Taking the Red Pill: Lessons Learned on Machine Learning for Computer Security

Daniel Arp | Technische Universität Berlin and University College London
Erwin Quiring | ICSI and Ruhr University Bochum
Feargus Pendlebury | University College London
Alexander Warnecke | Technische Universität Berlin and BIFOLD
Fabio Pierazzi | King's College London
Christian Wressnegger | KASTEL Security Research Labs and Karlsruher Institute of Technology
Lorenzo Cavallaro | University College London
Konrad Rieck | Technische Universität Berlin and BIFOLD

Artificial intelligence and machine learning have enabled remarkable progress in science and industry. This advancement has naturally also impacted computer security, with nearly every major vendor now marketing AI-driven solutions for threat analysis and detection. Similarly, the number of research papers applying machine learning to solve security tasks has literally exploded.

These works come with the implicit promise that learning algorithms provide significant benefits compared to traditional solutions. In recent years, however, different studies have shown that learning-based approaches often fail to provide the promised performance in practice due to various restrictions ignored in the original publications [e.g., 1, 2, 3, 4]. In this article, we want to ask: Are there *generic* pitfalls that can affect the experimental outcome when applying machine learning in security? And, if so, how can researchers avoid stepping into them?

Why Should I Care?

As a thorough researcher, one might tend to think "this can never happen to me." However, as we will discuss in this article, pitfalls come in various forms and flavors, some obvious but others noticeable only with a very cautious eye. Hence, even experienced researchers might step into them from time to time without noticing. With this work, we want to raise awareness of these issues in the research community to reduce their prevalence in security research. Detailed recommendations and guidelines for each of the pitfalls can be found in the original papaer [5].

Entering the Matrix

To illustrate the problem without pointing fingers at others, let us consider the fictional company *MetaCortex*. The company specializes in the discovery of security vulnerabilities in source code. The necessary code analysis, however, is time-consuming and tedious. To speed up the process, the company decides to leverage machine learning to automate the discovery of vulnerabilities, as corresponding methods have already been seemingly successfully applied in the literature [e.g., 6, 7].

Specifically, MetaCortex chooses to build its solution on a deep neural network due to the popularity of this learning model in other fields. And indeed, the developed model achieves an excellent performance on a common benchmark dataset, so that the company decides to integrate it into a new product and sell it to its customers, awaiting a big commercial success.

Taking the Red Pill

Unfortunately, it turns out that the model performance cannot keep up with the promises made, and MetaCortex's customers start to complain. What went wrong? A closer look at the company's machine learning pipeline uncovers various pitfalls the company accidentally stepped in while developing their model.

In the following, we discuss some of the factors that might have led to an over-estimation of the method's capabilities. Note that, while we focus on this solely fictive example, all discussed issues are inspired by actual mistakes found in previous works.

Spurious Correlations. Even though deep neural networks have led to major breakthroughs in various areas, it is often unclear *why* they achieve this impressive performance. Fortunately, in recent years, several methods have been developed that enable the interpretation of these models, shedding some light on the decision-making process of neural networks [8]. The application of an explanation method, like *Layer-wise Relevance Propaga-*

This a preprint version of an article published in IEEE Security & Privacy Magazine, September 2023

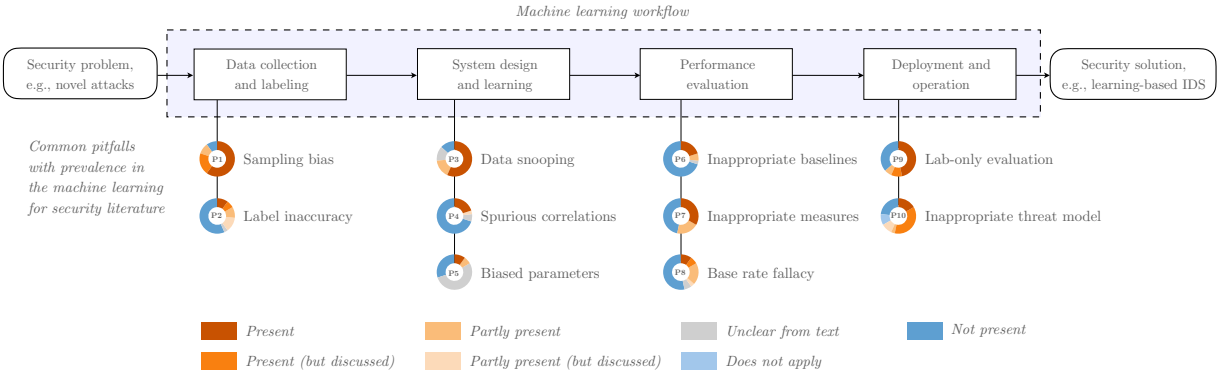

Figure 1: Common pitfalls of machine learning in computer security and their prevalence in the literature.

tion (LRP) [9], allows researchers to inspect the features most relevant to the prediction.

In the case of our fictive company, we find that the highlighted features are barely connected to security vulnerabilities. Instead, the most relevant features are meaningless tokens like brackets or commas in the source code, which have no semantic relevance to vulnerable code and thus represent non-causal *spurious correlations*. It seems as if these artifacts serve as shortcuts that allow the model to distinguish between vulnerable and non-vulnerable code. But how did this happen?

Sampling Bias. A possible and common reason for the presence of spurious correlations is sampling bias. In this case, the distribution of the training data does not sufficiently represent the distribution at test time. Consequently, the model is not able to learn the underlying concept of the given task, but rather relies on artifacts introduced in the training distribution. When composing a dataset for training the model, researchers need to be aware that there exist a variety of sources for sampling bias, some of which being very subtle. The strategy for collecting the data might thus bias the resulting dataset towards certain software versions, code authors, or programming languages. These factors are then (maybe indirectly) used by the machine learning model to distinguish between classes, instead of solving the actual task [e.g., 5, 10].

Data Snooping. Let us assume that the collected dataset does not suffer from sampling bias. Are there any other—less known–issues that might result in huge differences in the performance at training and test time? Indeed, another common pitfall that can lead to over-optimistic results is commonly referred to as *data snooping*. Here, a learning model is trained with information that would not be available in practice. While this appears to be a pitfall that can be avoided very easily at the first glance, it turns out to be much harder to avoid than expected in many cases.

The reason is that data snooping exists in many different forms, some of which can be easily overlooked. For example, using incorrect time splits that ignore time dependencies within the data can inflate the actual performance [1]. Similarly, this pitfall applies if noisy data is removed from the test set based on knowledge that would normally not be available at training time. Even more subtle, solely evaluating well-known benchmark data might also over-estimate the performance. Established benchmarks come with a history, so that researchers may unnoticeably use knowledge from prior work—including insights from the test distribution.

The Greater Picture

The previous example illustrates that there obviously exists a number of pitfalls that can potentially harm the experimental outcome. However, the previously discussed issues are just the tip of the iceberg and the MetaCortex company likely faces further problems.

In this section, we provide a systematic overview of common pitfalls and their prevalence in a set of 30 top publications we selected for our study. We follow the individual stages of a typical machine-learning pipeline that Figure 1 illustrates together with all pitfalls. The colored bar shows their prevalence in our study, with warmer colors depicting the presence of a pitfall. Table 1 summarizes each pitfall.

	Pitfall	Description
P1	Sampling Bias	The composed dataset does not sufficiently represent the actual distribution.
P2	Label Inaccuracy	The ground-truth labels are inaccurate, unstable, or erroneous.
P3	Data Snooping	Information is used at training time that is usually not available in practice.
P4	Spurious correlations	A learning model relies on false associations caused by artifacts unrelated to the task.
P5	Biased Parameter Selection	Final parameters of a learning method are indirectly determined on the test set.
P6	Inappropriate Baseline	No adequate baseline methods are used in the evaluation for comparison.
P7	Inappropriate Performance Measures	Used performance measures are not suitable for the application scenario.
P8	Base Rate Fallacy	Large class imbalance is ignored when interpreting the performance.
P9	Lab-Only Evaluation	The developed system is only tested in a laboratory setting.
P10	Inappropriate Threat Model	Attacks against the machine-learning component itself are not considered.

Table 1: Overview of common pitfalls of machine learning in computer security.

(1) Data collection and labeling phase. Before we can start with developing a new learning-based method, we first have to collect an expressive dataset that resembles the data distribution we assume to see in practice. Moreover, we also often require meaningful label information if we want to apply supervised learning. Unfortunately, the composition of a realistic dataset with labels is often challenging, leading to the first two pitfalls: (P1) *Sampling bias* and (P2) *label inaccuracy*.

We have already discussed how sampling bias can affect the experimental outcome. Similarly, in the case of P2, the labels are erroneous or unstable, which, in turn, can also impact the performance of a learning model if we do not correct this noise.

(2) System design and learning phase. Once we have composed a dataset, we can design and train our learning model. This stage includes the pre-processing of the data, the extraction of suitable features, as well as learning the actual model. In this stage, we can step in three pitfalls when not being careful: (P3) *Data snooping*, (P4) *spurious correlations*, and (P5) *biased parameter selection*.

In the case of P3 and P5, the separation of training and test partition is flawed, so that the model uses information that is unavailable at test time, biasing the outcome of the experimental setup. For instance, the developers might ignore time dependencies within the collected data, such that the machine learning model is trained on data comprising future knowledge (which is not available outside the matrix). Another issue arises from P4. Here, the feature design allows the model to pick up on artifacts unrelated to the security pattern, thus creating a shortcut for solving the actual tasks. While this can be unproblematic in some cases, it can also lead to serious problems and let the model fail completely during its deployment.

(3) Evaluation Phase. In the next stage, we evaluate the previously trained model and examine its performance on test data. Here, we have to pay attention to not step into one of the following three pitfalls: (P6) *Inappropriate baseline*, (P7) *inappropriate performance measures*, or (P8) *the base rate fallacy* [11].

In the case of P6, the learning model is not compared against suitable baseline approaches. For instance, a simple, non-learning-based method can sometimes achieve a similar or even better performance than a complex deep neural network. However, due the lack of comparison, this fact remains hidden. A similar problem arises if the chosen performance measures are not appropriate for the application scenario (P7). As an example, we often have to deal with highly imbalanced datasets in security, like in malware detection. In these cases, we have to identify malicious objects that represent only a small proportion of the entire data distribution. When using the wrong metrics in these settings, like the accuracy, one gets an entirely false estimate of the true performance of a learning-based system. Moreover, even if proper metrics are used, the performance of a system might still be overestimated by ignoring the base rate of the negative class in reality (P8). Let us assume, for example, a seemingly efficient classifier with 99% true positives at 1% false positives. Yet, if we have a class ratio of 1:100, even 1% false positives still cause 100 false positives for every 99 true positives.

(4) Deployment and Operation Phase. Finally, we obtain a learning model whose detection performance meets our requirements. We can now deploy and operate it in the wild. We might already assume that we have successfully escaped the matrix. Unfortunately, there are still two additional pitfalls that can have a severe impact on the performance.

First, we should account for any practical limitations that we did not consider throughout the evaluation. Oftentimes, new learning methods are solely evaluated in *lab-only environments* (P9), where crucial constraints of realistic settings are ignored, such as run-time or storage restrictions. As a result, a promising method might turn out to be unsuitable in a production setting. Furthermore, we need to consider *the security of our learning-based system*, as adversaries might run targeted attacks against it (P10). For instance, malicious actors could try to circumvent detection or derive information about the underlying learning model.

Bending the Spoon

Naturally, the question arises how likely each of the previously discussed pitfalls occurs. To get an intuition, we review 30 academic papers published at top conferences for security between 2011 and 2020. When selecting the papers, we ensure that they cover a wide range of security-related topics, ranging from learning-based malware detection to intelligent vulnerability discovery. If a pitfall's presence is unclear, the reviewers decide conservatively and always give the authors the benefit of the doubt. A precise description of our assessment criteria can be found in the original article [5].

Figure 1 highlights the outcome of the study. We find that the pitfalls are widespread even in top research. Each paper is affected by at least three of the discussed issues. The most prevalent pitfall is sampling bias (P1), followed by data snooping (P3), which are at least partly present in 90% and 73% of the considered publications, respectively. Similarly, other pitfalls occur frequently, such as the use of inappropriate performance measures (P7) or the evaluation in a lab-only setting (P9), both of which appear in at least 50% of all the papers. Interestingly, we find that that the presence of a pitfall is only accompanied by a discussion in 22% of the cases, indicating that there is a lack of awareness regarding these common issues.

To get a full picture of the situation, we have also collected feedback from the authors of the reviewed papers. The vast majority of the authors from which we received a response agreed that there is a lack of awareness for the identified pitfalls and confirm that these are widespread in security research.

Escaping the Matrix

The discussed pitfalls are more than just an academic problem. In fact, they introduce severe biases and hinder actual progress in research. As a result, we need to discuss within the community how to overcome these problems in the future.

First and foremost, it is possible to avoid the identified pitfalls in many cases. Therefore, we recommend double-checking each stage of the machine learning pipeline and looking out for potential issues when developing a new approach. To this end, detailed recommendations and guidelines for each of the pitfalls can be found in the original publication [5]. For instance, methods to fix inaccurate labels or methods of explainable AI (XAI) to check for spurious correlations are applicable.

Unfortunately, there exist cases in which it can be challenging to avoid a pitfall entirely. As an example, it might be hard to compensate for sampling bias due to a lack of data. In these cases, it is crucial to openly discuss the problem so that other researchers can solve it in the future. In general, we thus recommend to "do your best" by mitigating pitfalls where possible and acknowledge remaining problems openly.

Finally, we like to stress that it is not our intention to take the fun out of machine learning in security research. In fact, the opposite is true. We strive to promote sound research and bring the enormous potential of artificial intelligence into the reality of security, so that red pills are no longer necessary.

References

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

[2] M. Juarez, S. Afroz, G. Acar, C. Diaz, and R. Greenstadt. A Critical Evaluation of Website Fingerprinting Attacks. In: *Proc. of ACM Conference on Computer and Communications Security (CCS)*. 2014.

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

[4] L. Cavallaro, J. Kinder, F. Pendlebury, and F. Pierazzi. Are Machine Learning Models for Malware Detection Ready for Prime Time? In: *IEEE Security & Privacy* **21**(2) (2023), 53–56.

[5] 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: *Proc. of USENIX Security Symposium*. 2022.

[6] F. Yamaguchi, N. Golde, D. Arp, and K. Rieck. Modeling and Discovering Vulnerabilities with Code Property Graphs. In: *IEEE Symposium on Security and Privacy (S&P)*. 2014, pp.590–604.

[7] Z. Li, D. Zou, S. Xu, X. Ou, H. Jin, S. Wang, Z. Deng, and Y. Zhong. VulDeePecker: A Deep Learning-Based System for Vulnerability Detection. In: *Proc. of Network and Distributed System Security Symposium (NDSS)*. 2018.

[8] A. Warnecke, D. Arp, C. Wressnegger, and K. Rieck. Evaluating Explanation Methods for Deep Learning in Security. In: *Proc. of the IEEE European Symposium on Security and Privacy (EuroS&P)*. 2020.

[9] S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, and W. Samek. On Pixel-Wise Explanations for Non-Linear Classifier Decisions by Layer-Wise Relevance Propagation. In: *PLOS ONE* **10**(7) (July 2015), 1–46.

[10] S. Chakraborty, R. Krishna, Y. Ding, and B. Ray. Deep Learning Based Vulnerability Detection: Are We There Yet? In: *IEEE Transactions on Software Engineering* **48**(9) (2022), 3280–3296.

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