Are Machine Learning Models for Malware Detection Ready for Prime Time?

Lorenzo Cavallaro | University College London, United Kingdom
Johannes Kinder | Bundeswehr University Munich, Germany
Feargus Pendlebury | University College London, United Kingdom
Fabio Pierazzi | King's College London, United Kingdom

In academic research on malware detection, machine learning-based techniques have become the de-facto standard. Over the years, many papers have been published on the topic, using everything in the machine learning toolbox from traditional models [1, 2] to more recent neural network architectures [3, 4, 5]. With published accuracies and other performance numbers regularly close to the 100% mark, you could not be blamed for thinking that the problem is practically solved. Why would anyone be interested in continuing to use malware signatures and simple patterns?

Cannot Reproduce?

Yet, practitioners find themselves disillusioned when trying to put machine learning models from research into practice. In real-world deployments, machine learning-based malware classifiers are known to often be unreliable and are not trusted to act as the main line of defense.

Where does this discrepancy come from? It would be too easy to shrug this off as expected technical challenges in technology transfer; we must get to the bottom of the issue or risk exacerbating the reproducibility crisis decried in machine learning-based science [6]. So, what's the deal? If the same model performs so differently in production, then the lab settings that yield near-perfect performance must not be representative of how such models are deployed.

The typical research project on machine learning-based malware detection goes as follows: (1) obtain a malware dataset, from an academic project or public sources; (2) collect a dataset of benign software, e.g., through scraping repositories or app stores; (3) engineer or learn features to represent the malicious and benign apps in your datasets; and (4) train a classifier while following common good-practice guidelines, for instance to prevent overfitting [7]. Unfortunately, this seemingly straightforward approach comes with several pitfalls that may prevent your results from translating to practice [8, 9].

The first and maybe the most obvious issue is that performance metrics are influenced by the relative sizes of the classes, i.e., the ratio of malware to benign software in the evaluation dataset. To illustrate, imagine a silly malware detection model that classifies everything as malware. If your dataset consists of 95% malware and 5% benign software, then the model's performance will actually be quite decent! Widely used performance metrics are precision, recall, and F_1—the harmonic mean of the two. With precision being the fraction of true malware in everything the classifier detected as malware, we would get to a respectable 95%, because only 5% of the detections would be false. Recall is the percentage of actual malware detected, which would even be a perfect 100%. The F_1 then comes out at an impressive 97%, although the model is completely useless in practice. Now, this may be an extreme example, but the effect is gradual and gets worse the more the class ratio at testing time differs from the class ratio in the real world. This effect of misrepresenting the ratio between classes is also known as the *Base Rate Fallacy* [10]. In most practical settings, there is vastly more benign software than malware, making real datasets heavily imbalanced toward the benign class. A classifier that is good at detecting malware but misclassifies much benign software as malicious therefore has poor performance in practice. When tested on equal amounts of malware and benign software, or even on a majority of malware, its performance will be inflated.

A second issue is that, as a result of the dataset collection procedure, malicious and benign software datasets often end up being from separate time periods. It is not unusual for malware datasets to be several years old (the Drebin dataset [1] of Android malware from August 2010 to October 2012 is still commonly used today), while software scraped from app stores will be current. Because of this, they may differ in file formats, metadata, compiler versions or the API methods used. A machine learning model trained to optimally separate the two classes is then very likely to pick up on these spurious non-causal differences instead of just the actual malicious behavior. After all, if the easiest way to distinguish the old malware from the new benign applications is to look for the presence of some deprecated permission, then this is what the model will learn.

The third issue is also due to inconsistent dates in the ordering of samples. The canonical way to train and evaluate a model is to first randomly split the dataset into a training set and a test set [9]. The model is trained on the training set and evaluated against the test set. However, a model's performance on randomly chosen training and testing sets is not necessarily representative of its performance in practice! Because the partitioning is random, the training set can contain samples that would not have been available to learn from at the time it would have first encountered the samples in the test set. For instance, a randomly chosen test set can contain older versions of malware contained in the training set. This way, the classifier is allowed to learn from future knowledge that would not be available in realistic settings.

In other domains, such as natural language processing, this effect is not present or at least much less pronounced. There, past and future samples can be mixed, because the underlying *data distribution* from which the samples are drawn is mostly stable, as human language only changes slowly. In the malware detection context, the distribution of the samples changes relatively quickly. In fact, this phenomenon is common in security settings and is in part fueled by human attackers acting in an adversarial fashion [8, 11, 12]. Not only do newer malware samples use new API methods, their authors also actively try to circumvent detection and mitigation mechanisms and adapt their strategies accordingly. The resulting change in the distribution is known as *dataset shift* or *adversarial drift*, depending on the context. It makes any performance assessments partial and leaves us in uncharted territory when trying to understand the actual performance a system will deliver in practice.

We can do better

If we wish to estimate more accurately the performance of a malware classifier in a practical deployment, we have to model the experiments after reality. Of course, nothing can replace observing the performance of a system in production, but the best we can do in a lab setting is use the data we have to simulate how the system would have fared in practice against the malware that appeared in the past, replaying any dataset shifts that occurred.

First, we must ensure that the class ratio between our benign and malicious classes at test time mimics the one seen at deployment time. This means that researchers must have a reasonable understanding of what to expect "in the wild": will 50% of the applications seen be malicious, or rather 10%? In fact, this kind of understanding of the real-world environment is not just necessary for configuring the experimental setup, but also of great benefit when designing the approach in the first place. This lends yet more importance to an open exchange between practitioners and academics in the field. Do note, however, that one is free to choose different class ratios at *training time*. While a realistic class ratio at training time will minimize the overall error rate (the fraction of misclassifications over all samples), other ratios can be used to move the decision boundary and hence trade off the likelihood of false positives against that of false negatives [8].

Second, malware and goodware have to come from the same time frame, otherwise the classifier will pick up non-causal artifacts as discriminative features. This is no easy task, as datasets of malware are treated as trade secrets by IT-security firms, and benign software may not be readily available from past time periods or at all; this is true in particular for paid apps in closed source environments.

Notably, the easy availability of standard datasets has been one of the drivers in applications of machine learning to domains such as image recognition and natural language processing. If we are serious about advancing research in machine learning for security, we need to make high quality datasets publicly available. Only then can new generations of PhD students easily try out new ideas and compare approaches objectively.

The AndroZoo project [13] is a great example of such an effort. It provides access to over 20 million Android applications, both benign and malicious, from a time span of over ten years, and continues to be continuously updated. For the Windows platform, the SOREL [14] dataset contains pre-extracted features and metadata for 20 million PE files, half benign and half malicious. Unfortunately, it is typically not allowed to

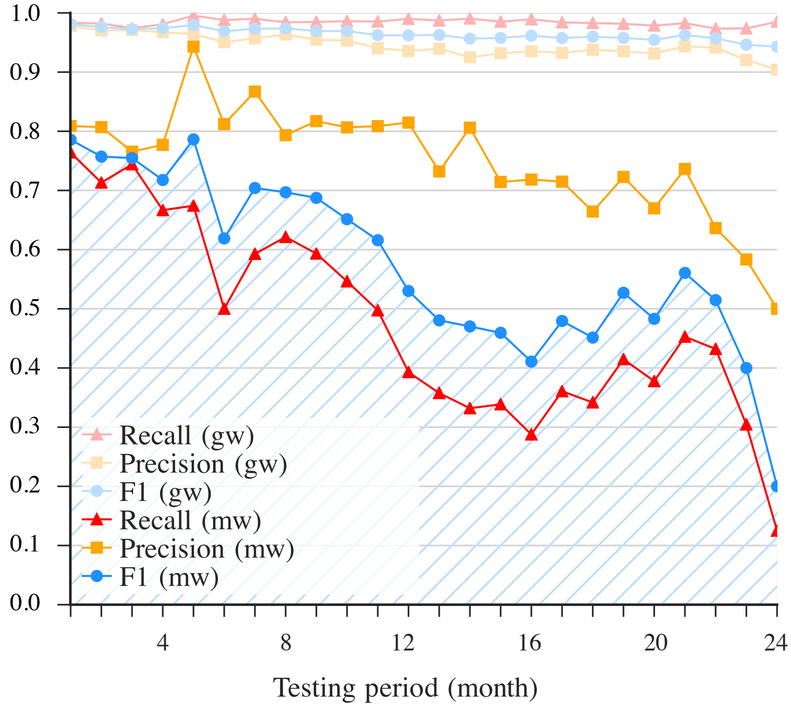

Figure 1: Performance decay in several metrics of a malware detection model trained at month 1 when evaluated against the new malware and benign software of each month, for 24 months (adapted from Pendlebury et al. [8]). Performance numbers are shown both for detecting malware (mw) and for detecting benign software (gw—goodware, in light colors), respectively. The shaded area under the F_1 curve for malware detection is the AUT, measuring robustness against decay.

freely redistribute Windows binaries, so SOREL has full binary files only for malware.

Having a dataset of timestamped malware and benign applications also allows us to avoid the third pitfall mentioned above: if the machine learning model should be used for proactive malware detection, i.e., detecting new malware as it comes in, then your training data should not be newer than your test data. For instance, this is a typical setting for a malware detector deployed at an app store that scans newly uploaded applications. Known malware may effectively be detected using signatures, so classifier performance on older specimens may be less relevant in practice. We can model this setup in an experiment through a sliding window of test data, using only applications with an earlier timestamp as training data [8]. Interestingly, this also allows us to evaluate how a classifier trained at a certain time in the past would have performed on the new malware as it comes in.

Performance Degradation

Once deployed, a trained malware classifier will degrade in performance over time if it is not regularly retrained [12, 11]. This is a result of the aforementioned dataset shift, due to malware and in fact all applications changing over time, as malicious strategies and the available APIs change.

A setup as described above, and detailed by Pendlebury et al. [8], can evaluate how the performance of classifiers of interest would have degraded in the past after some deployment date. Figure 1 shows the results for the Drebin classifier trained on a dataset containing apps up to December 2016. The classifier was tested 24 times against the new malware appearing each month, to cover a period of two years up to December 2018. The lines show how the various metrics decline, with the F_1 score on the malware class being the most commonly used metric for this type of work. Depending on the activity of individual malware campaigns, performance can rise temporarily, but the overall downward trend is clearly visible. From the plot we can also see that all performance metrics are much higher on the majority class, that is, the classifier continues to do a relatively good job in identifying most benign software. This is expected in an imbalanced setting and reaffirms the importance of picking the class ratios correctly. This type of plot lets us visualize the performance over time. For easier comparison, we can also capture the time-related decay in a single number: the AUT (area under time) is the area under the F_1 curve and provides a measure of the classifiers' robustness against dataset shifts.

A classifier has to be maintained just like any other detection mechanism. For a machine learning classifier, this essentially amounts to retraining, possibly incrementally. Designing a retraining strategy that trades off costs for labeling, training, and possibly model distribution against performance degradation is a subtle problem. We can equip classifiers with a *rejection* option to allow them to defer classification of a particular sample due to it being an outlier with respect to the known classes [11]. This allows the system to identify drift and trigger retraining.

Looking Forward

There is still much research to be done in the area, despite the host of available publications. We now have a better understanding of the performance of machine learning models for malware detection and related security tasks. With the appropriate precautions described above and fleshed out in recent work [8, 9, 6], the deployment and maintenance of machine learning-based classifiers is promising yet still a delicate matter [7]. For instance, the right abstractions to approximate program semantics, detecting and dealing with dataset shifts [11], the costs of labeling in retraining and risks in online learning strategies, and defenses from realizable adversarial attacks [15], are just some of the open research questions our community must deal with going forward.

References

[1] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, and K. Rieck. DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket. In: *Annu. Network and Distributed System Security Symposium (NDSS)*. Internet Society, 2014.

[2] L. Onwuzurike, E. Mariconti, P. Andriotis, E. D. Cristofaro, G. J. Ross, and G. Stringhini. MaMaDroid: Detecting Android Malware by Building Markov Chains of Behavioral Models (Extended Version). In: *ACM Trans. Priv. Secur.* **22**(2) (2019), 14:1–14:34.

[3] K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. McDaniel. Adversarial examples for malware detection. In: *European Symp. Research in Computer Security (ESORICS)*. Springer. 2017, pp.62–79.

[4] E. Raff, J. Barker, J. Sylvester, R. Brandon, B. Catanzaro, and C. K. Nicholas. Malware detection by eating a whole EXE. In: *AAAI Workshop on Artificial Intelligence for Cyber Security*. 2018.

[5] K. Pei, Z. Xuan, J. Yang, S. Jana, and B. Ray. Trex: Learning execution semantics from micro-traces for binary similarity. In: *arXiv preprint arXiv:2012.08680* (2020).

[6] S. Kapoor and A. Narayanan. Leakage and the Reproducibility Crisis in ML-based Science. In: *arXiv preprint arXiv:2207.07048* (2022).

[7] J. Saxe and H. Sanders. *Malware Data Science: Attack Detection and Attribution*. No Starch Press, 2018.

[8] F. Pendlebury, F. Pierazzi, R. Jordaney, J. Kinder, and L. Cavallaro. TESSERACT: Eliminating experimental bias in malware classification across space and time. In: *USENIX Security Symposium*. 2019, pp.729–746.

[9] D. Arp, E. Quiring, F. Pendlebury, A. Warnecke, F. Pierazzi, C. Wressnegger, L. Cavallaro, and K. Rieck. Dos and don'ts of machine learning in computer security. In: *USENIX Security Symposium*. 2022.

[10] S. Axelsson. The base-rate fallacy and the difficulty of intrusion detection. In: *ACM Trans. on Information and System Security (TISSEC)* **3**(3) (2000), 186–205.

[11] F. Barbero, F. Pendlebury, F. Pierazzi, and L. Cavallaro. Transcending Transcend: Revisiting Malware Classification in the Presence of Concept Drift. In: *IEEE Symp. Security and Privacy (S&P)*. IEEE, 2022.

[12] J. G. Moreno-Torres, T. Raeder, R. Alaíz-Rodríguez, N. V. Chawla, and F. Herrera. A unifying view on dataset shift in classification. In: vol. 45. 1. 2012, pp.521–530.

[13] K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon. AndroZoo: collecting millions of Android apps for the research community. In: *Proc. Int. Conf. Mining Software Repositories (MSR)*. ACM, 2016, pp.468–471.

[14] R. Harang and E. M. Rudd. SOREL-20M: A large scale benchmark dataset for malicious PE detection. In: *arXiv preprint arXiv:2012.07634* (2020).

[15] F. Pierazzi, F. Pendlebury, J. Cortellazzi, and L. Cavallaro. Intriguing properties of adversarial ml attacks in the problem space. In: *IEEE Symp. Security and Privacy (S&P)*. IEEE. 2020, pp.1332–1349.

Biographies

Lorenzo Cavallaro is Full Professor of Computer Science at University College London (UCL), London WC1E 6BT, United Kingdom, where he leads the Systems Security Research Lab. His research focuses on understanding and improving the effectiveness of machine learning methods for systems security in the presence of adversaries. In particular, Lorenzo and his lab investigate the intertwined relationships of program analysis and machine learning and the implications they have towards realizing Trustworthy ML for Systems Security.

Johannes Kinder is a Professor of Computer Science at Bundeswehr University Munich, 85577 Neubiberg, Germany. He heads the PATCH lab for Program Analysis, Transformation, Comprehension and Hardening and is a member of the Research Institute CODE and the Institute for Systems Security at the Department of Computer Science.

Feargus Pendlebury is a Visiting Scholar at University College London, London WC1E 6BT, United Kingdom. His expertise focuses on developing machine learning techniques for hostile environments where adversaries seek to evade detection.

Fabio Pierazzi is a Lecturer (Assistant Professor) in Computer Science, and Deputy Head of the Cybersecurity group at the Department of Informatics of King’s College London, London WC2R 2LS, United Kingdom. His research interests are at the intersection of systems security and machine learning, with a particular emphasis on settings in which attackers adapt quickly to new defenses.