Prescience: Probabilistic Guidance on the Retraining Conundrum for Malware Detection

Amit Deo
Royal Holloway, University of London

Santanu Kumar Dash
Royal Holloway, University of London

Guillermo Suarez-Tangil
Royal Holloway, University of London

Volodya Vovk
Royal Holloway, University of London

Lorenzo Cavallaro
Royal Holloway, University of London

ABSTRACT

Malware evolves perpetually and relies on increasingly sophisticated attacks to supersede defense strategies. Data-driven approaches to malware detection run the risk of becoming rapidly antiquated. Keeping pace with malware requires models that are periodically enriched with fresh knowledge, commonly known as *retraining*. In this work, we propose the use of *Venn-Abers* predictors for assessing the quality of binary classification tasks as a first step towards identifying antiquated models. One of the key benefits behind the use of Venn-Abers predictors is that they are automatically well calibrated and offer probabilistic guidance on the identification of nonstationary populations of malware. Our framework is agnostic to the underlying classification algorithm and can then be used for building better retraining strategies in the presence of concept drift. Results obtained over a timeline-based evaluation with about 90K samples show that our framework can identify when models tend to become obsolete.

CCS Concepts

•Security and privacy → Intrusion/anomaly detection and malware mitigation; Mobile platform security; •Computing methodologies → *Machine learning;*

Keywords

Concept drift; malware detection; probabilistic prediction

1. INTRODUCTION

Machine learning has become mainstream in identification of malicious software [4, 21, 24]. When combined with techniques that infer semantic properties of programs, machine learning techniques build models of program behavior that can be used to identify malicious samples. Building a sustainable classification model, however, is a challenging task. Firstly, the knowledge of the human analyst who builds the model based on patterns of malicious behavior is usually incomplete. Secondly, malware behavior tends to evolve over time becoming increasingly sophisticated which renders historical models obsolete. The performance that a classification model from the past is able to achieve on unseen samples, therefore, tends to degrade over time. This phenomenon is commonly referred to as *concept drift* [31].

A common strategy for dealing with concept drift is to retrain the model at periodic intervals. Periodic retraining allows new features that might be characteristic of newer samples to be included in the model. This is not a straightforward task and involves three distinct steps. The first step is to identify if and when test samples are being poorly classified. In the second step, a human analyst needs to look at these samples to identify characteristics which might be of interest. Finally, when a critical mass of poorly classified samples have been investigated, the model can be retrained to enrich it for future classification decisions. In this work, we deal with the first step in this process and investigate techniques for assessing decisions made by classification models. In particular, we propose techniques to identify when the classifier is being forced to make a decision based on ambiguous evidence. Thereby, our work can be used to trigger step two in retraining where a human analyst is involved to further investigate the test sample. It must be clarified, though, that we do not offer any guidance on when the model should be retrained. We believe this to be a subjective decision that relies on the time and resources available.

If the classification model is forced to make a decision based on ambiguous evidence, the security with which classification decisions are made drops hand-in-hand with the classification accuracy. Consequently, for algorithms that compute a probability score along with providing a classification decision, one may think it might be sufficient to look at these scores to understand if the classifier is unsure of the decisions it takes. However, there are two main drawbacks of these scores. Firstly, algorithms that compute these probabilities assume properties related to the distribution of the probabilities which may not always be true. For example, Platt scaling which is used to compute probabilities for Support Vector Machines (SVM), assumes that the probabilities would fit to a sigmoid function. This is not always true and it has been shown in the past that Platt's scaling performs poorly with linearly inseparable data where the distribution of probabilities is not sigmoidal [15]. Secondly, techniques for computing probabilities for a decision are not

AISec'16, October 28 2016, Vienna, Austria
© 2016 ACM. ISBN 978-1-4503-4573-6/16/10...$15.00
DOI: http://dx.doi.org/10.1145/2996758.2996769

transferable across different classifiers. For example, Platt scaling is a technique created specifically for SVM and does not necessarily work well with other classifiers.

In this work, we show how recent advances in machine learning techniques can enable decision assessment in a fast and reliable manner whilst providing theoretical guarantees on the assessment. We propose the use of Venn-Abers predictors which provide guarantees of perfect calibration when used to assess classification decisions. For the case of malware detection, perfect calibration entails that if we have a number of samples that are benign with a predicted probability of p, then the expected proportion of benign samples among these is precisely p. This property is of paramount importance when applying probabilities to decision-making frameworks such as in §2. Perfect calibration is hard to achieve with just a single probability score. For this purpose, Venn-Abers provides multiple probability scores for a classification decision. If the range of the scores is denoted as $[p, q]$, then the closer p is to q, the more accurate the probability score is. A wider range typically signifies that the probability prediction is unreliable and the sample is unusual with respect to the original training data. We corroborate this point by presenting empirical evidence on a causal relation between unsure (but potentially correct) classification decisions made by the model and poor performance during deployment. Apart from theoretically guaranteeing perfect calibration, an added advantage of using the Venn-Abers predictor is that it is agnostic to the underlying classification and can work well with a wide range of classification algorithms. This makes the Venn-Abers predictor a highly versatile technique for decision assessment.

The main contributions of this paper are as follows:

- We discuss a framework to highlight how probabilistic predictors can help minimize operational costs (§2). We make a critical assessment of probabilistic predictors where probabilities are used for decision assessment processes (§3).
- We further show that Venn-Abers probability ranges offer insight into the relative performance of a classification decision while being fully versatile and working with numerous underlying classification algorithms (§4).
- Using a timeline-based evaluation, we show how the Venn-Abers predictor can be used to indicate when retraining might be necessary (§5).

2. PROBABILISTIC GUIDANCE IN OPERATIONAL SETTINGS

In this section, we discuss the costs and benefits of using human analysts and how assessment of classification decisions made by the model can aid the process. Our work is focused on malware detection where samples are classified as being either malicious or benign. We show how a decision assessment framework can efficiently integrate the expertise of a malware analyst in the case of malware detection. For the subsequent discussions, we assume that we have a classification model that emits probabilities of a test sample being malicious or benign along with the predicted class.

Probabilities help in the creation of a decision making framework where we can selectively engage the help of a human analyst in the case of ambiguous decisions. Suppose that we have a malware detection system along with a policy of sending appropriate samples off to malware analysts for further inspection. We would only like to send off samples that are worth investigating because manual inspection can be expensive. In this situation, there are two possible actions that can be performed on each sample, i.e., to either *accept* or *investigate* the decision of the classification model.

-	$\mathtt{A}_{\checkmark}$	I	$\mathtt{A}_{\times}$
Benign	0	a	b
Malicious	0	c	d

Table 1: Table of costs incurred when taking an action on a sample that is either benign or malicious.

Accepting the decision of the classification can be further split up into two categories: those decisions that are correct with respect to the ground truth and those that are not. For the rest of our discussions, we denote the Investigate decisions as I and divide Accept (denoted by A) decisions into two sub-categories: $\mathtt{A}_{\checkmark}$ where the model's decision is correct and $\mathtt{A}_{\times}$ where the model's decision is incorrect.

Suppose we are able to attach costs to each of these actions, probabilities can help us make optimal decisions based on operational constraints. Consider as an example Table 1. Here, each column corresponds to the three actions available at the disposal of the malware analyst as enumerated above and each row corresponds to the true class of a sample.

It can be noted that when the model says that a sample belongs to a benign class (first row) and the decision is correct, the cost of accepting the decision (denoted by action $\mathtt{A}_{\checkmark}$ in the first column) is zero. Similarly, the cost of accepting a correct decision where the sample is deemed malicious is zero. For all other combinations of model decisions and actions taken, there is a fixed cost associated. These costs are subjective. For example, the values of a and c where decisions are referred to the human analyst depend on the size of the team and number of analysts available to perform further investigation. It should also be noted that there are costs involved in accepting a wrong decision. For example, the column $\mathtt{A}_{\times}$ contains non-zero costs b and d for accepting wrong decisions. Incorrect identification of benign/malicious samples often carries a bigger operational penalty than sending it to a human analyst. Therefore, the values for b and d are typically (but not necessarily always) bigger than a and c, respectively.

Malware detection teams would need to choose the constants representing the operational costs according to their goals. We now show how prediction probabilities combined with operational costs can aid the process. If misclassification is not acceptable, the parameters b and d should be relatively large. However, if the team has few analysts, a and c need to be relatively large. For an optimal decision, the expected loss given each action should be considered. In this case, given a probability prediction p of a sample being benign, and the costs of accepting: (i) a benign decision ($\mathtt{A}^{\mathtt{b}}$) and (ii) a malicious decision ($\mathtt{A}^{\mathtt{m}}$), we have the following:

$$\mathbb{E}(\text{Cost}|\mathtt{A}^{\mathtt{m}}) = (1-p)d \quad (1)$$

$$\mathbb{E}(\text{Cost}|I) = pa + (1-p)c \quad (2)$$

$$\mathbb{E}(\text{Cost}|\mathtt{A}^{\mathtt{b}}) = pb \quad (3)$$

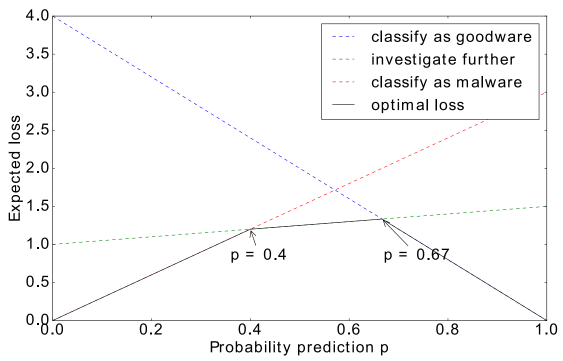


Figure 1: A plot of the expected loss associated with each action against the probability prediction p. The solid black line indicates the loss associated with the optimal action. Parameters chosen: $a = 1.5, b = 3, c = 4, d = 1$.

Equations 1 through 3 show a linear relationship between the expected operating costs for actions `I` and $\mathtt{A}^{(\mathtt{m},\mathtt{b})}$ and the probability p of a sample being benign. If these costs are plotted against the probability of a sample being benign, the optimal operational setting is where we minimize the cost for each probability value. This is an important aspect of using probabilistic guidance in an operational setting and is further explained using the example from Figure 1. In Figure 1, we have plotted the cost functions against probabilities after using constant values of $a = 1.5, b = 3, d = 4, c = 1$. It may be seen that if we minimize the operational cost for each probability value, it is now possible to identify the exact points at which alternate actions should be considered. For our example, we should investigate the decision of the classifier when $p \in [0.4, 0.67]$ and take action `I`. Otherwise, we should accept the classifier decision and take action `A`.

3. DECISION ASSESSMENT TECHNIQUES

Ambiguity in decisions made by a classification model occur when the differentiating factors of individual classes are ill-defined or missing from the classification model. In such cases, the classifier is forced to pick a class for the test sample but alternate choices may also be possible and, at times, almost as fitting as the chosen class. Techniques that derive the probability of a sample belonging to a class or a statistical p-value for the hypothesis that the sample belongs to a class can be used to see how well a sample fits into individual classes. In this section, we give an overview of the approaches available to assess when a classifier decision is reliable. We also provide empirical evidence of probability scores derived from conventional machine learning algorithms not being reliable techniques for decision assessment.

For discussing various decision assessment techniques, it is necessary to introduce the notation and terminology that will be used. First of all, we denote individual feature vectors of each sample with the symbol x. The set of all possible feature vectors is represented as X. Since we will only deal with the binary classification problem, the labels for the samples will be denoted as $y \in \{0, 1\}$. In this paper, we will always be considering decision assessment techniques that build on top of an underlying algorithm that outputs a score. A score could be a probability measure of a sample belonging to a particular class or distances from hyperplanes separating the classes. This algorithm will always be *trained* and its output score will be denoted by the function $s : X \to \mathbb{R}$ where $\mathbb{R}$ is the set of real numbers. Given an input feature vector x, the trained algorithm's output score is $s(x)$. Feature vectors will be interchangeably referred to as objects.

3.1 Assessment Using P-values

P-values have been used in the past to assess how well test samples fit into classes seen during training [32]. Conformal Prediction (CP), as detailed in [32], uses p-values to statistically assess how well a sample fits into a family and uses this information to identify, with a given confidence level, the sets of classes a test sample might belong to. Whenever a p-value for a test sample and class is calculated, a null hypothesis that the sample belongs to the class is assumed to be true. A large p-value indicates significant evidence in favor of the null hypothesis and therefore, if a test sample and class combination exhibit large p-values, it is likely that the test sample belongs to the class.

For computing p-values, CP takes a *non-conformity score* as input. This is a measure (e.g., geometric or probabilistic) of how well a sample fits into a class. For example, in the case of SVM, the non-conformity score could be derived from the sample's distance to the segregating hyperplane, with samples closer to the hyperplane being less conforming. Let's assume that a real-valued function $\mathcal{N}$ gives the non-conformity score for a test object z and a given class B. The p-value for the null hypothesis that z belongs to B can be derived by putting $\mathcal{N}(B, z)$ in context with other samples from the same class, i.e., $\{\mathcal{N}(B, z') : z' \in B, z' \neq z\}$. Thus, the p-value is computed to be the proportion of samples in a class with identical, or higher, non-conformity scores.

We now describe how p-values can be used in identifying a prediction set for a test object using techniques such as CP. When the p-values for all classes of a sample are written out in descending order, introducing a cut-off level ρ selects a potential set of classes to which the sample may belong to with a probability of at least ρ. All other classes having p-values less than ρ are treated as rejected options for classification. Recent attempts to statistically evaluate classification tasks [14], enabled to selectively invoke CP when applied to Android malware to improve the overall classification accuracy by including alternate high p-value choices [8].

Although decision assessment using p-values is insightful, it is very expensive. For testing the null hypothesis on the test sample, it needs to be placed in every possible class and its p-value compared with the p-values of other samples in the class. Whenever a p-value needs to be computed, a standard classification algorithm needs to be run[1]. If there are n samples in the training dataset and c possible classes, we need to run a traditional classification algorithm $(n \times c)$ times to obtain the non-conformity measures for all samples for all classes, and this makes assessment using p-values expensive for large datasets. We now discuss probabilistic predictors, which output a probability of a test object belonging to a particular class. Unlike p-value based assessment, the advantage of probabilistic predictors is that they emit probabilities of the test object belonging to a particular class for all classes with just a single run of the underlying algorithm.

[1]Non-conformity scores, used to compute p-values, are derived by a given classification algorithm.

3.2 Probabilistic Predictors

A probabilistic predictor is simply an algorithm that outputs a probability given a test object. This output will be the probability of the true label of the input being $y = 1$.

Calibrated Probabilities. Let Y be a random variable taking values in $\{0, 1\}$ and P be the random variable associated with the output of a probabilistic predictor. A probabilistic predictor is said to be *well calibrated* if the following holds for all possible values of p:

$$\mathbb{E}(Y|P = p) \approx p.$$

Alternatively, this equation can be written as $\Pr(Y = 1|P = p) \approx p$. The notion of *perfect calibration* is essentially the same apart from the requirement that *exact* equality must hold. In the case of perfect calibration, the probability of a true label being 1 corresponding to the prediction $P = p$ is precisely p. Therefore perfect calibration is the ideal aim for probabilistic predictors.

Platt Scaling. Suppose we have a binary classifier that outputs some score on each test object (denoted by function s) in order to predict its true label. An example of a score could be the distance to the separating hyper-plane of some SVM. How do we translate these scores into well-calibrated probability predictions on the true label? A popular solution to this problem is to apply Platt scaling [20]. This method was initially intended to produce well calibrated probabilities for the output of SVMs and works by fitting logistic regression to the output scores of the trained model. In more detail, the posterior probability of a label y given a score s' is assumed to take the form of a sigmoid function. This assumption is written as:

$$\Pr(y = 1|s') = \frac{1}{1 + e^{As'+B}}.$$

The parameters A and B are calculated using a maximum likelihood principle. This calculation follows in the traditional way for the most part, but Platt suggests a regularization technique to avoid overfitting. Suppose we have a training set $\{(x_1, y_1), ..., (x_n, y_n)\}$ and have calculated the scores $s_i = s(x_i)$. Let N_+ and N_- be the number of training objects with the labels $y = 1$ and $y = 0$ respectively. Then the relabeling for $y = 1$ is $t_+ = (N_+ + 1)/(N_+ + 2)$ and for $y = 0$, the label $t_- = 1/(N_- + 2)$ is used. The labels t_+ and t_- should be interpreted as target probabilities. Having defined the new labels t_i, the minimization problem used to find the parameters A and B is

$$\min_{A,B} -\sum_i t_i \log(p_i) + (1 - t_i) \log(1 - p_i)$$

where $p_i = \Pr(y = 1|s_i)$ as given by the sigmoid function. It turns out that Platt scaling isn't always the best choice when it comes to extracting well calibrated probabilities for arbitrary score outputting algorithms. For example, it has been shown that Platt scaling performs relatively poorly compared to alternatives in some settings [35, 15]. Another well known method is isotonic regression.

Isotonic Regression. Isotonic regression works by fitting an isotonic (i.e. non-decreasing) function to a set of data points. It does so by minimizing a weighted least squares function. First of all, let's assume that we have already trained a binary classifier and obtained the resulting scoring function $s : X \to \mathbb{R}$. In our context of binary classification, let's say that we are given a dataset $\{(x_1, y_1), \ldots, (x_k, y_k)\}$ where the $x_i \in X$ are feature vectors and the $y_i \in \{0, 1\}$ are true binary labels. We can then feed the feature vectors into the scoring algorithm to get a set of scores $\{s_1, \ldots, s_k\}$ where $s_i = s(x_i)$. Following this, we can reorder and remove duplicates to get the set of distinct scores $\{s'_1, ..., s'_{k'}\}$ where $s'_i < s'_{i+1}$. The isotonic regression of the scores is the set of numbers $\{f_1, \ldots, f_{k'}\}$ subject to the constraint $f_i < f_{i+1}$ that minimize the least squares function

$$\sum_{i=1}^{k'} \sum_{j:s_j = s'_i} (y_j - f_i)^2. \tag{4}$$

Intuitively, one can think of the numbers $f_1, \ldots, f_{k'}$ as values of a function f at the individual points $s'_1, \ldots, s'_{k'}$. In order to extend the domain of f, we create a step function by extending the points to the left. In other words, the step function is defined as $f(s) = f_i$ for $s \in (s_{i-1}, s_i]$. Isotonic regression can be used to produce well calibrated probability predictions [19, 35, 17] but there is a somewhat natural extension that we will focus on that is discussed in §4.

Empirical Comparison. In order to show the key characteristics of Platt scaling and isotonic regression, we present some empirical results in a manner similar to the literature in this area [19]. Figure 2 shows a plot with four distinct curves. The data points for these curves have been obtained after running the *Random Forests* algorithm in the Android malware detection domain. We use the Scikit-learn library whose *RandomForestClassifier* also gives probability scores for a decision through the `predict_proba` function. The details of the datasets that we use for training and testing can be found in §5.2. For training, we used the datasets McAfee goodware and Drebin malware sets and for testing we used the Marvin goodware and malware sets. Although not necessary at this point, the full details of the features used for this experiment can be found in §5.1. The output scores from the underlying algorithm (i.e. Random Forests), Platt scaling and isotonic regression are shown on the x-axis in Figure 2. The proportion of benign test objects in a set of samples are shown on the y-axis. Alternatively, this axis can be interpreted as an empirical estimate of the probability of a sample being benign for a given score. With this in mind, we now discuss the four lines shown in Figure 2.

1. **Perfect Calibration:** This is the ideal aim in probabilistic prediction that we can aspire to achieve. If the underlying algorithm has perfection calibration, then the proportion of benign samples among the objects with score s' should be s'.
2. **Random Forests:** After obtaining the scores from the underlying algorithm (Random Forests), we divided them into buckets of width 0.05 and calculated the proportion of benign samples in each bucket. For example, if the `predict_proba` function tell us that there are five samples with scores between [0.10, 0.15] and three of them are benign, we have a green cross at (0.125, 0.60). Note that the x-value is the midpoint of the range [0.10, 0.15]. It may be seen that the probability scores computed by the underlying algorithm case is far from ideal as the curve formed by the green

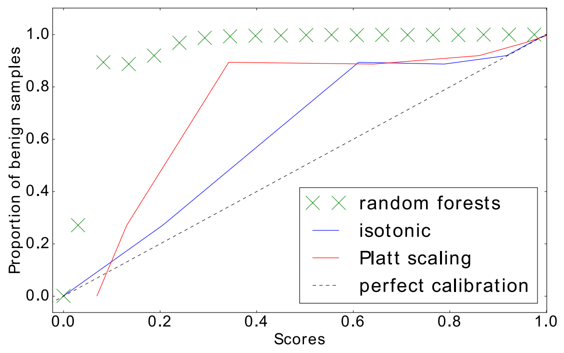


Figure 2: A comparison of the Random Forests, isotonic regression and Platt scaling probability scores against the observed benign proportions corresponding to each score. The Random Forest scores were placed into bins of width 0.05 and plotted against the benign proportion observed in each bin.

crosses is far from the $y = x$ line which represents perfect calibration.

3. **Platt Scaling:** In order to understand whether Platt scaling is able to fit the predicted scores from the underlying classifier, we fit the sigmoidal function as outlined in §3.2. The red line plotted shows the resulting probability prediction against the appropriate observed proportions of benign samples. Although Platt scaling offers an improvement over the underlying algorithm, the resulting predictions are still quite far away from being perfectly calibrated.

4. **Isotonic Regression:** Finally, we performed isotonic regression on the scores predicted by the underlying algorithm. The blue line plotted shows the relation between the probabilities output by isotonic regression against the corresponding observed probabilities (i.e. proportion of benign samples associated with each score). It is clear that the isotonic regression line produces relatively well calibrated results compared to the other methods since it agrees much more closely with the line for perfect calibration.

To summarize, we hope to emphasize two main points from Figure 2. The first is that there is a need for calibration methods such as isotonic regression and Platt scaling. This is clear from the fact that the output probabilities of Random Forests are poorly calibrated as seen in Figure 2. The next point is that Platt scaling is not always the best choice for calibrating probabilities and that isotonic regression can often produce better results.

4. INDUCTIVE VENN ABERS PREDICTORS

In this section we introduce the use of Venn-Abers predictors for measuring the confidence of a classification decision. For this, we first introduce the notion of loss functions, and then provide a formal representation of Inductive Venn-Abers predictors (IVAP). Finally, we discuss applying IVAP probabilities to assess loss in performance.

4.1 Loss Functions

A standard assessment method for probabilistic predictors is the use of loss functions. Two popular choices of loss function are log loss and Brier loss. The log loss of a probability prediction p whose true label is y is given by the expression

$$-y \log p - (1-y)\log(1-p) \tag{5}$$

where the log uses a base of 2. It is clear that a low probability p is punished with a high loss when the true label is 1 and a high probability is punished heavily when the true label is 0. This loss function also punishes undesirable probabilities near 0.5. Given a sizeable test set, the average loss incurred can be used to compare the performance of different probabilistic predictors.

4.2 Formal Description

Inductive Venn-Abers predictors build on the use of isotonic regression to produce perfectly calibrated results. However, perfect calibration comes at the cost of multi-probability output. In particular, IVAP returns the pair (p_0, p_1) where p_0 and p_1 are two different predictions of the probability that the true label of a test object is 1. A more precise account of IVAP's theoretical guarantee of perfect calibration is that just one out of p_0 and p_1 is perfectly calibrated. Sadly, there is no practical way of deciding which of the two it is. Despite this slight disadvantage, a single probability can still be extracted by minimizing the maximum expected value of some loss function (e.g. log loss or Brier loss). Using this minimax strategy, the single probability obtained is expected to be well calibrated if p_0 and p_1 are close in value.

Suppose we are given a full training set $\{z_1, z_2, \dots, z_n\}$ where $z_i = (x_i, y_i)$. Here, $x_i \in X$ represents an object and $y_i \in \{0, 1\}$ a label. Let x without a subscript represent a test object for which we want to predict the probability that its label is 1. The method of IVAP proceeds as follows:

- Split the training data into two sets: the calibration set $\{z_1, \dots, z_k\}$ and the proper training set $\{z_{k+1}, \dots, z_n\}$.
- Now train the scoring classifier on the proper training set to get a scoring function $s : X \to \mathbb{R}$.
- For $i = 0, 1$: let $g^{(i)} : \mathbb{R} \to [0, 1]$ be the isotonic regression on the points $(s(x_1), y_1), \dots, (s(x_k), y_k), (s(x), i)$.
- Output $(p_0, p_1) = (g^{(0)}(s(x)), g^{(1)}(s(x)))$.

It has been shown that after the scoring algorithm has been trained and the scores computed, IVAP can be implemented in $O(k \log k)$ steps [34] where k is the number of objects in the calibration set. Therefore, the computational cost of IVAP will often be determined by the efficiency of training the underlying scoring algorithm and calculating the scores on the calibration set.

4.3 Using the Probabilities

Our contribution utilizes the multi-probability output of IVAP in two different ways. In particular, we propose two different metrics for detecting a loss in accuracy of an underlying algorithm:

- `APW`: The Average Probability Width (APW) between the two different IVAP predictions $|p_0 - p_1|$ over a test set.

- `MAD`: The Mean Absolute Deviation (MAD) from 0.5 i.e. average $|p-0.5|$ over the test set where $p = p_1/(1-p_0+p_1)$.

The first metric (`APW`) considers the quantity $|p_0 - p_1|$ which can be interpreted as a measure of the uncertainty of a probability prediction. The intuition behind using this metric is that a large uncertainty corresponds to an unusual test sample that IVAP and the underlying algorithm struggle to process confidently. Therefore, we expect an increase in `APW` to indicate a loss in accuracy of the underlying classifier. The second metric (`MAD`) utilizes the absolute deviation of a single probability prediction p from 0.5. The formula for p given above corresponds to the minimax solution to the log loss function [33]. The idea behind `MAD` is that as accuracy of the underlying algorithm decreases, we would expect probabilities to move towards 0.5. As a result, we would expect a loss in accuracy of the underlying classifier to correspond to a reduction in `MAD`.

An empirical investigation of our ideas is given in §5.3. It should be noted that while we are aware that the theoretical guarantee of perfect calibration may be violated given a heavily drifting data population, our empirical findings show that these metrics can still provide insight into coping with the issue of concept drift.

5. EXPERIMENTS

Our experimental evaluation is mainly tailored to Android malware classification due to its rapid proliferation over the last years [29]. However Venn-Abers predictors can also be used in the desktop arena. In this section, we first describe the baseline detector and the datasets used in our work. Next, we experimentally confirm that Venn-Abers can be used to assess concept drift, and we evaluate the performance and complexity of our framework. Finally, we discuss the retraining strategies derived from our results.

5.1 Baseline Detector

To evaluate how well our probabilistic predictor performs we rely on DroidSieve [28], an Android malware classifier. DroidSieve uses static analysis to derive a number of features known to be characteristic of Android malware. These features are then used to fit a model capable of accurately discriminating goodware from malware. DroidSieve identifies two major classes of features: *syntactic* ones, which are derived from the code and metadata of the app; and *resource-centric* ones, which are obtained from resources used by the app. For the former, instances of the type of features extracted are API calls, or permissions used by the app. As for the latter, DroidSieve performs deep inspection of some resources such as certificates, or embedded native ELF executables to extract another set of features. Overall, we use both binary and continuous features. The presence or absence of a particular trait, such as a permission, is encoded as a binary feature; numeric properties, such as string lengths or op code frequency, are encoded as continuous features.

DroidSieve uses feature selection to restrict the classifier to important discriminating features. A feature is selected when the importance score assigned by the classifier is higher than the mean of all the features' scores. DroidSieve's architecture supports a variety of learning algorithms. For our experiments, we considered Extra Trees, Random Forests, and eXtreme Gradient Boost (XGBoost), and Support Vector Machines (SVM). While SVM is based on separation using hyperplanes and is less sensitive to outliers, the other classifiers are based on ensemble tree. The advantages of ensemble classifiers are that they can be parallelized giving significant speed-ups when dealing with large datasets. Additionally, ensemble tree-based classifiers usually obtain better predictive performance than a single classifier because they make the decision boundary smoother.

As for the choice of the baseline detector, we use DroidSieve because it performs consistently well when dealing with obfuscated malware [28]. However, it is worth noting that our framework is agnostic to the underlying baseline detector; at this point, we emphasize that the goal of this paper is to evaluate the presence of concept drift.

5.2 Datasets

The evaluation is based on a number of datasets containing both real-world malware samples and a set of goodware apps. For the malware, we rely on two well known datasets called MalGenome [36] and Drebin [5] containing a total of 5.5K unique samples. We further extended this with an additional set of 24.3K malwares given by Marvin [16] and McAfee. Similarly, we obtained about 107K goodwares also from Marvin and McAfee. The summary of the datasets used in our evaluation is described in Table 2a.

We split our datasets into three different modes: training, calibration and testing. For training and calibration, we retained 18.67% and 9.34% of the samples respectively; while for testing we retained the remaining 71.99% of the samples. For training and calibration we use Drebin, MalGenome and Marvin malware, whereas for testing we included the entire set of malware given by McAfee. On the one hand, the Drebin, MalGenome and Marvin malware sets are mainly samples found and vetted by security researchers and experts prior to 2013. On the other hand, the McAfee malware dataset are samples found by this Anti-Virus (AV) vendor between the first quarter of 2012 and the first quarter of 2016. Table 2b shows a summary of the dataset combinations and describes the number of samples in each mode.

The rationale behind this splitting is to keep some historical coherence in the selection of training and test datasets as this has a great impact on the performance of the detector [3]. We further aimed at obtaining a clean snapshot of the perception of an AV vendor that would go live in late 2011 and would span its activities up until 2016. The date span provides a good standpoint to evaluate the concept drift suffered by an AV vendor over four years.

Both the training and calibration modes contain the same ratio of malware to goodware, i.e.: one goodware per malware (1 : 1). Although the occurrence of malware in official markets is lower than the presence of goodware, undersampling the training set is a common practice to equally weigh both classes when building the classifier [22, 7]. For goodware, we use a similar splitting as malware except that we included in the testing set the remainder of the benign samples from Marvin after building the testing and calibration sets. This enabled us to test our model with higher ratio of 1 : 5 which is similar to other works in the area [37, 1, 6].

5.3 Experimental Results

In this section, we discuss a set of experiments to confirm that Venn-Abers can be used to assess concept drift in malware detection. In particular, we evaluate the quality of

ID	Dataset Name	Type	Samples
—	Drebin [5]	Malware	5,551
MgMW	MalGenome [36]	Malware	1,260
McGW	McAfee	Goodware	8,040
McMW	McAfee	Malware	13,777
MvGW	Marvin [16]	Goodware	84,980
MvMW	Marvin [16]	Malware	10,582
Summary		Malware	31,170
		Goodware	93,020

(a) Dataset sources

Mode	Type	Datasets	Ratio	Samples
Training	Malware Goodware	{Drebin, MgMW, MvMW} {MvGW}	1 : 1	23,191
Calibration	Malware Goodware	{Drebin, MgMW, MvMW} {MvGW}	1 : 1	11,595
Testing	Malware Goodware	{McMW} {MvGW, McGW}	1 : 5	89,404
Hold-out Ratio: 18.67% Training – 9.34% Calibration – 71.99% Testing				

(b) Dataset split

Table 2: Overview of chosen datasets. Figure 2a shows the source of our datasets together with the number of samples from each source, and Figure 2b shows the dataset splits and the ratio between malware and goodware used. The holdout ratio shows the overall percentage of samples retained for training, calibrating and testing.

Classifier Name	APW	MAD
XGBoost	−0.872	0.877
Extra Trees	−0.919	0.900
SVM	−0.909	0.922

Table 3: The Spearman's rank correlation coefficients of `APW` and `MAD` for each of the plots shown in Figures 3-4.

a given underlying classification algorithm by utilizing the multi-probability output of IVAP (see §4). To this end, we report our results using two different metrics over the test set: `APW` and `MAD` (see §4.3).

Overview. For our experiments, we train and calibrate our classifiers with the datasets described above. Then, we use the testing set to compute both `APW` and `MAD` for the samples observed in each quarter of a year from 2012 up to May 2016 (Q1). Note that we rely on McAfee's *first seen* (i.e., first detected) date to build the historical time-line of the malware. The average number of malware samples per quarter is about 800 (see Appendix A for details on the distribution of samples). Figure 3 shows the relation between the average recall and the `APW` for each quarter. Similarly, Figure 4 shows the average recall against the `MAD`. For the sake of clarity, we depicted the best fit (either linear or quadratic) to highlight how our metrics progress with respect to the recall. For the case of `APW`, we can observe that the average $|p_0 - p_1|$ decays as the recall improves. On the contrary, for `MAD` we can see that $|p - 0.5|$ increases as the recall improves.

To further reinforce the relationships shown in the plots, Spearman's rank correlation coefficients have been calculated. To briefly recap, Spearman's rank lies between the values $+1$ and -1. On the one hand, values near $+1$ indicate a strong increasing relationship between our metric and the recall. Note that the relationship is not necessarily linear. On the other hand, values near -1 indicate a strong decreasing relationship. The Spearman's rank correlation coefficients for each of the plots are included in Table 3.

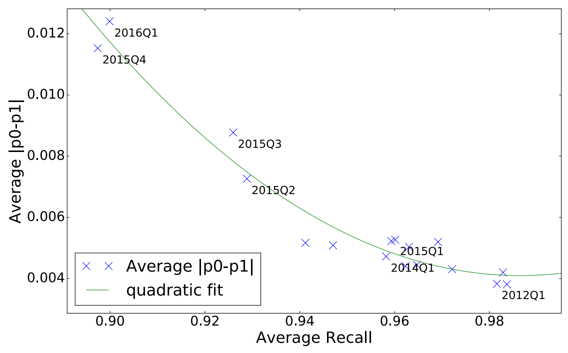
It is worth mentioning that the temporal order of the plotted points is expected to start from the rightmost side of the graph (i.e., with high recall) to the leftmost one (i.e., with lower recall) following a temporal order throughout the quarters. However, we found that the average recall did not strictly decrease with each quarter. This could be attributed to the way malware families evolve. Malware campaigns do not usually follow a continuous timeline and specimens that appear at a given time could disappear and then come back to the markets with new updates a few months later. Having said this, the lowest 4 values of recall do correspond to the last two quarters considered. Additionally, the 3 points with the highest recall correspond to the first three quarters in the case of XGBoost. This means that overall there is a decreasing trend in recall over time.

In total, we tested our metrics against samples from 17 quarters, showing a wide range of recall results with at least 0.1 points of difference. In the best case scenario, our recall lies near 100% accuracy over the unseen testing samples. The lower bound found in the worst case is about 76% recall. Altogether, this provides a baseline set of drifting models to study the role of `APW` and `MAD` metrics which is discussed next in detail for each of the baseline classifiers used.

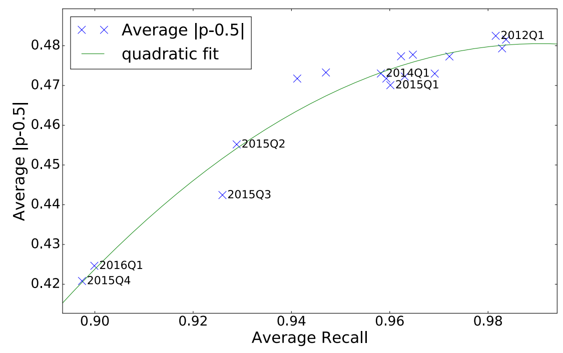
XGBoost. For the case of XGBoost we can observe that both `APW` and `MAD` closely predict the decreasing performance trend as shown in Figure 3a and Figure 4a. In particular, `APW` reports a quadratic decreasing trend between the average $|p_0 - p_1|$ score and the recall. This fits extremely well with our hypothesis that the uncertainty in our probability predictions increases as our accuracy decreases. On the other hand, `MAD` shows a similarly strong—but increasing—relationship between $|p - 0.5|$ and the recall. In both cases, the trend can be modeled quadratically and, interestingly, there appears to be an element of symmetry between the two models.

It is important to emphasize at this point that both `APW` and `MAD` can be calculated in the absence of true labels or *ground truths*. Therefore, these quantities can be used to estimate a loss in accuracy when feeding our underlying algorithm with unseen and unclassified samples.

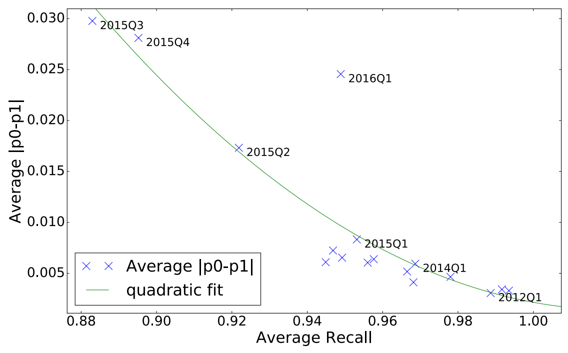
Extra Trees. When assessing the performance reported by the Extra Trees classifier, we found that only one quarter was predicted poorly as depicted in Figures 3b and 4b. Interestingly, this point corresponds to the 1st quarter of 2016. When considering this quarter as an outlier, we can observe a similar trend to the one reported for XGBoost albeit with different values. In the particular case of the `MAD`, the points seem slightly more scattered from the prediction line indicating a looser trend. Having said this, the Spearman's rank correlation coefficients are similar to those observed for XGBoost indicating a monotonic trend for both metrics.



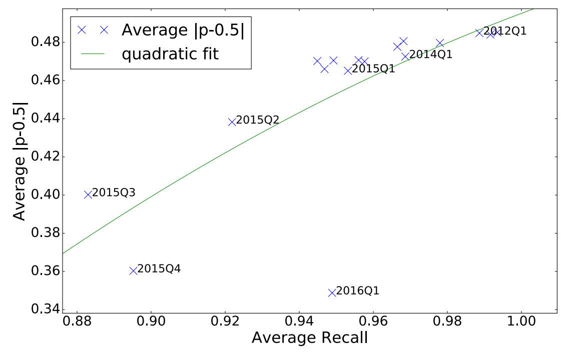
(a) XGBoost



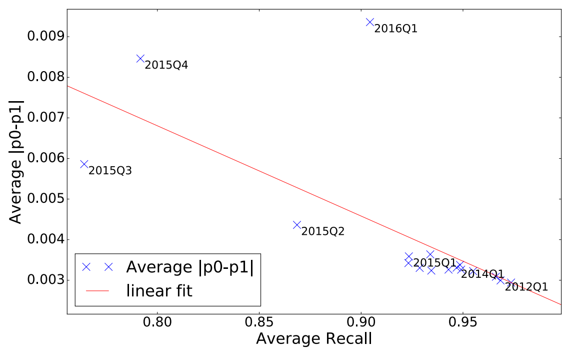
(a) XGBoost



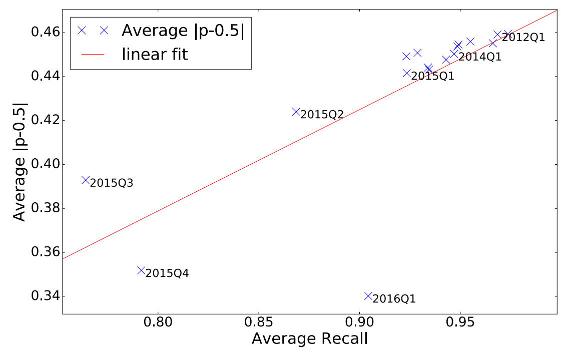
(b) Extra Trees



(b) Extra Trees



(c) SVM



(c) SVM

Figure 3: Average probability width $|p_0 - p_1|$ (i.e., APW) against average recall for the three underlying algorithms. Each data point refers to a quarter in 2012–2016.

Figure 4: Average deviation $|p - 0.5|$ (i.e. MAD) against average recall for the three underlying algorithms. Each data point refers to a quarter in 2012–2016.

	Scores		IVAP	
Algorithm	**Train**	**Test**	**Train**	**Test**
XGBoost	17	0.41	0.42	0.41
Extra Trees	56	0.71	0.71	0.71
SVM	500	79	79	80

Table 4: Running times (in *milliseconds* per sample) of each underlying algorithm together with the assessment time taken by our IVAP-based framework.

SVM. For the case of SVM, we can observe from the results that the number of outliers is slightly higher (see Figures 3c and 4c). We can observe that the range of probability is smaller than those observed in the other classification algorithms and there is a wider range of recalls present. In this case, we found that linear regression provided a more appropriate fit to the plotted data in comparison to a quadratic curve. Apart from that, the results are somewhat similar to the case of Extra Trees. More specifically, there is one quarter (quarter 1 of 2016) that breaks the trend with a recall of about 0.90. This confirms that this data point is more difficult to model and suggests that further analysis could lead to strong alternatives to `APW` and `MAD`. Spearman's rank correlation coefficients for this algorithm are still good at -0.909 for `APW` and 0.922 for `MAD`, showing that there is evidence of a monotonic relationship present in the data.

5.4 Performance and Complexity

We now give an account of the computational cost associated with our approach. Suppose we have a calibration set of size k. It has been shown that the theoretical complexity of the training phase of IVAP is $O(k \log k)$ and that the probability calculation for each sample can be done in $O(\log k)$ steps [34]. In order to see how this translates into practice, computation times were collected during our experiments. Experiments were run on a system with an Intel Core i5 processor and 8GB RAM.

In all cases, the computation time associated to IVAP was significantly shorter than the time taken to train the underlying algorithm. Table 4 shows a summary of the results. The second column in the table corresponds to training on 23,191 samples and testing on 4,502 test samples. whereas the third column corresponds to training IVAP on a calibration set of size 11,595 and outputting 4,502 probability predictions on the test set.

On the one hand, the time taken to train the underlying algorithm is about 392 seconds in total ($1.7 \cdot 10^{-2}$ seconds per sample) for XGBoost, and 1298 seconds ($5.6 \cdot 10^{-2}$ seconds per sample) for Extra Trees. On a different order of magnitude, SVM takes about 11496 seconds ($5.0 \cdot 10^{-1}$ seconds per sample). Contrary, the time taken to process the test samples is negligible for the tree classifiers at a total of 2 seconds for XGBoost and 3 seconds for Extra Trees, but significantly longer at a total of 354 seconds ($7.9 \cdot 10^{-2}$ per sample) for SVM.

On the other hand, results show that training IVAP takes just 5 seconds in total ($4.2 \cdot 10^{-4}$ seconds per sample) for XGBoost, 8 seconds ($7.1 \cdot 10^{-4}$ seconds per sample) for Extra Trees, and 915 seconds ($7.9 \cdot 10^{-2}$ seconds per sample) for SVM. IVAP also outputs probabilities on test samples efficiently taking a total of 2 seconds ($4.1 \cdot 10^{-4}$ seconds per sample) for XGBoost and 3 seconds ($7.1 \cdot 10^{-4}$ per sample) for Extra Trees. The total time taken to calculate the probabilities over the test set for SVM was longer at 360 seconds ($8.0 \cdot 10^{-2}$ seconds per sample).

In general, computing multi-probability predictions for SVM is significantly more expensive than the other underlying classifiers. However, this difference is due to the computational cost of calculating the scores associated with SVM and should not be attributed to the deployment of IVAP.

5.5 Retraining Strategies

Suppose we have decided on an unacceptable recall threshold R_*. One way of deciding when to retrain is to find the corresponding `APW` and `MAD` thresholds based on the model obtained during the calibration of our IVAP-base framework, i.e., use either the linear or the quadratic fit that shapes the regression. Note that this type of threshold can be violated by recalls that are higher than R_* and can easily be respected for recalls that are lower than the given threshold. Therefore, this strategy can only offer a rough guideline. In a setting where the analyst is willing to ignore the seemingly anomalous result (2016Q1), the model built can better fit the remaining points and the retraining strategy would appear to be more effective.

Nevertheless, some values of R_* have better cut-off thresholds depending on the metric used for the assessment (i.e., `APW` or `MAD`). For example, if we chose to use XGBoost with $R_* = 0.93$ a natural threshold for `APW` would be around 0.006, whereas the `MAD` threshold would be 0.465. Having access to different metrics enables us to customize our approach further. In scenarios where the retraining burden is high or a lower recall is not detrimental, we can decide to retrain only when *both* thresholds are breached. This strategy can lead to less retraining tasks at the expense more concept drift. On the contrary, if retraining is inexpensive we can decide to retrain based on the most conservative metric to minimize performance degradation.

6. RELATED WORK

Several approaches for automatically *detecting* malware have been presented in the literature, which typically use *static* and/or *dynamic* analysis to extract features from the programs under analysis. Once these features are extracted, many techniques can assist the analyst in detecting the malware, including machine learning [13], data mining [30], expert systems [23], and clustering [9]. In the past, SVM and Random Forest have been successfully applied to malware detection [27] and they have been shown to have better performance than others after comparing them to 180 classifiers on various datasets [10]. Ensemble tree-based classifiers perform well on many real world settings, however. For example, Extra Trees [11] and Gradient Tree Boosting [12] have been achieving great performance in most of recent "*Kaggle*" competitions [2]. Most of these techniques were first introduced for malware on desktop systems and then were adopted for mobile malware as smartphones become the platform of choice for malware developers [29].

In the Android realm, machine learning has been widely used for malware detection [5, 16]. Drebin [5] is a lightweight detection method that uses static analysis to gather the most important characteristics of Android applications such as permissions, API calls, and network addresses declared in clear text. It uses Support Vector Machines (SVM) to detect whether a given sample is malicious or benign. Marvin [16] shows how the combination of static and dynamic analysis can improve the detection rate as well as reduce the number of false positives. It uses a number of statically extracted features and combines them with additional dynamically extracted features, overall more than 490,000. Moreover, it leverages machine learning to detect malware as well as providing a risk score associated with a given unknown sample. Madam [25] proposed a host-based malware detection system analyzing features at four levels: kernel, application, user and package. It derived features such as system calls, sensitive API calls and SMS through dynamic analysis while complementing these with statically derived features such as permissions, the app's metadata and market information. More recently, DroidScribe [8] uses SVM with selective invocation of Conformal Prediction [32] by statistically evaluating SVM classifications [14] to generate prediction sets for malware family identification.

The main weakness of machine learning-based approaches is that resulting classification models often change over time becoming less accurate as malware evolves. Therefore, a retraining strategy that would adapt to drifting scenarios such as the one present in Android malware detection is necessary [3]. However, despite the rapid proliferation of malware in Android [29], none of the recent works has looked into the retraining conundrum. Authors in [18] suggested an online learning architecture to deal with this problem but it still remains unclear how concept drift can be predicted in practice. In other domains, related works [26] leverage on studying of the concept drift in a broad and general manner. For instance, Singh *et al.* [26] propose two different measures to track concept drift in malware family identification based on the similarity of feature vectors from different time periods. One major limitation of Singh's work is that adapting and adopting similarity-based measures to the malware detection problem might not work for large datasets. Furthermore, our work goes one step further by using probabilistic guidance to predict nonstationary populations.

7. FUTURE DIRECTIONS

We have empirically explored the use of a special class of probabilistic predictor. We envisage this work to develop into a mature framework for both decision assessment and as a paradigm for automatic retraining. In order to do this, we intend to build on this initial work to move to evaluating our work in an operational setting which will require an extensive evaluation of four verticals: versatility, alternatives, metrics and assessment epochs. These are discussed below.

Versatility. A major drawback of Venn-Abers predictor is that it is currently applicable only to the 2-class problem which inhibits its use in a wide variety of domains. For example, a key necessity in this domain is identifying the family of a malware. Family identification is central to threat mitigation strategies for malware containment. This multi-class problem renders Venn-Abers predictor unusable in its current form. Our immediate focus would be to extend IVAPs to the identification of families, and other linkage associations. Similar to conventional approaches to multi-class classification, extending this work from a 2-class IVAP to an n-class IVAP could be achieved by using a *a one vs one* or *one vs all* approach. As an extension to this work, we intend to study the theoretical and practical feasibility of using IVAP for the multi-class case. This would greatly enhance the practical impact of our research.

Alternatives. In order to project IVAP as a practical solution for decision assessment and retraining indicator, we intend to perform a thorough evaluation with other decision assessment algorithms. For example, other probabilistic predictors related to IVAP have been shown to perform extremely well according to loss assessment [34]. In particular, cross Venn-Abers are a combination of K IVAPs that can potentially provide better assessments of nonstationary populations of malware that introduce concept drift. Although, these types of predictors are computationally inefficient, we believe that there might be situations where the overhead introduced might be tolerable; especially if it leads to better assessments. The more standard predictors (§3.2) should also be considered in future work. Additionally, we also intend to evaluate the usefulness of other metrics non-related to IVAP, such as Conformal Evaluator [14]. Such metrics provide explanatory evidence on whether a sample belongs to a class. When combined with a first-cut evaluation by IVAPs, such techniques can provide a complementary means of assessment to precisely identify concept drift, assist a human analyst and model retraining.

Metrics. In this work, we have suggested two different metrics for predicting losses in accuracy. Although the results are shown to be promising, additional efforts could be translated into new retraining strategies. For instance, we are currently computing our metrics based on the entire testing set. Whether a subset of these samples could improve the assessment is currently unknown and we intend to investigate this. The selection of this subset could be guided by a search heuristic. For example, it is likely that selecting the samples for which the output probability is the least conclusive could provide a different notion for detecting concept drift. This can ultimately result in a better retraining strategy.

Assessment Epochs. We intend to further extend the dataset in order to try different time-lines with varying levels of granularity. This poses a significant challenge as there are no large repositories of history goodware publicly available for which the ground truth is guaranteed. Unfortunately, retrieving goodware from Google Play is insufficient for our work as we have restricted access to earlier versions of a sample which would make the time-line of samples incoherent.

8. CONCLUSION

We have demonstrated that IVAPs can be used for assessing the quality of malware detection algorithms in the presence of concept drift. We showed that well calibrated probabilities can be used to detect potential loss in detection accuracy. We also proposed novel metrics to identify obsolete models and evaluated the metrics using a large-scale operational setting and presented a framework that can be used to establish an adequate retraining strategy to minimize performance degradation in real-world scenarios.

We further discussed how probabilistic predictors can help to minimize operational costs, and we made a critical assessment of probabilistic predictors for decision assessment. We studied the performance and complexity of our framework and showed that our methods are practical and efficient. Finally, we have presented a number of future directions that can contribute to and foster research in the area.

Acknowledgments

This research has been partially supported by the UK EPSRC grants EP/K033344/1, EP/L022710/1, and EP/K035584/1.

APPENDIX

A. DISTRIBUTION OF MALWARE

We rely on McAfee's *first seen* date to build the historical time-line of the malware. The distribution of samples per quarter ranges from 23 to 5,182 samples as shown in Figure 5. The average number of malware samples per quarter is about 800. In general, the number of samples seen every quarter changes following a natural variation as new specimens are added to the knowledge base of the Antivirus Vendor. Note that in mid 2012 smartphones became the platform of choice for malware developers [29] and the number of samples increases notably in the subsequent quarters.

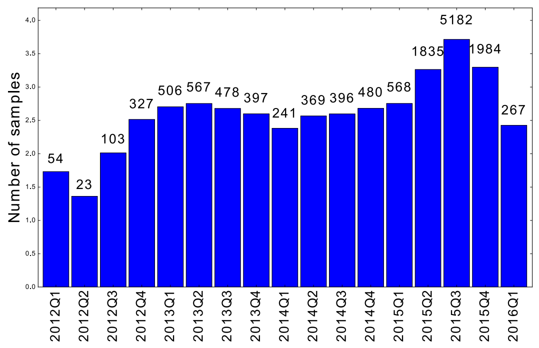


Figure 5: Distribution of malware samples per quarter from 2012 to 2016. The bars are plotted in logarithmic scale to highlight the variation of samples across quarters.

B. REFERENCES

[1] Yousra Aafer, Wenliang Du, and Heng Yin. Droidapiminer: Mining api-level features for robust malware detection in android. In *International Conference on Security and Privacy in Communication Systems*, pages 86–103. Springer, 2013.

[2] Mansour Ahmadi, Dmitry Ulyanov, Stanislav Semenov, Mikhail Trofimov, and Giorgio Giacinto. Novel feature extraction, selection and fusion for effective malware family classification. In *ACM Conference on Data and Application Security and Privacy (CODASPY)*, pages 183–194, 2016.

[3] 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.

[4] Brandon Amos, Hamilton Turner, and Jules White. Applying machine learning classifiers to dynamic android malware detection at scale. In *2013 9th international wireless communications and mobile computing conference (IWCMC)*, pages 1666–1671. IEEE, 2013.

[5] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, and Konrad Rieck. Drebin: Effective and explainable detection of Android malware in your pocket. In *Network and Distributed System Security Symposium (NDSS)*, 2014.

[6] Vitalii Avdiienko, Konstantin Kuznetsov, Alessandra Gorla, Andreas Zeller, Steven Arzt, Siegfried Rasthofer, and Eric Bodden. Mining apps for abnormal usage of sensitive data. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 1, pages 426–436. IEEE, 2015.

[7] Wei Chen, David Aspinall, Andrew D. Gordon, Charles Sutton, and Igor Muttik. More semantics more robust: Improving android malware classifiers. In *Proceedings of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, WiSec '16, New York, NY, USA, 2016. ACM.

[8] 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 *Mobile Security Technologies (MoST)*, volume 7148, pages 1–12, 2016.

[9] Sarah Jane Delany, Mark Buckley, and Derek Greene. Sms spam filtering: methods and data. *Expert Systems with Applications*, 39(10):9899–9908, 2012.

[10] Manuel Fernández-Delgado, Eva Cernadas, Senén Barro, and Dinani Amorim. Do we need hundreds of classifiers to solve real world classification problems? *The Journal of Machine Learning Research (JMLR)*, 15(1):3133–3181, January 2014.

[11] Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine Learning*, 63(1):3–42, 2006.

[12] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Springer, New York, 2 edition, 2009.

[13] Yung-Tsung Hou, Yimeng Chang, Tsuhan Chen, Chi-Sung Laih, and Chia-Mei Chen. Malicious web content detection by machine learning. *Expert Systems with Applications*, 37(1):55–60, 2010.

[14] Roberto Jordaney, Zhi Wang, Davide Papini, Ilia Nouretdinov, and Lorenzo Cavallaro. Misleading metrics: On evaluating machine learning for malware with confidence. Technical report, Royal Holloway, University of London, 2016.

[15] Antonis Lambrou, Harris Papadopoulos, Ilia Nouretdinov, and Alexander Gammerman. Reliable probability estimates based on support vector machines for large multiclass datasets. In *Artificial Intelligence Applications and Innovations: AIAI 2012 International Workshops: AIAB, AIeIA, CISE, COPA, IIVC, ISQL, MHDW, and WADTMB, Halkidiki, Greece, September 27-30, 2012, Proceedings, Part II*, 2012.

[16] M. Lindorfer, M. Neugschwandtner, and C. Platzer. Marvin: Efficient and comprehensive mobile app classification through static and dynamic analysis. In *Proceedings of the 39th Annual International Computers, Software & Applications Conference (COMPSAC)*, volume 2, pages 422–433, July 2015.

[17] Aditya Krishna Menon, Xiaoqian J Jiang, Shankar Vembu, Charles Elkan, and Lucila Ohno-Machado. Predicting accurate probabilities with a ranking loss. In *Machine learning: proceedings of the International Conference. International Conference on Machine Learning*, volume 2012, page 703. NIH Public Access, 2012.

[18] Annamalai Narayanan, Liu Yang, Lihui Chen, and Liu Jinliang. Adaptive and scalable android malware detection through online learning. *arXiv preprint arXiv:1606.07150*, 2016.

[19] Alexandru Niculescu-Mizil and Rich Caruana. Predicting good probabilities with supervised learning. In *Proceedings of the 22nd international conference on Machine learning*, pages 625–632. ACM, 2005.

[20] 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.

[21] 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.

[22] Sankardas Roy, Jordan DeLoach, Yuping Li, Nic Herndon, Doina Caragea, Xinming Ou, Venkatesh Prasad Ranganath, Hongmin Li, and Nicolais Guevara. Experimental study with real-world data for android app security analysis using machine learning. In *Proceedings of the 31st Annual Computer Security Applications Conference*, pages 81–90. ACM, 2015.

[23] Seda Sahin, Mehmet R Tolun, and Reza Hassanpour. Hybrid expert systems: A survey of current approaches and applications. *Expert Systems with Applications*, 39(4):4609–4617, 2012.

[24] Justin Sahs and Latifur Khan. A machine learning approach to android malware detection. In *Intelligence and Security Informatics Conference (EISIC), 2012 European*, pages 141–147. IEEE, 2012.

[25] A. Saracino, D. Sgandurra, G. Dini, and F. Martinelli. Madam: Effective and efficient behavior-based android malware detection and prevention. *IEEE Transactions on Dependable and Secure Computing*, PP(99):1, 2016.

[26] Anshuman Singh, Andrew Walenstein, and Arun Lakhotia. Tracking concept drift in malware families. In *Proceedings of the 5th ACM workshop on Security and artificial intelligence*, pages 81–92. ACM, 2012.

[27] Charles Smutz and Angelos Stavrou. Malicious pdf detection using metadata and structural features. In *28th Annual Computer Security Applications Conference (ACSAC)*, pages 239–248, New York, NY, USA, 2012. ACM.

[28] Guillermo Suarez-Tangil, Santanu Kumar Dash, Mansour Ahmadi, Johannes Kinder, Giorgio Giacinto, and Lorenzo Cavallaro. DroidSieve: Fast and accurate classification of obfuscated android malware. May 2016.

[29] Guillermo Suarez-Tangil, Juan E Tapiador, Pedro Peris-Lopez, and Arturo Ribagorda. Evolution, detection and analysis of malware in smart devices. *IEEE Communications Surveys & Tutorials*, 16(2):961–987, 2014.

[30] Sheela Thiruvadi and Sandip C Patel. Survey of data-mining techniques used in fraud detection and prevention. *Information Technology*, 10(4):710–716, 2011.

[31] Alexey Tsymbal. The problem of concept drift: definitions and related work. *Computer Science Department, Trinity College Dublin*, 106, 2004.

[32] Vladimir Vovk, Alex Gammerman, and Glenn Shafer. *Algorithmic learning in a random world*. Springer, New York, 2005.

[33] Vladimir Vovk and Ivan Petej. Venn–Abers predictors. In Nevin L. Zhang and Jin Tian, editors, *Proceedings of the Thirtieth Conference on Uncertainty in Artificial Intelligence*, pages 829–838, Corvallis, OR, 2014. AUAI Press.

[34] Vladimir Vovk, Ivan Petej, and Valentina Fedorova. Large-scale probabilistic predictors with and without guarantees of validity. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 892–900. Curran Associates, 2015.

[35] Bianca Zadrozny and Charles Elkan. Transforming classifier scores into accurate multiclass probability estimates. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 694–699. ACM, 2002.

[36] Yajin Zhou and Xuxian Jiang. Dissecting Android malware: Characterization and evolution. In *IEEE Symposium on Security and Privacy (SP)*, 2012.

[37] Yajin Zhou, Zhi Wang, Wu Zhou, and Xuxian Jiang. Hey, you, get off of my market: Detecting malicious apps in official and alternative android markets. In *NDSS*, 2012.