Malware Detection via Machine Learning

Abstract—

The use of supervised learning techniques for malware detection has been used increasingly to aid classical classification methods. In this work we aim at developing a malware detection model, analyzing the impact of the reliability of the training dataset on the final result of the classifier, and metrics to define the *ground-truth*. For this, we propose three datasets' scenarios whose content range from unambiguous malware and goodware samples to more ambiguous and real ones. We analyze each scenario in laboratory conditions, where standard cross-validation methodologies are applied, discarding the importance of *time* in malware detection, and also in real-world conditions, where temporal-based dependencies are proposed and applied. Furthermore, we modify our original model to both enrich the extractable information, by implementing a multi layer model, and to improve the final results, by using dynamic information about the samples. We then use our temporal-based methodologies to reduce the size of the training dataset without compromising optimal results, concluding that there exists an ideal number of necessary training folds, temporally consistent with the validation fold, that maximizes the overall score.

Index Terms—**Security, Malware Detection, Machine Learning, Temporal Consistency, Ground Truth**

I. Introduction

The number of reported security breaches due to *virus*, *worms*, *trojans*, etc, has been growing considerably in recent years [1] with reports of infections due to *malware* making the headlines, now more than ever. Almost every week one such security vulnerability is reported which may be seen as a failure by the security community on the control and detection of malicious content.

The very first challenge that we face is the definition of *malware*. Malware is considered to be "a program with malicious intent" [2] which in itself is a dubious definition. Not only the same programs are classified differently as *malware* and *goodware* depending on the vendor, but also some programs fall within a gray area for which no clear classification can be deemed correct. An example is what is called *adware*, advertising-supported software, that although not performing directly malicious actions, perform arguably non-requested actions. The non-existence of concrete metrics and properties that uniquely distinguish malware from goodware requires an extra effort in the preparation of datasets for evaluation of malware detectors.

Due to the significant number of malware attacks, and taking into consideration the increasing popularity and huge success of Machine Learning (ML) methodologies in classification in different domains [3]–[6], it is only natural to see these techniques applied to complement classical methodologies for malicious-content detection [2], [7]–[21], in particular, supervised learning techniques. However, several issues have arised regarding the usage of these ML techniques in the scope of malware analysis [18]–[21].

On the sometimes ambiguous scenario that is the task of distinguishing malware from goodware, some of these results largely depend on the datasets used for evaluation, often not representing the real-world. In this paper we address this pertinent questions and make a comparative analysis of a supervised learning approach in three different scenarios (depicted in Figure 1): a *strict scenario* where only very well-characterized samples are considered, a *loose scenario* where a wider set of still well-studied samples is considered, and finally a *realistic scenario* where we get closer to the reality faced by vendors of malware detection solutions.

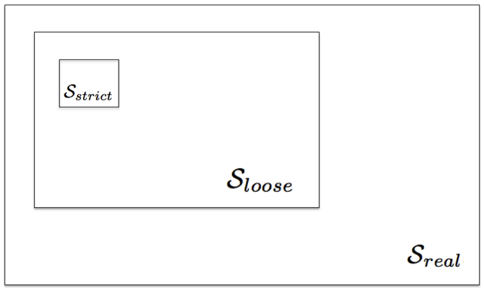

Fig. 1. Representation of our $\mathcal{S}_{strict}$, $\mathcal{S}_{loose}$ and $\mathcal{S}_{real}$ scenarios.

Throughout this work we will perform a comparative analysis of the three above scenarios under two distinct environments:

- *laboratory conditions* where traditional cross-validation methodologies can be applied;
- *real-world conditions* where time is relevant and we analyze the behavior of the classifier with temporal-based methodologies.

We show that without much tweaking, and using simple features, a Logistic Regression (LR) model is able to achieve an Area Under the Receiver Operating Characteristic (AUROC) of 0.91 when ideal laboratory conditions are met, but these results go down to an AUROC of 0.75 when we change into a real-world scenario.

With the purpose of actually evaluating the usage of ML methodologies for malware detection as a complement to traditional malware detection techniques, we also analyze how the size of the training set can influence the performance of the classifier under the three studied scenarios. For this, we use the aforementioned temporal-based methodologies to come up with conclusions on how large the training set need to be to ensure optimal results from the classifier.

We finish off by improving our base model to include more dynamic features, as well as providing a multi layer approach which adds the ability of classifying malware samples from different classes, in contrast with simply providing if a sample is malware or not. These improvements boost our AUROC to 0.98 in ideal laboratory conditions and 0.95 in a real-world scenario.

As the main contributions of this paper **a)** we propose three different scenarios to train and validate the model which range from a more simulated scenario to a more realistic one, **b)** temporal-based methodologies to train and validate the classifier, **c)** we study how much can we reduce the training dataset without compromising the optimal results, **d)** we improve our base model by 7% to 27%.

This paper is outlined as follows: in Section II we present the related work and justify how it motivated our work; in Section III we describe and analyze the available dataset and how it was labeled; in Section IV we propose a feature selection and describe the used model; Section V encloses our main contributions, describing our three scenarios and how they are to be evaluated, using regular cross-validation and our defined temporal-based methodology, together with the results of said evaluation; in Section VI we provide improvements to our base model; in Section VIII we discuss our main achievements; Section IX concludes the paper and discusses avenues for further research.

II. Related Work

We use this section to present prior work that closely relates to the topic of our research and our areas of contribution. Specifically, detecting malicious software by training supervised models on static information, and validation methodologies that resemble real-world conditions.

Shabtai et al. [22] provide a survey directed at the application of ML classifiers to detect malware from static features. Their work concerns the design and evaluation of such systems. Our contributions are inspired by the concern of how to correctly evaluate ML classifiers, regarding size, reliability of labeling metrics and chronological evaluation.

In the topic of methodologies that resemble real-world conditions, Srndić et al. [17] train and validate their malicious PDF detector under laboratory and real-world conditions. Laboratory conditions consist on applying regular cross-validation, whereas real-world conditions validate a newer dataset with a model created from outdated data (i.e. older then the validation), and also validate the model when the validation set spans one week and the training is gathered in the previous 4 weeks. They show that laboratory conditions inflate the results, when compared to real-world conditions.

Miller et al. [10] also introduce sample temporal consistency. They show the impact of performance measurement technique on a dataset containing 1.1 million samples, when using cross-validation and temporally consistent samples. As noted by others, regular cross-validation showed inflated results when compared to temporally consistent samples.

Our work enhances these methodologies by analyzing the performance variation when the distance between the training and validation set increases and decreases, as well as analysis on how reducing the size of the training without compromising the results.

Kolter et al. [9] learn to detect malicious executables in a dataset with under 4,000 samples, obtained from reliable sources, and evaluate their model under standard cross-validation and by gathering newer malware samples to validate for new and unseen samples. They show how the model provides optimal results under cross-validation, but for the unseen samples lower scores are obtained. Our work considers these results to compare how reliability affects performance.

Sebastin, M. et al. [23] develop *AVClass*, a tool that given a set of antivirus vendors, outputs the most likely family name. They test their tool under 10 datasets, totaling 8.9 M samples, with results showing an F1 measure up to 93.9 on labeled datasets. The tool takes as input the labels as seen in VirusTotal, tokenizes the labels, replaces known aliases and general names (e.g. win32, trojan, generic), ending with possible family names. These remaining names are counted and the most frequent one is given as family name. Our work takes advantage of this tool, not to label malware families, but minimal modifications, to label more general malware classes (e.g. trojan, virus).

Deo, A. et al. [18] focus over the problem of ML models becoming antiquated over time, given how malware evolves. This problem is seen as concept drift, where the performance of a model over time diminishes, as the statistical properties of malware (i.e. features) change over time. They study how probabilistic predictors can help minimize the aforementioned problem, by indicating when retraining of a given model is necessary.

Jordaney, R. et al. [20] also study the problem of concept drift in malware classification models, focusing on providing metrics, based on statistical comparison of samples, to detect when should a model be retrained.

Both [18] and [20] are on the subject of our work, regarding the problem of concept drift, but differ on the study-case. In our work we acknowledge the concept but focus on its relative effects and how one can balance the size of needed training data vs. validation data, when maintaining a temporal consistent dataset. Whereas their work provide indicators for when should a model be retrained when the problem of concept drift becomes significant.

III. Data Collection, Analysis and Labeling

Malware classification is non-deterministic task, as several nuances make vendors disagree on what they classify as malware. Not only subtleties related to programs such as *adware* or *remote management consoles*, for which one can find arguments to classify them in either class, but also because some vendors are more accurate than others when classifying malware.

In order to be able to study how *temporal consistency* and *ground truth* influences a model trained to detect malicious and non-malicious applications, we gathered a large dataset of publicly available samples that includes both *malware* and *goodware* with no a priori labeling.

In this section we will provide an overview of the collected data, in particular its sources, followed by an analysis on these samples, and a discussion on the cross-checking mechanisms we used to perform the labeling. In the end of this section we will provide 3 metrics for analysis that differ in the confidence one can provide on the samples' labeling.

A. Data Sources

To study how *temporal consistency* and *ground truth* influences a model trained to detect malicious and non-malicious applications, we started by gathering a dataset of malware and goodware. This dataset was obtained from *Malwr* [24], an online service that runs static and dynamic analysis on user submitted files using *Cuckoo Sandbox* [25]. Although Malwr accepts any kind of file, we are interested solely in Portable Executable (PE) files, not only because Windows' systems are relevant targets of malicious applications, but also due to the fact that our initial approach will focus on static information obtained from these PE files.

Malwr service provides the analysis of the submitted files as well as the MD5 of such files, but no labeling as whether a sample corresponds to a malicious or non-malicious application. To perform this labeling we use the anti-virus' signatures given by *VirusTotal* [26] at the time of analysis (incorporated in Malwr's reports), as a means of labeling the gathered reports. VirusTotal is an online service where one can submit a suspicious file and in return obtain a list with the analysis performed by a significant number of vendors on whether the sample is clean or not.

However, and as mentioned before, these classifications are not unanimous, and we still need to assign a labeling to each sample. For this reason, we enrich our knowledge regarding samples' ground truth by aggregating metadata from *National Software Reference Library (NSRL)* [27] and *VirusShare.com* [28], specifically the samples' MD5. NSRL contains a collection of digital signatures of known, traceable software applications, whereas VirusShare.com is a repository of malware samples, so a sample belonging to the NSRL set gives us a higher confidence that it is indeed goodware, whereas one belonging to VirusShare.com set gives us a higher confidence that it is malware.

In summary, we collected reports from the PE samples available in Malwr, and complemented it with metadata from NSRL and VirusShare. The following subsection quantifies our *corpus*, providing a better understanding of the available samples.

B. Data Analysis

We collected our data from Malwr between April 16th, 2013 and October 10th, 2016. Our data can be divided in three sets: **a)** a set $\mathcal{R}$ of raw reports, containing 388,702 PE samples; **b)** a set $\mathcal{C} \subseteq \mathcal{R}$ of 284,880 classified reports by 38 vendors ($\mathcal{V}$) that is obtained by restricting the original set $\mathcal{R}$ (that includes 97 vendors) to those reports whose vendors are present in at least 95% of the classified samples.

With regards to duplicated submissions, there are 27,798 samples submitted more than once, for a total of 74,916 duplicated submissions $\mathcal{C}_{dups}$ averaging 2.7 submissions per duplicate. Understanding how vendors change their signatures on samples is crucial, as we use it to label the dataset. With this in mind, and inspired by Miller et al. [10], we start our analysis by studying the differences between the number of positive (*i.e.* a sample of malware) and negative (*i.e.* a sample that is not malware) classifications over the last and first submission of the same sample.

For each duplicated sample in $\mathcal{C}_{dups}$ we counted the number of positive classifications on the first and last submissions, $\mathcal{P}_f$ and $\mathcal{P}_l$ respectively. If $\mathcal{P}_f > \mathcal{P}_l$ then we are looking at a possible *false positive (FP)*, as the number of vendors classifying the sample as malware decreased. Conversely, if $\mathcal{P}_l > \mathcal{P}_f$ we are looking at a potential *false negative (FN)*. For the case $\mathcal{P}_f = \mathcal{P}_l$ we conclude that vendors are confident regarding their classification for the sample.

Figure 2 shows the frequency of $\mathcal{P}_l - \mathcal{P}_f$ for each duplicated sample. We first note that 44.32% of duplicated samples change in classification, among which 38.72% increase its classification, whereas only 5.61% decrease. Such discrepancy between positive and negative changes suggest a preference for false negatives over false positives, as also noted by Miller et al. [10].

Another interesting analysis on our dataset is understanding the vendors *Detection Rate (DR)* (or True Positive Rate), and *False Postive Rate (FPR)*. Although these formulas are trivially defined respectively as

$$\mathrm{DR} = \frac{\mathrm{TP}}{\#malware} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}} \tag{1}$$

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\#goodware} = \frac{\mathrm{FP}}{\mathrm{TN}+\mathrm{FP}} \tag{2}$$

we lack ground truth for what is $\#malware$ and $\#goodware$. To solve this, we propose relative metrics to compute what is positive ($\#malware$) and negative ($\#goodware$).

Our first approach is to take advantage of duplicated submissions to define an accuracy metric, $\mathcal{M}_{acc}^{dups}$. As we have previously shown, 44.32% of duplicated samples change in classification, which can be translated into vendors acknowledging their own errors.

With that in mind, for each vendor $v \in \mathcal{V}$, we define a duplicated sample for v (according to $\mathcal{M}_{acc}^{dups}$) as:

- TP_v, *true positive for* v, if v classified it positively in both the first and last submissions;
- TN_v, *true negative for* v, if v classified it negatively in both the first and last submissions;
- FP_v, *false positive for* v, if v classified it positively in the first submission and negatively in the last submission;
- FN_v, *false negative for* v, if v classified it negatively in the first submission and positively in the last submission.

Figure 3 plots each vendors' DR_v vs. FPR_v. We note that vendors do acknowledge their classification errors, as we see

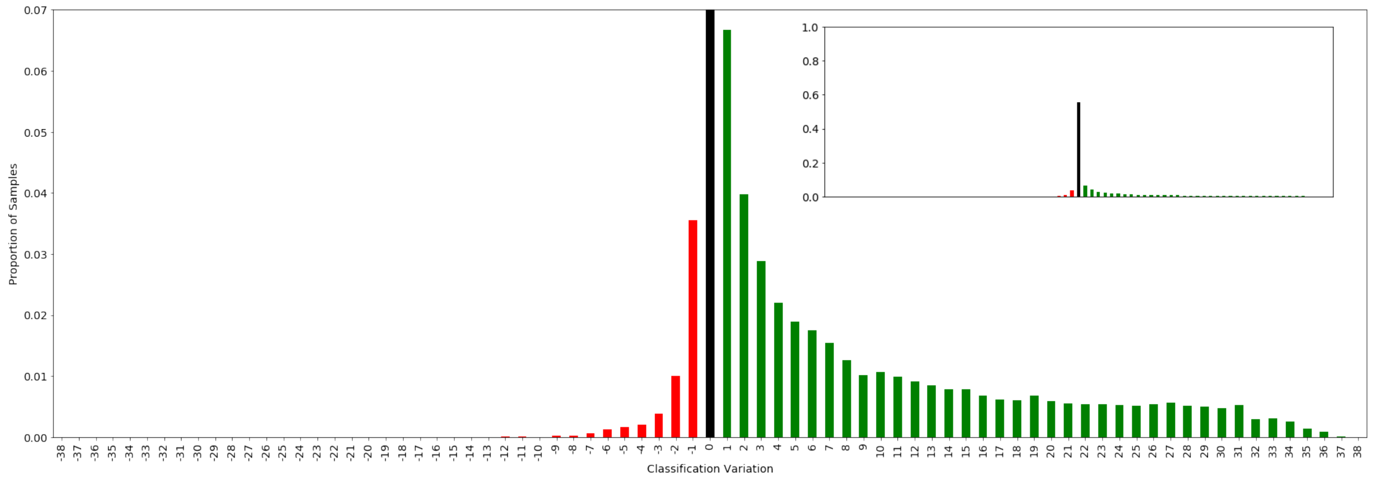

Fig. 2. Distribution of samples in terms of the changes in the number of positive classifications between last and first submissions, $(\mathcal{P}_l - \mathcal{P}_f)$.

a detection rate from 56.82% to 85.29%, with a false positive rate ranging from 0.03% to 6.91%. Had they kept their original classification, one would have that $\mathcal{P}_l - \mathcal{P}_f = 0$ for every duplicate and consequently $\text{DR}_v = 1$ and $\text{FPR}_v = 0$. Notice that by keeping the original classification all clean samples remain clean, hence $\text{FN}_v = 0$, and all malicious samples remain malicious, hence $\text{FP} = 0$.

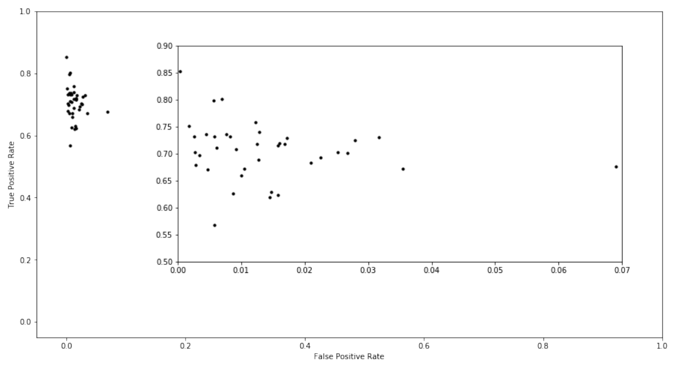

Fig. 3. TPR_v vs. FPR_v according to $\mathcal{M}^{dups}_{acc}$.

For our second approach regarding vendors' accuracy, we take into account our observations from Figure 2 and our dataset $\mathcal{C}$ to define another metric $\mathcal{M}^{\mathcal{C}}_{acc}$.

Intuitively a sample is classified as goodware according to this metric, *i.e.* negative, if every vendor $v \in \mathcal{V}$ classifies it as clean. To understand if our intuition is sound, we plot Figure 4, a subset of Figure 2, showing the frequency $\mathcal{P}_l - \mathcal{P}_f$ for samples that were classified as clean in their first submission, *i.e.*, samples with $\mathcal{P}_f = 0$. These account for 4,902 samples, 3,741 (76.32%) of which do not increase in classification.

To arrive at a positive (*i.e.* malware) sample definition, we relate Figure 4 with Figure 2. Specifically we want to find a minimum threshold of positive classifications to define a sample as malware. We chose five as the threshold, observing that percentage of samples that decrease in 5 or more positive

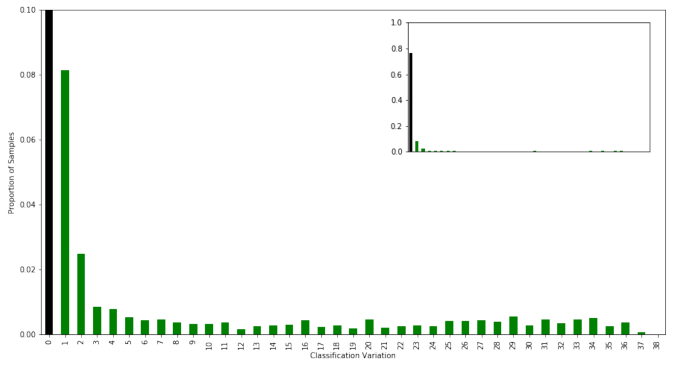

Fig. 4. Distribution of samples that started as goodware and changed in the number of positive classifications between the last and first submission, *i.e.*, $\mathcal{P}_l - \mathcal{P}_f$ for samples with $\mathcal{P}_f = 0$

classifications is 0.46%, meaning it is an upper bound for samples that decrease from 5 or more to zero classifications.

With the previous definitions, we can define a vendors' v classification (according to $\mathcal{M}^{\mathcal{C}}_{acc}$) as:

- TP_v, *true positive for* v, if v and at least 5 other vendors in $\mathcal{V}$ classify it positively;
- TN_v, *true negative for* v, if v and all other vendors in $\mathcal{V}$ classify it negatively;
- FP_v, *false positive for* v, if v is the only vendor in $\mathcal{V}$ classifying it positively;
- FN_v, *false negative for* v, if v classifies it negatively and at least 5 other vendors in $\mathcal{V}$ classify it positively.

Figure 5 plots each vendors' DR_v vs. FPR_v according to $\mathcal{M}^{\mathcal{C}}_{acc}$. Using this metric we note that vendors' detection rate is more scattered than under $\mathcal{M}^{dups}_{acc}$, ranging from 20.17% to 83.50%, whereas false positive rate is similar, ranging from 0.01% to 5.77%.

Given the impossibility of finding a source that is able to unanimously label each sample in our dataset, we want to find the vendors that perform the best under both metrics $\mathcal{M}^{dups}_{acc}$

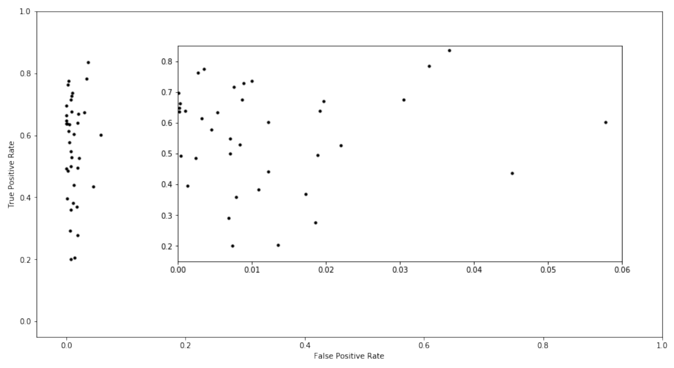

Fig. 5. TPR_v vs. FPR_v according to $\mathcal{M}_{acc}^{\mathcal{C}}$.

and $\mathcal{M}_{acc}^{\mathcal{C}}$.

Our approach to this problem is to filter the top 20 vendors $\mathcal{V}^{dups}$ and $\mathcal{V}^{\mathcal{C}}$, according to each metric and define $\mathcal{V}^* = \mathcal{V}^{dups} \cap \mathcal{V}^{\mathcal{C}}$. Given we have to maximize two variables, TPR and FPR, we decided to take advantage of the linear equation in the form $mx + b = y$ to choose the top vendors. This form allows us to choose an m and b such that there are 20 vendors above the line, the top vendors $\mathcal{V}^*$. By tweaking the variable m one can change the line's steepness, reflecting in a preference between TPR and FPR

- TPR preference: $m < 1$, less steepness therefore higher FPR and TPR values;
- FPR preference: $m > 1$, more steepness, lower FPR and TPR values.

Since we have no *a priori* preference between TPR nor FPR, we search for the maximum b such that there are exactly 20 vendors above $x + b$ in each graphic (Figures 3 and 5).

Figure 6 shows the DR_v vs. FPR_v for the resulting 11 vendors under each metric, $\mathcal{M}_{acc}^{dups}$ in green and $\mathcal{M}_{acc}^{\mathcal{C}}$ in red, with DR_v varying from 63.49% to 83.50% and FPR_v from 0.01% to 3.67%.

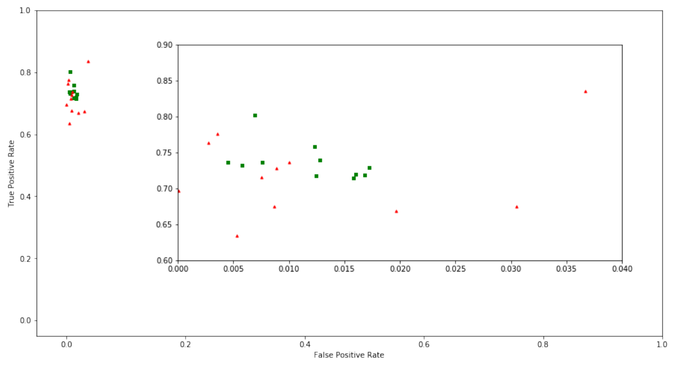

Fig. 6. TPR_v vs. FPR_v according to $\mathcal{M}_{acc}^{dups}$ (green) and $\mathcal{M}_{acc}^{\mathcal{C}}$ (red).

C. Data Labelling

Having defined a collegiate set of vendors $\mathcal{V}^*$ to classify the samples in our dataset, we can now turn our focus into labeling the reports as goodware or malware. To do so, we use $\mathcal{C}$, together with NSRL and VS, to derive three different metrics to label the reports as benign or malicious, over the set of vendors $\mathcal{V}^*$.

The first and most real metric we define is $\mathcal{M}_{\text{real}}^{\mathcal{V}^*}$ that labels a sample $s \in \mathcal{C}$ as:

- $s \in \text{Malware}_{\text{real}}$ if at least 5 vendors in $\mathcal{V}^*$ classify s positively;
- $s \in \text{Goodware}_{\text{real}}$ if all vendors in $\mathcal{V}^*$ classify s negatively.

Since the labeling information is solely provided by $\mathcal{V}^*$'s vendors, this metric's ground truth is highly dependent on their performance, which means labeling errors may be present (as we have discussed in III-B). Due to this, samples that are classified positively by no more than four vendors, are discarded.

Our second metric $\mathcal{M}_{\text{loose}}^{\mathcal{V}^*}$, restricts the previous metric $\mathcal{M}_{\text{real}}^{\mathcal{V}^*}$ to achieve a better ground truth. We do this by including information from NSRL and VirusShare.com. This metric labels a sample $s \in \mathcal{C}$ as:

- $s \in \text{Malware}_{\text{loose}}$ if $s \in \text{Malware}_{\text{real}}$ **and** it belongs to $\mathcal{C}_{\text{VS}}$ and does not belong to $\mathcal{C}_{\text{NSRL}}$;
- $s \in \text{Goodware}_{\text{loose}}$ if $s \in \text{Goodware}_{\text{real}}$ **and** it belongs to $\mathcal{C}_{\text{NSRL}}$ and does not belong to $\mathcal{C}_{\text{VS}}$.

that is

$$\begin{aligned}\text{Malware}_{\text{loose}} &= (\text{Malware}_{\text{real}} \cap \mathcal{C}_{\text{VS}}) \setminus \mathcal{C}_{\text{NSRL}} \\ \text{Goodware}_{\text{loose}} &= (\text{Goodware}_{\text{real}} \cap \mathcal{C}_{\text{NSRL}}) \setminus \mathcal{C}_{\text{VS}}\end{aligned}$$

By taking into account the presence in NSRL, that reinforces cleanliness, and VirusShare.com, that reinforces maliciousness, this metric is more reliable, ground truth wise, at the expense of a smaller number of samples.

Our third and final metric $\mathcal{M}_{\text{strict}}^{\mathcal{V}^*}$, is the strictest one, labeling a sample $s \in \mathcal{C}$ as:

- $s \in \text{Malware}_{\text{strict}}$ if all $v \in \mathcal{V}^*$ classify it positively **and** $s \in \mathcal{C}_{\text{VS}} \setminus \mathcal{C}_{\text{NSRL}}$;
- $s \in \text{Goodware}_{\text{strict}}$ if $s \in \text{Goodware}_{\text{loose}}$.

Obviously this is the most reliable metric, in the sense that it is closely related to the samples' ground truth, leaving little room for disagreement. However, this is achieved again at the cost of a smaller number of samples.

Taking the previously defined metrics, the task of creating labeled datasets based on them is trivial. We apply each of the previously defined metrics, $\mathcal{M}_{\text{strict}}^{\mathcal{V}^*}$, $\mathcal{M}_{\text{loose}}^{\mathcal{V}^*}$ and $\mathcal{M}_{\text{real}}^{\mathcal{V}^*}$ to our classified dataset $\mathcal{C}$ to obtained three new datasets, $\mathcal{C}_{strict} \subseteq \mathcal{C}_{loose} \subseteq \mathcal{C}_{real} \subseteq \mathcal{C}$. Table I provides information regarding the size and number of malware and goodware in each of our datasets.

IV. Feature and Model Selection

In this subsection we describe our approach to feature selection and linear model choice.

Dataset	$\mathcal{C}_{real}$	$\mathcal{C}_{loose}$	$\mathcal{C}_{strict}$
Malware	98,582	45,306	24,658
Goodware	56,475	1,989	1,989
Total	155,057	47,295	26,647

TABLE I
SIZES FOR DATASETS $\mathcal{C}_{real}$, $\mathcal{C}_{loose}$ AND $\mathcal{C}_{strict}$.

A. Feature Selection

One of the most important stages in Machine Learning is the selection of the features to analyze, and features based on static imports have shown promising results in ML applications for malware detection [10], [15]. In this section we describe the adopted static features that were fed into our model.

Although Cuckoo provides enormous amounts of usable information, we chose to start with simple features as to have a basic understanding of how doable our approach is. More so, one of our main concerns is how the same feature gives different results under our different scenarios, hence the performance between scenarios and methodologies is more relevant than absolute performance. With that in mind, we chose to use the static imports as features.

Using Celery [29], a distributed task queue for Python, we optimized the parsing of the available HTML reports, extracting samples that contained information regarding static imports into a new set $\mathcal{F}_{static}$. We then joined the samples with static imports $\mathcal{F}_{static}$ to the labeled samples $\mathcal{C}_{real}$, obtaining a total of 155,057 labeled samples with static imports $\mathcal{C}_{static} = \mathcal{C}_{real} \cap \mathcal{F}_{static}$.

We then vectorized imports by creating a binary vector where each position corresponds to a specific import. If a given import i is present in a sample, its feature vector x will have the value 1 at that position x_i. Likewise, if a given import j is not present in a sample, its feature vector x will have the value 0 at that position x_j.

Due to the amount of samples and variety of imports, each sample got a vector x of 7,280 dimensions (*i.e.* there are 7,280 different imports). To reduce this number, and to remove any noise due to incorrect parsing of static imports by Cuckoo, we applied a variance threshold.

The variance threshold calculates the variance for each import, removing those that are below a given threshold. In our case, since we are working with a binary vector, each import can be represented as Bernoulli random variable, hence their variance is given by $p(1-p)$. With that in mind, we removed any import that did not vary in more than 99% of samples.

The resulting dataset $\mathcal{C}_{static}$ got reduced to 153,374 samples, each with a 64 dimensional binary vector.

B. Model Selection

In this subsection we go over the classifier used to create the model that separates malware from goodware. Our main concerns when choosing a classifier regard the ability to produce a probabilistic output, good scaling for large number of features and samples, and ease of use.

Taking into consideration the guidelines given in [21], [22] and related work in [10], [11], [13], [15], we decide to use *Logistic Regression (LR)* as our model. This model fits our needs as it gives the probability of a random variable X being 0 or 1, given a set of constraints (*i.e.* features), scales well with samples and features and it is readily available from several libraries, facilitating implementation [30].

LR can be defined with the form

$$\rho(x) = \frac{1}{1+e^{-x}}, \quad x = \beta_0 + \beta_1 x_1 + ... + \beta_n x_n$$

where β_n is the learned weight for feature x_n. This weight is learned through iteration in order to minimize the error between the predicted values and the actual values. In other words, given an *n-th* dimensional set of features, LR will try to create an hyperplane that divides samples from two classes.

As LR is based on the logistic function (or sigmoid function), each feature x_n can vary from $-\infty$ to $+\infty$ and still the output is contained between 0 and 1, hence providing probabilistic values.

V. Evaluation and Results

In this section we aim at doing a comparative analysis on three different scenarios $\mathcal{S}_{\text{strict}}$, $\mathcal{S}_{\text{loose}}$ and $\mathcal{S}_{\text{real}}$, built on top of the previously defined metrics. This comparison is done using standard cross-validation methodologies and a proposed temporal-based methodology.

We further provide an analysis on how to reduce the size of the training set, without compromising the final results.

A. Evaluation

With regards to our evaluation methodology, as we have previously mentioned, our purpose is to understand how laboratory conditions compare to real-world conditions. We now detail how we achieve and compare these conditions.

Given the purpose of our work, we choose to measure our results by plotting an AUROC graph, which measures the TPR at different FPR levels, metrics that are commonly used across similar work [10], [11], [15].

The three scenarios that we will focus on will rely on metrics $\mathcal{M}^{\mathcal{V}^*}_{\text{real}}$, $\mathcal{M}^{\mathcal{V}^*}_{\text{loose}}$ and $\mathcal{M}^{\mathcal{V}^*}_{\text{strict}}$, over the dataset $\mathcal{C}$:

- Real Scenario $\mathcal{S}_{\text{real}}$, applies the metric $\mathcal{M}^{\mathcal{V}^*}_{\text{real}}$, containing 98,582 malware samples and 56,475 goodware samples.
- Loose Scenario $\mathcal{S}_{\text{loose}}$, applies the metric $\mathcal{M}^{\mathcal{V}^*}_{\text{loose}}$, containing 45,306 malware samples and 1,989 goodware samples.
- Strict Scenario $\mathcal{S}_{\text{strict}}$, applies the metric $\mathcal{M}^{\mathcal{V}^*}_{\text{strict}}$, containing 24,658 malware samples and 1,989 goodware samples.

Given these three scenarios, we consider the following evaluation metrics:

a) Cross-validation (Figure 7): To gain insight on how each model generalizes our scenarios, we apply a k-fold cross-validation, with $k = 10$. This methodology splits the dataset into k subsets (*i.e.*, folds), selecting a single fold for validation and the remaining $k-1$ folds for training. This process is repeated k times, ensuring every fold is used for validation and training.

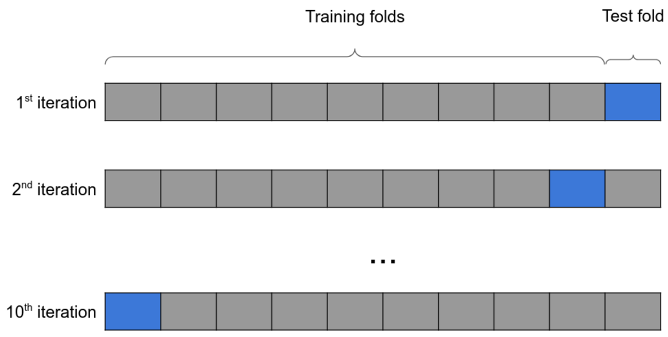

Fig. 7. Cross-Validation evaluation example with 10 folds.

Although the cross-validation methodology enables to measure the generalization capabilities of a model, it does not account for temporal ordering of the samples. Since we want to measure the score when training samples pre-date the validation samples, we now define a couple of temporal based validations. These are validated on the best performing model from cross-validation for all 3 scenarios.

b) Temporal based validation: The first temporal based validation, which we designate as *Past-to-Present validation*, Figure 8, can be resumed as an iterative methodology where the validation set is fixed with the most recent samples, and the training set with the oldest. At each iteration the training set is extended with more recent samples and scored against the validation, until all samples are used.

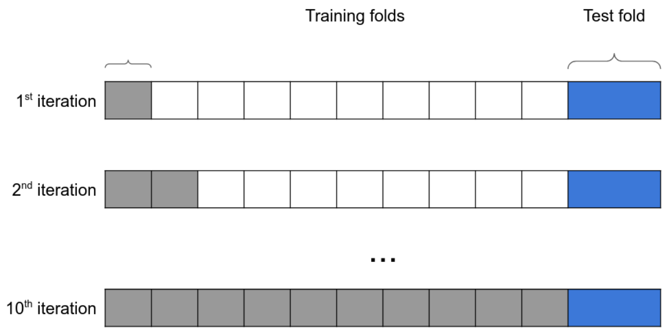

Fig. 8. Past-to-Present evaluation example with 20/80 test/training, 10 folds in training.

The second temporal based validation, which we designate as *Present-to-Past validation*, Figure 9, is the opposite of *Past-to-Present* with regards to the starting position of the training set. Again the validation is fixed the most recent samples, but now the training set starts with the temporally closest samples to the validation set. At each iteration the training set is extended, this time with older samples and scored against the validation, until all samples are used.

Past-to-Present and *Present-to-Past* validations both require two parameters, specifically the size of the validation set, and how the increments to the training set are made. For our evaluation, we use the 20% most recent samples as validation, and split the remaining 80% into 10 folds, hence the validation is done 10 times, with each iteration increasing the training size by one fold.

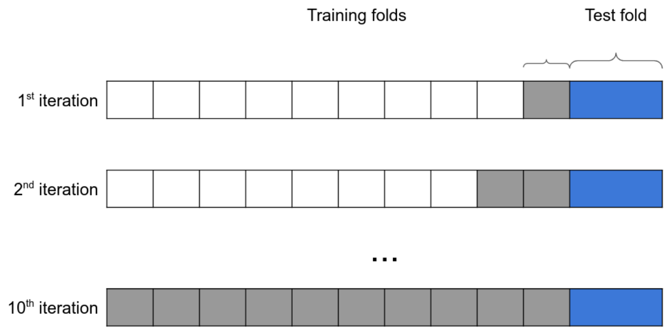

Fig. 9. Present-to-Past evaluation example with 20/80 test/training, 10 folds in training.

These two validation methodologies give us the ability to account for temporal consistency. Moreover, they enable us to compare the importance of older *vs* newer samples to classify recent samples.

We designate the third and last temporal based validation as *Temporal Window validation*, Figure 10. This validation methodology is inspired on regular cross-validation, in the sense that it splits the dataset into folds, but changes how the folds are used. Specifically it takes n temporal consistent and contiguous folds, *i.e.*, each fold immediately precedes the next one, and uses the last fold (more recent samples) for validation, and the previous folds for training (older samples). By starting with the n first folds and sliding one fold on each iteration, we apply a sliding window of size n over the dataset.

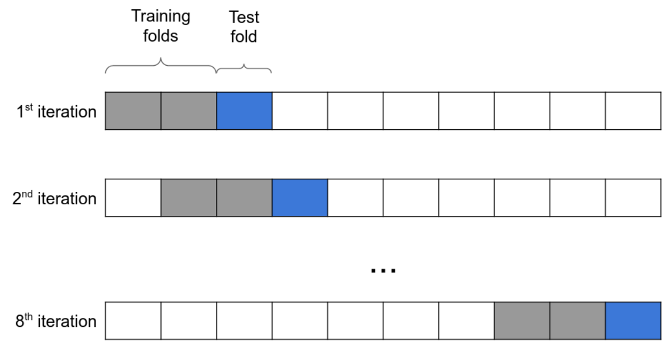

Fig. 10. Temporal window evaluation example with window of size 3 over a 10 fold dataset.

For this last validation methodology, we again split the dataset into 10 folds. The sliding window size, n, is chosen during the results phase, as its choice depends on previous results.

We measure the AUROC during each validation's iteration and use the average measurement to discuss the results.

B. Results

We implement our experiments in Python, by using Jupyter Interactive Notebooks [31] to facilitate data visualization. We use scikit-learn [32] for ML, and Pandas [33] for data analysis. Our experiments were conducted on an Ubuntu Virtual Machine with 16 cores and 16GB of RAM, in order to minimize training and validation times.

We now focus on applying the evaluation methodologies to our scenarios. This enables us to compare the different conditions, and consequently results, that affect malware detection.

We start with what we determine as *laboratory conditions*, ideal conditions for the problem of malware detection. These are met when we apply the strict metrics $\mathcal{M}^{\mathcal{V}^*}_{\text{strict}}$, to the dataset $\mathcal{C}$, obtaining scenario $\mathcal{S}_{\text{strict}}$.

Under these conditions, our model provides the best results, with an AUROC of 0.91, as shown by the red curve in Figure 11. We argue that such high values are easily attained from factors like a small and reliable dataset, and the use of cross-validation, which mixes samples and ignores possible dependencies on malware samples.

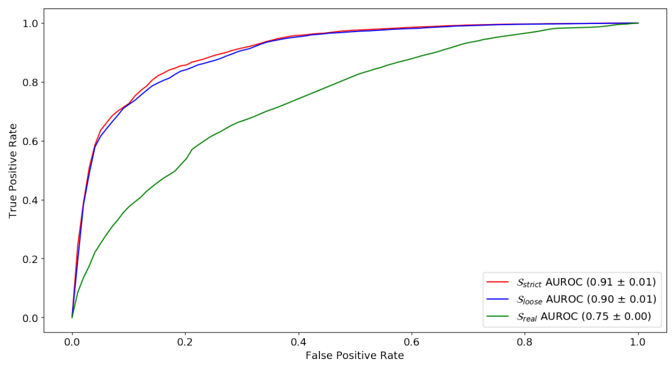

Fig. 11. Cross-Validation ROC and AUROC for our model under $\mathcal{S}_{\text{strict}}$, $\mathcal{S}_{\text{loose}}$ and $\mathcal{S}_{\text{real}}$.

To understand how reliability influences the models' result, we use scenarios $\mathcal{S}_{\text{loose}}$ and $\mathcal{S}_{\text{real}}$, which are less reliable and include more samples.

Under these more relaxed, *real-world* conditions, the model's results hold an AUROC of 0.90 under $\mathcal{S}_{\text{loose}}$, as shown by the blue curve in Figure 11, and an AUROC of 0.75 under $\mathcal{S}_{\text{real}}$, as seen by the green curve in Figure 11.

From $\mathcal{S}_{\text{strict}}$ to $\mathcal{S}_{\text{loose}}$, the only change is the amount of malware labeled samples, which significantly increase. The difference is interesting, as although the number of malware labeled samples increase significantly, the results are not that affected. This suggests that although the reliability for malware decreases, its impact is not as noticeable as expected. This might also suggest that vendors do converge on their definition of malware, under our $\mathcal{M}^{\mathcal{V}^*}_{\text{loose}}$ metric. If vendors did not converge on what is malware, adding more samples would culminate in worse results, as separation between malware and goodware would become harder.

When looking at the changes from $\mathcal{S}_{\text{loose}}$ to $\mathcal{S}_{\text{real}}$, not only the amount of malware labeled samples increase, but also the number of goodware labeled samples, both by a significant amount. The way this impacts the results is pretty significant, as we observe a high decrease in the AUROC. The metric $\mathcal{M}^{\mathcal{V}^*}_{\text{real}}$ that labels malware and goodware for this scenario $\mathcal{S}_{\text{real}}$ disregards the cross-check from outside repositories, which in turn degrade the reliability significantly, as well as increase the dataset size notably. We attribute the results' degradation mainly to the unreliability of goodware labeling, not only because we have previously seen that increase in malware does not significantly impact results (from $\mathcal{S}_{\text{strict}}$ to $\mathcal{S}_{\text{loose}}$), but also due to the tendency for false negatives in vendors (Figure 2), which in turn lead us to incorrectly label goodware for the samples in $\mathcal{C}$.

The results we described show how moving from *laboratory conditions* to more *real-world conditions* degrade the model's performance. We now focus on using our previously defined temporal based methodologies to further converge into a real-world scenario.

We start by applying our *Past-to-Present* validation to the three scenarios, $\mathcal{S}_{\text{strict}}$, $\mathcal{S}_{\text{loose}}$ and $\mathcal{S}_{\text{real}}$. As previously defined, this validation starts with an older set of training samples and iteratively adds newer samples, validating each iteration on a fixed set of the most recent samples. Since our interest is to measure performance variation over time, we plot in Figure 12 the AUROC at every iteration (*i.e.*, fold), for each of our three scenarios.

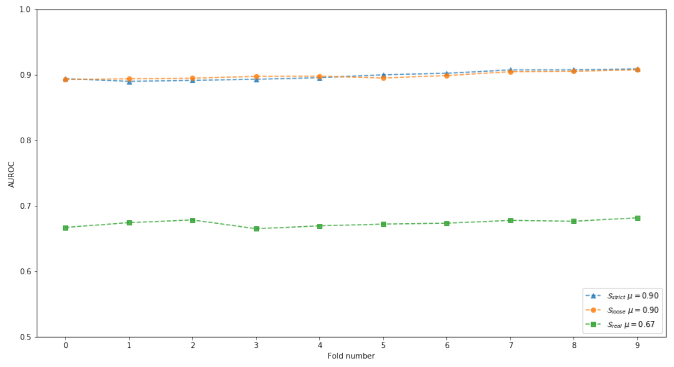

Fig. 12. AUROC for each iteration of the *Past-to-Present* evaluation. Folds order consistent with temporal order (*i.e.*, fold 0 contains older samples than fold 1)

When directly comparing the average AUROC for cross-validation and our *Past-to-Present* validation, we note that for both $\mathcal{S}_{\text{strict}}$ and $\mathcal{S}_{\text{loose}}$ the AUROC remains identical, while for $\mathcal{S}_{\text{real}}$ the score decreases from 0.75 to 0.67. This decrease is intuitive to the methodology, as we are forcing temporal consistency between samples.

For both $\mathcal{S}_{\text{strict}}$ and $\mathcal{S}_{\text{loose}}$ we note only a slight increase as new folds are added. They still relate, as we have previously noted for cross-validation, arguably given their metrics $\mathcal{M}^{\mathcal{V}^*}_{\text{strict}}$ and $\mathcal{M}^{\mathcal{V}^*}_{\text{loose}}$ are not very different. The small variation to the cross-validation methodology can be justified by using small dataset size for both cases.

As for $\mathcal{S}_{\text{real}}$ we note higher variation and lower overall score, as the reliability of the metric $\mathcal{M}_{\text{real}}^{\mathcal{V}^*}$ goes down. This is expected, not only because we are enforcing temporal consistency between samples, but as new folds are added, the training gets bigger, while the test remains the same.

Our main observation for this validation methodology is that there is a slight tendency for AUROC to increase, as we move forward in time, close to the validation set.

From these observations, we argue about the possibility that with fixed validation set of the most recent samples, a model benefits by using samples temporally closer to validation.

Our next result, which uses our *Present-to-Past* validation methodology will further help analyze the aforementioned detail. The *Present-to-Past* validation enhances the previous results under real-world conditions. This methodology starts by fixing the validation set to the most recent samples, but with the training set starting at the temporally closest samples to validation. At each iteration, older samples are added to the training set and validated on the fixed, most recent, samples.

By applying this methodology to the three scenarios, $\mathcal{S}_{\text{strict}}$, $\mathcal{S}_{\text{loose}}$ and $\mathcal{S}_{\text{real}}$, we plot Figure 13, where the X axis increases as older samples are added to the training set (*e.g.* fold 0 contains newer samples than fold 1), hence measuring the performance variance over time. Similarly to the previous observation, the average AUROC suffers a decrease when compared to cross-validation. For $\mathcal{S}_{\text{strict}}$ we note a change from 0.91 to 0.90, for $\mathcal{S}_{\text{loose}}$ the score is the same, and for $\mathcal{S}_{\text{real}}$ 0.75 to 0.69.

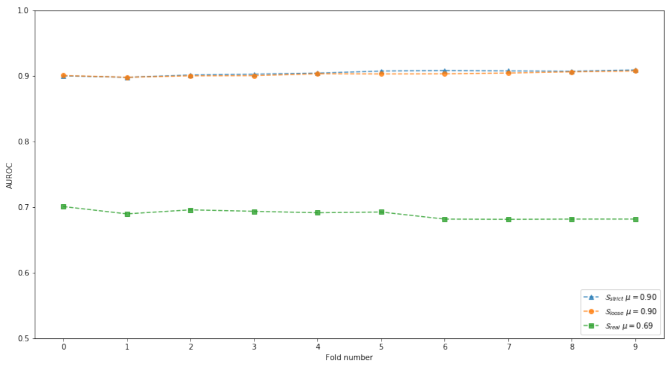

Fig. 13. AUROC for each iteration of the *Present-to-Past* evaluation. Folds order is the inverse of temporal order (*i.e.*, fold 0 contains newer samples than fold 1)

The comparison between scenarios is identical to what was observed in cross-validation and *Past-to-Present*: scenarios $\mathcal{S}_{\text{strict}}$ and $\mathcal{S}_{\text{loose}}$ display very similar results, with $\mathcal{S}_{\text{real}}$ dropping behind due to its less reliable labeling metric. It is noticeable that using the entire dataset does not bring much improvement to the final results. In fact, for $\mathcal{S}_{\text{real}}$ the score even drops after fold #2.

With these results, our original observation that samples closer to the validation set benefit the model becomes more convincing. In fact, we argue that there should be an ideal number of necessary training folds, temporally consistent with the validation fold (*i.e.* any fold from training predates validation), needed to maximize the overall score.

Finally, we analyze how does such reduced training set behaves in our scenarios; for this purpose, we define a sliding window that moves forward in time through each scenario for training and validation. We propose a reduction on the training size to $n = 3$ folds predating the validation fold. We choose $n = 3$, since we have seen that the scores either do not improve (for $\mathcal{S}_{\text{strict}}$ and $\mathcal{S}_{\text{loose}}$) or actually go down (for $\mathcal{S}_{\text{real}}$) with higher folds. In summary, we have selected 30% of each dataset for training purposes and the next 10% for validation (3 training folds, 1 validation fold), and then started moving the window forward in time (1 fold at a time) to obtain the following results (Figure 14): for $\mathcal{S}_{\text{strict}}$, $\mathcal{S}_{\text{loose}}$ and $\mathcal{S}_{\text{real}}$, we obtain AUROC values of 0.89, 0.88 and 0.76, respectively. These results come to reaffirm our argument that we can reduce the size of the training set, without losing any significant score.

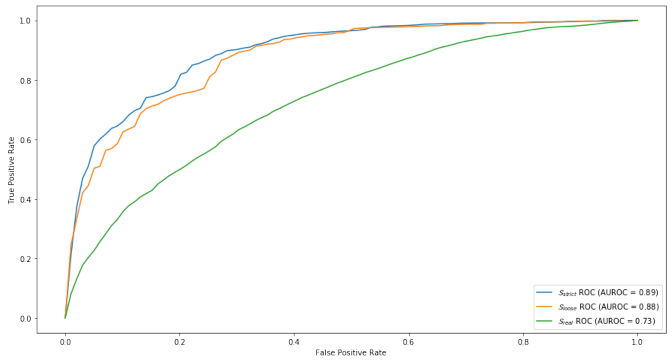

Fig. 14. ROC and AUROC for our three scenarios, under the *Temporal Window* methodology.

Comparing these results with the baseline cross-validation, we note a decrease for each scenario, specifically a decrease from 0.91 to 0.89 for $\mathcal{S}_{\text{strict}}$, from 0.90 to 0.88 for $\mathcal{S}_{\text{loose}}$ and from 0.75 to 0.73 for $\mathcal{S}_{\text{real}}$. We should highlight that the results that use temporal consistency should better reflect the reality than standard cross-validation, since we are requiring temporally ordered samples. Another important idea that should be stressed is that for cross validation we used a fairly reasonable amount of data for training purposes, whereas in this last case we used a restricted amount of data. This might be a relevant issue in a few year's time. The results obtained are summarized in Table II.

AUROC	$\mathcal{S}_{\text{strict}}$	$\mathcal{S}_{\text{loose}}$	$\mathcal{S}_{\text{real}}$	Train/Test %
Cross-Validation	0.91	0.90	0.75	90 / 10
Past-to-Present	0.90	0.90	0.67	10 to 90 / 10
Present-to-Past	0.90	0.90	0.69	10 to 90 / 10
Sliding-Window	0.89	0.88	0.73	30 / 10

TABLE II
SINGLE LAYER RESULTS SUMMARY.

With a better understanding of how the model behaves under different methodologies, we now diverge to how we improved not only the overall results, but also the information provided by the model.

VI. Model Improvements

Having a solid baseline model for our malware detection task together with how laboratory *vs.* real-world scenarios change the model outcome, we now take this section to present the improvements made in order to obtain a more robust model to detect malware. We start by describing our first improvement, applying a multi layer model to extract more information regarding a sample. We then take this enhanced model and increase the number of features to include dynamic content and how it impacted the model's results.

A. Multi Layer Model

On the previous chapter we ended up with a simple LR model $\mathcal{LR}$ that given a set of static imports from a sample, would give the probability of it being malware.

In this section we provide a new model $\mathcal{E}$ comprises a simple ensemble stacking approach, which instead of using a single LR classifier, multiple ones are used, layered into two steps.

The first step (layer $\mathcal{E}_{\mathcal{L}_0}$) is composed of n LR models, where n is the number of possible classes. Each model is trained to output the likelihood of sample belonging to one of the n classes, in a *one-vs-all* methodology (*i.e.* a sample either belongs to $\mathcal{C}_n$ or not), having as input the raw features (*e.g.* static imports).

The second step (layer $\mathcal{E}_{\mathcal{L}_1}$) is identical to $\mathcal{LR}$, but now takes as features the output of each classifier from the previous layer, outputting the likelihood of a sample being malware.

In summary, as depicted in Figure 15, we define a 2 layer ensemble stacking with n classifiers on the first layer to a single classifier in the second layer.

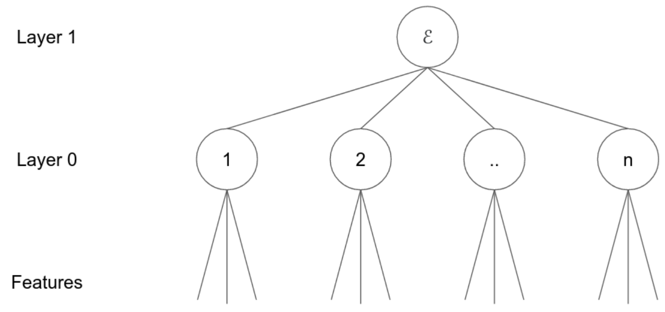

Fig. 15. Multi layer model representation.

B. Malware Classes

We now present our approach on selecting the n classes of interest. This represents another labeling problem, but now instead of having to label between goodware and malware, we have to label the malware as belonging to some subclass.

With this in mind, we chose 6 malware classes: *virus*, *trojan*, *worm*, *ransom*, *spyware* and *other*.

To help label our malware samples into the aforementioned classes, we take a tool by Sebastin, M. et al. [23], AVClass, which was built to normalize a malware sample name into the most likely family, and modify it such that instead of providing a family name, it would provide one (or more) of the 6 previously defined classes. Specifically, we changed it in a way that given a set of malware names, the output would be a distribution over the 6 malware classes.

To calculate each class weight we apply the following formula

$$\mathcal{W}_c = \frac{f_c}{\sum\limits_c f_c}$$

where f_c is the frequency for the class c and $\sum_c f_c$ is the number of times all classes appear. For example, if a given set of names contain the name *trojan* 3 times and the name *virus* one time, then the weights would be

$$\mathcal{W}_{trojan} = \frac{3}{4} = 0.75,\ \mathcal{W}_{virus} = \frac{1}{4} = 0.25,$$
$$\mathcal{W}_c = 0, c \in \{worm, spyware, other, ransom\}$$

Having these malware classes defined for our multi layer model, we also added the *goodware* class for samples that are not malware. Doing so gives us 7 possible classes, 6 of which are malware only. It is worth mentioning that if a sample belongs to the *goodware* class, it cannot belong to any other, likewise, if it belongs to any malware class, it cannot belong to the *goodware* class. Table III discriminates the amount of each malware class.

Dataset	$\mathcal{C}_{real}$	$\mathcal{C}_{loose}$	$\mathcal{C}_{strict}$
Trojan	97,054	44,329	24,176
Other	49,443	24,126	14,750
Worm	24,554	14,837	9,381
Virus	21,055	12,531	6,899
Spyware	20,724	10,172	5,955
Ransom	7,761	1,924	1,160
Malware Total	98,582	45,306	24,658

TABLE III
Samples belonging to each of the 6 malware classes for $\mathcal{C}_{real}$, $\mathcal{C}_{loose}$ and $\mathcal{C}_{strict}$.

In the base model, we used static imports as features for our malware detection model. Although the results are reasonable, the information which can be retrieved from static imports alone is limited. As an example, if a sample is compressed, encrypted or packed, its behavior cannot be inferred from static imports only. To overcome these limitations, we resort to more dynamic information provided by Cuckoo.

C. Category Calls

The first type of dynamic information we extracted were the number of dynamic category calls. When Cuckoo runs and monitors a sample, it registers some low level library calls, which it then assigns to a fixed number of categories.

There are a total of 14 different categories defined by Cuckoo: *anomaly*, *device*, *filesystem*, *hooking*, *misc*, *network*, *process*, *registry*, *services*, *socket*, *synchronization*, *system*, *threading* and *windows*. After using Celery [29] to extract the number of each category calls for the samples, we obtained a total of 148,036 samples with information regarding category calls.

To normalize the category calls value, we decide to transform the values to follow a normal distribution. We do this by using scikit-learn's [32] *QuantileTransformer* with a normal distribution, which splits the possible values into bins such that the resulting distribution is of type *Gaussian* with a mean of 0. This way we have a greater number of bins around the mean, allowing for better discrimination, whereas very large values fall into the same bin.

D. Library Calls

Our second type of dynamic information are the number of library calls. While *category calls* provide the number of calls for a given category, *library calls* provide the count for each library call, hence being a subset of the previous.

Cuckoo [25] registers the number of calls for 163 different functions, ranging from opening and closing files, to opening and closing sockets. Again we used Celery [29] to extract these numbers, obtaining information from 148,036 samples.

Given we are dealing with a high number of features (163 different library calls), we decided to apply the same variance threshold as in Section IV-A, to remove features that do not vary in most samples. By choosing a threshold of 80%, we remove library calls that do not vary in more than 80% of the samples, effectively reducing the number of library calls to 144. With regards to how these features can vary from 0 to $+\infty$, as before, we again apply a quantile transformer with a normal distribution.

E. Cuckoo Signatures

For our third and last type of dynamic information, we resort to Cuckoo's [25] custom signatures. These signatures are built from certain activities that Cuckoo deems malicious or suspicious. For example, if a sample allocates memory and then makes it executable, it might suggest some sort of packing or obfuscation.

To extract these signatures, we use Celery [29] and obtain a total of 124,821 samples and 61 different signatures. As with our static import features, we use a binary vector for each sample, where each position corresponds to a specific signature. Re-iterating on how we represent this, if a given signature i is present in a sample, its feature vector x will have the value 1 at that position x_i. Likewise, if a given signature j is not present in a sample, its feature vector x will have the value 0 at that position x_j.

We joined these features $\mathcal{F}_{dynamic}$ to the labeled samples $\mathcal{C}_{real}$, obtaining a total of 122,633 labeled samples with the new features $\mathcal{C}_{dynamic} = \mathcal{C}_{real} \cap \mathcal{F}_{dynamic}$.

Given there is a lower amount of available samples, we provide in Table IV the new sizes for $\mathcal{C}_{real}$, $\mathcal{C}_{loose}$ and $\mathcal{C}_{strict}$, which take into account the new features $\mathcal{F}_{dynamic}$.

Dataset	$\mathcal{C}_{real}$	$\mathcal{C}_{loose}$	$\mathcal{C}_{strict}$
Malware	94,248	42,911	23,437
Goodware	28,385	1,741	1,741
Total	122,633	44,652	25,178

TABLE IV
NEW SIZES FOR DATASETS $\mathcal{C}_{real}$, $\mathcal{C}_{loose}$ AND $\mathcal{C}_{strict}$.

VII. IMPROVED MODEL RESULTS

We now present the results of our new model $\mathcal{E}$, validated using the same methodology as described in V.

Specifically we test the model using the baseline cross-validation methodology, followed by our three temporally consistent scenarios: *Past-to-Present*, *Present-to-Past* and *Temporal Window*. We test each methodology using the three different scenarios: $\mathcal{S}_{\text{strict}}$, $\mathcal{S}_{\text{loose}}$ and $\mathcal{S}_{\text{real}}$.

Starting with *laboratory conditions*, we apply the cross-validation evaluation to model $\mathcal{E}$ with the labeled dataset $\mathcal{C}_{strict}$ and features $\mathcal{C}_{dynamic}$, providing scenario $\mathcal{S}_{\text{strict}}$. For this scenario, we obtain an AUROC of 98%, as presented by the red curve in Figure 16.

Relaxing to more *real-world conditions*, under the form of a less reliable ground truth, we test the datasets $\mathcal{C}_{loose}$ and $\mathcal{C}_{real}$ on features $\mathcal{C}_{dynamic}$. As shown in Figure 16, the score under AUROC is 98% for $\mathcal{S}_{\text{loose}}$ (blue curve) and 95% for $\mathcal{S}_{\text{real}}$ (green curve).

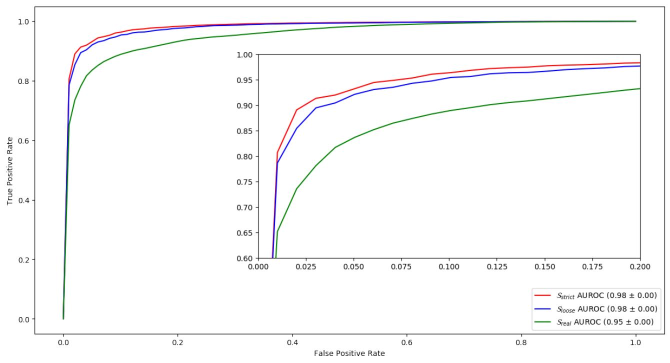

Fig. 16. Multi layer results for dynamic features in laboratory conditions.

As with previous results, one notes that from $\mathcal{S}_{\text{strict}}$ to $\mathcal{S}_{\text{loose}}$ the results are not affected at all, when the change between the scenarios is merely in the number of malware samples. Between $\mathcal{S}_{\text{loose}}$ to $\mathcal{S}_{\text{real}}$ we again note the already seen pattern, the score is lowest when using the most realistic dataset.

The comparison between scenarios does not yield any new information from what was seen in Section V-B. What is more interesting is that the absolute values are boosted in all scenarios, which show indeed that using the multi-layer approach with dynamic features improve the model's results.

Having applied the same cross-validation to our modified model, obtaining interesting results, we now go over to test how our temporal based methodologies are affected.

We start again with *Past-to-Present* validation to each scenario $\mathcal{S}_{\text{strict}}$, $\mathcal{S}_{\text{loose}}$ and $\mathcal{S}_{\text{real}}$. Figure 17 shows the AUROC at every iteration (*i.e.* fold) for each scenario: 96% for $\mathcal{S}_{\text{strict}}$ and $\mathcal{S}_{\text{loose}}$, and 92% $\mathcal{S}_{\text{real}}$.

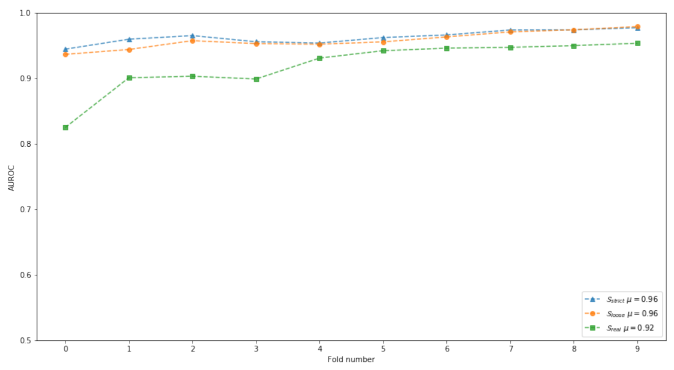

Fig. 17. AUROC for each iteration of the *Past-to-Present* evaluation. Folds order consistent with temporal order (*i.e.* fold 0 contains older samples than fold 1)

When comparing the average AUROC between cross-validation and *Past-to-Present* validation, we note that both $\mathcal{S}_{\text{strict}}$ and $\mathcal{S}_{\text{loose}}$ decrease 2%, while $\mathcal{S}_{\text{real}}$ decreases 3%. This decrease is not a surprise, given the temporal consistency enforcement between samples.

The results are consistent with was previously seen in Section V-B, with the added factor that the absolute values are higher, and the tendency to increase is more present as we move forward in time, close to the validation set.

Following the previous evaluation order, we now present the results using our *Present-to-Past* validation methodology. In Figure 18 we present the AUROC for each iteration, where higher folds represent older samples. Here we see values of 97% for $\mathcal{S}_{\text{strict}}$, 98% for $\mathcal{S}_{\text{loose}}$ and 96% $\mathcal{S}_{\text{real}}$.

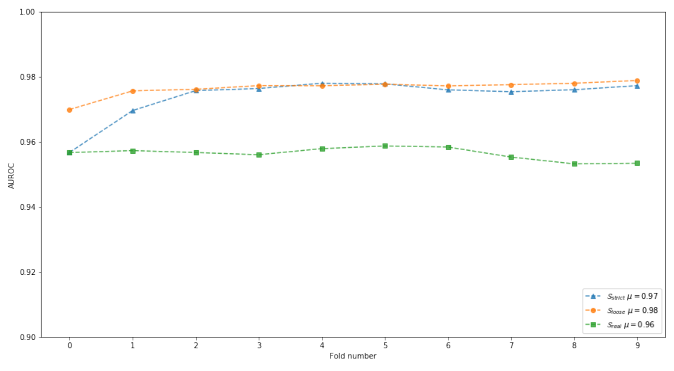

Fig. 18. AUROC for each iteration of the *Present-to-Past* evaluation. Folds order is the inverse of temporal order (*i.e.* fold 0 contains newer samples than fold 1)

When comparing to cross-validation, we note $\mathcal{S}_{\text{strict}}$ is affected as expected, whereas $\mathcal{S}_{\text{loose}}$ is not. The fact that the datasets vary in size and so the amount of goodware and malware used for testing in each may vary, can justify how the $\mathcal{S}_{\text{loose}}$ is not affected, whereas $\mathcal{S}_{\text{strict}}$ is.

$\mathcal{S}_{\text{real}}$ seems to have a higher value than in cross-validation, but it is rounded, which slightly inflates the value. In practice both cross-validation and *Present-to-Past* are identical for $\mathcal{S}_{\text{real}}$.

In these results, the noticeable increase in the first 3 folds (0, 1 and 2) for $\mathcal{S}_{\text{strict}}$ and $\mathcal{S}_{\text{loose}}$ goes even more in favor with our argument that samples closer to the validation set benefit the model. More so as the AUROC stabilizes from those folds on. This effect is not as accentuated for $\mathcal{S}_{\text{real}}$, although using more and older folds do not provide significantly better results.

Finally, we retest how a reduced training set behaves in our scenarios by using our *Temporal Window* methodology. As previously mentioned, the first 3 folds seem to provide enough information to obtain good results, hence we apply the same sliding window size of $n = 3$ as in Section V-B. Starting at the oldest fold, we apply this window and slide it by one fold at each iteration. Figure 19 shows how all our scenarios $\mathcal{S}_{\text{strict}}$, $\mathcal{S}_{\text{loose}}$ and $\mathcal{S}_{\text{real}}$ score the same AUROC of 94%, although with different curves.

For $\mathcal{S}_{\text{strict}}$, $\mathcal{S}_{\text{loose}}$ the score is equally and negatively affected by 4%. The jagged curve on both scenarios indicate that slight changes on the FPR threshold have significant impact on the True Positive Rate (TPR), this may be caused by the smaller dataset size, which in turn creates uneven folds for malware and goodware.

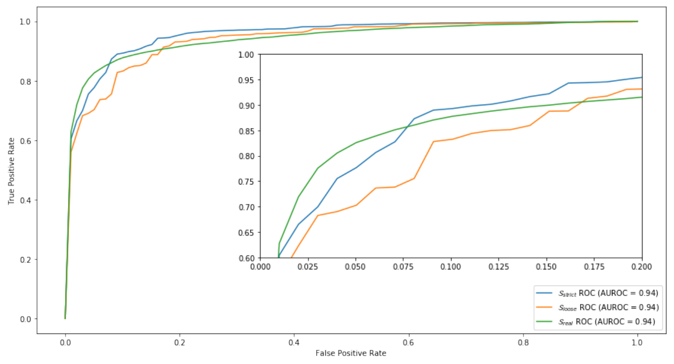

Fig. 19. AUROC for our three scenarios, under the *Temporal Window* methodology.

For $\mathcal{S}_{\text{real}}$, the fact that it only loses 1% when compared to cross-validation shows that indeed our argument about reducing the training set size is sustained. Its curve is much smoother when compared to the previous scenarios, as the dataset is bigger and more even.

The improved layer results are summarized in Table V.

VIII. Discussion

We have proposed different scenarios, based on different labeling metrics, to study laboratory *vs* real-world conditions. Our scenarios, $\mathcal{S}_{\text{strict}}$, $\mathcal{S}_{\text{loose}}$ and $\mathcal{S}_{\text{real}}$, vary both in reliability

AUROC	$\mathcal{S}_{\text{strict}}$	$\mathcal{S}_{\text{loose}}$	$\mathcal{S}_{\text{real}}$	Train/Test %
Cross-Validation	0.98	0.98	0.95	90 / 10
Past-to-Present	0.96	0.96	0.92	10 to 90 / 10
Present-to-Past	0.97	0.98	0.96	10 to 90 / 10
Sliding-Window	0.94	0.94	0.94	30 / 10

TABLE V
MULTI LAYER RESULTS SUMMARY.

and size, going from a more reliable and small dataset to a larger and less reliable one. This (un)reliability is due to the fact that for real world samples there is usually no agreement among vendors on how to classify a given sample, and for that we had to assign a labeling to such samples according to our proposed metrics. We have developed several comparative analysis between these three scenarios, to evaluate how much the nature of the dataset can influence the results. We have split the analysis into two major validation conditions: the cross validation methodology, where the time consistency is discarded; and temporal-based methodologies. We tested our methodologies by using a simple LR model, which was then improved to transmit better information and to use more features.

Following a cross-validation methodology, we have confirmed our intuitions: $\mathcal{S}_{\text{strict}}$ showed up an AUROC of 0.91, $\mathcal{S}_{\text{loose}}$ have presented 0.9, whereas $\mathcal{S}_{\text{real}}$ decreased to 0.75. As we have argued, the results on $\mathcal{S}_{\text{strict}}$ are justified by factors like a small and reliable dataset, and the use of cross-validation, which mixes samples and ignores possible dependencies between them. This scenario is composed by very well-known and analyzed samples. Although $\mathcal{S}_{\text{loose}}$ slightly relaxes these requirements, it is still composed by very well-known samples, which partially justifies the comparable AUROC (0.90). But this difference is interesting, as although the number of malware labeled samples increased significantly, the results are not that affected. As we have noticed, this might also suggest that vendors do converge on their definition of malware, under our $\mathcal{M}_{\text{loose}}$ metric. The changes observed from $\mathcal{S}_{\text{loose}}$ to $\mathcal{S}_{\text{real}}$ are more remarkable, but somehow expected. The metric that labels malware and goodware for the scenario $\mathcal{S}_{\text{real}}$ disregards the cross-check from outside repositories, which in turn degrades the reliability significantly, while increasing the dataset size notably. As we have already noticed, we attribute the result's degradation mainly to the unreliability of goodware labeling, not only because we have previously seen that the increase in malware does not significantly impact the results (from $\mathcal{S}_{\text{strict}}$ to $\mathcal{S}_{\text{loose}}$), but also due to the tendency for false negatives in vendors (Figure 2), which in turn lead us to incorrectly label as goodware some of the malicious samples in $\mathcal{C}$.

When temporal consistency comes into play, the results on different scenarios do not differ much, nevertheless we can observe more pronounce trends. The great conclusion that we can take stands on the relative position of the training set with respect to the validation set and its size. Indeed, samples closer to the validation set seem to benefit the model. We argue that there should be an ideal number of necessary training folds (30% of the dataset), temporally consistent with the validation fold (10% of the dataset), needed to maximize the overall score. This supports our argument that we can reduce the size of the training set, without losing any significant score.

We finished our analysis by validating this temporal-based results. For this purpose, we have defined a sliding window for each scenario, with the above parameters, that moved forward in time (1 fold at a time) and obtained the AUROC values of 0.89 for $\mathcal{S}_{\text{strict}}$, 0.88 for $\mathcal{S}_{\text{loose}}$ and 0.73 for $\mathcal{S}_{\text{real}}$. Comparing these results with the baseline cross-validation, we note a very slight decrease for each scenario. This decrease, although not significant, was more than expected due to the enforcement of temporal consistency as well as the significant reduction of the size of the training set. We should highlight that these results should be much closer to reality than the ones provided by cross validation techniques, since we are requiring temporal consistency and also a reasonable amount of data for training purposes, which might be a relevant issue in a few year's time. Indeed, aiming at complementing antivirus' vendors techniques with machine learning, we should not expect to gather and use all the samples ever seen for training purposes, and these results may be very useful on the choice of the right training set.

Finally, we describe multiple improvements to our base model $\mathcal{LR}$ in order to improve the overall results. We started by using a multi layer approach to build a new model $\mathcal{E}$, which enables the extraction of more detailed information regarding a malicious sample, specifically the malware class it belongs. We also introduced three new dynamic features, to improve the amount of information obtained from the samples. After applying the same evaluation methodologies to our new model $\mathcal{E}$, we observed an increase in all cases. We note that the bigger the dataset, the higher the improvement, as $\mathcal{S}_{\text{strict}}$ increased by 0.07 (cross-validation), $\mathcal{S}_{\text{loose}}$ by 0.08 and $\mathcal{S}_{\text{real}}$ (cross-validation) by 0.27 (*present-to-past*). This comes to show how the model was better able to learn from the new features.

IX. CONCLUSION

In this paper we analyzed how ML techniques fit into the scope of malware detection and how could the chosen dataset influence the results of the classifier.

Given the non-existence of a common agreement on how to label samples in a real world dataset, we have proposed three different metrics for labeling these samples, and presented three different scenarios, ranging from a more simulated scenario, where better results are achieved, to more realistic ones, where the AUROC results can go down by 23%. We have analyzed the different scenarios mainly on two kind of conditions: the laboratory conditions where the standard cross-validation methodology was applied discarding the importance of *time* in malware detection, and temporal-consistent techniques where we have trained and validated the model in a temporal-consistent manner. We have shown that for a modest compromise in accuracy temporal-consistent methodologies are adequate to classify malware samples.

We have also concluded that we can reduce the size of the training dataset to avoid the need of training with all ever seen samples, and argue on how much it can be reduced without compromising optimal results.

Having a sound understanding of the effects of different methodologies, we improved our model to yield higher results.

We believe that the pertinent question of how much should we seek for great results on ML techniques applied to malware detection is worth to be further discussed, bearing in mind that it leads to classifiers that would not perform better over realistic conditions. As future work we aim at optimizing our logistic regression model, at increasing and optimizing the features, and finally, at developing a supervised learning methodology to classify malware samples according to the main malware families.

References

[1] "AV-TEST Security Report 2016-2017," https://www.av-test.org/fileadmin/pdf/security_report/AV-TEST_Security_Report_2016-2017.pdf, accessed: 2018-05-08.

[2] M. Christodorescu, S. Jha, S. A. Seshia, D. Song, and R. E. Bryant, "Semantics-aware malware detection," in *Security and Privacy, 2005 IEEE Symposium on*. IEEE, 2005, pp. 32–46.

[3] W. S. Lee and B. Liu, "Learning with positive and unlabeled examples using weighted logistic regression," in *ICML*, vol. 3, 2003, pp. 448–455.

[4] T. Joachims, *Learning to classify text using support vector machines: Methods, theory and algorithms*. Kluwer Academic Publishers, 2002.

[5] L.-J. Li, H. Su, L. Fei-Fei, and E. P. Xing, "Object bank: A high-level image representation for scene classification & semantic feature sparsification," in *Advances in neural information processing systems*, 2010, pp. 1378–1386.

[6] C. H. Ding and I. Dubchak, "Multi-class protein fold recognition using support vector machines and neural networks," *Bioinformatics*, vol. 17, no. 4, pp. 349–358, 2001.

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

[8] B. Kolosnjaji, A. Zarras, G. Webster, and C. Eckert, "Deep learning for classification of malware system call sequences," in *Australasian Joint Conference on Artificial Intelligence*. Springer, 2016, pp. 137–149.

[9] J. Z. Kolter and M. A. Maloof, "Learning to detect and classify malicious executables in the wild," *Journal of Machine Learning Research*, vol. 7, no. Dec, pp. 2721–2744, 2006.

[10] B. Miller, A. Kantchelian, M. C. Tschantz, S. Afroz, R. Bachwani, R. Faizullabhoy, L. Huang, V. Shankar, T. Wu, G. Yiu *et al.*, "Reviewer integration and performance measurement for malware detection," in *Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2016, pp. 122–141.

[11] N. Nissim, A. Cohen, R. Moskovitch, A. Shabtai, M. Edry, O. Bar-Ad, and Y. Elovici, "Alpd: Active learning framework for enhancing the detection of malicious pdf files," in *Intelligence and Security Informatics Conference (JISIC), 2014 IEEE Joint*. IEEE, 2014, pp. 91–98.

[12] R. Perdisci, W. Lee, and N. Feamster, "Behavioral clustering of http-based malware and signature generation using malicious network traces." in *NSDI*, vol. 10, 2010, p. 14.

[13] K. Rieck, P. Trinius, C. Willems, and T. Holz, "Automatic analysis of malware behavior using machine learning," *Journal of Computer Security*, vol. 19, no. 4, pp. 639–668, 2011.

[14] I. Santos, F. Brezo, X. Ugarte-Pedrero, and P. G. Bringas, "Opcode sequences as representation of executables for data-mining-based unknown malware detection," *Information Sciences*, vol. 231, pp. 64–82, 2013.

[15] M. G. Schultz, E. Eskin, F. Zadok, and S. J. Stolfo, "Data mining methods for detection of new malicious executables," in *Security and Privacy, 2001. S&P 2001. Proceedings. 2001 IEEE Symposium on*. IEEE, 2001, pp. 38–49.

[16] G. Schwenk, A. Bikadorov, T. Krueger, and K. Rieck, "Autonomous learning for detection of javascript attacks: Vision or reality?" in *Proceedings of the 5th ACM workshop on Security and artificial intelligence*. ACM, 2012, pp. 93–104.

[17] N. Šrndic and P. Laskov, "Detection of malicious pdf files based on hierarchical document structure," in *Proceedings of the 20th Annual Network & Distributed System Security Symposium*, 2013.

[18] A. Deo, S. K. Dash, G. Suarez-Tangil, V. Vovk, and L. Cavallaro, "Prescience: Probabilistic guidance on the retraining conundrum for malware detection," in *Proceedings of the 2016 ACM Workshop on Artificial Intelligence and Security*. ACM, 2016, pp. 71–82.

[19] E. Gandotra, D. Bansal, and S. Sofat, "Malware analysis and classification: A survey," *Journal of Information Security*, vol. 2014, 2014.

[20] R. Jordaney, K. Sharad, S. K. Dash, Z. Wang, D. Papini, I. Nouretdinov, L. Cavallaro, and E. SpA, "Transcend: detecting concept drift in malware classification models," in *Proceedings of the 26th USENIX Security Symposium (USENIX Security 2017)*, 2017.

[21] C. Rossow, C. J. Dietrich, C. Grier, C. Kreibich, V. Paxson, N. Pohlmann, H. Bos, and M. Van Steen, "Prudent practices for designing malware experiments: Status quo and outlook," in *Security and Privacy (SP), 2012 IEEE Symposium on*. IEEE, 2012, pp. 65–79.

[22] A. Shabtai, R. Moskovitch, Y. Elovici, and C. Glezer, "Detection of malicious code by applying machine learning classifiers on static features: A state-of-the-art survey," *information security technical report*, vol. 14, no. 1, pp. 16–29, 2009.

[23] M. Sebastián, R. Rivera, P. Kotzias, and J. Caballero, "Avclass: A tool for massive malware labeling," in *International Symposium on Research in Attacks, Intrusions, and Defenses*. Springer, 2016, pp. 230–253.

[24] "Malwr, Free Malware Analysis Service," https://malwr.com/, accessed: 2017-05-26.

[25] "Cuckoo, Malware Analysis System," https://cuckoosandbox.org/, accessed: 2017-05-26.

[26] "VirusTotal," https://www.virustotal.com/, accessed: 2017-10-30.

[27] "National Software Reference Library," https://www.nist.gov/software-quality-group/national-software-reference-library-nsrl/, accessed: 2017-10-30.

[28] "VirusShare - Repository of malware samples," https://virusshare.com/, accessed: 2017-10-30.

[29] "Celery, Distributed Task Queue," http://www.celeryproject.org/, accessed: 2017-05-26.

[30] J. Friedman, T. Hastie, and R. Tibshirani, *The elements of statistical learning*. Springer series in statistics New York, 2001, vol. 1.

[31] "Jupyter - Interactive Python Notebook," http://jupyter.org/, accessed: 2017-10-30.

[32] "scikit-learn, Machine Learning in Python," http://scikit-learn.org/stable/, accessed: 2017-05-26.

[33] "Python Data Analysis Library," https://pandas.pydata.org/, accessed: 2017-05-26.