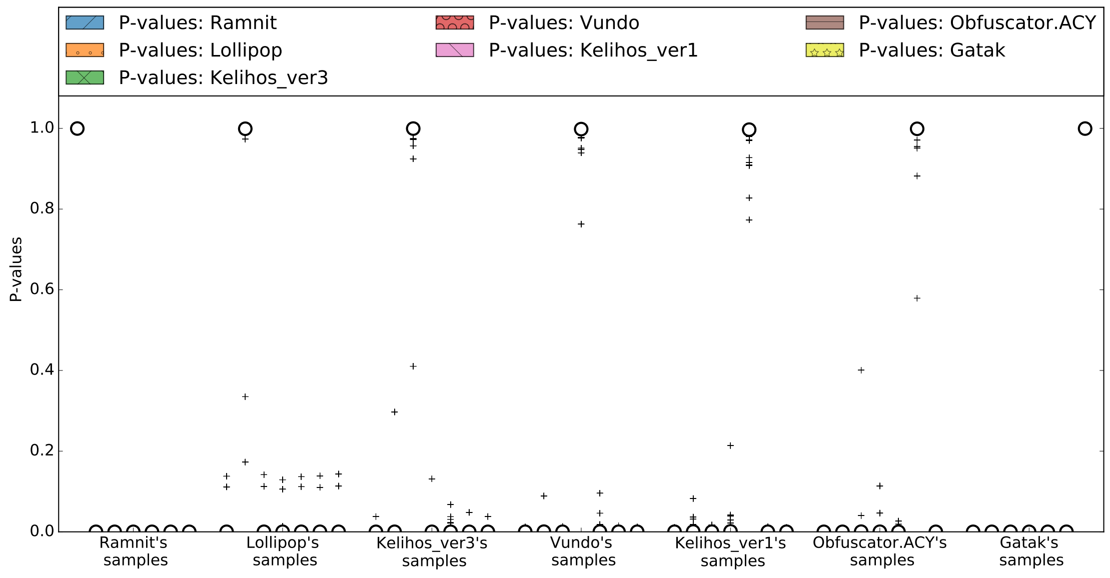


Figure 4.7: Multiclass Classification Case Study [3]: probability distribution for samples of families included in the training dataset. High probabilities support the algorithm classification choice.

Chapter 5

Malware Download Prediction

Malicious websites are the main infection vectors used by attackers to spread malware [71, 41]. The main countermeasure used to deal with this problem is the use of blacklist (e.g., Google Safe Browsing [32]) that prevents the malicious URLs to be accessed by unaware and inexperienced users. Although widely used, the main drawback of this technique is the inability to rapidly update itself [51, 92]. The update time can vary from few hours to few weeks, therefore, in order to conduct a successful malware campaign, malware authors have to change their strategies frequently. Such changes make it very difficult to design an effective defence strategy.

When the blacklist fails, a malware is downloaded. This is usually the result of a web navigation that eventually leads to a malicious web page where the user is tricked to download the malicious content. In this study, I follow this observation to build a predicting model that exploits the history of the web navigation before a download event happens, to decide if the download itself is malicious or benign. I will not consider the spread of a malware propagated by direct link (e.g., phishing) or by malicious attachment (e.g., word, PDF) because in these cases I can not leverage the historical pattern that will end up in a download. To validate this intuition I need a dataset containing the history of the navigation from a real-world environment. For this purpose, I created a dataset by recording the necessary information coming from a major US University for 2 months. To do so, I employed an extension of AMICO [105] that allows to record the HTTP requests coming from each host in a network and reconstruct binary downloads. To label each download I leverage VirusTotal's labelling capability.

The aim of this study is to prevent the download of a malicious binary before the actual executable reaches the user's PC. The results clearly indicate that there is a strong relation between the pages visited before a download and the download itself. This relation needs to be explored further in depth.

5.1 Introduction

Malicious websites are one of the most common sources of threats that spreads malware across the web. An inexperienced internet user is often unaware of the threat that is exposed when browsing malicious content. From scam, phishing, drive-by download, zero-day exploits (just to name few) it seems that there is an always increasing source of danger that threat a safe navigation. Researchers and antivirus firms have proposed many techniques to prevent or alleviate this threat but the problem presents many challenges.

The first challenge is the access to well labelled and documented dataset coming from real-world usage. Researchers often face restriction to access the information due to non-technical reasons mainly related to privacy. When the data is eventually made available to be studied, it is still almost impossible to release it in order to give the opportunity for a broader study by the community. This problem hinders the repeatability of the results and the comparison with new ideas. Once the non-technical challenges have been addressed, the dataset creation needs to care about different technical aspects of the internet, due to its variable nature. When visiting a URL, the server might return different versions of the page for many different reasons. Legitimate reasons include optimisation of the content depending on the browser used to access the page. A malicious server instead might return a malicious page depending on the geographic IP location of the client, exploitability of the browser used to get the page or might return legitimate contents to disguise possible investigation. Additionally, malicious web pages tend to be accessible for a very quick period of time before they are blacklisted therefore made inaccessible. Nevertheless, the efficacy of blacklisting is very limited in time [92, 51], malicious authors change the domain and strategy often to circumvent it. This behaviour makes it very difficult to replicate the experiments. In this work, the information was recorded passively by placing the recording tool at the edge of the network, in order to produce a new detection mechanism.

Many works in the literature exploit the information contained in the downloaded URLs [60, 14, 17, 85, 112, 2]. These works do not prevent the download of the malicious artefact because they analyse it as part of the process. The information that were explored in these works can be categorised as: IP information (geographical location, ASN), domain information (whois, TTL, word contained), page content (HTML and graphical comparison). In another work [72], the authors explore the previously visited pages as part of a broad investigation meant to deal with drive-by download. The work does not build a detection system but aims to provide context around malicious download. In [57], the authors performed active crawling to analyse the resulting web traces. They explore the properties of malicious advertising and their related content delivery network to build a detection mechanism that detects malicious advert. In [68], the authors exploit redirection chains, and the resulting redirection graph, that lead to malicious page to produce a detection system. In this work, I do not restrict the work to redirections but I look at the whole history of requests before a download. Following the redirection idea of [68], in [97] the authors looked at different web browsers and how they interact with websites, they aggregate the information to be more resilient when detecting a page that delivers malware.

In this work, I investigate the importance of the complete browsing history to detect malware downloads by focusing on users behaviours. The goal is to avoid the download of a malware before it reaches the user's machine. To the best of my knowledge, this is the first study that bases the prediction mechanism on the history of the previously visited URLs.

5.2 Methodology

Dataset. In order to obtain the data necessary for this research, I need to record the HTTP requests before a download event happens. For this purpose, I use as baseline for my research the tool AMICO [105]. AMICO is a tool that sits at the edge of a network and is able to reconstruct binaries downloads from the live network traffic (TCP flows) and is able to predict if the downloaded binaries are malware or not. I leverage its binary reconstruction capability and by extending its capability it was possible to record the history of requests and responses that were produced by each machine when they were downloading an executable.

In addition to the information recorded by [105], its extension, named *MIGLIOR* AMICO, recorded the following information from the requests before each download:

- Timestamp: the time of the request
- Server IP: destination of the request
- Hostname: hostname of the destination
- Referer: the field address of the referrer from the HTTP request
- User Agent: user agent of the HTTP request
- Client IP: the source IP address of the request

In order to produce the dataset used in this study, the system ran for 2 months in a major US University, from the mid-November 2017 to mid-January 2018. Due to the sensitivity of the data collected, a clearance process was submitted and authorised prior to the beginning of the study. The clearance process included an ethical committee to raise questions on the research itself. Major concerns raised by the committee were related to the storage and the anonymisation of the data obtained. Furthermore, only the data strictly used for the purpose of that research was authorised to be collected. To address the concern of the committee, the IP addresses were anonymised and the data was stored in a secure server. The anonymity of the data was performed by replacing the original IP address with fictional ones. Although it is not a strong anonymisation procedure and more advanced methods are available, they were not implemented. The data collected was strictly stored in secure servers in the premises of the university. The subsequent analyses on such data were authorised only if the data was not moved by its secure location.

To build a verifiable ground truth, Virus Total [31] was queried to label the downloaded binaries. A malicious label was given if at least 2 antivirus vendors labelled the binary as malicious, benign otherwise. The use of multiple antivirus vendors to confirm the label of a binary was exploited in other works [30, 53, 95]. The query to get the label for each binary was repeated after 1 or 2 weeks to check for consistency. It is known that antivirus vendors change their decisions on binary labels after further investigation [69].

Algorithm. The algorithm used in these experiments is divided into 4 main stages. The first part is involved in the reconstruction of the history of requests for each download starting from the raw data. In the second part, the features are extracted to create the feature vectors. Then a classification algorithm is trained to produce a predictive model and it is tested to produce the final predictions. The train-test set split is performed respecting the timeline (see § 2.4) to ensure real-world usage.

During the first stage of the algorithm, the history for each download must be reconstructed. Sitting at the edge of the network, the raw information of the requests is recorded for each IP address. The inherent limitation from the use of [105] makes the analysis possible only on HTTP traffic and not HTTPS. The reconstruct module tries to link the raw requests together with a limit of 5 minutes before the download event. The use of the time limit is an approximation that takes into account the fact the recent navigation history is usually more related to a download rather than the old one. It is possible that using a browser with multiple websites open at the same time affects the correct reconstruction. This is a limitation of the recording system architecture.

The next step is the extraction of the features to create feature vectors suitable for a ML algorithm. The feature vectors include the following information extracted from the history of each download:

- Alexa top: the number of domain in the Alexa top [8] 1000 websites
- Domain and second level domain name: number of domain and second level domain requested
- Domain and second level domain diversity: the unique number of domain and second level domain requested

- History length: number of requests before the download
- Automatic redirect: the number of automatic redirection (a request is marked as automatic if it comes less than 1 second after the previous one)
- BGP prefix: the BGP prefix extracted from the IP address of the requested hostname
- Words in the requested hostname: the unique words extracted from the hostname of the requests
- IP: the destination IP address of the request

It is not possible to directly use the features of each request before the download as single features because the history of each download may have a different size (the ML algorithm used in the next stage will require feature vectors with the same length). I describe now how I handle the variable history size. During the training phase, the algorithm created 2 counters for each one of the above features. The *benign-leading* counters contain, for each value of a feature, the number of times that a value was observed while the following download was leading to a benign download. The *malicious-leading* counters contain how many times a specific value of a feature was observed while the following download was leading to a malicious download. For example, for the domain name feature, the value "free-mac-update.com" was observed 26 times in the history of requests when a malicious download event eventually happened; the same value was observed 3 times in the history of the requests that ended up in a benign download. The value "google.com" was observed 121 times when downloading a benign binary and 111 times when downloading a malicious binary.

To obtain a fixed number of features suitable for a ML algorithm, I summed the counter's values for each feature (keeping the separation between malicious and benign). For example, 3 requests were observed before a download event. The requested domains before the download are "free-mac-update.com" (h_0), "google.com" (h_1), "google.com" (h_2). As explained before, the *domain name* feature will contains 2 counters. The benign-leading counter will be used to sum the history of benign events: 3 (h_0) + 121 (h_1) + 121 (h_2). The malicious-leading counter will be used to sum the number of malicious events: 26 (h_0) + 111 (h_1) + 111 (h_2). In this way, the length of the history does not matter because for each feature there will be only 2 counters.

The third phase is involved with the creation of the predicting model. I used the XGBoost classifier [101] (explained in § 2.2.3) to train the model and performed a grid search of model's parameters to optimize the F_1 score.

The testing of the model is performed slightly differently from the traditional approach. As described before, the testing was performed respecting the historical timeline, i.e., the testing feature vectors were extracted from the data strictly temporarily after the train set.

After the algorithm predicts a label for a download event, the benign and malicious leading counters, previously described, are updated using the correct label, not the predicted one. This step is performed because the goal is to prevent the download of a malicious sample. Once the download is completed, we can submit it to [31] to get the label or perform other analyses on it. Therefore it makes sense to update the benign and malicious counters with the correct label in order to help the classification of next binaries. I am not using an online ML algorithm thus this update is not changing the predicting model. The purpose of the update is to change only the metadata (i.e., the counters) that are used to generate the features used to test the model.

5.3 Results

The analyses were performed by splitting the timeline into 2 days slots. Initially, I used the first slot as train set and the following slot as test set. After that, the testing slot (previously used as test set) will be included in the train set to produce a new predictive model and it will be used to test the next slot. This mechanism will continue until there are no more slots to be tested.

In Fig. 5.1 and Fig. 5.2, it is shown the precision and recall of the classifier respectively. In Fig. 5.1, we can see that the precision for benign sample is always very high, i.e., above 0.96. The precision for the malware samples is above 0.90 until the beginning of January where it fell to 0.77 before rising again near the end of the examined period.

In Fig. 5.2, we can see that recall for benign samples is always very high, i.e. above 0.98. The recall for the malware samples varies quite a lot, most of the period it stays between 0.73 and 0.87 but during the initial period, it reached a minimum of 0.68 and a maximum of 0.92.

In Fig. 5.1 and Fig. 5.2, the number of benign and malicious elements are shown. The benign downloads increased considerably since the beginning of the data recording, the malware downloads instead, increased very little. Despite the change in the number of downloads, the results did not seem affected by much.

In addition to the aforementioned results, it is interesting to understand how long before the actual download it is possible to predict its maliciousness. Taking the samples that are correctly classified as malicious I ran a backward analysis to understand when the algorithm misclassified the samples by increasingly removing more requests from the history of a download. I divided the misclassification into 5 bands depending on how many requests were excluded from the request history, in order for the model to misclassify a malware. Looking at Fig. 5.4, only 10-20% of the misclassifications happened when removing the last request, i.e., it is possible to identify a malware download even before the last request for most of the correctly

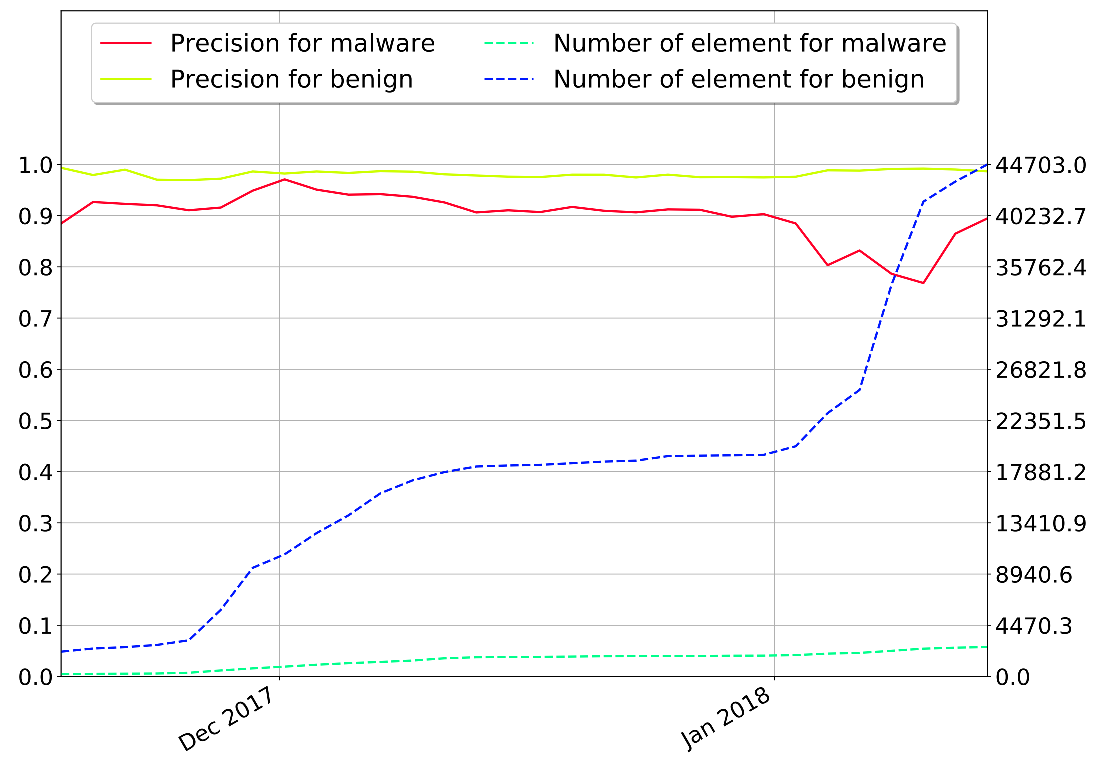


Figure 5.1: Precision for benign and malicious binaries along the timeline on the left y axis. On the right y axes, the number of tested samples.

labelled malware. It is possible to see that most of the downloads are labelled malicious relying on 10 or more requests close to the download event — in Fig. 5.4 these are marked in blue, orange and green. This result is validating the idea that the history of requests close to a download is important to decide its maliciousness. The red and purple represent the samples that were classified as malicious before the latest 10 or more requests.

It will be interesting to see how this analysis is related to the length of the download chain. This is because, in the current model, the value of each counter is summed across the whole history, therefore, long download chains are more likely to have a high score for both benign and malicious counters. Normalizing the history length or limiting its length will be part of the future works. Another interesting analysis that can be performed involves understanding how many times the decision label on a potential download changes during user navigation.

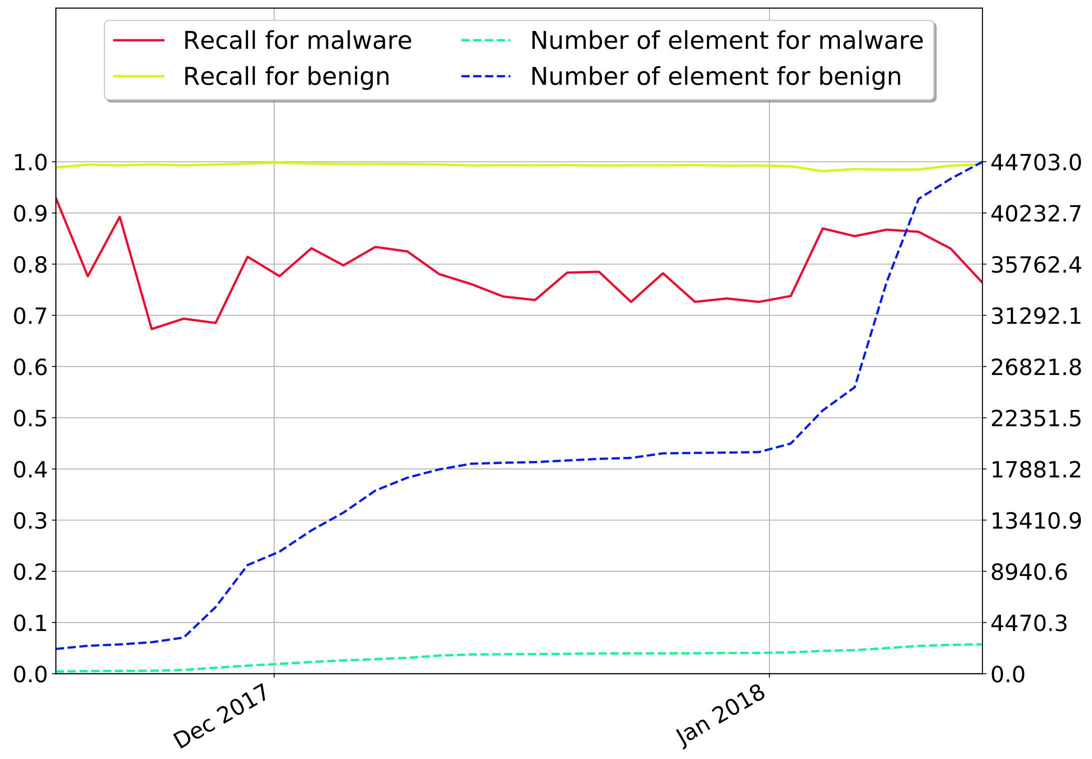


Figure 5.2: Recall for benign and malicious binaries along the timeline on the left y axis. On the right y axes, the number of tested samples.

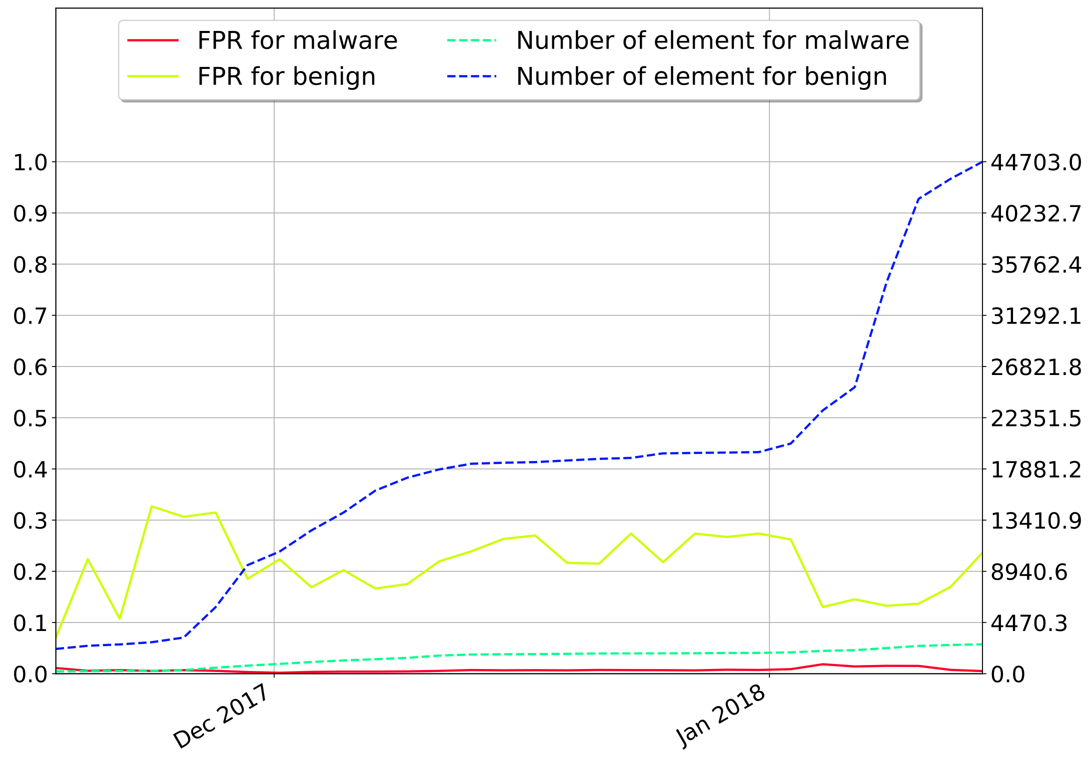


Figure 5.3: False positive rate (FPR) for benign and malicious binaries along the timeline on the left y axis. On the right y axes, the number of tested samples.

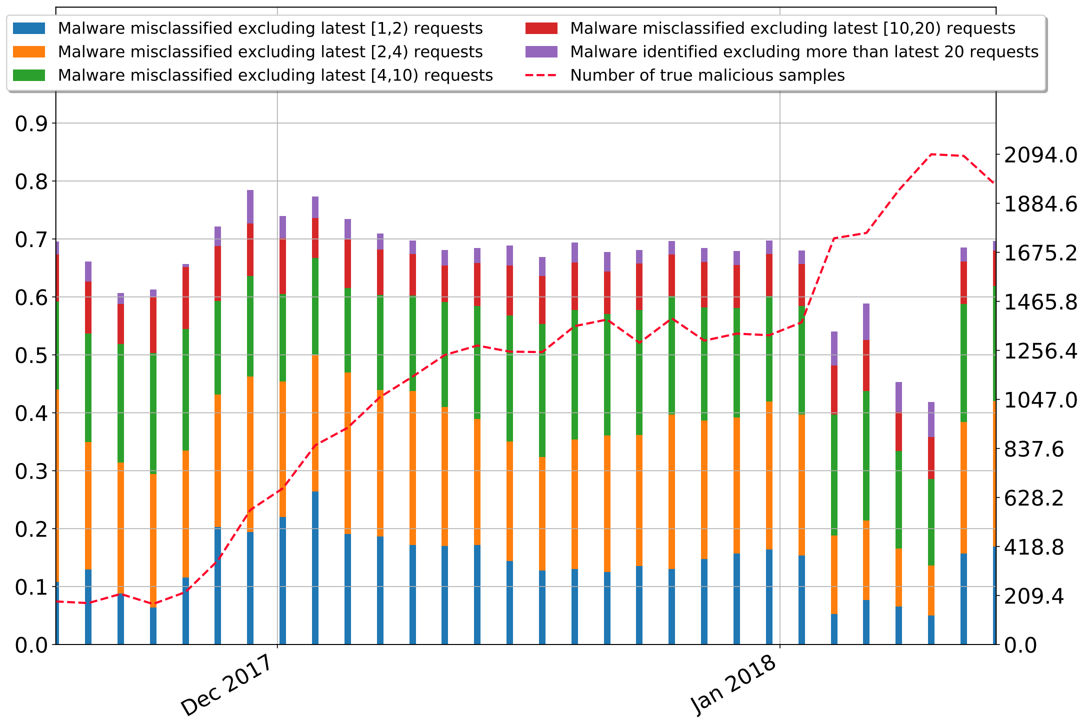


Figure 5.4: Early detection of malware download.

5.4 Future Works

The results show that there is a strong correlation between historical data and the download of malicious malware. Nevertheless, there are limitations that can be overcome in future works.

One of the limitations of this work is that is not possible to understand if the requests coming from a host are coming from the same tab or from different ones. This is due to the architecture of the system that places itself at the edge of the network. To overcome this limitation, a host agent can be developed to distinguish the source of each request. Developing the client agent can be used to produce a warning to the user when her surfing patterns will result in a malicious prediction if tested.

Another limitation of the current experiments is that the data used for the experiments only contains requests that eventually ended up in a download. There is no browsing history that did not complete in a download. To overcome this limitation the recording mechanism needs to be changed.

Other analysis can be performed in addition to the previously described ones. It will be interesting to know how long the request history needs to be, in order to reliably detect a malicious download. Currently, the history length is not limited by the number of requests but by a 5 minute counter.

It will also be interesting to test the system in a long-lived experiment to determine the time period that the ML algorithm would need to analyse to produce a meaningful predictive model, i.e., how long should the train set be? In the current set-up, the algorithm learns to download behaviours from the beginning of the recording up to the beginning of the test set, but malicious actors tends to change their strategies very often. For this reason, it may not be needed to learn several months before the test set.

Finally, another interesting work that can be explored regards to the transferability of a model, i.e., when a predictive model is trained using data collected in a certain environment but tested in a different one. This analysis is interesting because the model is created from the users' behaviour and it is possible that other environments will react differently.

Chapter 6

Conclusion

Learning-based approaches have long been acknowledged as promising in a number of different application domains. When applied in computer security and in particular, to detect and classify malicious software, they often perform *extraordinarily* well. Our community often relies on best-practices and evaluations are generally carried out in controlled lab settings only. Little or no attention is paid to understand whether a given technique would actually retain its staggering performance once deployed in realistic settings, of paramount importance when exploring a fast-paced evolving application domain such as the one malicious software represents.

In this thesis, I contributed to tackle the problem of evolving threats by following two main approaches: one system oriented and one more machine learning oriented. The machine learning approach tries to assess the ML decisions of a given classifier. 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, I have proposed conformal evaluator (§ 3), an evaluation framework built around conformal predictor, a machine learning algorithm originally designed for tailoring classification tasks. Within conformal evaluator, two novel analyses were proposed, which are able to evaluate similarity-based classification and clustering algorithms with respect to their own decisions and against the dataset itself. To this end, such analyses are built on top of algorithm confidence and credibility, which ultimately enable to assess whether the similarity function underpinning such machine learning approaches is poorly designed or badly handled.

I have shown 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 I have shown how apparently good results, according to traditional metrics, can result in inconsistent performances, according to conformal evaluator metrics. Additionally, the robustness of the conformal evaluator results was demonstrated by running the same tests on different datasets. It is worth to note that conformal evaluator can be beneficial to other research communities because of its versatility.

I have also proposed Transcend (§ 4)—a fully tunable tool for statistically filtering out unreliable classification decisions. At the heart of Transcend, CE's statistical confidence provides evidence for better understanding model generalization and class separation; for instance, CE has been successfully adopted to selectively invoke computationally expensive learning-based algorithms when predictions choose classes with low confidence [21], trading off performance for accuracy. My work details the CE metrics used in [21] and extend it to facilitate the identification of concept drift, thus bridging a fundamental research gap when dealing with evolving malicious software. Although evaluated within systems security domain, Transcend can be beneficial to other communities and its inner mechanism allows the use of other quality metrics, making it very versatile.

I presented two case studies as representative use cases of Transcend capability. The approach provides sound results for both binary and multi-class classification scenarios on different datasets and algorithms using proper training, calibration and validation, and testing datasets. The diversity of case studies presents compelling evidence in favour of our framework being generalizable.

In the second part of my thesis I approached the problem of evolving threats in a more algorithmic way. Especially, I tackled the problem of evolving malware campaigns. In this scenario, the attacker continuously changes part of the offensive infrastructure (e.g., URL, website content) to evade the detectors. When inexperienced users are fooled by the attacker they will download the malicious binary leading to an infection. If a user downloads the malware it is considered as *game over* in this model. My defence mechanism involves into updating the counters of websites that showed a malicious download. Higher is the counter more is likely that the website is malicious. It is clear that the very first infections will unlikely be recognised by the system as malicious but this countermeasure can be applied to a whole network infrastructure enabling knowledge sharing of different malicious entities. The system was tested using data coming from a big US university with promising results. Still, more experiments and investigations need to be performed to refine this technique.

By building tools that will allow to study and investigate non-stationary problems in a more comprehensive and scientific manner, the Pandora's Box has been opened; the hope is that more researchers might be stimulated to increase their efforts in the same direction hence bringing more insights than merely numerical results.

The 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.

Bibliography

[1] H.P. Siemon A. Ultsch. Kohonen's self organizing feature maps for exploratory data analysis. In *International Neural Network Conference*, volume 2, page 305–308, 1990.

[2] Sadia Afroz and Rachel Greenstadt. Phishzoo: Detecting phishing websites by looking at them. In *Semantic Computing (ICSC), 2011 Fifth IEEE International Conference on*, pages 368–375. IEEE, 2011.

[3] Mansour Ahmadi, Giorgio Giacinto, Dmitry Ulyanov, Stanislav Semenov, and Mikhail Trofimov. Novel feature extraction, selection and fusion for effective malware family classification. Technical Report, arXiv:submit/1402914, 2015.

[4] J Albert, E Aliu, H Anderhub, P Antoranz, A Armada, M Asensio, C Baixeras, JA Barrio, H Bartko, D Bastieri, et al. Implementation of the random forest method for the imaging atmospheric cherenkov telescope magic. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 588(3):424–432, 2008.

[5] Kevin Allix, Tegawendé F Bissyandé, Jacques Klein, and Yves Le Traon. Are Your Training Datasets Yet Relevant? In *ESSoS*. Springer, 2015.

[6] Kevin Allix, Tegawendé F Bissyandé, Jacques Klein, and Yves Le Traon. Androzoo: Collecting Millions of Android Apps for the Research Community. In *ACM Mining Software Repositories (MSR)*, 2016.

[7] Kevin Allix, Tegawendé François D Assise Bissyande, Jacques Klein, and Yves Le Traon. Machine learning-based malware detection for android applications: History matters! Technical report, University of Luxembourg, SnT, 2014.

[8] Amazon. Alexa-Top-Sites, 2019.

[9] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, and Konrad Rieck. DREBIN: effective and explainable detection of android malware in your pocket. In *21st Annual Network and Distributed System Security Symposium, NDSS 2014, San Diego, California, USA, February 23-26, 2014*, 2014.

[10] Adam J. Aviv and Andreas Haeberlen. Challenges in experimenting with

botnet detection systems. In *Proceedings of the 4th USENIX Workshop on Cyber Security Experimentation and Test (CSET '11)*, 2011.

[11] Stefan Axelsson. The Base-Rate Fallacy and the Difficulty of Intrusion Detection. *ACM Transactions on Information and System Security (TISSEC)*, 2000.

[12] Christopher M Bishop. *Pattern Recognition and Machine Learning.* 2006.

[13] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics).* Springer-Verlag, Berlin, Heidelberg, 2006.

[14] Aaron Blum, Brad Wardman, Thamar Solorio, and Gary Warner. Lexical feature based phishing url detection using online learning. In *Proceedings of the 3rd ACM Workshop on Artificial Intelligence and Security*, AISec '10, pages 54–60, New York, NY, USA, 2010. ACM.

[15] Olivier Bousquet, Stéphane Boucheron, and Gábor Lugosi. *Introduction to Statistical Learning Theory*, pages 169–207. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[16] Leo Breiman. Random Forests. *Machine Learning*, 45(1):5–32, 2001.

[17] Davide Canali, Marco Cova, Giovanni Vigna, and Christopher Kruegel. Prophiler: A fast filter for the large-scale detection of malicious web pages. In *Proceedings of the 20th International Conference on World Wide Web*, WWW '11, pages 197–206, New York, NY, USA, 2011. ACM.

[18] Thomas Cover and Peter Hart. Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27, 1967.

[19] Charlie Curtsinger, Benjamin Livshits, Benjamin G Zorn, and Christian Seifert. ZOZZLE: Fast and Precise In-Browser JavaScript Malware Detection. In *USENIX Security*, 2011.

[20] George E Dahl, Jack W Stokes, Li Deng, and Dong Yu. Large-Scale Malware Classification Using Random Projections and Neural Networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 3422–3426. IEEE, 2013.

[21] Santanu Kumar Dash, Guillermo Suarez-Tangil, Salahuddin Khan, Kimberly Tam, Mansour Ahmadi, Johannes Kinder, and Lorenzo Cavallaro. Droidscribe: Classifying Android Malware Based on Runtime Behavior. In *MoST-SPW*. IEEE, 2016.

[22] Jesse Davis and Mark Goadrich. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd international conference on Machine learning*, pages 233–240. ACM, 2006.

[23] Amit Deo, Santanu Kumar Dash, Guillermo Suarez-Tangil, Volodya Vovk, and Lorenzo Cavallaro. Prescience: Probabilistic guidance on the retraining conundrum for malware detection. In *Proceedings of the 2016 ACM Workshop on Artificial Intelligence and Security*, AISec '16, pages 71–82, New York, NY, USA, 2016. ACM.

[24] Jun Du and Charles X Ling. Active Learning with Human-Like Noisy Oracle. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, pages 797–802. IEEE, 2010.

[25] Eleazar Eskin, Jason Weston, William S Noble, and Christina S Leslie. Mismatch string kernels for svm protein classification. In *Advances in neural information processing systems*, pages 1441–1448, 2003.

[26] Valentina Fedorova, Alex J. Gammerman, Ilia Nouretdinov, and Volodya Vovk. Plug-in martingales for testing exchangeability on-line. In *Proceedings of the 29th International Conference on Machine Learning, ICML 2012, Edinburgh, Scotland, UK, June 26 - July 1, 2012*, 2012.

[27] Alex Gammerman and Vladimir Vovk. Hedging predictions in machine learning. *Computer Journal 50*, pages 151–177, 2007.

[28] Sebastián García, Alejandro Zunino, and Marcelo Campo. Survey on network-based botnet detection methods. *Security and Communication Networks*, 7(5):878–903, May 2014.

[29] Hugo Gascon, Fabian Yamaguchi, Daniel Arp, and Konrad Rieck. Structural Detection of Android Malware using Embedded Call Graphs. In *AISec*. ACM, 2013.

[30] Hugo Gascon, Fabian Yamaguchi, Daniel Arp, and Konrad Rieck. Structural detection of android malware using embedded call graphs. In *Proceedings of the 2013 ACM workshop on Artificial intelligence and security*, pages 45–54. ACM, 2013.

[31] Google. VirusTotal, 2004.

[32] Google. Google safe browsing. https://safebrowsing.google.com/, 2019.

[33] Google. Android Security 2017 Year In Review. https://source.android.com/security/reports/Google_Android_Security_2017_Report_Final.pdf, March 2018.

[34] David J Hand. Measuring Classifier Performance: a Coherent Alternative to the Area Under the ROC Curve. *Machine learning*, 77(1):103–123, 2009.

[35] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001.

[36] Shen-Shyang Ho. A martingale framework for concept change detection in time-varying data streams. In *Machine Learning, Proceedings of the Twenty-Second International Conference (ICML 2005), Bonn, Germany, August 7-11, 2005*, pages 321–327, 2005.

[37] Shen-Shyang Ho and Harry Wechsler. Query by transduction. *IEEE Trans. Pattern Anal. Mach. Intell.*, 30(9):1557–1571, 2008.

[38] Shen-Shyang Ho and Harry Wechsler. A martingale framework for detecting changes in data streams by testing exchangeability. *IEEE Trans. Pattern Anal. Mach. Intell.*, 32(12):2113–2127, 2010.

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

[40] Kaggle Inc. Microsoft malware classification challenge (big 2015), 2015.

[41] Luca Invernizzi, Stanislav Miskovic, Ruben Torres, Christopher Kruegel, Sabyasachi Saha, Giovanni Vigna, Sung-Ju Lee, and Marco Mellia. Nazca: Detecting malware distribution in large-scale networks. In *NDSS*, volume 14, pages 23–26, 2014.

[42] Alan Julian Izenman. Linear discriminant analysis. In *Modern Multivariate Statistical Techniques*, pages 237–280. Springer, 2008.

[43] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning: With Applications in R*. Springer Publishing Company, Incorporated, 2014.

[44] Nathalie Japkowicz and Shaju Stephen. The class imbalance problem: A systematic study. *Intelligent data analysis*, 6(5):429–449, 2002.

[45] Ian Jolliffe. *Principal component analysis*. Wiley Online Library, 2002.

[46] Roberto Jordaney, Kumar Sharad, Santanu Kumar Dash, Zhi Wang, Davide Papini, Ilia Nouretdinov, and Lorenzo Cavallaro. Transcend: Detecting Concept Drift in Malware Classification Models. In *USENIX Security*, 2017.

[47] Alex Kantchelian, Sadia Afroz, Ling Huang, Aylin Caliskan Islam, Brad Miller, Michael Carl Tschantz, Rachel Greenstadt, Anthony D. Joseph, and J. D. Tygar. Approaches to adversarial drift. In *AISec'13, Proceedings of the 2013 ACM Workshop on Artificial Intelligence and Security, Co-located with CCS 2013, Berlin, Germany, November 4, 2013*, pages 99–110, 2013.

[48] Kaspersky. Kaspersky Security Bulletin 2018. https://securelist.com/kaspersky-security-bulletin-2018-statistics/89145/, 2018.

[49] Dong Seong Kim, Ha-Nam Nguyen, and Jong Sou Park. Genetic algorithm to improve svm based network intrusion detection system. In *19th International Conference on Advanced Information Networking and Applications (AINA'05) Volume 1 (AINA papers)*, volume 2, pages 155–158. IEEE, 2005.

[50] Christopher Kruegel, Engin Kirda, Paolo Milani Comparetti, Ulrich Bayer, and Clemens Hlauschek. Scalable, behavior-based malware clustering. In *Proceedings of the 16th Annual Network and Distributed System Security Symposium (NDSS 2009)*, 1 2009.

[51] Marc Kührer, Christian Rossow, and Thorsten Holz. Paint it black: Evaluating the effectiveness of malware blacklists. In *International Workshop on Recent Advances in Intrusion Detection*, pages 1–21. Springer, 2014.

[52] Pavel Laskov and Nedim Šrndić. Static Detection of Malicious JavaScript-Bearing PDF Documents. In *ACSAC*. ACM, 2011.

[53] Pavel Laskov and Nedim Šrndić. Static detection of malicious

javascript-bearing pdf documents. In *Proceedings of the 27th annual computer security applications conference*, pages 373–382. ACM, 2011.

[54] Sangho Lee and Jong Kim. WarningBird: Detecting Suspicious URLs in Twitter Stream. In *NDSS*, 2012.

[55] Kun-Lun Li, Hou-Kuan Huang, Sheng-Feng Tian, and Wei Xu. Improving one-class svm for anomaly detection. In *Proceedings of the 2003 International Conference on Machine Learning and Cybernetics (IEEE Cat. No. 03EX693)*, volume 5, pages 3077–3081. IEEE, 2003.

[56] Peng Li, Limin Liu, Debin Gao, and Michael K Reiter. On challenges in evaluating malware clustering. In *Recent Advances in Intrusion Detection*, pages 238–255. Springer, 2010.

[57] Zhou Li, Kehuan Zhang, Yinglian Xie, Fang Yu, and XiaoFeng Wang. Knowing your enemy: understanding and detecting malicious web advertising. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 674–686. ACM, 2012.

[58] Martina Lindorfer, Matthias Neugschwandtner, and Christian Platzer. Marvin: Efficient and comprehensive mobile app classification through static and dynamic analysis. In *Proceedings of the 39th Annual International Computers, Software and Applications Conference (COMPSAC)*, 7 2015.

[59] Martina Lindorfer, Stamatis Volanis, Alessandro Sisto, Matthias Neugschwandtner, Elias Athanasopoulos, Federico Maggi, Christian Platzer, Stefano Zanero, and Sotiris Ioannidis. AndRadar: Fast Discovery of Android Applications in Alternative Markets. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 51–71. Springer, 2014.

[60] Justin Ma, Lawrence K Saul, Stefan Savage, and Geoffrey M Voelker. Beyond blacklists: learning to detect malicious web sites from suspicious urls. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1245–1254. ACM, 2009.

[61] Federico Maggi, Alessandro Frossi, Stefano Zanero, Gianluca Stringhini, Brett Stone-Gross, Christopher Kruegel, and Giovanni Vigna. Two Years of Short URLs Internet Measurement: Security Threats and Countermeasures. In *WWW*. ACM, 2013.

[62] Federico Maggi, William K. Robertson, Christopher Krügel, and Giovanni Vigna. Protecting a moving target: Addressing web application concept drift. In *Recent Advances in Intrusion Detection, 12th International Symposium, RAID 2009, Saint-Malo, France, September 23-25, 2009. Proceedings*, pages 21–40, 2009.

[63] Davide Maiorca, Giorgio Giacinto, and Igino Corona. A Pattern Recognition System for Malicious PDF Files Detection. In *Intl. Workshop on Machine Learning and Data Mining in Pattern Recognition*. Springer, 2012.

[64] Enrico Mariconti, Lucky Onwuzurike, Panagiotis Andriotis, Emiliano

De Cristofaro, Gordon Ross, and Gianluca Stringhini. MaMaDroid: Detecting Android Malware by Building Markov Chains of Behavioral Models. In *NDSS*, 2017.

[65] Zane Markel and Michael Bilzor. Building a Machine Learning Classifier for Malware Detection. In *Anti-malware Testing Research (WATeR) Workshop*. IEEE, 2014.

[66] McAfee. McAfee Labs Threats Reports. https://www.mcafee.com/enterprise/en-gb/threat-center/mcafee-labs/reports.html, December 2018.

[67] Michael Mccord and M Chuah. Spam detection on twitter using traditional classifiers. In *international conference on Autonomic and trusted computing*, pages 175–186. Springer, 2011.

[68] Hesham Mekky, Ruben Torres, Zhi-Li Zhang, Sabyasachi Saha, and Antonio Nucci. Detecting malicious http redirections using trees of user browsing activity. In *INFOCOM, 2014 Proceedings IEEE*, pages 1159–1167. IEEE, 2014.

[69] Brad Miller, Alex Kantchelian, Michael Carl Tschantz, Sadia Afroz, Rekha Bachwani, Riyaz Faizullabhoy, Ling Huang, Vaishaal Shankar, Tony Wu, George Yiu, et al. Reviewer Integration and Performance Measurement for Malware Detection. In *DIMVA*. Springer, 2016.

[70] Bradley Austin Miller. *Scalable Platform for Malicious Content Detection Integrating Machine Learning and Manual Review*. University of California, Berkeley, 2015.

[71] Antonio Nappa, M Zubair Rafique, and Juan Caballero. Driving in the cloud: An analysis of drive-by download operations and abuse reporting. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 1–20. Springer, 2013.

[72] Terry Nelms, Roberto Perdisci, Manos Antonakakis, and Mustaque Ahamad. Webwitness: Investigating, categorizing, and mitigating malware download paths. In *USENIX Security Symposium*, pages 1025–1040, 2015.

[73] Steven R Ness, Anthony Theocharis, George Tzanetakis, and Luis Gustavo Martins. Improving automatic music tag annotation using stacked generalization of probabilistic svm outputs. In *Proceedings of the 17th ACM international conference on Multimedia*, pages 705–708. ACM, 2009.

[74] Luiz S Oliveira and Robert Sabourin. Support vector machines for handwritten numerical string recognition. In *Ninth International Workshop on Frontiers in Handwriting Recognition*, pages 39–44. IEEE, 2004.

[75] Frank Olken and Doron Rotem. Simple random sampling from relational databases. In *Proceedings of the 12th International Conference on Very Large Data Bases*, VLDB '86, pages 160–169, San Francisco, CA, USA, 1986. Morgan Kaufmann Publishers Inc.

[76] Feargus Pendlebury, Fabio Pierazzi, Roberto Jordaney, Johannes Kinder,

and Lorenzo Cavallaro. TESSERACT: Eliminating Experimental Bias in Malware Classification across Space and Time. In *28th USENIX Security Symposium*, Santa Clara, CA, 2019. USENIX Association. USENIX Sec.

[77] Roberto Perdisci, Davide Ariu, and Giorgio Giacinto. Scalable fine-grained behavioral clustering of http-based malware. *Computer Networks*, 57(2):487–500, 2013.

[78] John Platt et al. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in large margin classifiers*, 10(3):61–74, 1999.

[79] M. Zubair Rafique and Juan 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.

[80] Babak Rahbarinia, Marco Balduzzi, and Roberto Perdisci. Exploring the Long Tail of (Malicious) Software Downloads. In *DSN*. IEEE, 2017.

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

[82] Konrad Rieck, Thorsten Holz, Carsten Willems, Patrick Düssel, and Pavel Laskov. Learning and classification of malware behavior. In *Detection of Intrusions and Malware, and Vulnerability Assessment, 5th International Conference, DIMVA 2008, Paris, France, July 10-11, 2008. Proceedings*, pages 108–125, 2008.

[83] Konrad Rieck, Thorsten Holz, Carsten Willems, Patrick Düssel, and Pavel Laskov. Learning and classification of malware behavior. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 108–125. Springer, 2008.

[84] Konrad Rieck, Tammo Krueger, and Andreas Dewald. Cujo: Efficient Detection and Prevention of Drive-By-Download Attacks. In *ACSAC*. ACM, 2010.

[85] Konrad Rieck, Tammo Krueger, and Andreas Dewald. Cujo: Efficient detection and prevention of drive-by-download attacks. In *Proceedings of the 26th Annual Computer Security Applications Conference*, ACSAC '10, pages 31–39, New York, NY, USA, 2010. ACM.

[86] Konrad Rieck, Philipp Trinius, Carsten Willems, and Thorsten Holz. Automatic analysis of malware behavior using machine learning. *Journal of Computer Security*, 19(4):639–668, 2011.

[87] VF Rodriguez-Galiano, M Chica-Olmo, F Abarca-Hernandez, Peter M Atkinson, and C Jeganathan. Random forest classification of mediterranean land cover using multi-seasonal imagery and multi-seasonal texture. *Remote Sensing of Environment*, 121:93–107, 2012.

[88] Justin Sahs and Latifur Khan. A machine learning approach to android malware detection. In *2012 European Intelligence and Security Informatics*

Conference, pages 141–147. IEEE, 2012.

[89] Luca Salgarelli, Francesco Gringoli, and Thomas Karagiannis. Comparing traffic classifiers. *ACM SIGCOMM Computer Communication Review*, 37(3):65–68, July 2007.

[90] Borja Sanz, Igor Santos, Carlos Laorden, Xabier Ugarte-Pedrero, Pablo Garcia Bringas, and Gonzalo Álvarez. Puma: Permission usage to detect malware in android. In *International Joint Conference CISIS'12-ICEUTE 12-SOCO 12 Special Sessions*, pages 289–298. Springer, 2013.

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

[92] Steve Sheng, Brad Wardman, Gary Warner, Lorrie Cranor, Jason Hong, and Chengshan Zhang. An empirical analysis of phishing blacklists. In *Sixth conference on email and anti-spam (CEAS)*. California, USA, 2009.

[93] Marina Sokolova and Guy Lapalme. A systematic analysis of performance measures for classification tasks. *Information Processing & Management*, 45(4):427–437, 2009.

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

[95] Nedim Šrndic and Pavel Laskov. Detection of malicious pdf files based on hierarchical document structure. In *Proceedings of the 20th Annual Network & Distributed System Security Symposium*, pages 1–16. Citeseer, 2013.

[96] Gianluca Stringhini, Christopher Kruegel, and Giovanni Vigna. Shady Paths: Leveraging Surfing Crowds to Detect Malicious Web Pages. In *CCS*. ACM, 2013.

[97] Gianluca Stringhini, Christopher Kruegel, and Giovanni Vigna. Shady paths: Leveraging surfing crowds to detect malicious web pages. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 133–144. ACM, 2013.

[98] Guillermo Suarez-Tangil, Santanu Kumar Dash, Mansour Ahmadi, Johannes Kinder, Giorgio Giacinto, and Lorenzo Cavallaro. DroidSieve: Fast and Accurate Classification of Obfuscated Android Malware. In *ACM CODASPY*, 2017.

[99] Symantec. 2018 Internet Security Threat Report. `https://www.symantec.com/security-center/threat-report`, 2018.

[100] Gil Tahan, Lior Rokach, and Yuval Shahar. Mal-id: Automatic malware detection using common segment analysis and meta-features. *JMLR*, 2012.

[101] Yuan Tang. extreme gradient boosting.

[102] Florian Tegeler, Xiaoming Fu, Giovanni Vigna, and Christopher Kruegel. Botfinder: Finding bots in network traffic without deep packet inspection. In *In Proc. Co-NEXT 12*, pages 349–360, 2012.

[103] Kurt Thomas, Chris Grier, Justin Ma, Vern Paxson, and Dawn Song. Design and evaluation of a real-time URL spam filtering service. In *32nd IEEE Symposium on Security and Privacy, S&P 2011, 22-25 May 2011, Berkeley, California, USA*, pages 447–462, 2011.

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

[105] Phani Vadrevu, Babak Rahbarinia, Roberto Perdisci, Kang Li, and Manos Antonakakis. Measuring and Detecting Malware Downloads in Live Network Traffic. In *ESORICS*. Springer, 2013.

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

[107] Harry Wechsler. Cyberspace security using adversarial learning and conformal prediction. *Intelligent Information Management*, 7(04):195, 2015.

[108] Jason Weston and Chris Watkins. Multi-class support vector machines. Technical report, Citeseer, 1998.

[109] Li Xu, Zhenxin Zhan, Shouhuai Xu, and Keying Ye. Cross-layer detection of malicious websites. In *Proceedings of the Third ACM Conference on Data and Application Security and Privacy*, CODASPY '13, pages 141–152, New York, NY, USA, 2013. ACM.

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

[111] Zhenlong Yuan, Yongqiang Lu, Zhaoguo Wang, and Yibo Xue. Droid-sec: Deep learning in android malware detection. In *ACM SIGCOMM Computer Communication Review*. ACM, 2014.

[112] Junjie Zhang, Christian Seifert, Jack W. Stokes, and Wenke Lee. Arrow: Generating signatures to detect drive-by downloads. In *Proceedings of the 20th International Conference on World Wide Web*, WWW '11, pages 187–196, New York, NY, USA, 2011. ACM.

[113] Mu Zhang, Yue Duan, Heng Yin, and Zhiruo Zhao. Semantics-Aware Android Malware Classification Using Weighted Contextual Api Dependency Graphs. In *CCS*. ACM, 2014.