Machine Learning for Computer Security Detection Systems: Practical Feedback and Solutions

Anaël Beaugnon[✉] and Pierre Chifflier

French National Cybersecurity Agency (ANSSI), Paris, France
{anael.beaugnon,pierre.chifflier}@ssi.gouv.fr

Abstract. Machine learning based detection models can strengthen detection, but there remain some significant barriers to the widespread deployment of such techniques in operational detection systems. In this paper, we identify the main challenges to overcome and we provide both methodological guidance and practical solutions to address them. The solutions we propose are completely generic to be beneficial to any detection problem on any data type and are freely available in SecuML [1].

Keywords: Computer Security Detection Systems · Machine Learning · Human-Computer Interaction

1 Introduction

Machine learning detection models can be deployed in detection systems to improve the detection of yet unknown threats. Various research papers have tackled detection problems successfully with supervised learning (e.g. Android applications, PDF files, and memory dumps). Despite these encouraging results, there remain some significant barriers to the widespread deployment of machine learning in operational detection systems.

Security experts responsible for setting up detection methods may have little or no knowledge about machine learning. They may need advice, and ready-to-use solutions, to build supervised detection models that fulfill their operational constraints. Can a detection method based on machine learning process streaming data ? Machine learning models are reputed to be black-box methods among security experts. How can they trust such techniques to deploy them in operational detection systems ?

Moreover, evaluating detection models is not a straightforward procedure, while it is a critical step to ensure successful deployments. Security experts may need advice to evaluate detection models thoroughly and to address potential performance issues.

Finally, machine learning algorithms require annotated datasets that are hard to acquire in the context of detection systems. Few annotated datasets are public and they quickly become outdated. Besides, crowd-sourcing cannot be leveraged to get annotated datasets at low cost since the data are often sensitive and annotating requires expert knowledge.

In this paper, we provide practical feedback on building and deploying machine learning based detection models. We highlight the pitfalls to avoid, we identify key good practice elements, and we provide practical solutions to help security experts build ready-to-deploy supervised detection models. The solutions we propose are completely generic to be beneficial to any detection problem on any data type, and are freely available in SecuML [1]. They consist in interactive tools that hide some of the machine learning machinery to let security experts focus mainly on detection.

Section 2 provides an overview of machine learning in detection systems and presents the steps of the machine learning pipeline: feature extraction, data annotation, training, and evaluation. Then, Section 3 details the first two steps of the machine learning pipeline, while Section 4 focuses on the last two ones.

2 Detection Systems and Machine Learning

Supervised detection models can be deployed in detection systems, as an adjunct to traditional techniques, to strengthen detection. In this section, we present the operational constraints of computer security detection systems that supervised methods must also meet. Then, we provide an overview of machine learning in detection systems and we list the steps of the machine learning pipeline that are then further detailed in the next sections.

2.1 Constraints of Computer Security Detection Systems

Computer security detection systems monitor a network, or a system, to identify potential threats such as malicious files attached to email messages, or data exfiltrations. They involve several detection methods based on diverse techniques (e.g. signatures, expert systems, anomaly detection, supervised learning) operating in parallel to strengthen detection capabilities.

We define two roles that operate in security operation centers: *security administrators* and *security operators*. *Security administrators* are responsible for setting up detection methods. They define the *detection target*, i.e. in which circumstances an alert should be triggered. They also set up an *alert taxonomy*, i.e. the way the malicious behaviors are grouped into families that are then exploited to tag the alerts. Once security administrators have defined the detection target and the alert taxonomy, they implement and deploy detection methods accordingly. Thereafter, *security operators* analyze the alerts: they discard false alarms and take the appropriate actions in case of security incident.

Detection methods must meet the following operational constraints to ensure their operability in detection systems [16, 19].

Local and Online Processing. The sensitivity of the data processed by detection systems hinders the use of cloud computing services. In addition, detection methods must analyze streaming data that cannot be stored due to volumetry and privacy: they must enjoy a low time complexity to be executed in near real time.

Effectiveness. Both false positives and negatives are extremely costly and must therefore be minimized. One the one hand, if there are too many false positives, meaningless alarms will overwhelm security operators, and they will have less time to analyze the significant ones. On the other hand, false negatives can cause serious damage to the defended system.

Transparency. Transparency is crucial for both security administrators and operators. Security operators need information about why an alert has been triggered to assess its criticality and to discard false positives. Moreover, security administrators will not deploy black-box detection methods. They want to understand how detection methods work to trust them before their deployment.

Controllability. Security administrators shall retain control over detection methods. They must be able to update them to correct potential errors forthwith. Updating detection methods should be straightforward, since swiftness is crucial to detect emerging threats as soon as possible.

Robustness. Detection methods are deployed in adversarial environments: malicious behaviors constantly evolve as attackers attempt to deceive detectors. Detection methods must therefore be designed to resist evasion attempts. This constraint is related to *Effectiveness* but it concerns the ability to detect malicious behaviors that have been especially crafted to evade a given detection method. Assessing robustness means evaluating how much the detection performance drops when attackers slightly modify malicious behaviors with the intent of misleading detection.

2.2 Deploying Supervised Detection Models

Supervised Detection Models. Supervised detection models are binary classifiers that take as input an instance (e.g. a PDF file, a Windows Office document, an Android application, or the traffic associated to an IP address) and return the predicted label, benign or malicious, as output.

They are built with training algorithms from annotated data, a set of instances for which the label is known. Training algorithms find the discriminating characteristics automatically to build the detection rules.

Once the model has been trained on an annotated dataset, it can be deployed in the detection system to detect malicious instances automatically. In practice, most classifiers do not provide a binary answer (benign vs. malicious), but rather a probability of maliciousness. The detection system triggers an alert only if the probability of maliciousness is above the detection threshold set by security administrators. This probability allows to sort alerts according to the confidence of the predictions, and thus to identify which alerts security operators should analyze foremost.

Supervised Learning Pipeline. First of all, security administrators must clearly define their detection target since it significantly drives the building process of supervised detection models. Then, it is not straightforward to train ready-to-deploy supervised detection models, it involves several steps. In this paper, we consider the whole pipeline with security administrators and operators as its core since it is crucial to pursue real-world impact [22].

1. **Feature Extraction.** Standard machine learning algorithms do not take raw data as input. Security administrators must transform the instances into fixed-length vectors of features.
2. **Data Annotation.** Security administrators must annotate the data, i.e. associate a binary label, malicious or benign, to each instance according to the detection target.
3. **Training.** There are many model classes (i.e. types of supervised models): neural networks, random forests, k-nearest neighbors, etc. Security administrators must pick a model class before launching the training algorithm.
4. **Evaluation.** Security administrators must check whether the detection model meets the *effectiveness* and *robustness* constraints.

The first model trained is usually not satisfactory. Setting up a detection model is an iterative process where the evaluation phase provides avenues for improvement. At each iteration, security administrators train and validate a detection model. Based on the evaluation results, they can either consider that the detection model is good enough for deployment or perform a new iteration by modifying the annotated dataset, the extracted features or the model class.

3 Gathering Training Data

Unlabeled data can be easily acquired from a production environment, but security administrators must perform two steps to build a training dataset : 1) *feature extraction* to transform each instance into a fixed-length vector of features, and 2) *annotation* to associate a binary label, malicious or benign, to each instance.

3.1 Feature Extraction

Features are boolean, numerical, or categorical values describing an instance that detection models exploit to make decisions. For instance, *presence of JavaScript*, *number of Open Actions*, *number of images*, or *mean of the size of the objects*, can be numerical features computed for PDF files. Netflow data can be described by the number of bytes and packets sent and received globally and for some specific port numbers. Other aggregated values can be computed such as the number of contacted IP addresses and ports.

The more discriminating the features are, the more efficient the detection model is. Security administrators should leverage their domain expertise, and rely on previous research works, to extract features that are relevant for their detection target. Besides, they should prefer features that attackers can hardly modify while maintaining the malicious payload to hinder evasion attempts. Finally, they should pay attention to the memory and time complexity of the features computation to meet the *local and online processing* constraint.

Feature extraction is the most manual step of the supervised learning pipeline and it must be implemented for each data type. No ready-to-use solution is available to assist security administrators perform this step. Some research works focus on automatic feature generation [13, 20], but they have not yet met the constraints of detection systems [4].

3.2 Data Annotation

Public Annotated Datasets. Some annotated datasets related to computer security are public, but they must be handled cautiously. First of all, they may not be consistent with the desired detection target, or outdated. Besides, they may not be representative of the data encountered in the deployment environment and therefore lead to *training bias*. The concept of *training bias* will be further explained in Section 4.3.

In-Situ Training. If no suitable public annotated dataset is available, *in-situ* training is a solution. It consists in asking security administrators to annotate data coming from production environments. This solution offers several advantages over public annotated datasets: 1) the annotations correspond perfectly to the detection target, 2) the data are up-to-date, and 3) the risk of training bias is reduced since the data come from the production environment. However, it is more expensive since it involves manual annotations by security administrators.

Active Learning. Active learning strategies [17] have been introduced in the machine learning community to reduce human effort in annotation projects. They ask annotators to annotate only the most informative instances to maximize the performance of detection models while minimizing the number of manual annotations.

The computer security community has exploited active learning [2, 11, 21], but they have mostly focused on query strategies and not on their integration in annotation systems. Very few studies and solutions focus on improving user experience in active learning procedures while it is critical to effectively streamline real-world annotation projects [18, 22]. Security administrators do not want to minimize only the number of manual annotations, but the overall time spent annotating.

ILAB: an End-to-End Active Learning System. In order to fill this gap, we have designed and implemented an end-to-end active learning system, ILAB (Interactive Labeling), tailored to computer security experts' needs. It consists in an active learning strategy [5] and a user interface [6] that both aim to effectively reduce the annotation workload.

ILAB is suitable for annotators who may have little knowledge about machine learning, and it can manipulate any data type. By default, the instances to be annotated are represented by the values of their features. However, this visualization may be hard to interpret especially when there are many features. In order to address this issue, ILAB enables security administrators to plug problem-specific visualizations. These visualizations should display the most relevant information from a detection perspective, and they may point to external tools or information to provide some context.

We have validated our design choices with user experiments [6]. We have asked intended end-users, security administrators, to use ILAB on a large unlabeled NetFlow dataset coming from a production environment. These experiments have demonstrated that ILAB is an efficient active learning system that security administrators can deploy in real-world annotation projects. We provide an open-source implementation of ILAB[1] in SecuML [1] to foster comparison in future research works, and to enable security administrators to annotate their own datasets.

3.3 Analysis of the Training Data

Training data play a central role in supervised learning. Their quality (the number of instances, the agreement of the annotations with the detection target, and the discriminating power of the features) impacts directly the performance of resulting detection models.

Before training a supervised detection model, security administrators should examine their training data. They can inspect descriptive statistics such as the minimum, the maximum, the mean and the variance of each feature. Thanks to these analyses, they may detect bugs in the feature extraction. For instance, they may notice that a feature defined as a ratio does not have all its values between 0 and 1. Besides, these statistics allow to identify useless features those variance is null on a given training dataset. Machine learning models trained on this dataset will never leverage these features to make decisions, so there is no need to compute them.

SecuML [1] offers a features analysis module [2] to carry out these checks before training a detection model. It also displays indicators assessing the discriminating power of features individually (chi square test, and mutual information).

4 Training and Evaluation

Once security administrators have gathered a training dataset they can train a supervised detection model. Many libraries dedicated to machine learning (e.g. scikit-learn in python, Mahout or Weka in java, or Vowpal Wabbit in C++) enable to train various model classes and propose several performance evaluation

[1] https://anssi-fr.github.io/SecuML/_build/html/SecuML.ILAB.html
[2] https://anssi-fr.github.io/SecuML/_build/html/SecuML.stats.html

metrics. In this section, we explain how security administrators should pick an appropriate model class, and a relevant evaluation metric. Finally, we present a four-step evaluation protocol that security administrators should carry out to diagnose and fix potential accuracy issues before deployment.

4.1 Model Classes that Suit the Operational Constraints

Neural networks have become so popular that the confusion between deep learning and machine learning is often made. However, neural networks, used in deep learning, are only one supervised model class, with both benefits and drawbacks. There are many others: decision trees, k-nearest neighbors, or logistic regression, to cite just a few examples. In this section, we explain how the operational constraints of detection systems (see Section 2.1) should drive the choice of the supervised model class.

Controllability. All supervised model classes enjoy a high level of controllability. Their decision rules can be updated automatically with both malicious and benign instances.

Local and Online Processing. This is not a difficult requirement to meet since the training phase of supervised models is usually time-consuming but not the predictions. The training phase is thus performed offline, and once the model has been trained, its application to new data is usually extremely fast.

Nevertheless, lazy learners, such as k-nearest neighbors, should be avoided in the context of detection systems. These models have no training phase, and they therefore need all the training data during the prediction phase. As a result, the temporal complexity of the prediction phase is too high and it increases over time when new instances are used to update the detection model.

To sum up, most model classes meet the online processing constraint. Only lazy learners, such as k-nearest neighbors, should be avoided.

Transparency. Machine learning based detection models are reputed to be black-box methods among the computer security community. Nonetheless, it is crucial to make them more transparent to deploy them successfully in operational detection systems [16]. We detail here three types of model classes that can be explained.

k-Nearest Neighbors. The predictions of k-nearest neighbors are rather interpretable: the k-nearest neighbors can be displayed to security operators to explain why an alert has been triggered. However, the detection model cannot be described as a whole and it does not suit the online processing constraint.

Linear Models. Linear models, such as logistic regression or SVM, meet perfectly the transparency constraint. The coefficients associated with each feature allow to understand their behavior. The greater the absolute value of the coefficient of a feature is, the more the feature influences the prediction. Features with a zero coefficient have no influence on predictions. Features with negative coefficients point out benign instances while features with positive coefficients point out the malicious ones.

Tree-Based Models. Decision trees are transparent models. If the decision tree does not contain too many nodes, security administrators can understand its global behavior, and security operators can interpret individual predictions with paths in trees.

Tree-based ensemble methods (e.g. random forests) are more complex than decision trees, but they are still rather transparent. Their overall behavior can be described by features' importance. It provides a highly compressed, global insight into the model behavior, but individual predictions are hard to interpret.

Computer security experts usually appreciate linear models and decision trees because they can make an analogy with expert systems. Their main advantage over expert systems is their controllability. The weights associated with each feature, or the decision rules in nodes, are not set manually with expert knowledge, but automatically from annotated data. As a result, the weights, or the decision rules, can be swiftly updated to follow threat evolution.

If we focus only on interpretable models, there are only a few options left. Since transparency matters in many application domains, model-agnostic interpretability methods have been introduced in the machine learning community [14]. These methods intend to separate explanations from machine learning models, in order to explain any model class, even the most complex ones.

Robustness. Attackers can craft adversarial examples that evade detection while maintaining the same malicious payload [8]. They usually make slight perturbations to their attack to cross the decision boundary.

Mathematical optimization [7] is a prominent approach to craft adversarial examples that has been broadly applied to image recognition. A famous illustration is a panda that is recognized as a gibbon with high confidence, if some random noise, imperceptible to humans, is added to the picture [9]. These evasion techniques may cast doubt on the usefulness of machine learning models in detection systems if attackers can so easily mislead them.

We want to point out that it is much harder to generate adversarial examples with these mathematical techniques in the context of threat detection. First of all, they do not directly manipulate real-world objects (e.g. PDF files, Android applications, event logs) but numerical vectors in the feature space. As a result, it is necessary to invert the feature mapping, i.e. to generate a real-world object from a feature vector, to get ready-to-use adversarial examples. It is particularly easy for image recognition since the features correspond directly to the pixel values, but it is often much more complex for threat detection. Besides, not all adverse perturbations are valid: they must not corrupt the data format and they must maintain the malicious payload. Some research works have focused on generating adversarial examples for PDF files [7] and Android applications [12], but these attacks work only for specific cases where the feature mapping can be easily inverted.

Even if adversarial examples are difficult to craft in the context of threat detection, it would be great to pick a model class particularly robust. Unfortunately, adversarial techniques are generic, they are not tailored to any specific model class, and there is still no consensus about the best method to make detection models robust against evasion attempts [10].

Effectiveness. There is no way to determine which model classes are the most effective since it depends deeply on the data. For instance, linear models are simple: they require benign and malicious instances to be linearly separable to work properly. If the data are not linearly separable, more complex models such as quadratic models, tree-based models, or neural networks must be trained.

The effectiveness of a model class is data-dependent, the model selection is thus performed empirically (see Section 4.3).

To sum up, the *controllability* and *online processing* constraints hardly restrict the set of model classes that suit detection systems, and there is still no consensus about the *robustness* constraint. *Transparency* and *effectiveness* are thus the most important criteria.

We advise to begin by training a linear model (e.g. logistic regression or SVM) that can be easily interpreted by security administrators and operators. If a linear model does not perform well-enough, then we may move ahead to more flexible model classes (e.g. tree-based models, neural networks). In Section 4.3, we explain how to decide whether a more flexible model class is required.

4.2 Relevant Detection Performance Metrics

False Negatives and Positives The confusion matrix (see Figure 1) allows to assess properly the performance of a detection method. It takes into account the two types of errors that can occur: *false negatives*, i.e. malicious instances which have not been detected, and *false positives*, i.e. benign instances which have triggered a false alarm.

The confusion matrix allows to express performance metrics such as the *detection rate* and the *false alarm rate*. There is a consensus on the definition of the *detection rate*, also called *True Positive Rate (TPR)*:

$$\text{TPR} = \frac{TP}{TP + FN}. \tag{1}$$

On the contrary, the *false alarm rate* is not clearly defined.

		Predicted label	
		`Malicious`	`Benign`
True label	`Malicious`	True Positive (TP)	False Negative (FN)
	`Benign`	False Positive (FP)	True Negative (TN)

True	the prediction is true (predicted label = true label)
False	the prediction is wrong (predicted label ≠ true label)
Positive	the prediction is `Malicious`
Negative	the prediction is `Benign`

Fig. 1: Explanation of the Confusion Matrix.

How to Compute the False Alarm Rate ? There are two competing definitions for the *false alarm rate*: the *False Positive Rate (FPR)*, the most commonly used, and the *False Discovery Rate (FDR)*:

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

$$\mathrm{FDR} = \frac{FP}{FP + TP}. \tag{3}$$

The FPR is the proportion of benign instances that have triggered a false alarm, while the FDR measures the proportion of the alerts that are irrelevant.

The FDR makes more sense than the FPR from an operational point of view. First of all, it can be computed in operational environments in contrast to the FPR (the number of benign instances, $FP + TN$, is unknown in practice). Moreover, the FDR reveals the proportion of security operators' time wasted analyzing meaningless alerts, while the FPR has no tangible interpretation. Finally, the FPR can be highly misleading when the proportion of malicious instances is extremely low because of the base-rate fallacy [3]. Let's take an example. We consider 10,000 instances (10 malicious and 9990 benign) and we suppose that there are 10 false positives and 2 false negatives ($FP = 10$, $FN = 2$, $TP = 8$, $TN = 9980$). In this case, the FPR seems negligible ($FPR = 0.1\%$) while security operators spend more than half of their reviewing time analyzing meaningless alerts ($FDR = 55\%$).

For all these reasons, the False Discovery Rate (FDR) should be preferred over the False Positive Rate (FPR) even if it is still less prevalent in practice.

ROC and FDR-TPR Curves. The ROC (or FPR-TPR) curve represents the detection rate (TPR) according to the FPR for various values of the detection threshold (see Figure 2). This curve is interesting to check whether machine learning has learned something from the data, i.e. the ROC curve is away from those of a random generator.

However, the FDR-TPR curve must be preferred to set the value of the detection threshold according to the desired detection rate or the tolerated FDR, for the same reason that the FDR should be preferred over the FPR.

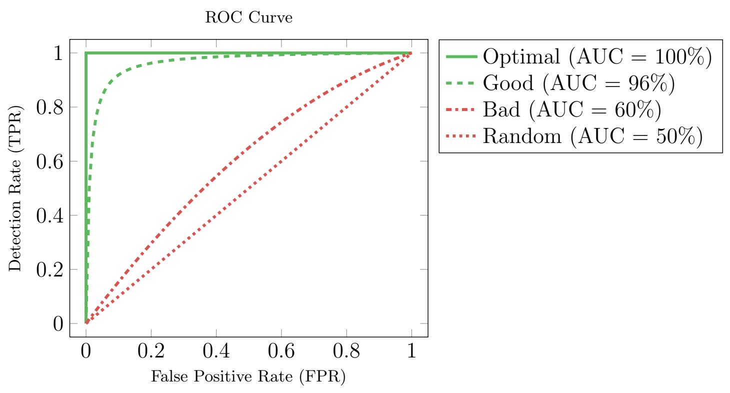


Fig. 2: Explanation of the ROC Curve.

4.3 Evaluation Protocol

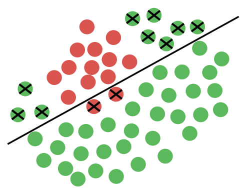
Training and Validation Datasets. In general, security administrators have access to a single annotated dataset to set up their detection model. They must therefore split it into a training and a validation dataset. The simplest way to split it up is to select some instances randomly for training, and to keep the remaining instances for validation.

If the instances are timestamped, i.e. the times of first appearance are known, then a better evaluation process can be devised. All the instances occurring before a given cutoff timestamp can constitute the training dataset, while the remaining instances are retained for validation purposes. This temporally consistent evaluation process better assesses the performance of detection models since in practice future instances are never available at training time.

a. How to Diagnose and Handle Underfitting ? Supervised detection models should capture the relevant relations between the input data and the labels we want to predict. *Underfitting* occurs when the detection model cannot capture the underlying trend of the data. Intuitively, the model is not complex enough to fit the data properly.

Diagnosis. Underfitting can be diagnosed by evaluating the performance of the detection model on the training dataset. A detection model suffers from underfitting if the ROC curve is close to that of a random generator (see the random curve in Figure 2).

Solution. Security administrators can solve underfitting in two ways: adding discriminating features or training a more complex classification model class. Figure 3a shows a two-dimensional dataset where a linear model is too simple and cannot properly separate malicious from benign instances, while a slightly more complex model, a quadratic model, is perfectly adapted (see Figure 3b).


(a) *Linear Model:* does not fit well the training data.

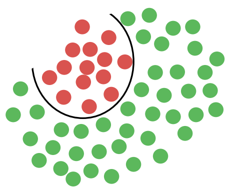

(b) *Quadratic Model:* fits well the training data.

Fig. 3: Illustration of Underfitting.

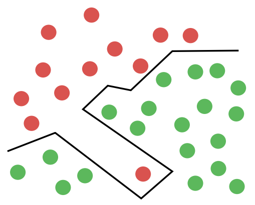
We want to point out that the performance of a detection model on its training data is not a satisfactory assessment of its true performance. Analyzing the training performance is a good way to diagnose underfitting, but it is not enough to ensure that the detection model makes accurate predictions. The purpose of detection models is not to classify training data correctly, but to be able to generalize, i.e. correctly classify data unseen during the training phase. The next section explains how to assess the generalization capabilities of detection models, and how to react if they are not satisfactory.

b. How to Diagnose and Handle Overfitting ? *Overfitting* means that the detection model is too flexible and fits too much the training data. The noise or random fluctuations in the training data are picked up and learned as concepts by the model. However, these concepts may not apply to new data and negatively impact the generalization capabilities of the model.

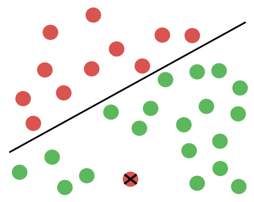
Diagnosis. Overfitting occurs when the detection model predicts accurately the label of training data, but fails to predict correctly the label of data unseen during the training phase. It can be diagnosed by analyzing the model performance on an independent validation set. If the detection model performs well on the training dataset, but poorly on the validation set, then it suffers from overfitting.

Solution. Overfitting is usually caused by a too complex model class that has too much flexibility to learn a decision boundary. When the detection model is too complex (see Figures 4a and 4c), it predicts perfectly the label of the training instances (see Figure 4a), but it makes many prediction errors on the validation dataset (see Figure 4c). Indeed, the complex model fits perfectly the training data, but it has weak generalization capabilities. On the other hand, a simpler model (see Figures 4b and 4d) is able to avoid the outlier to generalize much better on unseen data.

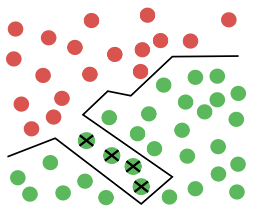
A detection model can make prediction errors on training data, but it must generalize well to unseen data. Training data may contain outliers or annotation errors, that the training algorithm should not take into account when building the model to improve its generalization capabilities.



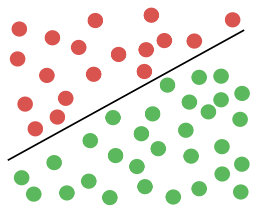
(a) *Complex Model:* no training errors.



(b) *Simple Model:* a single training error on an outlier.



(c) *Complex Model:* many classification errors on the validation dataset.



(d) *Simple Model:* no classification errors on the validation dataset.

Fig. 4: Illustration of Overfitting. The top row represents the training dataset while the bottom line represents the validation dataset.

To sum up, it is critical to strike an appropriate balance to avoid both underfitting and overfitting. The detection model must be neither too simple to avoid underfitting nor too complex to avoid overfitting. To check whether a sound compromise has been reached, security administrators must assess the performance of a detection model both on its training data, and on an independent validation dataset. However, these two assessment steps may not be sufficient to ensure successful deployments since they are not able to identify potential training biases.

c. How to Diagnose and Handle Training Biases ? There is a *training bias* when the detection model performs well on the training and validation datasets, but poorly in production. This issue occurs when the training dataset is not representative of the data encountered in the deployment environment.

Diagnosis. When detection models are interpretable (see Section 4.1), security administrators can inspect the features having the greatest impact on decision-making, and decide for each of them whether they are consistent with the detection target, or they reveal a training bias.

For instance, if a detection model for malicious PDF files relies mostly on the feature *contains JavaScript*, we examine the values of this feature in the training dataset. If all the malicious instances contain JavaScript, while none of the benign PDF file does, then the training dataset is stereotyped and leads to a training bias. In practice, benign PDF files may contain JavaScript, and the model is very likely to trigger false alarms for these files.

Solution. The detection model can be deployed and updated with the false and true positives analyzed by security operators to reduce the training bias. The false positives are likely to be ambivalent instances (e.g. benign PDF files with JavaScript) that will make the detection model smarter and more subtle. However, this method has a major flaw: some false negatives may never be detected because security operators inspect only the alerts. The best solution to avoid training bias is to perform *in-situ* training, i.e. security administrators annotate data directly from the production environment (see Section 3.2).

Training biases explain why outstanding results presented in academic papers are often hard to reproduce in operational detection systems. The experiments are usually carried out on public annotated datasets that may be biased. In practice, annotating a dataset *in-situ* may be costly, but it significantly reduces the risk of training bias. Finally, we want to emphasize that the model transparency is a real asset to diagnose and fix potential training biases.

d. How to Assess the Robustness Against Adversarial Examples ? Papernot et al. have created cleverhans [15], an open-source library for benchmarking the vulnerability of machine learning models to adversarial examples. However, this library cannot benchmark computer security detection models directly since it does not manipulate real-world objects (e.g. PDF files, Android applications, event logs) but numerical vectors in the feature space. No generic solution assesses threat detection models in adversarial settings, specific protocols must be devised for each data type [7, 12].

The robustness of an interpretable model can be assessed partly by examining the most prominent features. Security administrators must assess whether the value of these features can be easily modified by attackers while maintaining the malicious payload. Using the same example as above about PDF files, the feature *contains JavaScript* cannot be modified by attackers if the malicious payload is contained in a JavaScript object. On the contrary, if the feature *number of images* is associated to an extremely negative weight (i.e. the benign PDF files tend to have more images than the malicious ones), then attackers can easily add images to their malicious PDF file to bypass detection.

The first three evaluation steps, checking for underfitting, overfitting, and training bias, must be validated to get a ready-to-deploy detection model, while the last one, assessing the robustness against adversarial examples, is noncompulsory. Indeed, an unrobust model, relying mostly on features that can be easily controlled by attackers, may improve the detection capacities compared to already deployed detection techniques. However, robustness matters to detect more advanced attacks.

4.4 Training and Diagnosing Detection Models with DIADEM

DIADEM (DIAgnosis of DEtection Models) assists security administrators with training and evaluating detection models according to the protocol presented in Section 4.3. The main advantages of DIADEM over traditional machine learning libraries are two-fold: 1) it hides some of the machine learning machinery to let security administrators focus mainly on detection, and 2) it offers a graphical user interface to diagnose potential accuracy issues and to find solutions to address them.

DIADEM displays the performance of the detection model both on the training and validation datasets to diagnose both underfitting and overfitting. Besides, it displays information about the global behavior of transparent detection models. Security administrators can review the most influential features and analyze their descriptive statistics through the features analysis module (see Section 3.3). This way they can understand why a feature has a significant impact on decision-making, and they may diagnose biases in the training dataset.

We provide an open-source implementation of DIADEM [3] in SecuML [1] to allow security administrators to build and diagnose their own detection models.

5 Conclusion

Machine learning is often presented as a silver bullet for detection systems: it builds detection rules automatically from data, and its generalization capabilities improve the detection of yet unknown threats. In practice, security administrators must perform each step of the machine learning pipeline carefully to ensure successful deployments, and they should bear in mind three crucial points.

Be cautious with public annotated datasets. Public annotated datasets are appealing to avoid annotating data manually. However, they must be handled cautiously since they may not be consistent with the desired detection target, and they may be biased.

Neural networks are not always the best model class. There is no need to train a neural network if a linear model is complex enough to fit the data. Neural networks will only increase the risk of overfitting, and training biases will be harder to diagnose.

Evaluation must no be reduced to figures. Examining numerical performance measures such as false alarm and detection rates is not enough to ensure successful deployments. Security administrators must conduct a more in-depth analysis of the model behavior to diagnose potential training biases.

References

1. SecuML. `https://github.com/ANSSI-FR/SecuML` (2018)

[3] `https://anssi-fr.github.io/SecuML/_build/html/SecuML.DIADEM.html`

2. Almgren, M., Jonsson, E.: Using active learning in intrusion detection. In: CSFW. pp. 88–98 (2004)
3. Axelsson, S.: The base-rate fallacy and the difficulty of intrusion detection. ACM Transactions on Information and System Security (TISSEC) 3(3), 186–205 (2000)
4. Beaugnon, A.: Expert-in-the-loop Supervised Learning for Computer Security Detection Systems. Ph.D. thesis, École Normale Supérieure (2018)
5. Beaugnon, A., Chifflier, P., Bach, F.: Ilab: An interactive labelling strategy for intrusion detection. In: RAID (2017)
6. Beaugnon, A., Chifflier, P., Bach, F.: End-to-end active learning for computer security experts. In: AICS (2018)
7. Biggio, B., Corona, I., Maiorca, D., Nelson, B., Šrndić, N., Laskov, P., Giacinto, G., Roli, F.: Evasion attacks against machine learning at test time. In: Machine Learning and Knowledge Discovery in Databases, pp. 387–402. Springer (2013)
8. Biggio, B., Roli, F.: Wild patterns: Ten years after the rise of adversarial machine learning. arXiv preprint arXiv:1712.03141 (2017)
9. Goodfellow, I.J., Papernot, N.: Breaking things is easy. `http://www.cleverhans.io/security/privacy/ml/2016/12/16/breaking-things-is-easy.html` (2017)
10. Goodfellow, I.J., Papernot, N.: Is attacking machine learning easier than defending it? `http://www.cleverhans.io/security/privacy/ml/2017/02/15/why-attacking-machine-learning-is-easier-than-defending-it.html` (2017)
11. Görnitz, N., Kloft, M.M., Rieck, K., Brefeld, U.: Toward supervised anomaly detection. JAIR (2013)
12. Grosse, K., Papernot, N., Manoharan, P., Backes, M., McDaniel, P.: Adversarial examples for malware detection. In: European Symposium on Research in Computer Security. pp. 62–79. Springer (2017)
13. Kanter, J.M., Veeramachaneni, K.: Deep feature synthesis: Towards automating data science endeavors. In: DSAA. pp. 1–10. IEEE (2015)
14. Molnar, C.: Interprtable machine learning: A guide for making black box models explainable. `https://christophm.github.io/interpretable-ml-book/` (2018)
15. Papernot, N., Carlini, N., Goodfellow, I., Feinman, R., Faghri, F., Matyasko, A., Hambardzumyan, K., Juang, Y.L., Kurakin, A., Sheatsley, R., et al.: cleverhans v2. 0.0: an adversarial machine learning library. arXiv preprint arXiv:1610.00768 (2016)
16. Rieck, K.: Computer security and machine learning: Worst enemies or best friends? In: SysSec. pp. 107–110 (2011)
17. Settles, B.: Active learning literature survey. University of Wisconsin, Madison 52(55-66), 11 (2010)
18. Settles, B.: From theories to queries: Active learning in practice. JMLR 16, 1–18 (2011)
19. Sommer, R., Paxson, V.: Outside the closed world: On using machine learning for network intrusion detection. In: S&P. pp. 305–316 (2010)
20. Šrndić, N., Laskov, P.: Hidost: a static machine-learning-based detector of malicious files. EURASIP Journal on Information Security 2016(1), 22 (2016)
21. Stokes, J.W., Platt, J.C., Kravis, J., Shilman, M.: Aladin: Active learning of anomalies to detect intrusions. Technical Report. Microsoft Network Security Redmond, WA (2008)
22. Wagstaff, K.L.: Machine learning that matters. In: ICML. pp. 529–536 (2012)