Machine Learning (In) Security: A Stream of Problems

FABRÍCIO CESCHIN, Federal University of Paraná, Brazil and Georgia Institute of Technology, USA
MARCUS BOTACIN, Texas A&M University, USA
ALBERT BIFET, University of Waikato, New Zealand
BERNHARD PFAHRINGER, University of Waikato, New Zealand
LUIZ S. OLIVEIRA, Federal University of Paraná, Brazil
HEITOR MURILO GOMES, Victoria University of Wellington, New Zealand
ANDRÉ GRÉGIO, Federal University of Paraná, Brazil

Machine Learning (ML) has been widely applied to cybersecurity and is considered state-of-the-art for solving many of the open issues in that field. However, it is very difficult to evaluate how good the produced solutions are, since the challenges faced in security may not appear in other areas. One of these challenges is the concept drift, which increases the existing arms race between attackers and defenders: malicious actors can always create novel threats to overcome the defense solutions, which may not consider them in some approaches. Due to this, it is essential to know how to properly build and evaluate an ML-based security solution. In this paper, we identify, detail, and discuss the main challenges in the correct application of ML techniques to cybersecurity data. We evaluate how concept drift, evolution, delayed labels, and adversarial ML impact the existing solutions. Moreover, we address how issues related to data collection affect the quality of the results presented in the security literature, showing that new strategies are needed to improve current solutions. Finally, we present how existing solutions may fail under certain circumstances, and propose mitigations to them, presenting a novel checklist to help the development of future ML solutions for cybersecurity.

ACM Reference Format:
Fabrício Ceschin, Marcus Botacin, Albert Bifet, Bernhard Pfahringer, Luiz S. Oliveira, Heitor Murilo Gomes, and André Grégio. 2023. Machine Learning (In) Security: A Stream of Problems. 1, 1 (September 2023), 34 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The massive amount of data produced daily demands automated solutions capable of keeping Machine Learning (ML) models updated and working properly, even with all emerging threats that constantly try to evade these models. This arms race between attackers and defenders moves the cybersecurity research forward: malicious

Authors' addresses: Fabrício Ceschin, fjoceschin@inf.ufpr.br, Federal University of Paraná, Rua Cel. Francisco H. dos Santos, 100, Curitiba, PR, 81531-980, Brazil and Georgia Institute of Technology, 756 West Peachtree Street NW, Atlanta, GA, 30308-4016, USA; Marcus Botacin, botacin@tamu.edu, Texas A&M University, Department of Computer Science & Engineering, College Station, TX, 77843-3127, USA; Albert Bifet, abifet@waikato.ac.nz, University of Waikato, Department of Computer Science, Waikato, Hamilton, , New Zealand; Bernhard Pfahringer, bernhard@waikato.ac.nz, University of Waikato, Department of Computer Science, Waikato, Hamilton, , New Zealand; Luiz S. Oliveira, lesoliveira@inf.ufpr.br, Federal University of Paraná, Rua Cel. Francisco H. dos Santos, 100, Curitiba, PR, 81531-980, Brazil; Heitor Murilo Gomes, heitor.gomes@vuw.ac.nz, Victoria University of Wellington, School of Engineering and Computer Science, Wellington, , , New Zealand; André Grégio, gregio@inf.ufpr.br, Federal University of Paraná, Rua Cel. Francisco H. dos Santos, 100, Curitiba, PR, 81531-980, Brazil.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2023 Association for Computing Machinery.

XXXX-XXXX/2023/9-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

actors are continuously creating new variants of attacks, exploring new vulnerabilities, and crafting adversarial samples, whereas researchers are trying to counter those threats and improve detection models. For instance, 68% of phishing emails blocked by GMail are different from day to day [30], requiring Google to update and adapt its security components regularly. In addition, 30% of Windows malware released daily may belong to unknown families, which requires ML models to be updated with behavioral reports to detect drifting samples [151]. Furthermore, at least 53% of organizations are deploying AI in different areas of cybersecurity [91]. Thus, applying ML on cybersecurity is a challenging endeavor. One of the main challenges is the volatility of the data used for building models as attackers constantly develop adversarial samples to avoid detection [6, 34, 36]. This leads to a situation where the models need to be constantly updated to keep track of new attacks. Another challenge is related to the application of fully supervised methods since the class labels tend to depict an extreme imbalance, i.e., dozens of attacks among thousands of benign samples. Labeling such instances is also problematic as it requires domain knowledge and it can detain the learning method, i.e. the analyst labeling the data is a bottleneck in the learning process. This motivates the development of semi-supervised and anomaly detection methods [9, 49].

To improve the process of continuously updating an ML cybersecurity solution, the adoption of stream learning (also incremental learning or online learning) algorithms are recommended so they can operate in real-time using a reasonable amount of resources, considering that we have limited time and memory to process each new sample incrementally, predict samples at any time, and adapt to changes [20]. However, many works in the literature do not consider these challenges when proposing a solution, which makes them not feasible in reality. Thus, multiple cybersecurity research papers cannot be straightforwardly applied to solve real-world problems and sometimes are not focused on “machine learning that matters” [155], i.e., there is a high discrepancy between research and practice [8].

Some previous work reported the relevance of some of these problems and provided research directions. Ede et al. studied how to automatically correlate security events and automate parts of the security operator workload in a semi-supervised manner [49]. Papernot et al. systematized findings on ML security and privacy, focusing on attacks identified on ML systems and defense solutions, creating a threat model for ML, and categorizing attacks and defenses within an adversarial framework [117]. Maiorca et al. explored adversarial attacks against PDF (Portable Document Format) malware detectors, highlighting how the arms race between attackers and defenders has evolved over the last decade [100]. Arnaldo et al. described a set of challenges faced when developing a real cybersecurity ML platform, stating that many research papers are not valid in many use cases, with a special focus on label acquisition and model deployment [10]. Kaur et al. presented a comparative analysis of some approaches used to deal with imbalanced data (pre-processing methods, algorithmic-centered techniques, and hybrid ones), applying them to different data distributions and application areas [86]. Gibert et al. listed a set of methods and features used in a traditional ML workflow for malware detection and classification in the literature with emphasis on deep learning approaches, exploring some of their limitations and challenges, such as class imbalance, open benchmarks, concept drift, adversarial learning and interpretability of the models [62].

Boenisch et al. investigated the security and privacy awareness of ML practitioners through an online survey and concluded that their awareness of threats and ML security practices is relatively low, as well as their familiarity with security and privacy libraries for ML [24], which evidence that this kind of study is very important for the community. Grosse et al. found that there are occurrences of adversarial ML attacks but regular security threats still pose a larger concern in industry [69]. The authors deduce that adversarial machine learning in practice is not as common as regular security problems, but that monitoring ML security might be beneficial.Bieringer et al. showed that practitioners confuse ML security with threats that are not directly related to ML and consider that the security of ML is related to the entire workflow that consists of multiple components [19]. Apruzzese et al. highlight the lists of all the tasks where ML outperforms traditional security mechanisms and reveal some of the challenges that require the contribution of all stakeholders involved to improve the quality of ML-based security solutions [8]. Arp et al. conducted a study of 30 papers from top-tier security conferences within the past 10 years and showed that many pitfalls are widespread in the current security literature [11]. Finally, Apruzzese et al. state positions that increase the real-world impact of future research, bringing researchers and practitioners closer to improving the security of ML-based solutions [6].

In this work, we present a broad collection of gaps, pitfalls, and challenges that are still a problem in many scenarios of ML solutions applied in cybersecurity, which may overlap with other areas, suggesting, in some cases, possible mitigation for them. As a study case, in most cases, we focus on malware detection or classification tasks (for Android, Windows, and Linux operational systems) given that they may contain all the problems listed. We want to acknowledge that we are not pointing fingers at anyone, given that our own work is subject to many problems stated here. We also understand that, despite many problems stated, it does not mean that the findings of research papers are not useful from a practical perspective. Our main contributions are the following:

- We show that many of the pitfalls are related to improperly considering the time when proposing solutions. This problem may be present in all the ML pipeline, from the data collection to the evaluation steps, and may be related to not considering the problem as a data stream.
- We advocate that more observational studies (i.e., research that focuses on analyzing ecosystem landscape, platforms, or specific types of attacks), are required so to the creation of useful datasets and solutions.
- We state that different approaches for attribute extraction impose varied costs, and might also lead to distinct ML outcomes. Thus, it is important to understand that each ML model and feature extraction algorithm serve different threat models, and it should be considered when evaluating a solution to avoid common pitfalls, such as applying the same criteria for online and offline applications.
- We show that concept drift detectors do no work in practice as intended in realistic experimental scenarios. Thus, new drift detection strategies that consider the challenges related to cybersecurity, such as the delay of labels and class imbalance, are needed to produce better solutions.
- We point directions to future cybersecurity research works that make use of ML, aiming to improve their quality to be used in real applications. To do so, we developed a novel checklist[1] to help the development of future ML solutions for cybersecurity based on the challenges, pitfalls, and problems reported in this work. Thus, anyone developing or reviewing a solution can use this checklist and get feedback reporting what could be improved or corrected according to our findings.

This paper is organized as follows: first, in Section 2, we discuss how to identify the correct ML task for a given cybersecurity problem; Further, this work is organized according to each step of the pipeline to develop ML solutions to cybersecurity (also presented in Section 2), including data collection (obtaining data for the ML solution), in Section 3 , attribute extraction (extracting metadata from the data previously obtained) in Section 4, feature extraction (extracting features from the attributes collected) in Section 5, model (training and updating the model using the features extracted) in Section 6, and evaluation (evaluating the proposed solution) in Section 7. In Section 8 we discuss how ML applications should be understood, their limitations, and existing open gaps. Finally, we conclude our work in Section 9.

2 ON THE MODELING OF SECURITY TASKS AS MACHINE LEARNING PROBLEMS

ML has been applied in cybersecurity to solve different tasks. Table 1 presents illustrative examples we found in the literature on how different ML approaches can be used to solve different security problems. Our goal is not to present a new taxonomy or an exhaustive list of approaches, but to highlight that, on the one hand, (i) the same security problem might be addressed via different ML approaches and, on the other hand, (ii) the same ML solutions might be used for different security tasks and at different stages of a solution pipeline.

[1] The draft version of the checklist is available at https://secret.inf.ufpr.br/machine-learning-in-security-checklist/.

Table 1. **Representative Examples of Applications of ML to Cybersecurity**. Distinct approaches are applied according to the specific security need. The same approach might be used to solve distinct security tasks. The placement of the solution in each pipeline step will present different pros and cons.

Security Task	Specialized Task	ML Task
Attack Detection	Malware Detection	Classification [12]
	Intrusion Detection	Outlier Detection [93]
	Spam Filtering	Classification [115]
Incident Response	Malware Labelling	Clustering [136]
	Log Analysis	Clustering [50]
Security Analysis	Malware Analysis	Reinforcement Learning/Ranking [53]
	Risk Assessment	Regression [67]
System Forensics	Attribution	Clustering [128]
	Object Recognition	Dimensionality Reduction [87]

Observation: The same security problem presents different pros and cons depending on the adopted ML model. Attack detection tasks are typically modeled via binary classifiers that learn what are the benign and malicious classes. This strategy often achieves a reasonable detection level on the detection of known or similar attacks, but they are hardly ever able to detect new attacks (0 days). Modeling the attack detection problem as an outlier detection problem, in turn, tends to increase the ability to detect new threats. In this modeling, the ML model learns what is considered the normal behavior of the system and detects any deviation. A drawback of this approach in comparison to the typical classification problem is that it is hard to explain the detection result, as the model does not have knowledge of the attack class, but only about the normal profile. This difference might be key for choosing one or another approach for a given application, depending on the requirements for that scenario.

In some cases, the ML techniques present similar characteristics, such that they can be applied to multiple security tasks. For instance, clustering can be used as a classification approach in tasks ranging from (i) Adding malware variants to the same bucket for triaging; (ii) Adding security logs in a bucket of similar events; to (iii) Adding forensic evidence in similar buckets depending on the attacker authorship. In other cases, the nature of the ML solution must be as different from the others as the nature of the security task they need to solve. For instance, when malware analysts are reversing engineering samples, they want to know which strings are more informative about the malware nature. For that, they need a ranking algorithm, not a binary classifier.

Challenge: Selecting the proper ML task to solve the Security task. Whereas multiple ML tasks can be applied to security tasks, the identification of the right task for a given scenario is far from straightforward, and requires reasoning about multiple corner conditions, given that cybersecurity solutions are systems that may have multiple applications of ML integrated within a complex pipeline. For instance, for the malware detection case, it is usual to find in the literature multiple solutions proposing a ML-based engine to be applied by antiviruses (AVs). Most of these solutions are initially modeled as a classification problem (goodware vs. malware), which suffices for the detection task, but does not cover AV operation as a whole. In practice, AVs provide more than detection labels [26], they also attribute malware samples to families (e.g., ransomware, banker, and so on) to present a family label, which is essential to allow incident response procedures. Therefore, a ML engine [27] for an AV should also be modeled as a family attribution problem, which requires the application of both classifying and clustering algorithms [38].

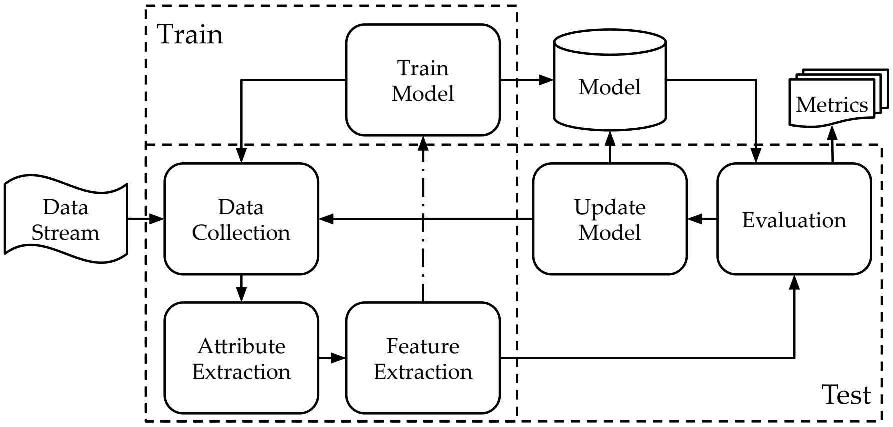


Fig. 1. **Pipeline to develop and evaluate Machine Learning solutions for cybersecurity.** Each step is executed in sequence, considering that (i) the model is first trained and then (ii) tested/evaluated and updated (if needed).

2.1 Machine Learning Pipeline for Cybersecurity

In Fig. 1, we show a pipeline to develop and evaluate ML solutions (both supervised and unsupervised methods) for cybersecurity based on the literature, which is similar to any pattern classification task. It consists of two phases: train, i.e., training a model with the data available at a given time (for supervised methods, using the available labels and features to create decision boundaries, for unsupervised, using only the features to create clusters) and test, i.e., testing it considering the new data collected (for both supervised and unsupervised, using their labels – to check if they were correctly predicted – and features to update decision boundaries and clusters). Note that we defined two steps after the data collection, given that, in many cybersecurity tasks, the raw data collected needs to have their metadata (attributes) extracted before actually being used by the ML model as features (attributes processed by a feature extractor). We understand that these steps may overlap in some cases, but for a more fine-grain discussion, we analyzed them separately. Also, it is important to notice that security data, the input of the process, is available from a data stream that is in constant production, and the model and its metrics, the output of the process, are produced during the execution of the scheme. Thus, different from many works in the literature, we focus on problems related to data streams since they are close to real-world solutions for cybersecurity [35].

In the following sections, we discuss the challenges and pitfalls of multiple modeling strategies. Our goal is to help the reader to spot the quirks of ML applications at their multiple steps and abstraction layers.

3 DATA COLLECTION CHALLENGES

The quality, quantity, and distribution of data inputted to a Machine Learning algorithm (dataset) are the basis of an adequate learning process, since ML algorithms rely on the samples presented to them in the training step and the resulting model will allow further decision making. Thus, data collection, which comprises acquisition, enrichment/augmentation, and labeling [68, 129], may be one of the most challenging steps of an ML project [130]. Besides real-world problems take these steps into consideration [64], certain research works might miss some of them. This might happen either due to the format of the dataset used or external reasons, such as privacy requirements about the data (might need differential privacy [48]), local laws that prevent the distribution of potentially harmful pieces of code, ease of accomplishing reproducibility of the experiments, and so on. Considering the steps performed in ML-based cybersecurity systems [133] and to provide a straightforward discussion, we assume that data can be available in three formats, which may overlap according to the task at hand or the chosen approach:

- **Raw Data:** data available in the same way they were collected, usually used to analyze their original behavior. For example: PE executables [37], ELF [153], APK packages [3], or network traffic data captured in PCAP format [73, 111, 114, 140]. Compared to computer vision, the raw data are the images collected to create an image recognition model [43]. In cybersecurity, AndroZoo [3] provides many APK packages collected from several sources that could be used for malware detection systems or forensics analysis;
- **Attributes:** filtered metadata extracted from the raw data with less noise and focus on the data that matters, being very suited for ML. For example, CSV with metadata, execution logs of software or data extracted from its header [5, 37], or a summary of information from a subset of network traffic data. In computer vision, the attributes would be the gradient images extracted from the original ones [43]. EMBER [5] is a good example of a dataset containing attributes, available in JSON, extracted statically from raw data (Windows PE files). DREBIN [12] is another example containing static attributes extracted from APK packages.
- **Features:** features extracted from the attributes or raw data that distinguish samples, ready to be used in a classifier. For example the transformation of logs into feature vectors for every software mentioned in the previous item, whose positions of this vector correspond to the features [37], or a transformation of traffic data into features containing the frequency of pre-determined attributes (i.e., protocols, amount of packages, bytes, etc.). In an image recognition problem, the features would be the histogram of gradients extracted using the gradient images created before [43]. This type of data is usually used by researchers to make their experiments faster, given that they extract the features once and can share them to use as input for their models.

The data collection or even any of the dataset formats above are susceptible to problems that may affect the whole process of using ML in any cybersecurity scenario and are frequently seen in the literature, such as data leakage (or temporal inconsistency), data labeling, class imbalance, and anchor bias.

3.1 Data Leakage (Temporal Inconsistency)

To evaluate a ML system, it is common to split the dataset into at least two sets: one to train the model and another to test it. The k-fold cross-validation is used to create k partitions and evaluate a given model k times, using one of the partitions as a test set and the remaining as a training set, taking the average of the metrics as a final result [105]. This is a common practice for many ML experiments, such as image classification, where temporal information may not be important and is used in batches. However, when considering cybersecurity data, which are obtained from a stream, it is not real to consider data from different or mixed epochs to train and test a model (known as data leakage [85], data snooping [11], or temporal inconsistency), given that it could increase the detection accuracy because the model knows how future threats are (i.e., the model is exposed to test data when it was trained) [37]. For instance, consider a malware detector that works similarly to an antivirus, i.e., given a software, our model wants to know if it is malware or not to block an unwanted behavior. To create this model we train it using a set of malicious and benign software that are known and were seen in the past. Then, this model is used to detect if files seen in the future are malign, even if they perform totally different attacks than before. These new threats will just be used to update the model after they are known and labeled, which will probably increase the coverage of different unknown attacks, detecting more malware than before.

Due to this temporal inconsistency problem, it is very important to collect the timestamp of the samples during the data collection and it is something not considered by some authors in cybersecurity. For instance, the DREBIN dataset [12] (malware detection) makes all the malicious APKs available and also their attributes, but does not include the timestamp that they were found in the wild, making all the research works that use this dataset exposed to data leakage. We understand that sometimes it is difficult to set a specific release date for a sample, but they are needed to avoid this problem. For malware detection, for example, researchers usually use the date when they were first seen in VirusTotal as a timestamp [37, 84, 121, 154].

3.2 Data Labeling

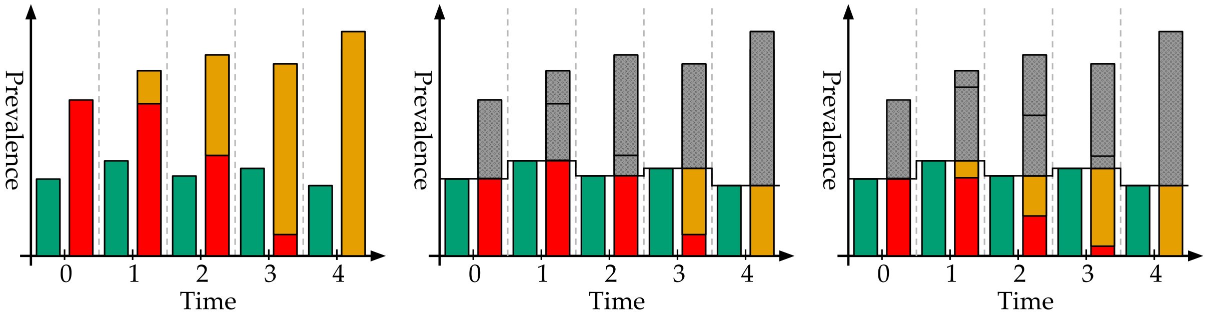
Labeling artifacts is essential to training and evaluating models. However, having artifacts properly labeled is as hard as collecting the malicious artifact themselves, as previously discussed. In many cases, researchers leverage datasets of crawled artifacts and make a strong hypothesis about them, such as "all samples collected by crawling a blacklist are malicious", or "all samples collected from an App Store are benign". These hypotheses are strong because previous work has demonstrated that even samples downloaded from known sources might be trojanized with malware [25]. Therefore, a model trained based on this assumption would make the model also learn some malicious behaviors as legitimate. Also, some labels may be inaccurate, unstable, or erroneous, which may affect the overall classification performance of ML-based solutions [11].

A common practice to obtain labels and mitigate the aforementioned problems is to rely on the labels assigned by AVs by using the VirusTotal service [154], which provides detection results based on more than sixty antivirus engines available on the platform. Unfortunately, AV labels are not uniform [26], with each vendor reporting a distinct label. Therefore, researchers have either to select a specific AV to label their samples (according to their established criteria) or adopt a committee-based approach to unify the labels. For Windows malware, AVClass [136] is widely used for this purpose and, for Android malware, Euphony [78] is used. Both of these techniques were evaluated by the authors using a high number of malware (8.9 million in AVClass and more than 400 thousand samples in Euphony), obtaining a significant F-measure score (bigger than 90% in both cases) and generating consistent results to create real datasets such as AndroZoo [3], which uses Euphony [78] to generate malware family labels. Although AVs can mitigate the labeling problem, their use should consider the drawbacks of AV internals. AVs provide two labels: detection and family attribution. Both the detection label as well as the family attribution label change very often over time: newly-released samples are often assigned a generic and/or heuristic label at the initial time and this is further evolved as new, more specific signatures are added to the AV to detect this threat class. Therefore, the date on which the samples are labeled might significantly affect the ML results. Recent research shows that AVs labels might not establish before 20 or 30 days after a new threat is released [26]. To mitigate this problem, delayed evaluation approaches should be considered, as shown in Section 7.

3.3 Class Imbalance

Class imbalance is a problem in which the data distribution between the classes of a dataset differs relatively by a substantial margin, and it is usually present in many research works [86]. If we consider the Android landscape, the AndroZoo dataset contains only about 18% of malicious apps (the remaining apps are considered benign [3, 121]). This makes the Android malware detection problem more challenging due to the presence of imbalanced data. There are several methods in the literature whose aim is to overcome this problem by making use of pre-processing techniques, improving the learning process (cost-sensitive learning), or using ensemble learning methods [64, 86]. The two latter methods will be discussed in Section 6, since they are part of the process of training/updating the model. The former method (pre-processing) relies on resampling, i.e., removing instances from the majority classes (undersampling) or creating synthetic instances for the minority classes (oversampling) [64].

In the context of cybersecurity, undersampling may affect the dataset representation, given that removing some samples from a certain class can affect its detection. For instance, considering malware detection, removing malware samples may reduce the detection of some malware families (the ones that had samples removed) and



(a) **No undersampling.** Dataset original distribution without undersampling.

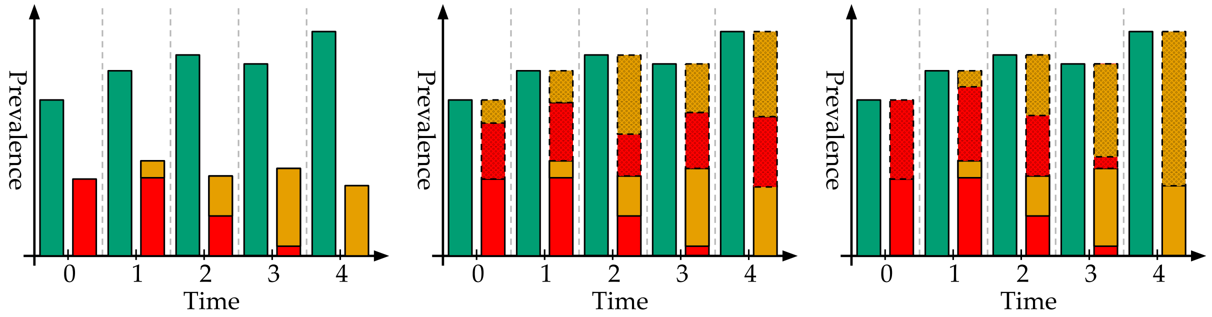
(b) **Undersampling with no temporal information.** Undersampling dataset distribution when simply removing samples to keep the same distribution by class.

(c) **Undersampling with temporal information.** Undersampling dataset distribution when keeping the same proportion of sub-classes of a given class.

Fig. 2. **Undersampling examples in a dataset.** Samples in green and red/orange represent different classes. Red and orange colors represent different sub-classes or concepts of the same class. The gray color with circles represents ignored/removed instances.

also make their prevalence in a given time (monthly or weekly) less important than reality when creating a dataset, possibly not capturing a concept drift or sub-classes of the majority classes. Fig. 2 presents a hypothetical dataset distribution with two classes (green and red/orange), one of the classes has two concepts or two sub-classes (red and orange) as time goes by, i.e., consider it as being a dataset with normal and malicious behavior, the malicious one behavior that evolves or has sub-classes of behaviors as time goes by, something that is known to happen in real-world cybersecurity datasets [37]. In Fig. 2a we can see the original distribution of the dataset, with no technique applied. In Fig. 2b we can see the dataset distribution when removing samples (in gray) to keep the same distribution by class, which results in a different scenario than the reality since the orange concept or sub-class is not seen in some periods. In Fig. 2c we see the ideal scenario for undersampling when the proportion of concepts or sub-classes is the same for the same period while keeping the same number of samples for both classes (this may be a mitigation for the undersampling problem, but even with this solution important samples may be discarded).

An example of the oversampling technique is SMOTE, which consists in selecting samples that are close in the feature space, drawing a line between them, and generating new samples along with it [39]. Although such techniques are interesting, they may generate results that produce data leakage if they do not consider the time when creating synthetic samples. For instance, consider Fig. 3, where we present again a hypothetical dataset distribution with two classes, with the same classes and problems as Fig. 2. In the first case, in Fig. 3a, we see the original dataset distribution, where the class red/orange is the minority one. In Fig. 3b the problem of data leakage is shown: the artificial data generated (with dashed lines and circles) is based on all the dataset, without considering any temporal information. Thus, we can see the orange concept/sub-class at time 0 in the synthetic instances, which does not represent the real distribution of this class at that time (this problem happens all the time in this case, with the red concept/sub-class being shown even at time 4). In contrast, in Fig. 3c we can see an oversampling technique that considers the temporal information, generating synthetic data that correspond to the actual concept/sub-class of a given time, resulting in the same distribution of concepts/sub-classes as the original data. Despite also being an interesting approach, for the cybersecurity context, oversampling works at



(a) **No oversampling.** Dataset original distribution without oversampling.

(b) **Oversampling with no temporal information.** Oversampling dataset distribution without considering temporal information.

(c) **Oversampling with temporal information.** Oversampling dataset distribution considering temporal information.

Fig. 3. **Oversampling examples in a dataset.** Samples in green and red/orange represent different classes. Red and orange colors represent different sub-classes or concepts of the same class. Samples with dashed line and circles are the synthetic instances created by oversampling.

the feature level, which means that it may not generate synthetic raw data. For instance, if we consider these data as being applications, oversampling will only create synthetic feature vectors and not real applications that work.

3.4 Dataset Size Definition

A major challenge in creating a dataset is to define its size. Although it is a problem that affects all ML domains, it has particularly severe implications in cybersecurity. On the one hand, a small dataset may not be representative of a real scenario, which invalidates the results of possible experiments using it. Also, some models may not generalize enough and not work as intended, presenting bad classification performance [68]. Limited evaluations are often seen in the cybersecurity context because collecting real malicious artifacts is a hard task, as most organizations do not share the threats that affect them to not reveal their vulnerabilities. On the other hand, a big dataset may result in long training time, and produce too complex models and decision boundaries (according to the classifier and parameters used) that are not feasible in reality (e.g., real-time models for resource-constrained devices), such as some deep learning models that usually requires a large amount of data to achieve good results [109]. As an analogy, consider that a dataset is a map and, for instance, represents a city. It is almost impossible that this map has the same scale as the city, however, it can be useful by representing what is needed for a given purpose. For example, a map of public transportation routes is enough if our objective is to use it, but if we need to visit touristic places, this very same map will not be useful, i.e., new data or a new map is required [145]. These circumstances reflect the ideas of both George Box and Alfred Korzybski. Box said that "essentially, all models are wrong, but some are useful" [141], i.e., a model built using a dataset might be useful for a given task, but it will not be perfect – there will be errors, we just need to know how wrong they have to be – and it will not be good for other tasks, which makes necessary to collect more (or new) data and select new parameters for a new model. Korzybski mentioned that "the map is not the territory" [88], meaning that it can be seen as a symbol, index, or representation of it, but it is not the place itself and it is prone to errors. In our case, a model or a dataset can represent a real scenario, but it is just a representation of it and may contain errors. These errors may present results that do not reflect reality. For instance, considering a malware detection task that uses grayscale images as representation for Windows binaries, when using a dataset that represents the real scenario with no data filtering, i.e., without removing specific malware families, the accuracy ($\approx$ 35%) was much worse

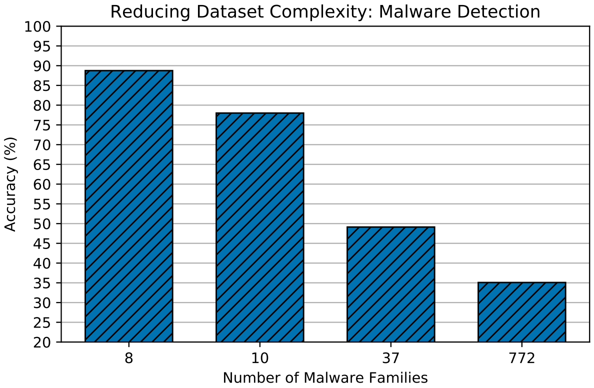


Fig. 4. **Reducing dataset complexity.** The more the dataset is filtered, the bigger the accuracy achieved, which means that researchers must avoid filtered datasets to not produce misleading results [18].

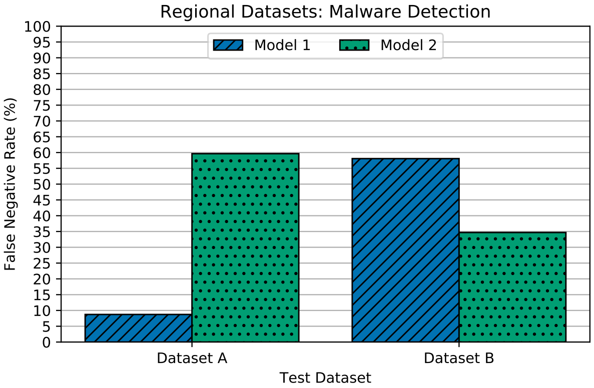
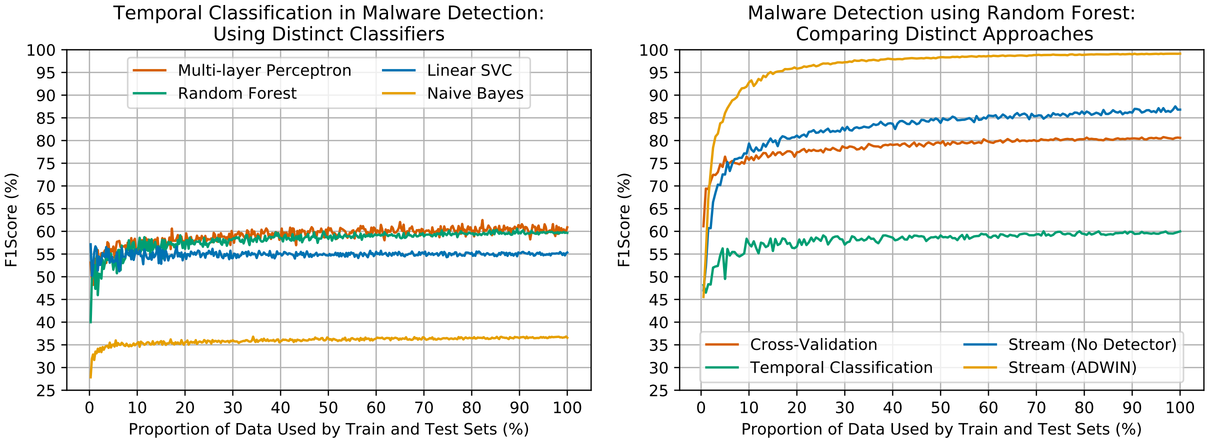


Fig. 5. **Regional datasets.** Models may have a bias towards the scenario where the dataset used to train them was collected, indicating that they need to be specially crafted in some cases [36].

than other scenarios where the authors filter the number of malware families, reducing the complexity of the problem and achieving almost 90% of accuracy, as shown in Fig. 4 [18].

Another point to consider when building a cybersecurity solution is that they may have regional characteristics that are reflected in the dataset and, as a consequence, in the model and its predictions [36]. For instance, considering a malware detection task with two models that are based on classification trees: Model 1 (random forest with PE metadata [37]) is trained using data from region A (BRMalware dataset [37]) and Model 2 (LightGBM with PE metadata [5]) is trained with data from region B (EMBER dataset [5]). Ceschin et al. showed that when testing both models with test data from both regions (test dataset A, from region A, and test dataset B, from region B), they perform better in their respective regions and present a much higher false negative rate (FNR) in the opposite region (allowing malware to be executed if this solution is applied in a different region that it was designed for, for example), as shown in Fig. 5. Thus, it is important to consider collecting new data when implanting a known solution to a new target scenario (or region).

To elaborate our discussion about dataset size definition, we created two experiments to better understand how much data we need to achieve representative results using a subset of AndroZoo dataset [3, 35] for malware detection, composed of 347, 444 samples (267, 342 benign and 80, 102 malicious applications) spanning from 2016 to 2018. Both experiments consist in understanding how much data we need to achieve a stable classification performance based on the dataset proportion used to train our models. To do so, we divided the dataset by months and reduced the proportion of goodware and malware in both the training and test set (the first half of the dataset, ordered by their first seen date in virus total, is used to train and the second, to test). Thus, in the first experiment, we tested different classifiers using temporal classification (using the training set with “known data” to train the models and ”future data“ to test them), with different proportions of data in the training and test dataset, to see how each one of them was going to perform. Surprisingly, all classifiers (Multi-layer Perceptron, Linear SVC, Random Forest, and Naive Bayes [120]) had similar behaviors, also presenting similar curves as consequence and reaching an “optimal” classification performance by using only around 10% to 20% of the original dataset, with multi-layer perceptron and random forest achieving the best overall results, as shown in Fig. 6a. It is clear to see that, after these proportions, the f1score was almost stable, improving just a little even when increasing a lot the amount of data used. Furthermore, in the second experiment, we compared different approaches in a unique classifier (Random Forest) to see if they present the same behavior as the first one. To do so, we used only



(a) **Comparing classifiers.** "Optimal" classification performance is achieved by using only around 10% to 20% of the original dataset in all cases.

(b) **Comparing approaches.** All of them stabilize their f1score after 30% to 40% proportions, less than half of the original dataset.

Fig. 6. **Dataset size definition in terms of f1score.** In both experiments a similar behaviour is seen, with a certain stability in classification performance after using only a given proportion of the original dataset.

random forest in four different approaches: cross-validation (randomly selecting train and test samples, with data leakage), temporal classification (the same approach as the first experiment), and stream with and without drift detector (initializing a stream version of the random forest [63] – known as adaptive random forest – with the training data and incrementally updating it with testing data, using the ADWIN drift detector [21] to detect changes in data distribution when it is enabled). As we can see in Fig. 6b, all the approaches also presented similar behavior and curves, but this time, all of them started to stabilize their f1score after 30% to 40% proportions, which means that they just need at least half of the dataset to present almost the same result as using the entire dataset. With both experiments, we conclude that, at some point, it may be more important to look for new features or representation strategies than to add more data to the training set, given that the classification performance is not improved so much according to our experiments.

The implication of these findings for cybersecurity is that more observational studies – research that focuses on analyzing ecosystem landscape, platforms, or specific types of attacks to inform the development of future solutions – are required so to allow for the creation of useful datasets and ML solutions. Due to the parsimonious number of this kind of study, one might fall for the Anchor bias [51], a cognitive bias where an individual relies too heavily on an initial piece of information offered (the "anchor") during decision-making to make subsequent judgments. When the value of the anchor is set, future decisions are made using it as a baseline. For instance, if we consider research that uses a one million samples dataset, it is going to become the anchor for future research, even if this dataset is not consistent with the real scenario. Thus, choosing or creating a dataset is not about its size, but its representation of the real world for the task being performed, as a map. In addition, concept drift and evolution are the nature of cybersecurity datasets, which makes it necessary to collect samples from different epochs to correctly evaluate any solution [37, 62]. We acknowledge here that some approaches need more data to achieve better results, such as deep learning techniques [152], and it may be a limitation for them. A good example of building a real dataset is the one built by Pendlebury et al. for malware detection, where the proportion of samples found in the dataset is the same found in the wild by AndroZoo collection of Android applications [3, 121].

4 ATTRIBUTE EXTRACTION

Extracting attributes, i.e., selecting filtered metadata collected from the raw data, is a key step to creating useful features for the ML models. In this section, we pinpoint the impact of different attributes in ML solutions. Note that we have separate definitions for attributes and features (see Section 3): the former is filtered metadata extracted from the raw data obtained in the data collection step and strictly related to the security task being performed, whereas the latter is the attributes transformation into a distinct set of samples representation ready to serve as input to a model. They usually cannot be directly used by an ML model, given that they need to be preprocessed and/or transformed into features (generally, numerical) to be used as input of a classifier.

4.1 The Impact of Different Attributes

Different approaches for attribute extraction impose varied costs, and might also lead to distinct ML outcomes. Naturally, executing an artifact to extract feature costs more than statically inspecting it, but the precision of the classification approach might get higher if this data is properly used. Therefore, the selection of the attribute extraction procedure should consider its effect on the outcome. Galante et al. show the effect of attribute extraction procedures over three distinct pairs of models to detect Linux malware (SVM with RBF kernel, Multi-layer Perceptron, and Random Forest) [57]. These three pairs of models consider the same set of features, one of them considering statical attributes and the other, dynamic attributes. For instance, in statically extracted attributes, the authors consider the presence of the `fork` system call in the import table of the sample's binary to build a feature vector, whereas, in the corresponding dynamic attributes, they consider the frequency of the fork system call invocation during the sample execution in a sandbox to build a feature vector as well. The authors used a balanced dataset of Linux binaries with benign and malign applications as input to all of the aforementioned models and the outcome was that the dynamic extraction approach outperformed the static approach, as shown in the accuracy rates (Fig. 7). Although the dynamic attributes greatly impacted SVM and Multi-layer Perceptron results, it did not hold for Random Forest – the difference between feature vectors resulting from dynamic and static attributes was not significant, which means that using only static attributes is enough for this particular model and scenario.

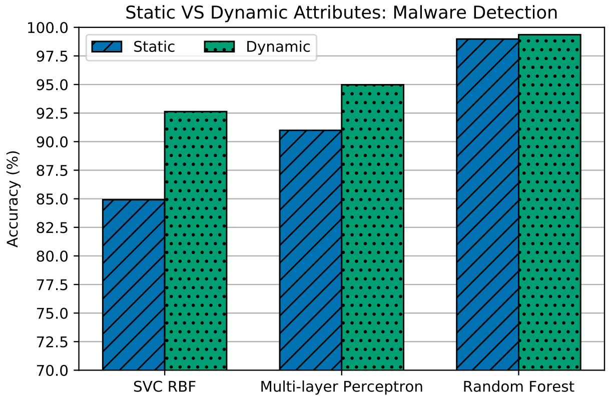


Fig. 7. **Static VS Dynamic attributes in malware detection.** According to the classifier used, accuracy is highly impacted by the type of attributes used (experiments derived from Galante et al. [57]).

Nguyen et al. compared four different attribute extraction methods for malware detection [112]: raw bytes [125], EMBER features (static PE file header attributes) [5], CAPA features [17], and dynamic analysis. While the raw bytes attributes take 0.002 seconds per file to be classified, the static attributes (EMBER) take 0.09 seconds, the CAPA features takes 45.75 seconds, and the dynamic analysis takes 526 seconds per file, i.e., using raw bytes on a file is over 26, 300 times faster than running dynamic analysis. Finally, according to their experiments, the raw bytes model achieved a higher malware detection accuracy than the dynamic analysis (≈ 90% vs ≈ 85%), while the ensemble using both of them achieves almost 93%, showing that, by combining different features, it is possible to improve classification performance considering that it would be much more expensive.

5 FEATURE EXTRACTION PITFALLS

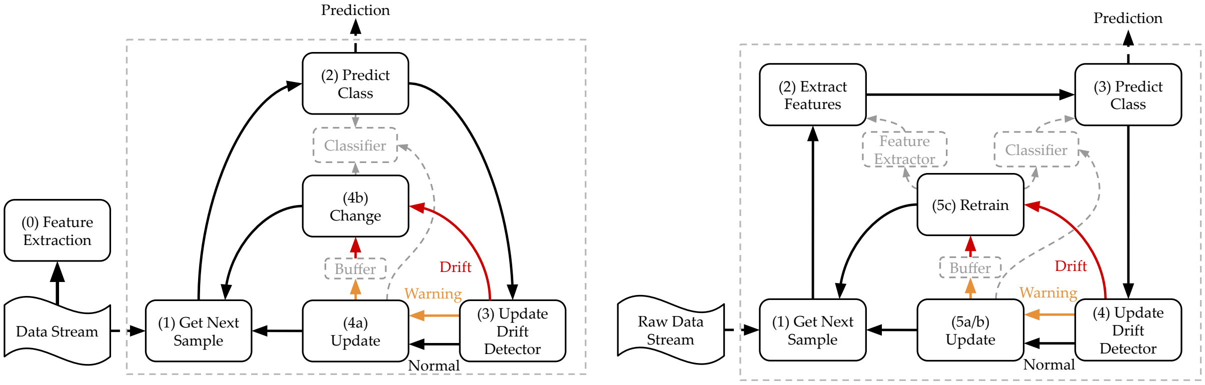
It is faster and simpler to use numerical or categorical attributes in any model, either by just encoding the categorical ones, or normalizing both of them. However, these attributes may not be directly used in the ML model, depending on their type after being extracted from raw data. For instance, a list of system calls or libraries used by software must pass through one more processing step before it can serve as input to the ML algorithm. This step is known as feature extraction, and its goal is to transform these attributes into something readable by the classifier and simplify the data while keeping the level of provided information, but reducing the number of resources used to describe them [68, 133]. Thus, there are several approaches to extracting features from attributes, and they usually rely on well-known techniques from ML literature, such as text classification, image recognition or classification, graph feature learning, and deep learning [15, 62]. Finally, according to the feature extraction selected, an ML solution may present several challenges and pitfalls.

5.1 Adapting to Changes

Many of the feature extractors mentioned in the literature need to be created based on a training dataset to, for instance, create a vocabulary (e.g., TF-IDF [81]) or to compute the weights of the neural network used (e.g., Word2Vec [106]), similar to an ML model. Thus, as time goes by, it is necessary to update the feature extractor used if a concept drift or evolution happens in the application domain, which is something very common in cybersecurity environments due to new emerging threats [37, 62, 83]. For instance, when using any vocabulary-based feature extractor for malware detection based on static features, such as the list of libraries used by a software, new libraries may be developed as time goes by (and they are not present in the vocabulary), which can result in a concept drift and make the representation created for all the new software outdated. In response to concept drift, the feature extractor may also need to be updated when it is detected and not only the classifier itself, requiring an efficient performance to update both of them.

To illustrate this challenge, Ceschin et al. created an experimental scenario using a proportionally reduced version of the AndroZoo dataset [3] with almost 350K samples (the same we used in Section 3.4), and the DREBIN dataset [12], composed of 129, 013 samples (123, 453 benign and 5, 560 malicious Android applications) [35]. The authors sorted the samples by their first seen date in VirusTotal [154] and trained two base Adaptive Random Forest classifiers [63] with the first year of data, both of them include the ADWIN drift detector [21], which usually has the best classification performance in the literature. When a concept drift is detected, the first classifier is updated using always the same features from the start, while the second one is entirely retrained from scratch, updating not only the classifier but also the feature extractor. In a traditional data stream learning problem that includes concept drift, the classifier is updated with new samples, which already had their features extracted previously using a feature extractor, when a change occurs (generally the ones that created the drift), as shown by steps in Fig. 8a [60]. Alternatively, the data stream pipeline proposed in this experiment as mitigation to this problem also considers the feature extractor under changes, retraining both feature extractor and classifier according to the following five steps in Fig. 8b [35].

Fig. 9 shows the results presented by the authors regarding the impact on classification performance caused when the feature extractor is updated after a concept drift is detected (AndroZoo dataset in Fig. 9a and DREBIN

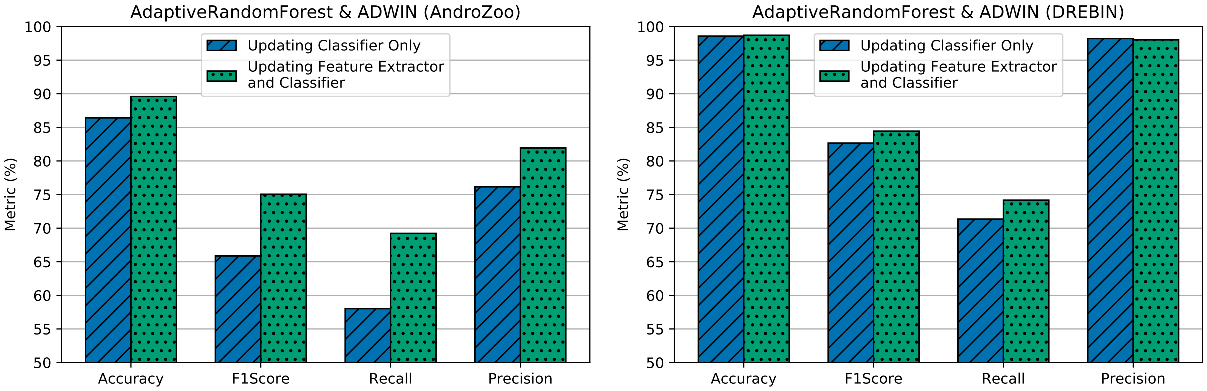


(a) **Traditional data stream pipeline.** Feature extraction is applied to the raw data stream data before using them in the learning cycle. Then, the classifier generates the prediction of the current sample and uses it to update the model, adding it to a buffer or changing the model according to the concept drift detector level.

(b) **New data stream pipeline with feature extractor.** Every time a new sample is obtained from the raw data stream, its features are extracted and presented to a classifier, generating a prediction, which is used by a drift detector that defines the next step: update the classifier or retrain both the classifier and feature extractor [35].

Fig. 8. **Data stream pipeline.** Comparing the traditional data stream pipeline with possible mitigation that adds the feature extractor in the process, generating updated features as time goes by [35].

dataset in Fig. 9b), improving by almost 10 percentage points of f1score when classifying AndroZoo dataset (from 65.86% to 75.05%). Thus, we emphasize the importance of including the feature extractor in the incremental learning process [35].



(a) **AndroZoo dataset.** Adaptive Random Forest with ADWIN when classifying AndroZoo [35].

(b) **DREBIN Dataset.** Adaptive Random Forest with ADWIN when classifying DREBIN [35].

Fig. 9. **Adapting features improves classification performance.** When considering the feature extractor in the pipeline, updating it when drift occurs is better than using a static representation (using a unique feature extractor based on the first training set). Experiments derived from Ceschin et al. [35].

5.2 Adversarial Features (Robustness)

There are multiple ways to choose features that represent the samples involved in a security problem, and the final accuracy is not the only metric that should be considered during the choosing step. Another important point to take into account is the robustness of the resulting model against adversarial machine learning: attackers may try to adapt their malicious samples to make their features look similar to benign samples while maintaining the same original behavior [34]. Thus, researchers must think like an attacker when choosing which attributes and features will be used to build an ML system for cybersecurity, given that some of them might be easily changed to trick the ML model.

To illustrate the impact of adversarial machine learning in cybersecurity solutions, let's consider the following malware detection models proposed in the academic literature: (i) `MalConv` [125] and `Non-Negative MalConv` [54], which are deep learning classifiers whose features are the raw bytes of an input file; and (ii) `LightGBM` [97], whose features consist of a matrix created using a hashing trick and histograms based on the inputted binary files characteristics (PE header information, file size, timestamp, imported libraries, strings, etc.). Both of these models can be easily bypassed using simple strategies that create totally functional adversarial malware. The formers (raw bytes-based models) can be tricked by simply appending goodware bytes or strings present in goodware applications at the end of the malware binary. The performed appendage does not affect the original binary execution and biases the detector toward the statistical identification of these goodware features. The latter (file characteristics-based model) can be bypassed by embedding the original binary into a generic file dropper, which extracts the embedded malicious content in runtime and executes it. The dropping technique, presented in Fig. 10, bypasses detection because the classifier would inspect the external dropper (only contains characteristics of benign applications, such as headers) instead of the embedded payload (the de facto malicious application). Previous work showed that the combination of the aforementioned counter-ML strategies can generate adversarial malware capable of bypassing both types of detection models, as well as affect the detection rate of antiviruses that rely on ML in their engines (as shown in Fig. 10, where a malware with 91.69% average confidence is transformed into a goodware with 93.28% of average confidence) [34].

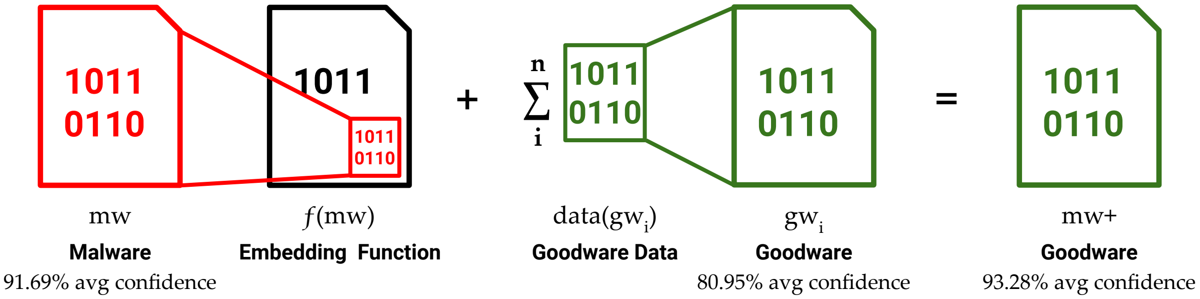


Fig. 10. **Adversarial malware generation.** It is possible to change classifiers' output by just using an embedding function to add malware payloads within a new file and adding goodware data to it, such as strings and bytes from a set of goodware [34].

Other types of attributes may be more resistant to these attacks. Although approaches based on Control Flow Graphs (CFGs) are not affected by the appending of extra, spurious bytes, they can be bypassed by malware samples that modify their internal control flow structures to resemble legitimate software pieces [31]. A strategy to handle this problem is to select the features that are more resistant to modifications on the original binary (e.g., the use of loop instructions whose code is enough to distinguish malicious programs, followed by the building of a feature space containing a set of labels for each of them, thus making adversarial feature vectors more difficult to attackers [99]).

In summary, we advocate for more studies about the robustness of features for cybersecurity, given that it is something crucial for the development of real applications. Thus, the aphorism "Garbage In, Garbage Out" used in many ML contexts is also valid for the quality of a solution, since it may become useless if subject to successful adversarial attacks. Proper ML models require high-quality training data and robust features in order to produce high-quality and robust classifiers [61]. Security-wise, it is important to understand that each ML model and feature extraction algorithm serve different threat models. Therefore, the resistance of a feature to a given type of attack should be evaluated considering the occurrence, prevalence, and impact of this type of attack in the specific application scenario (e.g., newly-released binaries being distributed with no validation codes, such as signatures and/or MAcs are more prone to be vulnerable to random data append attacks than the cases in which the original binary integrity is verified).

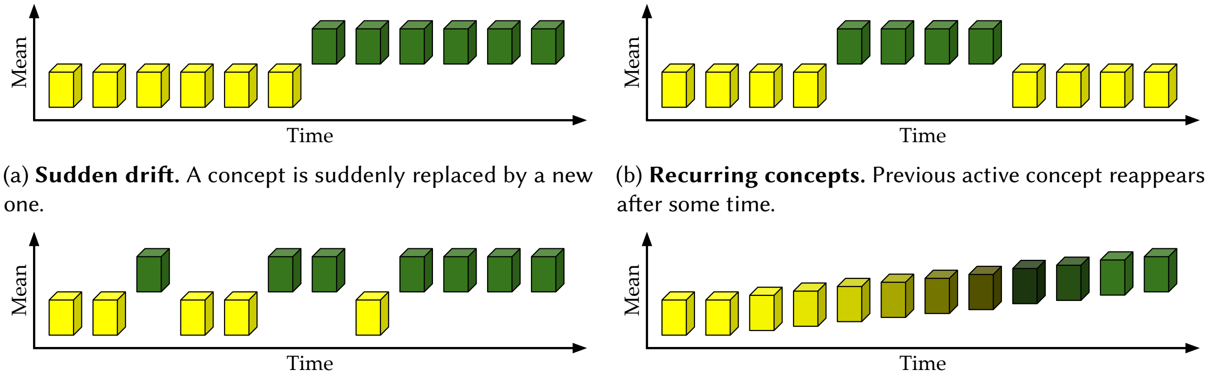
6 ML MODELLING ISSUES AND SOLUTIONS

An ML model is a mathematical model that generates predictions by finding relationships between patterns of the input features and labels in the data [137]. Thus, when using machine learning for any task, it is common to test different types of models and fine-tune them to find the one that best suits the application [23]. In cybersecurity, due to the dynamic scenarios presented in many tasks, streaming data models are strongly recommended to achieve a good performance, given that they belong to non-stationary distributions, new data are produced all the time, and they can be easily updated or adapted with them [20]. As a consequence, it is important to understand how to effectively use and, sometimes, implement an ML model in these scenarios, given that they may present many drawbacks that are not feasible in a real application.

6.1 Concept Drift and Evolution

Concept drift is the situation in which the relation between the input data and the target variable (the variable that needs to be learned, such as class or regression variable) changes over time [60]. It usually happens when there are changes in a hidden context, which makes it challenging since this problem spans different research fields [156]. In cybersecurity, these changes are caused by the arms race between attackers and defenders, once attackers are constantly changing their attack vectors when trying to bypass defenders' solutions [34]. In addition, concept evolution is another problem related to this challenge, which refers to the process of defining and refining concepts, resulting in new labels according to the underlying concepts [92]. Thus, both problems (drift and evolution) might be correlated in cybersecurity, given that new concepts may result in new labels, such as new types of attacks produced by attackers. As shown in Fig. 11, there are four types of concept drift according to the literature: (i) sudden drift, when a concept is suddenly replaced by a new one; (ii) recurring concepts, when a previous active concept reappears after some time; (iii) gradual drift, when the probability of finding the previous concept decreases and the new one increases until it is completely replaced; and (iv) incremental drift, when the difference between the old concept and the new one is very small and the difference is only noticed when looking at a longer period [95]. In security contexts, a sudden drift is when an attacker creates a totally new attack; gradual drift is when new types of attacks are created and replace previous ones; the recurring concept is when an old type of attack starts to appear again after a given time; and incremental drift is when the attackers make few modifications in their attacks in a way that their concepts change over a large period.

Despite being considered a challenge in cybersecurity [62], few works addressed both problems in the literature. For instance, Masud et al., to the best of our knowledge, were the first to treat malware detection as a data stream classification problem and mention concept drift. The authors proposed an ensemble of classifiers that are trained from consecutive chunks of data using v-fold partitioning of the data, reducing classification error compared



(a) **Sudden drift.** A concept is suddenly replaced by a new one.

(b) **Recurring concepts.** Previous active concept reappears after some time.

(c) **Gradual drift.** The probability of finding the previous concept decreases and the new one increases until it is completely replaced.

(d) **Incremental drift.** The difference between the old concept and the new one is very small and the difference is only noticed when looking at a longer period.

Fig. 11. **Drift types.** Different types of concept drift presented in the literature [95].

to other ensembles and making it more resistant to changes when classifying real botnet traffic data and real malicious executables [102]. Singh et al. proposed two measures to track concept drift in static features of malware families: relative temporal similarity and meta-features [143]. The former is based on the similarity score (cosine similarity or Jaccard index) between two time-ordered pairs of samples and can be used to infer the direction of the drift. The latter summarizes information from a large number of features, which is an easier task than monitoring each feature individually. Narayanan et al. presented an online ML-based framework named DroidOL to handle it and detect malware [110]. To do so, they use inter-procedural control-flow sub-graph features in an online passive-aggressive classifier, which adapts to the malware drift and evolution by updating the model more aggressively when the error is large and less aggressively when it is small. They also propose a variable feature-set regimen that includes new features to samples, including their values when present and ignoring them when absent (i.e., their values are zero). Deo et al. proposed the use of Venn-Abers predictors to measure the quality of binary classification tasks and identify antiquated models, which resulted in a framework capable of identifying when they tend to become obsolete [46]. Jordaney et al. presented Transcend, a framework to identify concept drift in classification models which compares the samples used to train the models with those seen during deployment [82]. To do it, their framework uses a conformal evaluator to compute algorithm credibility and confidence, capturing the quality of the produced results that may help to detect concept drift. Anderson et al. showed that, by using reinforcement learning to generate adversarial samples, it is possible to retrain a model and make these attacks less effective, also protecting it against possible concept drift, given that it hardens a machine learning model against worst-case inputs [4]. Xu et al. proposed DroidEvolver, an Android malware detection system that can be automatically updated without any human involvement, requiring neither retraining nor true labels to update itself [157]. The authors use online learning techniques with evolving feature sets and pseudo labels, keeping a pool of different detection models and calculating a juvenilization indicator, which determines when to update its feature set and each detection model. Finally, Ceschin et al. compared a set of Windows malware detection classifiers that use batch machine-learning models with ones that take into account the change of concept using data streams, emphasizing the need to update the decision model immediately after a concept drift is detected by a concept drift detector, which are state-of-the-art techniques used in the data stream learning literature [37]. The authors also show that the malware concept drift is strictly related to their concept evolution, i.e., due to the appearance of new malware families.

In contrast, data stream learning literature already proposed some approaches to deal with concept drift and evolution, called concept drift detectors, that, to the best of our knowledge, were not totally explored by cybersecurity researchers. There are supervised drift detectors that take into account the ground-truth label to make a decision and unsupervised ones that do not. DDM (Drift Detection Method [59]), EDDM (Early Drift Detection Method [14]) and ADWIN (ADaptive WINdowing [21]) are examples of supervised approaches. Both DDM and EDDM are online supervised methods based on sequential error (prequential) monitoring, where each incoming example is processed separately estimating the prequential error rate. This way, they assume that the increase in consecutive error rate suggests the occurrence of concept drifts. DDM directly uses the error rate, while EDDM uses the distance error rate, which measures the number of examples between two classification errors [14]. These errors trigger two levels: warning and drift. The warning level suggests that the concept starts to drift, updating an alternative classifier using the examples which rely on this level. The drift level suggests that the concept drift occurred, and the alternative classifier built during the warning level replaces the current classifier. ADWIN keeps statistics from sliding windows of variable size, which are used to compute the average of the change observed by cutting these windows at different points. If the difference between two windows is greater than a predefined threshold, it considers that a concept drift happened, and the data from the first window is discarded [21]. Different from the other two methods, ADWIN has no warning level. Once a change occurs, the data that is out of the window is discarded and the remaining ones are used to retrain the classifier. Unsupervised drift detectors such as the ones proposed by Žliobaitė et al. may be useful when delays are expected given that they do not rely on the real label of the samples, which need to be known by supervised methods, and most of the time in cybersecurity it does not happen in practice [161]. These unsupervised strategies consist in comparing different detection windows of fixed length using statistical tests over the data themselves, on the classifier output labels or its estimations (that may contain errors) to detect if both come from the same source. In addition, active learning may complement these unsupervised methods by requiring the labels of only a subset of the unlabeled samples, which could improve drift detection and overall classification performance.

Some authors also created different classification models and strategies that deal with both concept drift and concept evolution. Shao et al. proposed SyncStream, a classification model for evolving data streams that use prototype-based data representation, P-Tree data structure, and just a small set of both short and long-term samples based on error-drive representativeness learning (instead of using base classifiers or windows of data) [139]. ZareMoodi et al. created a new supervised chunk-based method for novel class detection using ensemble learners, local patterns, and connected components of neighborhood graphs [158]. The same authors also proposed a new way to detect evolving concepts by optimizing an objective function using a fuzzy agglomerative clustering method [159]. Hosseini et al. created SPASC (Semi-supervised Pool and Accuracy-based Stream Classification), an ensemble of classifiers where each classifier holds a specific concept, and new samples are used to add new classifiers to the ensemble or to update the existing ones according to their similarity to the concepts [74]. Dehghan et al. proposed a method based on the ensemble to detect concept drift by monitoring the distribution of its error, training a new classifier on the new concept to keep the model updated [45]. Ahmadi et al. created GraphPool, a classification framework that deals with recurrent concepts by looking at the correlation among features, using a statistical multivariate likelihood test, and maintaining the transition among concepts via a first-order Markov chain [2]. Gomes et al. presented the Adaptive Random Forest (ARF) algorithm, an adaptation of the classical random forest algorithm with dynamic update methods to deal with evolving data streams. The ARF also contains an adaptive strategy that uses a concept drift detector in each tree to track possible changes and to train new trees in the background [63]. Finally, Siahroudi et al. proposed a method using multiple kernel learning to detect novel classes in non-stationary data streams [142]. The authors do it by classifying each new instance by computing their distance to the previously known classes in the feature space and updating the model based on their true labels.

We advocate for more collaboration between data stream learning and cybersecurity, given that the majority of cybersecurity works presented in this section do not use data stream approaches (including concept drift detectors), they both have a lot of practice problems in common and may benefit each other. For instance, data stream learning could benefit from real cybersecurity datasets that could be used to build real-world ML security solutions, resulting in higher quality research that may also be useful in other ML research fields. Finally, developing new drift detection algorithms is important to test their effectiveness in different cybersecurity scenarios and ML models.

6.2 Adversarial Attacks

In most cybersecurity solutions that use Machine Learning, models are prone to suffer adversarial attacks, where attackers modify their malicious vectors to somehow make them not being detected [34]. These techniques were proven effective in both malware and intrusion scenarios [101], for instance. We already mentioned this problem related to feature robustness in Section 5.2, but ML models are also subject to adversaries. These adversarial attacks may have several consequences such as allowing the execution of malicious software, poisoning an ML model or drift detector if they use new unknown samples to update their definitions (without a ground truth from other sources), and producing, as a consequence, concept drift and evolution. Thus, when developing cybersecurity solutions using ML, both features and models must be robust against adversaries.

Aside from using adversarial features, attackers may also directly attack ML models. There are two types of attacks: white-box attacks, where the adversary has full access to the model, and black-box attacks, where the adversary has access only to the output produced by the model, without directly accessing it [13]. A good example of white-box attacks is gradient-based adversarial attacks, which consist in using the weights of a neural network to obtain perturbation vectors that, combined with an original instance, can generate an adversarial one that may be classified by the model as being from another class [66]. Many strategies use neural network weights to produce these perturbations [13], which not only affects neural networks but a wide variety of models [66]. Other simpler white-box attacks such as analyzing the model, for instance, the nodes of a decision tree or the support vectors used by an SVM, could be used to manually craft adversarial vectors by simply changing the original characteristics of a given sample in a way that it can affect its output label. In contrast, black-box attacks tend to be more challenging and real for adversaries, given that they usually do not have access to implementations of cybersecurity solutions or ML models, i.e., they have no knowledge about which features and classifiers a given solution is using and usually only known which is the raw input and the output. Thus, black-box attacks rely on simply creating random perturbations and testing them in the input data [71], changing characteristics from samples looking at instances from all classes [13], or trying to mimic the original model by creating a local model trained with samples submitted to the original one, using the labels returned by it, and then analyzing or using this new model to create an adversarial sample [118].

In response to adversarial attacks, defenders may try different strategies to overcome them, searching for more robust models that make this task harder for adversaries. One response to these attacks is Generative Adversarial Networks (GANs), which are 2-part, coupled deep learning systems in which one part is trained to classify the inputs generated by the other. The two parts simultaneously try to maximize their performance, improving the generation of adversaries, that are used to defeat the classifier, and then used to improve their detection by training the classifier with them [65, 76]. Another valid strategy is to create an algorithm that, given a malign sample, automatically generates adversarial samples, similar to data augmentation or oversampling techniques that insert benign characteristics into it, which are then used to train or update a model. This way, the model will learn not only the normal concept of a sample but also the concept of its adversaries' versions, which will make it more resistant to attacks [70, 119]. Instead of using the hard class labels, Apruzzese et al. propose using the probability labels to make random forest-based models more resilient to adversarial perturbations, achieving comparable or even superior results even in the absence of attacks [7].

Some approaches also tried to fix limitations of already developed models, such as `MalConv` [125], an end-to-end deep learning model, which takes as input raw bytes of a file to determine its maliciousness. `Non-Negative MalConv` proposes an improvement to `MalConv`, with an identical structure, but having only non-negative weights, which forces the model to look only for malicious evidence rather than looking for both malicious and benign ones, being less prone to adversaries that try to copy benign behavior [54]. Despite that, even `Non-Negative MalConv` has weaknesses that can be explored by attackers [34], which makes this topic an open problem to be solved by future research. We advocate for more work and competitions, such as the Machine Learning Security Evasion Competition (MLSEC) [16], that encourage the implementation of new defense solutions that minimize the effects of adversarial attacks.

6.3 Class imbalance

Class imbalance is a problem already mentioned in this work, but on the dataset side (Section 3.3). In this section, we are going to discuss the effects of class imbalance in the ML model and present some possible mitigation techniques that rely on improving the learning process (cost-sensitive learning), using ensemble learning (algorithms that combine the results of a set of classifiers to make a decision) or anomaly detection (or one-class) models [64, 86]. This way, when using cost-sensitive learning approaches, the generalization made by most algorithms, which makes minority classes ignored, is adapted to give each class the same importance, reducing the negative impact caused by class imbalance. Usually, cost-sensitive learning approaches increase the cost of incorrect predictions of minority classes, biasing the model in their favor and resulting in better overall classification results [86]. Such techniques are not easy to implement in comparison to sampling methods presented in Section 3.3 but tend to be much faster given that they just adapt the learning process, without generating any artificial data [64].

In addition, ensemble learning methods that rely on bagging [29] or boosting techniques (such as AdaBoost [55]) present good results with imbalanced data [58], which is one of the reasons that random forest performs well in many cybersecurity tasks with class imbalance problems, such as malware detection [37]. Bagging consists in training the classifiers from an ensemble with different subsets of the training dataset (with replacement), introducing diversity to the ensemble, and improving overall classification performance [29, 58]. The AdaBoost technique consists in training each classifier from the ensemble with the whole training dataset in iterations. After each iteration, the algorithm gives more importance to difficult samples, trying to correctly classify the samples that were incorrectly classified by giving them different weights, very similar to what cost-sensitive learning does, but without using a cost to update the weights [55, 58].

Even though all the methods presented so far are valid strategies to handle imbalanced datasets, sometimes the distribution of classes is too different that it is not viable to use one of them, given that the majority of the data will be discarded (undersampling), poor data will be generated (oversampling), or the model will not be able to learn the concept of the minority class [64]. In these cases, anomaly detection algorithms are strongly recommended, given that they are trained over the majority class only and the remaining ones (minority class) are considered anomalous instances [64, 131]. Two great examples of anomaly detection models are isolation forest [98] and one-class SVM [135]. Both of them try to fit the regions where the training data is most concentrated, creating a decision boundary that defines what is normal and what is an anomaly.

Finally, when building an ML solution that has imbalanced data, as well as testing several classifiers and feature extractors, it is also important to consider the approaches presented here for both the dataset and model sides. Also, it is possible to combine more than one method, for instance, generating a set of artificial data and using cost-sensitive learning strategies, which could increase classification performance in some cases. We strongly recommend that cybersecurity researchers include some of these strategies in their work, given that it is difficult to find solutions that actually consider class imbalance.

6.4 Transfer Learning

Transfer learning is the process of learning a given task by transferring knowledge from a related task that has already been learned. It has shown to be very effective in many ML applications [150], such as image classification [79, 126] and natural language processing problems [75, 124]. Recently, Microsoft and Intel researchers proposed the use of transfer learning from computer vision to static malware detection [40], representing binaries as grayscale images and using inception-v1 [147] as the base model to transfer knowledge [41]. The results presented by the authors show a recall of 87.05%, with only a 0.1% of false positive rate, indicating that transfer learning may help to improve malware classification without the need of searching for optimal hyperparameters and architectures, reducing the training time and the use of resources.

In addition, if the network used as the base model is robust, they probably contain robust feature extractors. Consequently, by using these feature extractors, the new model produced inherits their robustness, producing new solutions that are also robust to adversarial attacks, achieving high classification performances, without much data, and with no need to use a lot of resources as some adversarial training approaches [138]. At the same time that transfer learning might be an advantage, it may also be a problem according to the base model used because usually, these base models are publicly available, which means that any potential attackers might have access to them and produce an adversarial vector that might affect both models: the base and the new one [127]. Thus, it is important to consider the robustness of the base model when using it to transfer learning to produce a solution without security weaknesses. Finally, despite presenting promising results, the model proposed to detect malware by using transfer learning cited at the beginning of this subsection [41] may be affected by adversarial attacks, given that its base model is affected by them as already shown in the literature [32, 66].

6.5 Implementation

Building a good Machine Learning model is not the last challenge to deploying ML approaches in practice. The implementation of these approaches might also be challenging [113]. The existing frameworks, such as scikit-learn [120] and Weka [72], usually rely on batch learning algorithms, which may not be useful in dynamic scenarios where new data are available all the time (as a stream), requiring the model to be updated frequently with them [64]. In these cases, ML implementations for streaming data, such as Scikit-Multiflow [108], Massive Online Analysis (MOA) [22], River [107], and Spark [103, 144] are highly recommended, once they provide ML algorithms that could be easily used in real cybersecurity applications. Also, adversarial machine learning frameworks, such as CleverHans [116] and SecML [104], are important to test and evaluate the security of ML solutions proposed. Thus, contributing to streaming data and adversarial machine learning projects is as important as contributing to well-known ML libraries, and we advocate for that to make all research closer to real-world applications. Note that we are not just talking specifically about contributing with new models, but also prepossessing and evaluating algorithms that may be designed only in batch learning, and could also be a good contribution to streaming learning libraries. We believe that more contributions to these projects would benefit both industry and academia with higher quality solutions and research, given the high number of research works using only batch learning algorithms nowadays, even in cybersecurity problems.

In addition, multi-language codebases may be a serious challenge when implementing a solution, once different components may be written in different languages, not being completely compatible, becoming incompatible with new releases, or being too slow according to the code implementation and language used [134]. Thus, it is common to see ML implementations being optimized by C and C++ under the hood, given that they are much faster and more efficient than Python and Java, for instance. Despite such optimizations being needed to make many solutions feasible in real-world solutions, they are not always performed given that (i) researchers create their solutions as prototypes that only simulate the real world, not requiring optimizations, and (ii) optimizations require knowledge about code optimization techniques that are very specific or may be limited to a given type of hardware, such as GPUs [134]. Also, implementing data stream algorithms is a hard task, given that we need to execute the whole pipeline continuously: if any component of this pipeline fails, the whole system may fail [64].

Another challenge is to ensure a good performance for the proposed algorithms and models [42]. A good performance is essential to deploy ML in the security context because most of the detection solutions operate in runtime to detect attacks as early as possible and slow models will result in a significant slowdown to the whole system operation. To overcome the performance barriers of software implementations, many security solutions opt to outsource the processing of ML algorithms to third-party components. A frequent approach in the security context is to propose hardware devices to perform critical tasks, among which is the processing of ML algorithms [28]. Alternatively, security solutions might also outsource scanning procedures to the cloud. Many research works proposed cloud-based AVs [47, 80], which have the potential to include ML-based scans among their detection capabilities and streamline such checks to the market. We understand that these scenarios should be considered for the proposal of new ML-based detection solutions.

7 EVALUATION

Knowing how to correctly evaluate an ML system is essential to building a security solution, given that some evaluations may result in wrong conclusions that may backfire in security contexts, even when using traditional ML evaluation best practices [33, 37, 121]. For instance, consider that a malicious threat detection model is evaluated using ten samples: eight of them are benign and two are malign. This model has an accuracy of 80%. Is 80% a good accuracy? Assuming that the model classifies correctly only the eight benign samples, it is not capable of identifying any malign sample, and yet it has an accuracy of 80%, giving a false impression that the model works significantly well. Thus, it is important to take into account the metric used to produce high-quality systems that solve the problem proposed by a certain threat model.

7.1 Metrics

To correctly evaluate a solution the right metrics need to be selected to provide significant insights that can present different perspectives of the problem according to its real needs, reflecting the real world [68]. One of the most used metrics by ML solutions is accuracy, which consists in measuring the percentage of samples correctly classified by the model divided by the total number of samples seen by it (usually from the testing set) [52]. The main problem with this metric is that it may provide wrong conclusions according to the distribution of the datasets used, as already shown by the malicious threat detection model example. Thus, if the dataset is imbalanced, accuracy is not recommended since it will give much more importance to the majority class, presenting, for instance, high values even if a minority class is completely ignored by the classifier [52].

An interesting way to evaluate model performance is by checking the confusion matrix, a matrix where each row represents the real label and each column represents a predicted class (or vice versa) [68]. By using this matrix, it is possible to check a lot of information about the model, for instance, which class is more difficult to classify or which ones are being confused the most. In addition, it is also possible to calculate false positives and false negatives, which leads us to recall, precision, and, consequently, f1score. Recall measures the percentage of positive examples that are correctly classified, precision measures the percentage of examples classified as positive that are positive, and f1score is the harmonic mean of both of them [44]. With these metrics, it is possible to calibrate the model according to the task being performed [37]. For a malware detector, for instance, it may be better to not detect malware than to block benign software (high precision), given that it would affect directly the user experience. Imagine that a user is using his computer and, when opening Microsoft Word, the classifier believes that it is malware and blocks its execution. Even detecting the majority of the malware, the classifier may turn the computer useless, as it would block a great part of the benign applications. In contrast, in more sensitive systems, one might want the opposite (high recall), blocking all the malign actions, even with some benign being ”sacrificed“.

Recently, some authors introduced metrics to evaluate the quality of the models produced. Jordaney et al. proposed Conformal Evaluator (CE), an evaluation framework that computes two metrics (algorithm confidence and credibility) to measure the quality of the produced ML results, evaluating the robustness of the predictions made by the algorithm and their qualities [82]. Pendlebury et al. introduced another evaluation framework called Tesseract, which compares classifiers in a realistic setting by introducing a new metric, Area Under Time (AUT). This metric captures the impact of time decay on a classifier, which is not evaluated in many works, confirming that some of them are biased. Thus, we support the development of these kinds of work to better evaluate new ML solutions considering a real-world scenario that allows the implementation to be used in practice.

7.2 Comparing Apples to Orange

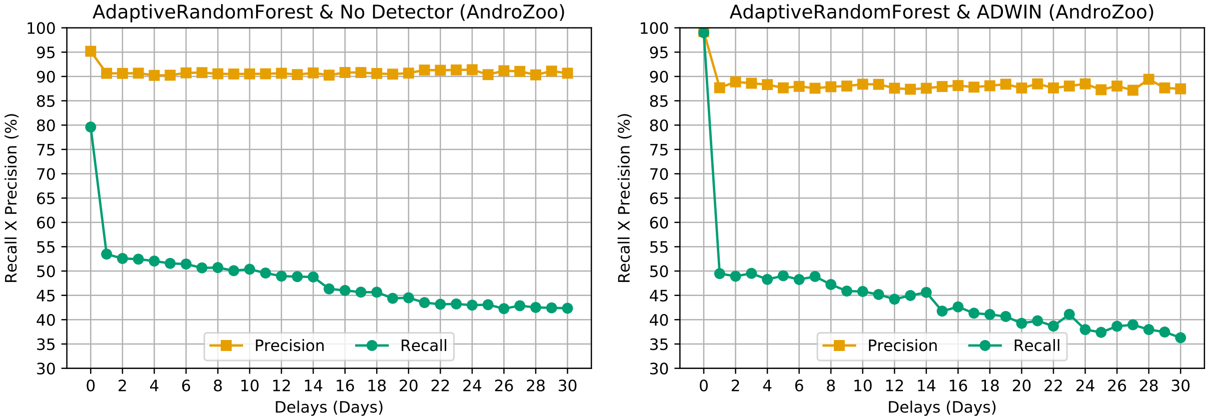
It is not unusual for researchers to compare their evaluation results with other prior literature solutions that face the same problems (e.g., comparing the accuracy of two detection models). Such comparison, however, should be carefully performed to avoid misleading conclusions.

The lack of standard publicly available repositories is responsible for lots of work using their own dataset when building a solution. These new solutions have their results usually compared with other works reported in the literature. Whereas comparing approaches seems to be straightforward, authors should care to perform fair evaluations, such as comparing studies leveraging the same datasets, avoiding presenting results deemed to outperform literature results but which do not achieve such performance in actual scenarios.

As an analogy, consider image classification problems whose objective is to identify objects represented in images (for instance, buildings, animals, locations, etc). These challenges often provide multiple datasets that are used as a baseline by many solutions. For instance, the CIFAR challenge [90] is composed of two datasets: CIFAR-100, which has one hundred classes of images, and CIFAR-10, which is a filtered version of CIFAR-100, containing just ten classes. Imagine two research work proposing distinct engineering solutions for image classification, one of them leveraging CIFAR-10 and the other leveraging CIFAR-100. Although one of the approaches presents a higher accuracy than the other, is it fair to say that this one is better than the other? No, because the task involved in classifying distinct classes is also distinct. The same reasoning is valid for any cybersecurity research involving ML. Thus, authors should care to not perform comparisons involving distinct classes of applications, such as comparing, for instance, approaches involving Dynamic System Call Dependency Graphs, a computationally costly approach, with static feature extraction approaches. This is misleading because each type of work presents different natures and challenges. Finally, it is strongly recommended that researchers share their source codes with the community to make their work compatible with any other dataset (which in the majority of the works are not shared), allowing future researchers to compare different approaches in the same scenario.

7.3 Delayed Labels Evaluation

One particularity of security data is that they usually do not have ground-truth labels available right after new data are collected, as already shown in Section 3.2. Due to that, there is a gap between the data collection and their labeling process, which is not considered in many cybersecurity types of research that uses ML, i.e., in many proposed solutions in the literature, the labels are available at the same time as the data, even in online learning solutions. Some of them ignore the ground-truth label and use the same labels that the ML classifier



(a) **AndroZoo dataset without drift detector.** Results for the adaptive random forest with no drift detector using AndroZoo.

(b) **AndroZoo dataset with drift detector.** Results for the adaptive random forest with ADWIN drift detector using AndroZoo.

Fig. 12. **Delayed Labels Evaluations.** Models with drift detection may not perform as expected in real-world solutions, despite having the best performance in scenarios where delays are not considered.

predicts [157], which may make the model subject to poisoning attacks and, consequently, decrease its detection rate as time goes by [148].

Considering malware detection, the majority of the works use only a single snapshot of scanning results from platforms like VirusTotal, without considering a given delay before actually using the labels, which may vary from ten days to more than two years [160]. A recent study from Botacin et al. analyzed the labels provided by VirusTotal for 30 consecutive days from two distinct representative malware datasets (from Windows, Linux, and Android) and showed that solutions took about 19 days to detect the majority of the threats [26]. To study the impact of these delayed labels in malware detectors using ML, we simulated a scenario using the AndroZoo dataset [3] and online ML techniques with and without drift detectors by providing the labels of each sample *N* days after they are available to further update the decision model (Adaptive Random Forest [63] trained using TF-IDF feature extractor). The results of this experiment are shown in Fig. 12a and Fig. 12b. We can note that when not considering a delay, using a drift detector improves the detection rate a lot. However, after one day of delay, this is not true anymore: the model that does not consider concept drift performs better overall, despite both being very affected by this problem, dropping to half of the original precision in the case of Adaptive Random Forest with ADWIN. These results indicate that: (i) in scenarios where delayed labels exist, ML models do not perform the way they are evaluated without these conditions, and (ii) models that make use of drift detectors, such as ADWIN, present lower detection rates in comparison to those that do not use them, since the concept being learned by the model is outdated when a drift is detected, increasing false negatives.

Finally, we advocate for drift detection strategies that consider the delay of labels and mitigation techniques to overcome this problem, such as active learning or approaches to provide real labels with less delay, to produce better solutions [89].

7.4 Online vs Offline Experiments

We previously advocated for real-world considerations when developing an ML model (Section 6). We also advocate for real-world considerations when evaluating the developed solutions. Each evaluation should consider the scenario in which it will be applied. A major implication of this requirement is that online and offline solutions must be evaluated using distinct criteria. Offline solutions are often leveraged by security companies in their internal analysis (e.g., an AV company investigating whether a given payload is malicious or not to develop a signature for it that will be distributed to the customers in the future). Online solutions, in turn, are leveraged by the endpoints installed in the user machines (e.g., a real-time AV monitoring a process behavior, a packet filter inspecting the network traffic, and so on). Due to their distinct nature, offline solutions are much more flexible than online solutions. Whereas offline solutions can rely on complex models that are run on large clusters, online solutions must be fast enough to be processed in real time without adding a significant performance overhead to the system. Moreover, offline solutions can collect huge amounts of data during a sandboxed execution and thus input them into models relying on a large moving window for classification. Online solutions, in turn, operate in memory-limited environments and aim to detect the threat as soon as possible. Thus, they cannot rely on large moving windows. These differences must be considered when evaluating the models to avoid common pitfalls, such as applying the same criteria for both applications. Due to their distinct nature, online solutions are expected to present more false negatives than offline solutions, as they have to make fast decisions about the threat. On the other hand, offline solutions will detect more samples, as they have more data to decide, but the detection is likely to happen later than in an online detector, only after multiple windows (e.g., when malware already infected the system and/or when a remote payload already exploited a vulnerable application). For a more complete security treatment, modern security solutions should consider hybrid models that employ multiple classifiers, each one with a distinct window size [146].

8 DISCUSSION: UNDERSTANDING ML

Once we have presented all the steps required to apply ML to security problems, we now discuss how this application should be understood, its limitations, and existing open gaps.

8.1 A ML model is a type of signature

A frequent claim of most products and research work leveraging ML as part of their operation, mainly antivirus and intrusion detection systems, is that this use makes their approaches more flexible and precise than the so-far applied signature schemes [122]. This is somehow a pitfall, as, in the last instance, an ML model is just a weighted signature of boolean values. One can even convert an ML model to typical YARA signatures [132], as deployed by many security solutions. Although the weights can be adjusted to add some flexibility to an ML model, the model itself cannot be automatically extended beyond the initially considered boolean values without additional processing (re-training or feature re-extraction), which is an already existing drawback for signature schemes. In this sense, the adversarial attacks against the ML models can see as analogous to the polymorphic attacks against typical signature schemes [149]: whereas a typical polymorphic threat attempt to directly change its features to pass the checks (weight=1), a modern attack against ML models indirectly attempts to do so by presenting a relative feature frequency higher or lower than the considered in the ML model.

8.2 There is no such thing as a 0-day detector

It is usual for many ML-based detector proposals to state that their approach is resistant to 0-day attacks (i.e., attacks leveraging so-far unknown threats and/or payloads) because they rely on an ML model [1, 94]. This is also somehow a pitfall, and we credit this as the poor understanding that ML models are a type of signature, as previously presented. In fact, the ability of a model to detect a new threat depends on the definition of what is new. If a new sample is understood as any sample that an attacker can create, certainly ML models will be able to detect many 0-days, as many of these new payloads will be variations of previously known threats, a scenario for which weighted signatures generalize well. However, if we consider as new something unknown to the model in all senses (e.g., a payload having a feature that is not present in the model), no ML model will be able to detect it as malicious, thus nature of the presented problem of concept drift and evolution. Therefore, ML models should not be understood as simply more resistant detectors than typical signature schemes, but as a type of signature that can be more generalized.

8.3 Security & Explainable Learning

Most research works put a big focus on achieving good detection rates, but not all of them put the same effort into discussing and explaining the results [12], which would be essential to allow security incident response teams (CSIRTs) to fill the identified security breaches. The scenario is even more complicated when deep learning approaches are considered, as in most cases there is no clear explanation for the model operation [56]. A model to detect cybersecurity threats can yield valuable insights besides its predictions since based on the explanation provided by the model, the security around the monitored object can be improved in future versions. As an example, a model able to explain that some threats were identified due to the exploitation of a given vulnerability might trigger patching procedures to fix the associated applications. Explainability has different levels of relevance according to the domain; we argue that for several cybersecurity tasks, they are essential to make it possible to apply countermeasures to security threats.

8.4 The arms race will always exist

The arms race between attack and defense solutions is a cycle that will always exist, requiring that both sides keep investing in new approaches to overcome their adversaries. Thus, new approaches from one the sides result in a reaction from another one, as shown in Fig. 13, where defense solutions follow the steps mentioned in this work to build ML defense solutions and attackers follow the steps required to create an attack: they first find the weakness of the defense solutions, use this weakness to develop an exploit, which is then delivered to be executed, producing new data for ML models to be trained, restarting the cycle [77]. Finally, we advocate that defense solutions try to reduce the gap present in this cycle (the development of new attacks and the generation of solutions for them) by following the recommendations of this work and other works in the literature that make use of robustness verification to prevent attacks against adversaries [96].

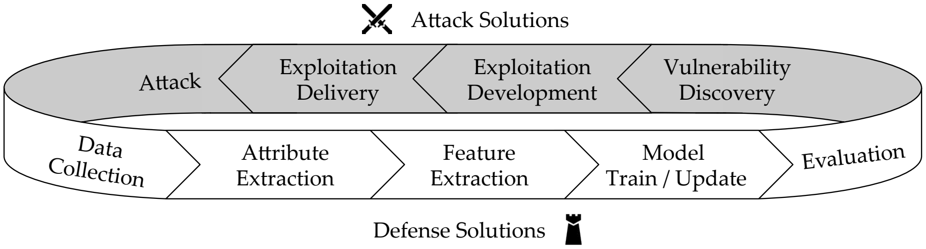


Fig. 13. **Attack VS Defense Solutions.** The arms race created by both generates a never-ending cycle.

To help the development of future ML solutions for cybersecurity, we created a checklist[2] that is an adaptation based on the challenges, pitfalls, and problems reported in this work. Thus, anyone developing or reviewing a solution can fill the questions on this checklist and get feedback reporting what could be improved or corrected according to our findings.

[2] The draft version of the checklist is available at https://secret.inf.ufpr.br/machine-learning-in-security-checklist/.

8.5 The future of ML for cybersecurity

ML for cybersecurity has become so popular that is hard to imagine a future without it. However, this does not mean that there is no room for improvement. As an immediate task, researchers will put their effort to mitigate adversarial attacks, which will enable even more applications to migrate ML solutions. A massive migration will require not only robust security guarantees but also privacy ones, which is likely to be achieved via merging ML and cryptography. The field is experiencing the rise of ML classifiers based on homomorphic cryptography [123], which allows data to be classified without decryption. We believe that this type of approach has the potential to upstream many ML-powered solutions, such as cloud-based AVs, which will be able to scan files in the cloud while preserving the privacy of the file's owners.

9 CONCLUSION

In this paper, we introduced a set of problems and challenges that have been observed (too often) in ML techniques applied to diverse cybersecurity solutions. We presented practical examples of cybersecurity scenarios in which ML might either be incorrectly applied or contain blind spots (important details that were not considered, discussed, or observed). When possible, we showed techniques to address those common issues. To make a step toward that, in this paper, we summarized and provided insights on the following main points: data collection issues (data leakage, data labeling, class imbalance, and dataset size definition); modeling (different attribute and feature extraction methods, adaptation to changes, i.e. concept drift/evolution, adversarial features and model attacks, one-class models, cost-sensitive, ensemble learning, transfer learning, and implementation challenges); and evaluation concerns (adequate use of metrics, thorough comparison of previous/existing solutions, delayed labels, and online vs. offline experiments).

Finally, our main recommendations to improve ML in security are the following:

- **Stop looking only at metrics, and start looking at effects:** many of the challenges presented in this work remain as open research problems, which would benefit both academia and industry if properly solved. In addition, their mitigation would foster the use of ML for cybersecurity problems, and improve the cybersecurity field in the same way ML advanced other research fields. Community players (security and machine learning) have to take into consideration the plenty of peculiarities associated with cybersecurity data and its sources.
- **Commit yourself to the real world:** we advocate that future research works always keep the motto "machine learning that matters" [155] in mind when developing new solutions. If your work is losing connection to problems of the real world, science, and society, we have a problem!
- **Check your work:** our recommendation is to carefully observe all the items, and to consider each of them during the design and implementation of ML models for cybersecurity. To encourage that, we presented a checklist (draft version available at https://secret.inf.ufpr.br/machine-learning-in-security-checklist/; validation tests ongoing) to serve as a reminder and prevent researchers and practitioners from committing these common mistakes or at least being aware of their existence.

REFERENCES

[1] F. Abri, S. Siami-Namini, M. A. Khanghah, F. M. Soltani, and A. S. Namin. 2019. Can Machine/Deep Learning Classifiers Detect Zero-Day Malware with High Accuracy?. In *2019 IEEE International Conference on Big Data (Big Data)*. 3252–3259.

[2] Zahra Ahmadi and Stefan Kramer. 2017. Modeling recurring concepts in data streams: a graph-based framework. *Knowledge and Information Systems* 55 (2017), 15–44.

[3] Kevin Allix, Tegawendé F. Bissyandé, Jacques Klein, and Yves Le Traon. 2016. AndroZoo: Collecting Millions of Android Apps for the Research Community. In *Proceedings of the 13th International Conference on Mining Software Repositories* (Austin, Texas) *(MSR '16)*. ACM, New York, NY, USA, 468–471. https://doi.org/10.1145/2901739.2903508

[4] Hyrum S. Anderson, Anant Kharkar, Bobby Filar, David Evans, and Phil Roth. 2018. Learning to Evade Static PE Machine Learning Malware Models via Reinforcement Learning. arXiv:1801.08917 [cs.CR]

[5] H. S. Anderson and P. Roth. 2018. EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models. *ArXiv e-prints* (April 2018). arXiv:1804.04637 [cs.CR]

[6] Giovanni Apruzzese, Hyrum S. Anderson, Savino Dambra, David Freeman, Fabio Pierazzi, and Kevin A. Roundy. 2023. "Real Attackers Don't Compute Gradients": Bridging the Gap between Adversarial ML Research and Practice. In *Proceedings of the 1st IEEE Conference on Secure and Trustworthy Machine Learning (SaTML)*.

[7] Giovanni Apruzzese, Mauro Andreolini, Michele Colajanni, and Mirco Marchetti. 2020. Hardening Random Forest Cyber Detectors Against Adversarial Attacks. *IEEE Transactions on Emerging Topics in Computational Intelligence* 4 (08 2020), 427–439. https://doi.org/10.1109/TETCI.2019.2961157

[8] Giovanni Apruzzese, Pavel Laskov, Edgardo Montes de Oca, Wissam Mallouli, Luis Brdalo Rapa, Athanasios Vasileios Grammatopoulos, and Fabio Di Franco. 2023. The Role of Machine Learning in Cybersecurity. *Digital Threats: Research and Practice* 4, 1 (mar 2023), 1–38. https://doi.org/10.1145/3545574

[9] Giovanni Apruzzese, Pavel Laskov, and Aliya Tastemirova. 2022. SoK: The Impact of Unlabelled Data in Cyberthreat Detection. In *2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P)*. IEEE. https://doi.org/10.1109/eurosp53844.2022.00010

[10] Ignacio Arnaldo and Kalyan Veeramachaneni. 2019. The Holy Grail of "Systems for Machine Learning": Teaming Humans and Machine Learning for Detecting Cyber Threats. *SIGKDD Explor. Newsl.* 21, 2 (Nov. 2019), 39–47. https://doi.org/10.1145/3373464.3373472

[11] Daniel Arp, Erwin Quiring, Feargus Pendlebury, Alexander Warnecke, Fabio Pierazzi, Christian Wressnegger, Lorenzo Cavallaro, and Konrad Rieck. 2022. Dos and don'ts of machine learning in computer security. In *Proc. of the USENIX Security Symposium*.

[12] Daniel Arp, Michael Spreitzenbarth, Malte Hübner, Hugo Gascon, and Konrad Rieck. 2014. DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket. *Symposium on Network and Distributed System Security (NDSS)*. https://doi.org/10.14722/ndss.2014.23247

[13] Anish Athalye, Nicholas Carlini, and David A. Wagner. 2018. Obfuscated Gradients Give a False Sense of Security: Circumventing Defenses to Adversarial Examples. *CoRR* abs/1802.00420 (2018). arXiv:1802.00420 http://arxiv.org/abs/1802.00420

[14] Manuel Baena-Garćıa, José del Campo-Ávila, Raúl Fidalgo, Albert Bifet, Ricard Gavaldà, and Rafael Morales-Bueno. 2006. Early Drift Detection Method.

[15] Maroua Bahri, Albert Bifet, Silviu Maniu, and Heitor Murilo Gomes. 2020. Survey on Feature Transformation Techniques for Data Streams. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, Christian Bessiere (Ed.). International Joint Conferences on Artificial Intelligence Organization, 4796–4802. https://doi.org/10.24963/ijcai.2020/668 Survey track.

[16] Zoltan Balazs. 2020. CUJO AI Partners with Microsoft for the Machine Learning Security Evasion Competition 2020. https://cujo.com/machine-learning-security-evasion-competition-2020/.

[17] Willi Ballenthin and Moritz Raabe. 2020. capa: Automatically Identify Malware Capabilities. https://www.mandiant.com/resources/blog/capa-automatically-identify-malware-capabilities. https://www.mandiant.com/resources/blog/capa-automatically-identify-malware-capabilities

[18] Tamy Beppler, Marcus Botacin, Fabrício Ceschin, Luiz E S Oliveira, and André Grégio. 2019. L(a)ying in (Test)Bed: How Biased Datasets Produce Impractical Results for Actual Malware Families' Classification. In *Information Security* (2019-01-01), Zhiqiang Lin, Charalampos Papamanthou, and Michalis Polychronakis (Eds.). Springer International Publishing, Cham, 381–401. https://link.springer.com/chapter/10.1007/978-3-030-30215-3_19

[19] Lukas Bieringer, Kathrin Grosse, Michael Backes, Battista Biggio, and Katharina Krombholz. 2022. Industrial practitioners' mental models of adversarial machine learning. In *Eighteenth Symposium on Usable Privacy and Security (SOUPS 2022)*. USENIX Association, Boston, MA, 97–116. https://www.usenix.org/conference/soups2022/presentation/bieringer

[20] Albert Bifet, Ricard Gavaldà, Geoff Holmes, and Bernhard Pfahringer. 2018. *Machine Learning for Data Streams with Practical Examples in MOA*. MIT Press. https://moa.cms.waikato.ac.nz/book/.

[21] Albert Bifet and Ricard Gavaldà. 2007. Learning from Time-Changing Data with Adaptive Windowing, In SIAM Int. Conf. on Data Mining. *SIAM Int. Conf. on Data Mining*.

[22] Albert Bifet, Geoff Holmes, Richard Kirkby, and Bernhard Pfahringer. 2010. MOA: Massive Online Analysis. *J. Mach. Learn. Res.* 11 (Aug. 2010), 1601–1604.

[23] C. M. Bishop. 2006. *Pattern Recognition and Machine Learning*. Springer.

[24] Franziska Boenisch, Verena Battis, Nicolas Buchmann, and Maija Poikela. 2021. "I Never Thought About Securing My Machine Learning Systems": A Study of Security and Privacy Awareness of Machine Learning Practitioners. In *Proceedings of Mensch Und Computer 2021* (Ingolstadt, Germany) *(MuC '21)*. Association for Computing Machinery, New York, NY, USA, 520–546. https://doi.org/10.1145/3473856.3473869

[25] Marcus Botacin, Giovanni Bertão, Paulo de Geus, André Grégio, Christopher Kruegel, and Giovanni Vigna. 2020. On the Security of Application Installer & Online Software Repositories *(DIMVA)*. Springer.

[26] Marcus Botacin, Fabricio Ceschin, Paulo de Geus, and André Grégio. 2020. We Need to Talk About AntiViruses: Challenges & Pitfalls of AV Evaluations. *Computers & Security* (2020), 101859. https://doi.org/10.1016/j.cose.2020.101859

[27] Marcus Botacin, Felipe Duarte Domingues, Fabrício Ceschin, Raphael Machnicki, Marco Antonio Zanata Alves, Paulo Lício de Geus, and André Grégio. 2022. AntiViruses under the microscope: A hands-on perspective. *Computers & Security* 112 (2022), 102500. https://doi.org/10.1016/j.cose.2021.102500

[28] M. Botacin, L. Galante, F. Ceschin, P. C. Santos, L. Carro, P. de Geus, A. Grégio, and M. A. Z. Alves. 2019. The AV says: Your Hardware Definitions Were Updated!. In *2019 14th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*. 27–34.

[29] Leo Breiman. 1996. Bagging Predictors. *Mach. Learn.* 24, 2 (Aug. 1996), 123–140. https://doi.org/10.1023/A:1018054314350

[30] Elie Bursztein and Daniela Oliveira. 2019. Deconstructing the Phishing Campaigns that Target Gmail Users. *Black Hat USA 2019* (2019). https://elie.net/talk/deconstructing-the-phishing-campaigns-that-target-gmail-users/.

[31] Alejandro Calleja, Alejandro Martín, Héctor D. Menéndez, Juan Tapiador, and David Clark. 2018. Picking on the family: Disrupting android malware triage by forcing misclassification. *Expert Systems with Applications* 95 (2018), 113 – 126. https://doi.org/10.1016/j.eswa.2017.11.032

[32] Nicholas Carlini and David A. Wagner. 2016. Towards Evaluating the Robustness of Neural Networks. *CoRR* abs/1608.04644 (2016). arXiv:1608.04644 http://arxiv.org/abs/1608.04644

[33] Lorenzo Cavallaro. 2019. When the Magic Wears Off: Flaws in ML for Security Evaluations (and What to Do about It). USENIX Association, Burlingame, CA.

[34] Fabrício Ceschin, Marcus Botacin, Heitor Murilo Gomes, Luiz S Oliveira, and André Grégio. 2019. Shallow Security: On the Creation of Adversarial Variants to Evade Machine Learning-Based Malware Detectors. In *Proceedings of the 3rd Reversing and Offensive-Oriented Trends Symposium* (2019-11-28) *(ROOTS'19)*. Association for Computing Machinery, Vienna, Austria. https://doi.org/10.1145/3375894.3375898

[35] Fabrício Ceschin, Marcus Botacin, Heitor Murilo Gomes, Felipe Pinagé, Luiz S Oliveira, and André Grégio. 2022. Fast & Furious: On the modelling of malware detection as an evolving data stream. *Expert Systems with Applications* (2022), 118590. https://doi.org/10.1016/j.eswa.2022.118590

[36] Fabricio Ceschin, Marcus Botacin, Gabriel Lüders, Heitor Murilo Gomes, Luiz Oliveira, and Andre Gregio. 2020. No Need to Teach New Tricks to Old Malware: Winning an Evasion Challenge with XOR-Based Adversarial Samples. In *Reversing and Offensive-Oriented Trends Symposium* (2020-11-01) *(ROOTS'20)*. Association for Computing Machinery, Vienna, Austria, 13–22. https://doi.org/10.1145/3433667.3433669

[37] Fabrício Ceschin, Felipe Pinage, Marcos Castilho, David Menotti, Luis S Oliveira, and André Gregio. 2018. The Need for Speed: An Analysis of Brazilian Malware Classifers. *IEEE Security Privacy* 16, 6 (2018), 31–41.

[38] Tanmoy Chakraborty, Fabio Pierazzi, and V. S. Subrahmanian. 2020. EC2: Ensemble Clustering and Classification for Predicting Android Malware Families. *IEEE Trans. Dependable Secur. Comput.* 17, 2 (mar 2020), 262–277. https://doi.org/10.1109/TDSC.2017.2739145

[39] Nitesh Chawla, Kevin Bowyer, Lawrence Hall, and W. Kegelmeyer. 2002. SMOTE: Synthetic Minority Over-sampling Technique. *J. Artif. Intell. Res. (JAIR)* 16 (01 2002), 321–357. https://doi.org/10.1613/jair.953

[40] Li Chen. 2018. Deep Transfer Learning for Static Malware Classification. *CoRR* abs/1812.07606 (2018). arXiv:1812.07606 http://arxiv.org/abs/1812.07606

[41] Li Chen, Ravi Sahita, Jugal Parikh, and Marc Marino. 2020. STAMINA Deep Learning for Malware Protection. https://www.intel.com/content/www/us/en/artificial-intelligence/documents/stamina-deep-learning-for-malware-protection-whitepaper.html

[42] Cody Coleman, Daniel Kang, Deepak Narayanan, Luigi Nardi, Tian Zhao, Jian Zhang, Peter Bailis, Kunle Olukotun, Chris Ré, and Matei Zaharia. 2019. Analysis of DAWNBench, a Time-to-Accuracy Machine Learning Performance Benchmark. *SIGOPS Oper. Syst. Rev.* 53, 1 (July 2019), 14–25. https://doi.org/10.1145/3352020.3352024

[43] N. Dalal and B. Triggs. 2005. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, Vol. 1. 886–893 vol. 1.

[44] Jesse Davis and Mark Goadrich. 2006. The Relationship between Precision-Recall and ROC Curves. In *Proceedings of the 23rd International Conference on Machine Learning* (Pittsburgh, Pennsylvania, USA) *(ICML '06)*. Association for Computing Machinery, New York, NY, USA, 233–240. https://doi.org/10.1145/1143844.1143874

[45] M. Dehghan, H. Beigy, and Poorya Zaremoodi. 2016. A novel concept drift detection method in data streams using ensemble classifiers. *Intell. Data Anal.* 20 (2016), 1329–1350.

[46] Amit Deo, Santanu Kumar Dash, Guillermo Suarez-Tangil, Volodya Vovk, and Lorenzo Cavallaro. 2016. Prescience: Probabilistic Guidance on the Retraining Conundrum for Malware Detection. In *Proceedings of the 2016 ACM Workshop on Artificial Intelligence and Security* (Vienna, Austria) *(AISec '16)*. Association for Computing Machinery, New York, NY, USA.

[47] M. Dev, H. Gupta, S. Mehta, and B. Balamurugan. 2016. Cache implementation using collective intelligence on cloud based antivirus architecture. In *2016 International Conference on Advanced Communication Control and Computing Technologies (ICACCCT)*. 593–595.

[48] Cynthia Dwork and Aaron Roth. 2014. The Algorithmic Foundations of Differential Privacy. *Found. Trends Theor. Comput. Sci.* 9, 3–4 (Aug. 2014), 211–407. https://doi.org/10.1561/0400000042

[49] Thijs van Ede, Hojjat Aghakhani, Noah Spahn, Riccardo Bortolameotti, Marco Cova, Andrea Continella, Maarten van Steen, Andreas Peter, Christopher Kruegel, and Giovanni Vigna. 2022. DEEPCASE: Semi-Supervised Contextual Analysis of Security Events. In *2022 IEEE Symposium on Security and Privacy (SP)*. 522–539. https://doi.org/10.1109/SP46214.2022.9833671

[50] M. A. El Hadj, M. Erradi, A. Khoumsi, and Y. Benkaouz. 2018. Validation and Correction of Large Security Policies: A Clustering and Access Log Based Approach. In *2018 IEEE International Conference on Big Data (Big Data)*. 5330–5332.

[51] Nicholas Epley and Thomas Gilovich. 2006. The Anchoring-and-Adjustment Heuristic: Why the Adjustments Are Insufficient. *Psychological Science* 17, 4 (2006), 311–318. https://doi.org/10.1111/j.1467-9280.2006.01704.x arXiv:https://doi.org/10.1111/j.1467-9280.2006.01704.x PMID: 16623688.

[52] C. Ferri, J. Hernández-Orallo, and R. Modroiu. 2009. An experimental comparison of performance measures for classification. *Pattern Recognition Letters* 30, 1 (2009), 27 – 38. https://doi.org/10.1016/j.patrec.2008.08.010

[53] FireEye. 2019. StringSifter. https://github.com/fireeye/stringsifter.

[54] William Fleshman, Edward Raff, Jared Sylvester, Steven Forsyth, and Mark McLean. 2018. Non-Negative Networks Against Adversarial Attacks. https://arxiv.org/abs/1806.06108.

[55] Yoav Freund and Robert E Schapire. 1997. A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *J. Comput. Syst. Sci.* 55, 1 (Aug. 1997), 119–139. https://doi.org/10.1006/jcss.1997.1504

[56] Krishna Gade, Sahin Geyik, Krishnaram Kenthapadi, Varun Mithal, and Ankur Taly. 2020. Explainable AI in Industry: Practical Challenges and Lessons Learned. In *Companion Proceedings of the Web Conference 2020* (Taipei, Taiwan) *(WWW '20)*. Association for Computing Machinery, New York, NY, USA, 303–304. https://doi.org/10.1145/3366424.3383110

[57] Lucas Galante, Marcus Botacin, André Grégio, and Paulo de Geus. 2019. Machine Learning for Malware Detection: Beyond Accuracy Rates *(Brazilian Security Symposium (SBSeg))*.

[58] M. Galar, A. Fernandez, E. Barrenechea, H. Bustince, and F. Herrera. 2012. A Review on Ensembles for the Class Imbalance Problem: Bagging-, Boosting-, and Hybrid-Based Approaches. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 42, 4 (2012), 463–484.

[59] João Gama, Pedro Medas, Gladys Castillo, and Pedro Rodrigues. 2004. Learning with Drift Detection. In *Advances in Artificial Intelligence – SBIA 2004*, Ana L. C. Bazzan and Sofiane Labidi (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 286–295.

[60] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. 2014. A Survey on Concept Drift Adaptation. *ACM Comput. Surv.* 46, 4, Article 44 (March 2014), 37 pages. https://doi.org/10.1145/2523813

[61] R. Stuart Geiger, Kevin Yu, Yanlai Yang, Mindy Dai, Jie Qiu, Rebekah Tang, and Jenny Huang. 2020. Garbage in, Garbage out? Do Machine Learning Application Papers in Social Computing Report Where Human-Labeled Training Data Comes From?. In *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency* (Barcelona, Spain) *(FAT* '20)*. Association for Computing Machinery, New York, NY, USA, 325–336. https://doi.org/10.1145/3351095.3372862

[62] Daniel Gibert, Carles Mateu, and Jordi Planes. 2020. The rise of machine learning for detection and classification of malware: Research developments, trends and challenges. *Journal of Network and Computer Applications* 153 (2020), 102526. https://doi.org/10.1016/j.jnca.2019.102526

[63] Heitor Murilo Gomes, Albert Bifet, Jesse Read, Jean Paul Barddal, Fabrício Enembreck, Bernhard Pfahringer, Geoff Holmes, and Talel Abdessalem. 2017. Adaptive random forests for evolving data stream classification. *Machine Learning* (06 2017), 1–27. https://doi.org/10.1007/s10994-017-5642-8

[64] Heitor Murilo Gomes, Jesse Read, Albert Bifet, Jean Paul Barddal, and João Gama. 2019. Machine Learning for Streaming Data: State of the Art, Challenges, and Opportunities. *SIGKDD Explor. Newsl.* 21, 2 (Nov. 2019), 6–22. https://doi.org/10.1145/3373464.3373470

[65] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative Adversarial Networks. arXiv:1406.2661 [stat.ML]

[66] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and Harnessing Adversarial Examples. arXiv:1412.6572 [stat.ML]

[67] Dimitris Gritzalis, Giulia Iseppi, Alexios Mylonas, and Vasilis Stavrou. 2018. Exiting the Risk Assessment Maze: A Meta-Survey. *ACM Comput. Surv.* 51, 1, Article 11 (Jan. 2018), 30 pages. https://doi.org/10.1145/3145905

[68] Aurlien Gron. 2017. *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems* (1st ed.). O'Reilly Media, Inc.

[69] Kathrin Grosse, Lukas Bieringer, Tarek R. Besold, Battista Biggio, and Katharina Krombholz. 2023. Machine Learning Security in Industry: A Quantitative Survey. *IEEE Transactions on Information Forensics and Security* 18 (2023), 1749–1762. https://doi.org/10.1109/tifs.2023.3251842

[70] Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick D. McDaniel. 2017. Adversarial Examples for Malware Detection. In *ESORICS*.

[71] Chuan Guo, Jacob R. Gardner, Yurong You, Andrew Gordon Wilson, and Kilian Q. Weinberger. 2019. Simple Black-box Adversarial Attacks. *CoRR* abs/1905.07121 (2019). arXiv:1905.07121 http://arxiv.org/abs/1905.07121

[72] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. 2009. The WEKA Data Mining Software: An Update. *SIGKDD Explor. Newsl.* 11, 1 (Nov. 2009), 10–18. https://doi.org/10.1145/1656274.1656278

[73] Paul Hick, Emile Aben, Kc Claffy, and Josh Polterock. 2007. the CAIDA DDoS attack 2007 dataset. https://www.caida.org/data/passive/ddos-20070804_dataset.xml.

[74] Mohammad Javad Hosseini, Ameneh Gholipour, and Hamid Beigy. 2015. An ensemble of cluster-based classifiers for semi-supervised classification of non-stationary data streams. *Knowledge and Information Systems* 46 (04 2015). https://doi.org/10.1007/s10115-015-0837-4

[75] Jeremy Howard and Sebastian Ruder. 2018. Fine-tuned Language Models for Text Classification. *CoRR* abs/1801.06146 (2018). arXiv:1801.06146 http://arxiv.org/abs/1801.06146

[76] Weiwei Hu and Ying Tan. 2017. Generating Adversarial Malware Examples for Black-Box Attacks Based on GAN. *CoRR* abs/1702.05983 (2017). arXiv:1702.05983 http://arxiv.org/abs/1702.05983

[77] Keman Huang, Michael Siegel, and Stuart Madnick. 2018. Systematically Understanding the Cyber Attack Business: A Survey. *ACM Comput. Surv.* 51, 4, Article 70 (July 2018), 36 pages. https://doi.org/10.1145/3199674

[78] Mederic Hurier, Guillermo Suarez-Tangil, Santanu Kumar Dash, Tegawende F. Bissyande, Yves Le Traon, Jacques Klein, and Lorenzo Cavallaro. 2017. Euphony: Harmonious Unification of Cacophonous Anti-Virus Vendor Labels for Android Malware. In *IEEE International Working Conference on Mining Software Repositories.* IEEE Computer Society, 425–435. https://doi.org/10.1109/MSR.2017.57

[79] Mahbub Hussain, Jordan J. Bird, and Diego R. Faria. 2019. A Study on CNN Transfer Learning for Image Classification. In *Advances in Computational Intelligence Systems*, Ahmad Lotfi, Hamid Bouchachia, Alexander Gegov, Caroline Langensiepen, and Martin McGinnity (Eds.). Springer International Publishing, Cham, 191–202.

[80] Chris Jarabek, David Barrera, and John Aycock. 2012. ThinAV: Truly Lightweight Mobile Cloud-Based Anti-Malware. In *Proceedings of the 28th Annual Computer Security Applications Conference* (Orlando, Florida, USA) *(ACSAC '12)*. Association for Computing Machinery, New York, NY, USA, 209–218. https://doi.org/10.1145/2420950.2420983

[81] Karen Spärck Jones. 1972. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation* 28 (1972), 11–21.

[82] Roberto Jordaney, Kumar Sharad, Santanu K. Dash, Zhi Wang, Davide Papini, Ilia Nouretdinov, and Lorenzo Cavallaro. 2017. Transcend: Detecting Concept Drift in Malware Classification Models. In *26th USENIX Security Symposium (USENIX Security 17)*. USENIX Association, Vancouver, BC, 625–642. https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/jordaney

[83] Alex Kantchelian, Sadia Afroz, Ling Huang, Aylin Caliskan Islam, Brad Miller, Michael Carl Tschantz, Rachel Greenstadt, Anthony D. Joseph, and J. D. Tygar. 2013. Approaches to Adversarial Drift. In *Proceedings of the 2013 ACM Workshop on Artificial Intelligence and Security* (Berlin, Germany) *(AISec '13)*. Association for Computing Machinery, New York, NY, USA, 99–110. https://doi.org/10.1145/2517312.2517320

[84] Alex Kantchelian, Michael Carl Tschantz, Sadia Afroz, Brad Miller, Vaishaal Shankar, Rekha Bachwani, Anthony D. Joseph, and J. D. Tygar. 2015. Better malware ground truth: Techniques for weighting anti-virus vendor labels. In *AISec 2015 - Proceedings of the 8th ACM Workshop on Artificial Intelligence and Security, co-located with CCS 2015*. Association for Computing Machinery, Inc, New York, New York, USA, 45–56. https://doi.org/10.1145/2808769.2808780

[85] Shachar Kaufman, Saharon Rosset, and Claudia Perlich. 2011. Leakage in Data Mining: Formulation, Detection, and Avoidance. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* 6, 556–563. https://doi.org/10.1145/2020408.2020496

[86] Harsurinder Kaur, Husanbir Singh Pannu, and Avleen Kaur Malhi. 2019. A systematic review on imbalanced data challenges in machine learning: Applications and solutions. *ACM Computing Surveys (CSUR)* 52, 4 (2019), 1–36.

[87] R. K. Keser and B. U. Töreyin. 2019. Autoencoder Based Dimensionality Reduction of Feature Vectors for Object Recognition. In *2019 15th International Conference on Signal-Image Technology Internet-Based Systems (SITIS)*. 577–584.

[88] A. Korzybski. 1931. *A Non-Aristotelian System and Its Necessity for Rigour in Mathematics and Physics: Abstract.*

[89] Georg Krempl, Indre Žliobaite, Dariusz Brzeziński, Eyke Hüllermeier, Mark Last, Vincent Lemaire, Tino Noack, Ammar Shaker, Sonja Sievi, Myra Spiliopoulou, and Jerzy Stefanowski. 2014. Open Challenges for Data Stream Mining Research. *SIGKDD Explor. Newsl.* 16, 1 (Sept. 2014), 1–10. https://doi.org/10.1145/2674026.2674028

[90] Alex Krizhevsky. 2012. Learning Multiple Layers of Features from Tiny Images. *University of Toronto* (05 2012).

[91] Nir Kshetri. 2021. Economics of Artificial Intelligence in Cybersecurity. *IT Professional* 23, 5 (2021), 73–77. https://doi.org/10.1109/MITP.2021.3100177

[92] Todd Kulesza, Saleema Amershi, Rich Caruana, Danyel Fisher, and Denis Charles. 2014. Structured labeling to facilitate concept evolution in machine learning. *Conference on Human Factors in Computing Systems - Proceedings.* https://doi.org/10.1145/2556288.2557238

[93] M. Kumar and R. Mathur. 2014. Unsupervised outlier detection technique for intrusion detection in cloud computing. In *International Conference for Convergence for Technology-2014*. 1–4.

[94] S. Kumar and C. Bhim Bhan Singh. 2018. A Zero-Day Resistant Malware Detection Method for Securing Cloud Using SVM and Sandboxing Techniques. In *2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT)*. 1397–1402.

[95] Vincent Lemaire, Christophe Salperwyck, and Alexis Bondu. 2015. *A Survey on Supervised Classification on Data Streams*. Springer International Publishing, Cham, 88–125. https://doi.org/10.1007/978-3-319-17551-5_4

[96] Linyi Li, Xiangyu Qi, Tao Xie, and Bo Li. 2020. SoK: Certified Robustness for Deep Neural Networks. arXiv:2009.04131 [cs.LG]

[97] LightGBM. 2018. LightGBM. https://lightgbm.readthedocs.io/en/latest/.

[98] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. 2012. Isolation-Based Anomaly Detection. *ACM Trans. Knowl. Discov. Data* 6, 1, Article 3 (March 2012), 39 pages. https://doi.org/10.1145/2133360.2133363

[99] Aravind Machiry, Nilo Redini, Eric Gustafson, Yanick Fratantonio, Yung Ryn Choe, Christopher Kruegel, and Giovanni Vigna. 2018. Using Loops For Malware Classification Resilient to Feature-Unaware Perturbations. In *Proceedings of the 34th Annual Computer Security Applications Conference* (San Juan, PR, USA) *(ACSAC '18)*. Association for Computing Machinery, New York, NY, USA, 112–123. https://doi.org/10.1145/3274694.3274731

[100] Davide Maiorca, Battista Biggio, and Giorgio Giacinto. 2019. Towards adversarial malware detection: Lessons learned from PDF-based attacks. *ACM Computing Surveys (CSUR)* 52, 4 (2019), 1–36.

[101] Nuno Martins, José Magalhães Cruz, Tiago Cruz, and Pedro Henriques Abreu. 2020. Adversarial Machine Learning Applied to Intrusion and Malware Scenarios: A Systematic Review. *IEEE Access* 8 (2020), 35403–35419. https://doi.org/10.1109/ACCESS.2020.2974752

[102] Mohammad M. Masud, Tahseen M. Al-Khateeb, Kevin W. Hamlen, Jing Gao, Latifur Khan, Jiawei Han, and Bhavani Thuraisingham. 2008. Cloud-Based Malware Detection for Evolving Data Streams. *ACM Trans. Manage. Inf. Syst.* (Oct. 2008).

[103] Michael Armbrust Matei Zaharia, Tathagata Das and Reynold Xin. 2016. *Spark Structured Streaming: A new high-level API for streaming*. https://databricks.com/blog/2016/07/28/structured-streaming-in-apache-spark.html

[104] Marco Melis, Ambra Demontis, Maura Pintor, Angelo Sotgiu, and Battista Biggio. 2019. secml: A Python Library for Secure and Explainable Machine Learning. *arXiv preprint arXiv:1912.10013* (2019).

[105] Donald Michie, D. J. Spiegelhalter, C. C. Taylor, and John Campbell (Eds.). 1994. *Machine Learning, Neural and Statistical Classification*. Ellis Horwood, Upper Saddle River, NJ, USA.

[106] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. *CoRR* abs/1301.3781 (2013). http://arxiv.org/abs/1301.3781

[107] Jacob Montiel, Max Halford, Saulo Martiello Mastelini, Geoffrey Bolmier, Raphael Sourty, Robin Vaysse, Adil Zouitine, Heitor Murilo Gomes, Jesse Read, Talel Abdessalem, and Albert Bifet. 2020. River: machine learning for streaming data in Python. arXiv:2012.04740 [cs.LG]

[108] Jacob Montiel, Jesse Read, Albert Bifet, and Talel Abdessalem. 2018. Scikit-Multiflow: A Multi-output Streaming Framework. *Journal of Machine Learning Research* (2018).

[109] Maryam M. Najafabadi, Flavio Villanustre, Taghi M. Khoshgoftaar, Naeem Seliya, Randall Wald, and Edin Muharemagic. 2015. Deep learning applications and challenges in big data analytics. *Journal of Big Data* 2, 1 (24 Feb 2015), 1. https://doi.org/10.1186/s40537-014-0007-7

[110] A. Narayanan, L. Yang, L. Chen, and L. Jinliang. 2016. Adaptive and scalable Android malware detection through online learning. In *2016 International Joint Conference on Neural Networks (IJCNN)*.

[111] NetResec. 2020. Publicly available PCAP files. https://www.netresec.com/?page=PcapFiles.

[112] Andre Nguyen, Richard Zak, Luke Edward Richards, Maya Fuchs, Fred Lu, Robert Brandon, Garay David Lopez Munoz, Ed Raff, Charles Nicholas, and James Holt. 2022. Minimizing Compute Costs: When Should We Run More Expensive Malware Analysis? *Conference on Applied Machine Learning in Information Security (CAMLIS) 2022*. https://www.camlis.org/andre-nguyen-2022

[113] Andrei Paleyes, Raoul-Gabriel Urma, and Neil D. Lawrence. 2022. Challenges in Deploying Machine Learning: A Survey of Case Studies. *ACM Comput. Surv.* 55, 6, Article 114 (dec 2022), 29 pages. https://doi.org/10.1145/3533378

[114] Ruoming Pang, Mark Allman, Mike Bennett, Jason Lee, Vern Paxson, and Brian Tierney. 2005. A first look at modern enterprise traffic. In *Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*. 2–2.

[115] P. K. Panigrahi. 2012. A Comparative Study of Supervised Machine Learning Techniques for Spam E-mail Filtering. In *2012 Fourth International Conference on Computational Intelligence and Communication Networks*. 506–512.

[116] Nicolas Papernot, Fartash Faghri, Nicholas Carlini, Ian Goodfellow, Reuben Feinman, Alexey Kurakin, Cihang Xie, Yash Sharma, Tom Brown, Aurko Roy, Alexander Matyasko, Vahid Behzadan, Karen Hambardzumyan, Zhishuai Zhang, Yi-Lin Juang, Zhi Li, Ryan Sheatsley, Abhibhav Garg, Jonathan Uesato, Willi Gierke, Yinpeng Dong, David Berthelot, Paul Hendricks, Jonas Rauber, and Rujun Long. 2018. Technical Report on the CleverHans v2.1.0 Adversarial Examples Library. *arXiv preprint arXiv:1610.00768* (2018).

[117] Nicolas Papernot, Patrick McDaniel, Arunesh Sinha, and Michael P. Wellman. 2018. SoK: Security and Privacy in Machine Learning. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*. 399–414. https://doi.org/10.1109/EuroSP.2018.00035

[118] Nicolas Papernot, Patrick D. McDaniel, Ian J. Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. 2016. Practical Black-Box Attacks against Deep Learning Systems using Adversarial Examples. *CoRR* abs/1602.02697 (2016). arXiv:1602.02697

http://arxiv.org/abs/1602.02697

[119] Nicolas Papernot, Patrick D. McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. 2015. The Limitations of Deep Learning in Adversarial Settings. *CoRR* abs/1511.07528 (2015). arXiv:1511.07528 http://arxiv.org/abs/1511.07528

[120] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[121] Feargus Pendlebury, Fabio Pierazzi, Roberto Jordaney, Johannes Kinder, and Lorenzo Cavallaro. 2019. TESSERACT: Eliminating Experimental Bias in Malware Classification across Space and Time. In *28th USENIX Security Symposium (USENIX Security 19)*. USENIX Association, Santa Clara, CA, 729–746. https://www.usenix.org/conference/usenixsecurity19/presentation/pendlebury

[122] Ryan Permeh. 2017. True AI/ML vs. Glorified Signature-Based Solutions. https://threatvector.cylance.com/en_us/home/true-ai-ml-vs-glorified-signature-based-solutions.html.

[123] L. T. Phong, Y. Aono, T. Hayashi, L. Wang, and S. Moriai. 2018. Privacy-Preserving Deep Learning via Additively Homomorphic Encryption. *IEEE Transactions on Information Forensics and Security* 13, 5 (2018), 1333–1345.

[124] Alec Radford. 2018. Improving Language Understanding by Generative Pre-Training.

[125] Edward Raff, Jon Barker, Jared Sylvester, Robert Brandon, Bryan Catanzaro, and Charles Nicholas. 2017. Malware Detection by Eating a Whole EXE. https://arxiv.org/abs/1710.09435.

[126] Maithra Raghu, Chiyuan Zhang, Jon M. Kleinberg, and Samy Bengio. 2019. Transfusion: Understanding Transfer Learning with Applications to Medical Imaging. *CoRR* abs/1902.07208 (2019). arXiv:1902.07208 http://arxiv.org/abs/1902.07208

[127] Shahbaz Rezaei and Xin Liu. 2019. A Target-Agnostic Attack on Deep Models: Exploiting Security Vulnerabilities of Transfer Learning. *CoRR* abs/1904.04334 (2019). arXiv:1904.04334 http://arxiv.org/abs/1904.04334

[128] A. Rocha, W. J. Scheirer, C. W. Forstall, T. Cavalcante, A. Theophilo, B. Shen, A. R. B. Carvalho, and E. Stamatatos. 2017. Authorship Attribution for Social Media Forensics. *IEEE Transactions on Information Forensics and Security* 12, 1 (2017), 5–33.

[129] Yuji Roh, Geon Heo, and Steven Whang. 2019. A Survey on Data Collection for Machine Learning: A Big Data - AI Integration Perspective. *IEEE Transactions on Knowledge and Data Engineering* PP (10 2019), 1–1. https://doi.org/10.1109/TKDE.2019.2946162

[130] Yuji Roh, Geon Heo, and Steven Euijong Whang. 2021. A Survey on Data Collection for Machine Learning: A Big Data - AI Integration Perspective. *IEEE Transactions on Knowledge and Data Engineering* 33, 4 (2021), 1328–1347. https://doi.org/10.1109/TKDE.2019.2946162

[131] Mahsa Salehi and Lida Rashidi. 2018. A Survey on Anomaly detection in Evolving Data: [with Application to Forest Fire Risk Prediction]. *ACM SIGKDD Explorations Newsletter* 20, 1 (2018), 13–23.

[132] Joshua Saxe. 2020. Sophos AI YaraML Rules Repository. https://github.com/sophos-ai/yaraml_rules.

[133] Joshua Saxe and Hillary Sanders. 2018. *Malware Data Science: Attack Detection and Attribution*. No Starch Press, San Francisco, CA, USA.

[134] Sebastian Schelter, Felix Bießmann, Tim Januschowski, David Salinas, Stephan Seufert, and Gyuri Szarvas. 2018. On Challenges in Machine Learning Model Management. *IEEE Data Eng. Bull.* 41 (2018), 5–15.

[135] Bernhard Schölkopf, Robert Williamson, Alex Smola, John Shawe-Taylor, and John Platt. 1999. Support Vector Method for Novelty Detection. In *Proceedings of the 12th International Conference on Neural Information Processing Systems* (Denver, CO) *(NIPS'99)*. MIT Press, Cambridge, MA, USA, 582–588.

[136] Marcos Sebastián, Richard Rivera, Platon Kotzias, and Juan Caballero. 2016. *AVclass: A Tool for Massive Malware Labeling*. Springer International Publishing, Cham, 230–253. https://doi.org/10.1007/978-3-319-45719-2_11

[137] Amazon Web Services. 2020. Amazon Machine Learning Key Concepts. https://docs.aws.amazon.com/machine-learning/latest/dg/amazon-machine-learning-key-concepts.html.

[138] Ali Shafahi, Parsa Saadatpanah, Chen Zhu, Amin Ghiasi, Christoph Studer, David W. Jacobs, and Tom Goldstein. 2019. Adversarially robust transfer learning. *CoRR* abs/1905.08232 (2019). arXiv:1905.08232 http://arxiv.org/abs/1905.08232

[139] Junming Shao, Zahra Ahmadi, and Stefan Kramer. 2014. Prototype-Based Learning on Concept-Drifting Data Streams. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (New York, New York, USA) *(KDD '14)*. Association for Computing Machinery, New York, NY, USA, 412–421. https://doi.org/10.1145/2623330.2623609

[140] Ali Shiravi, Hadi Shiravi, Mahbod Tavallaee, and Ali A Ghorbani. 2012. Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *computers & security* 31, 3 (2012), 357–374.

[141] Eddie Shoesmith, George E. P. Box, and Norman R. Draper. 1987. Empirical Model-Building and Response Surfaces. *The Statistician* 37 (1987), 82.

[142] Sajjad Kamali Siahroudi, Poorya Zare Moodi, and Hamid Beigy. 2018. Detection of evolving concepts in non-stationary data streams: A multiple kernel learning approach. *Expert Systems with Applications* 91 (2018), 187–197. https://doi.org/10.1016/j.eswa.2017.08.033

[143] Anshuman Singh, Andrew Walenstein, and Arun Lakhotia. 2012. Tracking Concept Drift in Malware Families. In *Proceedings of the 5th ACM Workshop on Security and Artificial Intelligence* (Raleigh, North Carolina, USA) *(AISec '12)*. Association for Computing Machinery, New York, NY, USA.

[144] Apache Spark. 2020. *Spark Streaming*. https://spark.apache.org/streaming/

[145] J. Michael Steele. 2006. Models: Masterpieces and Lame Excuses. http://www-stat.wharton.upenn.edu/~steele/Rants/ModelsMandLE.html.

[146] R. Sun, M. Botacin, N. Sapountzis, X. Yuan, M. Bishop, D. E. Porter, X. Li, A. Gregio, and D. Oliveira. 2020. A Praise for Defensive Programming: Leveraging Uncertainty for Effective Malware Mitigation. *IEEE Transactions on Dependable and Secure Computing* (2020), 1–1.

[147] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2014. Going Deeper with Convolutions. *CoRR* abs/1409.4842 (2014). arXiv:1409.4842 http://arxiv.org/abs/1409.4842

[148] Rahim Taheri, Reza Javidan, Mohammad Shojafar, Zahra Pooranian, Ali Miri, and Mauro Conti. 2019. On Defending Against Label Flipping Attacks on Malware Detection Systems. arXiv:1908.04473 [cs.LG]

[149] Vasilis G. Tasiopoulos and Sokratis K. Katsikas. 2014. Bypassing Antivirus Detection with Encryption. In *Proceedings of the 18th Panhellenic Conference on Informatics* (Athens, Greece) *(PCI '14)*. Association for Computing Machinery, New York, NY, USA, 1–2. https://doi.org/10.1145/2645791.2645857

[150] Lisa Torrey and Jude Shavlik. 2010. Transfer learning. In *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*. IGI global, 242–264.

[151] Xabier Ugarte-Pedrero, Mariano Graziano, and Davide Balzarotti. 2019. A Close Look at a Daily Dataset of Malware Samples. *ACM Trans. Priv. Secur.* 22, 1, Article 6 (jan 2019), 30 pages. https://doi.org/10.1145/3291061

[152] Todd Underwood. 2019. All of Our ML Ideas Are Bad (and We Should Feel Bad). USENIX Association, Dublin.

[153] VirusShare. 2019. VirusShare on Twitter. https://twitter.com/VXShare/status/1095411986949652480.

[154] VirusTotal. 2020. VirusTotal: Free Online Virus, Malware and URL Scanner. https://www.virustotal.com/.

[155] Kiri Wagstaff. 2012. Machine Learning that Matters. *CoRR* abs/1206.4656 (2012). arXiv:1206.4656 http://arxiv.org/abs/1206.4656

[156] Shenghui Wang, Stefan Schlobach, and Michel Klein. 2011. Concept drift and how to identify it. *Web Semantics: Science, Services and Agents on the World Wide Web* 9, 3 (2011), 247 – 265. https://doi.org/10.1016/j.websem.2011.05.003 Semantic Web Dynamics Semantic Web Challenge, 2010.

[157] K. Xu, Y. Li, R. Deng, K. Chen, and J. Xu. 2019. DroidEvolver: Self-Evolving Android Malware Detection System. In *2019 IEEE European Symposium on Security and Privacy (EuroS P)*.

[158] Poorya ZareMoodi, Hamid Beigy, and Sajjad Kamali Siahroudi. 2015. Novel class detection in data streams using local patterns and neighborhood graph. *Neurocomputing* 158 (2015), 234–245. https://doi.org/10.1016/j.neucom.2015.01.037

[159] Poorya ZareMoodi, Sajjad Kamali Siahroudi, and Hamid Beigy. 2019. Concept-evolution detection in non-stationary data streams: a fuzzy clustering approach. *Knowledge and Information Systems* 60 (09 2019). https://doi.org/10.1007/s10115-018-1266-y

[160] Shuofei Zhu, Jianjun Shi, Limin Yang, Boqin Qin, Ziyi Zhang, Linhai Song, and Gang Wang. 2020. Measuring and Modeling the Label Dynamics of Online Anti-Malware Engines. In *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, 2361–2378. https://www.usenix.org/conference/usenixsecurity20/presentation/zhu

[161] I. Žliobaitė. 2010. Change with Delayed Labeling: When is it Detectable?. In *2010 IEEE International Conference on Data Mining Workshops*. 843–850.