THÈSE DE DOCTORAT

de l'Université de recherche Paris Sciences et Lettres
PSL Research University

Préparée à l'École normale supérieure

Expert-in-the-Loop Supervised Learning for Computer Security Detection Systems

Apprentissage supervisé et systèmes de détection :
une approche de bout-en-bout impliquant les experts en sécurité

Ecole doctorale n°386

ÉCOLE DOCTORALE DE SCIENCES MATHÉMATIQUES DE PARIS CENTRE

Spécialité INFORMATIQUE

COMPOSITION DU JURY :

M. DEBAR Hervé
Télécom Sud Paris, Président du jury

M. AMINI Massih-Reza
Université Grenoble Alpes, Rapporteur

Mme. VIET TRIEM TONG Valérie
CentraleSupelec, Rapporteur

M. BACH Francis
INRIA Paris, DI ENS, Directeur de thèse

M. CHIFFLIER Pierre
ANSSI, Encadrant de thèse

M. GAILLARD Pierre
INRIA Paris, Examinateur

M. MORIN Benjamin
ANSSI, Invité

Soutenue par Anaël BEAUGNON
le 25 juin 2018

Dirigée par **Francis BACH**
et **Pierre CHIFFLIER**

ENS ÉCOLE NORMALE SUPÉRIEURE | PSL RESEARCH UNIVERSITY PARIS

Abstract

Supervised detection models can be deployed in detection systems, as an adjunct to traditional detection techniques, to strengthen detection. Supervised learning has been successfully applied to various computer security detection problems: Android applications, PDF, or portable executable files to mention only the most obvious examples. Despite these encouraging results, there remain some significant barriers to the widespread deployment of machine learning in operational detection systems.

The standard supervised learning pipeline consists of data annotation, feature extraction, training and evaluation. Security experts must carry out all these steps to set up supervised detection models ready for deployment. In this thesis, we adopt an end-to-end approach. We work on the whole machine learning pipeline with security experts as its core since it is crucial to pursue real-world impact.

First of all, security experts may have little knowledge about machine learning. They may therefore have difficulty taking full advantage of this data analysis technique in their detection systems.

This thesis provides methodological guidance to help security experts build supervised detection models that suit their operational constraints. Moreover, we design and implement DIADEM, an interactive visualization tool that helps security experts apply the methodology set out. DIADEM deals with the machine learning machinery to let security experts focus mainly on detection.

Besides, most research works assume that a representative annotated dataset is available for training while such datasets are particularly expensive to build in computer security. Active learning has been introduced to reduce expert effort in annotation projects. However, it usually focuses on minimizing only the number of manual annotations, while security experts would rather minimize the overall time spent annotating. Moreover, user experience is often overlooked while active learning is an interactive procedure that should ensure a good expert-model interaction.

This thesis proposes a solution to effectively reduce the labeling cost in computer security annotation projects. We design and implement an end-to-end active learning system, ILAB, tailored to security experts needs. Our user experiments on a real-world annotation project demonstrate that security experts can gather an annotated dataset with a low workload thanks to ILAB.

Finally, feature extraction is usually implemented manually for each data type. Nonetheless, detection systems process many data types and designing a feature extraction method for each of them is tedious. Automatic feature generation would significantly ease, and thus foster, the deployment of machine learning in detection systems.

In this thesis, we define the constraints that such methods should meet to be effective in building detection models. We compare three state-of-the-art methods based on these criteria, and we point out some avenues of research to better tailor these techniques to computer security experts needs.

Remerciements

En premier lieu, je souhaite remercier mes encadrants, Pierre Chifflier et Francis Bach, pour leurs précieux conseils, et leur soutien tout au long de ce doctorat. Je souhaite également remercier Valérie Viet Triem Tong et Massih-Reza Amini qui m'ont fait l'honneur de rapporter cette thèse, ainsi que l'ensemble des membres du jury.

Stéphane, Daniel et Nizar vous m'avez fait découvrir le monde de la recherche au cours de mon stage chez Orange Labs. Le sujet était passionnant et m'a donné envie de poursuivre en thèse.

Un grand merci à Benjamin, Yohann, José, Cyril et Vincent pour avoir rendu cette thèse possible au sein de l'ANSSI. J'ai eu une grande liberté dans le choix de mes sujets de recherche, et j'ai pu appliquer les résultats de mes travaux avec le COSSI. Je souhaite remercier tout particulièrement Pierre Ch. qui a été d'une grande aide autant sur le plan technique qu'humain.

Au cours de cette thèse, j'ai rencontré des difficultés techniques, mais aussi relationnelles pour le moins inattendues. Un grand merci à toutes les personnes qui m'ont soutenue dans ces moments difficiles. Il m'est impossible de toutes les nommer. Je souhaite néanmoins remercier nommément Aurélie, François, Julie, Marion, Olivier, Pierre Ch. et Pierre D. et Thomas. Votre soutien a été essentiel pour que je puisse achever ma thèse dans de bonnes conditions.

Je tiens aussi à remercier les beta-testeurs de SecuML, Julie, Antoine, Pierre et Corentin, ainsi que les participants des expériences utilisateur, Adrien, Antoine, Elie et Matthieu. Vos retours ont été essentiels pour rendre SecuML plus adapté aux besoins des experts en sécurité.

Merci Pierre Co. pour nos discussions très enrichissantes sur la génération automatique d'attributs. Merci Johan pour tes nombreuses relectures, très méticuleuses.

Thomas, c'était une vraie chance de pouvoir partager avec toi les moments de doute suite aux rejets des premières soumissions, et les joies des papiers acceptés !

Je souhaite remercier ma famille et ma belle-famille pour leur soutien tout au long de mes études. Mes derniers remerciements vont à Ulysse. Grâce à toi j'ai découvert l'informatique, l'ANSSI, et j'ai eu le courage de me lancer dans une thèse ! Ta patience, tes encouragements, et ton soutien sans faille ont fortement contribué à son aboutissement.

Contents

Chapter 1

Introduction

1.1 Computer Security Detection Systems

Computer security incidents can significantly disrupt governments, companies, and end-users in possession of computer platforms (e.g. desktop computers, laptops, tablets, smartphones, or other connected objects such as smart watches, or fitness trackers). As regards end-users, attackers may steal sensitive information (e.g. credit card numbers, compromising pictures), watch end-users through their webcam, or display unwanted adds. The consequences can be more serious for governments and companies. Attackers can exploit security incidents to commit industrial, military, or political espionage. They can also significantly undermine the functioning of organizations. For instance, ransomware is a type of malicious software that encrypts victims' data, making them inaccessible, and requests a ransom payment to decrypt them. Currently, most organizations rely entirely on their information system which makes them even more sensitive to security incidents.

Computer security detection systems are crucial to avoid or limit the impact of security incidents. They monitor a network, or a system to identify potential threats. Detection can take place at two levels: 1) detecting attempted attacks, and 2) detecting successful attacks.

Detecting attempted attacks involves for example identifying malicious files attached to email messages (e.g. PDF files, or Windows Office documents), or phishing email messages. In this case, there is no security incident as long as the email recipient has not opened the attached malicious file, or been lured by the phishing content.

Detecting successful attacks includes identifying data exfiltration, distributed denial of service, and malware communications. In this situation, there is already a security incident, but detection is still critical to act quickly and reduce its negative impact.

Detection systems involve several detection methods based on diverse techniques operating in parallel. The suspicious activities identified by the detection methods are generally reported to a security information and event management (SIEM) system. Then, the SIEM system combines the outputs from the multiple detection methods. It relies on aggregation and correlation techniques to expose a more condensed views of the security issues identified by all the detection methods [43, 38, 140, 31]. In this thesis, we focus on detection methods. Correlation and aggregation techniques are out of scope.

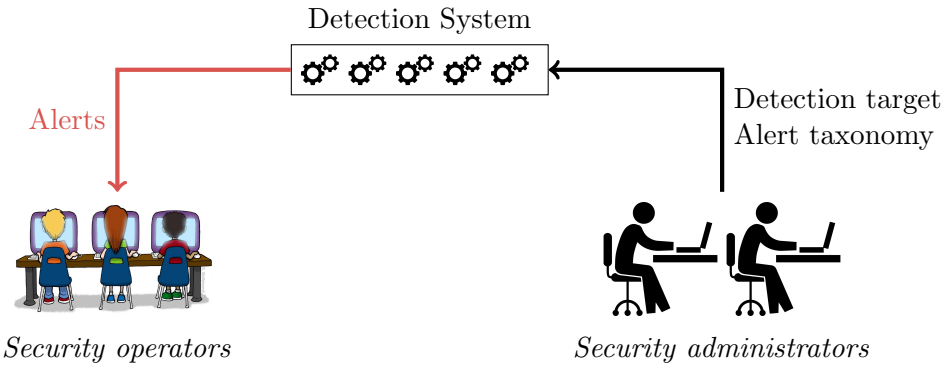


Figure 1.1: Security Operators and Security Administrators.

We define two roles that operate in security operation centers (SOC): *security administrators* and *security operators* (see Figure 1.1).

Security Administrators. Security administrators are responsible for setting up detection methods. First of all, they define the detection target, i.e. in which circumstances an alert should be triggered. Detection targets depend on deployment environments since they have their own security policy which may be more or less permissive. Moreover, security administrators usually set up an alert taxonomy. It establishes 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. Finally, they make detection methods evolve over time to follow threat evolution. They may also need to update the detection target and the alert taxonomy when new threats emerge.

Security Operators. Once the detection methods have been deployed, security operators exploit the alerts. They must decide whether alerts have been triggered rightly to discard false alarms. Then, they examine the actual alerts carefully in order to take the appropriate actions to avoid or handle potential security incidents.

Security operators need contextual information to conduct their analyses. For instance, the tags of the alert taxonomy can categorize the alerts. Since the alerts sharing the same tag are likely to have significant similarities, security operators can take advantage of the previous analyses associated to the same tag.

Our definitions of security administrators and operators may encompass several roles encountered in security operation centers. We rely on these two roles throughout the thesis, since they are precise enough for our needs.

1.2 Operational Constraints

Detection methods must meet operational constraints to ensure their operability in detection systems. We introduce the main constraints mostly synthesized from [127] and [111].

1.2.1 Local and Online Processing

The sensitivity of the data processed by detection systems, and regulatory requirements, hinder the use of cloud computing services. Detection methods should be processed locally.

In addition, detection methods must be integrated into security operation centers that analyze streaming data that cannot be stored due to volumetry and privacy. As a result, detection methods must be suitable for streaming data. They must enjoy a low time complexity to be executed in near real time.

In conclusion, detection methods should not require too much computation to enable local online processing.

1.2.2 Effectiveness

Detection methods can make two kinds of errors: *false negatives*, i.e. malicious events which have not been detected, and *false positives*, i.e. benign events which have generated a false alarm. Both types of errors are extremely costly in computer security.

One the one hand, a false positive requires a security operator to spend time analyzing the alert to eventually determine that it reflects a benign activity. If the false positive rate is too high, meaningless alarms will overwhelm security operators, and they will have less time to analyze the significant ones. As argued by Axelsson, even a very small false positive rate can quickly make a detection system unusable [12]. On the other hand, false negatives have the potential to cause serious damage to the defended network. Even a single compromised node can seriously undermine the integrity of the whole network.

Security administrators must choose the trade off between false positives and negatives, keeping in mind that the false positive rate must remain low enough to ensure the operability of the detection system.

1.2.3 Transparency

Security operators need information to discard false alarms, and to figure out the origin of actual alarms. A binary answer indicating only whether an alert is triggered or not is not satisfactory. Detection methods must provide information about the cause of alerts. Relevant contextual information can significantly streamline their exploitation, and thus increase the efficiency of detection systems. For instance, an alert taxonomy set up by security administrators can tag the alerts.

Moreover, security administrators will not deploy black-box detection methods. They need to trust detection methods before their deployment, they want to understand how they work. To that end, security administrators can inspect individual detection results, and infer the overall behavior. It is even better if they can grasp how detection methods work as a whole with higher-level descriptions.

Therefore, detection methods must be transparent to suit both security administrators and operators. The generated alerts must be understandable to ease their exploitation by security operators. Besides, we recommend detection methods to be understandable as a whole for security administrators.

1.2.4 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.

False positives must be fixed to avoid security operators from keeping analyzing always the same false alarms. Besides, if undetected malicious behaviors are identified by other means, the detection method must be patched to improve its performance in the future. For instance, security operation centers can have access to the knowledge base of an intern or commercial computer security incident response team (CSIRT). CSIRTs monitor technological developments related to security: they analyze new attacks, new malware, and the latest vulnerabilities. Their analyses should be leveraged to improve detection methods.

Basically, detection methods should not make mistakes on known examples. They should be adaptable to trigger alerts on known malicious behaviors, and not generate false alerts on known benign behaviors. If they make errors on known examples, security administrators must be able to revise detection methods swiftly.

1.2.5 Robustness

Detection methods are deployed in adversarial environments where attackers are willing to circumvent them. Malicious behaviors constantly evolve as adversaries attempt to deceive detectors. Detection methods must therefore be designed to resist evasion attempts, and to detect yet unknown threats as soon as possible.

For instance, *polymorphism* [34, 134, 48] and *mimicry* [144, 76, 83] are usual evasion techniques. Polymorphism modifies slightly attacks to create numerous variants. Every variant looks different, for the purpose of evading detection, and yet carries out the same malicious payload. As for mimicry evasion techniques, they attempt to make attacks look legitimate while maintaining the malicious payload.

In the next section, we present the main detection techniques deployed in detection systems, and we compare them according to the constraints stated in this section. We want to emphasize that the objective is not to determine which approach is best. These diverse detection techniques are complementary, and are often deployed in parallel to strengthen detection. The objective is rather to point out the pros and cons of each approach, and to explain why we focus on supervised learning in this thesis.

1.3 Detection Techniques

Since Anderson's seminal work [9] on computer security detection, many detection techniques have been designed. They are traditionally divided into two categories [42]: *misuse* and *anomaly* detection. *Misuse detection* relies on detection rules characterizing malicious activities while *anomaly detection* models the legitimate behavior and triggers alerts each time the behavior deviates too much from the baseline.

Additionally, some detection techniques rely on *supervised learning* that characterizes both malicious and benign behaviors. The traditional taxonomy above-mentioned does not allow to categorize these techniques consistently. Some consider that they belong to misuse detection [80] while others think they are a subcategory of anomaly detection techniques [56].

In line with Axelson [13], we decide to consider that supervised learning forms its own category of detection techniques. This distinction is relevant for the purposes of comparing detection techniques according to the operational constraints of detection systems.

In brief, we regroup detection techniques into three categories according to the type of behaviors they characterize. *Misuse detection* characterizes malicious activities (see Section 1.3.1), and *anomaly detection* defines a model of legitimate behaviors (see Section 1.3.2), while *supervised learning* models both legitimate and malicious behaviors (see Section 1.3.3).

1.3.1 Misuse Detection

Detection systems traditionally rely on misuse detection techniques: patterns or sets of detection rules characterizing malicious behaviors. Security administrators build them manually based on an in-depth analysis of malicious events. Signatures are the best known misuse detection techniques, but there are more sophisticated methods [42].

Highly Transparent. This approach is prevalent in practice thanks in part to its high level of transparency. Security administrators trust detection methods based on expert knowledge, and it is straightforward for security operators to understand why an alert has been triggered.

Rather Controllable. Moreover, misuse detection methods are rather controllable. False positives can be avoided by making detection rules more precise. False negatives can be analyzed by security administrators to make a detection rule more generic, or to create a new one. Thus, security administrators can update the detection rules to avoid both false positives and negatives. Controllability is, nevertheless, not perfect since the updates involve manual work that can be time-consuming. Analyzing a malicious behavior to characterize it can involve sophisticated methods of reverse engineering.

Prone to Polymorphism Attacks. Misuse detection is based on the knowledge of past attacks or known vulnerabilities. This approach is efficient against threats that have already been identified and for which a detection rule has been generated. However, it fails to generalize across evolving threats and thereby enables attackers to evade detection. It often cannot detect new threats, and slight changes, such as polymorphism, may circumvent detection rules. Security administrators need to update the knowledge base manually periodically to follow threat evolution.

1.3.2 Anomaly or Behavior-Based Detection

Anomaly detection methods have been introduced by Denning [46] to detect yet unknown threats. These methods proceed in two steps: 1) a set of rules defining the legitimate behavior are designed, and 2) the rules are applied to detect deviant behaviors automatically.

Anomaly detection is appealing because it may detect yet unknown threats. These detection techniques do not have any characterization of malicious behavior and are therefore more likely to detect new threats than misuse detection techniques.

Prone to Mimicry Attacks. Anomaly detection is not prone to polymorphism attacks, but to mimicry techniques. Attackers can make their attack look like the legitimate behavior while keeping the malicious payload to avoid detection.

In this section, we present two types of anomaly detection methods: *expert systems*, and *unsupervised learning*. They are distinguished by the way the first step, defining the legitimate behavior, is performed: manually or automatically. In the case of expert systems, security administrators build rules defining the legitimate behavior manually thanks to their domain expertise. On the contrary, unsupervised learning characterizes the legitimate behavior automatically from legitimate data gathered by security administrators. There are many other anomaly detection techniques that are not detailed in this thesis. The reader may refer to [53] for more information about anomaly detection.

Anomaly-Based Expert Systems

Expert systems are knowledge-based detection techniques: they consist of sets of rules manually defined. They can be misuse or anomaly detection methods depending on the type of behavior they characterize. Here we consider anomaly-based expert systems that characterize the legitimate behavior.

Security administrators establish a set of rules to define the model of legitimate behavior. Once the model has been designed, it can be applied to detect anomalous behaviors automatically.

Rather Transparent. Anomaly-based expert systems have almost the same level of transparency as misuse detection techniques. Security administrators understand without difficulty how detection works within expert systems since they have built the rules themselves. Nonetheless, the generated alerts can be more difficult to interpret than misuse detection rules, since they have not been triggered by a specific malicious behavior.

Rather Controllable. Anomaly-based expert systems are more likely to detect yet unknown threats than misuse detection, but they have the same drawback in terms of controllability. Indeed, security administrators can update the model of legitimate behavior to avoid both false positives and negatives, but the update is performed manually, and can thus be time-consuming.

Unsupervised learning is a solution to make anomaly detection more controllable: it automates the construction of the legitimate behavior model.

Unsupervised Learning

Unsupervised learning is a kind of machine learning that requires only non-malicious data to build anomaly detection models. The model of legitimate behavior is induced automatically from the benign data provided initially by security administrators. Then, the model triggers an alert each time an event differs too much from the legitimate behavior.

At first sight, unsupervised anomaly detection models are attractive for detection systems. They are more likely to detect yet unknown threats than misuse detection. Besides, they seem more controllable than expert systems since they automate the generation of the legitimate behavior model. Finally, their deployment is frequently claimed to be straightforward: it requires only a benign dataset which does not contain any malicious event.

High Risk of False Positives. Anomaly detection models rely on two strong assumptions: 1) the dataset is attack-free, and 2) all anomalous events are malicious [55]. However, these assumptions do not usually hold in practice which is likely to damage the performance of anomaly detection models.

Acquiring an attack-free dataset is not easy since there is no simple way to ensure the absence of malicious events. If the dataset that is assumed to be clean contains malicious events, it may mislead the detection model and prevent some threats from being detected.

Besides, anomaly detection models trigger alerts for anomalous events which are not necessarily malicious. For instance, an anomalous transmission/reception ratio on HTTPS may be evidence of a data exfiltration, but may also be caused by certain social networks ; popular web sites can generate much traffic without being malicious ; and mere configuration errors or policy changes can lead to anomalous behaviors generating false alerts. As a result, anomaly detection methods often suffer from a high false alarm rate and overwhelm security operators with meaningless alerts.

Unsupervised learning may be responsible for the poor reputation of machine learning techniques among security administrators. These techniques are reputed to suffer from high false positive rates that make them inconsistent with operational constraints. Since the training of unsupervised anomaly detection models seems straightforward, many security administrators have tried out these techniques. The resulting detection models have often triggered many false alarms since the two assumptions stated above hardly hold in practice.

Interpretation is not Straightforward. Unsupervised learning is less transparent than expert systems. The automation of the definition of the legitimate behavior comes at the expense of transparency. Indeed, anomaly detection models built with unsupervised learning cannot be interpreted as easily as sets of rules built with expert knowledge. Anomaly detection models can be interpreted, but it requires additional work.

Rather Controllable. Unsupervised anomaly detection models are more controllable than expert systems, since they update automatically the model of legitimate behavior. Nonetheless, they still suffer from a lack of controllability. If some false negatives are identified by other means, they cannot be injected in the model to improve the detection of similar attacks in the future. Security administrators cannot guide these models with malicious examples as their training takes into account only benign events. Only false alarms can be added to the attack-free dataset to update the anomaly detection model.

In conclusion, anomaly detection techniques are attractive to be deployed in detection systems, as an adjunct to misuse detection, since they are more likely to detect yet unknown threats. Anomaly-based expert systems are, nevertheless, as controllable as misuse detection rules: they cannot be updated automatically to correct false positives or negatives. Unsupervised learning is a good solution to deal with the lack of automation of expert systems, but at the expense of effectiveness and transparency. Finally, even if unsupervised learning automates the update of the legitimate behavior model, it still lacks controllability: security administrators cannot guide anomaly detection models with malicious examples. Supervised detection addresses this need for integration of expert knowledge.

1.3.3 Supervised Detection

Supervised learning is another kind of machine learning. It proceeds in two steps as unsupervised anomaly detection: 1) a supervised detection model is trained from annotated data provided by security administrators, and 2) the detection model is applied to detect malicious events automatically. The main difference with unsupervised learning is the content of the training dataset: it contains benign examples, but also malicious ones to guide the detection model. The training algorithm finds automatically the similarities of the examples within each class and the discriminating characteristics to build the detection model.

Highly Controllable. Thanks to supervised learning, security operators can easily improve the detection model from the alerts they analyze. False alarms can rectify the model and avoid generating the same false alarms in the future. True alarms can also be injected in the model to follow threat evolution. Besides, if malicious examples are detected by other means, they can guide the model to improve its detection performance. As a result, security administrators do not hand control of detection systems to automatic models, but they actively supervise them to improve their performance over time. In brief, supervised detection models are highly controllable: they can be updated automatically with both benign and malicious examples.

Interpretation is not Straightforward. Supervised detection models are built automatically from data like unsupervised anomaly detection models. They have thus the same drawback with regard to transparency. Their interpretation is not straightforward, and some supervised model classes are more easily interpretable than others.

Compromise Between Misuse and Anomaly Detection. Supervised learning does not rely on specific malicious behaviors as much as misuse detection. It builds a generic representation of malicious and benign behaviors from many benign and malicious examples. Supervised detection models are thus more likely to detect yet unknown threats than misuse detection rules.

Unlike unsupervised learning, supervised learning is driven by malicious examples provided by security administrators. Supervised detection models have some characterization of malicious behavior which makes them less likely to detect yet unknown threats than anomaly detection methods [79, 148]. However, supervision reduces the false positive rate which is crucial for operational deployments.

To sum up, supervised detection is a good compromise between misuse and anomaly detection with regard to effectiveness and robustness. Supervised detection models are more likely to detect yet unknown threats than misuse detection rules, and they are less prone to false positives than anomaly detection models.

In this thesis, we focus on supervised learning for two main reasons that have been detailed above. First, this detection technique is highly controllable: supervised detection models can be updated automatically with both benign and malicious examples. Second, we have shown that it is a good compromise between misuse and anomaly detection with regard to effectiveness and robustness. Hereinafter, we employ the terms *supervised learning* and *machine learning* interchangeably.

Despite these assets, there remain some challenges to improve the operability of supervised learning in detection systems. In the next section, we detail the main issues this thesis addresses.

1.4 Challenges for Deploying Supervised Detection Models

Supervised detection models can be deployed in detection systems, to improve the detection of yet unknown threats. Various detection problems have been tackled with supervised learning : Android applications [54], PDF files [124, 37], Flash files [49, 141], portable executable files [75], botnets [25, 10, 27], Windows audit logs [22], and memory dumps [17]. Despite these encouraging results, there remain some significant barriers to the widespread deployment of machine learning in operational detection systems.

The standard supervised learning pipeline consists of data annotation, feature extraction, training and evaluation. Security administrators must carry out all these steps to set up supervised detection models ready for deployment. Once supervised detection models are deployed, security operators exploit the alerts they trigger. In this section, we identify three issues that should be addressed to foster the deployment of machine learning in operational detection systems.

1.4.1 How to Set up the Whole Machine Learning Pipeline ?

Security administrators may have little knowledge about machine learning. It may therefore be difficult for them to take full advantage of this data analysis technique in their detection systems.

First, there are many supervised model classes: deep neural networks, random forests, Support Vector Machines (SVM), k-nearest neighbors or naive Bayes classifiers to mention only the most obvious examples. Security administrators may need guidance to pick a model class that suits their operational constraints. Can all supervised detection models make predictions quickly enough to operate on streaming data ? Are some model classes easier to interpret, or robuster than others ?

Besides, detection models must be thoroughly evaluated before deployment to operate successfully. Security administrators may need advice about the assessment protocol. They may wonder how they can diagnose and handle potential accuracy issues.

Finally, security administrators should not have to deal too much with the machine learning machinery. Machine learning tools should enable them to build detection models even they have little or no knowledge about machine learning. Graphical user interfaces should enable security administrators to understand how detection models behave, i.e. which factors mainly influence their decision-making to trigger alerts. They should also provide information to help security administrators identify and address potential accuracy issues.

To sum up, security administrators are usually not machine learning experts, and they may need support to take full advantage of machine learning in their detection systems. They may need guidance to understand machine learning methodology, and how to design well-performing detection models. Finally, they should have access to machine learning tools to focus more on detection than on machine learning machinery.

1.4.2 How to Ease Data Annotation ?

Training data play a central role in supervised learning. Their quality has a direct impact on the performance of resulting detection models. Most research works on supervised detection assume that a representative annotated dataset is available for training while such datasets are particularly expensive to build in computer security. Some annotated datasets related to computer security are public (e.g. Malicia project [91], KDD99 [136], kyoto2006 [128], Contagio [1], Drebin [11]) but they are quickly outdated and they often do not account for the idiosyncrasies of each deployment context.

Annotated datasets of files (e.g. portable executable, PDF, or Windows Office documents) can be exploited to train detection models deployed in diverse environments. These kinds of data are likely to vary slightly from one environment to another. On the contrary, event logs (network or operating system event logs) and detection targets are highly dependent on deployment environments: a same behavior can be legitimate in an environment, but irregular in another. As a result, a detection model trained for a given environment is likely to perform poorly in another. These types of data require to build detection models *in-situ* [132], i.e. with training data coming from production environments.

A solution to meet this *in-situ* constraint is to ask security administrators to annotate data coming from production environments. In comparison to designing new machine learning algorithms, designing new annotation interfaces has received far less attention [85]. Some annotation interfaces are readily available for image, video, or text data, but they are too specific and do not suit computer security detection problems.

Besides, expert knowledge is required to annotate and data processed by detection systems are often sensitive. As a result, crowd-sourcing [126] and gamification [65] cannot be applied as in computer vision or natural language processing to acquire annotated datasets at low cost.

Active learning strategies [119] have been introduced in the machine learning community to reduce the number of manual annotations. Active learning is an iterative process: at each iteration, the strategy selects the most informative examples, asks security administrators to annotate them, and update the detection model accordingly. Active learning is thus an expert-in-the-loop process, where user experience should not be overlooked.

The computer security community has exploited active learning, but the expert-model interaction has often been neglected. They have mostly focused on query strategies, and not on their integration in annotation systems. It is, however, crucial to design both components jointly, the active learning strategy and the annotation system, to effectively reduce the annotation workload. Security administrators do not want to minimize only the number of manual annotations, but the overall time spent annotating.

In brief, there is a real need for active learning systems that are tailored to security experts needs. User experience must be taken into account while designing such systems to effectively streamline annotation projects.

1.4.3 How to Automate Feature Extraction ?

Standard machine learning algorithms do not take raw data as input, but instances represented as fixed-length vectors of features. Features are binary, numerical, or categorical values describing an instance that are exploited by detection models to make decisions. The feature extraction step of the machine learning pipeline depends on the data type considered.

The vast majority of research works on supervised detection have implemented a feature extraction method specific to their detection problem. Nonetheless, detection systems process many data types and designing a feature extraction method for each of them is tedious.

Automatic feature generation [129, 28, 72, 78] can significantly ease the deployment of machine learning in detection systems. The primary aim of such techniques is to generate discriminating features to get well-performing detection models. In the context of detection systems, there are additional constraints. First, the generated features must be interpretable and robust, so that the resulting detection models have the same properties. Besides, security administrators have domain knowledge that can lead the feature generation process to get better features. The feature generation method should thus be semi-automatic to let security administrators share their expertise.

To sum up, we emphasize three points that we think are critical to ease, and thus foster, the deployment of machine learning in operational detection systems:

1. provide interactive tools to set up the whole machine learning pipeline ;
2. ease the acquisition of annotated datasets ;
3. automate feature extraction.

1.5 Contributions

The overall objective of this thesis is to foster the deployment of supervised learning in detection systems to increase detection capabilities. To that end, we adopt an end-to-end approach. We take into account the whole machine learning pipeline (data annotation, feature extraction, training, and evaluation) with security administrators and operators as its core since it is crucial to pursue real-world impact [145, 85].

Part I concerns the whole machine learning pipeline, while Parts II and III focus on data annotation and feature extraction respectively.

Part I. First of all, Part I aims to help security administrators understand how they can apply machine learning, and make this data analysis technique suits their operational constraints. This part addresses the challenge stated in Section 1.4.1. Its content has been published in French at the *Symposium sur la sécurité des technologies de l'information et des communications* (SSTIC 2017) [26].

Chapter 2 reviews machine learning techniques and methodologies with a computer security point of view. We explain how the operational constraints should guide the selection of the supervised model class. Besides, we expose a three-step evaluation protocol that security administrators should conduct to diagnose and address potential accuracy issues.

Chapter 3 introduces DIADEM, an interactive visualization tool we have designed and implemented to help security administrators apply the methodology set out. DIADEM deals with the machine learning machinery to let security administrators focus mainly on detection. We illustrate how DIADEM can be leveraged to set up detection models with a case study on malicious PDF files detection.

Part II. Then, Part II addresses the challenge stated in Section 1.4.2. It proposes a solution to effectively reduce the workload in computer security annotation projects. We present an end-to-end active learning system, ILAB, tailored to security administrators needs. We have designed the active learning strategy and the user interface jointly to effectively reduce the annotation effort.

First, Chapter 5 introduces ILAB active learning strategy that we have designed to minimize not only the number of manual annotations, but also the waiting-periods. We compare ILAB active learning strategy with three state-of-the-art methods [4, 130, 63]. It shows that ILAB improves the detection performance without increasing the execution time. This chapter content has been published at the *20th International Symposium on Research in Attacks, Intrusions and Defenses* (RAID 2017) [18].

Second, in Chapter 6 we explain how we have integrated ILAB active learning strategy into a user interface. We carry out user experiments with security administrators to get feedback from intended end-users. It demonstrates that ILAB is an efficient active learning system that security administrators can deploy in real-world annotation projects. This chapter content has been published at the *Artificial Intelligence for Computer Security workshop* (AICS 2018) [19].

Part III. Finally, Part III provides the beginning of an answer to the last challenge stated in Section 1.4.3. It focuses on automatic feature generation in the context of detection systems.

First, we define the constraints that such methods should meet to be effective to build detection models. Then, we review some related works in various communities. On the one hand, Hidost [129] has been introduced in the computer security community to generate features automatically from hierarchical files. On the other hand, more generic approaches, relying on relational data, have been proposed in the machine learning community [28, 72, 78]. To the best of our knowledge, these generic feature extraction techniques [28, 72, 78] have never been applied to detection systems, while they could be beneficial. Therefore, we compare Hidost [129] to [28, 72] on two detection problems. This comparison leads to some avenues of research to better tailor automatic feature generation to security administrators needs.

In this thesis, we work on the whole machine learning pipeline with security administrators and operators as its core since it is crucial to pursue real-world impact. Moreover, the solutions we propose are completely generic to be beneficial to any detection problem on any data type.

We have implemented the SecuML [20] framework to help security administrators build machine learning models and interact with them. We provide open-source implementations of DIADEM and ILAB in SecuML to ease comparison in future research works, and to enable security administrators to build their own detection models.

Part I

Setting up Supervised Detection Models

Chapter 2

Methodology of Supervised Detection for Security Administrators

Security administrators are willing to deploy supervised detection models, in parallel to traditional detection methods, to enhance detection. They may, however, have little or no knowledge about machine learning. They may therefore have trouble building supervised detection models ready for deployment.

This chapter reviews machine learning techniques and methodologies with a computer security point of view. We present the whole machine learning pipeline, and we explain how the operational constraints should drive the choice of the supervised model class. We place a particular emphasis on evaluating properly supervised detection models since it is critical for successful deployments. This chapter introduces a three-step evaluation procedure that security administrators should carry out to diagnose and handle potential accuracy issues before deployment.

This chapter content has been published in French at the *Symposium sur la sécurité des technologies de l'information et des communications* (SSTIC 2017) [26].

Contents

2.1 Introduction

Supervised detection models can be deployed in detection systems to improve detection capabilities. However, security administrators, who are responsible for setting up detection methods, may have little or no knowledge about machine learning.

They may wonder how to build supervised detection models that fulfill their operational constraints (see Section 1.2). Can a detection method based on machine learning process streaming data ? Is the false alarm rate of supervised detection models low enough to prevent meaningless alerts from overwhelming security operators ? Machine learning based detection models are reputed to be black-box methods among security administrators. How can they trust such models to deploy them in operational detection systems ? Are the alerts generated by such models sufficiently interpretable for security operators to exploit them ?

Moreover, evaluating detection models is not a straightforward procedure, while it is a critical step to ensure successful deployments. Security administrators may need advice to evaluate detection models thoroughly and to find solutions in case of performance issues.

This chapter reviews machine learning techniques and methodologies with a computer security point of view.

- We present the whole machine learning pipeline that security administrators must set up to build supervised detection models ready for deployment.
- We explain how each operational constraint (see Section 1.2) should drive the choice of the supervised model class.
- We expose a three-step evaluation protocol that security administrators should conduct to diagnose and address potential accuracy issues.

The rest of the chapter is organized as follows. Section 2.2 provides a broad overview of supervised learning, and explains the different stages of the machine learning pipeline. Then, Section 2.3 presents some supervised model classes, and Section 2.4 provides guidance for selecting a model class that meets the operational constraints of detection systems. Finally, Section 2.5 provides methodological advice to diagnose and handle potential accuracy issues.

2.2 Overview of Supervised Detection

2.2.1 A Two-Stage Process: Training and Detection

A binary classifier based on supervised learning can be deployed in detection systems as a means of detection (see Figure 2.1). The classifier takes as input an *instance*, a PDF file for example, and returns the *predicted class label*, benign or malicious (in green and red in the figure), as output. The binary classifier is called a *detection model* in the context of detection systems.

Our presentation of supervised learning is based on the example of malicious PDF files detection, but we want to emphasize that supervised learning is a generic approach. It can be applied to any data type: the instance can also be a Windows Office document, an Android application, or the traffic associated to an IP address.

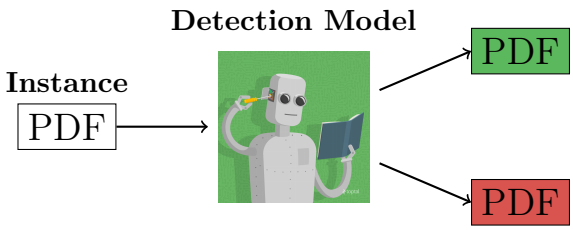


Figure 2.1: Detection Model based on a Binary Classifier.

Supervised learning involves two stages (see Figure 2.2) [52, 8]:

1. *Training*: the detection model is trained from annotated data ;
2. *Detection*: the detection model predicts whether an alert should be triggered or not.

Training. During the first step, the classifier is built from annotated data: a set of instances, malicious and benign, for which the label is known (see Figure 2.2a). The annotated data leveraged to train the model are called *training data*. The *training algorithm* finds automatically the similarities of the instances within each class and the discriminating characteristics to build the detection model.

Detection. 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 (see Figure 2.2b).

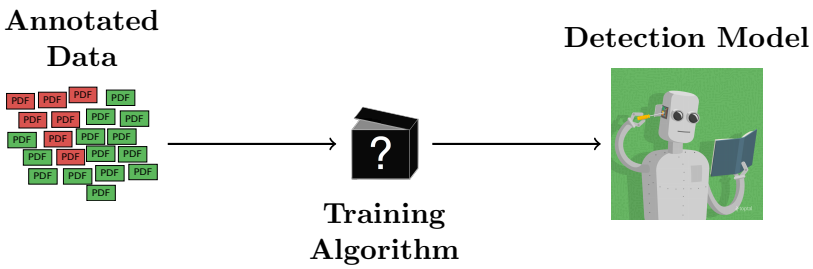
The detection system triggers an alert only if the probability of maliciousness is above the detection threshold set by security administrators. In the example depicted in Figure 2.2b, an alert will be triggered for the input PDF file only if the detection threshold is below 75%. The probability of maliciousness predicted by the detection model allows to sort alerts according to the confidence of the predictions, and thus to identify which alerts security operators should analyze foremost.

2.2.2 Detection Performance Metrics

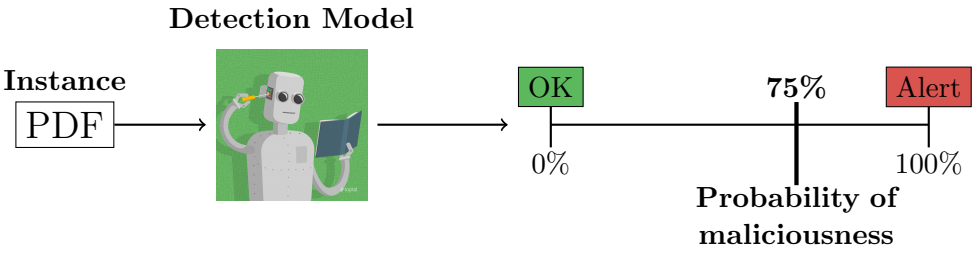
Classification Error Rate. The most prominent measure of performance is the classification error rate, i.e. the percentage of misclassified instances, but it is not suitable in the context of threat detection. Indeed, the data are usually unbalanced with a small proportion of malicious instances.

We present an example demonstrating the limits of the classification error rate. We consider 100 instances: 2 malicious and 98 benign. In this situation, a dumb detection model predicting always benign has a classification error rate of only 2% while it is not able to detect any malicious instance.

Confusion Matrix. To assess properly the performance of a detection method, we must begin by writing the confusion matrix. The confusion matrix takes into account the two possible types of



(a) Step 1: Training of the Detection Model.



(b) Step 2: Detection.

Figure 2.2: Supervised Learning: a Two-Stage Process.

errors: *false negatives*, i.e. malicious instances which have not been detected, and *false positives*, i.e. benign instances which have triggered a false alarm. Figure 2.3 explains the content of a confusion matrix.

		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`

Figure 2.3: Explanation of the Confusion Matrix.

False Positive and Detection Rates. The confusion matrix allows to express measures of performance such as the detection rate (2.1) and the false positive rate (2.2), also called false alarm rate.

$$\text{False Positive Rate} = \frac{FP}{FP + TN} \tag{2.1}$$

$$\text{Detection Rate} = \frac{TP}{TP + FN} \tag{2.2}$$

The performance of a detection model can be properly assessed with both measures taken jointly. On the one hand, the false positive rate must be low enough to prevent false alarms from overwhelming security operators. On the other hand, the detection rate must be high to avoid missing a threat that could lead to a security incident.

The detection threshold determines the sensibility of the detection: decreasing it increases the detection rate, but also the false alarm rate. Security administrators set the detection threshold depending on the desired tradeoff between detection and false alarm rates.

ROC Curve. The measures of performance we have introduced so far depend on the value of the detection threshold. The ROC (Receiver Operating Characteristic) curve [67] is another measure of performance. It has the benefit of being independent of the detection threshold. This curve represents the detection rate according to the false positive rate for various values of the detection threshold (see Figure 2.4).

For a threshold of 100%, the detection and false alarm rates are null, and for a threshold of 0% they are both equal to 100%. A good detection model has a ROC curve close to the upper left corner : a high detection rate with a low false alarm rate. The AUC, which stands for Area Under the ROC Curve, is often computed to assess the performance of a detection model independently of the detection threshold. A good detection model has an AUC close to 100%.

The ROC curve of a classifier predicting randomly the probability of maliciousness is the straight red line depicted in Figure 2.4. A ROC curve is always above this straight line and the AUC is at least 50%. Thanks to the ROC curve, security administrators can set the value of the detection threshold according to the detection rate desired or the false alarm rate tolerated.

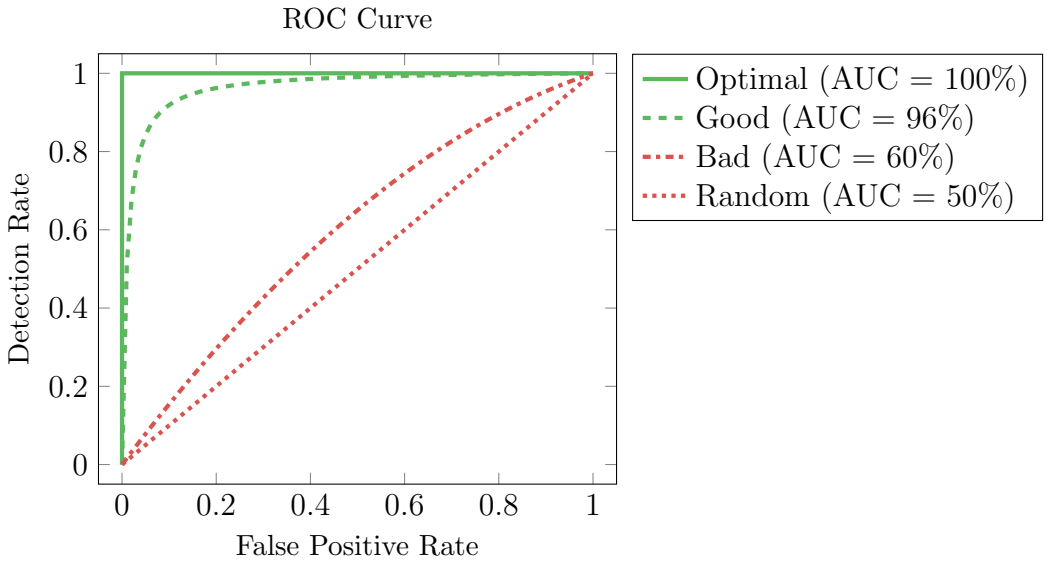


Figure 2.4: Explanation of the ROC Curve.

2.2.3 Underfitting-Overfitting Tradeoff

The underfitting-overfitting tradeoff is critical in supervised learning.

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.

Overfitting. The primary objective of supervised learning is not to predict perfectly the labels of training data, but to generalize properly to yet unseen instances. *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 is 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.

There is a tradeoff between underfitting and overfitting because these concepts are directly related to model complexity. Simple models with little flexibility are likely to suffer from underfitting while more complex and flexible models are likely to suffer from overfitting. The real challenge of machine learning is to find a detection model complex enough to fit the training data well, but not too complex to generalize properly to yet unseen instances.

2.2.4 The Whole Machine Learning Pipeline

Section 2.2.1 provides a simplified picture of the training step (see Figure 2.2a). It is not straightforward to build supervised detection models ready for deployment. It involves a whole processing

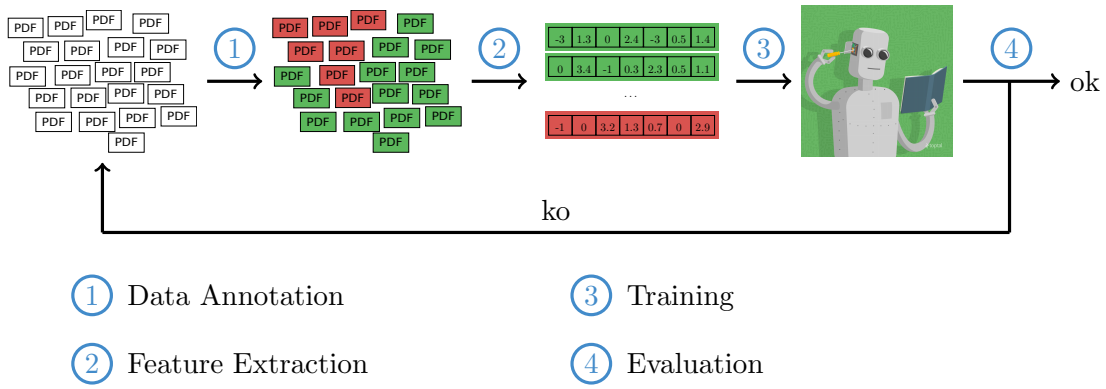


Figure 2.5: The Whole Machine Learning Pipeline.

pipeline composed of the following steps (see Figure 2.5): 1) data annotation, 2) feature extraction, 3) training, and 4) evaluation. In this section, we briefly outline each step.

1. Data Annotation. Building a supervised detection model requires training data for which the labels, benign or malicious, are known. These training data must contain malicious and benign instances consistent with the desired detection target, i.e. what security administrators wish to detect. Security administrators must supervise closely the collection of annotated data, to ensure it suits their detection target.

Some principles must be respected while collecting a training dataset. First of all, it must contain enough instances so that the supervised detection model generalizes the malicious and benign behaviors properly. Besides, it must not be too much unbalanced, i.e a label must not represent only a tiny proportion of the instances. We cannot provide a generic rule defining the minimal number of instances required to train a detection model, nor the level of imbalance allowed. These values depend deeply on the detection problem considered, and on the instances in the dataset.

2. Feature Extraction. Standard machine learning algorithms do not take raw data as input, but instances represented as fixed-length vectors of features. Features are binary, numerical, or categorical values describing an instance that detection models exploit to make decisions. The more discriminating the features are, the more efficient the detection model is. Security administrators should leverage their domain expertise to extract features that are relevant for their detection target.

Feature extraction is a three-step process: 1) identifying discriminating information, 2) parsing the data to extract this information, and 3) transforming this information into fixed-length vectors of features. The first two steps require a good knowledge of the data format and of the detection target while the last one can exploit generic techniques.

Feature extraction provides a standard representation of instances that makes training of detection models independent of the data type.

3. Training. There are many types of supervised detection models that are referred to as *model classes* [52]: neural networks, random forests, k-nearest neighbors, or support vector machines (SVM) to name only the most obvious examples. The reader may refer to Section 2.3 for more information about supervised model classes.

First, security administrators must pick a model class that suits their operational constraints. Then, they can leverage one of the many libraries dedicated to machine learning (e.g. scikit-learn in python, Spark, Mahout or Weka in java, or Vowpal Wabbit in C++) to train the model class they have chosen on their training data.

4. Evaluation. Supervised detection models are not flawless, they may make prediction errors. It is critical to carry out a thorough evaluation of detection models before their deployment in detection systems to avoid unpleasant surprises.

The detection rate must be satisfactory, and the false positive rate low enough to prevent false alarms from overwhelming security operators. The reader may refer to Section 2.2.2 for more detail about detection performance metrics.

The detection performance of detection models must be assessed on an independent dataset. It is crucial that the evaluation dataset, also called validation dataset, is completely independent from the training one. Indeed, the objective is to ensure that the detection model is able to predict accurately the label of instances unseen during the training phase.

The first trained model is usually not satisfactory. Setting up a detection model is an iterative process where the evaluation phase provides avenues for improving training (see Figure 2.5).

At each iteration, security administrators train and validate a detection model. Based on the 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.

In this chapter, we provide general guidance for two steps of the machine learning pipeline: training and evaluation. In Section 2.3, we provide a broad overview of supervised model classes, and in Section 2.4, we advise security administrators on how they should pick a model class that fulfills their operational constraints. Finally, in Section 2.5, we introduce a three-step evaluation protocol that security administrators should follow before any deployment.

2.3 Supervised Model Classes

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, random forests, k-nearest neighbors, linear or quadratic discriminant analyses, logistic regression, Support Vector Machines (SVM) or naive Bayes classifiers, to cite just a few examples.

The objective of this section is not to provide a comprehensive review of supervised model classes, but rather to present the best-known and to explain how they are related. We also provide some technical details about the model classes, because they are crucial to choose a model class that meets the operational constraints of detection systems.

Notations. In order to explain how supervised models work, we need some notations. Let $x \in \mathbb{R}^m$ denote an *input instance* represented by m real-valued features. In the context of computer security detection, an input instance x can represent a PDF file, an Android application, the traffic of an IP address, or the activity of a user on a computer platform. Let $y \in \{0, 1\}$ be the *output class variable*, i.e. the outcome we want to predict : 0 for `Benign`, and 1 for `Malicious`.

We seek a *mapping function* f for predicting y given x. Supervised learning fits the mapping function f from a training dataset $\mathcal{D} = \left\{(x^i, y^i) \mid x^i \in \mathbb{R}^m, y^i \in \{0, 1\}\right\}_{1 \leq i \leq n}$. Supervised learning makes the assumption that the training instances are independent and identically distributed.

The model classes differ by the form of the function f and by the criteria they intend to optimize while fitting f from training data. In this section, we review model classes from two broad categories [106]: *parametric* and *non-parametric* models.

2.3.1 Parametric Models

Definition. Russell et al. [114] define parametric models as follows: "A learning model that summarizes data with a set of parameters of fixed size (independent of the number of training examples) is called a parametric model. No matter how much data you throw at a parametric model, it won't change its mind about how many parameters it needs.".

Parametric models impose a specific form to the mapping function f that is characterized by a fixed number of parameters that do not depend on the amount of training data. They usually make assumptions about the data, such as their underlying distribution, while designing the form of the mapping function f.

Naive Bayes classifiers, logistic regression, Support Vector Machines (SVM), and neural networks (with a fixed architecture) are examples of parametric models.

Naive Bayes Classifiers

Mapping Function. Naive Bayes classifiers fit a function f_{NB} based on a maximum a posteriori decision rule (2.3).

$$f_{NB}(x) = \underset{k \in \{0,1\}}{\arg\max}\ P(y = k \mid x) \tag{2.3}$$

They make a strong *features independence assumption*. They assume that the value of a particular feature x_j is independent of the value of any other feature, given the class variable y (2.4).

$$P(x \mid y = k) = \prod_{j=1}^{m} P(x_j | y = k) \tag{2.4}$$

This assumption (2.4) combined with the Bayes rule (2.5) leads to a new formulation of the mapping function f (2.6).

$$P(y \mid x) = \frac{P(x \mid y) \cdot P(y)}{P(x)} \tag{2.5}$$

$$f(x) = \underset{k \in \{0,1\}}{\arg\max}\ P(y = k) \cdot \prod_{j=1}^{m} P(x_j \mid y = k) \tag{2.6}$$

Estimation of the Parameters. The training phase must estimate the probability distributions $P(y)$ and $P(x_j \mid y)$ from the training instances. $P(y)$ is usually estimated empirically by the class frequency in the training dataset. $P(y = 0)$ is estimated with $\hat{P}(y = 0)$, the empirical proportion of `Benign` instances within the training data (2.7). $\hat{P}(y = 1)$ is easily derived from $\hat{P}(y = 0)$ with $\hat{P}(y = 1) = 1 - \hat{P}(y = 0)$.

$$\hat{P}(y = 0) = \frac{\mid \left\{(x^i, y^i) \in \mathcal{D} \mid y^i = 0\right\} \mid}{n} \tag{2.7}$$

We need to make assumptions about the distribution of $x_j \mid y$ to parametrize it. Since the features are continuous, we can use Gaussian distributions (2.8).

$$x_j \mid y = k \sim \mathcal{N}(\mu_{j,k}, \sigma^2_{j,k}) \iff P(x_j \mid y = k) = \frac{1}{\sqrt{2\pi\sigma^2_{j,k}}} \exp\left(\frac{(x_j - \mu_{j,k})^2}{2\sigma^2_{j,k}}\right) \tag{2.8}$$

The parameters of the Gaussian distributions, $\mu_{j,k}$ and $\sigma^2_{j,k}$, are estimated empirically from the training data with the empirical mean, $\hat{\mu}_{j,k}$ (2.9), and the empirical variance, $\hat{\sigma^2}_{j,k}$ (2.10).

$$\hat{\mu}_{j,k} = \frac{1}{n} \sum_{\substack{i=1 \\ y^i = k}}^{n} x^i_j \tag{2.9}$$

$$\hat{\sigma}^2_{j,k} = \frac{1}{n} \sum_{\substack{i=1 \\ y^i = k}}^{n} (x^i_j - \hat{\mu}_{j,k})^2 \tag{2.10}$$

Naive Bayes classifiers are parametric models. Their mapping functions are characterized by $4m + 1$ parameters if the probability $x^i \mid y$ is modelized wih a Gaussian distribution: $\hat{P}(y = 0)$ and $\left\{(\hat{\mu}_{j,k}, \hat{\sigma}^2_{j,k}) \mid 1 \leq j \leq m,\ k \in \{0, 1\}\right\}$. They are highly scalable since they are characterized by a number of parameters linear in the number of features and these parameters can be estimated quickly with a closed-form expression. However, the assumptions made by naive Bayes classifiers (features independence, underlying distribution of $x_j \mid y$) may not hold in practice and therefore damage the detection performance.

Linear Models: Logistic Regression and SVM

Mapping Functions. Logistic regression and Support Vector Machines (SVM) are linear models. The computation of the output class variable y depends only on the sum (no product or more complex functions) of the input features and parameters. Logistic regression and SVM have their own type of mapping functions.

Logistic regression modelizes the probability of maliciousness $P(y = 1 \mid x)$ with a sigmoid function (2.11).

$$P(y = 1 \mid x) = \frac{1}{1 + \exp\left(-\left(\beta_0 + \beta^T x\right)\right)} \tag{2.11}$$

The parameters $\beta_0 \in \mathbb{R}$ and $\beta \in \mathbb{R}^m$ are learned from the training dataset $\mathcal{D}$. They characterize the linear decision boundary: β_0 is usually called the intercept, and β associates a weight to each feature. Once these parameters have been fit, the mapping function f_{LR} (2.12) is used to make predictions. It depends on a detection threshold $t \in [0, 1]$ that must be set by security administrators manually

according to the desired tradeoff between detection and false alarm rates. The mapping function f_{LR} only checks whether the predicted probability of maliciousness is above the detection threshold t.

$$f_{LR}(x) = \text{sign}\left(\frac{1}{1+\exp\left(-\left(\beta_0+\beta^T x\right)\right)} - t\right) \tag{2.12}$$

SVM builds a hyperplane that separates the benign from the malicious instances. The hyperplane is characterized by the equation $\beta_0 + \beta^T x = 0$, where $\beta_0 \in \mathbb{R}$ and $\beta \in \mathbb{R}^m$. These parameters are fit during the training phase. The function mapping f_{SVM} (2.13) predicts the outcome y for an input instance x according to its position in relation with the hyperplane.

$$f_{SVM}(x) = \text{sign}(\beta_0 + \beta^T x) \tag{2.13}$$

Estimation of the Parameters. The training algorithms of logistic regression and SVM optimize an objective function on the training data to fit the parameters. Equation (2.14) presents the general form of objective functions. θ corresponds to the parameters of the mapping function f that should be learned during the training phase. For logistic regression and SVM, $\theta = (\beta_0, \beta)$.

$$\min_{\theta} \frac{1}{n}\sum_{i=0}^{n} l(y^i, f(x^i, \theta)) + c \cdot \Omega(\theta) \tag{2.14}$$

The objective function (2.14) is composed of two terms: 1) a loss function, $l(y^i, f(x^i, \theta))$, and 2) a regularization penalty, $c \cdot \Omega(\theta)$. We explain the purpose of each term below.

Loss Function. The function l is a *loss function*, or *cost function*, that assesses the difference between the predictions made by the model $f(x^i, \theta)$, and the ground-truth labels y^i. The objective of the first term of the objective function (2.14) is to penalize the prediction errors on the training dataset.

Logistic regression and SVM rely on different loss functions. They are not detailed in this thesis. The reader may refer to [52] for more detail.

Regularization Penalty. The primary aim of mapping functions f is not to predict perfectly the output class variables of training data, but to generalize properly to unseen instances during the training phase. The second term, $c \cdot \Omega(\theta)$, called *regularization penalty*, is added to improve generalization. It controls the underfitting-overfitting tradeoff (see Section 2.2.3). The function Ω evaluates the complexity of the detection model according to its parameters, and the hyperparameter $c \in \mathbb{R}^+$ controls the strength of the regularization.

Increasing c decreases the strength of the regularization and tends therefore to generate more complex models that may overfit. On the contrary, decreasing c strengthens the regularization and favors simpler models that may underfit. To sum up, the hyperparameter c is crucial to control the underfitting-overfitting tradeoff. Its value must be set properly to avoid both overfitting and underfitting.

Many loss functions have been introduced in the literature. The most common are the ℓ_1 and ℓ_2 norms [52]. The ℓ_1 norm performs also feature selection: it induces sparse models where many components of β are null.

Optimization of the Objective Function. Training algorithms rely on optimization techniques [29] to fit the parameters θ from the training data. The logistic regression and SVM model classes were introduced decades ago in the machine community, and were originally trained with classical optimization techniques [149]. There is still ongoing research work on optimization to improve the fitting of their parameters [44, 115]. The new optimization techniques are more efficient on large datasets (the number of instances and/or the number of features are large), and they allow to train models on streaming annotated data. On the contrary, iterative optimization techniques (e.g. gradient descent, Newton methods, or quasi-Newton methods) need to store the training data. They go through the training instances several times to fit the parameters.

More Complex Model Classes

The parametric models we have introduced so far, naive Bayes classifiers, SVM and logistic regression, are simple, and may not be flexible enough to fit the data properly. Naive Bayes classifiers assume that the features are independent, while SVM and logistic regression assume that the data are linearly separable. These assumptions may not hold in practice and lead to underfitting.

Some parametric models are more flexible, and can fit more complex distributions. The objective function (2.14) is generic and leads to various model classes through the choice of the form of the mapping function f, the loss function l, and the regularization function Ω. SVM and logistic regression are linear models: their mapping function f is linear in the input features. More complex models are derived with non-linear mapping functions (e.g. quadratic discriminant analysis).

Neural networks are also parametric models, whose complexity can be arbitrarily increased by adding layers of neurons. Moreover, linear SVM can be extended to fit more complex distributions thanks to kernels that transform the input data. The reader may refer to [52] for more detail about more complex parametric models.

2.3.2 Non-Parametric Models

Definition. Non-parametric models have a potentially infinite number of parameters that grows with the number of training instances. They usually make mild structure assumptions, and can therefore adapt to any situation. However, they may perform poorly on high-dimensional data because of the curse of dimensionality [52].

k-Nearest Neighbors

k-nearest neighbors proceed as follows to predict the label of an instance x. First, it looks for the k-nearest neighbors of the instance x, denoted by $\mathcal{N}_k(x)$, among all the training data $\mathcal{D}$. Then, it predicts the most represented label among the k-nearest neighbors (2.15).

$$f_{NN}(x) = \underset{y \in \{0,1\}}{\arg\max} \left| \left\{ (x^i, y^i) \in \mathcal{D}, x^i \in \mathcal{N}_k(x), y^i = y \right\} \right| \tag{2.15}$$

k-nearest neighbors require to specify: 1) the number of neighbors k, and 2) the distance between the instances. The Euclidean distance is the default distance, but problem-specific distances can be defined.

k-nearest neighbors are called *lazy learners*. They have no training phase, and they therefore need all the training data during the prediction phase.

Decision Trees

Decision trees predict the output class variable by learning simple decision rules inferred from the training data. Internal nodes of decision trees contain binary conditions based on input features while leaves are associated with class labels (`Malicious` or `Benign`).

The class label for a new instance is predicted by going through the tree from the root to a leaf according to the conditions at each internal node and the instance input features. The final leaf predicts the class label.

Building Decision Trees. Decision trees are built recursively [104]. The root node contains all the training instances $\mathcal{D}$, and each internal node splits its training instances into two subsets (the children nodes) according to a condition based on the input features. The split conditions are selected in a way to minimize the node impurity.

A node is completely pure if all its instances share the same label: they are all benign or malicious. The impurity of a node is the largest when half the instances are benign, and the other half malicious. Several node impurity measures have been defined to build decision trees [52] (e.g. entropy or Gini index).

The stop condition of the recursive algorithm is usually specified by a minimum node impurity threshold. If the impurity of a node is below this threshold, then we stop splitting it. It becomes a leaf associated to the most common label among its training instances.

Tree-Based Ensemble Methods

Decision trees are very flexible models and may therefore suffer from overfitting. They may create over-complex trees that do not generalize the data well. In order to address this issue, ensemble methods, such as adaptive boosting classifiers [50] and random forests [30], have been introduced. They consist of several decision trees, called weak learners. The output class labels are computed through aggregations of the predictions of the individual decision trees.

In this section, we have explained how some model classes work and how they are related. These explanations will be leveraged in Section 2.4 to drive the choice of model classes that meet the operational constraints of detection systems.

Model classes (parametric or not) may have *hyperparameters*, i.e. parameters that must be set before the training phase. These parameters are not fit automatically by the training algorithm. For example, logistic regression has two hyperparameters: the regularization strength c and the penalty norm (ℓ_1 or ℓ_2). Regarding k-nearest neighbors, the number of neighbors k is an hyperparameter. In the next section, we explain how the values of the hyperparameters can be set automatically.

2.3.3 Setting the Hyperparameters

Grid Search. We usually consider all the combinations of hyperparameters values in a grid search to find the best ones. First, we evaluate the performance of the detection models trained with each combination. Then, we select the hyperparameters values that result in the best-performing detection model.

The grid search sets the values of the hyperparameters automatically. Security administrators must only specify: 1) the values of the hyperparameters they want to consider in the grid search, and 2), the criteria they want to maximize (e.g. AUC or F-score). The detection model performance is usually assessed with *cross validation*.

Cross Validation. Cross validation processes as follows to assess the performance of a detection model for a given combination of hyperparameters values.

1. The training dataset $\mathcal{D}$ is divided into g folds. Each fold has approximately the same number of instances and the same proportion of benign/malicious instances as the whole training dataset.

2. The detection model is trained g times: each time, one fold is the testing set and the other folds form the training set.

3. Each instance in the training dataset is used once for testing. As a result, at the end of the cross validation, we have a unique predicted label for each instance $x \in \mathcal{D}$. We rely on these predictions to compute the performance criteria that security administrators have selected for the grid search.

2.4 How to Choose a Suitable Model Class ?

The choice of the model class has a direct impact on the adequacy of the resulting detection model with the operational constraints (see Section 1.2). In this section, we explain how each constraint should drive the choice of the model class.

2.4.1 Controllability

When a detection model is deployed, security operators analyze the generated alerts. The primary purpose of their analyses is to detect false alarms and to take the necessary measures in the event of a security incident. Their analyses, both false and true positives, can also be integrated into the detection model to improve its detection capabilities.

All supervised model classes enjoy a high level of controllability. They can be updated with both malicious and benign examples. Besides, the update is performed automatically, so known detection errors can be corrected forthwith.

All supervised model classes fulfill perfectly the controllability constraint. This constraint does not restrict at all the set of suited model classes.

2.4.2 Online Processing

Supervised detection models must make predictions, whether an alert should be triggered or not, quickly enough to suit 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. The prediction phase of k-nearest neighbors has a temporal complexity in $O(nd)$ which depends on the number of features d, but also on the number of training instances n. However, n should be large to get well-performing detection models, and n increases over time when new instances are used to update the detection model. The prediction phase of this model class is far too long for detection systems.

To sum up, most model classes meet the online processing constraint. Only lazy learners, such as k-nearest neighbors, should be avoided. The online processing constraint is therefore not too restrictive.

2.4.3 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 [111]. First, security administrators need to trust detection models before their deployment. They want to understand their behavior as a whole. Second, security operators require information about why an alert has been triggered to handle it swiftly. Predictions of detection models should be interpretable to assist security operators in processing alerts. As a result, both individual predictions and detection models as a whole must be interpretable to meet security administrators and operators needs.

k-Nearest Neighbors. The predictions made by k-nearest neighbors are rather interpretable. The k-nearest neighbors can be displayed to security operators to explain why an alert has been triggered, but the detection model cannot be described as a whole. Besides, we have explained in Section 2.4.2 that it does not suit the online processing constraint.

Linear Models. Linear models, such as logistic regression or SVM, meet perfectly the transparency constraint. Both individual predictions and detection models as a whole can be easily interpreted.

The parameter $\beta \in \mathbb{R}^m$ associates a coefficient to each feature that allows 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 (the greater these features are, the lower the probability of maliciousness is) while features with positive coefficients point out malicious instances (the greater these attributes are, the greater the probability of maliciousness is).

The parameter β allows to interpret not only linear models as a whole, but also their individual predictions. For a given instance x, its prediction $f(x)$ can be interpreted with the product $\beta^T x$. Indeed, the components $\beta_j x_j$ having the greatest absolute values are the ones influencing the most the decision-making.

This interpretation approach has been criticized [89]: it may be misleading because of feature correlations. Indeed, features associated to a coefficient with a small absolute value are not necessarily useless to distinguish malicious from benign instances. For example, if two features are highly correlated, and share the same discriminative power, only one of them will be associated with a coefficient with a high absolute value. This is not an issue in our case. We are not interested in pointing out all the features that bring discriminative information, but rather the features that influence decision-making. In practice, security administrators and operators do not analyze all the coefficients, but only the ones having the highest absolute values.

Computer security experts usually appreciate linear models because they can make an analogy with expert systems (see Section 1.3.2). Linear models associate a weight to various variables that are called features in the machine learning context. The main advantage of linear models over expert systems is their controllability. The weights associated with each variable are not set manually with expert knowledge, but automatically from annotated data. As a result, the weights can be swiftly updated to follow threat evolution.

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.

Tree-based ensemble methods (e.g. adaptive boosting classifiers, random forests [30]) are more complex than decision trees, but they are still rather transparent. Their overall behavior can be described by the features' importance. The importance of a feature corresponds to the increase in the model prediction error after permuting the features values randomly [30]. Feature importance provides a highly compressed, global insight into the model behavior, but individual predictions are hard to interpret.

In this section, we have reviewed three supervised model classes that can be easily interpreted: k-nearest neighbors, linear models, and tree-based models. 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 [109]. These methods intend to separate explanations from machine learning models, in order to explain any model class, even the most complex. In this thesis, we do not detail how these methods work. The reader may refer to [110, 131, 84] for examples of such methods, or to [89] for more information about the interpretation of machine learning models.

2.4.4 Robustness

Evasion Attacks: Adversarial Examples. Supervised detection models are less prone to polymorphism attacks than misuse detection techniques since they are more generic. Attackers can, nevertheless, craft adversarial examples that evade detection while maintaining the same malicious payload [23, 24]. They usually make slight perturbations to their attack to cross the decision boundary of the detection model. Several methods to generate adversarial examples have been proposed: gradient-based evasion attacks [23], fast gradient sign method [60], and Jacobian based-saliency map approach [98]. These evasion techniques are generic, they are not tailored to any specific model class. Therefore, security administrators cannot pick a model class that is particularly robust. They are all vulnerable to adversarial examples.

Papernot et al. have created cleverhans [97], an open-source library for benchmarking the vulnerability of machine learning models to adversarial examples. This library cannot, however, benchmark computer security detection models directly. Indeed, adversarial methods do not directly manipulate real-world objects (e.g. PDF files, Android applications, event logs) but numerical vectors in the feature space. Adversarial learning has been mostly applied to image recognition where the mapping between the real-world objects, i.e. the images, and the features is straightforward. The features are simply the pixel values. Nonetheless, feature mappings may not be easily inverted in the context of threat detection. Besides, the adverse perturbations must not corrupt

the data format and must maintain the malicious payload. Adversarial methods must therefore be constrained to suit threat detection.

Biggio et al. [23] adapt their gradient-based evasion attacks to PDF files. In their experiments, they use the following feature mapping: each PDF file is represented by the number of occurrences of some keywords (e.g. `OpenAction`, `Comment`, `PageLayout`). They restrict their adversarial attacks to adding keywords since it is easy to insert new objects without corrupting the PDF file structure. In the feature space, this constraint is equivalent to accepting only feature increments. Adversarial examples have also been generated for Android applications [64, 45].

To sum up, supervised detection models can be evaded with adversarial examples whatever model class is chosen. Nonetheless, such examples are difficult to craft in the context of threat detection since feature mappings are usually hard to invert.

Defenses Against Adversarial Examples. Adversarial examples are hard to defend against because they require machine learning models to produce good outputs for every possible input. Most of the time, machine learning models work very well but only on a very small amount of all the many possible inputs they might encounter [59].

Two defense techniques against adversarial examples have been proposed, adversarial training [133, 60] and defensive distillation [99], but they do not work perfectly: they close some vulnerabilities, but leave others open [59].

In brief, making supervised detection models robust against evasion attempts is critical since they are deployed in adversarial environments, but it remains an open problem. There is currently no perfect defense against adversarial examples. Such examples are, nevertheless, more difficult to craft for detection systems since the feature mappings are usually harder to invert.

2.4.5 Effectiveness

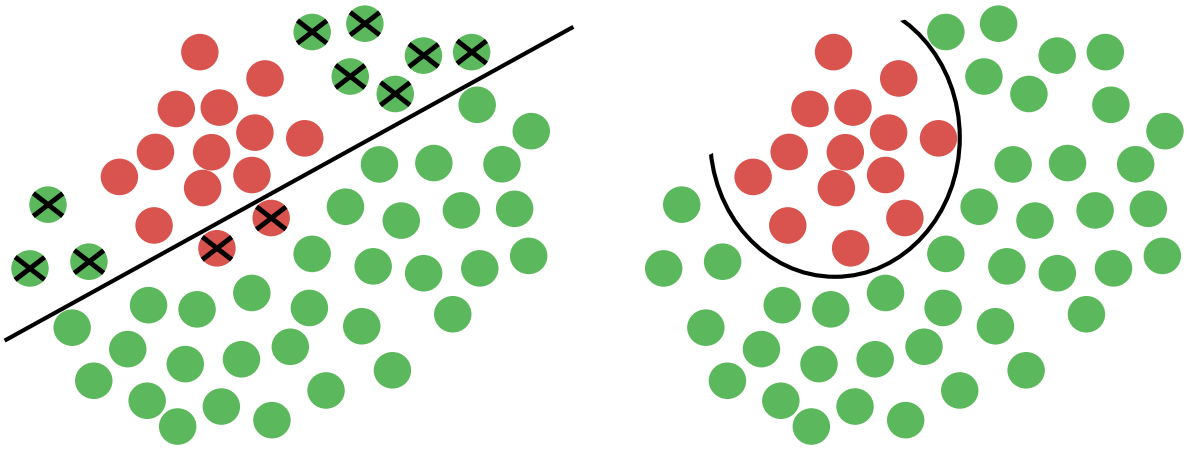
There is no way to determine which model classes are the most effective. The effectiveness of a given model class depends deeply on the data.

For instance, linear models are simple, but they require benign and malicious instances to be linearly separable to work properly. If the data are not linearly separable, more complex models such are quadratic models, tree-based models, or neural networks must be trained.

Selecting the most effective model class is data-dependent. The model selection is therefore performed empirically. In Section 2.5, we provide advice on how to choose the best model class empirically.

To sum up, the *controllability* and *online processing* constraints hardly restrict the set of model classes that suit detection systems. The *robustness* constraint is critical in adversarial environments, but there is still no consensus about the best method to make detection models robust against evasion attempts. *Transparency* and *effectiveness* are thus the most important criteria to pick a model class that suits detection systems.

We usually 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, we may move ahead to more flexible model classes (e.g. tree-based models, neural



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

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

Figure 2.6: *Illustration of Underfitting.* The two diagrams represent the training dataset. The bold black line represents the classifier decision boundary, and the crosses point out the classification errors.

networks). In the next section, we explain how to decide whether a more flexible model class is required.

2.5 How to Diagnose and Handle Potential Accuracy Issues ?

In this section, we introduce a three-step evaluation protocol that security administrators should follow before any deployment. Each step intends to diagnose a problem that may degrade the detection performance. We propose not only protocols to diagnose the problems, but also solutions to address them.

2.5.1 How to Diagnose and Handle Underfitting ?

Diagnosis. Underfitting (see Section 2.2.3) can be diagnosed by evaluating the performance of the detection model on the training dataset. There is underfitting if the ROC curve is close to that of a random generator (see the random curve in Figure 2.4). When there is underfitting, the model does not fit the training data well enough, and it makes prediction almost like a random generator.

Solution. Security administrators can solve underfitting in two ways: adding discriminating features or training a more complex classification model class.

When a model fails to discriminate between the malicious and benign instances on the training data, it is often because security administrators have not extracted good input features. They must therefore resume the feature extraction phase to add more discriminating characteristics.

Besides, security administrators may have provided discriminating features, but the chosen model class is not complex enough to discriminate between malicious and benign instances. Figure 2.6a 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 2.6b).

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.

2.5.2 How to Diagnose and Handle Overfitting ?

Diagnosis. Overfitting (see Section 2.2.3) 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, it suffers from overfitting.

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 datasets. The simplest way to split it up is to select some instances randomly for training, and to keep the remaining instances for validation.

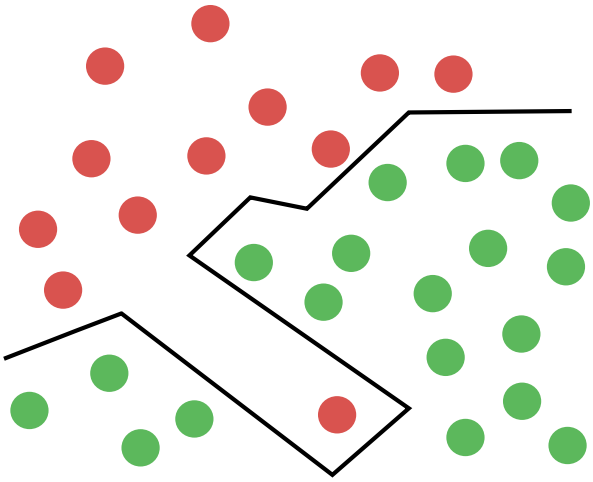
If the instances are timestamped, i.e. the times of first appearance are known, then a better evaluation process can be devised. Security administrators can set a cutoff timestamp: all the instances occurring before the cutoff timestamp constitute the training dataset, while the remaining instances are retained for validation purposes. This temporally consistent evaluation process [88] better assesses the performance of detection models since future instances are not available at training time. This evaluation process should thus be preferred when timestamps are available. The reader may refer to [71, 88, 108] for more detail about the validation of detection models.

Solution. Overfitting is usually caused by a too complex model class (see Figure 2.7) that has much flexibility to learn a decision boundary.

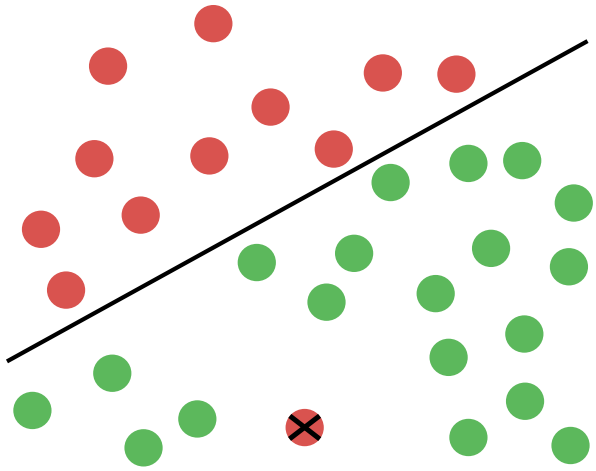
When the detection model is too complex (see Figures 2.7a and 2.7c), it predicts perfectly the label of the training instances (see Figure 2.7a), but it makes many prediction errors on the validation dataset (see Figure 2.7c). Indeed, the complex model fits perfectly the training data, but it has weak generalization capabilities. On the other hand, a simpler model (see Figures 2.7b and 2.7d). is able to avoid outliers 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.

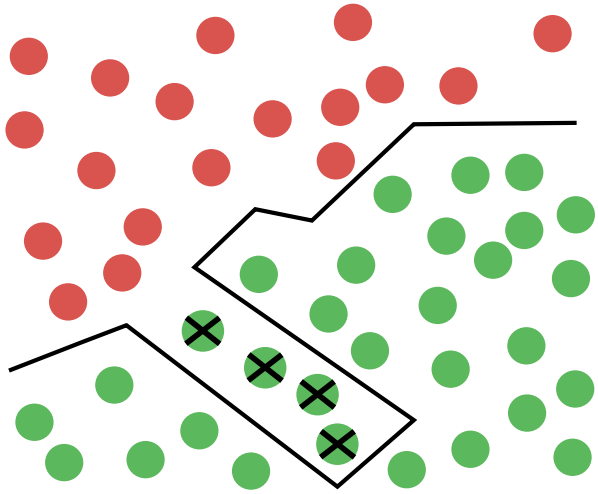
To sum up, it is critical to strike an appropriate balance to avoid both underfitting and overfitting. The detection model must be neither too simple to avoid underfitting nor too complex to avoid overfitting. To avoid both pitfalls, security administrators must assess the performance of a detection model both on its training data, and on an independent validation dataset.



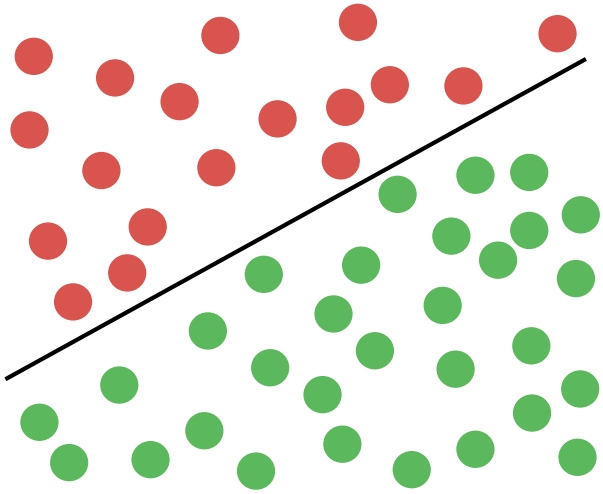
(a) *Complex Model:* no training errors.



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



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



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

Figure 2.7: *Illustration of Overfitting.* The top row represents the training dataset while the bottom line represents the validation dataset. The bold black line represents the classifier decision boundary, and the crosses point out the classification errors.

These two assessment steps may not be sufficient to ensure a successful deployment of detection models. They may not be able to identify a training bias. In the next section, we explain what a training bias is, and how to diagnose and solve this issue.

2.5.3 How to Diagnose and Handle Training Biases ?

Training Bias. There is a *training bias* if the training dataset is not representative of the data encountered in the deployment environment. Training biases negatively impact the performance of detection models once deployed.

Training bias is common with computer security detection models since training datasets do not often come from deployment environments. Annotating data is costly, so security administrators usually rely on public annotated datasets that may not be representative of deployment environments. Besides, they may contain biases that the model picks up and learns as concepts.

Diagnosis. The study of the model performance on the training and validation datasets is not sufficient to detect any training bias prior to deployment: the model behavior must be thoroughly analyzed. In Section 2.4.3, we have explained that detection models must be interpretable to fit security administrators and operators needs. This interpretability is also critical to diagnose training biases before deployment.

When detection models are interpretable, 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.

Solution. We propose two solutions to reduce the impact of training biases when the available annotated data, used as a training dataset, are not representative of the data in production.

An intermediate adaptation phase of the model in production can reduce the training bias. The false alerts and the true positives analyzed by security operators can be reinjected in the training data to reduce the training bias. However, this method has a major flaw since security operators inspect only the alerts, not the other predictions: some false negatives caused by the initial training bias may never be detected.

The best solution to avoid training bias is to perform *in-situ* training [132]. This method requires security administrators to annotate data directly from the production environment. This way annotated dataset is perfectly representative of the deployment environment, but it is expensive since it requires to annotate data manually. We propose a solution to reduce the cost of *in-situ* training in Part II.

In brief, security administrators must carry out these three evaluation steps thoroughly before any deployment. We want to emphasize that evaluation should not be reduced to analyzing numerical performance measures such as false alarm and detection rates. Security administrators must conduct a more in-depth analysis of the model behavior to diagnose potential training biases.

2.6 Conclusion

In this chapter, we present the whole machine pipeline that security administrators must set up to build supervised detection models ready for deployment. We focus more particularly on two steps: training and evaluation.

We provide a broad overview of supervised model classes and we explain how security administrators should choose a model class that suits their operational constraints. We point out two constraints that are critical factors in the selection process: transparency and robustness.

Transparency matters both to security operators and administrators. Security operators need information to exploit the triggered alerts. As for security administrators, they need to understand the detection model behavior before deployment to diagnose potential training biases. In this chapter, we present model classes that are inherently transparent: k-nearest neighbors, linear models (e.g. logistic regression, SVM) and tree-based models (e.g. decision trees, random forests). Moreover, model-agnostic interpretability methods can be exploited to interpret any model class, even the most complex.

Robustness is also crucial since detection models are deployed in adversarial environments where attackers are willing to circumvent them. Supervised detection models are less prone to evasion attempts than misuse detection techniques since they are more generic, but robustness should still be taken into account. Many research works focus on making detection models robuster to evasion attempts, but it remains an open problem. There is still no consensus to provide a solution ready to apply by security administrators.

This chapter places a particular emphasis on evaluating properly supervised detection models before deployment. It introduces a three-step evaluation procedure that security administrators should carry out thoroughly before any deployment. Each step explains not only how to diagnose a potential problem, but also how to address it.

In brief, this chapter provides methodological guidance to help security administrators build supervised detection models that suit their operational constraints. In the next chapter, we introduce DIADEM (DIAgnosis of DEtection Models) an interactive visualization tool that we have designed and implemented to help security administrators apply the methodology presented in this chapter.

Chapter 3

Build and Diagnose Detection Models with DIADEM

Security administrators are interested in machine learning to increase detection capabilities, but they usually have little knowledge about this data analysis technique. Besides, they do not want to deal too much with the machine learning machinery, they would rather focus mainly on detection.

In this chapter, we present DIADEM, an interactive visualization tool we have designed and implemented to help security administrators apply the methodology set out in the previous chapter. DIADEM deals with the machine learning machinery, and includes a graphical user interface to diagnose and handle potential accuracy issues. Besides, we illustrate how security administrators can leverage DIADEM to set up detection models with a case study on malicious PDF files detection. We provide an open-source implementation of DIADEM in SecuML [20] to enable security administrators to set up their own detection models.

This chapter content has been published in French at the *Symposium sur la sécurité des technologies de l'information et des communications* (SSTIC 2017) [26].

Contents

3.1 Introduction

Security administrators are willing to deploy supervised detection models in their detection systems to strengthen detection. Since they usually have little or no knowledge about machine learning, they need tools to help them build and diagnose supervised detection models. These tools should deal with the machine learning machinery to let security administrators focus mainly on detection.

Many libraries dedicated to machine learning (e.g. scikit-learn in python, Spark, Mahout or Weka in java, or Vowpal Wabbit in C++) allow to train various models with a low workload. These libraries are a crucial asset to encourage domain experts to apply machine learning even if they have little or no knowledge about this data analysis technique. However, they do not offer user interfaces to analyze the models behavior while it is crucial to diagnose and address potential accuracy issues before deployment.

Online services such as Google Cloud ML, Microsoft Azure, or Amazon Machine offer visualization solutions, but they require to store the data in the cloud which is incompatible with detection systems.

Some research works have introduced user interfaces to ease the application of machine learning by non-experts [6, 73, 86, 135]. Nonetheless, these publications present only screen shots of the user interfaces. They do not provide any implementation which is a strong barrier for their wide use [33, 85, 145].

To address these shortcomings, we have designed and implemented DIADEM, which stands for DIAgnosis of DEtection Models. DIADEM helps security administrators apply the methodology presented in Chapter 2. It allows to build supervised detection models with little machine expertise. It includes a graphical user interface to diagnose potential accuracy issues and to find solutions to address them. DIADEM is a generic solution that can be used on any detection problem. It is not intended for production, but rather to set up detection models before deployment.

This chapter presents the following contributions:

- We present DIADEM, a tool that helps security administrators build and analyze supervised detection models before deployment. The diagnosis interface provides information about the global behavior of detection models. It also enables more in-depth analysis of individual predictions.
- We illustrate how security administrators can leverage DIADEM to set up detection models with a case study on malicious PDF files detection.
- We provide an open-source implementation of DIADEM in SecuML [20] to enable security administrators to build and diagnose their own detection models.

3.2 DIADEM: a Tool to Diagnose Supervised Detection Models

We have designed and implemented DIADEM (DIAgnosis of DEtection Models) to help security administrators set up the machine pipeline that has been presented in Section 2.2.4. It allows to train detection models easily. Moreover, it includes a user interface to evaluate detection models

according to the protocol introduced in Section 2.5. We have released DIADEM as an open-source software in SecuML [20].

3.2.1 Running DIADEM

Security administrators must perform the first two steps of the machine learning pipeline : data annotation and feature extraction. They must provide as input to DIADEM a set of annotated instances represented as fixed-length vectors of features. Data annotation and feature extraction are specific to each detection problem, that is why security administrators must complete them beforehand.

In this section, we explain how DIADEM carries out the training and evaluation steps of the machine learning pipeline.

Training

DIADEM offers several model classes (e.g. naive Bayes classifiers, logistic regression, SVM, decision trees, random forests, and gradient boosting). Its modular design allows to easily add support for other model classes if necessary. The reader may refer to Section 2.4 for advice on the choice of the model class.

DIADEM relies on scikit-learn [101] to train the detection models. However, training a detection model requires some pre-processing steps: *feature standardization*, and *setting the hyperparameters*. In contrast with scikit-learn, DIADEM performs these pre-processing steps automatically to conceal some of the machine learning machinery to security administrators.

Feature Standardization. Standardization (or Z-score normalization) rescales the features so that they have the properties of a standard normal distribution with $\mu = 0$ and $\sigma = 1$ where μ is the mean and σ is the standard deviation from the mean.

Standardizing the features, so that they are centered around 0 with a standard deviation of 1, is not only important if we are comparing measurements that have different units, but it is also a general requirement for many machine learning algorithms [107].

DIADEM standardizes the features before training. This way, security administrators cannot forget to perform this critical pre-processing step.

Setting the Hyperparameters. Many model classes have hyperparameters whose values must be set before the training process begins (see Section 2.3.3). DIADEM sets the values of the hyperparameters automatically through a grid-search cross validation optimizing the AUC (see Section 2.2.2).

For example, DIADEM sets the hyperparameters of logistic regression models automatically: the penalty norm Ω, and the regularization strength c (see Section 2.3.1). The penalty norm is either ℓ_1 or ℓ_2, and the regularization strength c is selected among the values $\{0.01, 0.1, 1, 10, 100\}$.

Evaluation

DIADEM offers several ways to set the validation dataset used for evaluation. Security administrators can provide two separate annotated datasets, one for training and one for evaluation, or a single annotated dataset and DIADEM splits it up. In the latter case, security administrators can

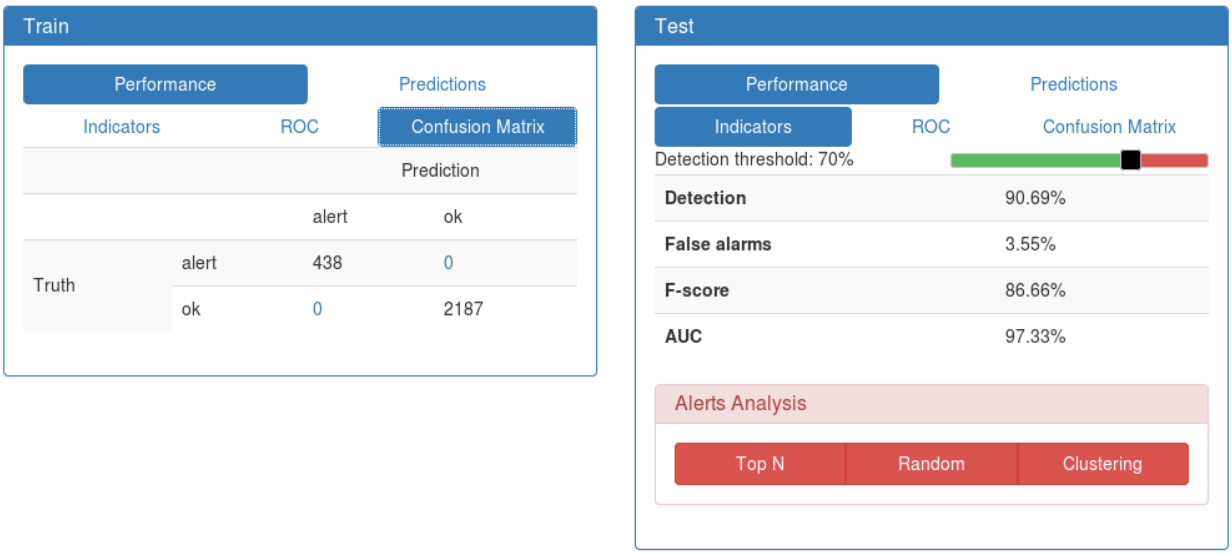


Figure 3.1: Visualization of the Model Performance with DIADEM.

specify how the annotated dataset should be split : randomly or temporally. The reader may refer to Section 2.5.2 for more information about ways to generate validation datasets.

DIADEM is launched with a single command line where security administrators must specify the model class they want to train, and the way the validation dataset should be set. Once DIADEM has trained the detection model and applied it to the validation dataset, security administrators can analyze its performance and behavior in DIADEM web user interface.

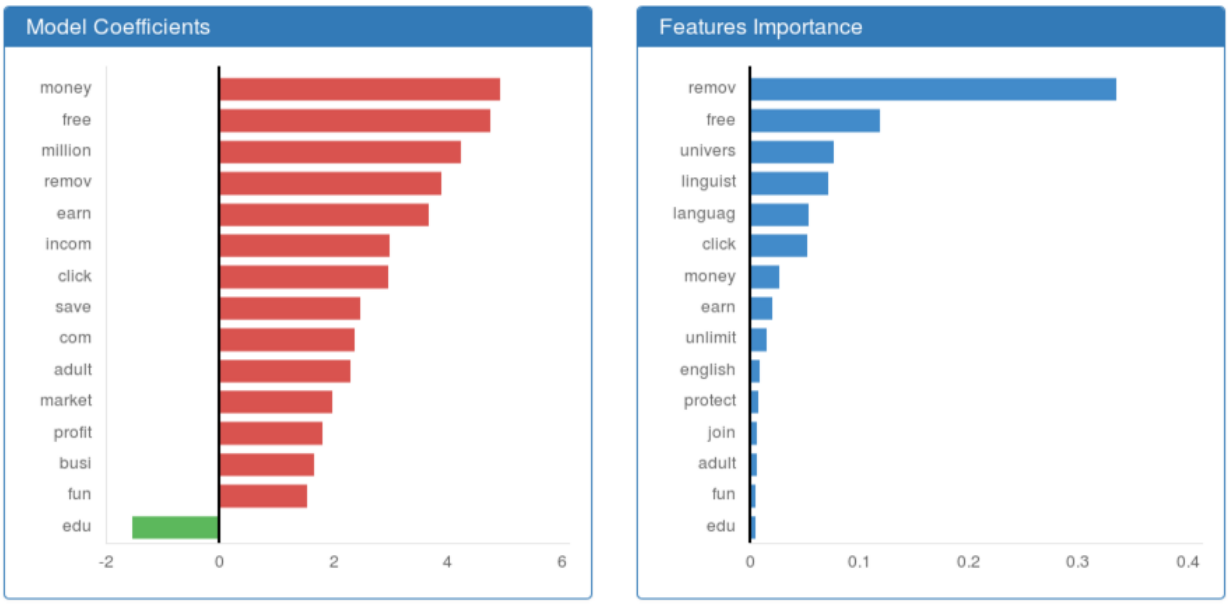
DIADEM primary purpose is to build detection models and to assess their performance with fully annotated datasets. Nonetheless, DIADEM can also train a model, and apply it to an unlabeled dataset. In this case, the diagnosis interface still displays some monitoring information, but DIADEM cannot assess the performance of the detection model on the validation dataset.

3.2.2 DIADEM Diagnosis Interface

The graphical interface displays the necessary elements to carry out the diagnosis steps introduced in Section 2.5. DIADEM provides visualizations for the global behavior of detection models, but also for individual predictions. Moreover, it allows to analyze the alerts triggered on the validation dataset.

High-Level Analysis of Detection Models

Visualization of the Model Performance. DIADEM displays the performance evaluation of the detection model both on the training and validation datasets (see Figure 3.1). This way, security administrators can diagnose both underfitting (see Section 2.5.1) and overfitting (see Section 2.5.2).



(a) Features Coefficients of Linear Models.

(b) Features Importance of Tree-based Models.

Figure 3.2: Visualization of the Model Behavior with DIADEM.

Performance evaluation panels display the performance metrics presented in Section 2.2.2. They display the confusion matrix, the ROC curve, and the detection and false alarm rates for a given detection threshold. Security administrators can change the value of the detection threshold through a slider to see the impact on the detection and false alarm rates.

Visualization of the Model Behavior. DIADEM displays information about the global behavior of detection models. This visualization allows security administrators to grasp how detection models make decisions and to diagnose potential training biases (see Section 2.5.3).

This visualization is currently implemented for linear (see Figure 3.2a) and tree-based models (see Figure 3.2b). DIADEM does not yet support model-agnostic interpretation methods (see Section 2.4.3).

Thanks to these graphic depictions, security administrators can focus on the most influential features of the detection model. They can click on a given feature to access to its descriptive statistics on the training data. These statistics allow security administrators to understand why a feature has a significant impact on decision-making, and may point out biases in the training dataset.

Analysis of Individual Predictions

Security administrators can examine individual predictions with the diagnosis interface. They can click on the confusion matrix to review the false positives and negatives.

Besides, DIADEM displays the histogram of the predicted probabilities of maliciousness. Security administrators can exploit this histogram to analyze the instances whose predicted probability is within a given range. For instance, they can review the instances close to the decision boundary

(a) All the Features.

(b) Most Important Features.

Figure 3.3: Default *Description* Panel for Spam Detection.

(probability of maliciousness close to 50%) to understand why the detection model is undecided. Moreover, they can inspect instances that have been misclassified with a high level of confidence. The analysis of these individual predictions can point out potential annotation errors, or help security administrators find new discriminating features.

Description Panel. DIADEM displays each instance in a *Description* panel. Figure 3.3 depicts the default *Description* panel for spam detection.

By default, the *Description* panel displays the features of the instance. This visualization may be hard to interpret especially when the feature space is in high dimension. Figure 3.3a displays the features extracted from an email: the number of occurrences of each word in the vocabulary. Since the vocabulary contains 1000 words, this visualization is hardly interpretable for humans.

If an interpretable model has been trained, DIADEM also displays the features that have the most impact on the prediction (see Figure 3.3b). This visualization is easier to interpret than the previous one since the features are sorted according to their impact in the decision-making process.

Other visualizations specific to the detection problem may be more relevant to analyze individual predictions. In order to address this need, DIADEM enables security administrators to plug problem-specific visualizations.

Problem-Specific Visualization. Custom visualizations should be easily interpretable by security administrators and display the most relevant elements from a detection perspective. They may point to external tools or information to provide some context. Security administrators can implement several custom visualizations for a same data type to show the instances from different angles.

Figure 3.4 depicts a problem-specific visualization we have implemented for spam detection. It displays simply the raw content of the email. We strongly encourage security administrators to provide convenient problem-specific visualizations, since they can significantly ease the analysis of individual predictions.

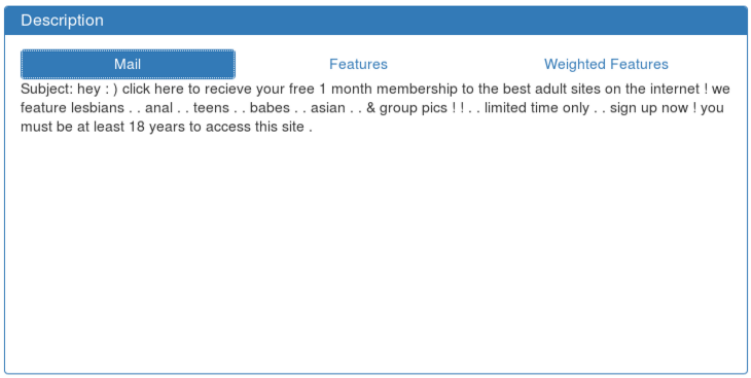


Figure 3.4: Problem-Specific Visualization for Spam Detection.

Analysis of the Triggered Alerts

DIADEM provides a graphical user interface to analyze the triggered alerts. This interface is similar to a supervision interface used by security operators to exploit the triggered alerts. DIADEM offers such a visualization to allow security administrators check the exploitability of the alerts before deployment.

Security administrators must first choose a detection threshold. Then, DIADEM alert interface offers three views of the triggered alerts. It can display the top N, or some randomly selected alerts, or cluster all the alerts to ease their analysis.

Alerts Clustering. Clustering similar alerts can significantly help security operators: they can take advantage of the analysis of alerts belonging to the same cluster. Accordingly, DIADEM provides solutions to cluster alerts.

If the malicious instances of the training dataset have been tagged with a malicious family, DIADEM can leverage this information to regroup the alerts according to the alert taxonomy set up by security administrators. In this case, DIADEM trains a multi-class classification model on the malicious instances to tag the triggered alerts automatically.

If the annotated dataset does not contain information about the malicious families, DIADEM can nonetheless cluster the alerts, but in an unsupervised way. In this case, the clustering is unlikely to regroup instances according to user-defined families.

To sum up, DIADEM is a generic solution that helps security administrators set up detection models before deployment. DIADEM can be applied to any detection problem on any data type. In the next section, we illustrate how security administrators can leverage DIADEM to build and diagnose a PDF detection model.

3.3 Case Study: Malicious PDF Files Detection

The PDF format is an open document description format created by the Adobe company in 1993 in order to preserve a document page layout regardless of the computer platform being used.

PDF viewers are available on many computer platforms. The PDF format is therefore widely used in most organizations to create and exchange electronic files. Besides, the PDF format is highly sophisticated: the technical specifications exceed the 1,300 pages in length, and it depends on many third-party libraries. As a result, vulnerabilities related to the PDF format are often discovered which makes it even more attractive for attackers.

In this section, we explain how to leverage DIADEM to build a supervised detection model of malicious PDF files. First of all, we prepare the input data for DIADEM: we collect an annotated dataset (see Section 3.3.1) and we extract features (see Section 3.3.2). Then, we run DIADEM on this annotated dataset (see Section 3.3.3), and we analyze the resulting detection model with DIADEM visualization interface (see Section 3.3.4).

3.3.1 Data Annotation

It is easy to collect a training dataset of PDF files, because this data format is popular and commonly misused to spread malicious code. This case study relies on two annotated datasets : Contagio [1] (9,000 benign files and 11,001 malicious files) and WebPDF (2,078 benign files and 767 malicious files). Contagio is a public annotated dataset used in various research works. WebPDF consists of benign files resulting from requests on the Google search engine, and of malicious files queried from the VirusTotal [3] platform.

In the PDF case study, we assume that we have only access to the Contagio dataset to set up the detection model before deployment. The WebPDF dataset simulates PDF files encountered in a deployment environment.

3.3.2 Feature Extraction

Feature extraction is a three step-process: 1) identifying discriminating information, 2) parsing the data to extract this information, and 3) transforming this information into fixed-length vectors of features. In this section, we explain how we perform each step for the PDF case study.

Identifying and Parsing Discriminating Information

PDF files are composed of metadata (e.g. author, producer, title, creation and modification dates) and of objects of different types. The objects can contain text, images, videos, or even JavaScript code. PDF files are hierarchically structured: the objects reference each other, they form a graph that may contain cycles.

In most cases, attackers forge malicious PDF files to exploit vulnerabilities that allow to execute arbitrary code on victims' computer platforms. For instance, some JavaScript code can exploit a vulnerability of the JavaScript engine included in the PDF viewer, or a TTF font can exploit a vulnerability of the operating system.

We list a few elements that can be examined to identify malicious PDF files : typical features linked to the triggering of vulnerabilities (e.g. JavaScript code, or OpenAction functions), presence of a malicious payload (e.g. shellcode), functions aiming to evade detection (e.g. multiple encoding technologies, or concealment of objects), or the aspect of the file which is more or less realistic (e.g. malformations, few pages, or few objects).

In the PDF case study, we leverage a PDF parser to extract the discriminating information identified. More generally, parsers are already deployed in detection systems for traditional detec-

tion methods (e.g. signatures or expert systems). Feature extraction can also exploit them to build machine learning-based detection models.

Generating Fixed-Length Vectors of Features

In this section, we present a few common techniques to transform discriminating information into fixed-length vectors of features, and we give practical examples based on the PDF case study.

Some information contained in PDF files can be directly inserted into a fixed-length vector of features : the size of the file, the creation and modification dates can be transformed into timestamps. Other information such as the author or the objects are not binary, numerical, or categorical values. They must be transformed before being inserted in the vector of features. Moreover, PDF files have a variable number of objects, and this information should be represented as a fixed-length vector of features.

Numerical Lists of Variable Sizes. Standard training algorithms do not take numerical lists of variable sizes as input. They must be transformed into fixed-length vectors of features beforehand. The usual method is to compute statistical indicators such as the mean, the standard deviation, the number of elements, or the extrema.

In the PDF case study, we have extracted the size of the objects composing PDF files which is a numerical list of variable size since PDF files do not always have the same number of objects. We have computed the mean, the standard deviation, the minimum, and the maximum of this list.

Character String. Discriminating information can take the form of character strings in the raw data. A classic way to handle with this type of data is to transform them into a fixed-length numerical vector where each feature corresponds to a family of characters (e.g. uppercase letters, lowercase letters, digits). The value of each feature is the number of occurrences, or the proportion, of the family of characters in the string.

Authors of PDF files are character strings that we have transformed into seven numerical features: the size of the string, the number of lowercase letters, uppercase letters, digits, occurrences of the character '`.`', spaces, and special characters.

Categorical Information. The type of PDF objects is an example of categorical variable that includes the values `Image`, `Text` and `Police` (there are other possible values, but we restrict the set of values in this example). These three values cannot be ordered, and we cannot define a similarity measure or a distance metric between them.

Some classification model classes accept categorical features as input (e.g. tree-based models such as decision trees or random forests), while others require to transform them into binary or numerical features beforehand(e.g. SVM, logistic regression). The simplest method to transform categorical features into numerical values is to associate a number to each category (e.g. `Image` $\rightarrow 0$, `Text` $\rightarrow 1$, and `Police` $\rightarrow 2$). However, this method is unsuitable since learning algorithms rely on distances between features. With this method, learning algorithms would interpret the categories as being ordered while it may not be the case.

A better solution is to transform a categorical feature with m possible values into m binary features with only one active (e.g. `Image` $\rightarrow [1,0,0]$, `Text` $\rightarrow [0,1,0]$, `Police` $\rightarrow [0,0,1]$). This transformation technique can be extended to lists of categorical features by counting the number of occurrences of each value (e.g. $[\texttt{Police}, \texttt{Text}, \texttt{Image}, \texttt{Image}, \texttt{Text}] \rightarrow [2,2,1]$).

In the PDF case study, we have extracted 113 numeric features from PDF files. These features are similar to those presented by Smutz and Stravou [124, 125].

In Sections 3.3.1 and 3.3.2, we have explained how to get an annotated dataset of PDF files ready to use for DIADEM. In the next section, we detail how we run DIADEM on this annotated dataset.

3.3.3 Running DIADEM on the PDF Annotated Dataset

In the PDF case study, we have access only to Contagio to set up a PDF detection model. This dataset does not have any timestamp information. The dates available in the metadata (creation and modification dates) are not relevant to timestamp the PDF files. Indeed, these dates do not correspond to the first date of appearance of the files. Besides, attackers can arbitrarily set their values. As a result, we ask DIADEM to split the Contagio dataset into a training and a validation dataset randomly: 90% of the data (Contagio_90%) is used for training, and the remaining 10% (Contagio_10%) for validation.

We ask DIADEM to train a logistic regression on the annotated dataset by following the advice given in Section 2.4. If this model class is too simple to discriminate malicious from benign PDF files, we will move toward a more complex model class.

In brief, we launch DIADEM with the following command line:

```
SecuML_DIADEM Pdf Contagio LogisticRegression --validation random --test-size 0.1.
```

Once DIADEM has trained the detection model on Contagio_90% and applied it to Contagio_10%, we can analyze its performance and behavior with the DIADEM diagnosis interface.

3.3.4 Diagnosis of the PDF Detection Model with DIADEM

In this section, we leverage DIADEM visualization interface (see Section 3.2.2) to diagnose the PDF detection model by following the steps presented in Section 2.5.

Problem-Specific Visualizations

We have implemented two problem-specific visualizations for PDF files to ease the analysis of individual predictions. The first view represents the PDF file as raw text, while the second one represents its graphs of objects.

Diagnosis of Underfitting and Overfitting

DIADEM allows to diagnose whether the detection model suffers from underfitting (see Section 2.5.1) thanks to the *Training* panel. In the PDF case study, it shows that the training AUC is 99.85%: there is no underfitting. Logistic regression is a model class complex enough to discriminate the malicious PDF files from the benign ones. There is no need to move toward a more complex detection model class.

Besides, DIADEM allows to diagnose if the detection model suffers from overfitting (see Section 2.5.2) thanks to the *Testing* panel. It reveals that the validation AUC of the PDF detection

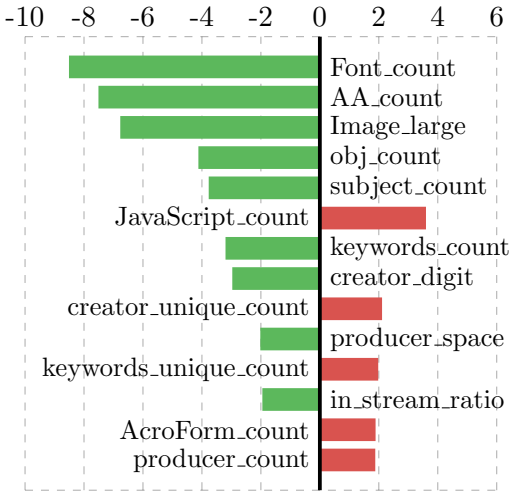


Figure 3.5: Coefficients Associated to the Most Influential Features of the PDF Detection Model.

model is 99.68%. The PDF detection model is able to predict accurately the label of instances unseen during the training phase.

To sum up, the detection model passes the first two diagnosis steps.

Diagnosis of Training Bias

This diagnosis step requires a more in-depth analysis of the behavior of the detection model. DIADEM displays the features that have the greatest impact on decision-making. Figure 3.5 depicts the coefficients of the logistic regression model trained on Contagio.

The number of embedded fonts (`Font_count`), the number of automatic actions (`AA_count`), the number of large images (`Image_large`) and the number of objects (`obj_count`) are among the most influential features, and they are associated to negative coefficients. The model tends therefore to consider that files with much content are benign. Conversely, the detection model considers that the presence of JavaScript is an indicator of maliciousness since the feature `JavaScript_count` is associated to a positive coefficient.

DIADEM displays the distributions of the most important features. The analysis of these distributions emphasizes that the malicious PDF files of Contagio are often small and simple, while the benign PDF files are more complex. Indeed, most malicious PDF files of Contagio contain only a malicious JavaScript payload that is executed on opening. These very simple files, consisting of a single page without images or fonts, are very different from the benign PDF files which are more complex with many objects, images and embedded fonts.

In Contagio, almost all the malicious files contain some JavaScript code, while hardly any benign files include such content. Besides, the vast majority of the malicious files in Contagio do not contain any content such as text or image objects.

These discrepancies are relevant with the detection target, but they are too pronounced in the Contagio dataset. As a result, the detection model trained on Contagio_90% considers almost that

the presence of JavaScript is a sufficient criterion to trigger an alert, as well as a reduced content. Contagio is a stereotyped dataset that lead to a training bias.

In order to emphasize this training bias, we have launched DIADEM with Contagio as training dataset, and WebPDF as validation dataset. It shows that the detection model is not as efficient on WebPDF as on Contagio_10%: the AUC on WebPDF is only 89.63% while it is 99.68% on Contagio_10%. DIADEM allows to inspect the false positives and negatives on WebPDF. Our analyses show that many false positives are due to the presence of JavaScript code in benign files. Besides, most false negatives are malicious PDF files whose malicious payload is concealed among a rich content.

In brief, DIADEM diagnosis interface displays the necessary information to diagnose training biases. First, security administrators should analyze the features that have the greatest impact on decision-making. Then, they decide whether they are relevant with the detection target, or they reveal a training bias. DIADEM can also display the distributions of the most important features to assist security administrators in their analysis.

A solution to address the training bias is to get a training dataset less stereotyped than Contagio. The training dataset should contain more ambivalent instances (e.g. malicious PDF files with richer content, and benign files with JavaScript code), to get a smarter and more subtle detection model.

3.4 Conclusion

This chapter presents DIADEM, a tool we have designed and implemented to help security administrators build and diagnose supervised detection models before deployment. DIADEM deals with the machine learning machinery (e.g. feature standardization, setting of the hyperparameters) to let security administrators focus mainly on detection. Besides, its diagnosis interface helps security administrators apply the three-step evaluation procedure exposed in Section 2.5.

The PDF case study conducted in this chapter illustrates the whole machine learning pipeline (data annotation, feature extraction, training, evaluation) and demonstrates how security administrators can leverage DIADEM to perform the training and evaluation steps. This case study shows how DIADEM diagnosis interface is helpful to carry out the evaluation procedure. Besides, it emphasizes how important it is to analyze the model behavior to detect potential training biases. Evaluation must not be reduced to analyzing numerical performance measures such as false alarm and detection rates. Security administrators must conduct a more in-depth analysis of the model behavior before deployment.

DIADEM helps security administrators with two steps of the machine learning pipeline: training and evaluation. The first two steps, data annotation and feature extraction, are left to security administrators. They must provide as input to DIADEM a set of annotated instances represented as fixed-length vectors of features.

The following parts of this thesis focus on these two steps that can be difficult and tedious to perform for security administrators. Part II introduces an end-to-end active learning system, ILAB, to help security administrators annotate datasets with a reduced effort. Part III examines

automatic features generation as a means to help security administrators extract discriminating features.

Part II

End-to-End Active Learning for Detection Systems

Chapter 4

Active Learning: Related Work and Problem Statement

The performance of supervised detection models depends deeply on the quality of training data. Good training datasets are, however, extremely difficult and expensive to acquire in the context of computer security detection.

Active learning has been introduced in the machine learning community to reduce human effort. A strategy asks the expert to annotate only the most informative examples to minimize the number of manual annotations.

In this chapter, we provide an overview of how active learning can be leveraged in detection systems. Then, we review the literature and we show the limits of existing approaches. Finally, we formalize the problem that this part of the thesis aims to address.

Contents

4.1 Introduction

The performance of supervised detection models depends deeply on the quality of the training data, but good training datasets are extremely difficult to acquire in the context of threat detection. Some annotated datasets related to computer security are public (e.g. Malicia project [91], KDD99 [136], kyoto2006 [128]) but they quickly become outdated and they often do not account for the idiosyncrasies of each deployment context.

Annotated datasets of files (e.g. portable executable, PDF, Windows Office documents) can be exploited to train detection models deployed in diverse environments. These kinds of data are likely to vary slightly from one environment to another. On the contrary, event logs (network or operating system event logs) and detection targets are highly dependent on deployment environments: a same behavior can be legitimate in an environment, but irregular in another.

As a result, a detection model trained for a given environment is likely to perform poorly in another. These types of data require to build detection models *in-situ* [132] with training data coming from production environments.

Security administrators can deploy *annotation systems* to build representative training datasets *in-situ*. The annotation system picks some instances from a pool of unlabeled data originating from the deployment environment, displays them to security administrators, and gathers their answers.

Security administrators are essential for annotating but they are an expensive resource. The labeling process must thus exploit their time efficiently. *Active learning strategies* [119] have been introduced in the machine learning community to reduce human effort. They select only the most informative examples to minimize the number of manual annotations.

In this thesis, we define an *active learning system* as an *annotation system* that leverages an *active learning strategy* to select the instances to be annotated. It is crucial to design both components, the active learning strategy and the annotation system, jointly to effectively reduce experts annotation effort. Security administrators do not want to minimize only the number of manual annotations, but the overall time spent annotating.

The rest of the chapter is organized as follows. First, Section 4.2 provides an overview of active learning in detection systems and Section 4.3 presents some related works with regard to annotation systems, active learning strategies, and applications to computer security detection systems. Then, Section 4.4 introduces the notations and formalizes the problem. Finally, Section 4.5 summarizes the contributions of our end-to-end active learning system, ILAB, that will be presented in detail in Chapters 5 and 6.

4.2 Overview of Active Learning in Detection Systems

Active learning systems are annotation systems that leverage active learning strategies to select the instances to be annotated. They rely on an interactive process where a domain expert is asked to annotate some unlabeled instances to improve the performance of the supervised model (see Figure 4.1). The active learning strategy queries the most informative instances to reduce the number of manual annotations. As for the annotation system, it displays the annotation queries and gathers the answers. It should be tailored to the needs of domain experts who perform the annotations.

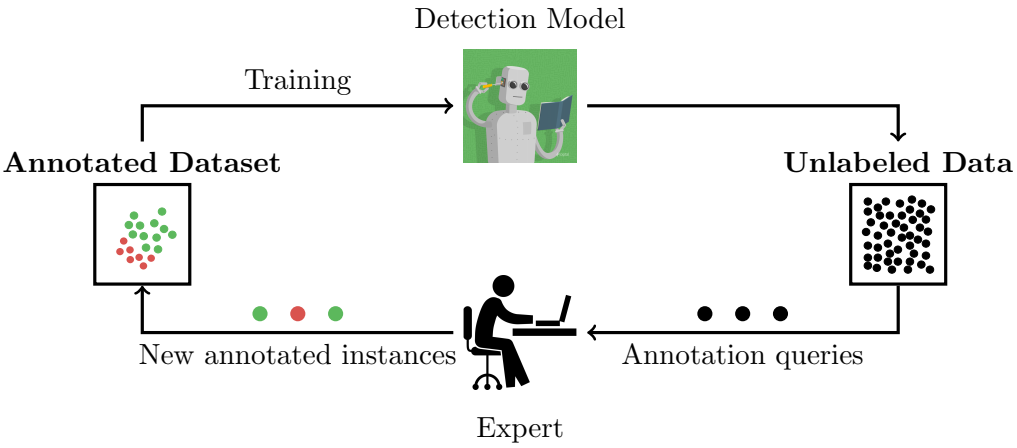


Figure 4.1: Active Learning: An Interactive Process.

In the context of detection systems, annotating consists in assigning a binary label, malicious or benign, and optionally a family detailing the binary label. Instances sharing the same family behave similarly and have the same level of criticality. For example, malicious instances belonging to the same family may exploit the same vulnerability, they may be polymorphic variants of the same malware, or they may be emails coming from the same spam campaign. The malicious families correspond to the alert taxonomy set by security administrators (see Section 1.1).

DIADEM can leverage the malicious families to cluster the alerts according to the alert taxonomy (see Section 3.2.2). This way, security operators can take advantage of the analysis of alerts belonging to the same cluster. Clustering alerts based on an alert taxonomy significantly helps security operators, and thus makes detection systems more efficient.

There are two active learning scenarios of interest for detection systems: *pool-based active learning* and *stream-based active learning*. Pool-based active learning can be used to acquire annotated datasets and deploy initial detection models. As for stream-based active learning, it can update already deployed detection models over time to follow threat evolution.

4.2.1 Pool-based Active Learning to Build Initial Detection Models

Pool-based active learning queries instances to be annotated from a pool of unlabeled data that does not change throughout the annotation process [119]. At each iteration, the active learning strategy queries some instances from the unlabeled pool for annotation. The instances that have not been selected at a given iteration can be picked afterward.

When security administrators do not have access to good training datasets, they can leverage pool-based active learning in annotation projects. Unlabeled data are usually easy to acquire from deployment environments. Pool-based active learning is therefore well-suited to build representative training datasets *in-situ*.

Security administrators should perform the annotations, or at least supervise the annotation process. One can be surprised that security operators do not execute this task, since annotating is closely related to checking whether an alert is a true or a false positive. Annotations are, however,

directly linked to the definitions of detection targets and alert taxonomies, which are under the responsibility of security administrators (see Section 1.1).

At the beginning of annotation projects, the detection target and the alert taxonomy are usually not perfectly delineated. Security administrators have usually vague specifications in mind that they refine as they examine new instances queried by the pool-based active learning strategy.

As a result, security administrators must be deeply involved in annotation projects that aim to build initial detection models. The annotation process determines both the detection target and the alert taxonomy.

4.2.2 Stream-based Active Learning to Follow Threat Evolution

Stream-based active learning queries instances to be annotated from a stream of unlabeled data [119]. Each time a new instance comes up, the active learning strategy decides whether to query it for annotation or to discard it. In this context, discarded instances cannot be queried for annotation afterward.

Security administrators can leverage stream-based active learning to update deployed detection models in order to follow threat evolution. In this scenario, a supervised detection model has already been deployed and security operators analyze the triggered alerts. The detection target and the alert taxonomy are therefore well specified.

Security operators can answer some annotation queries with the alerts they analyze, but security administrators must also be involved. First, the strategy may query supposedly benign instances that are not analyzed by security operators, and therefore security administrators must annotate them. Second, they should always review the annotations that are leveraged to update a detection model since they may change the detection target.

When a detection model is updated online through stream-based active learning, it is critical to monitor its evolution. Adding new annotated instances can significantly change the model behavior, and security administrators must make sure that the detection model converges properly.

In this thesis, we focus on pool-based active learning to build initial detection models, and we assume that a single security administrator performs the annotations. We aim to design and implement an end-to-end active learning system with security administrators at its core. We consider not only the active learning strategy, but also its integration in an annotation system.

In the next section, we review some related work about annotation systems, active learning strategies, and applications to computer security detection systems.

4.3 Related Work

4.3.1 Annotation Systems

Guidelines for Annotation Systems. Annotating data manually may be needed to train a supervised model, but it is a tedious work. Appropriate annotation systems can streamline annotation projects. Amershi et al. [5] and Settles [119] have described generic guidelines that any annotation system should follow.

First of all, annotation systems must **offer an ergonomic user interface** to display the queries and to gather the corresponding answers. Special care shall be taken to the ergonomics of the interface as it may greatly impact the overall annotation time.

Besides, annotation systems are intended for application domain experts who are likely to have no or little knowledge about machine learning. Consequently, the user interface must be **accessible to non-machine learning experts**. In particular, it should not contain words belonging to the machine learning jargon.

Moreover, people will invest time to improve their model only if they view the task as more beneficial than costly. Annotation systems must **provide feedback to users to show them the benefit of their annotations**, and that they are on track to achieve their goal.

At each iteration, the integration of new annotations improves not only the performance of the detection model but also the relevance of the following queries. Frequent updates are thus beneficial, but they must be performed efficiently to **minimize waiting-periods**. Annotation systems with long waiting-periods alter the expert-model interaction and are unlikely to be accepted by experts.

Structured Labeling [77]. Machine learning is based on the idea that similar inputs should have similar outputs. Annotators must thus provide consistent labels to avoid degrading the performance of the resulting classification model. Kulesza et al. [77] have introduced *structured labeling* to help annotators define and refine their *concept*, i.e. the abstract notion of the target class annotators are labeling for, as they annotate data. Thanks to structured labels, annotators can organize their concept definition by grouping and tagging data. Structured labeling increases label consistency by helping annotators recall labeling decisions. The structure is malleable (annotators can create, delete, split and merge tags), it is well suited for situations where annotators are likely to frequently refine their concept definition as they observe new data.

In the context of detection systems, the *concept* corresponds to the detection target, i.e. the abstract notion of benign behavior, and suspicious behaviors that should trigger alerts. Besides, we can draw a parallel between the tags defined in structured labeling and benign and malicious families. Structured labeling can be very convenient in annotation projects aiming to build computer security detection models (see Section 4.2.1). Indeed, at the beginning of annotation projects, security administrators have a vague idea of their detection target, and it may evolve throughout the annotation process. Some annotation queries may puzzle them: they may wonder whether an alert should be triggered or not. Some annotation queries can even question previous annotations.

Real-World Annotation Systems. Some research works have introduced whole annotation systems but they are especially designed for image [77, 122], video [14] or text [123, 36, 15, 105] annotations. In computer security, several kinds of data (e.g. PDF files, Windows Office documents, NetFlow, pcap, or event logs) are processed by detection systems. As a result, the annotation system must be generic and flexible enough to operate with these diverse data types.

How to Select the Instances for Annotation ? Many annotation systems select the instances to be annotated randomly, without leveraging active learning. There are some reassessment about the benefit of active learning strategies over random sampling [36]. Some consider it is not worth deploying active learning strategies in annotation systems: it may be complex and lead to a computation overhead.

Computer vision and natural language processing annotation projects can take advantage of low-cost annotators on crowd-sourcing market places such as Amazon Mechanical Turk[1]. In this scenario, there is no need for active learning since annotations are cheap. However, crowd-sourcing does not suit detection systems since the data they process are often sensitive, and annotating requires expert knowledge.

Moreover, active learning is crucial in the context of detection systems. The data is unbalanced, with a tiny portion of malicious instances, and the less common malicious behaviors are often the most interesting. If the annotation system queries instances selected uniformly, it is likely to query only benign and very common malicious behaviors.

To sum up, in the context of threat detection, annotation systems must leverage an active learning strategy to be effective. The active learning strategy should, nevertheless, be designed carefully to minimize the computation overhead.

4.3.2 Active Learning Strategies

Annotation systems can rely on an active learning strategy [119, 82, 139, 40] to maximize the impact of the annotations. A query strategy picks the most informative instances, i.e. the ones that lead to the best detection model, to minimize the number of manual annotations.

Search through the Hypothesis Space. Search through the hypothesis space is a broad category of active learning strategies that ask the expert to annotate the instances which provide the most information to improve a classification model. For instance, uncertainty sampling [82] and version space reduction [139] belong to this category and query the closest instances to the decision boundary. Query by committee [51] and error reduction [113] propose other utility measures to select the most informative instances which are more computationally expensive than uncertainty sampling and version space reduction.

The drawback of these active learning methods based on a utility measure is that they may introduce a *sampling bias*.

Sampling Bias. When active learning strategies focus only on the most informative instances, they may completely miss a family of observations. In this case, the overlooked family may have a negative impact on the performance of the detection model. The reader may refer to [116, 40] for a theoretical example.

Figure 4.2 provides an example of sampling bias in one dimension with uncertainty sampling [82] which queries the closest instances to the decision boundary. Each block represents a malicious or a benign family. With this data distribution, instances from the family M_1 are unlikely to be part of the initial training dataset, and so the initial decision boundary is likely to lie between the families B_2 and M_3. As active learning proceeds, the classifier will gradually converge to the decision boundary between the families B_2 and M_2 and will only ask the expert to annotate instances from these two families to further refine the decision boundary. The query algorithm completely overlooks the malicious family M_1 on the left as the classifier is mistakenly confident that the entire family is benign. As the malicious family M_1 is on the wrong side of the decision boundary, the classifier will not be able to detect this malicious family thereafter.

[1] `https://www.mturk.com/`

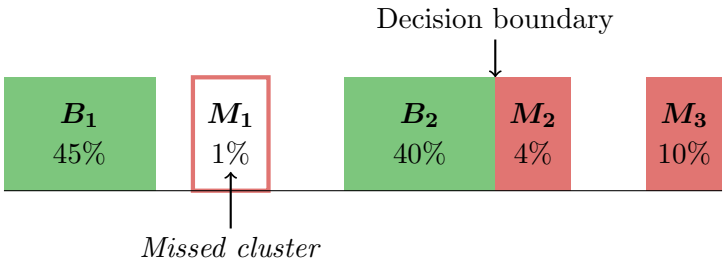


Figure 4.2: Sampling Bias Example.

Sampling bias is a significant problem for detection systems that may lead to malicious families remaining completely undetected. Besides, the risk of sampling bias is even higher for computer security detection than for other application domains because the initial annotations are not uniformly distributed. Uniform random sampling does not allow to acquire the initial annotated instances as the malicious class is too under-represented. Misuse detection techniques widely deployed in detection systems can provide initial annotations, but they likely all belong to the same family or to a small number of families.

Exploiting Structure in Data. In order to avoid sampling bias, a new kind of active learning strategies exploiting the structure in data has been introduced in the machine learning community [39, 152, 40]. For instance, Dasgupta et al. [40] have proposed to build a hierarchical clustering to annotate while exploring the dataset. This active learning strategy is efficient only if the clusters are aligned with the ground-truth labels, i.e. all their instances share the same class label. This is difficult to obtain in practice without any supervision. Building the hierarchical clustering with constraints based on initial annotations can ease this alignment [32]. Besides, if during the annotation process the clusters are not aligned with the ground-truth labels, the clustering can be rebuilt with constraints based on all the annotations performed by the expert so far. However, this active learning strategy has two practical issues. First, the method does not specify when the clusters should be rebuilt with the new constraints to ease the alignment of the clusters with the class labels. Second, each rebuilding of the clustering with the new constraints is highly time consuming since it takes into account all the instances. This leads to long waiting-periods for the expert who annotates, and thus damages the expert-model interaction.

Some works rely on rare category detection to avoid sampling bias [102, 68, 142, 130]. Categorical labels corresponding to families (a single benign family and several malicious families) are considered instead of malicious vs. benign binary labels, and the objective is to annotate at least one instance from each family. These approaches assume that all classification errors have the same cost while it is not the case in practice. Indeed, a misclassification between two malicious families is less severe than a misclassification between the benign family and a malicious family which corresponds to a false positive or a false negative. Besides, [142] does not scale to large datasets and [68] is unworkable on real-world annotation projects: it requires the number of families that should be discovered and their proportion to be known at the beginning of annotation projects.

Challenge. The real challenge for designing an active learning strategy is to avoid sampling bias while keeping a low computation cost. One the one hand, preventing sampling bias is crucial to

prevent the whole family from being misclassified. One the other hand, a low computation cost reduces the waiting-periods, and thus ensures a good expert-model interaction.

4.3.3 Applications to Computer Security Detection Systems

Triage of Network Alarms [7]. Amershi et al. have introduced CueT, an interactive machine learning system intended for computer security experts [7]. It helps them triage network alarms into existing alarm categories by making group recommendations via a ranked list accompanying confidence visualization. This system works in dynamic, continually evolving environments, but it has been specifically designed to triage network alarms. It is not a generic solution that can process any data type.

Stream-based Active Learning. Stream-based active learning (see Section 4.2.2) has been applied to computer security detection problems [147, 118, 143, 117, 87] to follow threat evolution. In this setting, the detection model in production has been initially trained on an annotated dataset representative of the deployment environment. In our case, such a representative annotated dataset is unavailable and the objective is to acquire it offline to train the initial detection model.

Pool-based Active Learning. Some works focus on pool-based active learning to build annotated datasets for detection systems. First, Almgren et al. [4] have applied plain uncertainty sampling [82] to intrusion detection before Schütze et al. [116] have introduced the sampling bias issue. Thereafter, many research works have applied active learning to computer security detection problems without taking sampling bias into account [90, 95, 93, 94, 92]. Moskovitch et al. [90] have applied plain version space reduction [139] and error reduction [113]. Nissim et al. [95] have introduced a new active learning strategy based on version space reduction that also queries the most malicious instances according to the detection model. They have then applied the same strategy to other data types: PDF files [93], Android applications [94], and Microsoft Office documents [92].

Some research works intend to avoid sampling bias. Aladin [130] and Görnitz et al. [63] have proposed new active learning strategies for intrusion detection that aim to discover the different malicious families. Aladin applies rare category detection [102] on top of active learning to foster the discovery of the different families, and Görnitz et al. rely on k-nearest neighbors to detect yet unknown malicious families. However, both approaches [130, 63] avoid sampling bias at the expense of the expert-model interaction. These strategies require heavy computations to generate the annotation queries. They cause long waiting-periods that experts cannot exploit.

User Experience is Often Overlooked. Most research works [4, 90, 95, 93, 94, 92, 63] have only run simulations on fully annotated datasets: an oracle answers the annotation queries automatically with the ground-truth labels. They have not set up their strategy in real-world annotation projects, and they have not mentioned any user interface.

Simulations consider that the annotation cost is the number of manual annotations whereas security administrators want to minimize the overall time spent annotating. The time required to compute the annotation queries is rarely monitored. It corresponds, nevertheless, to waiting-periods that should also be minimized. Besides, simulations ignore annotation interfaces while the way the instances are displayed impacts significantly the average annotation cost. Moreover, simulations assimilate annotators to mere oracles, while they are human experts. They are not

machines, they need feedback to understand the benefit of their annotations, otherwise they will stop annotating.

Stokes et al. [130] have carried out user experiments with computer security experts. Aladin includes a graphical user interface but the authors do not provide any detail about it. Besides, the interactions between the expert and the model are poor due to a high execution time. The expert is asked to annotate a thousand instances each day, and new queries are computed every night. Their solution reduces the waiting-periods, but it significantly damages the expert-model interaction since the expert feedback is integrated only once a day.

In brief, research works focus mostly on active learning strategies and not on their integration in annotation systems. User interfaces designed to set up active learning strategies in real-world annotation projects have, however, a significant impact on the overall user experience [119, 5, 47, 120] and on the actual application of such methods in practice [85, 15, 138].

4.4 Problem Statement

Out goal is to design and implement an end-to-end active learning system to help security administrators acquire representative annotated datasets with a reduced human effort. We consider that a single security administrator answers the queries of a pool-based active learning strategy. The security administrator who annotates the data is also referred to as *the expert* and *the annotator* throughout Part II.

We assume that there is no adversary attempting to mislead the annotation process: a trusted security administrator performs the annotations offline before the detection model is deployed in production. In this section, we introduce the notations used in Chapters 5 and 6, and we detail our objective.

4.4.1 Notations

Let $\mathcal{D} = \left\{x^i \in \mathbb{R}^m\right\}_{1 \leq i \leq n}$ be the dataset we want to annotate partially to train a supervised detection model $\mathcal{M}$. It contains n instances described by m real-valued features. For example, each instance $x^i \in \mathcal{D}$ could represent a PDF file, an Android application, the traffic of an IP address, or the activity of a user. Such unlabeled data are usually easy to acquire from the environment where the detection system is deployed (e.g. files, network traffic captures, or event logs).

Standard active learning strategies do not take raw data as input, but instances represented as fixed-length vectors of features (see Section 2.2.4). Many research works focus on feature extraction for given detection problems: Android applications [54], PDF files [37, 124], Windows audit logs [22], portable executable files [75]. In Chapters 5 and 6, we build upon these works, and we focus on reducing the cost of building a representative annotated dataset with an effective annotation system. All instances are represented by numeric vectors after feature extraction. As a result, active learning strategies are generic regarding the detection problem.

Let $\mathcal{L} = \{\texttt{Malicious}, \texttt{Benign}\}$ be the set of labels and $\mathcal{F}_y$ be the set containing the user-defined families of the label $y \in \mathcal{L}$. For example, malicious instances belonging to the same family may exploit the same vulnerability, they may be polymorphic variants of the same malware, or they may be emails coming from the same spam campaign.

Our aim is to create an annotated dataset

$$\mathcal{D}_L \subseteq \{(x, y, z) \mid x \in \mathcal{D},\ y \in \mathcal{L},\ z \in \mathcal{F}_y\}$$

maximizing the accuracy of the detection model $\mathcal{M}$ trained on $\mathcal{D}_L$. $\mathcal{D}_L$ associates a label $y \in \mathcal{L}$ and a family $z \in \mathcal{F}_y$ to each instance $x^i \in \mathcal{D}$. The annotated dataset $\mathcal{D}_L$ is built with an iterative active learning strategy. At each iteration, a security administrator annotates, with a label and a family, $b \in \mathbb{N}$ instances selected from the pool of remaining unlabeled instances denoted by $\mathcal{D}_U$. Throughout the annotation process, the expert cannot annotate more instances than the annotation budget $B \in \mathbb{N}$.

4.4.2 Objective

Our goal is to conceive an end-to-end pool-based active learning system tailored to security administrators needs. It consists of an active learning strategy integrated in an annotation system. The two components must fulfill the following constraints to effectively reduce human effort in annotation projects.

Active Learning Strategy. The objective of the active learning strategy is to build the annotated dataset $\mathcal{D}_L$ that maximizes the accuracy of the detection model $\mathcal{M}$ while asking the expert to annotate at most B instances. In other words, the strategy aims to ask the expert to annotate the B instances that maximize the performance of the detection model $\mathcal{M}$. Besides, the strategy must be scalable to work on large datasets while maintaining short waiting-periods.

The real challenge faced by the active learning strategy is to avoid the sampling bias issue (see Section 4.3.2) to ensure a well-performing detection model, while keeping short waiting-periods to guarantee a good expert-model interaction.

Annotation System. The annotation system must provide an ergonomic user interface to streamline the annotation process. It must be suitable for non-machine learning experts since it is intended for security administrators.

First of all, the annotation system must provide an annotation interface to display and gather the answers to the annotation queries. It must be workable on any annotation project for detection systems. As a result, the annotation interface should be able to display different data types such as PDF files, Windows Office documents, pcap, NetFlow, Windows audit logs, or memory dumps.

Moreover, the annotation system should not be reduced to an annotation interface. It should provide feedback frequently to show experts the benefit of their annotations, and that they are on track to achieve their goal. Besides, it should help security administrators provide consistent annotations, even if they delineate the detection target and the alert taxonomy throughout the annotation process.

4.5 Overview of our Contributions

The two following chapters introduce ILAB, an end-to-end active learning system designed to help security administrators build annotated datasets with a reduced effort. We describe the active learning strategy (see Chapter 5) and its integration in an annotation system (see Chapter 6). The active learning strategy reduces the number of manual annotations and the waiting-periods. As for

the user interface, it can be used on any data type and it provides feedback to show the benefit of the annotations.

We make the following contributions:

- We present an active learning strategy designed to avoid sampling bias and to reduce experts waiting-periods. Security administrators can annotate some instances while the algorithm is still computing new annotation queries.
- We compare ILAB active learning strategy with two state-of-the-art methods for intrusion detection [130, 63] on two detection problems. We demonstrate that ILAB improves the scalability without reducing the effectiveness. Up to our knowledge, [130, 63] have never been compared.
- We integrate this strategy in an annotation system tailored to security administrators needs. We have designed the interface for annotators who may have little knowledge about machine learning, and the generic interface can manipulate any data type. Moreover, it provides feedback to encourage security administrators to go on annotating. Finally, it helps them provide consistent annotations even if they delineate the detection target and the alert taxonomy as they annotate.
- We ask intended end-users, security administrators, to annotate a large unlabeled NetFlow dataset coming from a production environment with ILAB. These user experiments validate our design choices and highlight potential improvements.
- We provide an open-source implementation of the whole active learning system in SecuML [20] to foster comparison in future research works, and to enable security administrators to annotate their own datasets.

Chapter 5

ILAB Active Learning Strategy

Active learning has been introduced in the machine learning community to reduce the cost of building annotated datasets. A query strategy asks the expert to annotate only the most informative instances to reduce the number of manual annotations. The main challenge for an effective active learning strategy is to avoid sampling bias without inducing long waiting-periods.

In this chapter, we introduce a novel active learning strategy, ILAB, that helps security administrators annotate large datasets with a reduced workload. First, we present the active learning strategy and we explain how it avoids the sampling bias issue while keeping short waiting-periods. Then, we compare ILAB with two state-of-the-art approaches [130, 63] on public annotated datasets and demonstrate that it is both an effective and a scalable solution. We provide open-source implementations of ILAB and of the above-mentioned state-of-the-art strategies to ease comparison in future research works [20].

This chapter content has been mostly published at the *20th International Symposium on Research in Attacks, Intrusions and Defenses* (RAID 2017) [18]. It includes some additional content related to previous submissions.

Contents

5.1 Introduction

Active learning strategies have been proposed to reduce the annotation cost by asking experts to annotate only the most informative instances [119]. However, classical active learning methods often suffer from sampling bias [116, 121]: query strategies that pick only the most informative instances can completely overlook a family (a group of similar malicious or benign instances). Sampling bias is a significant issue for detection systems: it may lead to missing a malicious family during the labeling process, and being unable to detect it thereafter. The reader may refer to Section 4.3.2 for more information about sampling bias.

Moreover, active learning is an interactive process which must ensure a good expert-model interaction, i.e. a good interaction between the security administrator who annotates and the detection model [120, 138]. The annotations improve not only the detection model but also the relevance of the following annotation queries. A low execution time is thus required to allow frequent updates of the detection model with the expert feedback. Query strategies with a high execution time would alter the expert-model interaction and are unlikely to be accepted by security administrators. Besides, active learning strategies must scale to large datasets to be workable on real-world annotation projects.

To sum up, active learning strategies must avoid sampling bias to guarantee a good detection performance while keeping short waiting periods to ensure a good expert-model interaction.

In this chapter, we introduce a new active learning strategy, ILAB, that reduces the annotation cost to build supervised detection models. It relies on a new hierarchical active learning strategy with binary labels (malicious vs. benign) and user-defined malicious and benign families. It avoids the sampling bias issue encountered by classical active learning as it is designed to discover the different malicious and benign families. Moreover, the scalable algorithms used in ILAB make it workable on large datasets and guarantee short waiting-periods for a good expert-model interaction.

This chapter presents the following contributions:

- We present ILAB, which stands for Interactive LABeling, a novel active learning strategy designed to avoid sampling bias while keeping short waiting-periods. It has a low computation cost to ensure a good expert-model interaction, and it is scalable to large datasets. ILAB relies on a divide-and-conquer approach to reduce waiting-periods: experts can annotate some instances while the algorithm is still computing new annotation queries.

- We compare ILAB with two state-of-the-art active learning methods designed for intrusion detection, Aladin [130] and Görnitz et al. active learning strategy [63], on two detection problems. We demonstrate that ILAB improves the scalability without reducing the effectiveness. Up to our knowledge, [130] and [63] have never been compared.

- We provide an open-source implementation of ILAB, Aladin and Görnitz et al. active learning strategy in SecuML [20] to foster comparison in future research works.

The rest of the chapter is organized as follows. Section 5.2 presents ILAB active learning strategy and Section 5.3 explains our design choices. Finally, Section 5.4 presents comparisons with two state-of-the-art approaches [130, 63] on two public fully annotated datasets.

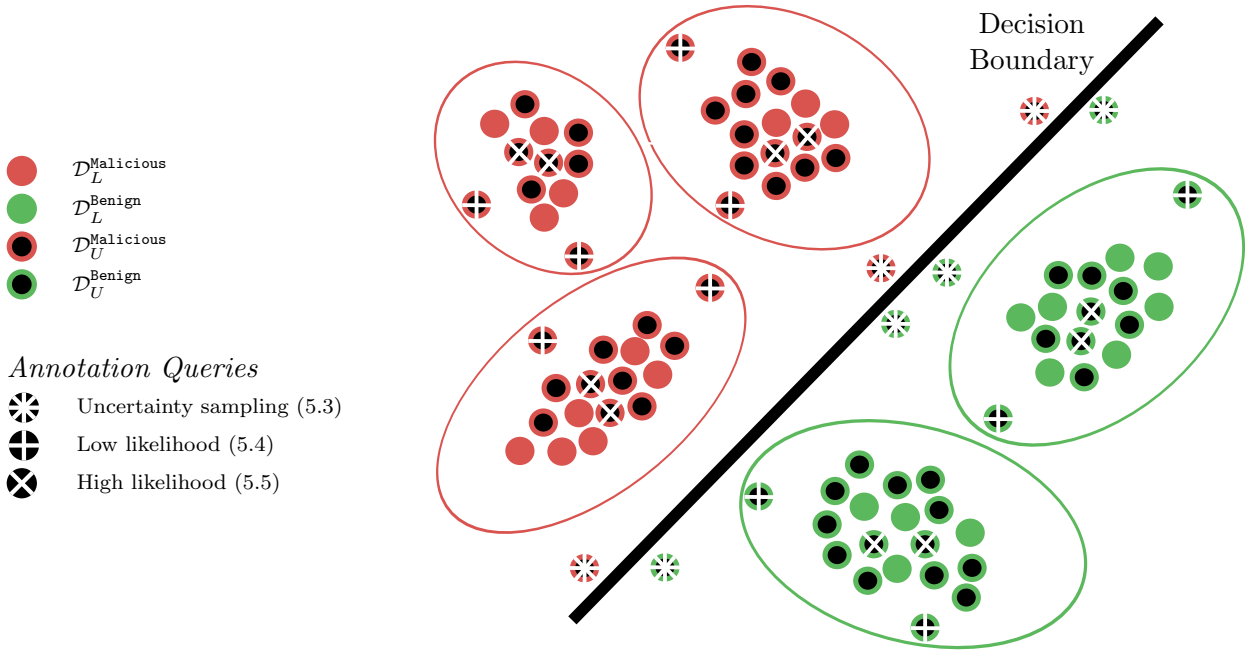


Figure 5.1: ILAB Active Learning Strategy.

5.2 ILAB Active Learning Strategy

ILAB is an iterative annotation process based on active learning [119] and rare category detection [102]. At each iteration, the expert is asked to annotate b instances to improve the current detection model and to discover yet unknown families. Active learning improves the binary classification model generating the alerts while rare category detection fosters the discovery of new families to avoid sampling bias.

The iterations are performed until the annotation budget B has been spent. At each iteration, $b_{\texttt{uncertain}}$ annotation queries are generated with *uncertainty sampling* to improve the detection model and $b_{\texttt{families}} = b - b_{\texttt{uncertain}}$ instances are queried for annotation with *rare category detection* to avoid sampling bias (see Figure 5.1). In this section, we explain the active learning strategy, i.e. which instances are selected from the unlabeled pool to be annotated by the security administrator.

5.2.1 Uncertainty Sampling

We train a binary probabilistic detection model $\mathcal{M}$ from the annotated instances in $\mathcal{D}_L$. We use a discriminant linear model, i.e. logistic regression (see Section 2.3.1).

Security administrators, who do not trust black-box detection models [111], highly value linear models. They can interpret these models because the coefficients associated with each feature represent their contribution to the detection model. The reader may refer to Section 2.4.3 for more information about the interpretation of logistic regression models.

Besides, discriminant models are known to be better than generative ones in active learning settings [151]. Finally, learning a logistic regression model and applying it to predict the label of new instances is fast so the expert does not wait a long time between iterations. Our approach

is generic: security administrators can choose another model class particularly suited for their application.

The rare malicious families are often the most interesting for detection systems, hence the impact of the training instances from rare families is increased. The logistic regression model is trained with sample weights inverse to the proportion of the family in the training dataset:

$$\alpha(x, y, z) = \frac{|\mathcal{D}_L|}{|\{(x', y', z') \in \mathcal{D}_L \mid y' = y \wedge z' = z\}|}. \tag{5.1}$$

The weights are capped, $\hat{\alpha} = \min(\alpha, 100)$, to avoid giving too much weight to very rare families. Training the logistic regression detection model with these weights is crucial to ensure a good detection of the rare malicious families.

During the training phase, the parameters $\beta_0 \in \mathbb{R}$ and $\beta \in \mathbb{R}$ of the logistic regression model $\mathcal{M}$ are fit from the annotated dataset $\mathcal{D}_L$. Then, the model $\mathcal{M}$ can compute the probability $p(x)$ that an unlabeled instance $x \in \mathcal{D}_U$ is `Malicious`:

$$\forall x \in \mathcal{D}_U,\ p(x) = P_{\mathcal{M}}(y = \texttt{Malicious} \mid x) = \frac{1}{1 + \exp\left(-\left(\beta_0 + \beta^T x\right)\right)}. \tag{5.2}$$

Annotation Queries. The security administrator annotates the $b_{\texttt{uncertain}}$ unlabeled instances which are the closest to the decision boundary of $\mathcal{M}$:

$$\arg\min_{x \in \mathcal{D}_U} |p(x) - 1/2|. \tag{5.3}$$

The detection model is uncertain about the label of these instances, that is why their annotations improve the detection model.

This step corresponds to uncertainty sampling [82], a classical active learning method applied to intrusion detection in [4]. Uncertainty sampling suffers from sampling bias [116], so we also perform rare category detection to foster the discovery of yet unknown families.

5.2.2 Rare Category Detection

We apply rare category detection on the instances that are more likely to be `Malicious` and `Benign` (according to the detection model $\mathcal{M}$) separately. Not all families are present in the initial annotated dataset and rare category detection [102] fosters the discovery of yet unknown families to avoid sampling bias.

One might think that we could run rare category detection only on the malicious instances since it is the class of interest for detection systems. However, a whole malicious family may be on the wrong side of the decision boundary (see the family M_1 in Figure 4.2), and thus, running rare category detection on the predicted benign instances is necessary. Hereafter, we only detail the rare category detection run on the `Malicious` predictions since the analysis of the `Benign` ones is performed similarly.

Let $\mathcal{D}_U^{\texttt{Malicious}}$ be the set of instances whose predicted label by $\mathcal{M}$ is `Malicious` (for a detection threshold $t = 50\%$), and $\mathcal{D}_L^{\texttt{Malicious}}$ be the set of malicious instances already annotated by the expert.

First, we train a multi-class logistic regression model from the families specified in $\mathcal{D}_L^{\texttt{Malicious}}$ to predict the family of the instances in $\mathcal{D}_U^{\texttt{Malicious}}$. We denote by $\mathcal{C}_f$ the set of instances from $\mathcal{D}_L^{\texttt{Malicious}} \cup \mathcal{D}_U^{\texttt{Malicious}}$ whose family (annotated or predicted) is f.

Then, we model each family f with a Gaussian distribution $\mathcal{N}(\mu_f, \Sigma_f)$ depicted by an ellipsoid is Figure 5.1. The mean μ_f and the diagonal covariance matrix Σ_f are learned with Gaussian Naive Bayes (see Section 2.3.1). We denote by $p_{\mathcal{N}(\mu_f,\Sigma_f)}(x)$ the probability that x follows the Gaussian distribution $\mathcal{N}(\mu_f, \Sigma_f)$.

Annotation Queries. The family annotation budget $b_{\texttt{families}}$ is evenly distributed among the different families. We now explain which unlabeled instances are queried for annotation from each family.

First, ILAB asks the security administrator to annotate instances that are likely to belong to a yet unknown family to avoid sampling bias. These instances are located at the edge of the ellipsoid, they have a low likelihood of belonging to the family f [102, 130]:

$$\underset{x \in \mathcal{C}_f \setminus \mathcal{D}_L^{\texttt{Malicious}}}{\arg\min} \; p_{\mathcal{N}(\mu_f,\Sigma_f)}(x). \tag{5.4}$$

Then, ILAB queries representative examples of each family for annotation. These instances are close to the center of the ellipsoid, they have a high likelihood of belonging to the family f:

$$\underset{x \in \mathcal{C}_f \setminus \mathcal{D}_L^{\texttt{Malicious}}}{\arg\max} \; p_{\mathcal{N}(\mu_f,\Sigma_f)}(x). \tag{5.5}$$

The budget is evenly allocated to low and high likelihood instances. Low likelihood instances are likely to belong to yet unknown families that is why these annotation queries foster the discovery of new families. They are, nonetheless, more likely to be outliers that may impair the detection model performance. ILAB also asks the security administrator to annotate high likelihood instances to get more representative examples of the families in the annotated dataset for a better generalization of the detection model.

5.3 Design Choices

5.3.1 Avoiding Sampling Bias

ILAB active learning strategy is a hybrid method (see Section 4.3.2): a) uncertainty sampling [82], a search through the hypothesis space method, improves the supervised detection model, and b) rare category detection exploits the structure in data to avoid sampling bias.

At the beginning, we have not considered rare category detection, but semi-supervised clustering [32, 150] to avoid sampling bias. This method leads to poorer results, but is still interesting to review.

We run the semi-supervised clustering analysis on the instances that are more likely to be `Malicious`, $\mathcal{D}_U^{\texttt{Malicious}}$, and `Benign`, $\mathcal{D}_U^{\texttt{Benign}}$, separately. Hereafter, we only detail the rare category detection run on the `Malicious` predictions since the analysis of the `Benign` ones is performed similarly.

We leverage semi-supervised clustering to get interpretable clusters corresponding to user-defined families. Indeed, a completely unsupervised clustering is unlikely to regroup instances according to the expert definition of families. The clusters are built with soft constraints steaming from the families assigned to previously annotated instances in $\mathcal{D}_L^{\texttt{Malicious}}$. As a result, instances belonging to the same family are more likely to share the same cluster than instances belonging to different families.

The semi-supervised clustering proceeds in three steps:

1. We learn a projection space where the `Malicious` instances with different families are further apart than the ones sharing the same family from $\mathcal{D}_L^{\texttt{Malicious}}$ with a metric learning algorithm [21].

   We have considered several algorithms during our experimentations: Linear Discriminant Analysis(LDA) [52], Large Margin Nearest Neighbors (LMNN) [146], Neighborhood Components Analysis (NCA) [57], Information-Theoretic Metric Learning (ITML) [41], Sparse Determinant Metric Learning (SDML) [103], and Relative Components Analysis (RCA) [16]. We have not considered Principal Component Analysis (PCA) during our experiments since it is unsuitable for semi-supervised clustering. It is an unsupervised projection method which finds the axes with maximum variance, and thus it may lose the discriminative information.

2. We project the instances of $\mathcal{D}_U^{\texttt{Malicious}}$ into this subspace.

3. We cluster the projected instances with Gaussian mixture models restricted to a diagonal covariance matrix. It is a scalable solution that reduces the expert waiting-periods and makes the query strategy workable on large datasets. It is a sound compromise between computation time and efficiency: Gaussian mixture models with a diagonal covariance matrix build more sophisticated clusters than k-means while keeping a low time complexity. The objective is to align each cluster with a family, so we set the number of clusters to the number of malicious families currently discovered, $k = |\mathcal{L}_{\texttt{Malicious}}|$.

The active learning strategy queries instances close to the center and the edge of the ellipsoids as in the strategy presented in Section 5.2.2.

Among the metric learning algorithms considered, only LDA, NCA, and RCA project the data into a lower dimensional feature space. This trait is appealing to shorten experts waiting-periods since reducing the dimension of the data decreases the cost of the clustering algorithm. Our experiments show that RCA meets our scalability objective, and provides the best results, but they are poorer than the ones obtained with rare category detection. Therefore, we have discarded semi-supervised clustering, and ILAB relies on the Pelleg and Moore's method [102].

5.3.2 Semi-Automatic Annotations

During our research, we have considered semi-automatic annotations in order to reduce expert effort during annotation projects. The idea is to identify homogeneous clusters aligned with families to assign them a semi-automatic annotation, and no longer consider them in the annotation process.

A cluster is said *aligned* with a family, if all its instances belong to this family, they share the same label and family. The instances in an aligned cluster are similar enough for the expert to interpret them as a whole. During the annotation process, we can consider that a cluster is aligned with a family if all its queried instances (near the center and close to the edge) share the same family. In this case, all the remaining unlabeled instances in the cluster are annotated semi-automatically with the shared label and family.

Semi-automatic annotations may be inaccurate if a cluster has been wrongly considered aligned with a family. Therefore, we do not add the semi-automatic annotations to the annotated dataset $\mathcal{D}_L$, they are not used to train the model. They are simply removed from the unlabeled pool $\mathcal{D}_U$: they cannot be queried for annotation in the following iterations.

We have looked at the concept of semi-automatic annotations for the following reasons. First, removing the most common behaviors from the unlabeled pool is expected to help the active learning strategy query instances behaving unusually.

Besides, semi-automatic annotations is a way to improve expert-model interaction by shortening expert-waiting periods. Indeed, semi-automatic annotations reduce the size of the unlabeled pool more significantly than manual annotations. When the size of the unlabeled pool decreases, the execution time of the analyses (rare category detection or semi-supervised clustering) decreases. Therefore, semi-automatic annotations reduce the expert waiting-periods across the iterations.

However, once the most common behaviors have been removed from the unlabeled pool, there remain only peculiar behaviors which raises two major problems. First, the clusters built from peculiar behaviors are hard to interpret. Second, annotating peculiar behaviors may mislead the detection model as the instances are not representative enough of the benign and malicious instances commonly encountered.

Besides, semi-automatic annotations can lead to missing rare families. Indeed, a cluster can be wrongly considered aligned with a family: the instances annotated near the center and close to the edge of the cluster share the same label and family, but the cluster contains instances belonging to a different family. In this situation, the expert cannot rectify the wrong semi-automatic annotations: the instances semi-automatically annotated with a wrong family will never be queried thereafter in the annotation process. If all the instances of a rare family are wrongly semi-automatically annotated, the active learning completely overlooks this family, and it may lead to false positives or negatives.

After having weighed up all the pros and cons, we have decided not to propose semi-automatic annotations in ILAB.

5.3.3 Reducing Waiting-Periods

We have designed ILAB keeping in mind that the expert waiting-periods must be minimized to ensure a good expert-model interaction. First of all, ILAB relies on efficient algorithms to reduce the execution time: uncertainty sampling [82] and Pelleg and Moore's rare category detection method [102].

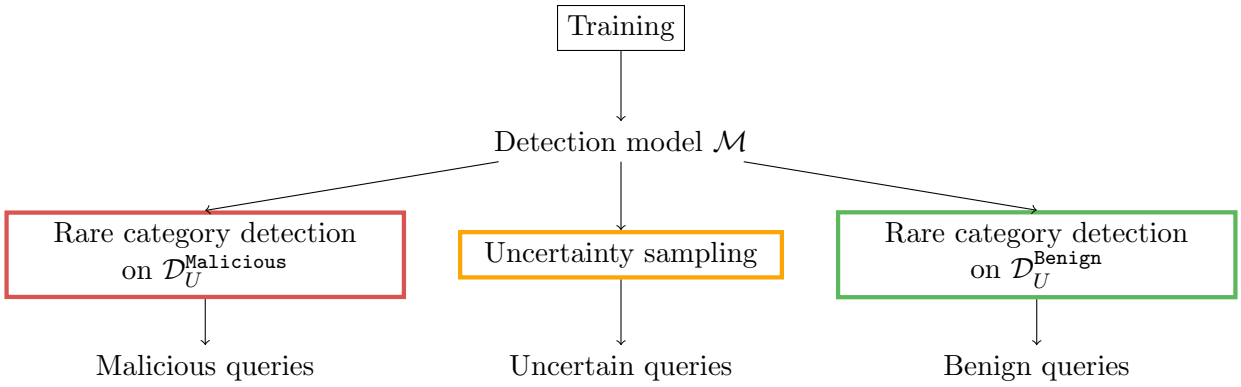


Figure 5.2: Parallelization of the Computations of the Annotation Queries.

Besides, ILAB computations can be parallelized with annotations. ILAB active learning strategy runs rare category detection on the benign and malicious instances independently. This divide-and-conquer approach allows the expert to annotate some instances while the strategy is still computing annotation queries.

The active learning strategy starts with training a binary detection model. To reduce the cost of this step, we pick a linear model, logistic regression, which can be trained faster than more complex model classes. If this model class is not flexible enough for a given annotation project, other model classes can be plugged into ILAB but may result in longer waiting-periods. The reader may refer to Chapter 2 for more information about model class selection.

Once the binary detection model has been trained, the active learning strategy generates the queries (see Figure 5.2). The generation of the different kinds of queries (uncertain, malicious and benign) are completely independent. This presents two advantages: 1) the computations can be parallelized, and, 2) the security administrator can start annotating while ILAB generates the remaining queries.

To sum up, ILAB design reduces the expert waiting-periods in two ways: 1) selection of efficient algorithms, and 2) parallelization of computations and annotations.

5.4 Evaluation

5.4.1 Datasets

Active learning strategies are generic methods that can be applied to any detection problem once the features have been extracted. In these experiments, we consider a system and a network detection problem: 1) detection of malicious PDF files with the Contagio dataset [1], and 2) network intrusion detection with the NSL-KDD dataset [2]. These datasets cannot be exploited to train models intended for production as they are non-representative of real-world data. Our comparisons are still relevant since we are not comparing attack detection models but active learning strategies in order to train attack detection models on new problems.

Contagio is a public dataset composed of 11,101 malicious and 9,000 benign PDF files. We transform each PDF file into 113 numerical features similar to the ones proposed by Smutz and

Dataset	#instances	#features	#malicious families	#benign families
Contagio_10%	10,000	113	16	30
NSL-KDD_10%	74,826	122	19	15

Table 5.1: Description of the Public Datasets.

Stavrou [124, 125] (see Section 3.3.2).

NSL-KDD contains 58,630 malicious and 67,343 benign instances. Each instance represents a connection on a network and is described by 7 categorical features and 34 numerical features. The 7 categorical features (e.g. `protocol_type` with the possible values `tcp`, `udp` or `icmp`) are encoded into several binary features corresponding to each value (e.g. `tcp` $\rightarrow [1,0,0]$, `udp` $\rightarrow [0,1,0]$, `icmp` $\rightarrow [0,0,1]$). We end up with 122 features.

The malicious instances in NSL-KDD are annotated with a family but the benign ones are not, and Contagio does not provide any family information. The families are, nevertheless, required to run simulations with Aladin [130] and ILAB, and to assess the sampling bias of the different active learning strategies. We have assigned families to the remaining instances with a k-means clustering and we have selected the number of families k visually with the silhouette coefficient [112].

Neither dataset has a proportion of malicious instances representative of a typical network (55% for Contagio and 47% for NSL-KDD). We have uniformly sub-sampled the malicious class to get 10% of malicious instances. Table 5.1 describes the resulting datasets: Contagio_10% and NSL-KDD_10%.

5.4.2 Active Learning Strategies

We compare ILAB to three other active learning strategies: uncertainty sampling [82], Görnitz et al. labeling strategy [63], and Aladin [130]. Since there is no open-source implementation of these techniques, we have implemented them in Python with the machine learning library scikit-learn [101]. We have released all the implementations in SecuML [20] to ease comparison in future research works.

These active learning strategies share two parameters: 1) the global annotation budget $B \in \mathbb{N}$, and 2) the number of annotations answered at each iteration $b \in \mathbb{N}$. In this section, we briefly present each method, we provide some details about our implementations and how we set the additional parameters if relevant.

Uncertainty Sampling [82]. At each iteration, a binary logistic regression model is trained on the annotated instances, and the expert is asked to annotate the b most uncertain predictions, i.e. the closest to the decision boundary. Uncertainty sampling has no additional parameter.

Görnitz et al. labeling strategy [63]. At each iteration, a semi-supervised anomaly detection model is trained on both the annotated and the unlabeled instances. The model relies on an adaptation of an unsupervised anomaly detection model, Support Vector Data Description (SVDD) [137], that takes into account annotated instances. It consists in a sphere defined by a center $c \in \mathbb{R}^m$ and

a radius $r \in \mathbb{R}$: the instances inside the sphere are considered benign, and the ones outside malicious. The labeling strategy queries instances that are both close to the decision boundary and have few malicious neighbors to foster the discovery of new malicious families. We compute the nearest neighbors with the Euclidean distance. We leverage the scikit-learn ball tree implementation [96] that is effective with a large number of instances in high dimension.

Semi-supervised SVDD has no open-source implementation, so we have implemented it for our experiments with the information provided in [62, 61, 63]. The center $c \in \mathbb{R}^m$, the radius $r \in \mathbb{R}$, and the margin $\gamma \in \mathbb{R}$ are determined with the quasi-Newton optimization method BFGS [149] available in scipy [69]. The optimization algorithm requires initial values for c, r, and γ that are not specified by the authors. We initialize c with the mean of the unlabeled and benign instances, r with the average distance of the unlabeled and benign instances to the center c, and γ with the default value 1. Moreover, the detection model has three hyperparameters that must be set before training: $\eta_U \in \mathbb{R}$ and $\eta_L \in \mathbb{R}$, the weights of the unlabeled and annotated instances, and κ the weight of the margin γ. The authors provide no information about how to set these hyperparameters. When we set them to the default value 1, numerical instabilities prevent the optimization algorithm from converging properly, and lead to an extremely high execution time and very poor performance (more than 2 hours for training the model on Contagio_10% to get an AUC below 93%). We have thus worked on the setting of these hyperparameters. We have set η_U and η_L to the inverse of the number of unlabeled and labeled instances, to give as much weight to unlabeled and labeled instances, and to ensure numerical stability. The detection model is trained without any kernel as in the experiments presented in [62, 61, 63].

Finally, the active learning strategy requires to set two additional parameters: $k \in \mathbb{N}$ the number of neighbors considered, and $\delta \in [0, 1]$ the trade-off between querying instances close to the decision boundary and instances with few malicious neighbors. We use $k = 10$ as in [63] and the default value $\delta = 0.5$.

Aladin [130]. Aladin runs rare category detection on all the data. It asks the expert to annotate uncertain instances lying between two families to refine the decision boundaries, and low likelihood instances to discover yet unknown families. Aladin does not have additional parameters.

This labeling strategy relies on a multi-class logistic regression model and a multi-class Gaussian Naive Bayes model. The logistic regression hyperparameters, the penalty norm and the regularization strength, are selected automatically with a grid-search 4-fold cross validation [52] optimizing the AUC [67]. The penalty norm is either ℓ_1 or ℓ_2 and the regularization strength is selected among the values $\{0.01, 0.1, 1, 10, 100\}$. The Gaussian Naive Bayes model is trained without any prior.

ILAB. ILAB active learning strategy has only an additional parameter: $b_{\texttt{uncertain}}$. We set its value to 10% of the number of annotations performed at each iteration. Some instances near the decision boundary are annotated to help the detection model make a decision about these instances, but not too many since these instances are often harder to annotate for the security administrator [120, 66, 15] and they may lead to a sampling bias [116].

The logistic regression and Gaussian Naive Bayes models are trained in the same way as for Aladin.

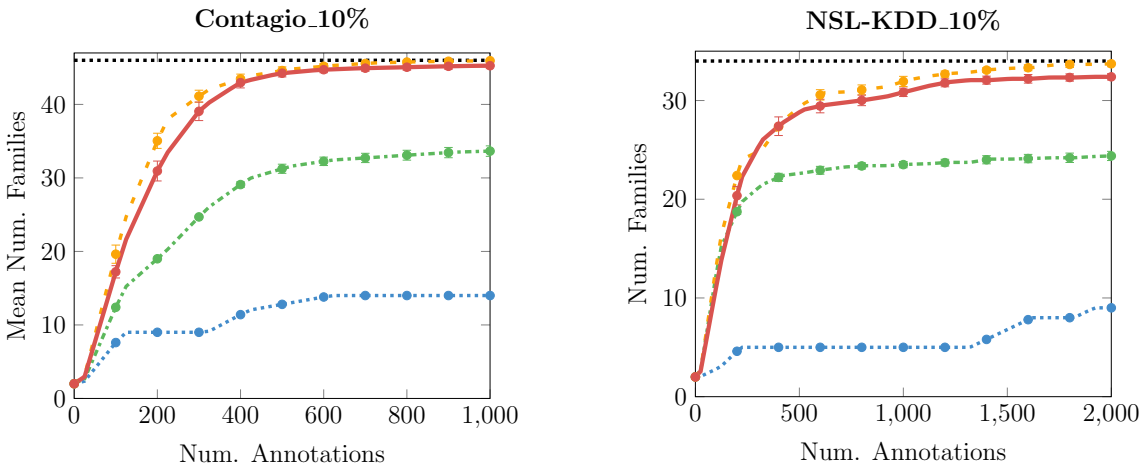


Figure 5.3: Average Number of Families Discovered.

In the next section, we compare all these methods to ILAB. We run all Linux 3.16 on a dual-socket computer with 64Go RAM. Processors are Intel Xeon E5-5620 CPUs clocked at 2.40 GHz with 4 cores each and 2 threads per core. We run each method 15 times and we report the average performance with the 95% confidence interval.

5.4.3 Comparison

The datasets Contagio_10% and NSL-KDD_10% are split uniformly into two datasets: 1) an active learning dataset (90%) used as an unlabeled pool queried to build the annotated dataset $\mathcal{D}_L$, and 2) a validation dataset (10%) to assess the performance of the detection model trained on $\mathcal{D}_L$.

We compare the different labeling strategies automatically without involving security administrators. An oracle answers the annotation queries with the ground-truth labels and families.

We run all the strategies with $b = 100$ annotations at each iteration. We set the annotation budget to $B = 1000$ for Contagio_10%, and to $B = 2000$ for NSL-KDD_10% as this dataset contains more instances. The initial annotated datasets are composed of instances belonging to the most represented families: 7 malicious instances and 13 benign instances.

First, we compare the number of known families across the iterations to assess sampling bias (see Figure 5.3). Then, we compare the performance of the detection models on the validation dataset (see Figure 5.4). Finally, we monitor the execution time of the query generation algorithms to evaluate the expert waiting time between iterations (see Figure 5.5).

Families Detection. Figure 5.3 shows that uncertainty sampling and Görnitz et al. labeling strategy miss many families during the annotation process. Both labeling strategies suffer from sampling bias. Görnitz et al. labeling strategy relies on k-nearest neighbors to detect yet unknown malicious families but only close to the decision boundary, that is why many families further from the decision boundary are not discovered. Their strategy to foster the discovery of yet unknown families is ineffective on both datasets.

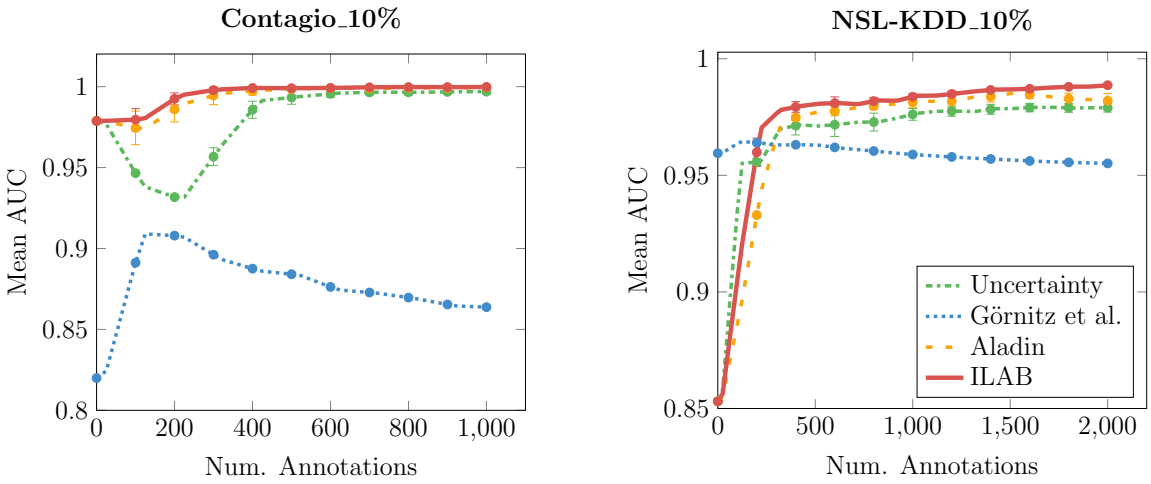


Figure 5.4: Average Detection performance (AUC) on the Validation Dataset.

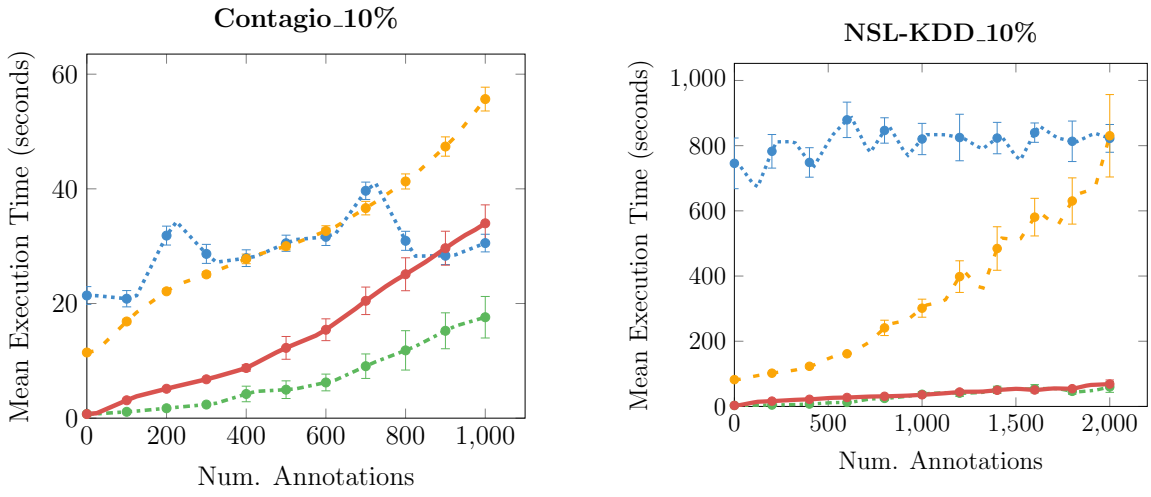


Figure 5.5: Average Queries Generation Execution Time.

ILAB dedicates only a part of its annotation budget to the detection of yet unknown families, that is why Aladin detects slightly more families than ILAB. ILAB queries some high likelihood instances which are unlikely to belong to new families, but they allow to keep the detection performance increasing across the iterations (see Figure 5.4).

ILAB and Aladin discover about as many families across the iterations on both datasets. These labeling strategies are effective at avoiding sampling bias. They are designed to detect rare categories, and they are able to discover almost all the families on both datasets.

Detection Performance. Figure 5.4 represents the evolution of the Area Under the Curve (AUC) [67] on the validation datasets. It shows that ILAB performs better than the other labeling strategies on both datasets.

Görnitz et al. labeling strategy performs very poorly on Contagio_10%. The detection performance increases at the first iteration, but then it keeps on decreasing when new instances are added to the annotated dataset. This peculiar behavior can be explained by the simplicity of the SVDD detection model which cannot discriminate the benign from the malicious instances properly. The geometry of the data prevents SVDD from isolating the benign instances from the malicious instances in a sphere. We notice the same behavior but less pronounced on NSL-KDD_10%. A solution to address this issue is to train SVDD with a kernel to increase its flexibility, but it will considerably increase the execution time which is already too high to ensure a good expert-model interaction (see Figure 5.5).

Görnitz et al. labeling strategy performs much better initially on NSL-KDD_10% than the other labeling strategies. Indeed, thanks to semi-supervision, Görnitz et al. leverage not only the 20 initial annotated instances to train their detection model, but also all the instances from the unlabeled pool. Görnitz et al. semi-supervised detection model is, however, not as effective as logistic regression initially on Contagio_10%. SVDD makes the assumption that the unlabeled instances are mostly benign, so the malicious instances in the unlabeled pool may damage the detection model performance.

Uncertainty sampling has a better detection performance than ILAB during the first iterations on NSL-KDD_10% because it allocates all its annotation budget to refining the decision boundary. On the contrary, ILAB dedicates 90% of its annotation budget to rare category detection to avoid sampling bias. In the end, uncertainty sampling suffers from sampling bias and converges to a poorer performance.

The detection performance of uncertainty sampling and Aladin decreases during the first iterations on Contagio_10%. Sampling bias causes this undesirable behavior: non-representative instances are queried for annotation, added to the training dataset and prevent the detection model from generalizing properly. Uncertainty sampling queries instances close to the decision boundary that are hard to classify for the detection model, but not representative of the malicious or benign behaviors. Aladin queries only uncertain and low likelihood instances which are not necessarily representative of the malicious and benign behaviors either. ILAB addresses this problem by dedicating a part of its annotation budget to high likelihood instances to get representative examples of each family. Therefore, the detection performance keeps on increasing across the iterations.

Scalability. Figure 5.5 depicts the query generation execution time (in seconds) across the iterations. Görnitz et al. query generation algorithm is very slow. For NSL-KDD_10%, the expert waits more than 10 minutes between each iteration while the labeling strategy computes the annotation

queries. A third of the execution time corresponds to the computation of the semi-supervised SVDD model, and the remaining two thirds corresponds to the k-nearest neighbor algorithm. The execution time of Görnitz et al. labeling strategy is thus too high to ensure a good expert-model interaction even on a dataset containing fewer than 100,000 instances.

ILAB has an execution time comparable to uncertainty sampling. For NSL-KDD_10%, the expert waits less than 1 minute between each iteration. On the contrary, Aladin execution time increases drastically when new instances are added to the annotated dataset and new families are discovered. Aladin runs rare category detection on all the instances, while ILAB runs it on the malicious and the benign instances separately. ILAB divide-and-conquer approach reduces the execution time as running rare category detection twice on smaller datasets with fewer families is faster than running it on the whole dataset. Aladin's authors are aware of this high execution time. During their experiments, the expert is asked to annotate a thousand instances each day, and the new annotation queries are computed every night. Their solution shortens the expert waiting-periods, but it significantly damages the expert-model interaction since the expert feedback is integrated only once a day.

In conclusion, uncertainty sampling and Görnitz et al. labeling strategy suffer from sampling bias. Aladin and ILAB are the only labeling strategies able to avoid sampling bias thanks to rare category detection performed at the family level (see Figure 5.3). As a result, Aladin and ILAB detect more families than the other strategies, and they also end up with better performing detection models (see Figure 5.4). ILAB main advantage over Aladin is its divide-and-conquer approach that significantly reduces the execution time (see Figure 5.5) and thus improves the expert-model interaction. Our comparisons show that ILAB is both an effective and a scalable active learning strategy.

5.5 Conclusion

This chapter introduces ILAB, a novel active learning strategy we have designed and implemented to streamline annotation projects. It minimizes the number of manual annotations, but also the expert waiting-periods.

It relies on uncertainty sampling and rare category detection to avoid sampling bias. Besides, its divide-and-conquer approach reduces the computation cost, and enables security administrators to annotate some instances while new annotation queries are computed. ILAB ensures a good expert-model interaction: the detection model can be updated frequently with expert feedback without inducing long waiting-periods.

We demonstrate that ILAB offers a better scalability than two state-of-the-art active learning strategies, Aladin [130] and Görnitz et al. labeling strategy [63], without damaging the effectiveness. Up to our knowledge, [130] and [63] had never been compared. We provide open-source implementations of ILAB, Aladin, and Görnitz et al. labeling strategy in SecuML [20] to foster comparison in future research works.

In the next chapter, we present the integration of ILAB active learning strategy into an annotation system and we evaluate the whole active learning system with user experiments performed on a real-world annotation project. Integrating ILAB active learning strategy into an annotation system is crucial to bridge the gap between theoretical active learning and real-world annotation projects [145, 85].

Chapter 6

ILAB Active Learning System

Active learning should not be reduced to a strategy querying instances for annotation. It is an interactive procedure where user experience must be taken into account to efficiently streamline annotation projects. In consequence, active learning strategies should not be evaluated only with simulations where oracles answer the annotation queries automatically. It is critical to carry out user experiments with intended end-users to ensure that active learning is effective at reducing the workload.

In this chapter, we integrate ILAB active learning strategy in an annotation system to get an end-to-end active learning system. It helps security administrators annotate large datasets with a reduced workload. Our user experiments show that ILAB is an effective active learning system that security administrators can deploy in real-world annotation projects.

This chapter content has been published at the *Artificial Intelligence for Computer Security workshop* (AICS 2018) [19].

Contents

6.1 Introduction

Most research works on active learning focus on query strategies [63, 4, 82, 139, 51, 40] to minimize the number of manual annotations. These works assume that annotators are mere oracles providing ground-truth labels while active learning is an interactive procedure where user experience should not be overlooked [120, 145]. A user interface is needed to gather the annotations and it must be suitable for security administrators who are usually not machine learning experts. Besides, some feedback must show the usefulness of the annotations, and annotators should not wait too long while the next annotation queries are computed.

We define an *active learning system* as an *annotation system* that leverages an *active learning strategy* to select the instances to be annotated. It is crucial to design both components jointly to effectively reduce annotation effort and to foster the adoption of active learning in annotation projects [85, 15, 138]. Security administrators do not want to minimize only the number of manual annotations, but the overall time spent annotating.

We have described ILAB active learning strategy and extensively compared it to state-of-the-art methods [4, 63, 130] in Chapter 5. It avoids sampling bias [116] without inducing long waiting-periods to ensure a good user experience. In this chapter, we integrate ILAB active learning strategy in an annotation system to bridge the gap between theoretical active learning and real-world annotation projects. We present the following contributions:

- We integrate ILAB active strategy in an annotation system tailored to security administrators needs. We have designed the graphical interface for annotators who may have little knowledge about machine learning, and it can manipulate any data type. Moreover, it helps security administrators provide consistent annotations even if they delineate the detection target and the alert taxonomy as they annotate.
- We ask intended end-users, security administrators, to use ILAB on a large unlabeled NetFlow dataset coming from a production environment. These experiments validate our design choices and highlight potential improvements.
- We provide an open-source implementation of the whole active learning system in SecuML [20] to foster comparison in future research works, and to enable security administrators to annotate their own datasets.

The rest of the chapter is organized as follows. Section 6.2 describes the annotation system we have designed and implemented to deploy ILAB active learning strategy in computer security annotation projects. Then, Section 6.4 presents the protocol of the user experiments carried out with security administrators on a real-world annotation project. Finally, Section 6.5 leverages the user experiments to validate ILAB design, and Section 6.6 presents additional feedback and points out some avenues of research to further improve user experience in annotation projects.

6.2 ILAB Annotation System

In this section, we describe ILAB annotation system, and we explain how it has been designed to ensure a good user experience. It obviously includes an *Annotation Interface* (see Section 6.2.1)

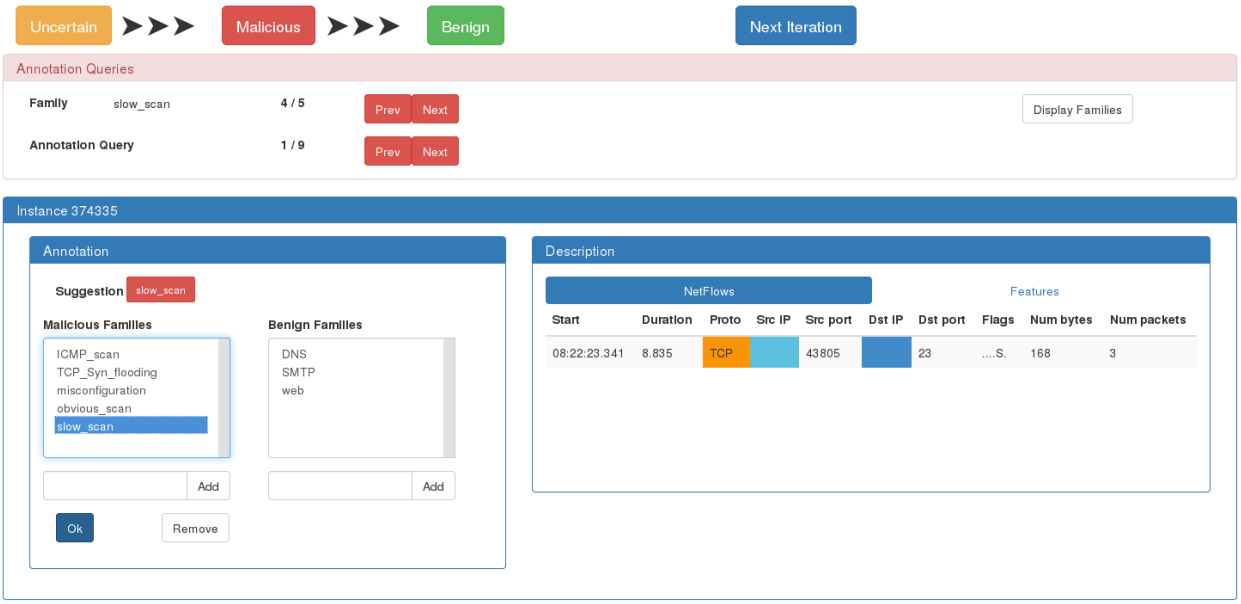


Figure 6.1: ILAB Annotation Interface.

to display and gather the answers to the annotation queries. Moreover, ILAB annotation system offers additional graphical user interfaces to ease data annotation: a *Monitoring Interface* (see Section 6.2.2), a *Family Editor* and an *Annotated Instances Interface* (see Section 6.2.3).

6.2.1 Annotation Interface

Security administrators answer ILAB queries from the graphical user interface depicted in Figure 6.1. It is intended for non-machine learning experts, it does not contain any word belonging to the machine learning jargon. The layout of the panels is designed to ensure a logical reading order.

Experts can select a type of queries with one of the top buttons: *Uncertain* for the instances near the decision boundary, *Malicious* and *Benign* for the queries generated by rare category detection. The reader may refer to Section 5.2 for more detail about ILAB active learning strategy.

The *Annotation Queries* panel displays the queries. Malicious and benign queries are grouped by families. The bottom panel displays the queried instances (*Description* panel), and gathers the annotations (*Annotation* panel).

Description panel. The *Description* panel contains information about the instance that the security administrator must annotate. It consists of a standard visualization depicting the instance features, and of optional problem-specific visualizations. Figure 6.1 shows the custom visualization we have implemented for NetFlow data[1].

We strongly encourage security administrators to design and implement convenient problem-specific visualizations, since they can considerably ease the annotations. The custom visualizations should display the most relevant information to help annotators make decisions, i.e. assigning

[1]We have hidden the IP addresses for privacy reasons.

a label and a family to a given instance. Security administrators can implement several custom visualizations to show the instances from different angles.

ILAB and DIADEM (see Chapter 3) rely on the same *Description* panel to display instances. The reader may refer to Section 3.2.2 for more information about this panel. As a result, once security administrators have implemented custom visualizations for a given detection problem, they can take advantage of them both with ILAB to annotate data and with DIADEM to diagnose detection models.

Annotation panel. Experts can annotate the selected instance with the *Annotation* panel. For each label, it displays the list of the families already discovered. Experts can pick a family among a list or add a new one.

The interface suggests a family for high likelihood queries and pre-selects it. It helps experts since the model is confident about these predictions. On the contrary, ILAB makes no suggestion for uncertain and low likelihood queries. The model is indeed unsure about the family of these instances and unreliable suggestions may mislead experts [15].

The next query is displayed automatically after each annotation validation. Experts can click on the `Next Iteration` button to generate the next queries after answering all the queries of the current iteration. If some queries have not been answered, a pop-up window asks the annotator to answer them.

6.2.2 Monitoring Interface

ILAB *Monitoring Interface* (see Figure 6.2) displays information about the current detection model (*Model Coefficients*, *Train* and *Cross Validation* panels), and feedback about the annotation progress (*Feedback* panel). Moreover, the *Monitoring Interface* gives access to the *Annotated Instances Interface* and to the *Family Editor* that are introduced in the next section.

Feedback about Annotation Progress. Annotation systems must provide feedback to experts to show them the benefit of their annotations, and that they are on track to achieve their goal [5]. In simulated experiments, where an oracle answers the queries automatically with the ground-truth labels, the performance of the detection model $\mathcal{M}$ on an independent validation dataset is usually reported. Nevertheless, this approach is not applicable in a real-world setting: when security administrators deploy an annotation system to build a training dataset they do not have access to an annotated validation dataset.

ILAB *Feedback* panel displays two kinds of feedback that do not require an annotated validation dataset: 1) the number of malicious and benign families discovered so far, and, 2) the accuracy of the suggested labels and families. At each iteration, ILAB suggests a family for the high likelihood queries. At the next iteration, ILAB computes the accuracy of these suggestions according to the last annotations performed by the expert.

This feedback can provide insight into the impact of new annotations. If the number of families discovered and the accuracy of the suggestions are stable for several iterations, the security administrator may stop annotating.

Current Detection Model. The *Monitoring Interface* displays information about the detection model trained at a given iteration: the coefficients of the logistic regression model (*Model Coeffi-*

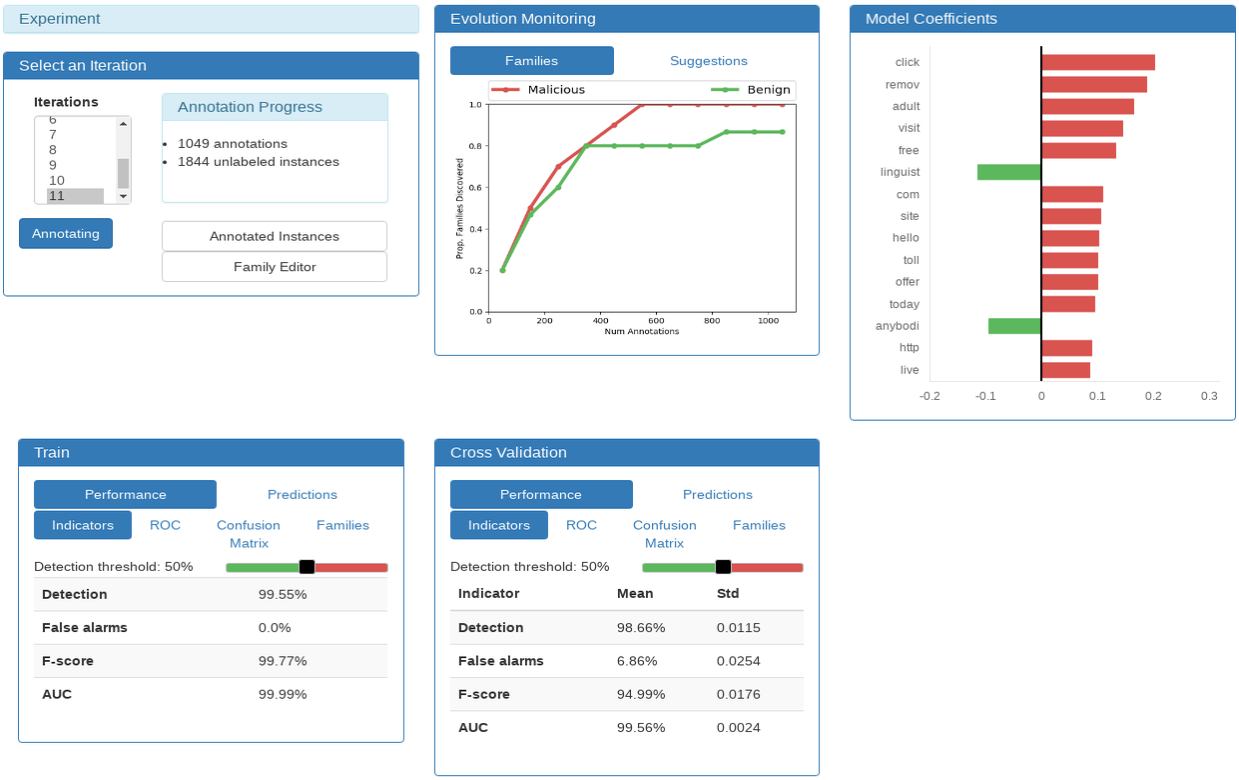


Figure 6.2: ILAB Monitoring Interface.

cients panel) and performance indicators (*Train* and *Cross Validation* panels). These monitoring panels are part of DIADEM. The reader may refer to Section 3.2 for a more detailed presentation.

ILAB *Monitoring Interface* can be used in two scenarios: 1) real-world annotation projects where human annotators answer the annotation queries, and 2) simulated experiments where oracles answer the annotation queries automatically with the ground-truth labels and families.

In the first case, security administrators leverage ILAB to build annotated datasets. The *Monitoring Interface* provides some feedback to annotators even if there is no annotated validation dataset available.

The second scenario is interesting for researchers working on new active learning strategies. They can run ILAB in automatic mode : an oracle answers the annotation queries automatically. In this case, ILAB can monitor the accuracy of the detection model across the iterations on an annotated validation dataset. We have run ILAB in this setting for the experiments presented in Chapter 5.

6.2.3 Annotated Instances and Family Editor

At the beginning of annotation projects, security administrators have usually vague specifications in mind of the detection target and of the alert taxonomy they want to set up. They refine these specifications as they examine new instances queried for annotation. Some annotation queries may puzzle them, make them adjust the detection target and/or the alert taxonomy, or even question previous annotations.

Even if the detection target and the alert taxonomy evolve in security administrators' mind, they must provide consistent annotations not to mislead the detection model.

ILAB offers two user interfaces to help security administrators refine the detection target and the alert taxonomy while remaining consistent with their previous annotations: a *Family Editor* and an *Annotated Instances Interface*.

Family Editor. The family editor enables annotators to perform three actions over the families:

1. *Change Name* to clarify the name of a family ;
2. *Merge Families* to regroup similar families ;
3. *Swap Malicious / Benign* to change the label corresponding to a given family.

The family editor is similar to the one introduced by Kulesza et al. [77] (see Section 4.3.1).

Annotated Instances Interface. This interface enables experts to review their previous annotations. It displays the currently annotated instances grouped according to their associated label or family.

Security administrators can leverage this interface to examine the instances of a given family, or to rectify previous annotations. Thanks to the *Family Editor*, they can perform high-level changes on the families, but they cannot split them. They can split a family thanks to the *Annotated Instances Interface* by going through all its instances and updating the annotations.

Security administrators work on diverse data types. A strength of ILAB is to be generic, so that they can use a unique annotation system. Once they get used to ILAB on a given detection problem, they will be more efficient at using it on other detection problems.

Moreover, ILAB and DIADEM are both part of SecuML [20], and they rely on the same *Description* panel to display instances. As a result, once security administrators have implemented custom visualizations for a given detection problem, they can take advantage of them both with ILAB to annotate data and with DIADEM to diagnose detection models.

6.3 Deployment of ILAB Active Learning System

In this section, we explain the steps that security administrators should take to deploy ILAB active learning system in real-world annotation projects.

First of all, ILAB does not take raw data as input, but instances represented as fixed-length vectors of features. Feature extraction must thus be performed before launching ILAB. Besides, security administrators are strongly encouraged to provide problem-specific visualizations to ease the annotations.

We detail below how security administrators should set the parameters of ILAB active learning strategy, and how they can collect an initial annotated dataset to launch the first active learning iteration.

6.3.1 Settings of the Parameters

Security administrators need to set three parameters to deploy ILAB in annotation projects. First, two parameters are shared by all active learning strategies: 1) the global annotation budget $B \in \mathbb{N}$ and 2) the number of annotation queries answered at each iteration, $b \in \mathbb{N}$. Besides, ILAB strategy has one specific parameter, $b_{\texttt{uncertain}} \in \mathbb{N}$, the number of uncertain queries (see Section 5.2).

How to Set b ? At each iteration, ILAB active learning strategy queries b instances for annotation. A security administrator annotates them, they are added to the annotated dataset $\mathcal{D}_L$, and the detection model is updated. The parameter b controls the trade-off between reducing waiting-periods and integrating expert feedback frequently.

One the one hand, simulations where oracles answer annotation queries with ground-truth labels are often carried out with $b = 1$. This setting does not suit real-world annotation projects since it would induce too frequent waiting-periods for security administrators. One the other hand, Stokes et al. [130] have set b to 1000 in their user experiments. This high iteration budget damages the expert-model interaction : security administrators spend their day annotating, and their feedback is taken into account only every night to improve the detection model.

Security administrators should set the value of the parameter b on the following principle: experts should not spend more time waiting for queries than annotating, but their feedback must still be integrated rather frequently to show them the benefit of their annotations. The value of b is therefore data dependent: it must be set according to the average time required to answer annotation queries.

How to Set $b_{\texttt{uncertain}}$? The parameter $b_{\texttt{uncertain}}$ fixes the portion of the iteration budget b dedicated to uncertain queries. Some instances near the decision boundary are annotated to help the detection model make decision about these instances [82], but not too many since they are often harder to annotate [15, 66, 120], and they may lead to sampling bias [116].

How to Set B ? The global annotation budget B specifies the stop condition of the annotation process. It can be set to a maximum number of annotations, or to a global time spent annotating.

In the case of an unlimited budget, security administrators can end the annotation process when convergence is reached: several iterations have not led to the discovery of a new family, and the model predictions are consistent with the expert annotations. ILAB *Monitoring Interface* (see Section 6.2.2) informs security administrators about the state of convergence of the annotation procedure: it depicts the number of families discovered, and the accuracy of the suggested annotations.

6.3.2 Initialization

The active learning process needs some initial annotated instances to train the first supervised detection model. This initial supervision can be difficult to acquire for computer security detection problems. The `Malicious` class is usually underrepresented for uniform random sampling to be effective at collecting a representative annotated dataset.

If a public annotated dataset is available for the detection problem considered, it can serve as initial supervision. Otherwise, misuse detection techniques widely deployed in detection systems can provide `Malicious` examples at low cost, and random sampling can provide `Benign` examples. In both cases, the initial annotated dataset does not contain all the malicious families we want to detect, and it is not representative of the data in the deployment environment. We use ILAB to enrich the initial annotated dataset with more diverse malicious behaviors and to make it representative of the environment where the detection system is deployed.

6.4 Setting of the User Experiments

In this section, we ask security administrators to use ILAB to acquire an annotated dataset from unlabeled NetFlow data coming from a production environment. The primary objective is to collect feedback from intended end-users to validate our design choices: both the active learning strategy (see Section 5.2) and the annotation system (see Section 6.2). Another objective is to highlight possible improvements that will be beneficial to other annotation projects.

The competing active learning methods [4, 63, 130] compared to ILAB with simulations in Chapter 5 have not designed or they provide too few details about the user interface. As a result, these active learning strategies are not considered during the user experiments.

6.4.1 Annotation Project

The annotation project consists in acquiring an annotated dataset from unlabeled NetFlow data recorded at the border of a defended network. The objective is to train a supervised detection model that identifies external IP addresses (i.e. IP addresses communicating with the defended network) with an anomalous behavior.

	Day 1	Day 2
Number of flows	$1.2 \cdot 10^8$	$1.2 \cdot 10^8$
Number of IP addresses	463913	507258
Number of features	134	134
Number of TRW [70] alerts	72	82

Table 6.1: NetFlow Datasets.

As stated in [25]: "NetFlow is a network protocol proposed and implemented by Cisco [35] for summarizing network traffic as a collection of network flows. A flow is a unidirectional sequence of packets that share specific network properties (e.g. IP source/destination addresses, and TCP or UDP source/destination ports)." Each flow is described by features and summary statistics: source and destination IP addresses, source and destination ports, protocol (e.g. TCP, UDP, ICMP, or ESP), start and end time stamps, number of bytes, number of packets, and aggregation of the TCP flags for TCP flows.

Security administrators involved in the annotation project must annotate external IP addresses according to their flows during a 24-hour time window.

The NetFlow data are recorded at the border of a defended network during two consecutive working days in 2016 (see Table 6.1). The `Day 1` dataset constitutes the unlabeled pool from which some instances are queried for annotation, and the `Day 2` dataset serves as a validation dataset to assess the performance of the resulting detection model.

6.4.2 ILAB Deployment

In this section, we explain how we deploy ILAB active learning system before the user experiments without involving participants.

Feature Extraction

We compute features describing each external IP address communicating with the defended network from its flows. We compute the mean and the variance of the number of bytes and packets sent and received at different levels: globally, for some specific port numbers (80, 443, 53 and 25), and for some specific TCP flags aggregates (e.g. `....S`, `.A..S.`, or `.AP.SF`). Besides, we compute other aggregated values: number of contacted IP addresses and ports, number of ports used, entropy according to the contacted IP addresses and according to the contacted ports.

In the end, each external IP address is described by 134 features computed from its list of flows.

Problem-Specific Visualization

Figure 6.1 displays the problem-specific visualization we have implemented for the NetFlow annotation project. An instance is represented by its list of flows. A color coding eases the analysis: the extern IP address and the intern IP addresses it communicates with have different colors, and the main protocols (TCP, UDP, and ICMP) have also their own color.

Settings of the Parameters

We set he parameters of ILAB active learning strategy, b and $b_{\texttt{uncertain}}$, according to the guidelines presented in Section 6.3.1: $b = 100$ and $b_{\texttt{uncertain}} = 10$. We do not set the global annotation budget B to a number of manual annotations, but we stop the annotations after 90 minutes while letting annotators complete their current iteration.

Initialization

The active learning process is initialized with some annotated instances. The alerts triggered by the Threshold Random Walk (TRW) [70] module of Bro [100] provide the initial anomalous examples and we draw the normal examples randomly. All the initial labels are checked manually. The initial annotated dataset is composed of 70 *obvious scans* detected by TRW, and of 70 normal examples belonging to the *Web*, *SMTP* and *DNS* families. Malicious activities in well-established connections cannot be detected without the payload, which is not available in NetFlow data, that is why we consider the families *Web*, *SMTP* and *DNS* to be normal.

ILAB is deployed to enrich this initial annotated dataset. The detection model should not be restricted to the detection of obvious scans. ILAB should discover additional anomalous behaviors from the NetFlow data.

6.4.3 Experimental Protocol

Four security administrators take part in the experiments. They are used to working with NetFlow data, but they have no or little knowledge about machine learning. They have never used ILAB or any other annotation system before.

We carry out the experiments independently with each expert for half a day. Each participant starts the annotation process from scratch. In real-world annotation projects, security administrators would take into account the annotations performed by previous annotators. Our objective is, however, to validate our design choices, so participants must annotate under the same conditions.

We run all the experiments on a dual-socket computer with 64Go RAM. Processors are Intel Xeon E5-5620 CPUs clocked at 2.40 GHz with 4 cores each and 2 threads per core. We timestamp and log all the users' actions in ILAB graphical interface to assess the time required for annotating and the waiting-periods.

Presentation of the Experiment to the Participants. First, we inform the participants that they are going to use ILAB to build a machine learning detection model from NetFlow data interactively, and that the aim of the experiments is to get their feedback to improve the tool. We emphasize that NetFlow is but one example, and that ILAB is a generic tool that can be deployed on any data type. Their feedback will therefore be beneficial to other applications.

The task is divided into two parts. First, the experts acquire an annotated dataset with ILAB from the unlabeled pool `Day 1`. Then, they analyze the alerts triggered on `Day 2` by the detection model trained on the annotated instances. Once the participants have completed the task, we collect their feedback.

Data Annotation with ILAB

Information about Extracted Features. The first two participants have no information about the features of the detection model for the purposes of hiding the machine learning complexity. This approach may lead annotators to create families that the detection model cannot properly discriminate due to a lack of information about the features extracted from the NetFlow data.

The last two participants know the features of the model, and we briefly explain the implications on the families they may create. Port numbers are a relevant example. The features include the number of bytes and packets sent and received globally, and for the port numbers 80, 443, 53 and 25. We emphasize that it is therefore counterproductive to create families corresponding to scans on specific services such as *Telnet scans* (port 23) or *SSH scans* (port 22).

Consistent Annotations. Before launching the ILAB annotation process, we ask the experts to check the initial annotated instances, and tell them that they may change the assigned labels and families as they wish. This step is crucial to ensure that the annotations they perform afterward are consistent with the initial ones.

Besides, we highlight that the annotations must be consistent throughout the whole annotation process, otherwise the detection model will perform poorly. We point out that the *Family Editor* and the *Annotated Instances Interface* can help them to remain consistent. Finally, we point out that ILAB suggestions, for labels and families, may be wrong. The objective of the interactive process is to correct the model if it makes a mistake to improve its performance.

Alerts Analysis with DIADEM

Once the security administrator has annotated a dataset with ILAB, we leverage DIADEM (see Chapter 3) to train a supervised detection model and to apply it to `Day 2` data. We train a logistic regression model to follow the advice provided in Section 2.4. Then, the security administrator analyzes the alerts triggered on `Day 2` data from DIADEM alert visualization interface (see Section 3.2.2).

This step is crucial: the objective of security administrators is not to acquire an annotated dataset, but to build a detection model and to assess its performance.

Feedback Collection

Once the experts have annotated a dataset with ILAB and analyzed the alerts triggered with DIADEM, we collect their feedback through an informal discussion. We cover several topics: the relevance of the alerts triggered by the resulting detection model, the ergonomics of the user interface, the waiting-periods, the usefulness of the *Family Editor* and *Annotated Instances* interfaces, and the feedback provided across the iterations.

In Section 6.5, we explain how the user experiments validate our design choices. Then, in Section 6.6, we present additional feedback from the participants and point out some avenues of research to further improve user experience in annotation projects.

User	#Iter.	# Queries	# Created Families	# Final Families
1	3	300	15	10
2	2	200	16	15
3	2	200	29	26
4	2	200	22	17

Table 6.2: Number of Created and Final Families.

6.5 Validation of ILAB Design

6.5.1 Accessible to Non-Machine Learning Experts

The participants have not faced any difficulty in building a detection model with ILAB even if they have little or no knowledge about machine learning. They have reported some minor ergonomic problems not related to machine learning especially. We will address these issues to further improve the user experience. Globally, the participants have been pleased with ILAB, and convinced that it will be beneficial to other annotation projects.

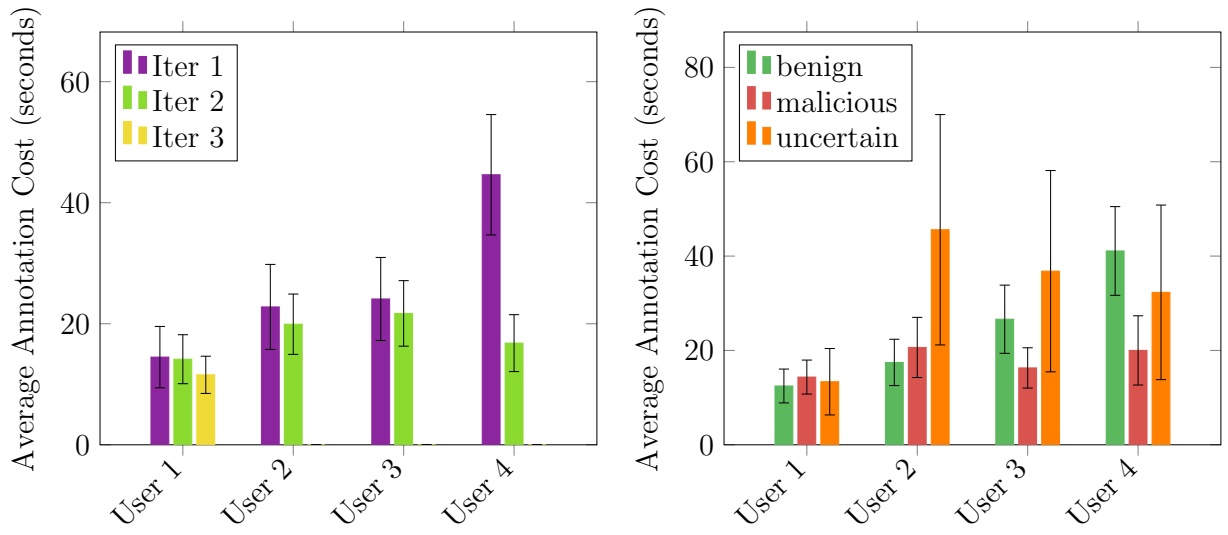
6.5.2 Detection Target and Alert Taxonomy

Across the iterations, ILAB has queried stealthier scans than the ones detected by TRW: *slow scans* (only one flow with a single defended IP address contacted on a single port), and *furtive scans* (a slow scan in parallel with a well-established connection). Besides, it has detected *TCP Syn flooding* activities designed to exhaust the resources of the defended network. Finally, ILAB has asked the participants to annotate IP addresses with anomalous behaviors which are not malicious: *misconfigurations* and *backscatters*.

To sum up, the detection target has evolved across the iterations thanks to ILAB queries. At the beginning of the annotation process, the annotated dataset contains only obvious scan activities, and ILAB queries other anomalous behaviors. The rare category detection analyses carried out by ILAB (see Section 5.2.2) are effective for pointing out new anomalous behaviors.

Table 6.2 presents the number of families created by each participant, and the number of families at the end of the annotation process. It reveals that the participants begin by creating specific families and they end up merging some of them with the *Family Editor* to remove needless detail. The participants have declared that the *Family Editor* and the *Annotated Instances Interface* help them provide consistent annotations throughout the annotation process. Furthermore, they have stated that these tools are crucial if the annotation process lasts several days.

In brief, the participants rely on ILAB to define the malicious and benign families. At the beginning of the annotation project, they have a vague idea of how to group the benign and malicious behaviors into families. Then, ILAB queries bring them to change their families definitions. The user experiments show that the *Family Editor* and the *Annotated Instances Interface* are critical components of ILAB to define the families interactively.



(a) Average Annotation Cost across the Iterations. (b) Average Annotation Cost for each Query Type.

Figure 6.3: Average Annotation Costs.

6.5.3 Annotation Cost

The cost of the annotation process is usually reported as the number of manual annotations [4, 63, 130]. However, all the annotations do not have the same cost in terms of time for making a decision and experts have their own annotation speed. Figure 6.3 presents the average annotation cost, i.e. the average time required to answer annotation queries, with the corresponding 95% confidence interval, for each participant, at each iteration and for each query type.

Figure 6.3a shows that the annotation speed varies significantly from participant to participant. Besides, the annotation cost always decreases across the iterations: they get used to the data they annotate and to ILAB user interface and so they answer queries faster. As the participants annotate, they get a more precise idea of the detection target and of the malicious and benign families, so they spend less time making decision.

Uncertain queries, close to the decision boundary, are often considered harder to annotate [15, 66, 120]. The statistics presented in figure 6.3b support this statement for only two participants out of four. This low agreement may be explained by the fact that we have run only a few iterations, and therefore the model has not yet converged and is still uncertain about instances easy to annotate for security administrators.

Figure 6.3b also points out that the benign queries are harder to annotate than the malicious ones for two out of four participants. One explanation is that security administrators are not used to analyzing benign behaviors and to group them into families. They analyze malicious behaviors when they design misuse detection techniques, and they are accustomed to grouping malicious behaviors into families when they define alert taxonomies.

User	Whole Process	Computations	Waiting-Periods	Efficiency
1	1h28min	197.53 sec	10.81 sec	99.76%
2	1h29min	91.56 sec	7.32 sec	99.86%
3	1h36min	87.54 sec	7.31 sec	99.87%
4	1h57min	93.56 sec	7.37 sec	99.89%

Table 6.3: Computations and Waiting-Periods.

6.5.4 Resulting Detection Model

The participants have analyzed the alerts triggered by their detection model on `Day 2` with DIADEM alert analysis interface (see Section 3.2.2). They have assessed that the alerts are consistent with their malicious annotations and that the number of false positives is low enough to meet operational constraints. The top N alerts are obvious scans where many ports are scanned on many IP addresses. The randomly selected alerts correspond to the most common anomalies, i.e. slow Syn scans on port 23.

The participants have pointed out that grouping alerts according to their predicted malicious families eases their analysis, and reveals more interesting alerts, i.e. less common malicious behaviors, than top N and random. The families of some alerts have, however, been wrongly predicted due to a lack of annotated instances for some malicious families. Some families have been discovered only at the last iteration and too few examples are in the annotated dataset for the detection model to generalize properly.

More iterations are required to improve the automatic qualification of the alerts.

6.5.5 Short Waiting-Periods

Table 6.3 presents an analysis of the cumulated computation times and waiting-periods throughout the whole annotation process. The column *Computations* stores the duration of the computation of all the annotation queries. The column *Waiting-Periods* corresponds to the cumulated waiting-periods : the time during which the users are waiting for the active learning strategy to compute new annotation queries. *Efficiency* represents the percentage of time allocated to the improvement of the detection model (annotating, editing families, inspecting annotated instances) during the annotation process.

The cumulated waiting-periods are smaller than the cumulated computation times since ILAB parallelizes the annotations and the computations (see Section 5.3.3): experts can annotate some instances while the remaining annotation queries are computed. Experts wait only while the detection model is trained on the current annotated instances, and the uncertain queries are generated. Then, they start answering the uncertain queries while ILAB generates the malicious and benign queries. During our experiments, ILAB has always completed the computation of the malicious and benign queries before the experts have finished answering the uncertain queries. As a result, the participants have waited less than 5 seconds between each iteration. All the participants have declared that the waiting-periods are short enough not to damage the expert-model interaction.

ILAB divide-and-conquer approach ensures a good expert-model interaction: the detection model is updated frequently with expert feedback without inducing long waiting-periods.

6.5.6 Feedback to Annotators

Three out of the four participants have declared that they have perceived the benefit of their annotations across the iterations. In their view, they appreciate the improvement of the detection model thanks to the clustering of the queries according to labels and families. They assess the false negatives while annotating the `Benign` queries, and the false positives while annotating the `Malicious` ones. Moreover, they evaluate the relevance of the predicted families with the suggestions.

The participants have not mentioned the two feedback graphs displayed by ILAB (the number of discovered families and the precision of the suggestions), as a means of seeing the benefit of their annotations. These graphs depict a global evolution over several iterations, while the method used by the participants grasps local evolutions between two consecutive iterations. The number of iterations performed during the user experiments may be too low to show the relevance of these graphs.

6.6 Further Feedback from the User Experiments

6.6.1 Security Administrators Annotate Differently

The participants answer the queries very differently. In particular, they disagree on the label corresponding to the *misconfigurations*. Some consider they are anomalous, while others think they are too common in network traffic. Besides, they annotate the instances with different levels of detail. Some create families based on combinations of services (e.g. *Web-DNS*, *Web-SMTP*, *Web-SMTP-DNS*), while others differentiate the services depending on whether the external IP address is the client or the server (e.g. *Web-from-external* and *Web-from-internal*). They also build scan families with different levels of detail (e.g. *obvious-Syn-scans*, *obvious-Syn-scans-Reset-response*, *multihosts-multiports-scans*, *udp-multihosts-scans*).

In short, at the end of the annotation project, the participants have neither the same detection target, nor the same alert taxonomy.

These discrepancies are not surprising since the participants have annotated independently, but we can draw lessons from these user experiments for future annotation projects involving several annotators. We have designed ILAB for a single annotator. How can we adapt it to work this several annotators ?

The main difficulty with several annotators is the definitions of the detection target and the alert taxonomy. They are usually not well delineated at the beginning of annotation projects, and security administrators refine their specifications as they annotate queried instances.

One way is to ask the annotators to agree on the detection target and the alert taxonomy during a preliminary stage. During the first iterations, the annotators answer the annotation queries together. Once the iterations do not lead to the identification of new families, we can stop the preliminary stage. At the end of the preliminary stage, the detection target and the alert taxonomy are more precisely delineated than at the beginning of the annotation project. The annotators can then use ILAB alternately to gather more annotations. If they discover a new family, or they are uncertain about an annotation, they can consult each other to make decision. An interesting avenue of research is to adapt ILAB to work with multi-annotators.

6.6.2 Security Administrators are Willing to Skip Some Annotation Queries

ILAB makes sure that experts answer all the queries at each iteration, and all participants have asked during the experiments a question like "Do I need to answer all the annotation queries ?" or "Is there a middle category between `Malicious` and `Benign` ?".

Experts are often reluctant to answer tricky queries as they worry about making mistakes. ILAB does not propose a middle category, and forces experts to answer each query, to prevent them from taking no risk. Indeed, if we let them discard queries as they like, they may annotate only obvious instances which can lead to a model detecting only the most common malicious behaviors.

Annotators can create a new family to isolate some uncertain annotations, and come back to these instances later with the *Annotated Instances Interface* and the *Family Editor*. One participant has had no difficulty in assigning a binary label, nor in assigning a family to benign instances. However, he has had trouble assigning a family to some malicious instances whose behavior is hard to explain or even to describe. He has created an *unknown* malicious family to handle these tricky cases.

Security administrators may want some instances to stay close to the decision boundary, because they cannot make a decision with their expert knowledge. Skipping these queries is not a solution, as the query strategy may keep on asking to annotate the same kind of instances in the following iterations. A future line of research is to design an active learning algorithm that can take into account a middle category to generate the next queries. Moreover, we should make sure that annotators do not choose too often the middle category to take no risk. A potential solution is to allow to pick the middle category up to k times at each iteration.

6.6.3 Data Annotation and Feature Extraction are Intertwined

The first two participants have no information about the features of the detection model to hide the machine learning complexity. This lack of information has led to the creation of families that the detection model could not discriminate. The first participant has ended up merging these too specific families, there has therefore been no negative impact on the resulting detection model. On the contrary, the second participant has kept the specific families until the end of the annotation process. It has damaged the performance of the detection model.

The last two participants know the features, and they have not created families that the detection model could not discriminate properly by the detection model. They have had no difficulty understanding the features included in the model, nor their implications on the families they can create. They have, however, confirmed their desire to build more specific families that necessitate additional features.

ILAB, as the state-of-the-art active learning strategies [4, 130, 63], assumes that the features are set at the beginning of the annotation process and do not change across the iterations. The user experiments have, nevertheless, shown that the discovery of new families may necessitate adding new features so that the detection model discriminates them properly. The detection target and the alert taxonomy are usually not well delineated at the beginning of the annotation process, so it is hard to anticipate which features should be extracted.

A new avenue of research is to consider active learning strategies where the features change across iterations. Human annotators could change the features manually, or they could be updated

automatically according to the new annotations with a method similar to [28, 72, 78]. In both cases, a particular care shall be taken to maintain short waiting-periods and to avoid numerical instabilities.

6.6.4 Security Administrators are More Than Oracles

Annotation projects where security administrators build detection models differ significantly from crowd-sourcing annotation projects. Security administrators involved in annotation projects are not mere oracles. At each iteration, they do more than answering queries. They delineate the detection target and the alert taxonomy, and they want to understand the behavior of the detection model.

During the user experiments, some participants have wondered why the detection model is uncertain or wrong about a prediction. Security administrators are willing to follow the evolution of the detection model across the iterations.

In order to address this need, ILAB annotation system is not reduced to an *Annotation Interface*. It also offers a *Monitoring Interface* to help security administrators understand the behavior of detection models. Moreover, the *Family Editor* and the *Annotated Instances Interface* assist security administrators in delineating the detection target and the alert taxonomy.

To sum up, ILAB active learning system is not a mere annotation system, it helps security administrators build detection models interactively.

6.7 Conclusion

In this chapter, we integrate ILAB active learning strategy (see Chapter 5) in an annotation system. This way, we get an end-to-end active learning system that security administrators can deploy in real-world annotation projects.

ILAB annotation system is not reduced to an annotation interface, it provides additional user interfaces (*Monitoring Interface*, *Family Editor*, *Annotated Instances Interface*) to assist security administrators. During annotation projects, they do not simply answer annotation queries, but they define the detection target and the alert taxonomy. We have designed ILAB active learning system to help them build detection models interactively.

The user experiments show that ILAB in an effective active learning system that security administrators can deploy in a real-world annotation project. They also point out some avenues of research to further improve user experience in annotation projects. We provide an open-source implementation of the whole active learning system in SecuML [20] to foster research in this area, and to enable security administrators to annotate their own datasets.

The machine learning pipeline is composed of four steps: 1) data annotation, 2) feature extraction, 3) training, and 4) evaluation. DIADEM (see Chapter 3) helps security administrators carry out the last two steps, while ILAB streamlines data annotation. The next and last part of this thesis focuses on the remaining step of the machine learning pipeline: feature extraction. It examines automatic feature extraction as a means to help security administrators extract discriminating features.

Part III

Automatic Feature Generation for Detection Systems

Chapter 7

Automatic Feature Generation for Detection Systems

Most research works on supervised threat detection have implemented a feature extraction method specific to their detection problem. However, detection systems process many data types and designing a feature extraction method for each of them is tedious. Automatic feature generation can significantly ease and thus foster the deployment of machine learning in computer security detection systems.

In this chapter, we start by defining the constraints that automatic feature extraction techniques should meet to be tailored to detection systems. Then, we compare three state-of-the-art methods [129, 28, 72] according to these constraints on PDF files and Windows event logs. Finally, we point out some interesting avenues of research to better tailor automatic feature extraction to security administrators needs.

Contents

7.1 Introduction

Machine learning has been successfully applied to various computer security detection problems: Android applications [54], PDF files [124, 37], Flash files [49, 141], portable executable files [75], botnets [25, 10, 27], Windows audit logs [22], and memory dumps [17]. Most research works implement their own feature extraction method to represent raw data as fixed-length vectors of features.

Computer security detection systems process many data types and designing a feature extraction method for each of them is tedious. Worst, threat evolution may also require to go through this manual process multiple times. Indeed, the discovery of new threats may require to add new features to detect them properly.

Feature extraction is a three step-process: 1) identifying discriminating information, 2) parsing the data to extract this information, and 3) transforming this information into fixed-length vectors of features. The first two steps can be hardly automated since it requires a good knowledge of the data format and specific domain expertise to extract the discriminating information. On the contrary, the last step can be easily automated since it usually exploits generic techniques (see Section 3.3.2).

Automatic feature generation can significantly help security administrators with this tedious task. They still need to be involved in the first two steps, since their domain expertise is critical. However, it can remove the burden of the last step. Besides, if annotated data are available, they can be exploited to identify discriminating information automatically.

In the computer security community, Hidost [129] extracts features automatically from hierarchical file formats such as PDF or XML. In the machine learning community, Khiops [28] and Featuretools [72] exploit already structured data to generate features automatically. To the best of our knowledge, these generic feature extraction techniques [28, 72] have never been applied to threat detection, while they could be beneficial.

In this chapter, we outline the specific constraints that feature extraction methods should meet to operate successfully on detection problems. We compare three state-of-the-art methods [129, 28, 72] in the context of threat detection on two data types, PDF files and Windows event logs. Our comparison shows the strengths and limitations of each approach, and points out some interesting avenues of research to make automatic feature extraction better tailored to security administrators needs.

First, Section 7.2 states the problem of automatic feature extraction and the specific constraints related to detection systems. Section 7.3 presents some automatic feature generation methods developed in diverse communities. Then, Section 7.4 explains the experimental protocol designed to compare three state-of-the-art methods [129, 28, 72] on two data types, PDF files and Windows event logs. Finally, Section 7.5 presents the results of the comparison and Section 7.6 highlights some avenues of research.

7.2 Problem Statement

Standard machine learning algorithms do not take raw data as input, but instances represented as fixed-length vectors of features. Features are binary, numerical, or categorical values describing an instance that detection models exploit to make decisions. The more discriminating the features are, the more efficient the detection model is.

The main objective of automatic feature generation techniques is to derive features automatically from raw data, and to get well-performing detection models. Besides, some constraints specific to computer security detection systems should be met.

Support Files and Event Logs. Detection systems usually process two broad categories of data: files (e.g. PDF, Flash, or Windows Office documents) and event logs (network or operating system event logs). The automatic feature generation algorithm must, thus, support both data types.

Scalability. The size of computer security datasets can be considerable. The method must, therefore, be able to process arbitrarily large datasets.

Reasonable Execution Time. First of all, security administrators must be able to update detection models swiftly to detect new threats or to avoid overwhelming false positives (see Section 1.2.4).

Moreover, automatic feature generation algorithms can be integrated into active learning systems, such as ILAB (see Chapters 5 and 6), to make features evolve automatically during the annotation process (see Section 6.6.3). In this scenario, the feature generation must be quick enough to not damage too much the expert-model interaction.

The execution time of the generation algorithm must therefore be low enough to suit security administrators needs.

Interpretable Features. Security administrators need to trust detection models before their deployment. They want to understand their behavior as a whole. Besides, security operators need information to understand why an alert has been triggered. Binary answers (an alert has been triggered or not) are not enough to handle alerts swiftly (see Section 1.2.3).

As a result, the generated features must be interpretable so that the resulting detection models and their predictions have the same property.

Robust Features. Detection methods are deployed in adversarial environments where attackers are willing to circumvent them (see Section 1.2.5). The generated features must not be too specific to resist evasion attempts.

Specific features would act like signatures and would therefore be vulnerable to polymorphism attacks (see Section 1.3.1). The generated features must be generic enough to enable detection of yet unknown threats.

Moreover, the choice of good features is not trivial. Automatic feature generation algorithms can usually compute an extremely large number of features, but finding the most discriminating ones

is not straightforward. Automatic feature generation methods should, therefore, take advantage of both expert knowledge and annotations if available.

Expert in the Loop. Security administrators are usually reluctant to hand over control to a fully automated learning method [111]. Furthermore, their expertise can guide the feature generation process to create better features. The automatic feature extraction technique should thus enable security administrators to inject their knowledge.

Efficient Use of Annotations. Automatic feature generation techniques can be exploited to build supervised and unsupervised detection models. In the case of supervised learning, the generation process should leverage the annotated data to create discriminating features.

7.3 Related Work

Automatic feature generation is an active area of research in many domains. Most research works focus on deep learning techniques [58] which perform well on image, audio and text classification tasks. However, such techniques are hardly suited to threat detection since the generated features are hard to interpret. In this section, we review methods generating more understandable features from diverse communities: computer security, compilation and machine learning.

7.3.1 Computer Security

FeatureSmith [153] generates features automatically from scientific research papers to detect malicious Android applications. It relies on natural language processing techniques to extract good features from a corpus of relevant documents without human intervention. This method identifies only features which are named entities such as Android API calls or permissions and therefore, tends to generate signature lookalike features. Moreover, their solution cannot be applied if there is no literature yet available. In our case, we are more interested in data-oriented solutions that generate features directly from raw data even if there is no prior work.

Hidost [129] is a data-oriented solution that operates on hierarchical files (e.g. PDF, SWF, XML, or HTML). First, it converts raw files into trees that represent the hierarchical structure of files. Then, it creates features that correspond to paths in the hierarchy. The first step requires a specific module for each file format. The authors provide the modules to support PDF and SWF files.

7.3.2 Compilation

Leather et al. [81] have designed an automatic feature generation technique for optimizing compilation. As Hidost, this method operates on hierarchical data, intermediate representations of programs, but it has a broader feature space. Indeed, it generates features based on subgraphs of the hierarchy, whereas Hidost is limited to paths. Besides, a genetic algorithm leverages annotated data to drive the feature generation process.

7.3.3 Machine Learning

Some generic feature generation techniques, not tailored to any application domains, have been recently proposed [28, 72, 78]. These methods rely on structured and relational data, i.e. a set of tables with relational links, where the rows of the table called *root table* correspond to the instances. They basically follow relationships from the root table to a given variable (a column in a table). Then, they sequentially apply aggregation functions (e.g. `mean`, `var` or `max` for numerical variables, `mode`[1] or `num_unique` for categorical variables) to compute features. These techniques [28, 72, 78] are hereinafter referred to as *relational-based feature generation methods*.

They cannot be directly applied to files or event logs. The raw data need to be transformed into structured and relational data beforehand. We explain in Section 7.4.2 how to perform this transformation.

To sum up, two kinds of data-oriented feature generation methods are relevant for detection systems: hierarchy-based methods [81, 129], and relational-based methods [28, 72, 78]. They have never been compared with regards to the specific constraints of threat detection. In the next sections, we compare [129], [28] and [72] with the implementations provided by the authors. [78] and [81] are not considered in this study since the authors did not provide any implementation.

7.4 Experimental Protocol

We compare three tools designed for automatic feature generation: Hidost [129], Featuretools [72], and Khiops [28]. We use the open-source implementations provided for Hidost[2] and Featuretools[3] and an evaluation license for Khiops[4]. We consider two data types during the comparison: PDF files and Windows event logs. We describe the datasets in Section 7.4.1. Unlike Hidost, Featuretools and Khiops do not take raw data as input, but relational data. We explain in Section 7.4.2 how we meet this requirement.

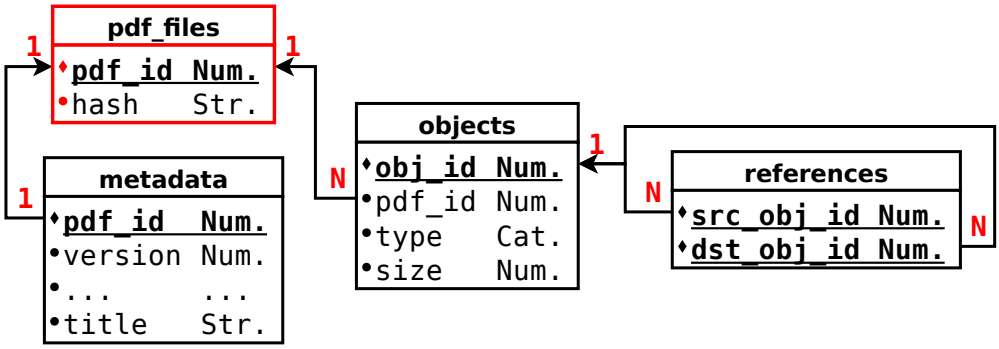
We run Hidost with the configuration recommended by the authors: the paths that appear in fewer than 1% of the instances are discarded. Featuretools is executed with its default parameters. Khiops requires to indicate the number of features to generate. We set this parameter to 200 to get a number of features comparable with Featuretools and Hidost.

We run all the experiments on Linux 3.16 on a dual-socket computer with 64Go RAM. Processors are Intel Xeon E5-5620 CPUs clocked at 2.40 GHz with 4 cores each and 2 threads per core.

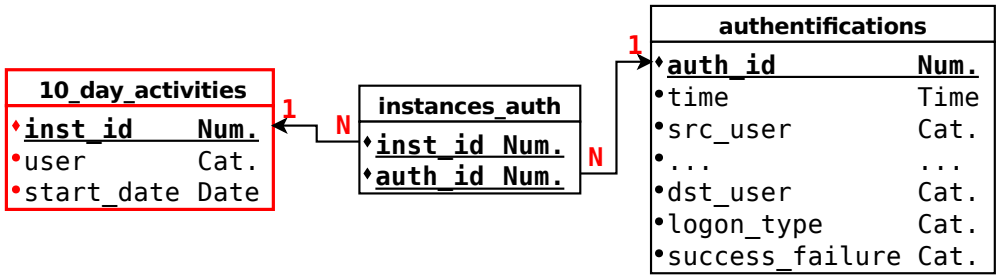
7.4.1 Datasets

PDF Files. We take into account two datasets: Contagio [1] (9,000 benign files and 11,001 malicious files) and WebPDF (2,078 benign files and 767 malicious files). Contagio is a public annotated dataset. WebPDF consists of benign files resulting from requests on the Google search engine, and of malicious files queried from the VirusTotal [3] platform.

[1] The value that has the highest number of occurrences.
[2] https://github.com/srndic/hidost, version from November 29th 2015.
[3] https://github.com/Featuretools/featuretools, version 0.1.14.
[4] https://khiops.com/, version 8 with an evaluation license.



(a) PDF Relational Diagram.



(b) Windows Event Logs Relational Diagram.

Figure 7.1: Relational Diagrams. Root tables are represented in red. Arrows represent relationships between tables. N and 1 correspond respectively to the many and to the one sides of relationships.

Windows Event Logs. Kent et al. [74] have released a dataset[5] consisting of 58 consecutive days of event data collected within Los Alamos National Laboratory's computer network. The objective is to build a supervised detection model that identifies compromise events based on the Windows-based authentication events (about one billion events).

7.4.2 Transforming Files and Logs into Relational Data

Featuretools and Khiops require relational data as input, i.e. a set of tables with relational links, where the rows of the root table correspond to the instances. We explain here how we transform the raw data presented in Section 7.4.1 into this standard format. Figure 7.1 depicts the resulting relational diagrams and Table 7.1 the size of the resulting tables.

PDF Files. PDF files are hierarchically structured. They are composed of objects of different types (e.g. `stream` or `dict`) which reference each other. The references form a graph which may

[5] `https://csr.lanl.gov/data/cyber1/`

Datasets	`files`	`objects`	`references`
WebPDF	2,828	1,086,231	2,655,197
Contagio	20,078	2,379,128	6,128,275

(a) PDF Files.

`10_day_activities`	113,575
`instances_auth`	1,510,947,533
`authentifications`	1,051,430,459

(b) Windows Event Logs.

Table 7.1: Number of Rows in each Table.

contain cycles. Besides, PDF files have metadata such as author, producer, title, creation and modification dates.

We leverage a PDF parser to represent the metadata and the structure of PDF files into a relational format described in Figure 7.1a. More generally, parsers can easily transform files into structured and relational data (trees or graphs). Parsers are already deployed in detection systems for traditional detection methods. We can build upon them to make automatic feature extraction generic regarding file formats.

Windows Event Logs. Event logs are already in a structured format: they contain a list of events described by a predetermined number of fields. Nevertheless, we cannot build a detection model directly from individual events, they do not contain enough information to detect malicious activities. An instance must, therefore, group several event logs.

In our experiments, each instance represents the activity of a given user during a 10-day-time window. Figure 7.1b depicts the relational diagram we obtain. It does not require to pre-process the event logs beforehand and it prevents the duplication of the original data which is crucial when handling large datasets.

7.5 Comparison

7.5.1 Deployment

Hidost. Hidost is an end-to-end solution that relies on specific modules to convert raw files into hierarchical structures. In our experiments, it is straightforward to run Hidost on the PDF datasets, since the authors provide the specific module for this file format. However, the specific modules are tightly coupled with the feature generation process. Hidost is therefore hardly extensible to other types of files.

Additionally, Hidost does not suit event logs data. It is highly specialized in deriving features from hierarchical structures and logs are flat data that do not contain such structures. Hence, we do not deploy Hidost on the Windows event logs dataset.

Featuretools and Khiops. Featuretools and Khiops take relational data as input but none of them accept all relational patterns. Featuretools does not support `1 to 1` relationships: tables linked this way must be merged manually beforehand.

Besides, Featuretools and Khiops do not support diamond patterns that are crucial to describe hierarchies. In Figure 7.1a, the relationship between the tables `objects` and `references` is an example of diamond pattern. In order to avoid this unsupported pattern, we chose to keep only the for-

Methods	**Hidost** [129]		**Khiops** [28]		**Featuretools** [72]	
Datasets	#Features	Time	#Features	Time	#Features	Time
WebPDF	784	7min28s	200	1min13s	100	9h30min
Contagio	561	23min34s	200	4min16s	71	48h

Table 7.2: Number of Generated Features and Execution Times.

eign key referencing `references.src_obj_id` and not the one referencing `references.dst_obj_id`. As a consequence, some information is lost: the number of references for a particular PDF object can be computed but not the number of references pointing to it.

More generally, Khiops does not support association tables such as `instances_auth` in Figure 7.1b. A solution to make the Windows event logs dataset compliant with Khiops is to flatten the relationships. However, this would duplicate rows which is not tractable on large datasets. Thus, we do not generate features for the Windows event logs dataset with Khiops.

To sum up, we generate features for the PDF datasets with Hidost, Featuretools, and Khiops. It is straightforward to run Hidost on PDF files. On the contrary, Featuretools and Khiops do not accept all relational patterns, so we adapt the input data to suit their requirements.

We generate features for the Windows event logs dataset only with Featuretools. Indeed, Hidost cannot generate features from flat data, it needs hierarchical structures. Besides, Khiops does not support association tables, and we cannot afford to duplicate the rows of such a large dataset.

7.5.2 Scalability and Execution Time

Memory Usage. Khiops and Hidost allow to set the maximum memory usage to work on larger datasets at the expense of the execution time. Featuretools is hardly scalable since it loads all the data into RAM. Nonetheless, the authors claim in the documentation that their implementation can be adjusted to work on Spark infrastructures to scale to big data.

Execution Time. Table 7.2 shows that Hidost and Khiops have both completed their computations within minutes, while Featuretools requires several hours. The time difference between Hidost and Khiops is explained by their input data: Hidost parses raw PDF files, while Khiops takes already pre-processed PDF files as input. Anyway, their execution times are both low enough to enable periodic updates of detection models. On the contrary, Featuretools execution times are extremely high even if the PDF datasets fit into RAM.

The Windows event logs dataset is much larger than the PDF datasets, and does not fit into RAM. We did not use Khiops and Hidost on this dataset for reasons given in Section 7.5.1. As for Featuretools, its high execution time and its lack of scalability prevent the feature generation process from completing. These shortcomings have been demonstrated in [78] too and make Featuretools' current implementation unsuitable for detection systems.

In brief, Hidost and Khiops enjoy low execution times which enable periodic updates. On the contrary, Featuretools execution time is far too high which prevents it from completing on the

Windows event log dataset.

As a result, none of the compared methods have managed to generate features for the Windows event logs dataset. From now on, we compare them only on the PDF datasets.

7.5.3 Features Interpretability

Featuretools and Khiops. Featuretools and Khiops generate similar features whose names are easy to understand. For example, `NUM_UNIQUE(objects.type)` corresponds to the number of different object types and `MEAN(objects.COUNT(references))` computes the average number of references of the objects.

These approaches based on relational data may, nevertheless, rapidly generate complex features with nested aggregation functions. We understand how such features are computed, but do not know what they mean in practice. For instance, it is hard to understand what several nested variance computations mean in practical terms.

Hidost. The names of the features generated by Hidost are easily interpretable since they correspond to paths from the file structure. For example, the feature `OpenAction/JS` corresponds to a `JS` object referenced by an `OpenAction` object.

However, the values of the features are hard to interpret since they result from a sophisticated computation process. Each feature does not simply correspond to the number of occurrences of a given path. First, Hidost associates each path with the content of its last element. Then, non-numeric values are arbitrarily replaced by the constant value 1. Finally, Hidost groups the values by path and aggregates them with a `median` function to generate the features.

7.5.4 Resulting Detection Models

Once the features have been generated, we leverage DIADEM (see Chapter 3) to train and diagnose supervised detection models. We train a logistic regression model to follow the advice provided in Section 2.4. Indeed, security administrators, who do not trust black-box detection models [111], highly value linear models. They can understand their overall behavior because the coefficients associated with each feature represent their contribution to the decision-making process.

Detection Performance. Our experiments do not point out significant differences between the three resulting models on the basis of detection performance. They all perform similarly on both PDF datasets with an AUC (Area Under the ROC Curve) near 99%. Moreover, the detection models perform as well as the ones trained with the manually generated features presented in Section 3.3.2.

Overall Behavior. Moreover, we leverage DIADEM diagnosis interface (see Section 3.2.2) to analyze the most discriminating features of each model.

Hidost detection models rely mostly on the presence of JavaScript code (`Names/JavaScript`) and automatic actions at the opening (`OpenAction/S`) that are usual in malicious files. Featuretools and Khiops models rely not only on these aspects, but also on others that Hidost generation algorithm cannot express. For instance, `MEAN(objects.size)` and `NUM_UNIQUE(objects.type)` are useful features since benign files tend to contain more content than malicious ones that often include only a payload.

7.5.5 Features Robustness

Khiops. Khiops associates a complexity cost to each feature to penalize the most sophisticated ones. The original intent of Khiops' author is not to generate robust features but to avoid overfitting. Nonetheless, this is relevant for threat detection since complex features are usually specific and easy to forge to perform evasion attacks.

Hidost. Hidost's authors consider that features based on simple string properties can be easily evaded, so Hidost replaces strings with the constant value 1. Besides, security administrators can provide rules to merge similar features in order to remove artifacts that may lead to evasion attacks (see Section 7.5.6 for more detail).

Overall, Hidost's authors have decided to rely mostly on the structure, and slightly on the content, to hinder evasion attacks. However, they have not carried out any evasion robustness evaluation, and their conclusion claims that leveraging more the content can improve detection accuracy.

In brief, only Hidost and Khiops have paid attention to generating robust features. Besides, there is a real trade-off between robustness and accuracy that should be further analyzed. Including more information, to create more specific features, can improve the performance of the detection model, but it may also lead to evasion attacks.

7.5.6 Expert in the Loop

Hidost. Hidost enables security administrators to provide rules to merge similar paths by defining a list of regular expressions. The objective is to transform paths that may be polymorphic, into a more general form removing artifacts. This injection of expert knowledge offers two advantages. First, it reduces drastically the number of generated features. Second, it reduces the attack surface since the features are less specific. Despite these advantages, we want to emphasize that it is a tedious work to create these merging rules.

Featuretools and Khiops. Khiops and Featuretools enable security administrators to exclude some variables from the feature generation process. They also propose a high level description language to create features manually.

Moreover, Featuretools enables security administrators to define *interesting* values for each variable. These *interesting* values are then exploited to add filters, i.e. `WHERE` clauses, to the feature generation formulas (e.g. `MAX(objects.COUNT(references)) WHERE objects.type = stream`). Khiops also generates features with filters but completely automatically. It does not provide security administrators the opportunity to focus the generation process on some specific filter conditions.

All presented methods offer some mechanisms to enable security administrators to inject domain-specific knowledge, but they are rather restrictive. Experts are forced to follow a specific format to express their knowledge.

7.5.7 Efficient Use of Annotations

Khiops. Khiops is the only method that leverages annotations to drive the feature generation process. It relies on annotations to perform feature selection during the generation process.

However, feature selection is performed on each feature independently without taking correlations into account. Therefore, two features highly correlated can be both selected, while they bring almost the same information. Besides, if two features have little information separately, but are informative taken together, the selection algorithm will not pick them.

7.5.8 Summary

Our experiments show that none of the compared approaches fulfills the constraints stated in Section 7.2. Featuretools suffers from a high execution time which would impede periodic updates of detection models. Hidost is not appropriate to process non-hierarchical data like event logs. Moreover, it is hardly extensible to other types of files, since its specific modules are tightly coupled with the feature generation process. Finally, Khiops and Featuretools do not support all relational patterns which hinders their application to some real-world detection problems.

Hidost is highly specialized in hierarchically formatted data, while Khiops and Featuretools bring a more generic approach relying on relational data. Future research works should thus follow up on the line of Khiops and Featuretools while paying more attention to the constraints related to threat detection. In the next section, we point out some major avenues of research to make relational based feature generation methods better tailored to security administrators needs.

7.6 Avenues of Research

Relational based feature generation methods can be applied to any threat detection problem as long as they support all relational patterns. These methods leverage large feature spaces: they combine various transformations to compute features. In this section, we highlight some avenues of research to improve the exploration of feature spaces while fulfilling the constraints of detection systems.

Restricting the Feature Space. Features resulting from numerous transformations tend to be less interpretable and too specific. In that regard, restricting the feature space efficiently to get robust and interpretable features, without damaging the detection performance, remains an open problem.

Leveraging Annotated Instances. Feature generation algorithms can leverage annotated instances to explore the feature space more efficiently. A feature selection algorithm can select the most discriminating features while they are generated. Besides, annotated instances could drive the generation process to decide which features should be computed first.

Interacting with Experts. Expert knowledge is crucial for feature engineering. Automatic feature generation techniques should thus offer advanced means for experts to guide the generation process. They should enable experts to express their domain-specific knowledge at the beginning of the generation process, but also to provide feedback in the middle. It would make automatic feature generation an interactive tool to help experts explore the feature space [5, 85].

7.7 Conclusion

In this chapter, we outline the constraints that any feature generation solution should meet to suit threat detection. Then, we compare Hidost [129], highly specialized in hierarchically formatted files, with Khiops [28] and Featuretools [72] that bring a more generic approach relying on relational data. Our comparison shows that none of these approaches fulfill the operational constraints of detection systems.

We are, however, confident that automatic feature generation can significantly ease, and thus foster the deployment of machine learning in computer security detection systems. Khiops and Featuretools bring a more generic approach than Hidost, so future research works should follow up on their line. In this perspective, we point out some avenues of research to make relational based feature generation methods better suited to security administrators needs.

Chapter 8

Conclusion

8.1 Summary

Supervised detection models can be deployed in detection systems, as an adjunct to traditional detection techniques, to strengthen detection. The aim of this thesis is to help security administrators deploy supervised detection models that suit their operational constraints. To that end, we take into account the whole machine learning pipeline (data annotation, feature extraction, training and evaluation) with security administrators and operators as its core. We propose an expert-in-the-loop approach on the whole machine learning pipeline since it is crucial to pursue real-world impact.

In the first part, we present the steps of the machine learning pipeline, and we review machine learning techniques and methodologies with a computer security point of view. We place a particular emphasis on evaluating properly supervised detection models to ensure successful deployments. Moreover, we design and implement an interactive visualization tool, DIADEM, to help security administrators apply the methodology set out.

DIADEM handles two steps of the machine learning pipeline: training and evaluation. Security administrators must perform the first two steps: data annotation and feature extraction. They must provide as input to DIADEM a set of annotated instances represented as fixed-length vectors of features. The following parts of the thesis deal with these remaining steps.

The second part concerns the first step of the machine learning pipeline: data annotation. It aims to reduce the workload in computer security annotation projects. We present an end-to-end active learning system, ILAB, tailored to security administrators needs. We have designed the active learning strategy and the annotation system jointly to effectively reduce the annotation effort.

ILAB active learning strategy minimizes not only the number of manual annotations, but also the waiting-periods. The comparison of ILAB with three state-of-the-art active learning strategies [4, 130, 63] demonstrates that it improves the detection performance without increasing the number of manual annotations nor the waiting-periods.

We explain how ILAB active learning strategy has been integrated into an annotation system. ILAB annotation system is not reduced to an annotation interface. It provides additional user interfaces (*Monitoring Interface*, *Family Editor*, *Annotated Instances Interface*) to assist security administrators. Security administrators do not simply answer annotation queries in annotation

projects, they define the detection target and the alert taxonomy. We have designed ILAB active learning system to help security administrators build detection models interactively. The user experiments demonstrate that ILAB in an effective active learning system that helps security administrators carry out real-world annotation projects.

Finally, the last part focuses on feature extraction. It considers automatic feature generation techniques as a means to reduce security administrators' workload while setting up supervised detection models. First, we outline the constraints that such methods should meet to suit security administrators needs. Then, we compare three state-of-the-art methods based on these criteria. Unfortunately, none of these approaches fulfill the constraints. We are, however, confident that automatic feature generation can significantly ease the deployment of machine learning in detection systems. In this perspective, we point out some avenues of research to make automatic feature generation better suited to security administrators needs.

The solutions we propose in this thesis are completely generic to be beneficial to any detection problem on any data type. We have implemented the SecuML framework, available online through GitHub [20], to help security administrators build machine learning models and interact with them. We provide open-source implementations of DIADEM and ILAB in SecuML to ease comparison in future research works, and to enable security administrators to build their own detection models.

8.2 Perspectives

This thesis presents solutions to ease, and thus foster, the deployment of supervised detection models in detection systems. Our end-to-end approach has also pointed out some avenues of research to better meet security administrators needs.

Make Detection Models More Robust to Evasion Attempts

Attackers can evade supervised detection models. They can craft adversarial examples designed to circumvent detection models while keeping the malicious payloads. There is currently no perfect defense against adversarial examples (see Section 2.4.4).

Even if adversarial examples are difficult to craft in the context of threat detection (feature mappings are hard to invert) they remain a major problem. Indeed, detection models are deployed in adversarial environments where attackers are constantly improving. It is therefore critical to enhance the robustness of supervised detection models.

Better Tailor Automatic Feature Generation to Detection Systems

We are convinced that automatic feature generation can significantly reduce security administrators' workload to set up supervised detection models. However, we have compared three state-of-the-art methods, and none of them fulfills the constraints related to threat detection.

Our comparison has pointed out some avenues of research to better tailor automatic feature generation to security administrators needs (see Section 7.6). First, the feature space should be restricted to generate interpretable and robust features. Second, expert knowledge and annotations should be better leveraged to generate discriminating features.

Update Features Automatically during Annotation Projects

The first two steps of the machine learning pipeline, data annotation and feature extraction, are usually considered completely independent. However, our user experiments, with ILAB active learning system, have pointed out that these two steps can be strongly intertwined (see Section 6.6.3). The detection target and the alert taxonomy are usually not well delineated at the beginning of annotation projects. In this case, new annotations may question the extracted features, and require to generate new features.

A solution is to update features automatically during annotation projects. Even if an automatic feature generation technique tailored to threat detection is available, there remain some challenges to overcome. First, the active learning system must identify when the features should be updated. Second, experiments should be carried out to inspect the behavior of an active learning system where both the annotations and the features evolve across iterations.

Bibliography

[1] Contagio. `http://contagiodump.blogspot.com/`.

[2] NSL-KDD. `http://www.unb.ca/cic/research/datasets/nsl.html`.

[3] VirusTotal. `https://www.virustotal.com/`.

[4] Magnus Almgren and Erland Jonsson. Using active learning in intrusion detection. In *Computer Security Foundations Workshop (CSFW)*, pages 88–98, 2004.

[5] Saleema Amershi, Maya Cakmak, William Bradley Knox, and Todd Kulesza. Power to the people: The role of humans in interactive machine learning. *AI Magazine*, 35(4):105–120, 2014.

[6] Saleema Amershi, Max Chickering, Steven M Drucker, Bongshin Lee, Patrice Simard, and Jina Suh. Modeltracker: Redesigning performance analysis tools for machine learning. In *International Conference of Human-Computer Interaction (CHI)*, pages 337–346, 2015.

[7] Saleema Amershi, Bongshin Lee, Ashish Kapoor, Ratul Mahajan, and Blaine Christian. Cuet: human-guided fast and accurate network alarm triage. In *International Conference of Human-Computer Interaction (CHI)*, pages 157–166, 2011.

[8] Massih-Reza Amini. *Apprentissage machine: de la théorie à la pratique*. Editions Eyrolles, 2015.

[9] James P Anderson. Computer security threat monitoring and surveillance. Technical report, 1980.

[10] Manos Antonakakis, Roberto Perdisci, Yacin Nadji, Nikolaos Vasiloglou, Saeed Abu-Nimeh, Wenke Lee, and David Dagon. From throw-away traffic to bots: detecting the rise of dga-based malware. In *USENIX Security Symposium*, pages 491–506, 2012.

[11] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, Konrad Rieck, and CERT Siemens. Drebin: Effective and explainable detection of android malware in your pocket. In *Network and Distributed System Security Symposium (NDSS)*, volume 14, pages 23–26, 2014.

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

[13] Stefan Axelsson. Intrusion detection systems: A survey and taxonomy. Technical report, 2000.

[14] Stéphane Ayache and Georges Quénot. Video corpus annotation using active learning. pages 187–198, 2008.

[15] Jason Baldridge and Alexis Palmer. How well does active learning actually work?: Time-based evaluation of cost-reduction strategies for language documentation. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, volume 1, pages 296–305, 2009.

[16] Aharon Bar-Hillel, Tomer Hertz, Noam Shental, and Daphna Weinshall. Learning a mahalanobis metric from equivalence constraints. *Journal of Machine Learning Research (JMLR)*, 6(6):937–965, 2005.

[17] Thomas Barabosch, Niklas Bergmann, Adrian Dombeck, and Elmar Padilla. Quincy: Detecting host-based code injection attacks in memory dumps. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*, pages 209–229, 2017.

[18] Anaël Beaugnon, Pierre Chifflier, and Francis Bach. Ilab: An interactive labelling strategy for intrusion detection. In *International Symposium on Research in Attacks, Intrusions, and Defenses (RAID)*, pages 120–140, 2017.

[19] Anaël Beaugnon, Pierre Chifflier, and Francis Bach. End-to-end active learning for computer security experts. In *Workshop on Artificial Intelligence for Cyber Security (AICS)*. 2018.

[20] Anaël Beaugnon, Pierre Collet, and Antoine Husson. SecuML. `https://github.com/ANSSI-FR/SecuML`.

[21] Aurélien Bellet, Amaury Habrard, and Marc Sebban. A survey on metric learning for feature vectors and structured data. *arXiv preprint arXiv:1306.6709*, 2013.

[22] Konstantin Berlin, David Slater, and Joshua Saxe. Malicious behavior detection using windows audit logs. In *Workshop on Artificial Intelligence and Security (AISEC)*, pages 35–44, 2015.

[23] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. Evasion attacks against machine learning at test time. In *Joint European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD)*, pages 387–402. 2013.

[24] Battista Biggio and Fabio Roli. Wild patterns: Ten years after the rise of adversarial machine learning. *arXiv preprint arXiv:1712.03141*, 2017.

[25] Leyla Bilge, Davide Balzarotti, William Robertson, Engin Kirda, and Christopher Kruegel. Disclosure: detecting botnet command and control servers through large-scale netflow analysis. In *Annual Computer Security Applications Conference (ACSAC)*, pages 129–138, 2012.

[26] Anaël Bonneton and Antoine Husson. Le machine learning confronté aux contraintes opérationnelles des systèmes de détection. In *Symposium sur la Sécurité des Technologies de l'Information et des Communications (SSTIC)*, pages 317–346, 2017.

[27] Anaël Bonneton, Daniel Migault, Stephane Senecal, and Nizar Kheir. Dga bot detection with time series decision trees. In *Workshop on Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS)*, pages 42–53, 2015.

[28] Marc Boullé. Towards automatic feature construction for supervised classification. In *Joint European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD)*, pages 181–196, 2014.

[29] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[30] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[31] Guillaume Brogi and Valérie Viet Triem Tong. Terminaptor: Highlighting advanced persistent threats through information flow tracking. In *International Conference on New Technologies, Mobility and Security (NTMS)*, pages 1–5, 2016.

[32] Olivier Chapelle, Bernhard Schölkopf, Alexander Zien, et al. *Semi-supervised learning*. MIT press Cambridge, 2006.

[33] Dong Chen, Rachel KE Bellamy, Peter K Malkin, and Thomas Erickson. Diagnostic visualization for non-expert machine learning practitioners: A design study. In *Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 87–95, 2016.

[34] Mihai Christodorescu and Somesh Jha. Testing malware detectors. *Software Engineering Notes*, 29(4):34–44, 2004.

[35] Benoit Claise. Cisco systems netflow services export version 9. 2004.

[36] Seamus Clancy, Sam Bayer, and Robyn Kozierok. Active learning with a human in the loop. Technical report, MITRE Corporation, 2012.

[37] Igino Corona, Davide Maiorca, Davide Ariu, and Giorgio Giacinto. Lux0r: Detection of malicious pdf-embedded javascript code through discriminant analysis of api references. In *Workshop on Artificial Intelligence and Security (AISEC)*, pages 47–57, 2014.

[38] Frédéric Cuppens and Alexandre Miege. Alert correlation in a cooperative intrusion detection framework. In *Symposium on Security and Privacy (S&P)*, pages 202–215, 2002.

[39] Sanjoy Dasgupta. Two faces of active learning. *Theoretical Computer Science*, 412(19):1767–1781, 2011.

[40] Sanjoy Dasgupta and Daniel Hsu. Hierarchical sampling for active learning. In *International Conference on Machine Learning (ICML)*, pages 208–215, 2008.

[41] Jason V Davis, Brian Kulis, Prateek Jain, Suvrit Sra, and Inderjit S Dhillon. Information-theoretic metric learning. In *International Conference on Machine Learning (ICML)*, pages 209–216, 2007.

[42] Hervé Debar, Marc Dacier, and Andreas Wespi. Towards a taxonomy of intrusion-detection systems. *Computer Networks*, 31(8):805–822, 1999.

[43] Hervé Debar and Andreas Wespi. Aggregation and correlation of intrusion-detection alerts. In *International Symposium on Recent Advances in Intrusion Detection (RAID)*, pages 85–103, 2001.

[44] Aaron Defazio, Francis Bach, and Simon Lacoste-Julien. Saga: A fast incremental gradient method with support for non-strongly convex composite objectives. In *Conference on Neural Information Processing Systems (NIPS)*, pages 1646–1654, 2014.

[45] Ambra Demontis, Marco Melis, Battista Biggio, Davide Maiorca, Daniel Arp, Konrad Rieck, Igino Corona, Giorgio Giacinto, and Fabio Roli. Yes, machine learning can be more secure! a case study on android malware detection. *Transactions on Dependable and Secure Computing (TDSC)*, 2017.

[46] Dorothy E Denning. An intrusion-detection model. *Transactions on Software Engineering (TSE)*, (2):222–232, 1987.

[47] Gregory Druck, Burr Settles, and Andrew McCallum. Active learning by labeling features. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, volume 1, pages 81–90, 2009.

[48] Prahlad Fogla, Monirul I Sharif, Roberto Perdisci, Oleg M Kolesnikov, and Wenke Lee. Polymorphic blending attacks. In *USENIX Security Symposium*, pages 241–256, 2006.

[49] Sean Ford, Marco Cova, Christopher Kruegel, and Giovanni Vigna. Analyzing and detecting malicious flash advertisements. In *Annual Computer Security Applications Conference (ACSAC)*, pages 363–372, 2009.

[50] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

[51] Yoav Freund, H Sebastian Seung, Eli Shamir, and Naftali Tishby. Selective sampling using the query by committee algorithm. *Machine Learning*, 28(2-3):133–168, 1997.

[52] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The Elements of Statistical Learning*, volume 1. Springer series in statistics, 2001.

[53] Pedro Garcia-Teodoro, J Diaz-Verdejo, Gabriel Maciá-Fernández, and Enrique Vázquez. Anomaly-based network intrusion detection: Techniques, systems and challenges. *Computers & Security*, 28(1-2):18–28, 2009.

[54] Hugo Gascon, Fabian Yamaguchi, Daniel Arp, and Konrad Rieck. Structural detection of android malware using embedded call graphs. In *Workshop on Artificial Intelligence and Security (AISEC)*, pages 45–54, 2013.

[55] Carrie Gates and Carol Taylor. Challenging the anomaly detection paradigm: a provocative discussion. In *New Security Paradigms Workshop (NSPW)*, pages 21–29, 2006.

[56] Ali A Ghorbani, Wei Lu, and Mahbod Tavallaee. *Network Intrusion Detection and Prevention: Concepts and Techniques*, volume 47. Springer Science & Business Media, 2009.

[57] Jacob Goldberger, Geoffrey E Hinton, Sam T Roweis, and Ruslan R Salakhutdinov. Neighbourhood components analysis. In *Conference on Neural Information Processing Systems (NIPS)*, pages 513–520, 2005.

[58] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[59] Ian J Goodfellow and Nicolas apernot. 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.

[60] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[61] Nico Görnitz, Marius Kloft, and Ulf Brefeld. Active and semi-supervised data domain description. In *Joint European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD)*, pages 407–422, 2009.

[62] Nico Görnitz, Marius Kloft, Konrad Rieck, and Ulf Brefeld. Active learning for network intrusion detection. In *Workshop on Artificial Intelligence and Security (AISEC)*, pages 47–54, 2009.

[63] Nico Görnitz, Marius Micha Kloft, Konrad Rieck, and Ulf Brefeld. Toward supervised anomaly detection. *Journal of Artificial Intelligence Research (JAIR)*, 2013.

[64] Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick McDaniel. Adversarial examples for malware detection. In *European Symposium on Research in Computer Security (ESORICS)*, pages 62–79, 2017.

[65] Bruno Guillaume, Karën Fort, and Nicolas Lefebvre. Crowdsourcing complex language resources: Playing to annotate dependency syntax. In *International Conference on Computational Linguistics (COLING)*, 2016.

[66] Ben Hachey, Beatrice Alex, and Markus Becker. Investigating the effects of selective sampling on the annotation task. In *Conference on Computational Natural Language Learning (CoNLL)*, pages 144–151, 2005.

[67] James A Hanley and Barbara J McNeil. The meaning and use of the area under a receiver operating characteristic (roc) curve. *Radiology*, 143(1):29–36, 1982.

[68] Jingrui He and Jaime G Carbonell. Nearest-neighbor-based active learning for rare category detection. In *Conference on Neural Information Processing Systems (NIPS)*, pages 633–640, 2007.

[69] Eric Jones, Travis Oliphant, and Pearu Peterson. Scipy: open-source scientific tools for python. 2014.

[70] Jaeyeon Jung, Vern Paxson, Arthur W Berger, and Hari Balakrishnan. Fast portscan detection using sequential hypothesis testing. In *Symposium on Security and Privacy (S&P)*, pages 211–225, 2004.

[71] Alex Kantchelian, Sadia Afroz, Ling Huang, Aylin Caliskan Islam, Brad Miller, Michael Carl Tschantz, Rachel Greenstadt, Anthony D Joseph, and JD Tygar. Approaches to adversarial drift. In *Workshop on Artificial Intelligence and Security (AISEC)*, pages 99–110, 2013.

[72] James Max Kanter and Kalyan Veeramachaneni. Deep feature synthesis: Towards automating data science endeavors. In *International Conference on Data Science and Advanced Analytics (DSAA)*, pages 1–10, 2015.

[73] Ashish Kapoor, Bongshin Lee, Desney Tan, and Eric Horvitz. Interactive optimization for steering machine classification. In *International Conference of Human-Computer Interaction (CHI)*, pages 1343–1352, 2010.

[74] Alexander D Kent. Cyber security data sources for dynamic network research. In *Workshop on Dynamic Networks and Cyber-Security*, pages 37–65. 2016.

[75] Khaled N Khasawneh, Meltem Ozsoy, Caleb Donovick, Nael Abu-Ghazaleh, and Dmitry Ponomarev. Ensemble learning for low-level hardware-supported malware detection. In *International Symposium on Research in Attacks, Intrusions, and Defenses (RAID)*, pages 3–25. 2015.

[76] Christopher Kruegel, Engin Kirda, Darren Mutz, William Robertson, and Giovanni Vigna. Automating mimicry attacks using static binary analysis. In *USENIX Security Symposium*, volume 14, pages 11–11, 2005.

[77] Todd Kulesza, Saleema Amershi, Rich Caruana, Danyel Fisher, and Denis Charles. Structured labeling for facilitating concept evolution in machine learning. In *International Conference of Human-Computer Interaction (CHI)*, pages 3075–3084, 2014.

[78] Hoang Thanh Lam, Tran Ngoc Minh, Mathieu Sinn, Beat Buesser, and Martin Wistuba. Learning features for relational data. *arXiv preprint arXiv:1801.05372*, 2018.

[79] Pavel Laskov and Nedim Šrndić. Static detection of malicious javascript-bearing pdf documents. In *Annual Computer Security Applications Conference (ACSAC)*, pages 373–382, 2011.

[80] Aleksandar Lazarevic, Vipin Kumar, and Jaideep Srivastava. Intrusion detection: A survey. In *Managing Cyber Threats*, pages 19–78. 2005.

[81] Hugh Leather, Edwin Bonilla, and Michael O'Boyle. Automatic feature generation for machine learning based optimizing compilation. In *International Symposium on Code Generation and Optimization (CGO)*, pages 81–91, 2009.

[82] David D Lewis and William A Gale. A sequential algorithm for training text classifiers. In *International Conference on Research and Development in Information Retrieval*, pages 3–12, 1994.

[83] Daniel Lowd and Christopher Meek. Good word attacks on statistical spam filters. In *Conference on Email and Anti-Spam (CEAS)*, 2005.

[84] Scott Lundberg and Su-In Lee. An unexpected unity among methods for interpreting model predictions. *arXiv preprint arXiv:1611.07478*, 2016.

[85] Oisin Mac Aodha, Vassilios Stathopoulos, Gabriel J Brostow, Michael Terry, Mark Girolami, and Kate E Jones. Putting the scientist in the loop–accelerating scientific progress with interactive machine learning. In *International Conference on Pattern Recognition (ICPR)*, pages 9–17, 2014.

[86] Thorsten May, Andreas Bannach, James Davey, Tobias Ruppert, and Jörn Kohlhammer. Guiding feature subset selection with an interactive visualization. In *Conference on Visual Analytics Science and Technology (VAST)*, pages 111–120, 2011.

[87] Brad Miller, Alex Kantchelian, Sadia Afroz, Rekha Bachwani, Edwin Dauber, Ling Huang, Michael Carl Tschantz, Anthony D Joseph, and JD Tygar. Adversarial active learning. In *Workshop on Artificial Intelligence and Security (AISEC)*, pages 3–14, 2014.

[88] Bradley Austin Miller. Scalable platform for malicious content detection integrating machine learning and manual review. 2015.

[89] Christoph Molnar. Interprtable machine learning: A guide for making black box models explainable. `https://christophm.github.io/interpretable-ml-book/`, 2018.

[90] Robert Moskovitch, Nir Nissim, and Yuval Elovici. Malicious code detection using active learning. In *Workshop on Privacy, Security, and Trust*, pages 74–91. 2009.

[91] Antonio Nappa, M Zubair Rafique, and Juan Caballero. The malicia dataset: identification and analysis of drive-by download operations. *International Journal of Information Security (IJIS)*, 14(1):15–33, 2015.

[92] Nir Nissim, Aviad Cohen, and Yuval Elovici. Aldocx: detection of unknown malicious microsoft office documents using designated active learning methods based on new structural feature extraction methodology. *Transactions on Information Forensics and Security*, 12(3):631–646, 2017.

[93] Nir Nissim, Aviad Cohen, Robert Moskovitch, Assaf Shabtai, Mattan Edry, Oren Bar-Ad, and Yuval Elovici. Alpd: Active learning framework for enhancing the detection of malicious pdf files. In *Joint Intelligence and Security Informatics Conference (JISIC)*, pages 91–98, 2014.

[94] Nir Nissim, Robert Moskovitch, Oren BarAd, Lior Rokach, and Yuval Elovici. Aldroid: efficient update of android anti-virus software using designated active learning methods. *Knowledge and Information Systems*, 49(3):795–833, 2016.

[95] Nir Nissim, Robert Moskovitch, Lior Rokach, and Yuval Elovici. Novel active learning methods for enhanced pc malware detection in windows os. *Expert Systems with Applications*, 41(13):5843–5857, 2014.

[96] Stephen M Omohundro. *Five balltree construction algorithms*. International Computer Science Institute Berkeley, 1989.

[97] Nicolas Papernot, Nicholas Carlini, Ian Goodfellow, Reuben Feinman, Fartash Faghri, Alexander Matyasko, Karen Hambardzumyan, Yi-Lin Juang, Alexey Kurakin, Ryan Sheatsley, et al. cleverhans v2. 0.0: an adversarial machine learning library. *arXiv preprint arXiv:1610.00768*, 2016.

[98] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *European Symposium on Security and Privacy (EuroS&P)*, pages 372–387, 2016.

[99] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *Symposium on Security and Privacy (S&P)*, pages 582–597, 2016.

[100] Vern Paxson. Bro: a system for detecting network intruders in real-time. *Computer Networks*, 31(23):2435–2463, 1999.

[101] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research (JMLR)*, 12(Oct):2825–2830, 2011.

[102] Dan Pelleg and Andrew W Moore. Active learning for anomaly and rare-category detection. In *Conference on Neural Information Processing Systems (NIPS)*, pages 1073–1080, 2005.

[103] Guo-Jun Qi, Jinhui Tang, Zheng-Jun Zha, Tat-Seng Chua, and Hong-Jiang Zhang. An efficient sparse metric learning in high-dimensional space via l 1-penalized log-determinant regularization. In *International Conference on Machine Learning (ICML)*, pages 841–848, 2009.

[104] J. Ross Quinlan. Simplifying decision trees. *International Journal of Man-Machine Studies*, 27(3):221–234, 1987.

[105] Maria Eugenia Ramirez-Loaiza, Aron Culotta, and Mustafa Bilgic. Anytime active learning. In *Conference on Artificial Intelligence (AAAI)*, pages 2048–2054, 2014.

[106] Sebastian Raschka. What is the difference between a parametric learning algorithm and a non-parametric learning algorithm? `https://sebastianraschka.com/faq/docs/parametric_vs_nonparametric.html`.

[107] Sebastian Raschka. About feature scaling and normalization. `http://sebastianraschka.com/Articles/2014_about_feature_scaling.html`, 2014.

[108] Sebastian Raschka. Model evaluation, model selection, and algorithm selection in machine learning. `https://sebastianraschka.com/pdf/manuscripts/model-eval.pdf`, 2018.

[109] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Model-agnostic interpretability of machine learning. *arXiv preprint arXiv:1606.05386*, 2016.

[110] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Why should i trust you?: Explaining the predictions of any classifier. In *International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 1135–1144, 2016.

[111] Konrad Rieck. Computer security and machine learning: Worst enemies or best friends? In *SysSec Workshop*, pages 107–110, 2011.

[112] Peter J Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, 1987.

[113] Nicholas Roy and Andrew McCallum. Toward optimal active learning through monte carlo estimation of error reduction. pages 441–448, 2001.

[114] Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. 2016.

[115] Mark Schmidt, Nicolas Le Roux, and Francis Bach. Minimizing finite sums with the stochastic average gradient. *Mathematical Programming*, 162(1-2):83–112, 2017.

[116] Hinrich Schütze, Emre Velipasaoglu, and Jan O Pedersen. Performance thresholding in practical text classification. In *International Conference on Information and Knowledge Management (CIKM)*, pages 662–671, 2006.

[117] D Sculley. Online active learning methods for fast label-efficient spam filtering. In *Conference on Email and Anti-Spam (CEAS)*, pages 1–4, 2007.

[118] D Sculley, Matthew Eric Otey, Michael Pohl, Bridget Spitznagel, John Hainsworth, and Yunkai Zhou. Detecting adversarial advertisements in the wild. In *International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 274–282, 2011.

[119] Burr Settles. Active learning literature survey. Technical report, University of Wisconsin, Madison, 2010.

[120] Burr Settles. From theories to queries: Active learning in practice. In *Workshop on Active Learning and Experimental Design*, pages 1–18, 2011.

[121] Burr Settles. Active learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 6(1):1–114, 2012.

[122] Burr Settles, Mark Craven, and Lewis Friedland. Active learning with real annotation costs. In *Workshop on Cost-Sensitive Learning*, pages 1–10, 2008.

[123] Maria Skeppstedt, Carita Paradis, and Andreas Kerren. Pal, a tool for pre-annotation and active learning. *Journal for Language Technology and Computational Linguistics (JLCL)*, 31(1):91–110, 2017.

[124] Charles Smutz and Angelos Stavrou. Malicious pdf detection using metadata and structural features. In *Annual Computer Security Applications Conference (ACSAC)*, pages 239–248, 2012.

[125] Charles Smutz and Angelos Stavrou. Malicious pdf detection using metadata and structural features. Technical report, 2012.

[126] Rion Snow, Brendan O'Connor, Daniel Jurafsky, and Andrew Y Ng. Cheap and fast—but is it good?: evaluating non-expert annotations for natural language tasks. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 254–263, 2008.

[127] Robin Sommer and Vern Paxson. Outside the closed world: On using machine learning for network intrusion detection. In *Symposium on Security and Privacy (S&P)*, pages 305–316, 2010.

[128] Jungsuk Song, Hiroki Takakura, Yasuo Okabe, Masashi Eto, Daisuke Inoue, and Koji Nakao. Statistical analysis of honeypot data and building of kyoto 2006+ dataset for nids evaluation. In *Workshop on Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS)*, pages 29–36, 2011.

[129] Nedim Šrndić and Pavel Laskov. Hidost: a static machine-learning-based detector of malicious files. *EURASIP Journal on Information Security*, 2016(1):22, 2016.

[130] Jack W Stokes, John C Platt, Joseph Kravis, and Michael Shilman. Aladin: Active learning of anomalies to detect intrusions. Technical report, Microsoft Network Security, 2008.

[131] Erik Štrumbelj and Igor Kononenko. Explaining prediction models and individual predictions with feature contributions. *Knowledge and Information Systems*, 41(3):647–665, 2014.

[132] Christopher T Symons and Justin M Beaver. Nonparametric semi-supervised learning for network intrusion detection: combining performance improvements with realistic in-situ training. In *Workshop on Artificial Intelligence and Security (AISEC)*, pages 49–58, 2012.

[133] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[134] Peter Szor. Advanced code evolution techniques and computer virus generator kits. *The Art of Computer Virus Research and Defense*, 2005.

[135] Justin Talbot, Bongshin Lee, Ashish Kapoor, and Desney S Tan. Ensemblematrix: interactive visualization to support machine learning with multiple classifiers. In *International Conference of Human-Computer Interaction (CHI)*, pages 1283–1292, 2009.

[136] Mahbod Tavallaee, Ebrahim Bagheri, Wei Lu, and Ali-A Ghorbani. A detailed analysis of the kdd cup 99 data set. In *Symposium on Computational Intelligence for Security and Defense Applications (CISDA)*, 2009.

[137] David MJ Tax and Robert PW Duin. Support vector data description. *Machine Learning*, 54(1):45–66, 2004.

[138] Katrin Tomanek and Fredrik Olsson. A web survey on the use of active learning to support annotation of text data. In *Workshop on Active Learning for Natural Language Processing*, pages 45–48, 2009.

[139] Simon Tong and Daphne Koller. Support vector machine active learning with applications to text classification. *Journal of Machine Learning Research (JMLR)*, 2(Nov):45–66, 2001.

[140] Fredrik Valeur, Giovanni Vigna, Christopher Kruegel, and Richard A Kemmerer. Comprehensive approach to intrusion detection alert correlation. *Transactions on Dependable and Secure Computing (TDSC)*, 1(3):146–169, 2004.

[141] Timon Van Overveldt, Christopher Kruegel, and Giovanni Vigna. Flashdetect: Actionscript 3 malware detection. In *International Symposium on Research in Attacks, Intrusions, and Defenses (RAID)*, pages 274–293, 2012.

[142] Pavan Vatturi and Weng-Keen Wong. Category detection using hierarchical mean shift. In *International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 847–856, 2009.

[143] Kalyan Veeramachaneni and Ignacio Arnaldo. Ai2: Training a big data machine to defend. In *International Conference on Intelligent Data and Security (IDS)*, pages 49–54, 2016.

[144] David Wagner and Paolo Soto. Mimicry attacks on host-based intrusion detection systems. In *Conference on Computer and Communications Security (CCS)*, pages 255–264, 2002.

[145] Kiri L Wagstaff. Machine learning that matters. In *International Conference on Machine Learning (ICML)*, pages 529–536, 2012.

[146] Kilian Q Weinberger and Lawrence K Saul. Distance metric learning for large margin nearest neighbor classification. *Journal of Machine Learning Research (JMLR)*, 10(Feb):207–244, 2009.

[147] Colin Whittaker, Brian Ryner, and Marria Nazif. Large-scale automatic classification of phishing pages. In *Network and Distributed System Security Symposium (NDSS)*, volume 10, 2010.

[148] Christian Wressnegger, Guido Schwenk, Daniel Arp, and Konrad Rieck. A close look on n-grams in intrusion detection: anomaly detection vs. classification. In *Workshop on Artificial Intelligence and Security (AISEC)*, pages 67–76, 2013.

[149] Stephen Wright and Jorge Nocedal. *Numerical Optimization*, volume 35. Springer Science & Business Media, 1999.

[150] Eric P Xing, Michael I Jordan, Stuart Russell, and Andrew Y Ng. Distance metric learning with application to clustering with side-information. In *Conference on Neural Information Processing Systems (NIPS)*, pages 505–512, 2002.

[151] Tong Zhang and F Oles. The value of unlabeled data for classification problems. In *International Conference on Machine Learning (ICML)*, pages 1191–1198, 2000.

[152] Xiaojin Zhu, John Lafferty, and Zoubin Ghahramani. Combining active learning and semi-supervised learning using gaussian fields and harmonic functions. In *Workshop on the Continuum from Labeled to Unlabeled Data in Machine Learning and Data Mining*, pages 58–65, 2003.

[153] Ziyun Zhu and Tudor Dumitras. Featuresmith: Automatically engineering features for malware detection by mining the security literature. In *Conference on Computer and Communications Security (CCS)*, pages 767–778, 2016.

Résumé

L'objectif de cette thèse est de faciliter l'utilisation de l'apprentissage supervisé dans les systèmes de détection pour renforcer la détection. Dans ce but, nous considérons toute la chaîne de traitement de l'apprentissage supervisé (annotation, extraction d'attributs, apprentissage, et évaluation) en impliquant les experts en sécurité.

Tout d'abord, nous donnons des conseils méthodologiques pour les aider à construire des modèles de détection supervisés qui répondent à leurs contraintes opérationnelles. De plus, nous concevons et nous implémentons DIADEM, un outil de visualisation interactif qui aide les experts en sécurité à appliquer la méthodologie présentée. DIADEM s'occupe des rouages de l'apprentissage supervisé pour laisser les experts en sécurité se concentrer principalement sur la détection.

Par ailleurs, nous proposons une solution pour réduire le coût des projets d'annotations en sécurité informatique. Nous concevons et implémentons un système d'apprentissage actif complet, ILAB, adapté aux besoins des experts en sécurité. Nos expériences utilisateur montrent qu'ils peuvent annoter un jeu de données avec une charge de travail réduite grâce à ILAB.

Enfin, nous considérons la génération automatique d'attributs pour faciliter l'utilisation de l'apprentissage supervisé dans les systèmes de détection. Nous définissons les contraintes que de telles méthodes doivent remplir pour être utilisées dans le cadre de la détection de menaces. Nous comparons trois méthodes de l'état de l'art en suivant ces critères, et nous mettons en avant des pistes de recherche pour mieux adapter ces techniques aux besoins des experts en sécurité.

Mots Clés

Sécurité informatique, systèmes de détection, apprentissage supervisé, systèmes interactifs, apprentissage actif, génération automatique d'attributs.

Abstract

The overall objective of this thesis is to foster the deployment of supervised learning in detection systems to strengthen detection. To that end, we consider the whole machine learning pipeline (data annotation, feature extraction, training, and evaluation) with security experts as its core since it is crucial to pursue real-world impact.

First, we provide methodological guidance to help security experts build supervised detection models that suit their operational constraints. Moreover, we design and implement DIADEM, an interactive visualization tool that helps security experts apply the methodology set out. DIADEM deals with the machine learning machinery to let security experts focus mainly on detection.

Besides, we propose a solution to effectively reduce the labeling cost in computer security annotation projects. We design and implement an end-to-end active learning system, ILAB, tailored to security experts needs. Our user experiments on a real-world annotation project demonstrate that they can annotate a dataset with a low workload thanks to ILAB.

Finally, we consider automatic feature generation as a means to ease, and thus foster, the use of machine learning in detection systems. We define the constraints that such methods should meet to be effective in building detection models. We compare three state-of-the-art methods based on these criteria, and we point out some avenues of research to better tailor automatic feature generation to computer security experts needs.

Keywords

Computer security, detection systems, supervised learning, interactive systems, active learning, automatic feature generation.