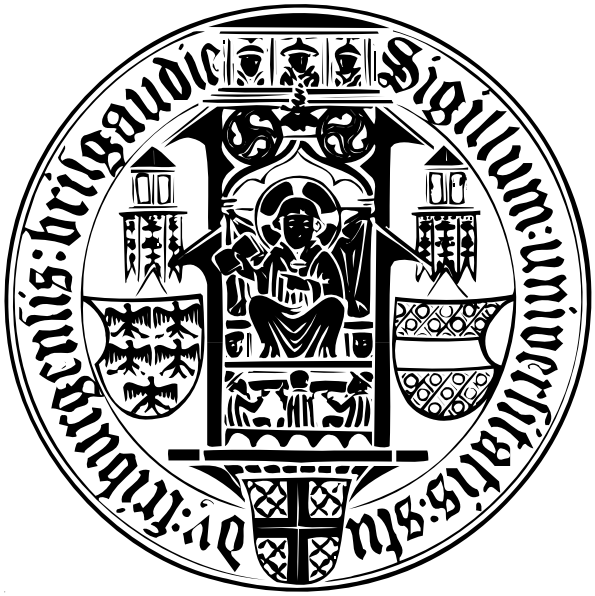
# ALERT CLASSIFICATION TO REDUCE FALSE POSITIVES IN INTRUSION DETECTION

July 2006




Dissertation zur Erlangung des Doktorgrades
der Fakultät für Angewandte Wissenschaften
der Albert-Ludwigs-Universität Freiburg im Breisgau

Tadeusz Pietraszek
tadek@pietraszek.org

Institut für Informatik, Albert-Ludwigs-Universität Freiburg
Georges-Köhler-Allee 52, 79110 Freiburg i. Br., Germany

| | |
|---|---|
| Dekan: | Prof. Dr. Jan G. Korvink |
| Erstreferent: | Prof. Dr. Luc De Raedt |
| Zweitreferent: | Prof. Dr. Johannes Fürnkranz |

Tag der Disputation: 05.12.2006

"A new star has been discovered,
which doesn't mean that things have gotten brighter
or that something we've been missing has appeared.
..."

Wisława Szymborska, "Surplus" [Szy00]

# Erklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus anderen Quellen direkt oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet. Insbesondere habe ich hierfür nicht die entgeltliche Hilfe von Vermittlungs- oder Beratungsdiensten (Promotionsberaterinnen oder Promotionsberater oder anderer Personen) in Anspruch genommen. Niemand hat von mir unmittelbar oder mittelbar geldwerte Leistungen für Arbeiten erhalten, die im Zusammenhang mit dem Inhalt der vorgelegten Dissertation stehen. Die Arbeit wurde bisher weder im In- noch im Ausland in gleicher oder ähnlicher Form einer anderen Prüfungsbehörde vorgelegt.

Desweitern habe ich mich nicht bereits und bewerbe ich mich auch nicht gleichzeitig an einer in- oder ausländischen wissenschaftlichen Hochschule um die Promotion.

*I hereby certify that the work embodied in this thesis is the result of original research and has not been submitted for a higher degree to any other university or institution.*

*Tadeusz Pietraszek*

Zürich, Switzerland
July 4, 2006

# Contents

# Acknowledgments

Defending a PhD is a one-man show, however, the process of pursuing one is definitely not a one-person effort and I have a lot of people to thank for helping me in this stage of my life.

First of all, I would like to thank my professor, Luc De Raedt, who saw value in my research and agreed to supervise it, providing support and giving directions for research. Being a remote PhD student working at IBM Zurich Research Lab is a special situation and I am really grateful that such an arrangement was possible. For all this and more, thank you, Luc.

I would also like to thank Andreas Wespi, my former manager and mentor at IBM, for hiring me and supporting me during my PhD quest, always finding time for meetings and very thoroughly scrutinizing my work. I had greatly benefited from his experience in the field of intrusion detection and computer security. I would also like to thank Lucas Heusler, my current manager for giving me a lot of flexibility in doing my research, making the finishing of my PhD possible.

During my PhD work I was greatly supported by my mentor, Klaus Julisch, who spent a considerable amount of time explaining the arcane of scientific work, forcing me to write and tirelessly correcting my scribbles. He also never hesitated to ask those difficult questions, which helped me to become a more mature researcher.

I would like to thank the IBM Global Services Managed Security Services team, in particular Mike Fiori and Chris Calvert for allowing me to use their data and Jim Treinen and Ken Farmer for providing support on the technical side.

I would also like to thank my friends at IBM, James Riordan & Daniela Bourges-Waldegg for being great friends, expanding my horizons (both scientific and non-scientific) through always interesting discussions and giving me (and other PhD students) a motto "The goal of PhD is to finish it". I have had a great time with Diego Zamboni who, in spite of (or maybe rather because of) thinking of me as a very apt procrastinator and giving me motivation to work on my numerous pet projects, always found time to say "Tadek, work on your thesis".

During my stay in the lab, I have also met many interesting friends and colleagues: Chris Giblin, Marcel Graf, Christian Hörtnagl, Ulf Nielsen, Mike Nidd, René Pawlitzek, Ulrich Schimpel, Morton Schwimmer, Abhi Shelat, Dieter Sommer, Axel Tanner, and others, who provided an excellent and stimulating working environment and always had time for interesting discussions. Among my colleagues, special thanks go to my office-mate, Chris Vanden Berghe for putting up with me in one office during these three years, always-interesting discussions and arguments, and many interesting ideas that got born this way.

I am also grateful to the friendly people who volunteered to read through, and give me invaluable comments on this dissertation and its earlier versions: my professor Luc De Raedt, Birgit Baum-Waidner, Axel Tanner (also for the help with the German abstract) and Andreas

Wespi. Without your help this thesis would not have gotten to this stage. Clearly, I am solely responsible for any mistakes that had remained in the report.

Last but not least, I am deeply indebted to my family for their everlasting support while abroad and having by far more faith in me than anybody else, including myself! My special thanks go to Annie for being the best girlfriend and a wonderful life companion and for putting up with me during the hectic time while working on my PhD.

*Tadeusz Pietraszek*

Zürich, Switzerland
July 4, 2006

# Abstract

*Intrusion Detection Systems* (IDSs) aim at detecting intrusions, that is actions that attempt to compromise the confidentiality, integrity and availability of computer resources. With the proliferation of the Internet and the increase in the number of networked computers, coupled with the surge of unauthorized activities, IDSs have become an integral part of today's security infrastructures. However, in real environments IDSs have been observed to trigger an abundance of alerts. Most of them are *false positives*, i.e., alerts not related to security incidents. This dissertation deals with the problem of false positives in intrusion detection.

We propose the novel concept of training an *alert classifier* using a human analyst's feedback and show how to build an efficient alert classifier using machine-learning techniques. We analyze the desired properties of such a system from the domain perspective and introduce ALAC, an Adaptive Learner for Alert Classification, and its two modes of operation: a recommender mode, in which all alerts with their classification are forwarded to the analyst, and an agent mode, in which the system uses autonomous alert processing. We evaluate ALAC in both modes on real and synthetic intrusion detection datasets and obtain promising results: In our experiments ALAC reduced the number of false positives by up to 60% with acceptable misclassification rates.

*Abstaining classifiers* are classifiers that in certain cases can refrain from classification, which is similar to a domain expert saying "I don't know". Abstaining classifiers are advantageous over normal classifiers if they perform better than normal classifiers when they make a decision.

In this dissertation we provide a clarification of the concept of optimal abstaining classifiers and introduce three different models, in which normal and abstaining classifiers can be compared: the cost-based model, the bounded-abstention model, and the bounded-improvement model. In the first cost-based model, the classifier uses an extended $2 \times 3$ cost matrix, whereas in the bounded models, the classifier uses a standard $2 \times 2$ cost matrix and boundary conditions: the abstention window or the desired cost improvement. Looking at a common type of abstaining classifiers, namely classifiers constructed from a single ROC curve, we provide efficient algorithms for selecting these classifiers optimally in each of these models. We perform an experimental validation of these methods on a variety of common benchmark datasets.

Applying abstaining classifiers to ALAC, we introduce ALAC+, an extension of our alert-classification system. We select the most suitable abstaining classifier models and show that by using abstaining classifiers one can significantly reduce the misclassification cost. For example, in our experiments with a 10% abstention the system reduced the overall misclassification cost by up to 87%. This makes abstaining classifiers particularly suitable for alert classification.

In the final part of this dissertation, we extend CLARAty, the state-of-the-art alert clustering system by introducing automated cluster processing, and show how the system can be used to investigate missed intrusions and correct initial analyst's classifications. Based on this, we build a *two-stage alert-classification system* in which alerts are processed by the automated cluster-processing system and then forwarded to ALAC. Our experiments with real and synthetic datasets showed that the automated cluster-processing system is robust and on average reduces the total number of alerts by 63% which further reduces the analyst's workload.

# Zusammenfassung

*Eindringerkennungssysteme* (*Intrusion Detection Systems*, abgekürzt *IDSs*) zielen auf die Erkennung von Angriffen, d.h. Aktionen, die versuchen die Konfidenzialität, Integrität und Verfügbarkeit von Computer-Resourcen zu kompromittieren. Durch das enorme Wachstum des Internets und der Zahl der vernetzten Computer bei gleichzeitiger starker Zunahme von nicht-autorisierten Aktivitäten sind IDSs zu einem integralen Bestandteil der typischen aktuellen Sicherheits-Infrastruktur geworden. In realen Umgebungen beobachtet man jedoch, daß IDSs sehr viele Alarme produzieren, dabei zu einem großen Teil auch *Fehlalarme (false positives)*, d.h. Alarme, die keinen Sicherheits-Zwischenfällen entsprechen. Diese Dissertation beschäftigt sich mit dem Problem von Fehlalarmen in der Intrusion Detektion.

Wir schlagen hierzu ein neuartiges Konzept vor, bei dem ein *Alarm-Klassifizierer* aus der Rückmeldung eines menschlichen Analysten lernen kann, und zeigen, wie ein solcher effizienter Alarm-Klassifizierer mit Hilfe der Techniken maschinellen Lernens erstellt werden kann. Wir analysieren die wünschenswerten Eigenschaften eines solchen Systems aus dem Blickwinkel der Domäne der Intrusion Detektion und stellen ALAC vor, den Adaptiven Lerner für Alarm-Klassifikation (Adaptive Learner for Alert Classification). ALAC hat zwei Betriebsarten: eine empfehlende Betriebsart (recommender mode), bei der alle Alarme mit ihrer Klassifikation an den Analysten weitergeleitet werden, und eine Betriebsart als Agent (agent mode), in welcher das System Alarme teilweise eigenständig verarbeitet. Wir evaluieren ALAC in beiden Modi mit realen und synthetischen Daten aus dem Gebiet der Intrusion Detektion und erhalten dabei viel versprechende Ergebnisse: ALAC reduziert in diesen Experimenten die Zahl der Fehlalarme um bis zu 60% bei annehmbaren Raten der Fehlklassifikation.

*Sich-enthaltende Klassifizierer* (abstaining classifiers) nehmen in bestimmten Fällen keine Klassifizierung vor, ähnlich einem "Ich weiß nicht" eines Domain-Experten. Es besteht die Annahme, daß ein solcher Klassifizierer, der sich enthalten kann, insgesamt eine bessere Leistung bringen kann als normale Klassifizierer, die in jedem Fall eine Entscheidung treffen müssen.

In dieser Dissertation klären wir das Konzept des optimalen sich-enthaltenden Klassifizierers und stellen drei verschiedene Modelle vor, in denen sie mit normalen Klassifizierern verglichen werden können: ein kosten-basiertes Modell, ein Modell mit begrenzter Enthaltung und ein Modell mit begrenzter Verbesserung. Im kosten-basierten Modell benutzt der Klassifizierer eine erweiterte $2 \times 3$ Kosten-Matrix, während in den anderen Modellen der Klassifizierer eine normale $2 \times 2$ Kosten-Matrix verwendet mit zusätzlichen Randbedingungen: der Menge der Alarme, bei denen sich der Klassifizierer enthält, beziehungsweise die gewünschte Verbesserung der Kosten. Für eine übliche Gruppe von sich-enthaltenden Klassifizierern, die

aus einer einzelnen ROC-Kurve hervorgehen, zeigen wir effiziente Algorithmen um diese Klassifizierer in optimaler Art auszuwählen in allen genannten Modellen. Diese Methoden werden experimentell bestätigt mit einer großen Zahl von Benchmark-Daten.

Unter Anwendung von sich-enthaltenden Klassifizierern auf ALAC führen wir ALAC+ ein, eine Erweiterung unseres Alarm-Klassifikations-Systems. Wir wählen die am besten geeigneten sich-enthaltenden Klassifizierer und zeigen, daß dadurch die Fehlklassifikations-Kosten signifikant reduziert werden können. So reduzieren sich beispielsweise in unseren Experimenten bei 10% Enthaltung die allgemeinen Fehlklassifikations-Kosten um bis zu 87%. Dies macht sich-enthaltende Klassifizierer besonders geeignet für die Alarm-Klassifizierung.

Im letzten Teil der Arbeit erweitern wir CLARAty, ein aktuelles Alarm-Clustering-System, durch die Einführung einer automatisierten Cluster-Verarbeitung und zeigen, wie das System dazu benutzt werden kann eventuell übersehene Angriffe zu untersuchen und initiale Klassifikationen eines Analysten zu korrigieren. Hierauf aufbauend entwickeln wir ein zweistufiges Alarm-Klassifikations-System, in welchen Alarme zuerst durch die automatisierte Cluster-Verarbeitung prozessiert und dann an ALAC weitergeleitet werden. Unsere Experimente mit realen und synthetischen Daten zeigen, daß das automatisierte Cluster-Verarbeitungs-System robust ist und die Gesamtzahl von Alarmen, und damit auch die Arbeitslast des Analysten, durchschnittlich um 63% reduziert.

# List of Figures

# List of Tables

# Chapter 1

# Introduction

*"There was once a shepherd-boy who kept his flock at a little distance from the village. Once he thought he would play a trick on the villagers and have some fun at their expense. So he ran toward the village crying out, with all his might,– 'Wolf! Wolf! Come and help! The wolves are at my lambs!'*

*The kind villagers left their work and ran to the field to help him. But when they got there the boy laughed at them for their pains; there was no wolf there.*

*Still another day the boy tried the same trick, and the villagers came running to help and got laughed at again. Then one day a wolf did break into the fold and began killing the lambs. In great fright, the boy ran for help. 'Wolf! Wolf!' he screamed. 'There is a wolf in the flock! Help!'*

*The villagers heard him, but they thought it was another mean trick; no one paid the least attention, or went near him. And the shepherd-boy lost all his sheep.*

*There is no believing a liar, even when he speaks the truth." [Aes]*

## 1.1 Motivation

Intrusion Detection Systems (IDSs), a concept originally introduced by Anderson [And80] and later formalized by Denning [Den87], have received increasing attention over the past 15 years. IDSs are systems that aim at detecting *intrusions*, i.e., sets of actions that attempt to compromise the integrity, confidentiality or availability of a computer resource [HLMS90]. The explosive increase in the number of networked machines and the use of the Internet in every organization have lead to an increase of unauthorized activities, not only from external attackers but also from internal sources, such as fraudulent employees or people misusing their privileges for personal gain. On the other hand, with the massive deployment of IDSs, their operational limits and problems have become apparent [Axe99, BHC+00, Jul03b, MCZH00].

One of the most important problems faced by intrusion detection today [MMDD02] are so-called *false positives*, i.e., alerts that mistakenly indicate security issues and require attention from the intrusion-detection analyst. In fact, it has been estimated that up to 99% of alerts reported by IDSs are not related to security issues [Axe99, BHC+00, Jul03b]. The challenge is to reduce the number of false positives and improve the quality of alerts.

### 1.1.1 False Positives

One of the most important requirements for IDSs is that they should be *effective*, that is, detect a substantial percentage of intrusions, while keeping the false-positive rate at an acceptable level. Clearly, these requirements are contradictory, and in the pursuit of detecting as many intrusions as possible current IDSs produce too many false positives.

However, building an effective IDS that generates only a small number of false positives is an extremely difficult task. Reasons for this include:

**Runtime limitations:** In many cases an intrusion differs only slightly from normal activities; sometimes even only the context in which the activity occurs determines whether it is intrusive. However, owing to the harsh real-time requirements, IDSs cannot analyze the context of all activities to the extent required [PN98].

**Specificity of detection signatures:** Writing signatures that describe intrusive patterns for misuse-based IDSs (cf. Section 2.1.1) is a very difficult task [Pax99]. In some cases, the right balance between an overly specific signature (which is not able to capture all attacks or their variations) and an overly general one (which classifies legitimate actions as intrusions) can be difficult to determine.

**Dependence on the environment:** Actions that are normal in certain environments may be malicious in others [Bel93]. For example, performing a network scan is malicious unless the computer performing it has been authorized to do so. IDSs deployed with a standard out-of-the-box configuration will most likely identify many normal activities as malicious.

**Base-rate fallacy:** From the statistical point of view, even very low false-positive rates of a detector do not result in equally favorable *Bayesian detection rates* because intrusions are rare phenomena.

Following an example by Axelsson [Axe99], assuming that an IDS analyzes 1,000,000 packets a day, which contain 20 intrusion packets, this yields a probability of intrusion $P(I) = 2 \cdot 10^{-5}$. Knowing the sensor's detection rate ($P(A \mid I)$) and its false-positive rate ($P(A \mid \neg I)$), we can use Bayes theorem to calculate the *Bayesian detection rate* ($P(I \mid A)$) or, in other words, the probability that an alarm really indicates an intrusion:

$$P(I \mid A) = \frac{P(I) \cdot P(A \mid I)}{P(I) \cdot P(A \mid I) + P(\neg I) \cdot P(A \mid \neg I)} \quad . \tag{1.1}$$

Using the above intrusion probability $P(I) = 2 \cdot 10^{-5}$ and assuming an unrealistically high detection rate $P(A \mid I) = 1.0$ and a very low *false-positive rate* $P(A \mid \neg I) = 10^{-5}$, we obtain $P(I \mid A) = 0.66$, which means that one third of all alerts are not related to intrusive activities. With the same false alarm rate and a more realistic detection rate of 0.7, we obtain that 42% of the alerts will be false positives.

### 1.1.2 Existing Solutions

The problem of false positives is critical in intrusion detection and has received considerable attention from researchers as well as practitioners. We have grouped these approaches into four levels as shown in Figure 1.1. We will provide a basic introduction to intrusion detection in Chapter 2 and further discuss these approaches in Chapter 3 in more detail. Here we will show the hierarchy of different solutions and the position of our work within this hierarchy.

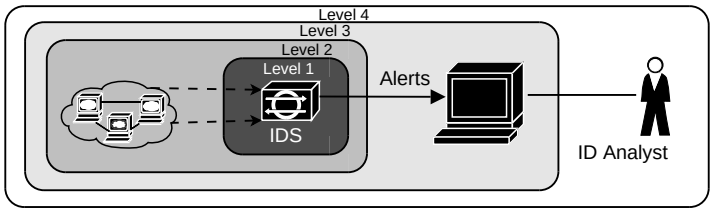



Figure 1.1: Evolution of the scope for addressing false positives in intrusion detection. Shaded areas represent the scope discussed in the text.

**Level 1: Improving IDSs themselves** At first, most of the efforts focused on building better sensors, i.e., sensors that detect more intrusions or sensors with very low false-positive rates. In network-based intrusion detection, the sensors evolved from simple pattern-matching engines to specialized sensors that understand the underlying transport protocols and some of the possible obfuscation techniques (e.g., IP-level fragmentation, TCP-level fragmentation). With the increasing number of application-level vulnerabilities [NIS04], IDSs started to understand different application-level protocols (e.g., HTTP, RPC). At the same time, the signature languages became increasingly powerful, supporting regular expressions (e.g., Snort [Roe05]) or even complex protocol interactions (e.g., Bro [Pax99]).

In contrast to building general, all-purpose IDSs, specialized IDSs focus on particular types of intrusions or attacks promising very low false-positive rates. For example, Sekar et al. [SGVS99] proposed a network-based IDS that exclusively focuses on low-level attacks such as reconnaissance scans and denial-of-service attacks. Billygoat [RZD05], in contrast, focuses on detecting worms and viruses. Finally, Pietraszek and Vanden Berghe [PV05] and Valeur et al. [VMV05] proposed IDSs targeted at detecting application-level SQL injection attacks.

Note that these special-purpose IDSs need to be complemented by additional IDSs to achieve a comprehensive attack coverage, which then creates the need to deploy and maintain a heterogeneous network of complementary IDSs.

**Level 2: Leveraging the Environment** IDSs have a limited view of the environment and, in many cases, cannot distinguish between attacks and nonattacks with certainty. By using information about the environment (provided by vulnerability scanners, OS-fingerprinting or asset databases), IDSs can better understand the environment and significantly lower their false-positive rates.

For example, Ptacek and Newsham [PN98] showed that without knowing how target hosts handle certain anomalies in network packets, intruders can efficiently use fragmentation to avoid being detected by network-based IDS. Addressing these concerns, Active Mapping [SP01] builds profiles of the environment, which can then be leveraged by IDSs. Context signatures [SP03], on the other hand, can understand application-level protocol interactions and thus determine the impact of an attack. A similar effect can be achieved by correlating alerts with vulnerabilities [LWS02, VVCK04].

**Level 3: Alert Postprocessing** Alert postprocessing uses alerts generated by an IDS as input and tries to improve their quality by processing them. This includes systems using data mining and so-called *alert correlation systems*. For example, in the data-mining space Julisch [Jul03b] showed how root cause analysis can be used to effectively discover large groups

of false positives and remove up to 70% of reoccurring false positives in the future.

Moreover, alert correlation, in addition to false positives, addresses another problem of IDS, namely, the redundancy in alert stream. Alert correlation systems (e.g., [CAMB02, DW01, VS01, VVCK04]) aim at producing high-level events and thus reduce the total number of events the system generates.

**Level 4: Analyst's Involvement** Very few of the systems operating on the previous levels take advantage of the fact that alerts generated by IDSs are passed to the human analyst to be analyzed in real time or with only a short delay.

This dissertation introduces a novel paradigm of using machine-learning techniques to reduce the number of false positives in intrusion detection by building a classifier learning from a human analyst and assisting in the alert classification.

The idea is orthogonal to and can be used with all the techniques discussed above: improving IDSs, leveraging the environment and most other alert postprocessing techniques.

### 1.1.3 Introducing the Analyst: The Global Picture of Alert Management

On the practical side, in spite of their operational limits, IDSs have been developed and deployed in different environments, forming an integral part of defense in depth. In fact, there are a few dozens of IDSs available [Cuf05], including many commercial and open source IDSs (e.g., Bro [Pax99], Snort [Roe05]).

Clearly, IDSs can only be useful if the alerts generated by them are collected and reviewed. In practice, many companies use Security Operations Centers (SOCs) (either in-house or outsourced), where *security analysts* review alerts typically 24 hours a day and 7 days a week, so that they can respond to intrusions as soon as possible.

This architecture is depicted in Figure 1.2, in which alerts generated by IDSs are passed on to a human security analyst. The analyst uses his or her knowledge to distinguish between false and true positives and to understand the severity of the alerts. We will refer to this process as *alert classification* or classification for short.

Depending on this classification, the analyst may report security incidents, investigate intrusions, or identify network and configuration problems. The analyst may also try to modify bad IDS signatures or install alert filters to remove alerts matching some predefined criteria.

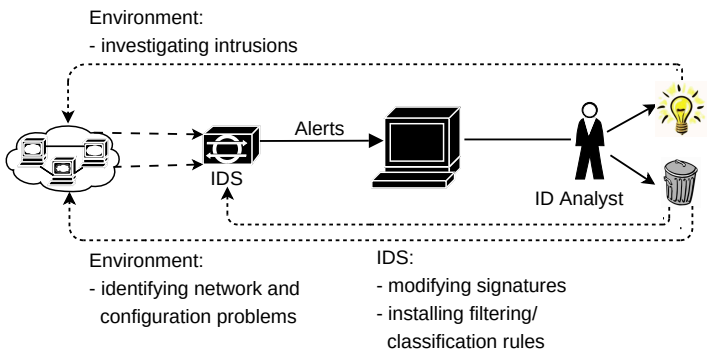



Figure 1.2: The global picture of alert management.

Note that in this conventional setup, manual knowledge acquisition (shown by dashed lines in the figures) is separated from, or not directly used for, classifying alerts. The conventional setup does not take advantage of two facts: First, usually a large database of historical alerts exists that can be used for automatic knowledge acquisition and, second, an analyst is analyzing the alerts as they occur.

These two facts form the basis of the alert-handling paradigm using two different approaches: *unsupervised learning* (descriptive modeling) and *supervised learning* (predictive modeling). While the first paradigm alone has been introduced and validated by Julisch [Jul03b], this dissertation proposes the second alert-handling paradigm, namely, using machine learning on alerts labeled by the operator. We then proceed to evaluate how these approaches can be combined and used together in a *two-stage alert-processing system*.

In this thesis we propose to address the problem of false positives in intrusion detection by building an *alert classifier* (or *classifier* for short) to assist the human analyst classifying alerts. The idea is the following: the classifier classifies alerts into a set of predefined classes in real-time and presents these classifications to the intrusion detection analyst. Based on the analyst's feedback, the system generates new training examples, which are used to initially build and subsequently update the classifier. The resulting classifier is then used to classify new alerts. This process is continuously repeated to improve the classification accuracy. We require that the classification algorithm use explicit and symbolic knowledge representation, so that the analyst can inspect it and verify its correctness.

More formally we formulate the problem as:

| | |
|---|---|
| **Given** | – A sequence of alerts: $(A_1, A_2, \ldots, A_i, \ldots)$ in the alert log $L$,<br>– a set of classes $C = \{C_1, C_2, \ldots, C_n\}$,<br>– an intrusion detection analyst $\mathcal{O}$ sequentially and in real-time assigning classes to alerts,<br>– a utility function $\mathcal{U}$ describing the value of a classifier to the analyst $\mathcal{O}$. |
| **Find** | A classifier classifying alerts, maximizing the utility function $\mathcal{U}$. |

This specification may look like a normal incremental classification task; however, the desired solution depends on the definition of the utility function $\mathcal{U}$, which is based on the assumptions how the system will be used in practical settings. More specifically, we identified the following three components of the utility function $\mathcal{U}$:

**Misclassified alerts:** classifiers that make a smaller number of misclassifications are better, however, not all misclassifications are the same. This can be modeled by the confusion and cost matrices or a scalar misclassification cost.

**Analyst's workload:** assuming that the alerts are analyzed by the human analysts, systems automatically processing some of the alerts reduce the analyst's workload, which can be quantified.

**Abstentions:** if the classifier does not attempt to classify certain alerts, these remaining alerts will have to be classified by the human analysts, hence it increases the analysts workload. However, if such a classifier has a lower misclassification cost, it might be preferred over the one that classifies all alerts, but it sometimes wrong.

Ultimately, the goal is to combine all these components into a single *utility value* and design the system that maximizes it. The problem, however, is that such a model would

require a number of ad-hoc assumptions (e.g., a linear combination of cost and workload) and introduce a number of parameters, which are difficult to quantify or estimate. As a result such a model would have a low practical value.

Therefore, we do not attempt to provide guarantees that all models presented in this dissertation are optimal in terms of the global utility function $\mathcal{U}$. Instead we use heuristics, based of our expertise in intrusion detection, which ensure that the system is easy to understand by the domain experts and can be used in practice. We then focus on the local optimization: the optimal model selection and model parameters to ensure that the system yields a high utility function $\mathcal{U}$.

To motivate the reader and show different definitions of the utility function $\mathcal{U}$ used in this dissertation, we show the following four examples.

In the first setting, assuming that the human analyst will ultimately analyze all alerts classified by the system (analyzing both alerts and their predicted classification and correcting it), the desired solution is in fact an incremental classification system minimizing the misclassification cost (e.g., the utility function is the inverse of the misclassification cost). From the utility standpoint, the analyst would receive some suggestion as to how the alerts should be classified, but at the end he will need to analyze the same number of alerts as before.

In the second setting, assuming that the work of the analyst is a costly resource, a better system could process some alerts automatically, thus limiting the amount of work the analyst has to perform. A natural candidate for this automatic processing is to automatically discard some false positives, such as those are alerts that do not report any security-related problems. Clearly, such a system may incur a higher risk of missing an attack, namely, in the case when an alert gets erroneously discarded.

In the third setting, the system may allow abstentions, that is the target classifier, in addition to classifying alerts into a set of classes $C$, may also assign an additional class "?" representing abstention. Such alerts would then be classified by the analyst as if the alert-classification system was not used. The system would be advantageous if alerts that are classified are classified with a much higher precision (i.e., the probability that the alert classified as class $C_i$ belongs in fact to the class $C_i$ is high). In this way a human analyst would gain confidence regarding the reliability of the system.

Finally, in the last setting by using unsupervised learning, mining for patterns in historical alert logs and correlating them with assigned class labels, we can ensure the quality and consistency in alert labeling (i.e., similar alerts receive similar classification). This would require additional work from the human analyst, but in turn provide a higher assurance that the classification was correct. Pursuing this idea further, one can build an abstaining classifier using those mined patterns as the classification rules, and cascade it with any of the settings discussed above. In this way, some alerts can be reliably classified and discarded early on.

By means of these examples we showed that building a usable alert-classification system is more complex than merely designing a machine-learning technique supporting it. Giving the reader a preview, these examples are based on the techniques and the systems developed in the course of the dissertation: The first and the second example describe an alert-classification system called ALAC, introduced in Chapter 5, the third example describes ALAC+, an alert-classification system using abstaining classifiers introduced in Chapter 7, and the fourth example describes a two-stage alert-classification system with CLARAty (Chapter 8).

## 1.2 Why Learning Alert Classifiers Works and Why It is a Difficult Learning Problem

Our approach is based on the following two assumptions: (i) the analyst is able to classify alerts correctly, and (ii) it is possible to learn a classifier based on historical alerts. The first assumption is justified because the analyst must be an expert in intrusion detection to perform incident analysis and initiate appropriate responses. Nonetheless, the analyst may not be accurate in all cases, introducing an error in the classification. As for the second assumption, it has been shown [Jul03b, MCZH00] that a large fraction of alerts has a clear structure and that the learner should be able to generalize so that future "similar" alerts are classified correctly.

If these assumptions hold, this raises the question of why analysts do not write alert classification rules themselves or do not write them more frequently. An explanation of these issues can be based on the following facts:

**Analysts' knowledge is implicit:** Analysts find it hard to generalize, i.e., to formulate more general rules, based on individual alert classifications. For example, the analyst might be able to individually classify some alerts as false positives, but may not be able to write a general rule that characterizes the whole set of these alerts. Nonetheless we observed that in the MSSP environment analysts do write rules for removing the most frequently reoccurring alerts.

**Environments are dynamic:** In real-world environments the characteristics of alerts change, e.g., different alerts occur as new computers and services are installed or as certain worms or attacks gain and lose popularity. The classification of alerts may also change. As a result, rules need to be maintained and managed. This process is labor-intensive and error-prone.

This shows that it is possible to learn an alert classifier from examples; however, viewed as a machine-learning problem, alert classification poses several challenges.

First, the distribution of classes is often skewed, i.e., false positives are more frequent than true positives. Second, it is also common that the cost of misclassifying alerts is asymmetric i.e., misclassifying attacks for non-attacks is usually more costly than vice versa. Third, because the characteristics of the alert stream changes, the classifier should work in real-time and update its logic as new alerts become available. The learning technique should be efficient enough to perform in real-time and work incrementally, i.e., be able to modify its classifier as new data becomes available. Fourth, we require the machine-learning technique to use background knowledge, i.e., additional information such as network topology, alert database, alert context, etc., which is not contained in alerts, but allows us to build more accurate classifiers (e.g., classifiers using generalized concepts). In fact, research in machine learning has shown that the use of background knowledge frequently leads to more natural and concise rules [LD94]. However, the use of background knowledge increases the complexity of a learning task and only some machine-learning techniques support it.

So we are facing a highly challenging machine learning problem that requires great care to solve properly. We revisit these challenges in Section 5.3, in which we present a suitable learning technique.

## 1.3 Classifying Alerts: False Positives, True Positives or Other Classes?

So far we have introduced the problem of false positives in intrusion detection and have informally defined false positives alerts that mistakenly indicate security issues and require attention from the intrusion detection analyst. This would suggest that we are facing a binary classification problem. However, this may not necessarily be the case, and to better understand the nature of the problem let us look at alert classification in more detail.

Looking at alerts the intrusion-detection analyst tries to determine (i) the *root cause* of an alert, and (ii) its *impact* on the environment and based on this, (iii) the actions that need to be taken.

The root cause of an alert explains how it came into existence. More formally, according to Paradies and Busch [PB88] "a root cause is the most basic cause that can be reasonably identified and that management has control to fix". Clearly, determining the root cause plays an important role in analyzing alerts.

It might be tempting to try to classify all possible root causes, in order to determine the classes that can be used by an alert-classification system. However, building taxonomies of root causes and attacks is an extremely difficult and controversial task [How97, Jul03b, Krs98, LBMC94]. Moreover, we have no evidence that the actual analysts use any systematic classification.

On the other hand, analysts typically use ad-hoc classifications determining the intent of an activity that triggered it (intentional, inadvertent) and its type, for example:

**intentional/malicious:** Possible types of activities in this category include scanning (reconnaissance), unauthorized access (attempt), privilege escalation and malware infection, policy violation, DoS attack and suspicious activity.

**inadvertent/non-malicious:** Possible types of activities in this category include: network misconfiguration and normal activity (underspecified or intent-guessing signature).

The second important factor is determining the impact the activity has on the environment. To illustrate this with an example, an Internet-facing server may receive hundreds of automated attacks every day, coming from worm-infected computers around the world. In spite of being malicious, such alerts have only a marginal impact on the environment (unless of course the server is vulnerable to the attack being analyzed) and typically do not constitute security threats.

On the other hand, if the same worm-infected machines were performing a distributed denial of service (DoS) attack on the same server, the impact would be high, and such an incident would need to be reported. Continuing with the first example, if the worm infection attempt was launched against an internal web-server, this may be either as the result of an internal machine being infected with a worm or there could be a covert channel, allowing external machines to access internal machines. In both cases, the impact on the environment is high, and such incidents should be reported.

Finally, a scanning/reconnaissance incident is intentional and typically malicious unless performed by authorized sources. On the other hand, some machines are being notoriously scanned, and thus the scanning is not considered a security incident.

To summarize, whereas many ad-hoc classifications exist and can be used in classifying alerts, defining a taxonomy of alert root causes is difficult. On the other hand, it is a combination of the root cause and the impact on the environment that determines whether the alert is actionable and constitutes a security incident. From the analyst's standpoint this property is the most important one in classifying alerts. In the remainder of the thesis we will consider *actionable alerts* equivalent to true positives and *non-actionable alerts* equivalent to false positives.

Therefore a binary classification is useful and can frequently be applied in classifying alerts. Having said this, multi-class classification can be used if a predefined set of classes is given. We will discuss the topic of multi-class classification further in Section 5.4.1.

## 1.4 Thesis Statement and Contributions

This dissertation describes the work done to validate the following three-part thesis statement.

**Thesis Statement**

> *(1) Using machine learning, it is possible to train classifiers of IDS alerts in the form of human-readable classification rules by observing the human analyst.*
>
> *(2) Abstaining classifiers can significantly reduce the number of misclassified alerts with acceptable abstention rate and are useful in intrusion detection.*
>
> *(3) Combining supervised and unsupervised learning in a two-stage alert-processing system forms a robust framework for alert processing.*

The main ideas underlying this dissertation have been published in several articles [Pie04, PT05, Pie05, PV05], [Pie07]* and [Pie06]†. The novel contributions can be summarized as follows:

- We develop ALAC, a technique for adaptive learning of classification rules based on the intrusion detection analyst's feedback, which operates in two modes: a recommender mode and an agent mode. We analyze the requirements of such a system with regards to the input data and the machine-learning techniques.

  We discuss the problems of IDS evaluation and select two real-world and one synthetic dataset we use throughout this dissertation. In experiments with these datasets, we analyze the impact of background knowledge on the accuracy of alert classification and perform a general system evaluation, based on which we conclude that ALAC can be applied in real-world intrusion-detection systems.

- We analyze the concept of abstaining classifiers, i.e., classifiers that can say "I don't know" and compare them against normal binary classifiers in a cost-sensitive classification setup. We introduce three models under which abstaining binary classifiers can be evaluated: Cost-Based Model, Bounded-Abstention Model and Bounded-Improvement Model, and develop algorithms for constructing cost-optimal abstaining classifiers using ROC analysis. We also show how our methods can be applied to other graphical representations, such as precision-recall curves and DET curves.

*To appear.

†Accepted with minor revisions. To appear.

Finally, we perform an extensive evaluation of abstaining classifiers in these models on multiple benchmark datasets.

- We develop ALAC+, an alert-classification system using abstaining classifiers in the above models. Using a real-world and a synthetic dataset, we show that abstaining classifiers are particularly advantageous for alert classification.

- We propose and evaluate a semi-automated and automated framework for unsupervised learning and alert clustering based on CLARAty [Jul03b] on multiple real-world datasets. We propose novel alert cluster evaluation methods based on clustering precision and recall charts.

- We propose a two-stage alert-classification system, combining the unsupervised and supervised frameworks for robust handling of highly skewed datasets.

## 1.5 Overview

The thesis is organized as follows:

**Chapter 2** provides a brief introduction to the domains of intrusion detection and machine learning this dissertation builds upon. As this dissertation is targeted at both the security and the machine learning communities, this chapter summarizes the required background information for non-experts.

In the intrusion-detection part we give an overview of basic security concepts, the basic architecture of IDSs and their principles of operation based on two examples: Snort, an open source network-based IDS and CSSE our specialized host-based IDS. In the machine learning part, we summarize basic machine-learning techniques, with the main focus on predictive techniques, their evaluation and ROC analysis.

**Chapter 3** reviews the state-of-the-art intrusion-detection alert-management approaches, including alert classification and alert correlation. We also explore similarities to other, related domains, which gives inspiration for our work. This chapter provides the motivation and justification for this dissertation.

**Chapter 4** discusses the three different types of datasets used in the thesis and analyzes their properties and difficulties. This shows how alert classification differs from other machine-learning problems and justifies our approach.

**Chapter 5** presents and evaluated ALAC, an Adaptive Learning for Alert Classification system, its two modes of operation and the analysis of suitable machine-learning techniques. We also analyze the classification done by the human analysts and discuss multiclass vs. binary classification.

**Chapter 6** can be viewed both as an independent contribution to machine learning and also as a part of the alert-classification system. It uses a systematic approach to construct an abstaining binary classifier that minimizes the misclassification cost in a linear cost model. We present three different application-dependent evaluation models and show algorithms for building abstaining classifiers in each of these models.

**Chapter 7** presents and evaluates ALAC+, an alert-classification system using abstaining classifiers. We show how the use of abstaining classifiers introduced in the preceding chapter improves alert classification.

**Chapter 8** presents how the alert-classification system can be extended by using unsupervised learning (CLARAty). This extension is mostly motivated by real-world data, for which it is extremely important to reduce the redundancy of the data. We show how to integrate the unsupervised-learning framework into ALAC and how to configure it for alert classification. Finally we evaluate the system on a variety of datasets, including 14 million real alerts.

**Chapter 9** presents the conclusions, summarizes the thesis contributions, and discusses future work.

The thesis can be read sequentially, however, its organization is not strictly linear as shown in Figure 1.3.

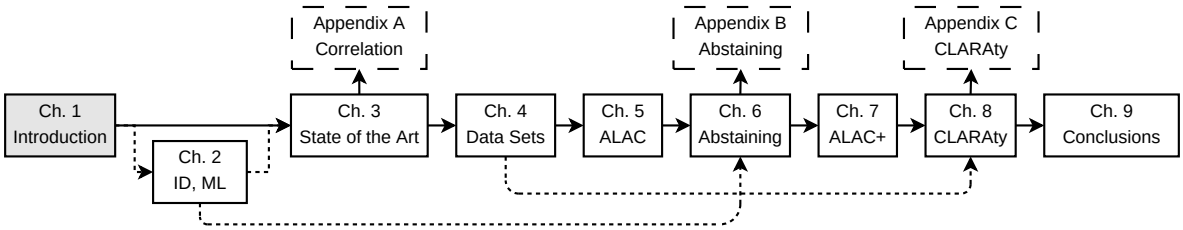



Figure 1.3: Thesis outline.

Readers interested in abstaining classifiers can proceed directly to Chapter 6, which is self-contained, and then continue to Chapter 7, the evaluation of abstaining classifiers with ALAC. We call this an "abstaining-classifiers track."

Similarly, readers interested in the evaluation results of CLARAty on real-world datasets and the new evaluation methods, can continue to read Chapter 4 and then jump directly to Chapter 8. We call this a "CLARAty track."

# Chapter 2

# Intrusion Detection and Machine-Learning Background

This dissertation builds upon machine-learning techniques and applies them to the security domain of intrusion detection. Hence, its targeted audience includes both the security community, for which the dissertation shows a practical solution to a real problem, as well as the machine-learning community, for which the dissertation presents an interesting application domain and reformulates the classification problem using abstaining classifiers.

In this chapter we introduce the basic concepts and definitions in both intrusion detection and machine learning. We provide a concise description of the domains upon which this dissertation builds. Readers familiar with intrusion detection or machine learning can skip the corresponding sections.

## 2.1 Intrusion Detection

In this section we will introduce the basic notions in computer security, define intrusion detection and describe the theory and practice of operation of intrusion detection systems used.

*Computer security* has always played an important role in electronic computing; however the explosive increase in the number of networked machines and the use of the Internet in the nineties combined with the growth of the number of unauthorized activities and increase in attacker's sophistication have made it one of the most important problems of computing.

In short, computer security deals with the protection of data and the computing resources and is commonly associated with the following three properties (commonly referred to as *C.I.A. triad*) [Com91]:

**Confidentiality:** prevention of any intentional or unintentional unauthorized disclosure of data. For example, an intruder learning about the customer credit card database or getting access to the proprietary source code is considered a breach of confidentiality. Note that typically such a breach is irreversible and cannot be confined easily.

The term confidentiality can also be understood in a broader context in which it also pertains to the non-delivery of services to unauthorized users, even though this would not compromise confidentiality in itself.

**Integrity:** prevention of intentional or unintentional unauthorized modification of data. For example, an intruder defacing the company's web server or modifying the bank's database content for personal gain is an attack against data integrity. Note that typically integrity can be restored, e.g., from other sources such as backup copies, although this process may be costly, time-consuming and not always complete.

**Availability:** prevention of the unauthorized withholding of computing resources. Examples of availability include the denial-of-service (DoS) attack, in which the attacker blocks the computing resources so that authorized users cannot use them, or physical equipment theft.

Based on this definition of the C.I.A triad, we can define intrusion as follows:

**Working definition 1 (Intrusion [HLMS90])** *Intrusion is any set of actions that attempt to compromise the confidentiality, integrity or availability of a computer resource.*

Note that this definition does not distinguish whether the action is intentional or unintentional and whether it is successful or not. This is different from the definition by Axelsson [Axe05], which requires that the intrusion be malicious.

Typically, the requirements for confidentiality, integrity and availability are not absolute, but are defined by a *security policy*. The security policy states which information is confidential, who is authorized to modify given information and what kind of use of computer systems is acceptable. Therefore we can reformulate the initial definition of intrusion as follow:

**Working definition 2 (Intrusion)** *Intrusion is a violation of a security policy.*

### 2.1.1 Intrusion Detection Systems

As stated by Mukherjee at al. [MHL94], the conventional approach to secure a computer or network is to build a "protective shield" around it. To this end, modern computer systems implement *identification* (identifying who a given user is), *authentication* (verifying whether the user is the one he or she claims to be) and *authorization* (verifying whether the user has sufficient access rights to perform a given operation). Similarly, there has been a number of both theoretical and practical approaches to building more secure systems, such as mandatory access control and compartmentalized security systems.

However, these approaches have the following problems:

- There is a tradeoff between security and usability: it is not possible to build a completely secure and still usable system.
- Current systems are built in an "open" mode, which is regarded to be useful for promoting user productivity.
- "Secure" systems are still vulnerable to attacks exploiting internal programming errors, such as buffer overflows, input-validation errors or race conditions.
- Even the most secure systems can still be vulnerable to insider attacks, i.e., authorized users misusing their privileges intentionally (e.g., for personal gain or some kind of revenge) or unintentionally (e.g., though social engineering).

As a result, an alternative approach, called *intrusion detection*, has been proposed [Den87]. The idea is to retrofit existing systems with security by detecting attacks, preferably in real time, and alerting a system security officer (SSO) or a security analyst.

Using either definition of an intrusion, we define an Intrusion Detection System (IDS) in the following way:

**Working definition 3 (Intrusion Detection System [Axe05])** *An automated system detecting and alarming of any situation where an intrusion has taken or is about to take place.*

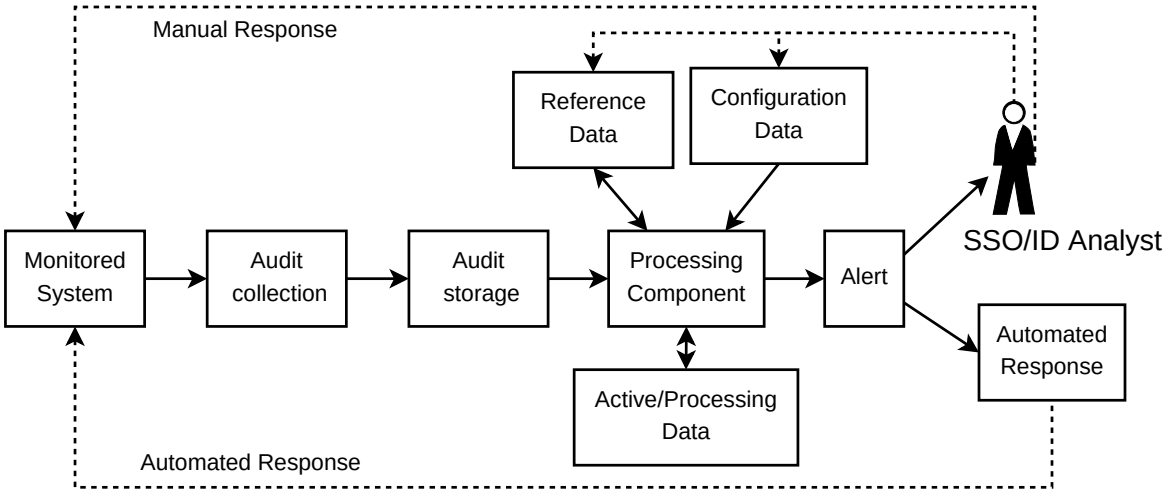



Figure 2.1: The general architecture of an IDS (based on [Axe05]).

The general architecture of an IDS is shown in Figure 2.1. The IDS observes the behavior of the system monitored and collects the audit information, which can subsequently be stored prior to processing. The processing component uses reference and configuration data (e.g., information about prior intrusions or the baseline of normal behavior for anomaly-based IDSs), processes the audit information and generates alerts. While doing this, the processing component may temporarily store data needed for further processing (e.g., context, session information). Finally, the output in the form of *alerts* is processed by either a human analyst, who can initiate a response to an intrusion, or an *automated response system*.

The latter approach is particularly appealing as only a fully automated response can prevent attacks in real time; however, it is potentially dangerous if implemented incorrectly. In fact, some automated intrusion response systems have been known to cause serious damage, including preventing legitimate users from using the systems or even being misused by an intruder to trick the system to perform a denial-of-service attack against itself. Therefore automated response systems have to be used with extreme caution.

In the next two sections we will discuss two main components of IDSs: the audit collection and the processing component.

### Audit Sources

The type of the audit sources distinguishes the two main groups of IDS: host-based IDSs and network-based IDSs.

*Host-based IDSs* (also referred to as HIDSs) use the audit information generated by the host as the source of data. The IDS itself can either use standard auditing tools (e.g., BSM [Sun95]), or specially instrumented operating systems (e.g., [Zam01]) or application platforms (e.g., [PV05]). The key advantage of these systems is that they may have access to all context information needed to determine whether an intrusion has taken place, which in turn makes them more accurate. On the other hand, host-based IDSs are tailored to the system they are protecting, which makes them more difficult to deploy and configure. Also the fact that some HIDSs are running on the system being protected may make the IDSs vulnerable to tampering by the intruder.

*Network-based IDSs* (also referred to as NIDSs) monitor network communications between networked machines, reconstruct the traffic from raw network packets and analyze them for signs of intrusions. Network-based IDSs are typically easier to deploy and manage (need only one sensor per network segment), although they have a more limited view of the situation and thus are also more prone to false positives. As shown by Ptacek and Newsham [PN98] if a NIDS has no additional information about the protected host, the malicious attacker can easily avoid detection by taking advantage of different handling by overlapping IP/TCP fragments by IDS and a target host. On more practical grounds, with faster networks, often working in switched environments or using end-to-end encryption, the efficacy of NIDSs have diminished.

Originally IDSs were implemented as host-based systems as they were a more natural choice for larger multi-user systems used then. With the evolution of computing towards networked workstations, the focus gradually shifted towards network-based IDSs. They were also considered easier to build and deploy in a networked environment. This gave rise to a number of both research and commercial IDSs currently in use. However, currently, due to the deployment of high speed switched networks and the use of end-to-end encryption, host-based IDSs seem to be regaining significance. In addition to a number of specialized host-based IDSs, there has been also a number of systems using a hybrid approach, therefore obtaining even better coverage.

### Processing Component

Detecting attacks requires the use of a model of operation or, in other words, what the IDS should look for. The following two models are currently in use: anomaly-based model and misuse-based model.

*The Anomaly-based model* is based on the premises that the behavior of the intruder is noticeably different from those of legitimate users. Anomaly-based IDSs establish a so-called normal model, against which the new activities are compared. Activities which are significantly different from this normal behavior are considered suspicious and reported. The advantage of this approach is that it can detect new types of intrusions, even the ones that have not been seen before. The disadvantage is that in many cases there is no single "normal profile" and anomaly-based systems tend to produce many false positives.

*The Misuse-based model* aims at detecting known intrusion patterns and activities. Such patterns are typically encoded into so-called *signatures*. Clearly, the disadvantage of such a system is that it detects only already known types of attacks it has a signature for. It is therefore natural to ask what is the point of detecting known attacks rather than protecting the system against them in the first place. The answer to these questions is the following:

**window of vulnerability:** Some systems are patched with a certain delay (i.e., when the software vendor releases a proper patch) and some cannot be patched at all. IDSs detect attacks during this *window of vulnerability* [LWS02].

**detecting failed attacks:** Even if the system is not vulnerable to an attack, information that someone tried to exploit a certain vulnerability is important in itself and often constitutes a policy violation.

**policy violations:** Misuse-based IDSs are an excellent tool for detecting policy violations (e.g., scans, peer-to-peer usage, running certain services), which are not vulnerabilities and hence cannot be "patched".

**additional layer of protection:** IDSs provide an additional layer of protection through redundancy (signatures independent from patching the original software) and diversity. Many incidents happen not because there was no patch available, but because somebody forgot to apply it to all the machines.

**generalized and intent-guessing signatures:** Misuse-based systems also employ *generalized signatures*, detecting also unknown variants of known attacks. Similarly *intent-guessing signatures* aim at detecting certain types of suspicious behavior, typically associated with malicious actions. While both are also a source of false positives, they also increase the usability of a misuse-based IDS.

This shows why misuse-based systems are useful in protecting against attacks, policy violations and attack attempts.

Currently most of the IDSs deployed are misuse-based IDSs with a mixture of misuse signatures, generalized and intent-guessing signatures and also signatures detecting policy violations. This high number of signatures allows the users to customize the IDS depending on their needs and the particular environment. Anomaly-based IDSs are more frequently used by the research community, due to their higher false-positive rates and the difficulty characterizing the "normal behavior" in real environments.

### 2.1.2 Two examples of IDSs

In this section we will present two IDSs: Snort, one of the most popular open-source NIDSs, frequently used by both researchers and practitioners, and CSSE, a specialized HIDS defending against injection attacks in applications.

While the main focus of this dissertation is the reduction of false positives of NIDS using machine-learning techniques, we developed CSSE [PV05] as a proof-of-concept new-generation host-based IDS with extremely low false-positive rates. While CSSE is not the main result of this thesis it is related to it in to ways: First, by providing an example of a HIDS, the reader gets a better understanding of the domain of intrusion detection. Second, and more importantly, it shows how by addressing the root cause of an attack, it is possible to develop an IDS with extremely low false-positive rates. We believe that this represents a trend in intrusion detection to build specialized but highly accurate sensors.

#### Snort—an open-source IDS

Snort [Roe05] is a network-based IDS, using raw network packets received through a pcap interface [JLM03] as its audit source. As with other NIDSs, when used in a switched envi-

ronment, it needs to be connected to the mirrored port.

The principle of operation of Snort is very simple: it applies an ordered list of rules (called *signature*) to each packet it receives. The first rule that fires generates an alert. Signatures in Snort are a conjunction of conditions pertaining to fields in IP/TCP/UDP packets as well as the packet payload. For payload matching predicates Snort provides a set of functions yielding a powerful expression language, offering regular expressions and stateful analysis (i.e., predicates can refer to previously received and analyzed packets).

However, if Snort was implemented this way it would only be able to reliably detect attacks at the lowest, network layer (IP protocol), whereas most of the current attacks happen at higher layers: transport (TCP/UDP) or application layers (e.g., HTTP, RPC). To address this, Snort uses so-called preprocessors which perform stream reassembly and normalization of higher-level protocols. The result of this preprocessing is assembled into a special virtual packet to which the list of signatures is applied. Although conceptually simple, the real complexity of such processing should not be underestimated. To illustrate this with an example: detection of an attack targetting a web server, requires the preprocessors to perform the following operations:

1. IP-level traffic normalization and defragmentation.

2. TCP state machine emulation and stream reassembly, including retransmissions, handling of overlapping fragments, etc.

3. HTTP-level normalization, defragmentation and unicode decoding.

Clearly not all of the alerts can be generated statelessly by applying a list of rules to a packet. Those situations, in which it is necessary to keep some state are also handled by specialized plugins (e.g., portscan detection, arp-spoof detection).

```
   Input: a real-time sequence of packets
   Result: a sequence of alerts
 1 parse the configuration files and rulesets
 2 while (P=receivePacket()) != NULL do
 3 |   for R ∈ getPreProcessorList() do
   |   |   /* may create a virtual packet                          */
 4 |   |   (P,A) ← applyPreprocessor(R,P);
 5 |   |   if A != NULL then
   |   |   |   /* report alerts generated by preprocessors        */
 6 |   |   |   reportAlert(A);
 7 |   |   end
 8 |   end
 9 |   for S ∈ getOrderedSignatureList() do
10 |   |   if (A=signatureMatch(S,P)) != NULL then
11 |   |   |   reportAlert(A);
   |   |   |   /* report only the first match                    */
12 |   |   |   break;
13 |   |   end
14 |   end
15 end
```

**Algorithm 1**: Simplified Snort detection algorithm.

To illustrate this, we show two representative signatures (selected from more than 4000 signatures currently available in Snort). The first signature is a typical misuse signature, detecting an exploitation attempt in the AWStats tool:

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS
 (msg:"WEB-CGI awstats.pl configdir command execution attempt";
flow:to_server,established; uricontent:"/awstats.pl?"; nocase;
uricontent:"configdir="; nocase;
pcre:"/awstats.pl?[^\r\n]*configdir=\x7C/Ui";
reference:bugtraq,12298; reference:cve,2005-0116; reference:nessus,16189;
classtype:attempted-user; sid:3813; rev:2;)
```

Interpreting the signature, it is applied to TCP traffic originating from the external network and sent to HTTP servers running on typical web server ports (all these are configuration variables). The signature is applied to the flow sent to the server and containing the string `/awstats.pl?` and `configdir=`. The actual signature is a regular expression containing the first string, not followed by new line characters and followed by the second ending with a pipe "`|`" (hex code 7c). It is actually this very last character that exploits the vulnerability, allowing for arbitrary command execution (e.g., `/awstats.pl?configdir=|/bin/ls|`), however the entire signature is important to put this dangerous pipe character into a proper context. The signature refers to a vulnerability known as #12298 in the Bugtraq [Sec04] database, CVE-2005-0116 in the CVE [MIT04] dictionary, and with a corresponding Nessus [Der03] detection plugin #16189. Note that Snort cannot determine at this stage whether the attack has succeeded or even if the user uses awstats at all! For example, if Snort's website was protected by Snort itself, any user accessing a URL `http://www.snort.org/awstats?configdir=|/bin/ls|` would trigger an alert.

Whether this is desirable is another question: On one hand, the administrator of the website may be interested in finding out whether somebody is trying to find a hole in it. On the other hand, most of Internet facing websites receive several such requests every day.

Another signature is of a different sort: it does not detect an exploitation of a known vulnerability, but rather detects activities that are probably suspicious. It is therefore called an intent-guessing signature:

```
alert tcp $EXTERNAL_NET any -> $SQL_SERVERS 1433
(msg:"MS-SQL sp_adduser - database user creation"; flow:to_server,established;
content:"s|00|p|00|_|00|a|00|d|00|d|00|u|00|s|00|e|00|r|00|"; nocase;
classtype:attempted-user; sid:685; rev:5;)
```

The rule fires if an external machine attempts to execute a stored procedure `sp_adduser` in an MSSQL database server, which creates a database user. While there is nothing wrong with calling this stored procedure in itself, it has been observed that many attacks launched against SQL servers (either directly or through SQL injections) use these stored procedures to circumvent database security mechanisms. However, such intent guessing signatures are prone to false positives: every time an authorized database administrator uses this procedure, it will trigger a Snort alert.

### CSSE—a specialized HIDS

In recent years we have seen a steady increase in the importance of application-level security vulnerabilities, i.e., vulnerabilities affecting applications rather than the operating system or

middleware of computer systems. Among application-level vulnerabilities, the class of *input validation* vulnerabilities is the most prominent one [NIS04] and deserves particular attention. However, other than specific exploits (like in the example of Snort signatures above) attacks exploiting these vulnerabilities are typically difficult to detect by network-based IDSs.

Input validation vulnerabilities (e.g., buffer overflow, integer overflow, injection vulnerabilities) are flaws resulting from implicit assumptions made by the application developer about the application input. More specifically, injection vulnerabilities, addressed by our HIDS, exist when the assumptions concerning the presence of syntactic content in the application input can be invalidated using maliciously crafted input to effect a change of application behavior in a way that is beneficial to the attacker.

In injection attacks the attacker provides maliciously crafted input carrying syntactic content that changes the semantics of an expression in the application. The results are application-dependent, but typically lead to information leakage, privilege escalation or execution of arbitrary commands.

Context-Sensitive String Evaluation [PV05] (CSSE) is a host-based intrusion detection and prevention system, for injection attacks. It uses an instrumented execution environment (such as PHP or Java Virtual Machine) and therefore has access to all necessary context required to detect and, more importantly, prevent injection attacks. The prevention is possible thanks to extremely low false-positive rates.

**Addressing the Root Cause** Analyzing injection vulnerabilities, we see that a common property is the use of textual representations of output expressions constructed from user-provided input. *Textual representations* are representations in a human-readable text form. *Output expressions* are expressions that are handled by an external component (e.g., database server, shell interpreter).

User input is typically used in the data parts of output expressions, as opposed to developer-provided constants, which are also used in the control parts. Therefore, user input should not carry syntactic content. In an injection attack, specially crafted user input influences the syntax, resulting in a change of the semantics of the output expression. We will refer to this process as *mixing of control and data channels*.

Injection vulnerabilities are not caused by the use of the textual representation itself, but by the *way* the representation is constructed. Typically user-originated variables are serialized into a textual representation using string operations (string concatenation or string interpolation, as in our example). This process is intuitively appealing, but ultimately ad hoc: variables lose their type information and their serialization is done irrespectively of the output expression. This enables the mixing of data and control channels in the application, leading to injection vulnerabilities.

We thus consider the *ad-hoc serialization of user input* for creating the textual representation of output expressions as the root cause of injection attacks. Ad-hoc serialization of user input (or variables in general) can lead to undesired mixing of channels, but has also some desirable properties. The most important one is that it is intuitively appealing and, consequently, more easily written and understood by the application developers. Second, for many types of expressions (e.g., XPath, shell command) ad-hoc serialization of user input using string operations is the only way of creating the textual representation.

Considering this, a defense against injection attacks should enable the application developer to use the textual representation in a safe manner.

**CSSE Operation** CSSE addresses the root cause of injection vulnerabilities by enforcing strict channel separation, while still allowing the convenient use of ad-hoc serialization for creating output expressions. A CSSE-enabled platform ensures that these expressions are resistant to injection attacks by automatically applying the appropriate checks on the user-provided parts of the expressions. CSSE achieves this by instrumenting the platform so that it is able to: (i) distinguish between the user- and developer-provided parts of the output expressions, and (ii) determine the appropriate checks to be performed on the user-provided parts.

The first condition is achieved through a tracking system that adds metadata to all string fragments in an application in order to keep track of the fragments' origin. The underlying assumption is that string fragments originating from the developer are trusted, while those originating from user-provided input are not and therefore cannot carry syntactic content. The assignment of the metadata is performed without interaction of the application developer or modification of the application source code. Instead, it is achieved through the instrumentation of the input vectors (e.g., network, file) of the CSSE-enabled platform. CSSE further instruments the string operations to preserve and update the metadata assigned to their operands. As a result, the metadata allows us to distinguish between the developer-provided (trusted) and user-provided (untrusted) parts of the output expressions at any stage in their creation. Figure 2.2 illustrates the dataflow of the vulnerable application executed in a CSSE-enabled platform.

The second condition is achieved by deferring the necessary checks to a very late stage, namely to the moment when the application calls the API function to pass the output expression on to the handling component (output vector). CSSE intercepts all API calls related to output vectors, and derives the type of output vector (e.g., MySQL, shell) from the actual function called (e.g., `mysql_query()`, `exec()`). This allows CSSE to apply the checks appropriate for this particular output vector.

At this point, CSSE knows the complete context. The first part of the context is provided by the metadata, which describes the fragments of the output expression that require checking. The second part of the context is provided by examining the intercepted call to the API function, which determines which checks will be executed. CSSE then uses this context information to check the unsafe fragments for syntactic content. Depending on the mode CSSE is used in, it can raise an alert (intrusion detection), prevent the execution of the dangerous content or escape it (both intrusion detection and prevention).

**Implementation and Discussion** Currently CSSE is available as a research-prototype IDS for the PHP platform [PHP04a]. We have evaluated the prototype with an old known to be vulnerable version of phpBB [php04b], a popular web application with known security vulnerabilities and verified that it detected and prevented all known SQL-injection attacks. Because CSSE was designed without knowledge of these specific attacks, we expect that with a complete implementation it would have a similarly good performance also with other applications.

Moreover CSSE incurs a reasonable runtime and memory overhead. In our experiments we observed that the run-time overhead rarely exceeded 10%, whereas the memory consumption increased by less than 2%. This shows that CSSE is a good IDS for detecting injection attacks.

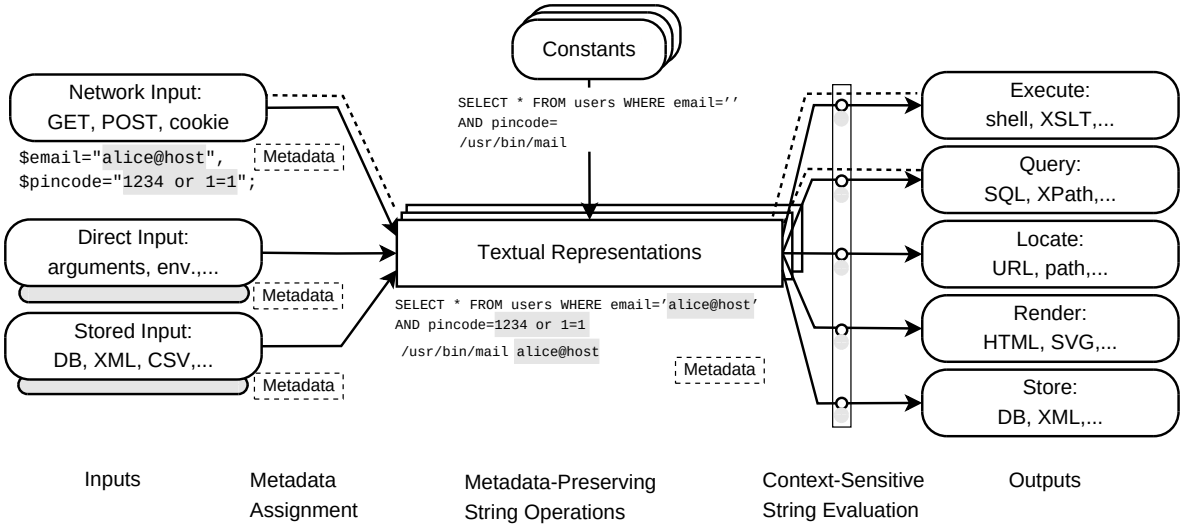



Figure 2.2: Using CSSE to preserve the metadata of string representations and to allow late string evaluation. Shaded areas represent string fragments originating from the user.

### 2.1.3 Conclusions

In this section we gave a short introduction to computer security in general and intrusion detection in particular. We introduced the notions of intrusions and intrusion detection systems, showed the general IDS architecture and their classification.

We also presented two IDS examples in more detail: Snort, an open-source network-based IDS, used in this dissertation, and CSSE, a highly specialized and robust host-based IDS detecting injection attacks. CSSE is not directly used in this thesis but gives an example of a specialized IDS with extremely low false-positive rates and shows the trends in the IDS evolution.

## 2.2 Machine Learning

*Machine Learning* (ML) is concerned with building systems that automatically improve their performance with experience. *Classification*, studied in this thesis is one of the standard tasks in machine learning.

### 2.2.1 Classification

A *classifier* is a function that assigns a class label from a finite set of labels to an instance. For example, given information about a news article, the classifier might specify the topic it deals with; given an image, a classifier might decide which letter of an alphabet it depicts. Classifiers can be constructed automatically or built by human experts. Depending on their structure, classifiers built automatically can be *interpretable* by humans (e.g., classification rules, decision trees) or not (e.g., support vector machines, neural networks). The latter are often referred to as *black-box classifiers*. In this thesis we focus on the first type, i.e., automatically constructed interpretable classifiers.

One way to build classifiers automatically is to use supervised machine-learning techniques, which are typically applied in two stages: learning stage and testing stage. In the learning stage, the classifier is given a set of instances together with correct class labels, which allows it

to modify its internal structure. In the testing stage, the classifier is presented with unlabeled, previously unseen instances, for which it predicts class labels. The testing phase allows the user to evaluate the performance of a classifier.

Many classification tasks are binary, in which the classifier assigns one of two possible classes. In most of these cases the classifier tests an instance with respect to a particular property (e.g., an object is a plane; an e-mail is a legitimate email or not; a security event is an intrusion). Without loss of generality, we will denote the labels assigned by a *binary classifier* as positive, "+" (assigned when the property at question exists) and as negative, "−" (assigned otherwise). Binary classification might seem a bit restrictive, but it is very well understood and has some properties that make it particularly appealing.

More formally, we define terms classifier, binary classifier, ranker and a calibrated binary classifier used in the remainder of the thesis.

**Definition 4 (Classifier)** *A classifier $\mathcal{C}$ as a function $\mathcal{C}(i) : I \mapsto C$, where $I$ is an instance space and $C = \{C_1, C_2, \ldots, C_n\}$ is a class space.*

**Definition 5 (Binary classifier)** *A binary classifier $\mathcal{C}_b$ is a classifier mapping an instance space $I$ to a binary class space: $\mathcal{C}_b(i) : I \mapsto \{$ "+", "−" $\}$.*

**Definition 6 (Ranker)** *A Ranker (a scoring classifiers) is an extension of a binary classifier that assigns scores to instances $\mathcal{R} : I \mapsto \mathbb{R}$. The value of the score denotes the likelihood that the instance is "+" and can be used to sort instances from the most likely to the least likely positive (ranking). The scores do not necessarily have to be calibrated probabilities.*

A ranker $\mathcal{R}$ can be converted to a binary classifier $\mathcal{C}_\tau$ as follows: $\forall i : \mathcal{C}_\tau(i) =$ "+" $\iff \mathcal{R}(i) > \tau$. Variable $\tau$ in $\mathcal{C}_\tau$ denotes the parameter (in this case the threshold) used to construct the classifier.

**Definition 7 (Calibrated binary classifier)** *A calibrated binary classifier is a classifier that outputs probabilities that an instance belongs to the positive class $p(+ \mid i)$.*

Calibrated classifiers have the advantage that the optimal decision point using the maximum a posteriori rule, however, learning such classifiers is not trivial [CG04, ZE01].

**Learning Methods** Machine-learning techniques are techniques that learn with experience. This very broad and vague definition aims at covering most of the tasks machine learning solves. More precisely, as shown by Mitchel [Mit97], there are three important aspects of the training experience that greatly influence the choice of the learning method.

First, the question whether learner receives a *direct or indirect feedback* regarding the choices made. For example, a system classifying alerts into true and false positives may be told exactly what the correct classification is. Conversely, a checkers playing program, may only be given a set of move sequences and the results of the games played, which is a more difficult learning task. Second, the question is to what degree the learner control the sequence of training examples it is given. Third, the question is how well the distribution of training examples represents the distribution of examples the system is measured against.

This dissertation deals with learning systems that receive direct feedback to the choices made. In such a setup, central to the learning process is the notion of *labeled instances*, that is pairs $(i, c_i)$, where $i \in I$ is an instance and $c_i \in C$ is a corresponding class label. Depending

on the availability of labeled instances we distinguish four main types of learning: supervised learning, semi-supervised learning, active learning and unsupervised learning.

In a typical, *supervised learning*, a machine-learning method $\mathcal{L}$ uses a set of example-class label pairs to construct a classifier: $\mathcal{L} : \{(i_j, C_{i_j})\} \mapsto \mathcal{C}$. Such a classifier $\mathcal{C}$ is subsequently used for classifying new instances. This type of learning is the most commonly used and will be further explored in this section.

In a *semi-supervised learning*, a machine-learning method $\mathcal{L}$ uses two sets for learning: a labeled set, a set of example-class label pairs and an unlabeled set: $\mathcal{L} : (\{(i_j, C_{i_j})\}, \{i_k\}) \mapsto \mathcal{C}$. Ideally, the performance of such a classifier is better than a corresponding classifier built using only labeled instances and obviously worse than the one built using all instances, assuming that their class labels are known.

In *active learning* [LC94], a machine learning method $\mathcal{L}$ is given a set of unlabeled instances $I$ and an oracle $\mathcal{O}$ . The oracle $\mathcal{O}$ tells the correct class label given an instance $i$. The goal is to label a complete set of instances $\{i\}$ with the smallest number of questions to the oracle $\mathcal{O}$. Typically such a setup is used when labeling of instances is expensive (e.g., involves human decision). In general the learning process proceeds as follows: Given an initial classifier $\mathcal{C}_i$ the learner $\mathcal{L}$ selects some unlabeled instances for which the classification is less confident. These instances are subsequently presented to the oracle $\mathcal{O}$ to be labeled and added to the labeled set. The labeled set is subsequently used to iteratively build an improved classifier $\mathcal{C}_{i+1}$. The process is repeated until the classification performance is satisfactory or the number of queries to the oracle has been exhausted. Finally the remaining unlabeled instances are classified using a classifier $\mathcal{C}_n$.

Finally, in *unsupervised learning* a machine learning method uses only an unlabeled set (or sequence) of instances $I$ with the goal to detect interesting patterns in the data. Examples of unsupervised learning include clustering, association rule mining, outlier detection or time series learning.

This dissertation deals with a spectrum of machine-learning techniques: adaptive alert classification (Chapter 5) is a supervised learning although it bears certain resemblance with active learning. Abstaining classifiers introduced in Chapter 6 introduce the abstention, which is implicitly used as a part of active learning. In fact, abstaining classifiers can be used to construct active learning techniques. Finally, Chapter 8 leverages unsupervised learning to build better alert classifiers.

**Common Assumptions** Typically machine learning methods make the following two assumptions, which simplify machine-learning techniques, but makes it also more difficult to apply in many real-world applications:

**attribute-value (propositional) representation:** typically instances are represented as tuples over the $n$-dimensional attribute space $dom(A_1) \times \ldots \times dom(A_n)$, where $dom(A_i)$ is the domain of attribute $A_i$. Commonly supported attributes are *categorical attributes* (discrete and unordered, e.g., company names: "IBM", "Microsoft", "Dell"), *numerical attributes* (e.g., counters, size attributes), *free-text attributes* (arbitrary and unforeseeable text values). Additional extensions allow to handle set-valued attributes [Coh96] or bags of words (frequently used in text classification).

This representation is powerful although insufficient for some purposes (e.g., representing the structure of molecules, linked web-pages). Such applications can be handled

either by *feature construction* (supported by the domain knowledge, in which non-propositional instances are augmented with a fixed number of features) or moving to more expressive representations (e.g., Prolog predicates, multi-relational representation) and learning systems (e.g., inductive logic programming [LD94], link mining [Get03]).

For example, in inductive logic programming, given background knowledge $\mathcal{B}$ and a set of examples $\mathcal{E}$ the goal is to find the simplest and consistent hypothesis $\mathcal{H}$ that $\mathcal{B} \wedge \mathcal{H} \models \mathcal{E}$. As both the background knowledge and the hypothesis use expressive first-order framework, it is much more expressive than attribute-value representations used by conventional machine-learning techniques. On the other hand, more expressive representations lead to a much larger hypothesis search space and learning times.

**independent and identically distributed (i.i.d.):** for statistical inference it is typically assumed that instances are independent of each other and identically distributed. This means that each instance $i \in I$ is drawn according to some unknown, but fixed, probability distribution and the class labels given for each instance are an unknown but fixed function of instance attributes.

This simplifies the underlying mathematics of many statistical methods, however is unrealistic in many practical applications. The possible approaches of overcoming this assumption is to use the background knowledge to encode temporal dependencies or use approaches like sequence mining (e.g., frequent episodes [MT96]) handling non-identically distributed instances explicitly.

### 2.2.2 Basic Techniques

In this section we will present representative examples of machine-learning techniques used for classification. We will come back to and further expand the discussion of interesting techniques in Chapter 5 in which we look at techniques suitable for our classification problem. The section has the goal to give the reader an overview on the techniques most commonly used.

**Predictive Rules** Predictive rules are patterns of the form `IF <conjunction of conditions> THEN <conclusion>`. The individual conditions in the conjunction are tests concerning the values of individual attributes. For predictive rules, the conclusion gives a prediction for the value of the target class (variable).

Predictive rules can be ordered or unordered. Unordered rules are considered independently and several of them may apply to a new instance that we need to classify. A conflict resolution mechanism is needed if two rules that recommend different classes apply to the same example. Such a resolution is typically based on rule's coverage or precision. Ordered rules form a so-called decision list. Rules in that list are considered from top to bottom of the list. The first rule that applies to a given example is used to predict its class value. There is typically a default rule whose prediction is taken if no other rule applies.

```
(physician-fee-freeze = y) and (synfuels-corporation-cutback = n) =>
    Class=republican (138.0/3.0)
(physician-fee-freeze = y) and (export-administration-act-south-africa = y) =>
    Class=republican (19.0/3.0)
(physician-fee-freeze = y) and (adoption-of-the-budget-resolution = n) =>
```

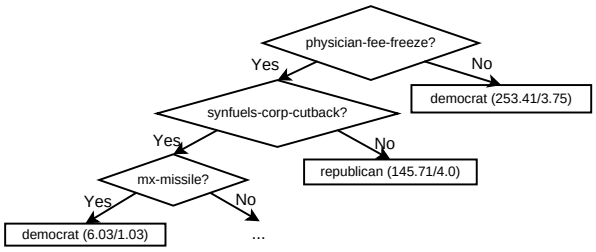



Figure 2.3: A sample decision tree.

```
    Class=republican (16.0/4.0)
 => Class=democrat (262.0/5.0)
```

Rule learners are programs for learning predictive rules. One of the first and best known algorithms is CN2 [CN89]. The algorithm performs a covering approach over the examples. It repeatedly finds the *best rule* according to some criterion (e.g., Laplace, m-estimate or weighted relative accuracy). The best rule covers some of the positive classification examples and none of the negative examples. The rule is then added to the hypothesis and examples covered by it are removed from set. The process finishes when there are no more examples to cover. The procedure finding the best rule uses beam search and a heuristic algorithm.

More complex approaches, like RIPPER [Coh95] (further discussed in Section 5.4) apply pruning while building the rule set to prevent overfitting and achieve good generalization capabilities.

**Decision Trees** Decision tree learning is one of the most widely used and practical methods for inductive inference for learning discrete valued target functions [Mit97]. Decision tree learning is robust to noisy data and is capable of learning disjunctive expressions. Learned trees can be represented as sets of human readable if-then rules.

Decision trees (an example shown in Fig. 2.3) classify instances by sorting them down from the root to some leaf node, which provides classification of an instance. Each node in the tree tests some attribute of the instance and each branch descending from the node specifies all the possible values of this attribute. Leafs contain the class value of the function the decision tree is approximating. In other words, a decision tree represents disjunctions of conjunctions of constraints on attribute values of instances, where each path from root to leaf corresponds to one value of the attribute sets.

There are several decision tree learning algorithms, among the most popular are ID3 and C4.5 [Qui86, Qui93]. These algorithms employ top-down greedy search through the space of possible decision trees evaluating how well attributes classify examples. The quantitative measure of attribute classification properties is often *information gain*. The inductive bias of decision tree learning is that shorter trees are preferred over longer trees and attributes that give high information gain are placed close to the root.

Decision trees can be easily converted to prediction rules, by rewriting the tree into a disjunctive set of rules, however such rules are known to be less comprehensible than the ones learned by the rule learners. Similarly, whereas simple trees are easily interpretable, larger trees are not. Therefore decision rules are preferred to trees when the interpretability of a classifier is required.

**Bayesian Networks** Bayesian networks are forms of specific graphical models, in the form of a directed acyclic graph. The vertices in the graph represent variables, while edges in the graph represent dependencies between variables. The variables can represent observed measurements, latent variables hypothesis or parameters. Bayesian networks represent joint probability distributions for all the nodes, assuming that the probabilities of each node are dependent only on their parents $Pr(X \mid parents(X))$. The problem of learning of the structure of Bayesian networks is NP complete, but many algorithms exist (e.g., [HGC95]).

A special case of Bayesian networks is a *Naive Bayes* classifier, which uses strong independence assumptions about variables, which often have no bearing in reality. In spite of this, Naive Bayes classifiers have been shown to perform surprisingly well in many domains in which the assumptions are clearly violated (a theoretical justification for this can be found in [DP97]). Hence it is typically used as a baseline classifier to compare against more complex methods.

**Support Vector Machines** Support Vector Machines (SVMs) [Vap95] use a so-called "kernel-trick" to apply linear classification methods to non-linear classification problems. SVMs try to separate two classes of data points in a multi-dimensional space using a maximum-margin hyperplane, i.e., a hyperplane that has a maximum distance to the closest data point from both classes.

The problem of learning SVMs is theoretically well-founded and well understood. They are particularly useful for application domains with a high number of dimensions, such as text classification, image recognition, bioinformatics or medical applications. The disadvantage of these methods is that the models are not understandable by humans.

**Artificial Neural Networks** Artificial Neural Networks (or Neural Networks for short) have been originally inspired by the examination of the central nervous system and neurons, which constitute its information processing element. A neural network is constructed as a group of interconnected artificial neurons (non-linear processing elements). In a supervised learning setting, the goal is to minimize the average error between the network's output and the target value.

Neural networks provide a general, practical method for learning real-valued, discrete-valued and vector-valued functions from examples. Neural networks can be applied to the same classes of problems as decision tree learning and have been shown to achieve comparable accuracy.

The main disadvantages of artificial neural networks is that their structure has to be suitable for the given application (e.g., it is difficult to design a neural network where number and type of inputs are unknown) and requires significant experience to train correctly. Also, neural networks exhibit long learning times and their structure is not interpretable by humans.

**Instance-Based Learning** Contrary to the previous methods, instance-based learning (IBL) [AKA91] does not generate the general description of the target function when training examples are provided, but simply stores the training examples. Conversely, IBL generates a number of local approximations, but only when it needs to classify a particular instance. Examples of instance-based learning include $k$-Nearest Neighbor algorithm and its derivatives: distance-weighted $k$-NN, locally weighted regression. In all these cases instances are represented as points on a $n$-dimensional Euclidean space and the target function is evaluated as a

Table 2.1: The confusion and cost matrices for binary classification. The columns (C) represent classes assigned by the classifier; the rows (A) represent actual classes.

| A \ C | + | − | |
|---|---|---|---|
| + | *TP* | *FN* | *P* |
| − | *FP* | *TN* | *N* |

(a) Confusion matrix $C$

| A \ C | + | − |
|---|---|---|
| + | 0 | $c_{12}$ |
| − | $c_{21}$ | 0 |

(b) Cost matrix *Co*

combination of selected (or all) data points. For more complex (symbolic) instance representations, case-based reasoning (CBR) techniques are used. CBR tries to solve a query based on the retrieval and combination of similar cases, however, the process of combining those instances can be very different and rely on knowledge-based reasoning rather than statistical methods, like in $k$-NN.

The main advantage of this approach is that it is fully incremental and can learn even very complex target functions. The disadvantage is that instance-based learning does not generate human-interpretable global representations of a target function, which makes it more difficult to validate the correctness of the prediction. Moreover, although the 'learning' is fast, the classification is not, and depends on the number of instances stored.

### 2.2.3 Evaluating Classifiers

The performance of a classifier using $k$ class labels is described by means of a $k \times k$ dimensional matrix $C$, known as the *confusion matrix*. Rows in $C$ represent the actual class labels and columns represent the class labels assigned by the classifier. Element $C_{i,j}$ represents the number of instances of class $i$ classified as class $j$ by the system. For a binary classifier $\mathcal{C}_b$, the elements of the matrix are called true positives (*TP*), false negatives (*FN*), false positives (*FP*) and true negatives (*TN*) as shown in Table 2.2a. The sum of *TP* and *FN* is equal to the number of positive instances ($P$). Similarly, the number of negative instances ($N$) equals $FP + TN$.

Many real-world classification problems are asymmetric, which means that some misclassifications are more "costly" that the other ones. This can be modeled in a cost-sensitive setup by introducing a so called *cost matrix Co* with identical meaning of rows and columns as the confusion matrix. The value of $Co_{i,j}$ represents the cost of assigning class $j$ to an instance belonging to class $i$. Most often the cost of correct classification is zero, i.e., $Co_{i,i} = 0$. In such cases, for binary classifications (Table 2.2b), there are only two values in the matrix: $c_{21}$ (cost of misclassifying a false alert as a real one) and $c_{12}$ (cost of misclassifying a true alert as a false one). In fact, such a cost matrix has only one degree of freedom, the ratio between these two values, called *cost ratio* (*CR*). As for intrusion detection, the value of *CR* is smaller than one, it is more intuitive to use its inverse, the *inverse cost ratio* (*ICR*) defined as:

$$CR = \frac{c_{21}}{c_{12}} \qquad ICR = \frac{c_{12}}{c_{21}} \ . \tag{2.1}$$

Classifiers in a cost-sensitive setup can be characterized by the cost *rc*, a cost-weighted sum of misclassifications divided by the number of classified instances:

$$rc = \frac{\sum_{i=1}^{k} \sum_{j=1}^{k} (C_{i,j} \cdot Co_{i,j})}{\sum_{i=1}^{k} \sum_{j=1}^{k} C_{i,j}} \ , \tag{2.2}$$

which in the binary case simplifies to:

$$rc = \frac{FN \cdot c_{12} + FP \cdot c_{21}}{TP + FN + FP + TN} = \frac{FN \cdot c_{12} + FP \cdot c_{21}}{N + P} \quad . \tag{2.3}$$

In this cost-sensitive setup, the *optimal classifier* (selected from a certain set of classifiers, e.g., a set of binary classifiers $\mathcal{C}_\tau$ derived from a single ranker $\mathcal{R}$) is the one that minimizes the misclassification cost $rc$. Note that our optimal classifier has to be selected from a pool of classifiers, otherwise a truly optimal classifier is the one that does not make any misclassifications, i.e., with $rc = 0$. If none of the classifiers are perfect, the selection of the optimal classifier $\mathcal{C}$ depends on both the confusion and the cost matrices.

In machine learning the performance of a classifier is typically evaluated on a test set, which is independent of the training set. In cases where a learning method produces a set of classifiers out of which we want to select the optimal one, typically three independent sets are used: a training set, for building a classifier; a testing set, for selecting the optimal model; and a validation set, for the evaluation of the classifier. Because the testing set and the validation set are independent of each other and the validation set has not been used in setting the parameters of the classifier, it can be used for an unbiased estimation of classifier's performance.

### 2.2.4 ROC Analysis

ROC (Receiver Operating Characteristic) analysis offers a flexible and robust framework for evaluating classifier performance with varying class distributions or misclassification costs [Faw03]. The most typical applications of ROC analysis include: model selection (i.e., choosing the optimal classifier or the optimal operation point of a classifier, given misclassification costs and class distribution) and model evaluation (e.g., calculating area under ROC curves).

In this dissertation we use ROC analysis in two ways: First, in Chapters 5, 7 and 8 we use it to estimate classifiers' performance and compare them. Second, in Chapter 6 we extend ROC analysis and develop a method for selecting abstaining classifiers based on ROC curves.

A ROC plane ($fp \times tp$) has axes ranging from 0 to 1 and labeled *false-positive rate* ($fp = FP/(FP + TN) = FP/N$) and *true-positive rate* ($tp = TP/(TP + FN) = TP/P$), as shown in Figure 2.4. Evaluating a binary classifier $\mathcal{C}_\tau$ on a dataset produces exactly one point $(fp_\tau, tp_\tau)$ on the ROC plane. Many classifiers (e.g., Bayesian classifiers) or methods for building classifiers have parameters $\tau$ that can be varied to produce different points on the ROC plane. In particular, a single ranker can be used to generate a set of points on the ROC plane efficiently [Faw03]. Very briefly, the method sorts instances according to assigned scores ($O(n \log n)$) and, exploring the monotonicity of threshold classifications, generates the curve in a single scan ($O(n)$) by iterating over all thresholds, counting positive and negative instances and updating true- and false-positive rates accordingly.

Given a set of points on a ROC plane, the ROC Convex Hull (ROCCH) method [PF98] constructs a piecewise-linear convex down curve (called $ROCCH$) $f_{ROC} : fp \mapsto tp$, having the following properties: (i) $f_{ROC}(0) = 0$, (ii) $f_{ROC}(1) = 1$, and (iii) the slope of $f_{ROC}$ is monotonically nonincreasing. The ROCCH method works by selecting a subset of points on the ROC plane using one of the convex hull generation algorithms (e.g., Graham Scan, Andrew's Monotone Chain algorithm).

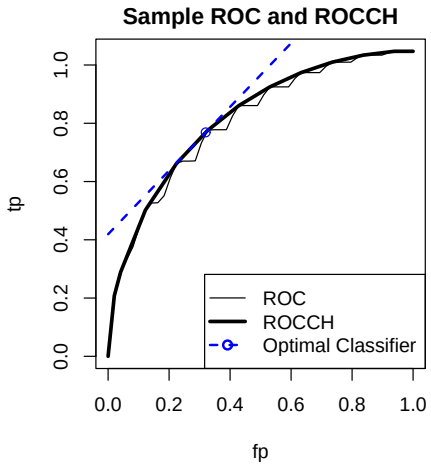



Figure 2.4: Examples of ROC and ROCCH curves and the cost-optimal classifier.

We denote the slope of a point on the ROCCH as ${f'_{ROC}}^*$.

To find the classifier that minimizes the misclassification cost $rc$, we rewrite Equation (2.3) as a function of one variable, $FP$

$$rc = \frac{FP \cdot c_{21} + FN \cdot c_{12}}{N + P} = \frac{FP \cdot c_{21} + P\left(1 - f_{ROC}\left(\frac{FP}{N}\right)\right) c_{12}}{N + P} \ , \tag{2.4}$$

calculate the first derivative $d\,rc/d\,FP$ and set it equal to 0. This yields a known equation of iso-performance lines

$$f'_{ROC}(fp^*) = CR\,\frac{N}{P} \ , \tag{2.5}$$

which shows the optimal slope of the curve given a certain cost ratio ($CR$), $N$ negative, and $P$ positive instances. Similarly to Provost and Fawcett [PF98], we assume that for any real $m > 0$ there exists at least one point $(fp^*, tp^*)$ on the ROCCH having $f'_{ROC}(fp^*) = m$.

Note that the solution of this equation can be used to find a classifier that minimizes the misclassification cost for the instances used to create the ROCCH. We call such a classifier *ROC-optimal*. However, it may not be optimal for unseen instances. Nevertheless, if the testing instances used to build the ROCCH and the other instances are representative, such a ROC-optimal classifier will also perform well on other testing sets.

When the exact misclassification costs and class distributions are not known, the *area under curve (AUC)* is typically used to measure the performance and compare classifiers [HM82, Bra87]. AUC is the area under the ROC curve calculated using a trapezoid rule (linear approximation between consecutive points)

$$AUC = \frac{1}{2}\sum_{i=1}^{n}(tp_{i-1} + tp_i)(fp_i - fp_{i-1}) \ . \tag{2.6}$$

The value of $AUC$ ranges from 0 to 1, with a perfect classifier achieving $AUC = 1$ and a random classifier achieving $AUC = 0.5$[†]. $AUC$ is equivalent to Mann-Whitney-Wilcoxon

[*] We assume that the slope at vertices of a convex hull takes all values between the slopes of adjacent line segments.

[†] However, as noticed by Flach [Fla04] not all classifiers with $AUC = 0.5$ are random.

rank test and can be interpreted as the probability that a randomly chosen positive is ranked before a randomly chosen negative.

While ROC analysis and ROC curves are by far the most commonly used graphical representation of classifier performance, there are also alternative representations: precision-recall curves, DET-curves and cost curves. In Sections 6.7.1–6.7.3 we will discuss the advantages and disadvantages of those alternative representations and how they are related to ROC curves.

### 2.2.5 Unsupervised Techniques

Machine-learning techniques presented in the previous sections focused on classification, i.e., a technique focusing on predicting the value of one target variable based on known values of other variables. However in data mining, i.e., nontrivial extraction of implicit, previously unknown, and potentially useful information from data [FPSM92], also unsupervised machine-learning techniques are used. As we will show in Chapter 8, unsupervised learning algorithms are also useful in the management of intrusion detection alerts. Hence, we will discuss three such techniques: association and episode rules, and clustering.

**Association and Episode Rules** *Association rules* [AIS93, AMS+96] capture implications between attribute values. More formally, an association rule is an implication $X \Rightarrow Y \quad [s, c]$, where $X$ and $Y$ are predicates on a single tuple (data record). Confidence $c$ of such a rule is the conditional probability with which all the predicates $Y$ in the database are satisfied, given that predicates $X$ are satisfied by a tuple. The support $s$ is a fraction of all tuples that satisfy the rule. To illustrate this with an example, assuming that personal preferences are represented as a single tuple, an example of an association rule may be

```
(academic education) and (male) => (interested in golf) [0.02, 0.6]
```

which is interpreted as follows: The probability that a male higher education is interested in golf is 0.6. Moreover, in our dataset there is 2% individuals fulfilling both these criteria.

*Episode rules* [MT96, MTV97] capture relationships between successive tuples. More formally, a serial (parallel) episode $\alpha$ is a collection (multi-set) of tuple predicates. Given a tuple sequence $S$ an episode $\alpha$ occurs in a time interval $[t_s, t_e]$ if it contains a distinct tuple that holds for each predicate from $\alpha$. For serial episodes to occur, tuples must additionally match the predicate order. The interval $[t_s, t_e]$ is the minimal occurrence of $\alpha$ if there is no proper sub-interval that would also be an occurrence from $\alpha$. The episode rule is an expression $\beta[w_1] \Rightarrow \alpha[w_2][s, c]$, where $\beta$ is a sub-episode of $\alpha$ and the two episodes are either serial or parallel episodes. The interpretation of the rule is that if the sub-episode $\beta$ has a minimal occurrence in $[t_s, t_e] \leq w_1$ then the episode $\alpha$ occurs at the interval $[t_s, t'_e] \leq w_2$. The confidence $c$ is the conditional probability that $\alpha$ occurs given $\beta$ occurs under the time constraints specified by the rule. The frequency is the number of times the rule holds in the database [Kle99]. To illustrate this with an example, a rule discovered in a customer-relationship database may say

```
(buy server) and (bootup problem reported) [2 days]
    => (server warranty return) [5 days] [0.01, 0.9]
```

which is interpreted as follows: If a customer bought a server and reported bootup problem within 2 days, the server would be returned to the store within 5 days with the probability 0.9. Moreover, the frequency of such an episode is 1%.

Episode rule mining has been applied to intrusion detection alerts and telecommunication logs. We will discuss these applications in more detail in Section 3.6.

**Clustering** The goal of *clustering* [JD88] is to group tuples into clusters, so that the tuples within the clusters are *similar* whereas the tuples of different clusters are dissimilar. Clearly the notion of similarity is central in this definition. However, while similarity in Euclidean spaces is relatively easy to define, clustering of variables with many categorical variables is less obvious. Clustering has been shown to work well on intrusion detection alerts [Jul03b] and will be further discussed in Section 3.8 and Chapter 8.

## 2.3 Summary

In this chapter we gave a brief introduction to intrusion detection, discussing the basic concepts and principles of operation of two IDSs: Snort, an open source network-based IDS and CSSE, a novel host-based IDS, and machine learning, introducing basic classification techniques, their evaluation, ROC analysis and some related data-mining techniques. We will build upon these concepts in the subsequent chapters.

# Chapter 3

# State of the Art

To the best of our knowledge, machine learning has not previously been used to incrementally build alert classifiers that take background knowledge into account. However, some of the concepts we apply in this dissertation have already been successfully used in intrusion detection and related domains. We therefore present them in this chapter.

## 3.1 Multiple Facets of Related Work

In the first chapter, we presented and categorized the evolution of techniques aiming at reducing the number of false positives into four levels, with our system addressing the problem at the highest, fourth level. While doing this our system integrates three key elements: (i) machine learning, (ii) real-time user feedback, and (iii) alert postprocessing.

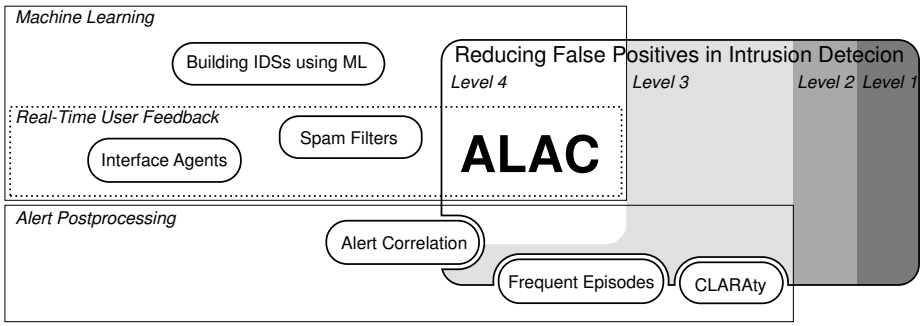



Figure 3.1: Multiple facets of related work.

In this chapter we will show how similar solutions have been used in all these areas as shown in Figure 3.1. The interpretation of the figure as follows: The systems using supervised machine-learning techniques typically use labeled instances coming from the operator (intersection at the fourth level). However, only a subset of those applications acknowledge that the data is acquired incrementally and take advantage of this fact in the learning or the evaluation process. Alert processing approaches, with a few exceptions, address the problem of false positives at the third level. Those systems include alert-correlation systems and data-mining techniques: frequent episodes, association rules and cluster analysis.

## 3.2 Building IDSs Using Machine Learning

In intrusion detection, machine learning has so far been primarily used to build systems that classify network connections (e.g., 1999 KDD CUP [HB99]) or system call sequences (e.g., [MM02]) into one of several predefined classes.

This task proved to be very difficult because it aimed at building IDSs only from training examples. Lee [Lee99] developed a methodology to construct additional features using data mining. He also showed the importance of domain-specific knowledge in constructing such IDSs. The key differences with our work is that we employ the real-time use of analyst feedback and that we classify alerts generated by IDSs, whereas other researchers used machine learning to build a new IDS.

Fan [Fan01] performed a comprehensive study of cost-sensitive learning using classifier ensembles with RIPPER, therefore his work is particularly relevant to ours. The work differs from ours in design goals: we developed a system to assist human users to classify alerts generated by an IDS, whereas Fan built an IDS using machine-learning techniques. We also used a simplified cost model, in order to reduce the number of variable parameters in the system. Finally, the type of learning methods used is also different: ensemble-based learning methods vs. a single classifier in our case.

Lee and Stolfo [LS98] and Hemler et al. [HWHM02] applied RIPPER [Coh95] on system calls to generate small and concise set of rules to classify intrusions for host based IDS.

Similarly, numerous classifiers have been trained and tested on 1999 KDD Cup Dataset [HB99]. The dataset was prepared by Lee [Lee99] and contains a number of network connections have to be classified into one of five categories: normal, DOS (denial of service), R2L (unauthorized access from a remote machine), U2R (unauthorized access to local superuser) and probing. Note that the records are presented in an attribute-value format, with a number of "high-level" features suggested by domain knowledge and generated by an network-based IDS, without which the classification could not have been successful.

Maloof and Michalski [MM95] investigate incremental learning algorithms and their application to intrusion detection. They underline the significance the symbolic representation language and human understandability of background knowledge and learned concepts and criticize a neural network approach. Human understandability is important because should the system act in a way that is harmful to humans, then the concepts responsible for this behavior can be inspected and modified.

Another example of the application of symbolic learning to intrusion detection is learning user signatures [KCM03].

## 3.3 Spam Filtering

The idea of using machine-learning techniques to learn a classifier from a human operator is clearly related to *spam* filtering.

The amount of spam (a.k.a. *unsolicited bulk email*), has greatly increased over the last 10 years, crossing the boundary of 50% and, as of today, constitutes about 65% of all sent email [spa06b]. This means that a random email message is almost twice more likely to be spam than normal mail. In spite of being easily recognizable by humans, the sheer quantity of spam causes significant costs to businesses and individuals, measured in terms of distraction, annoyance, wasted time and increased load on the infrastructure. Consequently, automated

spam-filtering tools are playing an important role in the fight against spam.

The first generation of spam-filtering tools (e.g., early versions of SpamAssassin [spa06a]) used a number of hand-crafted rules, certain keywords or other manually constructed features discriminating between spam and non-spam messages. However, as spammers were typically the first ones to "test" anti-spam solutions and modify their spamming techniques to avoid filtering, such solutions typically did not stay effective for very long.

This gave rise to the second generation of anti-spam filters, which use machine-learning techniques, to discriminate between spam and nonspam messages. In particular Bayesian networks and support vector machines have been successfully used for spam classification [SDHH98, DWV99, Gra02].

Extremely high accuracies of those techniques enabled the wide-spread used and acceptance of supervised spam filters with Bayesian networks, either trained personally by the users (e.g., Mozilla Thunderbird JunkMail Controls [Moz06]), or with boosted learning with other classifiers (e.g., hand-crafted rules, IP-blacklists, mailsinks).

Email classification shares many similarities with our alert-processing system. First, both use binary supervised classification, to classify instances into two groups and, second, both interact with users and use real-time feedback to improve classification. Finally, in both cases the classification is a cost-sensitive issue: typically users prefer false positives (spam messages misclassified as non-spam), than the other way round, especially, if automated actions are implemented (e.g., automatically discarding spam messages).

On the other hand, there is a number of differences between email and alert classification: Spam filtering is essentially text classification (supported by hand-crafted features), while alerts typically consist only of a small number of categorical attributes. Second, in email classification messages are independent and identically distributed (whitelisting and similar techniques are used when the classification depends on previously received messages). Finally in the case of alert classification, we would like to see human-readable rules, which can be interpreted by the analyst, whereas for spam, black-box classifiers are typically acceptable.

## 3.4 Interface Agents

The concept of learning from an operator has been successfully applied in interface agents, that is agents that learn from the user behavior. The example of such agents is email classification into arbitrarily user-defined classes in Magi [PEG97] using two classifiers: rule induction CN2 and nearest neighbor k-NN.

The issues of monitoring and assisting the operator have been investigated in network management and telecommunication networks. Esfandiari et al. [EDQ96] and Nock and Quinqueton [NE98] describe chronicle recognition system learning actions from an operator.

## 3.5 Alert Correlation

Alert correlation (see Appendix A.1 for the clarification of terminologies) tries to solve a different, though related goal of alert processing, namely reconstructing *attacks* and *incidents* from alerts.

Alert correlation is related to the classification discussed in this dissertation in two ways: First, in most cases, an alert-classification system can be chained with alert correlation and classify incidents instead of raw alerts. This way the system benefits from alerts correlation

systems. Second, alert classification can benefit from many ideas in alert correlation, namely properties and heuristics used. Hence, we will examine the work done in this area in greater detail

In this section we provide a summary and our classification of existing alert correlation systems. We discuss those systems in more detail in Appendix A.

**Incident Reconstruction.** Following Howards's classification [HL98], by an *attack* we understand a series of steps taken by an attacker to achieve an unauthorized result. Ideally, there should be one-to-one mapping between alerts reported by an IDS and attacks. However, some attacks can trigger many alerts. For example, a single attack `PORTSCAN` can trigger many IDS alerts, for each host/port scanned. Another example of redundant alerts is in the environment with many IDSs, with some attacks being reported by more than one. Alerts that can be aggregated will be called *redundant*.

Attacks most often occur in distinctive groups, which are called *incidents* (a.k.a. multi-staged attacks). There is a combination of certain factors, some of which we may not know about, which make that several attacks belong to the same incident. Attacks might be launched by one or many attackers related in some way. Attackers might be using similar or completely different tools, pursuing to achieve certain objectives. Attacks can be launched from a single or multiple locations. Finally, attacks might be happening simultaneously or at different times. To summarize, we define *incident* as a group of attacks that can be distinguished from other attacks because of the distinctiveness of the attackers, attacks, objectives, sites and timing.

In the general case, it is not possible to reconstruct incidents from alerts. To illustrate this problem, consider a set of alerts that were triggered by various source hosts. Knowing this, with no additional background knowledge, it is not possible to decide with certainty if these attacks constitute a single coordinated attack, or independent attacks that happen to be interleaved. In case of a single coordinated attack, alerts would have to be grouped into a single incident. By contrast, in the case of multiple interleaved attacks, the alert would have to be partitioned into multiple incidents, namely one incident per attack.

The task of grouping alerts constituting a single attack and replacing them with a single meta-alert will be called *aggregation* and the task of grouping alerts into incidents will be called *correlation*.

Aggregation and correlation are difficult tasks, but can be done correctly in some cases. We should have in mind that the goal of these tasks is to help the human operators as much as possible, but not to replace them. Analyzing partly reconstructed incidents is much easier than browsing an unordered list of IDS alerts, even if some grouping is done incorrectly.

Incident reconstruction and discarding false positives are closely related issues and incident reconstruction can be used to classify alerts. It has been shown [NCR02b, GHH+01] that most of alerts triggered by real attacks can be correlated. Conversely, false positives have different characteristics and correlation algorithms consider them unrelated events. Therefore, filtering out non-correlated alerts significantly reduces a false-positive rate.

Correlation algorithms can be grouped into categories presented in the sections below.

**Exact Feature Similarity.** The simplest form of incident recognition is based on exact similarities between alert features. This approach is intuitive, simple to implement and proved to be effective in the real environment. As proposed by Debar and Wespi [DW01, IBM02] (see

Appendix A.2.1 for details), given three alert attributes (namely attack class, source address and destination address) we can group alerts into scenarios depending on number of matching attributes from the most general (only one attribute matches) to the most specific (all three attributes match). Each of these scenarios has a practical meaning.

This approach can be successfully used in alert pre-processing, although this technique can be limited in correlating more complex attacks (e.g., an attacker compromising the host and launching another attack from there). Another problem is that this technique relies on some numerical parameters and the authors provide little guidance as how to set it.

Another problem is that it would be easy for an intruder to generate a high number of IDS alerts which will not be grouped by this technique. This could make it extremely difficult for the operator to determine which alerts are fake and which constitute the real attack. This type of attack is called "denial of service against the operator" and is also inherent to other correlation techniques.

Therefore, this technique has been more recently used as only one step in multi-step alert correlation systems. Valeur et al. [VVCK04] (see Appendix A.2.9 for more details) use exact feature similarity in two out of ten steps in their alert correlation system: thread reconstruction and focus reconstruction.

**Approximate Feature Similarity.** There have been several heuristic approaches introducing *alert similarity function* i.e., the function representing confidence/probability that two alerts belong to the same group.

The work by Valdes and Skinner [VS01] (see Appendix A.2.2 for details) presents heuristic approach for alert correlation. Dain and Cunningham [DC01] (see Appendix A.2.3) show how parameters can be learned from manually classified scenarios from DEF CON 2000 CTF data. A similar approach, however with different similarity measures, data mining and statistic methods was presented Wang et al. [WL01] (Appendix A.2.6).

Approximate feature similarity techniques, although intuitively appealing and easy to implement, have little theoretical explanation and are ultimately ad hoc. To illustrate this, following the reasoning of Dain and Cunningham, it is difficult to find the justification for using *weighted sum* to calculate the overall similarity given similarities between attributes (e.g., Valdes and Skinner use their *product* instead). Similarly, the choice of sigmoid function for time proximity and the linear function to represent IP addresses similarity is not explained.

Since the heuristic approach is constructed by experts and based on their knowledge, trying to capture some dependencies observed in the real data, its importance cannot be underestimated. The problems, however, are that it is difficult to prove correctness of the similarity functions used. With a variety of traffic, some systems may perform significantly better or worse depending on the particular environment.

There is a lack of formal methods allowing to tune parameters in heuristic algorithms or to prove their usefulness. It is difficult to compare existing algorithms, since no representative training and test data exist (if representative data are at all possible as argued by McHugh [McH00]).

**Semi-formal Attack Graphs.** Semi-formal attacks graphs use domain knowledge such as prerequisites and consequences of intrusions, as well as user defined scenarios (e.g., "recon-breakin-escalate" or "island-hopping") to detect correlation between alerts, even if it is not possible to formalize/capture all prerequisites/consequences relations. To illustrate this with

an example, the prerequisite of an DoS attack is having a root access on a machine and the consequence of a buffer-overflow attack is privilege escalation leading to root access on a machine, thus two alerts representing a buffer-overflow and DoS attacks are related. Therefore such two alerts can be represented as two nodes connected with an edge.

The work by Cuppens et al. [Cup01, CM02, CAMB02] (see Appendix A.2.6) implements this concept using Prolog predicates, while Ning et al. [NCR02b, NCR02a, NRC01, NC02] (see Appendix A.2.4) consider correlation graphs implemented in a relational database.

More recently, Qin and Lee [QL04] (see Appendix A.2.8 for more details) use a simple Bayesian network with a set of predicates based on the domain knowledge at its input to calculate the probability that two (hyper-)alerts should be correlated. The authors also use an adaptive method for updating conditional probability tables based on the performance of the model.

Both of these approaches help to substantially reduce false positives and discover incidents. One of the main problems is that they require the definitions of prerequisites and consequences for each alert, which in the general case is difficult to achieve.

**Formal Attack Graphs** Formal attack graphs take the prerequisite/consequence approach to model the networking environment, computer systems, the software they run and their vulnerabilities. An example of such model is an M2D2 model [MMDD02] (see Appendix A.2.7 for more details).

An attack graph is a succinct representation of all paths through a system that end in a state where an intruder has successfully achieved his goal. Using formal model checking and attack-graph analysis allow us to evaluate the vulnerability of a network of hosts and find the global vulnerabilities and can serve as a basis for detection, defense and forensic analysis [RA00].

Attack graphs can enhance both heuristic and probabilistic correlation approaches [JSW02, SHJ+02]. Given the graph describing all likely attacks and IDSs, we can match individual alerts to attack edges in the graph. Being able to match successive alerts to individual paths in the attack graph significantly increases the likelihood that the network is under attack. Hence, it is possible to predict the attacker's goals, aggregate alarms and reduce the false alarm rates.

Formal techniques have a great potential, but at the current stage of research can serve as a proof-of-concept rather than a working implementation. It can be expected that with more formal definitions of vulnerabilities and security properties, formal analysis techniques will gain more significance.

## 3.6 Frequent Episodes & Association Rules

Recall in Section 2.2.5 we introduced association rules and frequent episodes. Both these techniques were used with a great success on alarms in telecommunication networks in a system called TASA [HKM+96, KMT99, Kle99]. The system have been actively used by telecommunication operators and have been found useful in the following three areas: (i) finding long-term rather frequently occurring dependencies, (ii) creating an overview of a short-term alert sequences and (iii) evaluating the alert database consistency and correctness [KMT99].

Considering the similarities between the domains of telecommunication networks and computer security, it is natural to ask how association rules and frequent episodes perform on

IDS alert logs.

As reported by Julisch [Jul03b] mining IDS alert logs for episodes and episode rules yielded a number of interesting patterns, including episodes characteristic of certain attack tools, IDS properties (certain alerts always entail another ones) as well as legitimate system operations. However, this only accounted for less than 5% of all alerts. The remaining episodes and episode rules were irrelevant and redundant patterns, otherwise difficult to interpret. Consequently, Julisch concluded that episode rule mining was not suitable for mining IDS alert logs.

## 3.7 Sensor Profiling

Manganaris et al. [MCZH00] create an anomaly-based sensor processing alerts generated by commercial IDS sensors. The system partitions alerts into alert bursts and mines the bursts for frequent itemsets and association rules. Subsequently, the set of discovered association rules is used without any inspection or modification as the reference for the normal alarm behavior. In this anomaly-based approach, deviations from this model are reported to the analyst.

The fact that only anomalous bursts are reported and that the association rules are not inspected (in fact it is not feasible with thousands of association rules mined), incurs a potentially high risk of true alarms being discarded if they "creep" into the reference model and prevent future detection of attacks. The alternative approach of using attack-free alarm logs to learn association rules was reported to be equally difficult to implement [Jul03b].

## 3.8 CLARAty—Data Mining and Root Cause Analysis

Julisch [Jul03b, Jul03a] applied data-mining techniques to discover root causes in alert logs, which in turn can be either eliminated (by modifying the environment) or used to create alert filters to prevent similar alerts from occurring in the future.

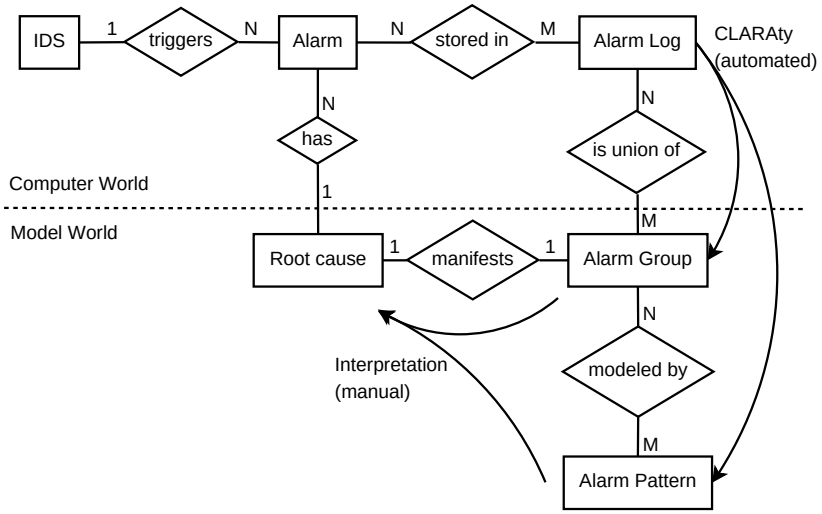



Figure 3.2: Entity-relationship diagram of concepts used by CLARAty [Jul03b].

Underlying this method is the concept that for every observed alert there is a *root cause*, which is the reason for its generation. More specifically, as shown in Figure 3.2, IDS triggers

a number or *alarms*, stored in *alarm log*. Data-mining techniques discover *alarm groups*, modeled by *alarm patterns*. Those alarm patterns and alarm groups can be used to reconstruct the original root causes with little manual intervention.

The hypothesis, verified by [Jul03b], which makes this approach interesting is (*i*) that large groups of alerts have a common root cause, (*ii*) that few of these root causes account for a large volume of alerts, and (*iii*) that these root causes are relatively persistent over time. This means that discovering and removing a few most prominent root causes can safely and significantly reduce the number of alerts the analyst has to handle.

To verify this hypothesis, Julisch developed a clustering algorithm called *CLARAty* (CLustering Alerts for Root cause Analysis), based on a modified attribute-oriented induction (AOI) [HCC92, HCC93] algorithm, to make it more suitable for alert processing and applied it to a selection of alerts collected in the MSSP environment. He then showed that by interpreting those clusters and judiciously converting them to alert filters, one can remove up to 70% of future alerts.

This approach is orthogonal to human-assisted alert classification proposed in this thesis, as cluster analysis can be done independently of human-assisted alerts classification, however, it is natural to ask if combining these two approaches works well in practice. To answer this question in the third part of the thesis, we propose a *two-stage alert-classification system* with automated cluster processing and validate this system in practice.

## 3.9 Summary

In this chapter we categorized existing efforts aiming at reducing false positives in intrusion detection on the following four levels: improving IDSs themselves, leveraging the environment, alerts postprocessing and analyst's involvement. While, with a few exceptions, most of the existing work has focused on the first three levels, our approach focuses on the fourth level.

Our system, combining the use of machine learning, real-time user feedback and alert postprocessing is similar to techniques from related domains, like developing IDSs using machine-learning techniques, building spam filters or developing interface agents. We investigated these techniques and analyzed differences and similarities to our approach.

Finally, we discussed other alert postprocessing approaches in more detail, which can, in many cases, be used together with our system in a multi-stage alert-classification system. The effects of integration with one of them, namely the data mining algorithm called CLARAty, will be further investigated in Chapter 8.

# Chapter 4

# Datasets Used

In this chapter we discuss the datasets which can be used in the evaluation of our system and their characteristics. This will give the reader a better understanding of the datasets available and the ones we chose for the evaluation of our system. Since the evaluation is a central problem in this thesis, the characteristics motivates our work and gives more insight to the design of our system.

## 4.1 Datasets Available

The lack of publicly available and representative datasets hinders intrusion-detection research and makes the comparison of different intrusion detection systems and algorithms difficult. Whereas it is easy to generate a large set of intrusion detection alerts (e.g., by running an IDS in a private or Internet-exposed network), such an approach poses two major problems: First, such datasets are unlabeled (i.e., it is not clear which attacks are false positives and which are true positives), and their labeling poses a major challenge. Unless done by multiple independent analysts, such labeling can be questioned and subject to discussion. The second problem is of a different nature. In most cases the data gathered in by intrusion detection systems is of a confidential nature. The data inherently contains information about network topology, hosts and other confidential information (e.g., the content of e-mails, visited websites or even passwords transmitted in clear text). Hence, access to this data is strictly limited and cannot be shared with others.

**Simulated Datasets** There has been an attempt to provide a publicly available dataset (DARPA 1998 [LFG+00] and DARPA 1999 [LHF+00, MIT99]) generated in a simulated environment, however many flaws have been identified both in the simulation as well as the evaluation procedures [McH00, McH01, MC03]. These datasets are, nonetheless, the only publicly available datasets and are useful in evaluating IDSs. In fact there has been a number of publications both in intrusion detection as well as in the machine learning communities using these datasets (e.g., [NCR02b, Lee99, FLSM00]).

**Honeypot Datasets** Another common approach is to use data collected by so-called *honeypots* [Bel92], systems deploying fake [Pro04, RZD05] (or real) network services with the purpose of being attacked. Any access to these services is being monitored and subsequently

collected data can be used for data mining and manual or automated forensic analysis, in order to understand new intrusion patterns and techniques. The data is by definition abnormal and indicative of intrusions or intrusion attempts as normal systems have no legitimate reasons to contact honeypots. Hence, the data typically does not contain sensitive information and can be easily shared (e.g., Leurré.com Project [PDP05]). However, due to its nature, the honeypot data is more useful for detecting automated attacks such as worms or malware-infected machines, rather than human attackers [LLO+03, RZD05]. Moreover, as all the data is by definition suspicious, there is little point in building an alert classifier to classify it into true and false positives.

**"Attack-only" Datasets** DEFCON 9 CTF [Dar01] is another dataset commonly used in intrusion detection evaluation. DEFCON is an annual underground hacking conference, which includes a hacking contest Capture The Flag (CTF). All traffic generated during the competition is recorded and made available. The dataset can be used for testing intrusion detection systems, although its small number of IP addresses, an unusually high concentration of intrusions and hence a lack of background traffic makes it not really suitable for the evaluation of IDS alert classification.

The Treasure Hunt dataset [Vig03] is a dataset collected during a Cyber Treasure Hunt competition organized as a part of a graduate class at some university. Similarly to the DEFCON 9 CTF this dataset exhibits an unusually high concentrations of intrusions, which makes it more suitable for IDS stress-testing and alert correlation, rather than alert classification.

**Real Datasets** Finally, another source of data are alerts from real IDSs deployed in corporate environments. As IDSs are only useful when the alerts they produce are reviewed regularly and IDSs are commonly deployed, we can assume that most IDS alerts are in fact reviewed. During the review the analyst tries to understand the root cause of an alert, determines if it is benign or malicious and if there is any action that needs to be taken. Depending on the environment, the review can be more or less formal: In some cases every alert gets assigned a label determining its root cause; in others only the successful attacks are investigated, making this an implicit classification. Obviously, the first scenario is much more useful for us, because there is a variety of root causes and their distribution is more balanced. In the second scenario, successful attacks are typically very rare events, making supervised classification difficult.

The advantage of using real datasets is that these are real environments the system can be used in, hence the problem whether the background traffic is representative of the "real" background traffic does not exist. On the other hand, the real data has the problem that the alert classification is subjective and sometimes incomplete or incorrect. To illustrate this with an example, assume that the analyst had investigated an intrusion $I$, which triggered three alerts $A1$ and, at some time later, $A1'$ and $A2$. If the analyst had missed an initial alert $A1$ and only marked $A1'$ and $A2$ as intrusive, we would get the following training examples $\{(A1, FALSE), (A1', TRUE), (A2, TRUE)\}$. If the alerts $A1$ and $A1'$ were similar, the analyst would not have made a big mistake in his classification, as it does not change the classification of the intrusion. On the other hand this makes it much more difficult to learn how to classify these intrusions correctly. Similarly, the human analysts can make other types of mistakes: miss an intrusion or erroneously classify benign activities as intrusive. This, given the fact that intrusions are a rare phenomenon in general, makes the classification of alerts an extremely difficult task.

To summarize, none of the discussed datasets is ideal and each has some advantages and disadvantages. As our goal is alert classification of IDS alerts we are focusing on datasets with the most realistic attack data and background traffic, thus yielding a similar mix of true and false positives as can be seen in real environments. To make our results the most representative, we will evaluate the system on multiple datasets, which are discussed in the following section.

## 4.2 Datasets Used & Alert Labeling

In our previous work [Pie04], we used two independent datasets: DARPA1999 Data Sets and Data Set B, a proprietary dataset collected in a mid-sized corporate network over the period of 1 month. For both of these datasets we ran Snort [Roe05], an open-source signature-based intrusion detection system. The resulting alerts have been labeled using attack truth tables and manually by the author, for DARPA 1999 Data Set and Data Set B, respectively.

The statistics for these datasets are shown in Table 4.1.

Table 4.1: Statistics generated by the Snort sensor with DARPA 1999 Data Set and Data Set B.

| | DARPA 1999 | Data Set B |
|---|---|---|
| Duration of experiment: | 5 weeks | 1 month |
| Number of IDS alerts: | 59812 | 47099 |
| False alerts: | 48103 | 33220 |
| True alerts: | 11709 | 13383 |
| Unidentified: | — | 496 |

### 4.2.1 Alert Representation

Alerts $\{A_1, \ldots, A_n\}$ generated by different IDSs and stored in an alert log $L$, may have different forms, however, they can be easily represented as tuples of $n$ attributes $(T_1, T_2, \ldots, T_n)$. Alert attributes $T_i$ capture intrinsic properties of alerts, such as the address of an attacker, the type of an alert and the timestamp. In the case of network-based IDSs used in this dissertation, the common attributes include:

**timestamp** is in general a numerical attribute, commonly represented as a number of seconds since the "Unix epoch" (Jan 1, 1970). Note that this representation captures the continuous aspect of time, however disregards its periodic nature (e.g., hours, minutes, days of week, days of month, months).

**protocol** is a categorical attribute determining the protocol e.g., `TCP`, `UDP`, `ICMP`.

**source address** is a categorical attribute describing the attacker, i.e., the computer system that triggered an alert. The source address is typically a 32-bit IP address, however in case of commonly used protocols like TCP and UDP, the source address is a pair $(ip, sport)$, where $ip$ is a 32-bit IP address and $sport$ is a 16-bit source port. In such cases, the address is typically represented as two distinct attributes. Note that even if IP addresses are often represented as numerical attributes they are in fact categorical

(e.g., comparison like $ip < 12345678$ does not make sense). However, IP addresses and networks are hierarchical (e.g., `192.168.0.1` and `192.168.0.10` are covered by `192.168.0.0/24`, which is covered by `192.168.0.0/16`).

**destination address** is a categorical attribute describing the victim, i.e., the computer system under attack. Similarly, to the source address, in case of protocols like TCP or UDP, the address also contains a 16-bit port however, unlike the source ports, which are typically random, the destination port determines the service of the computer (e.g., 22 is a SSH port, 80 is an HTTP port).

**signature** is a categorical attribute determining the type of an attack detected by an IDS, e.g., `IIS exploit`, `PORTSCAN`. Typically signatures can be further grouped into groups of similar types of attacks (e.g., scanning, webserver exploits, policy violation).

**payload** is a free text attribute allowing the analyst to further determine the nature of the problem, by inspecting the fragment of actual traffic, which triggered an alert.

From the above attributes, we selected the following attributes for our representation: source and destination IP addresses, signature type and three additional attributes determining the type of a scanning alert (one categorical attribute `SCAN`, `NOSCAN` and two attributes determining the number of hosts and port scanned). We did not use directly the time attribute and the payload as they cannot be directly represented in our learning framework. Similarly, we did not use the protocol attribute and the ports, as they are either a direct (or almost direct) function of the signature (e.g., protocol and the destination port) or not relevant (e.g., source port). In addition to this basic alert representation we used additional background knowledge discussed in Section 5.2.

### 4.2.2 DARPA 1999 Data Set

In spite of its weaknesses we decided to use the DARPA 1999 Data Set as to the best of our knowledge it is the only publicly available "reference" dataset, which can be used be independent researchers to reproduce our results.

The DARPA 1999 Data Set is a synthetic dataset collected from a simulated medium-sized computer network in a fictitious military base. The network was connected to the outside world by a router. The router was set to open policy, i.e., not blocking any connections. The simulation was run for five weeks which yielded three weeks of training data and two weeks of testing data. Attack truth tables describing the attacks that took place exist for both periods. DARPA 1999 data consists of: two sets of network traffic (files with tcpdump [JLM03] data) both inside and outside the router for network-based IDSs; and BSM [Sun95] and NT audit data, and daily directory listings for host-based IDSs.

For our purposes we need classified IDS alerts, not only raw data. For this we used Snort, to detect attacks and generate alerts. We ran Snort in batch mode using traffic collected from outside the router for both training and testing periods. Note that Snort missed some of the attacks in this dataset. We purposely used the basic out-of-the box configuration and rule set to demonstrate the performance of our system in reducing the amount of false positives and therefore reducing time-consuming IDS tuning. Some of them could only be detected using a host-based IDS, whereas for others Snort simply did not have the right signature. It is important to note that our goal was not to evaluate the detection rate of Snort on this data, but to validate our system in a realistic environment.

The DARPA 1999 Data Set has many well-known weaknesses (e.g., [MC03, McH00, SS04]) and we want to make surethat using it we get representative results for how ALAC performs in real-world environments. Therefore we tried to analyze how the weaknesses identified by McHugh [McH00], namely the generation of attack and background traffic, the amount of training and test data for anomaly-based systems (also investigated by Sabhnani and Serpen [SS04]), attack taxonomy and the use of ROC analysis can affect ALAC.

With respect to the training and test data for the original system, we did not use the original training-testing data split and combined both datasets for the evaluation of incremental system. In our incremental setup, we start with a small training set, evaluate the system on new alerts and subsequently add them to the training set for incremental classification. With respect to the attack taxonomy, we are not using the scoring used in the original evaluation, criticized by McHugh. Finally, we use ROC analysis correctly. Therefore these issues identified by McHugh do not apply to our evaluation.

The problem of the simulation artifacts is more thoroughly analyzed by Mahoney and Chan [MC03] thus we use their work to understand how these artifacts can affect ALAC. These artifacts manifest themselves in various fields, such as the TCP and IP headers and higher protocol data. Snort, as a signature based system, does not take advantage of these artifacts and ALAC sees only a small subset of them, namely the source IP address. We verified that the rules learned by ALAC seldom contain a source IP address and therefore the system does not take advantage of simulation artifacts present in source IP addresses. On the other hand, we cannot easily estimate how these regularities affect aggregates used in the background knowledge. This is still an open issue.

Nonetheless, due to its public availability and prior work using it, DARPA1999 Data Set is valuable for evaluation of our research prototype.

**Alert Labeling** Our system assumes that alerts are labeled by the analyst. Recall that, in a first step we generated alerts using Snort running in batch mode and writing alerts into a relational database. In the second step we used automatic labeling of IDS alerts using the provided attack truth tables.

For labeling, we used an automatic approach, which can be easily reproduced by researchers in other environments, even with different IDS sensors. We consider all alerts meeting the following criteria related to an attack: (i) matching source IP address, (ii) matching destination IP address and (iii) alert time stamp in the time window in which the attack has occurred. We masked all remaining alerts as false alerts. While manually reviewing the alerts we found that, in many cases, the classification is ambiguous (e.g., a benign PING alert can be as well classified as malicious if it is sent to the host being attacked). This may introduce an error in class labels. We will further discuss this issue in Section 8.4.7.

Note that different attacks triggered a different number of alerts (e.g., wide network scans triggered thousands of alerts). For the evaluation of our system we discarded the information regarding which alerts belong to which attack and labeled all these alerts as true alerts.

### 4.2.3 Data Set B

Owing to the problems with the DARPA 1999 Data Set we decided to use another independent dataset to validate our system. Data Set B is a real-world dataset collected over the period of one month in a medium-sized corporate network. The network connects to the Internet through firewalls and to the rest of the corporate intranet and does not contain any externally

accessible machines. Owing to privacy issues this dataset cannot be shared with third parties. We do not claim that it is representative for all real-world datasets, but it is an example of a real dataset on which our system could be used.

Similarly to the previous dataset, we used the same configuration of Snort and run the sensor observing information exchanged between the Internet and the intranet.

**Alert Labeling** As opposed to the first dataset we did not have information concerning attacks. The alerts have been classified based on the author's expertise in intrusion detection into groups indicating possible type and cause of the alert. There was also a certain number of alerts that could not be classified into true or false positives. Similarly to the first dataset we used only binary classification to evaluate the system, and labeled the unidentified alerts as true positives.

Note that this dataset was collected in a well-maintained and well-protected network with no direct Internet access. We observed a low number of attacks in this network, but many alerts were generated. We observed that large groups of alerts can be explained by events such as a single worm infection and unauthorized network scans. The problem of removing such redundancy can be solved by so-called *alert correlation systems* [Cup01, DW01, VS01], where a group of alerts can be replaced by a meta-alert representative of the alerts in the group, prior to classification.

Another issue is that the classification of alerts was done by only one analyst and therefore may contain errors. This raises the question of how such classification errors affect the performance of our system. To address this issue, one can ask multiple analysts to classify the dataset independently. Then the results can be compared using interrater reliability analysis.

### 4.2.4 MSSP Datasets

Considering the problems wit the datasets above we decided to use another data source, alerts collected in a SOC (Security Operations Center) run by IBM Global Services Managed Security Services team, further referred to as a Managed Security Service Provider (MSSP). Unlike the previous datasets, these are alerts generated by commercial IDSs and reviewed by real security analysts as part of their job duties.

For our experiments we used a period of six consecutive months (2H 2005) for a selection of 20 undisclosed customers of the MSSP. In many cases a single customer has more than one IDS, including both network-based (NIDS) and host-based (HIDS) sensors. As the scope of our analysis are NIDSs we used only those sensors for the analysis. The sensors here were a mix of two different commercial products (not an open-source Snort sensor with the previous two datasets). To avoid unintentional commercial implications we will leave the name of these products undisclosed.

During our experiment we performed an automated correlation of all alerts with the alerts flagged as *security incidents* by the security analysts at the MSSP. Incidents are disjunctive and each of them contains at least one alert (in fact many incidents several hundred alerts). We do not know if the classification we received in the database is final—the incidents were likely to be investigated by additional analysts and also consulted with the customer and their classification could have changed.

For our purposes we used only information whether an alert belongs to an incident and marked all remaining alerts as false positives. Note that this approach may underestimate the number of all true positives if not all alerts constituting an incident were classified as such.

As in the previous example, the fact that the incident took place is by far more important for the analyst than enumerating all alerts it triggered. Knowing that the incident has taken place, the customer examines the systems affected and performs forensic investigation, not focuses on individual IDS alerts.

The statistics for these datasets are shown in Table 4.2, where undisclosed customers are identified by a numeric identifier.

Table 4.2: Statistics generated for 20 companies from MSSP database. Customer are identified by means of a unique identifier.

| Customer | Days | Total Alerts | Total Positives | (%) | Total Incidents |
|---|---|---|---|---|---|
| 3288 | 178.0 | 107081 | 4060 | 3.79 | 10 |
| 3359 | 90.0 | 74884 | 400 | 0.53 | 12 |
| 3362 | 179.0 | 1261026 | 499 | 0.04 | 21 |
| 3363 | 179.0 | 1812067 | 2089 | 0.12 | 28 |
| 3380 | 178.0 | 392563 | 416 | 0.11 | 7 |
| 3408 | 178.0 | 55781 | 535 | 0.96 | 2 |
| 3426 | 179.0 | 805435 | 199 | 0.02 | 9 |
| 3473 | 178.0 | 466285 | 107 | 0.02 | 29 |
| 3482 | 144.0 | 102255 | 994 | 0.97 | 47 |
| 3488 | 179.0 | 2415761 | 247 | 0.01 | 34 |
| 3491 | 178.0 | 99079 | 1012 | 1.02 | 8 |
| 3520 | 178.0 | 118434 | 1330 | 1.12 | 6 |
| 3532 | 178.0 | 78324 | 804 | 1.03 | 5 |
| 3565 | 178.0 | 134933 | 1515 | 1.12 | 30 |
| 3569 | 179.0 | 890919 | 14247 | 1.60 | 22 |
| 3590 | 178.0 | 573442 | 405 | 0.07 | 4 |
| 3626 | 178.0 | 115400 | 440 | 0.38 | 6 |
| 3647 | 180.0 | 1341856 | 6159 | 0.46 | 16 |
| 3669 | 179.0 | 971635 | 738 | 0.08 | 7 |
| 4043 | 180.0 | 2036648 | 5249 | 0.26 | 18 |

We notice that there are on average 3896 alerts per customer per day, out of which only 12 are true positives (0.31%). In addition, alerts are not uniformly distributed and occur in groups called incidents. There are on average 0.096 incidents per company per day or, in other words, an incident takes place every 10 days for every company. This shows that we are working with an extremely skewed dataset, which requires special care to handle properly.

**Why are These Datasets Different** Looking at the statistics for all our datasets we notice that they have quite different characteristics. In contrast to the MSSP datasets, DARPA 1999 and Data Set B have on average 1472 alerts per company per day, out of which as much as 359 are true positives (24%). We think that the differences among the datasets can be attributed to the following factors:

**Different Classification Method:** In the classification of Data Set B we assumed that any event, which cannot be easily explained (or is otherwise suspicious) was classified as true positive. Conversely, the MSSP Datasets contain events which were classified

by analysts as intrusions or even only successful intrusions. This explains the lower incident rate.

**Fewer Incidents:** The DARPA dataset assumed an unrealistically high incident frequency. In the normal, well protected environment incidents are very rare events. Similarly, even the systems facing the Internet are typically protected with a firewall, which prevents most attack attempts.

Table 4.3: Comparison of properties of three datasets used in this dissertation.

| Dataset | Public | IDS | #Alerts | Classification | Comments |
|---|---|---|---|---|---|
| DARPA 1999 Data Set | yes | Snort | $\sim 60k$ | automatic, attack truth tables | + reproducible by other researchers<br>+ objective classification<br>− many flaws have been identified, simulation artifacts |
| Data Set B | no | Snort | $\sim 50k$ | manual, author | + real environment<br>− classification bias |
| MSSP Datasets | no | two types of commercial IDSs | $\sim 14M$ | automatic, security analysts | + real environment, real analysts<br>− extremely skewed class distribution<br>− incomplete classification, due to the way the data was originally used |

The summary of the characteristics used in this dissertation is showed in Table 4.3. In spite of these differences, all these datasets represent a possible sample of IDS alerts, the alert-classification system proposed in this dissertation can face in real environments. We will base the design system on these observations and try to make it robust and perform well even in such varied environments.

## 4.3 Summary

In this chapter we discussed the problem of datasets for evaluation intrusion detection systems and alert-classification systems. We analyzed the possible data sources we could use, discussed their properties, advantages and disadvantages. We selected the following three datasets: (i) DARPA 1999 Data Set, a simulated dataset with alerts generated using Snort; (ii) Data Set B, a proprietary dataset with alerts generated by Snort, classified by the author; (iii) MSSP datasets, a set of proprietary datasets with alerts generated by commercial IDSs and intrusions analyzed by professional security analysts.

# Chapter 5

# Adaptive Alert Classification

In this chapter we investigate the task of alert classification performed by intrusion detection analysts and present a novel concept of building an adaptive alert classifier using machine-learning techniques. We then analyze the requirements of such a system, select a suitable machine-learning technique and validate the system on one synthetic and one real dataset.

## 5.1 ALAC—Adaptive Learner for Alert Classification

This chapter is organized as follows: Starting from the problem specification we introduce ALAC, our alert-classification system and its two modes: the agent mode and the recommender mode. We will then discuss important aspects of the system: the representation of background knowledge (Section 5.2), suitable machine-learning techniques (Sections 5.3 and 5.4). Subsequently, we will evaluate ALAC in Section 5.5.

Recall in Section 1.1.3 we introduced the problem specification and the utility function $\mathcal{U}$ defining the value of the classification system for the analyst $\mathcal{O}$. We also showed how different definitions of the utility function $\mathcal{U}$ determine the systems are developing. In this chapter we focus on the first two components of the utility function $\mathcal{U}$, namely misclassified alerts and analyst's workload, not allowing for abstentions. Assuming that the system classifies all alerts and passes them to the analyst, we identified two main types of assistance, namely a recommender mode and an agent mode, depending on whether the system allows for autonomous alert processing of alerts. We will discuss these modes in the following sections.

We present ALAC, an *Adaptive Learner for Alert Classification*, whose architecture is depicted in Figure 5.1. The system uses an alert classifier to assign class labels to alerts and passes them to the analyst. Subsequently, the analyst investigates the labels and corrects them if necessary.

Currently we use a simple human-computer interaction model, in which the analyst sequentially classifies alerts into true and false positives, which are converted to training examples, however, more sophisticated interaction techniques are also possible. For example, the analyst can see alerts out of order (e.g., grouped by IP addresses, signatures or their classification), and upon changing the classification given by the system, the system can update the classifier and relabel all visible alerts affected.

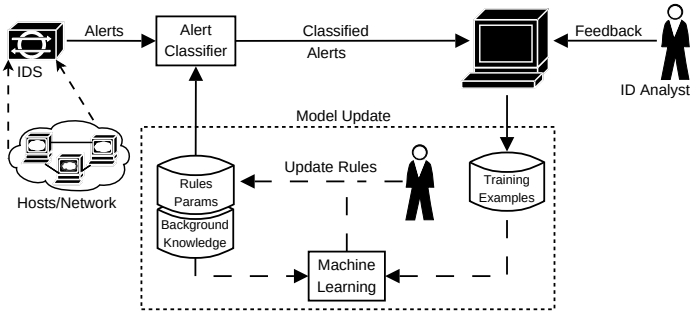
More formally, there is a human intrusion detection analyst $\mathcal{O}$ reviewing a sequence of intrusion detection alerts $(A_1, A_2, \ldots, A_i, \ldots)$ in the alert log $L$. The review is done by assigning one of the predefined set of classes $\{C_1, C_2, \ldots, C_n\}$ (which can be in particular

two classes: true positives and and false positives {"+", "−"}) to each alert. The review is typically done sequentially and in real-time, which means that alert $A_{i+1}$ is reviewed only after alerts $(A_1, \ldots, A_i)$ have been reviewed and, at this time, alerts $(A_{i+2}, \ldots)$ are not known. The goal in defined as:

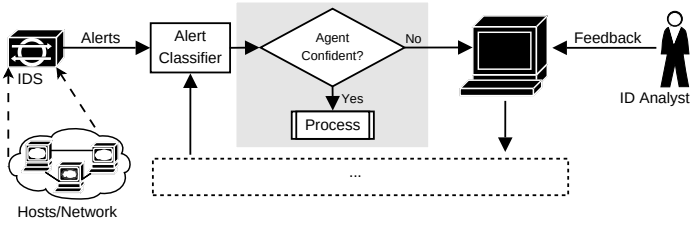
| | |
|---|---|
| **Given** | – A sequence of alerts: $(A_1, A_2, \ldots, A_i, \ldots)$ in the alert log $L$,<br>– a set of classes $C = \{C_1, C_2, \ldots, C_n\}$,<br>– an intrusion detection analyst $\mathcal{O}$ sequentially and in real-time assigning classes to alerts,<br>– a utility function $\mathcal{U}$ minimizing the misclassification cost, |
| **Find** | A classifier classifying alerts, maximizing the utility function $\mathcal{U}$. |

In addition to training examples, we use background knowledge to learn improved classification rules. These rules are then used by ALAC to classify alerts. Thanks to using interpretable classifiers, the analyst can inspect the rules to make sure they are correct.

The architecture presented above describes the operation of the system in the *recommender mode*. The second mode, the *agent mode*, introduces autonomous processing to reduce the analyst's workload.




(a) Recommender mode




(b) Agent mode

Figure 5.1: Architecture of ALAC in agent and recommender modes.

### 5.1.1 Recommender Mode

In recommender mode (Figure 5.1a), ALAC classifies alerts and passes all of them to the console to be verified by the analyst. In other words, the system assists the analyst suggesting the correct classification. The advantage for the analyst is that each alert is already preclassified and that the analyst only has to verify its correctness. Hence, the analyst can prioritize his or her work, e.g., by dealing with alerts classified as true positives first or sorting the alerts

by classification confidence. It is important to emphasize that in the end, the analyst will review all classifications made by the system.

This is a conventional incremental learning setup. Algorithm 2 shows the operation of ALAC in the recommender mode. The function *goodClassificationPerformance()* estimated the performance of the classifier on a confusion matrix using a weighted accuracy ($WA$) with a threshold $WA_{th}$.

```
   Input: a sequence of alerts (A_1, A_2, ..., A_n)
   Result: a sequence of classified alerts ((A_1, C_{A_1}), (A_n, C_{A_n}), ..., (A_n, C_{A_n}))
 1 initialize;
   /* alerts used for the initial training                                    */
 2 x ← x_0;
 3 while x < n do
 4 |   S_i ← subsequence(A_1, ..., A_x);
 5 |   C_i ← learnUpdateClassifier(C_{i-1}, S_i);
 6 |   while goodClassificationPerformance(WA_th) do
 7 |   |   C_x ← classify(C_i, A_x);
 8 |   |   C_{A_x} ← askAnalystVerifyClassification(A_x, C_x);
 9 |   |   updateClassificationPerformance(C_x, C_{A_x});
10 |   |   x ← x + 1;
11 |   end
12 |   i ← i + 1;
13 end
```

**Algorithm 2**: ALAC algorithm—recommender mode.

### 5.1.2 Agent Mode

In the agent mode, the high level goal uses a modified utility function $\mathcal{U}$, so that it also limits analyst's workload and therefore a system that autonomously processes some alerts is preferred over the one that does not. More formally, similarly to the previous case we specify the goal as:

| | |
|---|---|
| **Given** | – A sequence of alerts: $(A_1, A_2, \ldots, A_i, \ldots)$ in the alert log $L$,<br>– a set of classes $C = \{C_1, C_2, \ldots, C_n\}$,<br>– an intrusion detection analyst $\mathcal{O}$ sequentially and in real-time assigning classes to alerts,<br>– a utility function $\mathcal{U}$ minimizing the misclassification cost and analyst's workload (allowing for autonomous alert processing). |
| **Find** | A classifier classifying alerts, maximizing the utility function $\mathcal{U}$ |

In agent mode (Figure 5.1b), ALAC autonomously processes some of the alerts based on criteria defined by the analyst (i.e., classification assigned by ALAC and classification confidence). By processing alerts we mean that ALAC executes user-defined actions associated with the class labels and classification confidence values. For example, attacks classified as false positives can be automatically removed, thus reducing the analyst's workload. In contrast, alerts classified as true positives and successful attacks can initiate an automated response, such as reconfiguring a router or firewall. It is important to emphasize that such

actions should be executed only for alerts classified with high confidence, whereas the other alerts should still be reviewed by the analyst.

Note that autonomous alert processing may change the behavior of the system and negatively impact its classification accuracy. To illustrate this with an example, suppose the system classifies alerts into true and false positives and it is configured to autonomously discard the latter if the classification confidence is higher than a given threshold value. Suppose the system learned a good classifier and classifies alerts with high confidence. In this case, if the system starts classifying all alerts as false positives then these alerts would be autonomously discarded and would never be seen by the analyst. These alerts would not become training examples and would never be used to improve the classifier.

Another problem is that alerts classified and processed autonomously cannot be added to the list of training examples as the analyst has not reviewed them. If alerts of a certain class are processed autonomously more frequently than alerts belonging to other classes (as in the above example), as a consequence we change the class distribution in the training examples. This has important implications as machine-learning techniques are sensitive to class distribution in training examples. In the optimal case, the distribution of classes in training and testing examples should be identical.

To alleviate these problems, we use a technique called *random sampling*. In this technique we randomly select a fraction $s$ of alerts which would normally be processed autonomously and instead forward them to the analyst. This ensures the stability of the system. The value of $s$ is a tradeoff between how many alerts will be processed autonomously and how much risk of misclassification is acceptable.

The exact operation of ALAC in the agent mode is shown in Algorithm 3.

**Input**: a sequence of alerts $(A_1, A_2, \ldots, A_n)$
**Result**: a sequence of classified alerts $((A_1, C_{A_1}), (A_n, C_{A_n}), \ldots, (A_n, C_{A_n}))$
initialize;
`/* alerts used for the initial training */`
$x \leftarrow x_0$;
**while** $x \leq n$ **do**
  $S_i \leftarrow$ subsequence$(A_1, \ldots, A_x)$;
  $\mathcal{C}_i \leftarrow$ learnUpdateClassifier$(\mathcal{C}_{i-1}, S_i)$;
  **while** *goodClassificationPerformance(*$WA_{th}$*)* **do**
    $C_x \leftarrow$ classify$(\mathcal{C}_i, A_x)$;
    **if** $(\neg confident(C_x, c_{th})) \vee (C_x == TRUE\_ALERT) \vee (randomSample(s))$ **then**
      $C_{A_x} \leftarrow$ askAnalystVerifyClassification$(A_x, C_x)$;
    **end**
    **else**
      `/* e.g., discarding of false positives */`
      automaticProcessing() ;
      $C_{A_x} \leftarrow \emptyset$;
    **end**
    updateClassificationPerformance$(C_x, C_{A_x})$;
    $x \leftarrow x + 1$;
  **end**
  $i \leftarrow i + 1$;
**end**

**Algorithm 3**: ALAC algorithm—agent mode.

## 5.2 Background Knowledge

Recall that we use machine-learning techniques to build the classifier. In machine learning, if the learner has no prior knowledge about the learning problem, it learns exclusively from examples. However, difficult learning problems typically require a substantial body of prior knowledge [LD94], which makes it possible to express the learned concept in a more natural and concise manner. In the field of machine learning such knowledge is referred to as *background knowledge*, whereas in the field of intrusion detection it is quite often called *context information* (e.g., [SP03]). Background knowledge is very important in intrusion detection [MMDD02] and includes:

**Network Topology.** Network topology contains information about the structure of the network, assigned IP addresses, etc. It can be used to better understand the function and role of computers in the network. Possible classifications can be based on the location in the network (e.g., `Internet`, `DMZ`, `Intranet`, `Subnet1`, `Subnet2`, `Subnet3`) or the function of the computer (e.g., `HTTPServer`, `FTPServer`, `DNSServer`, `Workstation`). In the context of machine learning, the network topology can be used to learn rules that make use of *generalized concepts*.

**Alert Semantics and Installed Software.** By alert semantics we mean how an alert is interpreted by the analyst. For example, the analyst knows what type of intrusion the alert refers to (e.g., scan, local attack, remote attack) and the type of system affected (e.g., Linux 2.4.20, Internet Explorer 6.0). Such information can be obtained from proprietary vulnerability databases or public sources such as Bugtraq [Sec04] or ICAT [NIS04].

Typically the alert semantics is correlated with the operating system (or the device type, e.g., `Cisco PIX`) or the software installed to determine whether the system is vulnerable to the reported attack [LWS02] (cf. Level 2 in Section 1.1.1). The result of this process can be used as additional background knowledge to classify alerts.

Note that the information about the installed software and alert semantics can be used even if alert correlation (cf. Level 2 in Section 1.1.1) is not performed, as it allows us to learn rules that make use of generalized concepts such as `OS Linux`, `OS Windows`, etc.

**Alert Context.** Alert context, i.e., other alerts *related* to a given one, is for some alerts (e.g., portscans, password guessing, repetitive exploits attempts) crucial to their classification. In intrusion detection various definitions of alert context are used. Typically, the alert context includes all similar alerts, however the exact definition of similarity varies greatly [DC01, Cup01, VS01].

From the machine-learning perspective, the first two types of background knowledge i.e., network topology and alert semantics, pertain to individual alerts and can be easily propositionalized [LD94] to create additional features describing it. We will call this first type of background knowledge *environmental*.

To illustrate this with an example, assuming that we have the following predicates `alert`, describing the alert, `topology` encoding information about the network topology, `software` describing the software installed on machines and `attack` describing the attacks:

```
topology('10.0.0.1','DMZ'), ...
software('10.0.0.2','IIS'), software('10.0.0.2,'Linux2.6.10'), ...
```

```
attack('ApacheExploit','Apache2.1','remote2user'), ...
vulnerable(X,A) :- alarm(X,...,A), software(X,S), attack(A,S).
```

an alert `alert('10.0.0.1','123.45.67.85',...,'ApacheExploit');` can be propositionalized into a single tuple:

```
alert('10.0.0.1','DMZ','123.45.67.85','Internet',...,
   'ApacheExploit','remote2user','not_vulnerable'...)
```

In contrast, the alert context is much more difficult to represent as it pertains to link information expressed in links between alerts. This type of link mining [Get03] is known as collective classification. We distinguish between two types of this link-based background knowledge, a *non-recursive* and a *recursive link-based* background knowledge (Figure 5.2).

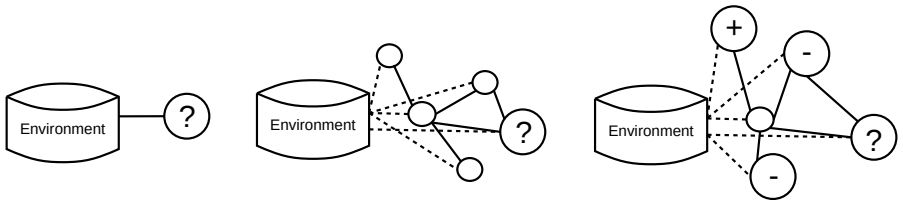



Figure 5.2: Three types of background knowledge for classifying IDS alerts.

In the non-recursive link-based background knowledge the mere existence of the linked alert (and its attributes) affects its classification. In the recursive background knowledge in addition to the existence of linked alerts, their assigned classes affect the classification. To illustrate this with an example, a non-recursive link-based rule may read: "IF a current alert is of type X AND there was previously an alert of type Y AND they were both targeted at the same source and destination IP addresses THEN the current alert is a true positive". In contrast, a recursive link-based rule may read: "IF a current alert originates from S to D AND there was previously an alert A originating from S to D AND A was classified as a true positive THEN the current alert is a true positive".

In the classification of streamed IDS alerts, link-based recursive background knowledge has important implications as to how the system can be used. To illustrate this with an example, suppose a system learned a rule, which bases the classification of the existence of linked alerts classified as positives (like in the example above). In this case, such a rule can achieve high accuracy, however if the first alert is misclassified (false negative), all those "chained" alerts would also be missed. Moreover, such a system would only work, if the alerts are classified strictly sequentially, which means that the analyst can see a predicted label for the $i^{\text{th}}$ alert only after classifying all preceding $i-1$ alerts. In contrast, the non-recursive link-based background knowledge does not impose such limitations.

It should be made clear, that unlike in the classification of, e.g., web pages with linking information or scientific papers with citations, links between IDS alerts are not explicit. Typically, the links in intrusion detection are determined by means of necessarily ad-hoc heuristics (cf. Section 3.5), taking one or more of the following four attributes: source and destination IP addresses, alert type and time. For example with the following three attributes: source and destination IP addresses and alert type, one obtains seven types of links, linking alerts for which up to three of these attributes are equal. These links have an intuitive

interpretation from a domain perspective, e.g., with equal source and destination IP addresses one obtains a group of alerts linking different attacks launched against from the same source at a single destination. Such a representation yields an extremely dense graph, even if due to practical reasons, these links are calculated only for alerts in a close time proximity.

## 5.3 Choosing Machine-Learning Techniques

Until now we have been focusing on the general system architecture and issues specific to intrusion detection. In this section we focus on the machine-learning component in our system. Based on the discussion in Section 1.2 and the proposed system architecture, we can formulate the following requirements for the machine-learning technique:

1. Learn from training examples (alert classification given by the analyst).

2. Build an interpretable classifier, so its correctness and behavior can be verified by the human analyst.

3. Be able to incorporate the background knowledge required.

4. Be efficient enough to perform real-time learning.

5. Be able to assess the confidence of classifications. Confidence is a numerical value attached to the classification, representing how likely it is to be correct.

6. Support cost-sensitive classification and skewed class distributions.

7. Learn incrementally.

### 5.3.1 Learning an Interpretable Classifier from Examples.

The first requirement yields *supervised* machine-learning techniques. These are techniques that can learn from training examples. The requirement for an understandable classifier further limits the range of techniques to *symbolic* learning techniques, which are techniques that present the learned concept in a human readable form (e.g., predictive rules, decision trees, Prolog clauses) [MM02].

We considered the requirement for symbolic representation a fundamental one, and therefore did not even consider any non-symbolic learning methods, like Support Vector Machines (SVMs) [Vap95], Bayesian Networks (BNs) [Pea88] Artificial Neural Networks (ANNs) or Instance-Based Learning (IBL) [AKA91].

However, those non-symbolic representations have two advantages over symbolic learners: First, they are either inherently incremental (e.g., IBL) or can easily be converted in an incremental one (e.g., BN, IBL). Second, these learning methods can also be used to rank instances not only classify them (scoring classifiers). Therefore, an interesting area of future work would be to use those techniques instead, applying further techniques we developed in Chapters 6 and 7 and compare them against our system.

### 5.3.2 Background Knowledge and Efficiency.

The ability to incorporate background knowledge differentiates two big groups of symbolic learners: inductive logic programming and symbolic attribute-value learners. In general, inductive logic programming provides the framework for the use of the background knowledge, represented in the form of logic predicates and first-order rules, whereas attribute-value learners exclusively learn from training examples. Moreover, training examples for attribute-value learners are limited to a fixed number of attributes.

Inductive Logic Programming (ILP) is a research area at the intersection of machine learning and logic programming. The goal of learning in ILP can be defined as follows [LD94]: given a set of training examples $E$ and background knowledge $B$, find a hypothesis $H$ expressed in some concept description language $L$ such that $H$ is complete and consistent with respect to the background knowledge $B$ and examples $E$.

The examples of ILP systems include FOIL [QCJ93], Aleph [Mug95], LINUS [LD94], ICL [vL02] and TILDE [Blo98]. Rules derived from ILP are human readable and capture general dependencies in the data, expressing it using background knowledge. Since rules have human readable form, they can be verified by the analyst and only meaningful rules can be included in the classification. More recently, there have been systems combining ILP with probabilistic frameworks (e.g., [DK03]). One particular example is nFOIL [LKD05] combining naive Bayes with FOIL. For learning, nFOIL uses a covering approach (similar to FOIL), in which features are learned one after another and combined with naive Bayes. However, the search heuristic is based on class conditional likelihood.

The ILP framework can easily handle the background knowledge introduced in Section 5.2, including the alert context as well as arbitrary Prolog clauses. However, one of the main concerns regarding ILP is the size of the hypothesis search space. Since the search space is significantly bigger than in propositional rule learners, the size of problems that can be solved by ILP is accordingly smaller.

On the other hand, attribute-value learners can use a limited form of background knowledge using so-called *feature construction* (also known as propositionalization [LD94]) by creating additional attributes based on values of existing attributes or existing background knowledge. Using propositionalization, the background knowledge can be expressed in the form of *functions* of attribute values or *relations* among attribute values. When learning, these functions and descriptions yield new attributes which are considered in the learning process. If the background knowledge is represented in the form of functions, the value of new attributes is computed as a function of existing attributes. In the relational representation of background knowledge, values of new attributes are `true` or `false` depending on whether attributes of an example satisfy or do not satisfy the relation. Subsequently, tuples with an extended attribute set are used for learning.

Given that most background knowledge for intrusion detection can be converted to additional features using feature construction, and considering the run-time requirements, symbolic attribute-value learners seem to be a good choice for alert classification.

### 5.3.3 Confidence of Classification.

Symbolic attribute-value learners are decision tree learners (e.g., C4.5 [Qui93]) and rule learners (e.g., AQ [Mic69], C4.5rules [Qui93], RIPPER [Coh95]). Both of these techniques can estimate the confidence of a classification based on its performance on training set. How-

ever, it has been shown that rules are much more comprehensible to humans than decision trees [Mit97, Qui93]. Hence, rule learners are particularly advantageous in our context.

Table 5.1: Comparison of properties of different machine learning algorithms with respect to ALAC requirements.

| | | 1. Training Examples | 2. Interpretable | 3. Background knowledge | 4. Efficient | 5. Confidence | 6. Cost-sensitive / Skewed class distributions | 7. Learn incrementally |
|---|---|---|---|---|---|---|---|---|
| Non-symbolic | | | | | | | | |
| | SVM | yes | no | yes[a] | yes | yes | yes[f] | some |
| | Bayesian Networks | yes | partially[b] | yes[a] | no | yes | yes[f] | yes |
| | IBL | yes | no[h] | yes[a] | yes[c] | yes | yes[f] | yes |
| | ANN | yes | no | yes[a] | no | yes | yes[f] | yes |
| Symbolic | | | | | | | | |
| | ILP | yes | yes | yes | no | no[d] | yes[f] | no |
| | Decision Trees | yes | no[g] | yes[a] | yes | yes | yes[f] | some[e] |
| | Rule learners | yes | yes | yes[a] | yes | yes | yes[f] | some |

[a] Background knowledge can be propositionalized. [b] Large automatically learned Bayesian Networks are not interpretable. [c] Classification time in IBL depends on the number of instances. [d] Probabilistic extensions (e.g., nFOIL output probabilities). [e] Some algorithms are incremental (e.g., ID5). [f] Most algorithms either support asymmetric misclassification cost or general methods (e.g., Weighting, MetaCost can be used). [g] Large decision trees are not interpretable. [h] With instance-based learning the interpretation can be based on similar items.

We analyzed the characteristics of available rule learners, shown in Table 5.1, as well as published results from applications in intrusion detection and related domains. We have not found a good and publicly available rule learner that fulfills all our requirements, in particular cost-sensitivity and incremental learning.

## 5.4 Applying RIPPER to ALAC

Amongthe techniques that best fulfill the remaining requirements, we chose RIPPER [Coh95]—a fast and effective rule learner. It has been successfully used in intrusion detection (e.g., on system call sequences and network connection data [Lee99, LFM+02, FLSM00]) as well as related domains and it has proved to produce concise and intuitive rules. As reported by Lee [Lee99], RIPPER rules have two very desirable conditions for intrusion detection: a good generalization accuracy and concise conditions. Another advantage of RIPPER is its effectiveness with noisy datasets.

RIPPER has been well documented in the literature, however, for the sake of a better understanding of the system we will briefly explain how RIPPER works. As shown in Algorithm 4, RIPPER learns a sequence $RS$ of rules $R_i$ in the form:

`if (condition1 and condition2 and ... conditionN) then class`.

A single condition is in the form $A_i = v$ (in the case of categorical attributes) or $A_i \geq \theta$, $A_i \leq \theta$ (in the case of numerical attributes). The rule evaluates to true if and only if all its conditions hold, in which case, the prediction is made and no further rules are evaluated.

In a multi-class setting, RIPPER sorts the classes $C_1, C_2, \cdots, C_n$ in increasing frequency and induces the rules sequentially from the least prevalent class ($SC_1$) to the second to last most prevalent class ($SC_{n-1}$). The most prevalent class $SC_n$ is called a default class, for which no rules are induced. Hence, in the binary case, RIPPER induces rules only for the minority class.

The process of inducing rules for a single class proceeds in two stages: the *building stage* (lines 4–4) and the *optimization stage* (lines 4–4). In the building stage, RIPPER builds the rules in the following two steps: *growing* and *pruning*. In the growing step, rules are greedily "grown" by adding conditions that maximize the information gain [QCJ93]. In the pruning step, rules are pruned using a criterion, which is equivalent to precision. The goal of pruning is to improve both the generalization and the simplicity of the rule. In in the optimization stage, building and pruning is executed on both an initial rule and an empty rule set, with the evaluation done on the entire ruleset. Finally, the best variant of the two is selected for the final ruleset.

Unfortunately, the standard RIPPER algorithm is not cost-sensitive and does not support incremental learning. We used the following methods to circumvent these limitations.

### 5.4.1 Cost-Sensitive and Binary vs. Multi-Class Classification

**Cost-Sensitive Modeling.** In a cost-insensitive world, both types of misclassifications (false negatives and false positives) carry equal weights and hence the performance of a classifier can be evaluated by means of accuracy. However, in the real-world the costs of misclassifications are most often not equal, e.g., missing an intrusion is intuitively more expensive than investigating one false positive. This, together with the fact that cost-sensitive problems are typically skewed, increases an importance of cost-sensitive modeling.

In general, cost-sensitive modeling is a difficult issue [LFM+02] as there can be many costs that need to be taken into account. For example, Fan [Fan01] defines two types of costs in the domain of intrusion detection: the damage cost $DCost$, which characterizes the maximum amount of damage inflicted by an attack and a response cost $RCost$, which is the cost to take action when a potential intrusion is detected. In this case, false negatives would incur the $DCost$ for the given attack, false positives and true positives would incur $DCost$ for the given attack and the wrongly identified attacks would incur both the response cost for the action taken and the damage cost of the missed attack. Moreover, the damage and the response costs are typically not constant and depend on both the attack class and in some cases, the particular instance of an attack (e.g., the damage incurred as a result of an attack against an important server is typically much higher than for the same attacks against a workstation, which can simply be switched off). In addition, Fan showed that certain features used for testing have different costs than others, e.g., analyzing a flag TCP header is much "cheaper" in terms of resources than calculating statistics over an entire TCP flow. The approach proposed by Fan allows to take this fact into account in building ensemble-based learning systems.

**Input**: A set of labeled instances (alerts) $\{(I_i, C_{I_i})\}$
**Result**: A sequence of rules $RS = (R_1, R_2, \ldots, R_n)$ defining a classifier $\mathcal{C}$
`/* sort classes by instance counts */`
1 $(SC_1, SC_2, \ldots, SC_m) = \text{sort}(\{(I_i, C_{I_i})\})$;
2 $RS \leftarrow \emptyset$;
`/* induce rules for all classes, starting from least prevalent */`
3 **foreach** $SC \in (SC_1, SC_2, \ldots, SC_{m-1})$ **do**
  `/* induce ruleset for class SC */`
  `/* one-vs-all classification */`
4  (positiveExamples, negativeExamples) $\leftarrow$ labelOneVsAll($SC, \{(I_i, C_{I_i})\}$);
  `/* split examples for training and pruning */`
5  split(positiveExamples, negativeExamples) into (growPositiveExamples, growNegativeExamples) and (prunePositiveExamples, pruneNegativeExamples);
  `/* building stage - growAndPrune */`
6  **while** *(growPositiveExamples) and (errorRate* $< 50\%$*) and (MDLCondition)* **do**
7  $R_i \leftarrow \emptyset$;
  `/* grow phase on growXXXExamples */`
8  **while** $R_i$ *not perfect* **do**
  `/* use information gain as metric */`
9  greedily add antecedents $N_j$ (in the form $A_j = v$) using information gain: $R_i \leftarrow R_i \cup N_j$;
10  **end**
  `/* prune phase on pruneXXXExamples */`
  `/* use` $(P - N)/(P + N)$ `as metric */`
11  $R'_i \leftarrow$ pruneRule($R_i$);
12  **if** *(errorRate* $< 50\%$*) and (MDLCondition)* **then**
13  $RS \leftarrow RS \cup R'_i$;
14  remove examples covered by rule $R_i$ from growPositiveExamples;
15  **end**
16  **end**
  `/* optimization stage */`
17  **foreach** $R_i \in RS$ **do**
  `/* pruning metric is` $(TP + TN)/(P + N)$ `*/`
18  $R'_i \leftarrow$ growAndPrune($R_i, (R_1, \ldots, R_{i-1}, R_i, R_{i+1} \ldots, R_n)$);
19  $R''_i \leftarrow$ growAndPrune($\emptyset, (R_1, \ldots, R_{i-1}, R_{i+1} \ldots, R_n)$);
  `/* choose a better rule in terms of DL */`
20  $RS \leftarrow (R_1, \ldots, R_{i-1}, \text{chooseBetter}(R'_i, R''_i), R_{i+1} \ldots, R_n)$;
21  **end**
  `/* add uncovered residuals */`
22  **while** *positiveExamples* **do**
23  $RS \leftarrow RS \cup growAndPrune(\emptyset, \emptyset)$;
24  **end**
25 **end**
26 addDefaultRule($SC_m$);

**Algorithm 4**: RIPPER Algorithm [Coh95].

However, while this approach is correct in the formal sense, it has two main problems. First, Fan used boosting methods, in which misclassified instances are "penalized" according to the misclassifications the weak learner made. While taking both *RCost* and *DCost* into account can be easily achieved in this iterative learning method, as a side effect it produces

a number of weak classifiers, which make their rules less interpretable. For example with 200 boosting rounds, each of them building a classifier producing 50 rules, there would be 10000 rules that would need to be investigated. In contrast, our approach focuses on a single classifier. Second, the multi-cost sensitive approach introduces a high number of parameters that wold need to be investigated.

**Multi-Class vs. Binary Classification.** Recall Section 1.3, in which we investigated the job of an intrusion detection analyst and possible classifications of intrusions. Here, we argue that our setup, in which the human analyst analyzes alerts generated by an IDS, can be without loss of functionality considered a binary classification problem.

First, if multiple classes are used, they are not very systematic and, in most cases, describe a nature of a problem, which either is uniquely determined by the type of IDS alerts at question (e.g., an `PORTSCAN alert` if it is a true positive is a "scanning incident"), or cannot be determined with certainty (e.g., the distinction between "policy violation", "unauthorized access attempt" or "suspicious activity" is not always clear). This means that in many cases, such a classifier, knowing that an alert is a true positive, can be built as a second-stage classifier, or should not be built at all. Second, the costs of misclassifying a certain type of an intrusion as another one are extremely hard to determine (e.g., "What is a cost of misclassifying a scanning incident as a policy violation?").

However, the actual cost of misclassifying different types of alerts as non-attacks is not identical. To illustrate this with an example, missing a scanning incident is much less costly than missing a single stealthy attack that installs a root-kit on a machine. However, the problem is that those "cheap" attacks are fairly easy to identify and moreover, they constitute a large numbers of alerts. Conversely stealthy attacks are much more difficult to detect (that is why they are called "stealthy").

This problem of redundancy in the data stream can be solved in two ways: First, alert correlation systems (cf. Level 3 in Section 1.1.1) aim at reducing the redundancy in the alert stream and the number of alerts passed to the analyst. Second, we propose to assign a weight to alerts normalizing them so that the costs of missing different attacks would be identical. This weight should be a function of an alert type (e.g., a `PORTSCAN alert` could have a weight 1 and `IIS exploit` could have a weight 10), so that with $n$ categories of alerts, only $n$ parameters would have to be estimated. In our evaluation, as we wanted to evaluate minimizing the number of parameters that need to be set, we decided not to take this approach and assumed that the cost of missing all attacks is identical.

**Cost-sensitive RIPPER.** As the base version of RIPPER is cost-insensitive, we had to adapt it to support misclassification costs. Among the various methods of making a classification technique cost-sensitive, we focused on those that are not specific to a particular machine-learning technique: Weighting [Tin98] and MetaCost [Dom99]. By changing costs appropriately, these methods can also be used to address the problem of skewed class distributions. These methods produce comparable results, although this can be data dependent [Dom99, MH02]. Experiments not documented here showed that in our context Weighting gives better run-time performance, most likely because of the learning of multiple models by MetaCost. Therefore we chose Weighting for our system.

Weighting resamples the training set so that a standard cost-insensitive learning algorithm builds a classifier that optimizes the misclassification cost. The input parameter for Weighting is a cost matrix, which defines the costs of misclassifications for individual class pairs.

### 5.4.2 Batch-Incremental Learning.

Our is an incremental learning task which is best solved with an incremental learning technique, but can also be solved with a batch learner [GC00]. As we did not have a working implementation of a purely incremental rule learner (e.g., AQ11 [Mic69], AQ11-PM [MM02]) we decided to use a "batch-incremental" approach.

In this approach we add subsequent training examples to the training set and build the classifier using the entire training set as new examples become available. It would not be feasible to rebuild the classifier after each new training example, therefore we handle training examples in batches. The size of such batches can be either constant or dependent on the current performance of the classifier. In our case we focused on the second approach. We evaluate the current classification accuracy and, if it drops below a user-defined threshold, we rebuild the classifier using the entire training set. Note that the weighted accuracy is more suitable than the accuracy measure for cost-sensitive learning. Hence, the parameter controlling "batch-incremental" learning is called the *threshold weighted accuracy*.

The disadvantage of this technique is that the size of the training set grows infinitely during a system's lifetime. This was not a problem in our experiments, as the number of alerts was small. In real environments the number of training examples should thus be limited to a certain time window (or a window of a fixed size). As an alternative, a technique called partial memory [MM02] can be used to reduce the number of training examples.

To summarize, we have not found a publicly available machine-learning technique that addresses all our requirements, in particular cost-sensitivity and incremental learning. Considering the remaining requirements the most suitable techniques are rule learners. Based on desirable properties and successful applications in similar domains, we decided to use RIPPER as our rule-learner. To circumvent its limitations with regard to our requirements, we used a technique called Weighting to implement cost-sensitivity and adjust for skewed class distribution. We also implemented incremental learning as a "batch-incremental", approach, whose batch size dependent on the current classification accuracy.

## 5.5 ALAC Evaluation

In this section we evaluate ALAC, our adaptive alert-classification system presented in Chapter 5. Recall that ALAC can operate in two modes: (i) the recommender mode, in which alerts are classified and forwarded to the analyst and (ii) the agent mode, which in addition to the classification allows for a fraction of alerts to be processed automatically (e.g., false positives can be discarded) without analyst's intervention. In this section we will verify the operation of ALAC in both these modes. In particular, we would like to test the following two hypotheses:

**Hypothesis 5.5.1** *The proposed background knowledge improves the accuracy of alert classification.*

**Hypothesis 5.5.2** *ALAC has acceptable false-positive and false-negative rates in both recommender and agent modes and is useful for intrusion detection.*

For the evaluation, the following remark is in place: While evaluating the performance of any binary classifier (or alert-classification system in particular), we characterize its performance by its confusion matrix and the terms: true positives, false positives, false negatives and true negatives. This causes conflict with terms false positives, true positives commonly used in the domain of intrusion detection and referring to the classification of alerts. In fact, an IDS is a special type of a binary classifier and these names are justified.

To avoid confusion, in the remainder of this dissertation we use terms *false negatives*, *true positives* and *false positives* only in the context of the evaluation of alert-classification systems. From now on we will refer to the original classification of alerts as *true alerts* and *false alerts*.

### 5.5.1 Evaluation Methodology

The evaluation of supervised components of our system is performed in a streamline fashion classifying alerts sequentially as they would be seen by the human analyst. We purposely did not use standard machine learning evaluation techniques using stratified cross-validation, because the streamline method better reflects the way the system would be used in practice.

Note also that cross-validation hinges on the i.i.d. assumption in the data and we know that alerts are not independent and identically distributed. In fact, the system leverages the dependency between alerts by its incremental nature: Misclassified alerts are used to learn an improved alert classifier and classify future similar alerts correctly.

In the evaluation we use ROC analysis to determine the influence of background knowledge and set system parameters. Subsequently, we evaluate false-negative ($fn$) and false-positive ($fp$) rates. We also plot evaluation charts showing how these rates vary during system's runtime (as a function of classified alerts) and evaluate the overall cumulative numbers.

### 5.5.2 Background Knowledge

We decided to focus on the first two types of background knowledge presented in Section 5.1, namely network topology and alert context. Owing to a lack of required information concerning installed software, we decided not to implement matching alert semantics with installed software and vulnerability scanning. This would also be a repetition of the experiments by Lippmann et al. [LWS02].

As discussed in Section 5.2, we used an attribute-value representation of alarms with the background knowledge represented as additional attributes. Specifically, the background knowledge resulted in 25 attributes, which are calculated as follows:

**Environmental Background Knowledge** yielded two attributes for both source and destination IP addresses:

- Classification of IP addresses resulted in an additional attribute for both source and destination IP classifying machines according to their known subnets (e.g., Internet, intranet, DMZ (demilitarized zone, a zone between intranet and the Internet, in which servers are typically located)).
- Classification of hosts resulted in additional attributes indicating the operating system and the host type for known IP addresses.

**Non-recursive Aggregates** resulted in additional attributes with the number of alerts in the following categories (we calculated these aggregates for alerts in time windows of 1 minute, 5 minutes and 30 minutes preceding a given alert):

- alerts with the same source IP address,
- alerts with the same destination IP address,
- alerts with the same source or destination IP address,
- alerts with the same signature.

**Recursive Aggregates** resulted in additional attributes with the number of past alerts classified (predicted classification) as intrusions and the number of alerts from a given IP address classified as intrusions (calculated for alerts in time windows of 1 minute, 5 minutes and 30 minutes).

This choice of background knowledge, which was motivated by heuristics used in alert correlation systems, is necessarily a bit ad hoc and reflects the author's expertise in classifying IDS attacks. As this background knowledge is not especially tailored to training data, it is natural to ask how useful it is for alert classification. We discuss the answer to this question in the following sections.

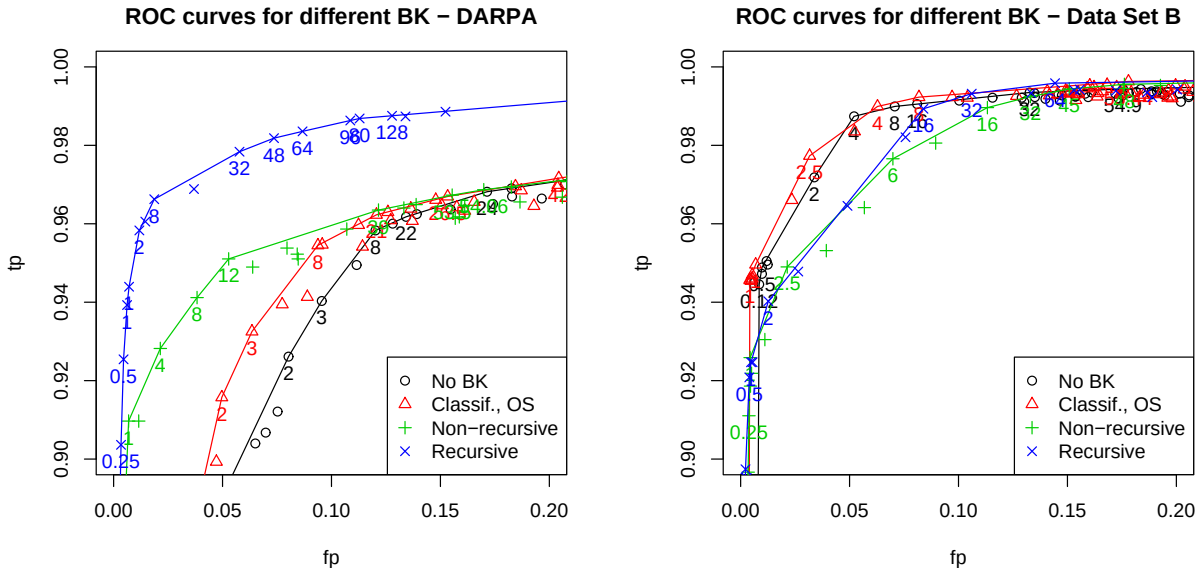
### 5.5.3 Results Obtained with DARPA 1999 Data Set

Our experiments were conducted in two stages. In the first stage we evaluated the performance of the classifier and the influence of adding background knowledge to alerts on the accuracy of classification. The results presented here allowed us to set some parameters in ALAC. In the second stage we evaluated the performance of ALAC in recommender and agent mode.

**Background Knowledge and Setting ALAC Parameters.** First we describe the results of experiments conducted to evaluate background knowledge and to set ALAC parameters. Note that in the experiments we used only the machine learning component of ALAC, namely a RIPPER module, to build classifiers for a stratified 10% sample of the entire dataset. Hereafter we refer to these results as *batch classification*. Recall that the system is evaluated in a "steamline mode" and this is just an estimate allowing us to tune its parameters.

Since the behavior of classifiers depends on the assigned costs, we used ROC (Receiver Operating Characteristic) analysis [PF01] to evaluate the performance of our classifier for different misclassification costs. Figure 5.3a shows the performance of the classifier using data with different amounts of background knowledge. Each curve was plotted by varying the cost ratio for the classifier. Each point in the curve represents results obtained from 10-fold cross validation for a given misclassification cost and type of background knowledge.

As we expected, the classifier with no background knowledge (circle series) performs worse than the classifier with simple classifications of IP addresses and operating systems running on the machines (triangle series) in terms of false positives. Using the background knowledge consisting of the classifications above and aggregates introduced in Section 5.5.2 significantly reduces the false-positive rate and increases the true-positive rate (plus series). Full background knowledge (including recursive background knowledge) performs much better to the reduced one (cross vs. plus series). These observations were also confirmed by additional experiments comparing the results of ALAC with different background knowledge. Hence, in our experiments with ALAC we decided to use full background knowledge.




(a) DARPA1999 Data Set

(b) Data Set B

Figure 5.3: ROC curves for the base classifier used with different types of background knowledge. The fragments represent areas of practical interest (low false-positive rates and high true-positive rates)

ROC curves show the performance of the system under different misclassification costs, but they do not show how the curve was built. Recall from Section 5.4.1 that we use weighting with a weight $w$ to make RIPPER cost sensitive and varied this parameter to obtain a multiple points on the curve. Subsequently, we used this curve to select good parameters of our model.

ALAC is controlled by a number of parameters, which we had to set in order to evaluate its performance. To evaluate the performance of ALAC as an incremental classifier we first selected the parameters of its base classifier.

The performance of the base classifier at various costs and class distributions is depicted by the ROC curve. Using this curve is possible to select an optimal classifier for a certain misclassification cost and class distribution (cf. Section 2.2.4). As these values are not defined for our task, we could not select an optimal classifier using the above method. Therefore we arbitrarily selected a base classifier that gives a good tradeoff between false positives and false negatives, for $w = 50$, which means that positive examples have a weight 50 times bigger than negative instances.

The second parameter is the threshold weighted accuracy ($WA$) for rebuilding the classifier. The value of threshold weighted accuracy should be chosen carefully as it represents a tradeoff between classification accuracy and how frequently the machine learning algorithm is run. We chose the value equal to the accuracy of a classifier in batch mode. Experiments not documented here showed that using higher values increases the learning frequency with no significant improvement in classification accuracy.

We assumed that in real-life scenarios the system would work with an initial model and only use new training examples to modify its model. To simulate this we used 30% of input data to build the initial classifier and the remaining 70% to further train and evaluate the system.

**ALAC in Recommender Mode.** In recommender mode the analyst reviews each alert and corrects ALAC misclassifications. We plotted the number of misclassifications: false-positive rate (Figure 5.4a) and false-negative rate (Figure 5.4b) as a function of processed alerts. Note that we cropped high error rates at the beginning of the run. These are transient effects and we are interested in the asymptotic values.

The resulting overall false-negative rate ($fn = 0.024$) is much higher than the false-negative rate for the batch classification on the entire dataset ($fn = 0.0076$) as shown in Figure 5.3a. At the same time, the overall false-positive rate ($fp = 0.025$) is less than half of the false-positive rate for batch classification ($fp = 0.06$). These differences are expected due to different learning and evaluation methods used, i.e., batch incremental learning vs. 10-fold cross-validation. Note that both ALAC and a batch classifier have a very good classification accuracy and yield comparable results in terms of accuracy.

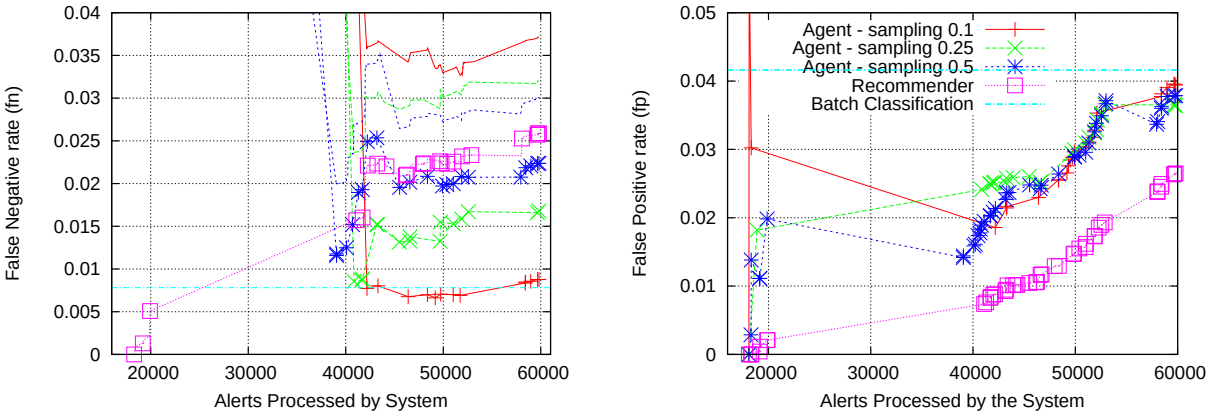



Figure 5.4: False negatives and false positives for ALAC in agent and recommender modes (DARPA1999 dataset, $w = 50$).

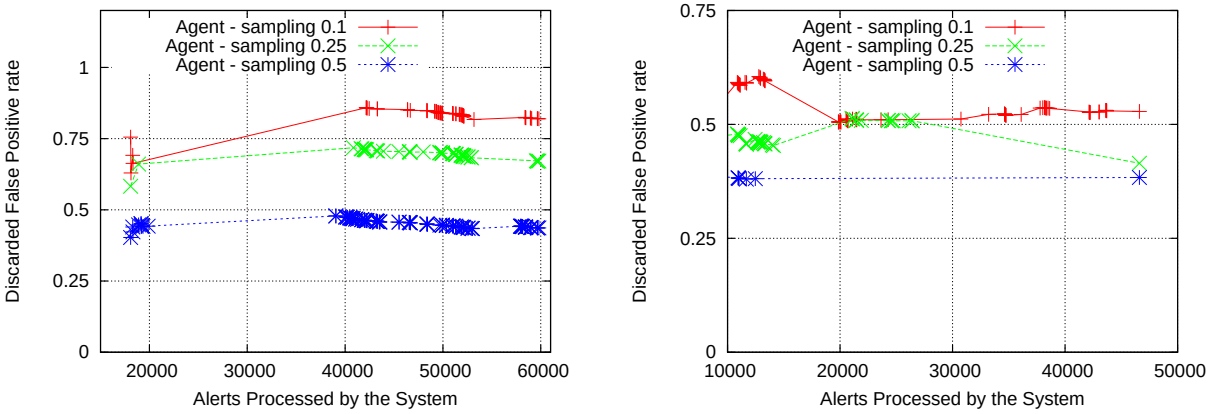
**ALAC in Agent Mode.** In agent mode ALAC processes alerts autonomously based on criteria defined by the analyst, described in Section 5.1. We configured the system to forward to the analyst all alerts classified as true alerts and those false alerts that were classified with low confidence ($confidence < c_{th}$). The system discarded all other alerts, i.e., false alerts classified with high confidence, except for a fraction $s$ of randomly chosen alerts, which were also forwarded to the analyst.

Similarly to the recommender mode, we calculated the number of misclassifications made by the system. We experimented with different values of $c_{th}$ and sampling rates $s$. We then chose $c_{th} = 90\%$ and three sampling rates $s$: 0.1, 0.25 and 0.5. Our experiments show that the sampling rates below 0.1 make the agent misclassify too many alerts and significantly changes the class distribution in the training examples. On the other hand, with sampling rates much higher than 0.5, the system works similarly to recommender mode and is less useful for the analyst.

Notice that there are two types of false negatives in agent mode — the ones corrected by the analyst and the ones the analyst is not aware of because the alerts have been discarded. We plotted the second type of misclassification as mirrored series with no markers in Figure 5.4a. Intuitively with lower sampling rates, the agent will have fewer false negatives of the first

type, in fact missing more alerts. As expected the total number of false negatives is lower with higher sampling rates.

We were surprised to observe that the recommender and the agent have similar false-positive rates ($fp = 0.025$ for both cases) and similar false-negative rates, even with low sampling rates ($fn = 0.026$ for $s = 0.25$ vs. $fn = 0.025$). This seemingly counterintuitive result can be explained if we note that automatic processing of alerts classified as false positives effectively changes the class distribution in training examples in favor of true alerts. As a result the agent performs comparably to the recommender.




(a) DARPA1999 Data Set, $w = 50$

(b) Data set B, $w = 50$

Figure 5.5: Number of alerts processed autonomously by ALAC in agent mode.

As shown in Figure 5.5a, with the sampling rate of 0.25, more than 60% of false alerts were processed and discarded by ALAC. At the same time the number of unnoticed false negatives is half the number of mistakes for recommender mode. Our experiments show that the system is useful for intrusion detection analysts as it significantly reduces the number of false positives, without making too many mistakes.

### 5.5.4 Results Obtained with Data Set B

We used the second dataset as an independent validation of the system. To avoid "fitting the model to the data" we used the same set of parameters as for the first dataset. However, a ROC curve in Figure 5.3b shows that the classifier achieves much higher true-positive rate and much lower false-negative rate than for the first dataset, which means that Data Set B is easier to classify. The likely explanation of this fact is that Data Set B contains fewer intrusions and more redundancy than the first data set. Moreover, we observed that the actual performance of the system was much better than compared with the result obtained with batch classification on a 10% stratified sample. Hence we built another ROC curve on the entire dataset for comparison as batch classification.

**Background Knowledge and Setting ALAC Parameters.** Results with ROC analysis (Figure 5.3b) show that the classifier correctly classifies most of the examples, and adding background knowledge has little effect on classification. To have the same conditions as with the first dataset, we nonetheless decided to use the full background knowledge. We

also noticed that $w = 50$ is not the optimal value for this dataset as it results in a high false-positive rate ($fn = 0.002, fp = 0.05$).

We observed that ALAC, when run with 30% of the alerts as an initial classifier, classified the remaining alerts with very few learning runs. Therefore, to demonstrate its incremental learning capabilities, we decided to lower the initial amount of training data from 30% to 5% of all the alerts.

**ALAC in Recommender Mode.** Figure 5.6 shows that in recommender mode the system has a much lower overall false-negative rate ($fn = 0.0045$) and a higher overall false-positive rate ($fp = 0.10$) than for DARPA 1999 Data Set, which is comparable to the results of the classification in batch mode. We also observed that the learning only took place for approximately the first 30% of the entire dataset and the classifier classified the remaining alerts with no additional learning. This phenomena can also be explained by the fact that Data Set B contains more regularities and the classifier is easier to build.

This is different in the case of the DARPA1999 dataset, where the classifier was frequently rebuilt in the last 30% of the data. For DARPA1999 Data Set the behavior of ALAC is explained by the fact that most of the intrusions actually took place in the last two weeks of the experiment.

**ALAC in Agent Mode.** In agent mode we obtained results similar to those in recommender mode, with a great number of alerts being processed autonomously by the system ($fn = 0.0065, fp = 0.13$). As shown in Figure 5.5b, with the sampling rate of 0.25, more than 38% of all false positives were processed by the agent. At the same time the actual number of unnoticed false negatives is one third smaller than the number of false negatives in recommender mode. This confirms the usefulness of the system tested with an independent dataset.

Similarly to observation in Section 5.5.3 with lower sampling rates, the agent will have seemingly fewer false negatives, in fact missing more alerts. As expected the total number of false negatives is lower with higher sampling rates. This effect is not as clearly visible as with DARPA1999 Data Set.

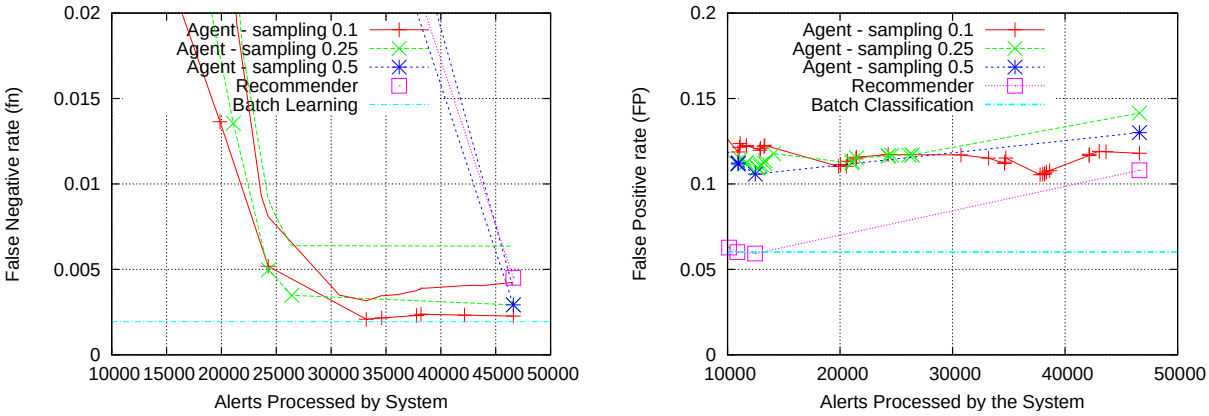



Figure 5.6: False negatives and false positives for ALAC in agent and recommender modes (Data Set B, $w = 50$).

### 5.5.5 Understanding the Rules

One requirement of our system was that the rules can be reviewed by the analyst so that their correctness can be verified. The rules built by RIPPER are generally human interpretable and thus can be reviewed by the analyst. Here is a representative example of two rules used by ALAC:

```
(cnt_intr_w1 <= 0) and (cnt_sign_w3 >= 1) and (cnt_sign_w1 >= 1)
        and (cnt_dstIP_w1 >= 1) => class=FALSE
(cnt_srcIP_w3 <= 6) and (cnt_int_w2 <= 0) and (cnt_ip_w2 >= 2)
         and (sign = ICMP PING NMAP) => class=FALSE
```

The first rule reads as follows: If a number of alerts classified as intrusions in the last minute (window `w1`) equals zero and there have been other alerts triggered by a given signature and targeted at the same IP address as the current alert, then the alert should be classified as false positive. The second rule says that, if the number of `NMAP PING` alerts originating from the same IP address is less than six in the last 30 minutes (window `w3`), there have been no intrusions in the last 5 minutes (window `w2`) and there has been at least 1 alert with identical source or destination IP address, then the current alert is false positive.

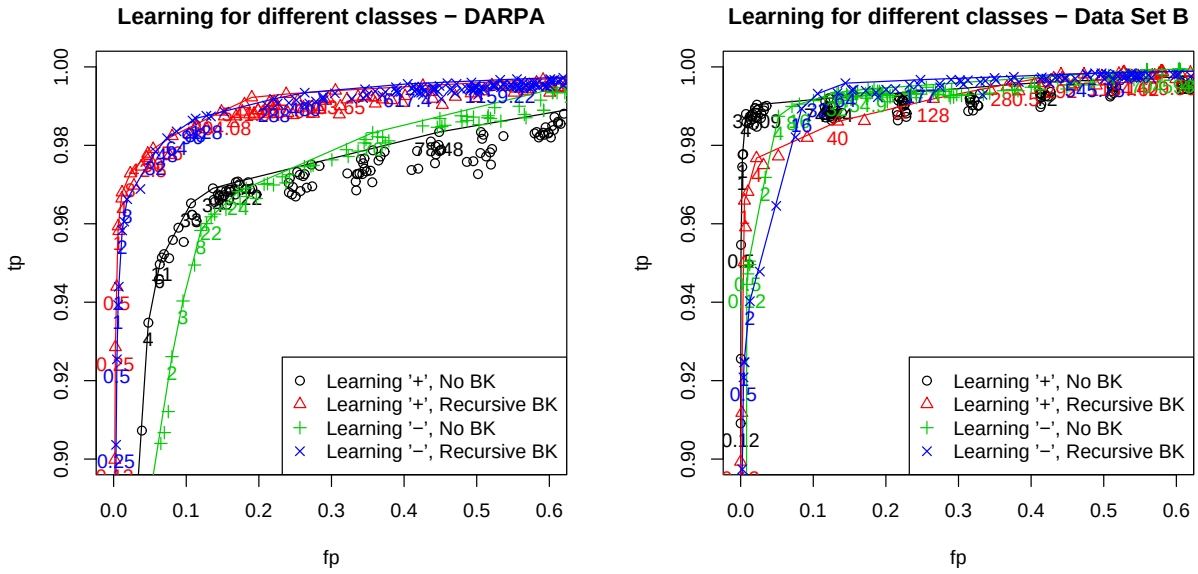
These rules are intuitively appealing: If there have been similar alerts recently and they were all false alerts, then the current alert is also a false alert. The second rule says that if the number of `NMAP PING` alerts is small and there has not been any intrusions recently, then the alert is a false alert.

**RIPPER+ and RIPPER−** Recall from Section 5.4 that RIPPER in the unordered ruleset mode induces rules only for the least prevalent class, which depending on the costs of misclassifying instances can be either a positive class or negative class. However, this inducing of the class has important implications how interpretable the rules are and also on the misclassification costs.

To analyze this effect on the classification accuracy we performed experiments in which we control the class, for which RIPPER induces rules: RIPPER+ is an algorithm, which induces rules describing true alerts and RIPPER− induces rules describing false alerts. The results are shown in Figure 5.7

Note that for the curves for lower misclassification costs, RIPPER+ performs better and for higher ones, RIPPER− performs better. These results are intuitive, as learning an "anomaly model" (RIPPER−) gives a better performance when false negatives are mode expensive. However, we noticed that rules generated by RIPPER− tend to be overly general and difficult to interpret (e.g., `(col2 = ip_inside) and (col3 = other) => class=FALSE`), saying that any alert generated by an internal machine with the operating system classified as `other` is a false positive).

We discovered that generating statistics showing the distribution of values of remaining attributes (in particular source and destination IP addresses and the signature type) can greatly improve the comprehensibility of the rules, in particular subsequent rules, induced after many alerts have been removed. Hence, a suggestion for the future work is to generate sets describing the values of these three attributes and add them to rules (RIPPER cannot learn such rules as it does not support set-valued attributes). This would make those rules




(a) DARPA1999 Data Set

(b) Data Set B

Figure 5.7: ROC performance for algorithms inducing rules for different classes: "+" and "−".

more specific and also decrease the chances for alerts to be removed. Another possible modification increasing the comprehensibility of RIPPER rules is to bias its search heuristics so that it is more likely to select one (or more) of these key attributes in the rule being built.

### 5.5.6 Conclusions

In our evaluation of ALAC carried on two datasets, DARPA 1999 Data Set and the Data Set B we validated the hypothesis we stated at the beginning of the section.

We showed that background knowledge is useful for alert classification. The results were particularly clear for the DARPA 1999 data set. For the real-world dataset, adding background knowledge had little impact on the classification accuracy. The second set was much easier to classify, even with no background knowledge. Hence, we did not expect improvement from background knowledge in this case. This confirms the first hypothesis stated in the beginning of the evaluation section.

We also showed that the system is useful in recommender mode, where it adaptively learns the classification from the analyst. For both datasets we obtained false-negative and false-positive rates comparable to batch classification. Note that in recommender mode all system misclassifications would have been corrected by the analyst.

In addition, we found that our system is useful in agent mode, where some alerts are autonomously processed (e.g., false positives classified with high confidence are discarded). More importantly, for both datasets the false-negative rate of our system is comparable to that in the recommender mode. With synthetic data the system reduced the number of false positives by 60% with a false-negative rate below 0.026 (half of these alerts would have been shown to the analyst) and a false-positive rate 0.025. Experiments with real data, run with the same settings show that the number of false positives has been reduced by 38% with

a false-negative rate 0.006 (half of these alerts would have been shown to the analyst) and a false-positive rate 0.13. Although these figures depend on the input data and the chosen classifier, they nicely show the expected performance estimates for our application. This confirms the second hypothesis we were validating in this section, showing that ALAC in both modes is useful in intrusion detection.

The system has a few numeric parameters that influence its performance and should be adjusted depending on the input data including the weighting parameter $w$ choosing the point of operation on the ROC curve and the confidence threshold $c_{th}$ which determines whether an alert is processed autonomously or not. ALAC+ using abstaining classifiers which we will introduce in Chapter 7 addresses these two concerns and tries to optimize the misclassification cost of the system.

## 5.6 Summary

In this chapter, we presented a novel concept of building an adaptive alert classifier based on an intrusion detection analyst's feedback using machine-learning techniques. We discussed the issues of human feedback and background knowledge, and reviewed machine-learning techniques suitable for alert classification. We selected the most suitable technique for our purposes.

In the evaluation section we used ROC analysis to show that background knowledge improves the alert classification and used it to set parameters of our system. We also validated the system in both agent and recommender modes on one synthetic and one real dataset and showed that the system has acceptable error rates. Moreover, in agent mode the system automatically processed up to 60% of false positives, reducing the analyst workload by this factor. This shows the potential of the system and makes it attractive for this application domain.

# Chapter 6

# Abstaining Classifiers using ROC Analysis

In this chapter we introduce abstaining classifiers and their three evaluation models, cost-based, bounded-abstention and bounded-improvement models and evaluate them on a variety of datasets. While this can be viewed as an independent contribution to machine learning, in the subsequent chapters, we show that the above methods are particularly advantageous for IDS alert classification.

## 6.1 Introduction

In recent years, there has been much work on ROC analysis [Faw03, FW05, PF98]. An advantage of ROC analysis in machine learning is that it offers a flexible and robust framework for evaluating classifier performance with varying class distributions or misclassification costs [Faw03].

*Abstaining classifiers* are classifiers that can refrain from classification in certain cases and are analogous to a human expert, who in certain cases can say "I don't know". In many domains (e.g., medical diagnosis) such experts are preferred to those who always make a decision and sometimes make mistakes.

Machine learning has frequently used abstaining classifiers directly [Cho70, FHO04, PMAS94, Tor00] and also as parts of other techniques [FFHO04, GL00, LC94]. Similarly to the human expert analogy, the motivation is that when such a classifier makes a decision it will perform better than a normal classifier. However, as these classifiers are not directly comparable, the comparison is often limited to coverage–accuracy graphs [FHO04, PMAS94].

In this chapter, we apply ROC analysis to build an abstaining classifier that minimizes the misclassification cost. Our method is based solely on ROC curves and is independent of the classifiers used. In particular we do not require that the underlying classifier gives calibrated probabilities, which is not always easy [CG04, ZE01]. We look at a particular type of abstaining binary classifiers, i.e., metaclassifiers constructed from two classifiers described by a single ROC curve, and show how to select such abstaining classifiers optimally. In our previous work [Pie05], we proposed and evaluated *three different optimization criteria* that are commonly encountered in practical applications. Whereas selecting an optimal classifier in the first model, namely the cost-based model, was based only on the slope of a ROC curve (as in the case of a single optimal classifier), we showed that such a simple algorithm not

possible in the remaining two models. In this work, we present our previous results and further investigate the two models, for which we develop an efficient algorithm to select the optimal classifier.

The contribution of the chapter is twofold: We define an abstaining binary classifier built as a metaclassifier and propose three models of practical relevance: the cost-based model (an extension of [Tor00]), the bounded-abstention model, and the bounded-improvement model. These models define the optimization criteria and allow us to compare binary and abstaining classifiers. Second, we propose efficient algorithms to practically build an optimal abstaining classifier in each of these models using ROC analysis, and evaluate our method on a variety of UCI KDD datasets.

The chapter is organized as follows: Section 6.2 gives background and introduces the notation used in this chapter. In Section 6.3 we introduce the concept of ROC-optimal abstaining classifiers in three models. Section 6.4 discusses the first model, the cost-based model, and Section 6.5 proposes algorithms for efficient construction of abstaining classifiers in the other two models: bounded-abstention and bounded-improvement models. Section 6.6 discusses the evaluation methodology and presents the experimental results. In Section 6.7 we discuss alternative representation to ROC curves and how they can be used with our method. Section 6.8 presents related work. Finally, Section 6.9 contains the conclusions and future work.

## 6.2 Background

Recall in Section 2.2.3 we introduced a binary classifier $\mathcal{C}$, a ranker $\mathcal{R}$, a confusion matrix $C$ describing its performance (cf. Table 2.2a) and a cost matrix $Co$ (cf. Table 2.2b), in a linear cost model, in which false positives are $CR$ times more expensive than false negatives. Finally, we also defined a misclassification cost $rc$ calculated per classified example, determining the average cost incurred while classifying an instance.

In Section 2.2.4 we introduced ROC analysis, including ROC curves, ROCCH (ROC Convex Hull) curves and (2.5), the known equation of iso-performance lines. In the remainder of this chapter we will build upon these concepts to introduce cost-optimal abstaining classifiers.

## 6.3 ROC-Optimal Abstaining Classifier

*Abstaining binary classifiers* $\mathcal{A}$ (or abstaining classifiers for short) are classifiers that in certain situations abstain from classification. We denote this as assigning a third class "?". Such nonclassified instances can be classified using another (possibly more reliable, but more expensive) classifier (e.g., a multi-stage classification system [Sen05]) or a human domain expert.

Our method builds an *ROC-optimal* abstaining classifier as a metaclassifier using a ROC curve and the binary classifiers used to construct it. A ROC-optimal classifier is defined as described in Section 2.2.4. The method constructs an abstaining metaclassifier $\mathcal{A}_{\alpha,\beta}$ using two binary classifiers $\mathcal{C}_\alpha$ and $\mathcal{C}_\beta$ as follows:

$$\mathcal{A}_{\alpha,\beta}(x) = \begin{cases} + & \text{if} \quad \mathcal{C}_\alpha(x) = \text{“+”} \wedge \mathcal{C}_\beta(x) = \text{“+”} \\ ? & \text{if} \quad \mathcal{C}_\alpha(x) = \text{“−”} \wedge \mathcal{C}_\beta(x) = \text{“+”} \\ - & \text{if} \quad \mathcal{C}_\beta(x) = \text{“−”} \wedge \mathcal{C}_\alpha(x) = \text{“−”} \end{cases} . \tag{6.1}$$

Each classifier has a corresponding confusion matrix, $(TP_\alpha, FN_\alpha, FP_\alpha, TN_\alpha)$ and $(TP_\beta, FN_\beta, FP_\beta, TN_\beta)$, which will be used in the next sections. Classifiers $\mathcal{C}_\alpha$ and $\mathcal{C}_\beta$ belong to a family of classifiers $\mathcal{C}_\tau$, described by a single ROC curve with $FP_\alpha \leq FP_\beta$ (as shown in Figure 6.1).

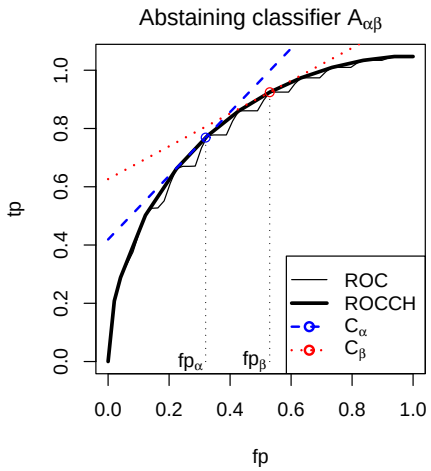



Figure 6.1: Abstaining classifier $\mathcal{A}_{\alpha,\beta}$ constructed using two classifiers $\mathcal{C}_\alpha$ and $\mathcal{C}_\beta$.

Our method is independent of the machine-learning technique used. However, we require that for any two points $(fp_\alpha, tp_\alpha)$, $(fp_\beta, tp_\beta)$ on the ROC curve, with $fp_\alpha \leq fp_\beta$, corresponding to $\mathcal{C}_\alpha$ and $\mathcal{C}_\beta$, the following conditions hold:

$$\begin{aligned} \forall i : (\mathcal{C}_\alpha(i) = \text{“+”} \implies \mathcal{C}_\beta(i) = \text{“+”}) \wedge \\ (\mathcal{C}_\beta(i) = \text{“−”} \implies \mathcal{C}_\alpha(i) = \text{“−”}) \ . \end{aligned} \tag{6.2}$$

Conditions (6.2) (we will hereafter refer to them as the *watermark condition*) are the ones used in [FW05]. These are met in particular if the ROC curve and $\mathcal{C}_\alpha$ and $\mathcal{C}_\beta$ are built from a single ranker $\mathcal{R}$ (e.g., a Bayesian classifier) with two threshold values $\alpha$ and $\beta$ ($\alpha \geq \beta$). The advantage is that for such a classifier, a simple and efficient algorithm exists for constructing a ROC curve [Faw03]. For arbitrary classifiers (e.g., rule learners), (6.2) is generally violated. However, we observed that the fraction of instances with $\mathcal{C}_\alpha(i) = \text{“+”} \wedge \mathcal{C}_\beta(i) = \text{“−”}$ typically is small. As this is an interesting class of applications, we plan to elaborate on it as a future work item.

Given a particular cost matrix and class distribution $N/P$, the optimal binary classifier can easily be chosen as one that minimizes the misclassification cost (2.3). However, no such notion exists for abstaining classifiers, as the tradeoff between nonclassified instances and the cost is undefined. Therefore, we proposed and investigated [Pie05] three different criteria and models of optimization $\mathcal{E}$: the cost-based, the bounded-abstention and the bounded-improvement model, which we discuss in the following sections. Models $\mathcal{E}$ determine how nonclassified instances are accounted in the misclassification cost and other boundary conditions. We formulate our goals as:

Table 6.1: Cost matrix *Co* for an abstaining classifier. Columns and rows are the same as in Table 2.1. The third column denotes the abstention class.

| A \ C | + | − | ? |
|---|---|---|---|
| + | 0 | $c_{12}$ | $c_{13}$ |
| − | $c_{21}$ | 0 | $c_{23}$ |

| | |
|---|---|
| **Given** | – A ROC curve generated using classifiers $\mathcal{C}_\tau$, such that the watermark condition (6.2) holds.<br>– A Cost matrix *Co*.<br>– Evaluation model $\mathcal{E}$. |
| **Find** | A classifier $\mathcal{A}_{\alpha,\beta}$ such that $\mathcal{A}_{\alpha,\beta}$ is optimal in model $\mathcal{E}$. |

**Cost-Based Model** In the first evaluation model $\mathcal{E}_{CB}$, a so-called cost-based model, we use an extended $2 \times 3$ cost matrix with the third column representing the cost associated with abstaining from classifying an instance. This cost may or may not be dependent on the true class of the instance.

**Bounded Models** To address the shortcomings of the cost-based model and allow for situations in which the extended cost matrix is not available, we propose two models $\mathcal{E}_{BA}$ and $\mathcal{E}_{BI}$ that use a standard $2 \times 2$ cost matrix and calculate the misclassification cost per instance actually classified. The motivation is to calculate the cost only for instances the classifier attempts to classify.

In such a setup, a classifier randomly abstaining from classification would have the same misclassification cost as a normal classifier. Conversely, classifiers abstaining from classifying only for *difficult* instances may have a significantly lower misclassification cost.

However, such system is underspecified as we do not know how to trade the misclassification cost for the number of nonclassified instances. To address this, we propose two bounded evaluation models having boundary conditions:

**Bounded-abstention model $\mathcal{E}_{BA}$,** where the system abstains for not more than a fraction $k_{\max}$ of instances and has the lowest misclassification cost,

**Bounded-improvement model $\mathcal{E}_{BI}$,** where the system has a misclassification cost not higher than $rc_{\max}$ and abstains for the lowest number of instances.

## 6.4 Cost-Based Model

In this model, we compare the misclassification cost, $rc_{CB}$, incurred by a binary and an abstaining classifier. We use an extended $2 \times 3$ cost matrix, with the third column representing the cost associated with classifying an instance as "?". Note that this cost can be different for instances belonging to different classes, which extends the cost matrix introduced in [Tor00].

| | |
|---|---|
| **Given** | – ROC curve generated using classifiers such that the watermark condition (6.2) holds<br>– $2 \times 3$ cost matrix *Co* |
| **Find** | Classifier $\mathcal{A}_{\alpha,\beta}$ such that it minimizes misclassification cost $rc_{CB}$ |

Having defined the cost matrix, we use a similar approach as in Section 2.2.4 for finding the optimal classifier. Note that the classifications made by $\mathcal{C}_\alpha$ and $\mathcal{C}_\beta$ are not independent. Equation (6.2) implies that false positives for $\mathcal{C}_\alpha$ imply false positives for $\mathcal{C}_\beta$. Similarly, false negatives for $\mathcal{C}_\beta$ imply false negatives for $\mathcal{C}_\alpha$, and we can thus formulate (6.3). The misclassification cost $rc_{CB}$ is defined using a $2 \times 3$ cost matrix similar to (2.3), with the denominator equal to the total number of instances.

$$\begin{aligned} rc_{CB} \cdot (N+P) &= \underbrace{(FP_\beta - FP_\alpha)\, c_{23}}_{\mathcal{C}_\alpha,\, \mathcal{C}_\beta \text{ disagree, } -} + \underbrace{(FN_\alpha - FN_\beta)\, c_{13}}_{\mathcal{C}_\alpha,\, \mathcal{C}_\beta \text{ disagree, } +} + \underbrace{FP_\alpha \cdot c_{21}}_{FP \text{ for both}} + \underbrace{FN_\beta \cdot c_{12}}_{FN \text{ for both}} \\ &= (FN_\alpha \cdot c_{13} + FP_\alpha \cdot (c_{21} - c_{23}) + FN_\beta \cdot (c_{12} - c_{13}) + FP_\beta \cdot c_{23}) \\ &= P\left(1 - f_{ROC}\left(\frac{FP_\alpha}{N}\right)\right) c_{13} + FP_\alpha\ (c_{21} - c_{23}) \\ &\qquad + P\left(1 - f_{ROC}\left(\frac{FP_\beta}{N}\right)\right)\ (c_{12} - c_{13}) + FP_\beta \cdot c_{23} \end{aligned} \tag{6.3}$$

We rewrite (6.3) as a function of only two variables $FP_\alpha$ and $FP_\beta$, so that to find the local minimum we calculate partial derivatives for these variables

$$\begin{aligned} \frac{\partial rc_{CB}}{\partial FP_\alpha} \cdot (N+P) &= -\frac{P}{N} f'_{ROC}\left(\frac{FP_\alpha}{N}\right) c_{13} + c_{21} - c_{23} \\ \frac{\partial rc_{CB}}{\partial FP_\beta} \cdot (N+P) &= -\frac{P}{N} f'_{ROC}\left(\frac{FP_\beta}{N}\right) (c_{12} - c_{13}) + c_{23}\ , \end{aligned} \tag{6.4}$$

set the derivatives to zero and making sure that the function has a local extremum. After replacing $FP_\alpha$ and $FP_\beta$ with the corresponding rates $fp_\alpha$ and $fp_\beta$, we obtain the final result:

$$\begin{aligned} f'_{ROC}(fp^*_\beta) &= \frac{c_{23}}{c_{12} - c_{13}} \frac{N}{P} \\ f'_{ROC}(fp^*_\alpha) &= \frac{c_{21} - c_{23}}{c_{13}} \frac{N}{P}\ , \end{aligned} \tag{6.5}$$

which, similarly to (2.5), allows us to find $fp^*_\alpha$ and $fp^*_\beta$, and the corresponding classifiers $\mathcal{C}_\alpha$ and $\mathcal{C}_\beta$.

This derivation is valid only for metaclassifiers (6.1) with (6.2), which implies $fp^*_\alpha \le fp^*_\beta$ and $f_{ROC}(fp^*_\alpha) \le f_{ROC}(fp^*_\beta)$. As an ROCCH is increasing and convex, its first derivative is nonnegative and nonincreasing, and we obtain $f'_{ROC}(fp^*_\alpha) \ge f'_{ROC}(fp^*_\beta) \ge 0$. Using the $2 \times 3$ cost matrix these conditions can be rewritten as:

$$(c_{21} \ge c_{23}) \wedge (c_{12} > c_{13}) \wedge (c_{21}c_{12} \ge c_{21}c_{13} + c_{23}c_{12})\ . \tag{6.6}$$

If condition (6.6) is not met, our derivation is not valid; however, the solution is trivial in this case.

**Theorem 8** *If (6.6) is not met, the classifier minimizing the misclassification cost is a trivial binary classifier, namely, a single classifier described by (2.5).*

**Proof** Calculating (6.4) we obtain that if the rightmost part of (6.6) does not hold, $\partial rc_{CB}/\partial fp_\alpha$ is negative for all values $f'_{ROC}(fp^*_\alpha) \le f'_{ROC}(fp^*) = c_{21}/c_{12} \cdot N/P$ and, similarly, $\partial rc_{CB}/\partial fp_\beta$

is positive for all values $f'_{ROC}(fp^*_\beta) \geq f'_{ROC}(fp^*) = c_{21}/c_{12} \cdot N/P$. This, together with the basic assumption $fp_\alpha \leq fp_\beta$ and the properties of the ROCCH, implies that $fp^*_\alpha = fp^*_\beta$, which means that the optimal abstaining classifier is binary classifier. Such a classifier is the binary classifier described by (2.5). □

Equation (6.6) allows us to determine whether for a given $2 \times 3$ cost matrix *Co* a trivial abstaining classifier minimizing $rc_{CB}$ exists, but gives little guidance to setting parameters in this matrix. For this we consider two interesting cases: (i) a symmetric case $c_{13} = c_{23}$ and (ii) a proportional case $c_{23}/c_{13} = c_{21}/c_{12}$.

The first case has some misclassification cost *CR* with identical costs of classifying instances as "?". This case typically occurs when, for example, the cost incurred by the human expert to investigate such instances is irrespective of their true class. In this case, (6.6) simplifies to the harmonic mean of two misclassification costs: $c_{13} = c_{23} \leq c_{21}c_{12}/(c_{21} + c_{12})$. The second case yields the condition $c_{13} \leq c_{12}/2$ (equivalent to $c_{23} \leq c_{21}/2$). This case occurs if the cost of classifying an event as the third class is proportional to the misclassification cost. These simplified equations allow a meaningful adjustment of parameters $c_{13}$ and $c_{23}$ for abstaining classifiers.

To summarize, the ROC-optimal abstaining classifier in a cost-based model can be found using (6.5) if (6.6) (or the special cases discussed below) holds on a given cost matrix. In the opposite case, our derivation is not valid; however the ROC-optimal classifier is a trivial binary classifier ($\mathcal{C}_\alpha = \mathcal{C}_\beta$).

## 6.5 Bounded Models

In the simulations using a cost-based model, we noticed that the cost matrix and in particular cost values $c_{13}$, $c_{23}$ have a large impact on the number of instances classified as"?". Therefore we think that, while the cost-based model can be used in domains where the $2 \times 3$ cost matrix is *explicitly given*, it may be *difficult to apply in other domains* where parameters $c_{13}$, $c_{23}$ would have to be estimated. Therefore, in the bounded models, we use a standard $2 \times 2$ cost matrix and calculate the misclassification cost only per instances classified.

Using a standard cost equation (2.3), with the denominator $TP + FP + FN + TN = (1-k)(N+P)$, where $k$ is the fraction of nonclassified instances, we obtain the following set of equations:

$$\begin{aligned} rc_B &= \frac{1}{(1-k)(N+P)} \left(FP_\alpha \cdot c_{21} + FN_\beta \cdot c_{12}\right) \\ k &= \frac{1}{N+P} \left((FP_\beta - FP_\alpha) + (FN_\alpha - FN_\beta)\right) \quad , \end{aligned} \tag{6.7}$$

determining the relationship between the fraction of classified instances $k$ and the misclassification cost $rc_B$ as a function of classifiers $\mathcal{C}_\alpha$ and $\mathcal{C}_\beta$. Similarly to the cost-based, model we can rewrite these equations as functions of $fp_\alpha$ and $fp_\beta$:

$$\begin{aligned} rc_B &= \frac{1}{(1-k)(N+P)} \left(N fp_\alpha \cdot c_{21} + P\left(1 - f_{ROC}(fp_\beta)\right) \cdot c_{12}\right) \\ k &= \frac{1}{N+P} \left(N\left(fp_\beta - fp_\alpha\right) + P\left(f_{ROC}\left(fp_\beta\right) - f_{ROC}\left(fp_\alpha\right)\right)\right) \quad . \end{aligned} \tag{6.8}$$

By putting boundary constraints on $k$ and $rc_B$ and trying to optimize the other variable, $rc_B$ and $k$ respectively, we create two interesting evaluation models we discuss in the following sections.

### 6.5.1 Bounded-Abstention Model

By limiting $k$ to some threshold value $k_{\max}$ ($k \leq k_{\max}$), we obtain a model the bounded-abstention model, in which the classifier can abstain for at most a fraction $k$ of instances. In this case the optimization criteria is that the classifier should have the lowest misclassification cost $rc_B$ (hereafter referred to as $rc_{BA}$).

This has multiple real-life applications, e.g., in situations where nonclassified instances will be handled by a classifier with limited processing speed (e.g., a human expert). In such cases, assuming a constant flow of instances with speed $c$ and a constant manual processing speed $m$, $m < c$, we obtain $k_{\max} = m/c$.

| | |
|---|---|
| **Given** | – ROC curve generated using classifiers such that (6.2) holds<br>– $2 \times 2$ cost matrix $Co$<br>– Fraction $k$ |
| **Find** | Classifier $\mathcal{A}_{\alpha,\beta}$ such that the classifier abstains for not more than a fraction of $k$ instances and has the lowest cost $rc_{BA}$. |

Unfortunately, unlike the cost-based model, the set of equations (6.8) for a bounded-abstention model does not have an algebraic solution in the general case, and in [Pie05] we used general numerical optimization methods to solve it. Here we present an algorithm finding the solution that is extremely efficient for piecewise-linear ROCCH curves.

To find the solution for the bounded improvement model, we will use the constrained optimization method for the function of two variables. We will proceed in the following three steps: First, we will present the algorithm for a smooth convex down ROC curve having a derivative in $[0, 1]$ and assuming that *exactly* a fraction $k_{\max}$ of instances remains unclassified. Second, we will show under which conditions the optimal classifier can abstain for less than a fraction $k_{\max}$ of instances. Finally, we will extend the method to the piecewise linear ROCCH curves.

#### Optimal Classifier for a Smooth Convex Down Curve

Our minimization task can be defined as follows: Find the minimum of $rc_{BA}(fp_\alpha, fp_\beta)$, subject to condition $k^*(fp_\alpha, fp_\beta) = k(fp_\alpha, fp_\beta) - k_{\max} = 0$.

For this, we will use the Lagrange method, which is a method for constrained optimization of a differentiable function under equality constrains (see e.g., [Ste92, Wol06] for a more complete coverage). Very shortly, given differentiable functions $F$ and $G$ the goal is to find the minimum of $F(\mathbf{X})$ given the constraint $G(\mathbf{X}) = 0$. The method calculates the so-called Lagrange multipliers $\lambda$ such that

$$\nabla F(\mathbf{X}) = \lambda \nabla G(\mathbf{X}) \ . \tag{6.9}$$

By solving (6.9) for $\mathbf{X}$ and $\lambda$ and given the constraint $G(\mathbf{X}) = 0$ we obtain the desired solution.

In our case we have functions of two variables ($fp_\alpha$ and $fp_\beta$) and in this two-dimensional case (6.9) can has an interpretation that vectors $\nabla rc_{BA}$ and $\nabla k$ have the same direction. This is equivalent to $\nabla rc_{BA} \times \nabla k = \mathbf{0}$ and

$$\frac{\partial rc_{BA}}{\partial fp_\alpha}\frac{\partial k}{\partial fp_\beta} - \frac{\partial k}{\partial fp_\alpha}\frac{\partial rc_{BA}}{\partial fp_\beta} = 0 \ . \tag{6.10}$$

The second condition is that $k^*(fp_\alpha, fp_\beta) = k(fp_\alpha, fp_\beta) - k_{\max} = 0$.

Calculating the derivatives and marking the denominator $D$ of $rc_{BI}$ as

$$D = N(\underbrace{fp_\alpha - fp_\beta + 1}_{\geq 0}) + P(\underbrace{f_{ROC}(fp_\alpha) - f_{ROC}(fp_\beta) + 1}_{\geq 0}) \ , \tag{6.11}$$

we obtain

$$\begin{aligned}
\frac{\partial rc_{BA}}{\partial fp_\alpha} &= \frac{1}{D^2}\Big(c_{12}P\left(Pf'_{ROC}(fp_\alpha) + N\right)\left(f_{ROC}(fp_\beta) - 1\right) + \\
&\quad + c_{21}NP\left(f_{ROC}(fp_\alpha) - f_{ROC}(fp_\beta) + 1 - fp_\alpha f'_{ROC}(fp_\alpha)\right)\Big) \\
\frac{\partial rc_{BA}}{\partial fp_\beta} &= \frac{1}{D^2}\Big(c_{12}P\left(-Pf_{ROC}(fp_\alpha)f'_{ROC}(fp_\beta) + f'_{ROC}(fp_\beta)N(fp_\alpha - fp_\beta + 1) - N\right) + \\
&\quad + c_{21}N\left(-Pfp_\alpha f'_{ROC}(fp_\beta) - fp_\alpha N\right)\Big) \\
\frac{\partial k}{\partial fp_\alpha} &= \frac{-Pf'_{ROC}(fp_\alpha) - N}{N+P} \\
\frac{\partial k}{\partial fp_\beta} &= \frac{Pf'_{ROC}(fp_\beta) + N}{N+P} \ .
\end{aligned} \tag{6.12}$$

Using (6.11) and (6.12), condition (6.10) simplifies to

$$\frac{\left(f'_{ROC}(fp_\alpha)f'_{ROC}(fp_\beta)c_{12}P^2 - f'_{ROC}(fp_\beta)NP(c_{21} - c_{12}) - c_{21}N^2\right)}{D} = 0 \ . \tag{6.13}$$

Based on the properties of the ROC curve, denominator $D$ is always positive (with an exception for an all-abstaining classifier, $fp_\alpha = 0 \wedge fp_\beta = 1$, for which $rc_{BA}$ is undefined), which means that (6.13) is equivalent to

$$f'_{ROC}(fp_\beta)\left(f'_{ROC}(fp_\alpha) + \frac{N}{P}(1 - \frac{c_{21}}{c_{12}})\right) = \left(\frac{N}{P}\right)^2\frac{c_{21}}{c_{12}} \ . \tag{6.14}$$

**Theorem 9** *If $f''_{ROC}$ is nonzero, assuming that the optimal classifier in the bounded-abstention model abstains for* exactly *a fraction $k = k_{\max}$ of instances, for any given $k \in [0,1]$ there exists* exactly *one classifier $\mathcal{A}_{\alpha,\beta}$.*

**Proof** In the first step we show that when $k = 0$, $fp_\alpha = fp_\beta = fp$, such that $f'_{ROC}(fp) = \frac{N}{P}\frac{c_{21}}{c_{12}}$. The equality $fp_\alpha = fp_\beta$ results from the properties of the ROC curve and (6.8). Condition $f'_{ROC}(fp) = \frac{N}{P}\frac{c_{21}}{c_{12}}$ results from (6.14).

In the second step we show that given an optimal classifier $\mathcal{A}_{\alpha,\beta}$ abstaining for exactly a fraction $k_{\max}$ of instances, we can easily generate a optimal classifier $\mathcal{A}^*_{\alpha,\beta}$ abstaining for a fraction $k^*_{\max} = k_{\max} + \delta_k$ (where $\delta_k \to 0$) of instances.

Such a classifier has coordinates $(fp_\alpha + \delta_\alpha, fp_\beta + \delta_\beta)$, in which the following condition holds:

$$\delta_k = \nabla k \cdot (\delta_\alpha, \delta_\beta) \ . \tag{6.15}$$

The derivative of a smooth convex down ROC curve is positive, which means that all components of $\nabla k$ are nonzero.

Using (6.14) for the new point, we obtain the relationship between $\delta_\alpha$ and $\delta_\beta$:

$$\begin{aligned} &\left(f'_{ROC}(fp_\beta) + f''_{ROC}(fp_\beta)\delta_\beta\right) \cdot \\ &\left(f'_{ROC}(fp_\alpha) + f''_{ROC}(fp_\alpha)\delta_\alpha + \frac{N}{P}(1 - \frac{c_{21}}{c_{12}})\right) = \left(\frac{N}{P}\right)^2 \frac{c_{21}}{c_{12}} \end{aligned} \tag{6.16}$$

and after simplifications we get the following result:

$$(\delta_\alpha, \delta_\beta) \cdot \left(f''_{ROC}(fp_\alpha), (f''_{ROC}(fp_\beta) \frac{\frac{N}{P}\frac{c_{21}}{c_{12}}}{(f'_{ROC}(fp_\beta))^2}\right) = 0 \ . \tag{6.17}$$

Equations (6.15) and (6.17) show that for nonzero $f''_{ROC}$ (i) there exists only one pair of $(\delta_\alpha, \delta_\beta)$ for given $\delta_k$, (ii) $\delta_k \leq 0 \Rightarrow \delta_\alpha \leq 0 \wedge \delta_\beta \geq 0$, and (iii) $k \to 0 \Rightarrow fp_\alpha \to 0 \wedge fp_\beta \to 1$.

This completes the proof. Note that an almost similar inductive proof can be shown for classifier starting from $fp = 0 \wedge fp = 1$, with negative increments $\delta_k$ (note that the second step of the proof did not make any assumptions about the sign on $\delta_k$). The advantage of such an approach is that there is no need to compute the value of starting point $fp_\alpha = fp_\beta = fp$ in this case. However, the derivative of $rc_{BA}$ at $fp_\alpha = 0 \wedge fp_\beta = 1$ formally does not exist. □

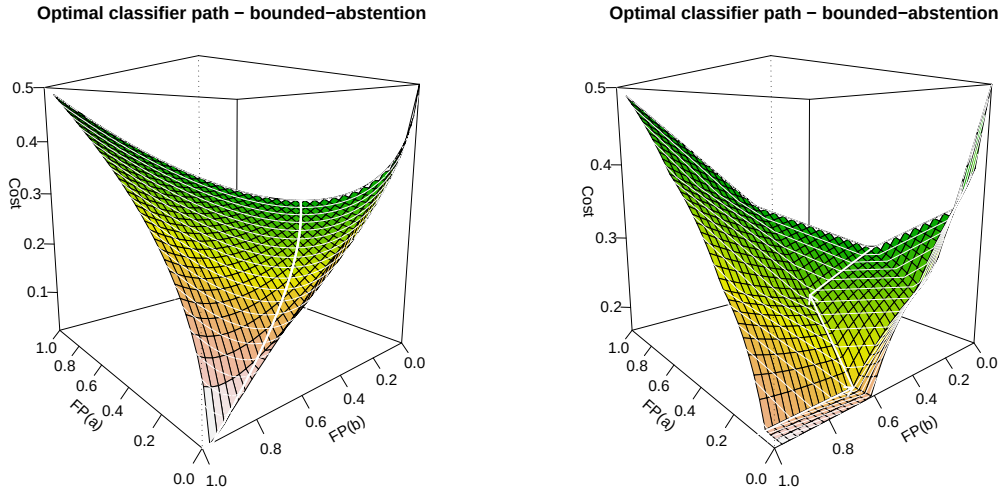
This proof generates an *optimal classifier path* on the surface of $rc_{BA}$ when varying $k_{\max}$ between 0 and 1, as shown in Figure 6.2a (thick white line). Note that the thin lines show isolines of constant $k$ (in this case with a constant increment of 0.1). The path can be constructed either by varying $k$ from 0 to 1 or by varying $k$ from 1 to 0. We will refer to these construction methods as "top-down" and "bottom-up" respectively.

The above derivation can be used to formulate Algorithm 5 for finding the optimal classifier in the bounded-abstention model:

**Input**: ROC curve $f_{ROC}$, fraction $k_{\max}$
**Result**: $(fp_\alpha, fp_\beta)$ defining a classifier $\mathcal{A}_{\alpha,\beta}$, abstaining for no more than $k_{\max}$ instances and having the lowest misclassification cost $rc_{BA}$

1 $fp_\alpha = fp_\beta = fp$, such that $f'_{ROC}(fp) = \frac{N}{P}\frac{c_{21}}{c_{12}}$.;
2 **while** $k(fp_\alpha, fp_\beta) < k_{\max}$ **do**
3 pick a small negative $\delta_k \to 0$ and find $\delta_\alpha, \delta_\beta$ such that $(\delta_\alpha, \delta_\beta) \cdot \left(f''_{ROC}(fp_\alpha), (f''_{ROC}(fp_\beta) \frac{\frac{N}{P}\frac{c_{21}}{c_{12}}}{(f'_{ROC}(fp_\beta))^2}\right) = 0$ and $\delta_k = \nabla k \cdot (\delta_\alpha, \delta_\beta)$;
4 $fp_\alpha \leftarrow fp_\alpha + \delta_\alpha$;
5 $fp_\beta \leftarrow fp_\beta + \delta_\beta$;
6 **end**

**Algorithm 5**: Algorithm for finding the optimal classifier.




(a) Optimal classifier path for a smooth convex up curve (Algorithm 5)

(b) Optimal classifier path for a piecewise ROCCH (Algorithm 6)

Figure 6.2: Optimal classifier paths in a bounded-abstention model.

### Optimal Classifier Abstaining for fewer than $k_{\max}$ Instances

In this section we will determine when the optimal classifier can abstain for a fraction smaller than $k_{\max}$ of instances. We will show when such a classifier exists and that when it exists it has the same misclassification cost as the optimal classifier abstaining for exactly a fraction $k_{\max}$ of instances.

Recall that the optimal abstaining classifier requires that (6.14) is met. In this section we will prove the following theorem:

**Theorem 10** *Given an optimal classifier $\mathcal{A}_{\alpha,\beta}$ with a cost $rc_{BA}$ abstaining for* exactly *a fraction $k_{\max}$ of instances, no optimal classifier $\mathcal{A}^*_{\alpha,\beta}$ abstaining for a fraction $k^*_{\max} \leq k_{\max}$ of instances and having a misclassification cost lower than $rc_{BA}$ exists.*

**Proof** Given an optimal classifier abstaining for *exactly* $k_{\max}$ instances, there exists a classifier abstaining for ($k^*_{\max} < k_{\max}$) and having the same or a lower misclassification cost iff $\partial rc_{BA}/\partial fp_\alpha \leq 0$ or $\partial rc_{BA}/\partial fp_\beta \geq 0$. In the remainder we will show when such a classifier exists.

In the calculations below we will use the following substitutions:

$$\begin{aligned} A_\alpha &= f'_{ROC}(fp_\alpha); \\ B_\alpha &= f_{ROC}(fp_\alpha) - fp_\alpha f'_{ROC}(fp_\alpha) \\ A_\beta &= f'_{ROC}(fp_\beta); \\ B_\beta &= f_{ROC}(fp_\beta) - fp_\beta f'_{ROC}(fp_\beta) \ . \end{aligned} \tag{6.18}$$

Note that for a nondecreasing and convex down $f_{ROC}$, the following conditions hold:

$$\begin{aligned} A_\alpha &\geq A_\beta \geq 0 \\ 0 \leq B_\alpha &\leq B_\beta \leq 1 \\ A_\alpha + B_\alpha &\geq A_\beta + B_\beta \geq 1 \ . \end{aligned} \tag{6.19}$$

**Calculating $\partial rc_{BA}/\partial fp_\beta$.** Calculating $\partial rc_{BA}/\partial fp_\beta \geq 0$, assuming that (6.14) holds and using substitution (6.18), yields the following condition:

$$\frac{\partial rc_{BA}}{\partial fp_\beta} \geq 0 \Leftrightarrow \underbrace{B_\alpha}_{\leq 0} A_\beta c_{12} P^2 + \underbrace{(-A_\beta - B_\beta + 1)}_{\leq 0} c_{12} NP \geq 0 \tag{6.20}$$

Equation (6.20) only holds if $B_\alpha = 0$ and $A_\beta + B_\beta = 1$ and in this case $\partial rc_{BA}/\partial fp_\alpha = 0$. From the properties of the ROC curve, this is only possible when $f_{ROC}$ contains line segments $(0,0) - (fp_\alpha, f_{ROC}(fp_\alpha))$ and $(fp_\beta, f_{ROC}(fp_\beta)) - (1,1)$. In addition, condition (6.14) must hold.

**Calculating $\partial rc_{BA}/\partial fp_\alpha$.** Calculating $\frac{\partial rc_{BA}}{\partial fp_\alpha} \leq 0$, assuming that (6.14) holds and using substitution (6.18), produces the following condition:

$$\frac{\partial rc_{BA}}{\partial fp_\beta} \leq 0 \Leftrightarrow (A_\beta + B_\beta - 1)(A_\alpha c_{12} P^2 + (c_{12} - c_{21}) NP) \leq -B_\alpha c_{21} NP \ . \tag{6.21}$$

Dividing both sides by (6.14) we obtain:

$$\underbrace{\frac{A_\beta + B_\beta - 1}{A_\beta}}_{\geq 0} \leq \underbrace{-B_\alpha}_{\leq 0} \frac{1}{NP} \ . \tag{6.22}$$

Similarly, this equation has a solution only if $B_\alpha = 0$ and $A_\beta + B_\beta = 1$.

To summarize, we proved that the classifier $\mathcal{A}_{\alpha,\beta}$ in the bounded-abstention model with a boundary abstention of $k_{\max}$ will have the lowest cost $rc_{BA}$ when it abstains for exactly a fraction of $k_{\max}$ instances. Moreover in the special case, when $\mathcal{A}_{\alpha,\beta}$ is such that $f_{ROC}$ contains the two line segments $(0,0) - (fp_\alpha, f_{ROC}(fp_\alpha))$ and $(fp_\beta, f_{ROC}(fp_\beta)) - (1,1)$, there exists an optimal classifier $\mathcal{A}^*_{\alpha,\beta}$ having the same misclassification cost and abstaining for fewer than $k_{\max}$ of instances. Such a classifier will be described by the ends of following line segments: $(0,0) - (fp^*_\alpha, f_{ROC}(fp^*_\alpha))$ and $(fp^*_\beta, f_{ROC}(fp^*_\beta)) - (1,1)$. Such cases correspond to a flat area in Figure 6.2b. □

### The Algorithm for a Convex Hull $f_{ROC}$

Algorithm 5 is does not allow an efficient generation of a solution, as the increments $\delta_\alpha$, $\delta_\beta$ it uses are infinitely small. Moreover the property of nonzero $f''_{ROC}$, required by Algorithm 5, does not hold. However, our function $f_{ROC}$ is a convex hull, a piecewise linear function, for which an efficient algorithm for finding the optimal classifier exists.

Assume the function $f_{ROC}$ is a piecewise linear convex down curve, constructed from $n$ line segments $S_1, S_2, \cdots, S_n$ connecting $n+1$ points $P_1, P_2, \cdots, P_{n+1}$. Each line segment $S_i$

is described by a line segment $tp = A_i fp + B_i$, where $A_i$ and $B_i$ are the coefficients of a line connecting points $P_i$ and $P_{i+1}$.

In this case, the value of derivatives $f'_{ROC}$ is equal to $A_i$ for $fp \in ]fp_{P_i}; fp_{P_{i+1}}[$ and is undefined for arguments $fp_{P_i}$ and $fp_{P_{i+1}}$. For our computations we assume that the value of $f'_{ROC}$ for every argument $fp_{P_i}$ takes all values between $[A_{i-1}; A_i]$. Moreover, we also assume that for $fp_{P_1}$ the derivative takes all values $]\infty; A_1]$ and for $fp_{P_{n+1}}$ the derivative takes all values $[0; A_n]$.

Note that with a piecewise linear ROCCH, (6.17) cannot be used because the values of $f''_{ROC}$ are either zero or undefined (at the vertices). However, (6.14) still can be used provided we allow that derivatives at vertices take all values in a range of slope values of adjacent segments.

Assuming the classifier $\mathcal{A}_{\alpha,\beta}$ optimal for a fraction $k_{\max}$ is defined by $(fp_\alpha, fp_\beta)$ where $fp_\alpha$ lies on the line segment $S_i$ and $fp_\beta$ lies on the line segment $S_j$, we construct the optimal classifier path "bottom-up" (i.e., constructing an optimal classifier $\mathcal{A}^*_{\alpha,\beta}$ for $k^*_{\max} < k_{\max}$)*. The coordinates $(fp^*_\alpha, fp^*_\beta)$ of the classifier $\mathcal{A}^*_{\alpha,\beta}$ will depend on the value of the following expression:

$$X = A_j \left( A_i + \frac{N}{P}(1 - \frac{c_{21}}{c_{12}}) \right) - \left( \frac{N}{P} \right)^2 \frac{c_{21}}{c_{12}} \ . \tag{6.23}$$

When $X < 0$, the classifier $fp_\alpha$ is located at the vertex (so that (6.14) holds) and the optimal classifier $\mathcal{A}^*_{\alpha,\beta}$ with $k^*_{\max} < k_{\max}$ will have $fp^*_\beta < fp_\beta$. Similarly, when $X > 0$, the classifier $fp_\beta$ is located at the vertex and the optimal classifier $\mathcal{A}^*_{\alpha,\beta}$ with $k^*_{\max} < k_{\max}$ will have $fp^*_\alpha > fp_\alpha$. In both these cases the corresponding points $fp^*_\beta$ and $fp^*_\alpha$ can be calculated from equation

$$k^*_{\max} = \frac{1}{N+P} \left( N \left( fp_\beta - fp_\alpha \right) + P \left( A_j fp_\beta + B_j - A_i fp_\alpha - B_i \right) \right) \ , \tag{6.24}$$

given that the corresponding points $fp_\alpha$ and $fp_\beta$ are fixed.

Finally, when $X = 0$, the classifier $\mathcal{A}_{\alpha,\beta}$ is located on line segments $S_i$, $S_j$ outside vertices. In this case, the optimal classifier is defined ambiguously for a given $k_{\max}$ and these classifiers can be generated by finding all pairs satisfying (6.24) given the constraints that $fp_\alpha$ is within a line segment $S_i$ and $fp_\beta$ is within a line segment $S_j$. Specifically, it is also possible to use either of the classifiers for the two preceding cases ($X < 0$ or $X > 0$).

This leads to the efficient Algorithm 6 for finding the optimal classifier. The algorithm constructs the abstaining classifier "bottom-up" starting from points $P_1$ and $P_n$. At each step the algorithm calculates the value of $X$ using (6.23) and depending on its sign, it advances either $i$ or $j$ as shown in Figure 6.3. If the abstention rate for new points $P_{i+1}, P_j$ (or $P_i, P_{j-1}$ correspondingly) is larger than $k_{\max}$ the solution is calculated by solving a linear equation $k(fp_\alpha, fp_\beta) = k_{\max}$ with respect to $fp_\alpha$ ($fp_\beta$) and the algorithm terminates. Otherwise, in the next iteration the evaluation of $X$ starts from the new point $P_{i+1}, P_j$ ($P_i, P_{j-1}$).

Assuming the ROCCH consists of $n$ line segments, the algorithm terminates in at most $n$ steps. Therefore its complexity is $O(n)$.

*If $fp_\alpha$ lies on the vertex connecting $S_{i-1}$ and $S_i$, we assume the value $A_i$. Similarly, for $fp_\beta$ lying on the vertex connecting $S_j$ and $S_{j+1}$, we assume the value $A_j$.

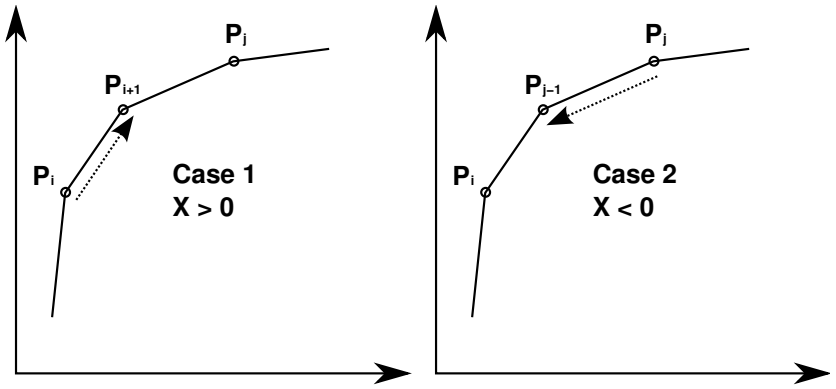



Figure 6.3: Finding the optimal classifier in a bounded model: visualization of $X$.

### 6.5.2 Bounded-Improvement Model

The second bounded model is when we limit $rc_B$ (hereafter referred to $rc_{BI}$) to a threshold value $rc_{\max}$ ($rc_{BI} \leq rc_{\max}$) and require that the classifier abstain for the smallest number of instances.

This model has multiple real-life applications, e.g., in the medical domain, where, given a certain test and its characteristics (ROC curve), the goal is to reduce the misclassification cost to a user-defined value $rc_{\max}$ and allow only the smallest number of abstentions:

| | |
|---|---|
| **Given** | – ROC curve generated using classifiers such that (6.2) holds<br>– $2 \times 2$ cost matrix $Co$<br>– Cost $rc_{\max}$ |
| **Find** | Classifier $\mathcal{A}_{\alpha,\beta}$ such that the cost of the classifier is no greater than $rc_{\max}$ and the classifier abstains for the smallest number of instances. |

This model is an inverse of the preceding model and can be solved by an algorithm similar to Algorithm 6. To show the solution for this model we use a similar approach as in the first model: First we will show the algorithm for a smooth convex down ROC curve having the derivative in $[0, 1]$ and assuming that the classifier has $rc_{BI}$ equal to $rc_{\max}$. Second, we will show under which conditions the optimal classifier can have a misclassification cost smaller than $rc_{\max}$. Finally, we will present an efficient algorithm for piecewise linear ROCCH curves.

#### Optimal Classifier for a Smooth Convex-down Curve

To show the solution for the bounded improvement model, we will use the constrained optimization method for the function of two variables. The minimization task can be defined as follows: Find the minimum of $k(fp_\alpha, fp_\beta)$, subject to condition $rc^*_{BI}(fp_\alpha, fp_\beta) = rc_{BI}(fp_\alpha, fp_\beta) - rc_{\max} = 0$. Similarly as in Section 6.5.1, we use the Lagrange method and obtain the same condition (6.14). The second condition is that $rc_{BI}(fp_\alpha, fp_\beta) = rc_{\max}$.

**Theorem 11** *If $f''_{ROC}$ is nonzero, assuming that the optimal classifier in the bounded-improvement model has $rc_{BI}$ equal to $rc_{\max}$, for a given $rc \in [0, rc^*]$, where $rc^*$ is the rc for the optimal binary classifier, there exists* exactly *one classifier $\mathcal{A}_{\alpha,\beta}$.*

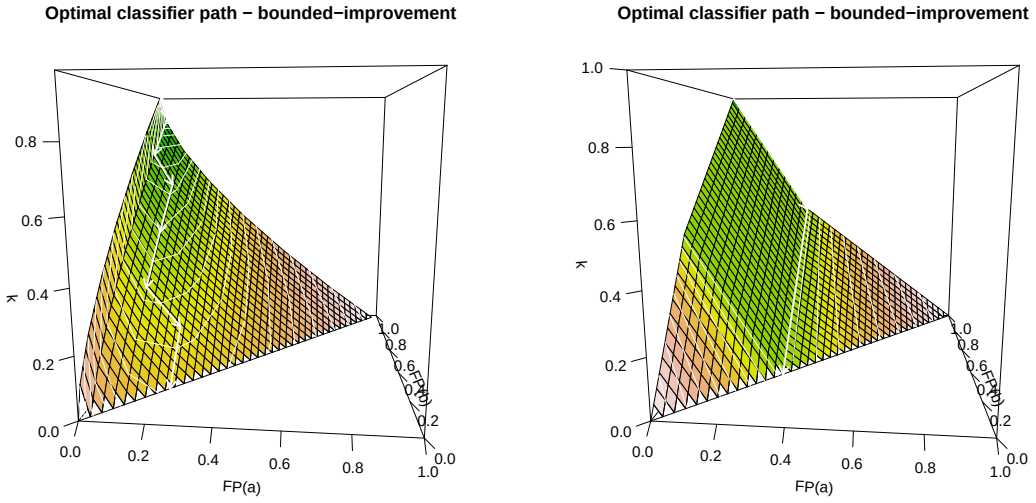
**Input**: ROCCH curve $f_{ROC}$, defined by $(n+1)$ points $P_1, \cdots P_{n+1}$, fraction $k_{\max}$
**Result**: $(fp_\alpha, fp_\beta)$ defining a classifier $\mathcal{A}_{\alpha,\beta}$, abstaining for no more than $k_{\max}$ instances and having the lowest misclassification cost $rc_{BA}$

$i_\alpha = 1, i_\beta = n+1$, $found \leftarrow$ FALSE ;
**while** *(!found)* **do**
  /* calculate coefficients for line segments $S_{i_\alpha}$ $S_{i_\beta - 1}$ */
  $A_{i_\beta - 1} \leftarrow \frac{P_{i_\beta}[tp] - P_{i_\beta - 1}[tp]}{P_{i_\beta}[fp] - P_{i_\beta - 1}[fp]}$ ;
  $B_{i_\beta - 1} \leftarrow P_{i_\beta}[tp] - A_{i_\beta - 1} P_{i_\beta}[fp]$ ;
  $A_{i_\alpha} \leftarrow \frac{P_{i_\alpha + 1}[tp] - P_{i_\alpha}[tp]}{P_{i_\alpha + 1}[fp] - P_{i_\alpha}[fp]}$ ;
  $B_{i_\alpha} \leftarrow P_{i_\alpha}[tp] - A_{i_\alpha} P_{i_\alpha}[fp]$ ;
  /* evaluate which point to advance */
  $X \leftarrow A_{i_\beta - 1} \left( A_{i_\alpha} + \frac{N}{P}(1 - \frac{c_{21}}{c_{12}}) \right) - \frac{N}{P} \frac{c_{21}}{c_{12}}$;
  **if** $X > 0$ **then**
    /* advance $fp_\alpha$ */
    **if** $k(P_{i_\alpha + 1}[fp], P_{i_\beta}[fp]) \geq k_{\max}$ **then**
      $i_\alpha \leftarrow i_\alpha + 1$;
    **else**
      $fp_\beta \leftarrow P_{i_\beta}[fp]$;
      /* solve a linear eq. $k(fp_\alpha, fp_\beta) = k_{\max}$ with respect to $fp_\alpha$ */
      $fp_\alpha \leftarrow -\frac{k_{\max}(N+P) - P(B_{i_\beta - 1} - B_{i_\alpha}) - fp_\beta(N + PA_{i_\beta - 1})}{N + PA_{i_\alpha}}$ ;
      $found \leftarrow$ TRUE;
    **end**
  **else if** $X < 0$ **then**
    /* advance $fp_\beta$ */
    **if** $k(P_{i_\alpha}[fp], P_{i_\beta - 1}[fp]) \geq k_{\max}$ **then**
      $i_\beta \leftarrow i_\beta - 1$;
    **else**
      $fp_\alpha \leftarrow P_{i_\alpha}[fp]$;
      /* solve a linear eq. $k(fp_\alpha, fp_\beta) = k_{\max}$ with respect to $fp_\beta$ */
      $fp_\beta \leftarrow \frac{k_{\max}(N+P) - P(B_{i_\beta - 1} - B_{i_\alpha}) + fp_\alpha(N + PA_{i_\alpha})}{N + PA_{i_\beta - 1}}$ ;
      $found \leftarrow$ TRUE;
    **end**
  **else**
    /* can move either $i_\alpha$ or $i_\beta$ */
    (...);
  **end**
**end**

**Algorithm 6**: Algorithm for finding the optimal classifier in a bounded-abstention model for a piecewise-linear ROCCH curve.

**Proof** The proof is similar to Theorem 9 with an identical first condition (6.14) and the second condition $\delta_{rc} = \nabla rc_{BI} \cdot (\delta_\alpha, \delta_\beta)$. However, unlike the in the preceding case, $\nabla rc_{BI}$ can be equal **0** (under conditions shown in the proof of Theorem 10). In such a situation, the misclassification cost $rc_{BI}$ will not change with the change of $fp_\alpha$ and $fp_\beta$. □

Similarly to the preceding case, the proof generates an optimal classifier path as shown in Figure 6.4a, where thin isolines show classifiers with identical misclassification cost $rc_{BI}$[†]. The optimal classifier crosses these isolines at the points of minimal $k$.




(a) Optimal classifier path for a piecewise ROCCH (Algorithm 7)

(b) The special case for an abstaining classifier

Figure 6.4: Optimal classifier paths in a bounded-improvement model.

### Optimal classifier with $rc_{BI}$ lower than $rc_{\max}$

As we proved in Theorem 10, if $f_{ROC}$ contains the line segments $(0,0)-(fp_\alpha, f_{ROC}(fp_\alpha))$ and $(fp_\beta, f_{ROC}(fp_\beta))-(1,1)$ and (6.14) holds, the classifier has the same $rc_{BI}$ for all classifiers in these line segments.

Moreover, in this case, this $rc_{BI}$ for this line segment is the lowest $rc_{BI}$ an abstaining classifier can achieve with this ROC curve. Therefore for a higher $rc_{\max}$ the optimal classifier will have a lower misclassification cost. Such a situation is illustrated in Figure 6.4b.

### The Algorithm for a Convex Hull $f_{ROC}$

The algorithm is similar to Algorithm 6, however it uses different conditions in lines 6 and 6, namely evaluating the misclassification cost given by

$$rc_{\max} = \frac{1}{(1-k)(N+P)}(Nfp_\alpha \cdot c_{21} + P(1-(A_j fp_\beta + B_j)) \cdot c_{12}) \ , \tag{6.25}$$

where $k$ is determined by (6.24). This yields

$$rc_{\max} = \frac{Nfp_\alpha \cdot c_{21} + P(1-(A_j fp_\beta + B_j)) \cdot c_{12}}{fp_\alpha(N+PA_i) - fp_\beta(N+PA_j) + P(B_j - B_i) + N + P} \ . \tag{6.26}$$

Similarly, lines 6 and 6 solve a linear equation (6.26) with respect to $fp_\alpha$ and $fp_\beta$. This yields Algorithm 7.

[†]Note that $rc_{BI}(0,1)$ is undefined—the algorithm should use a boundary value

**Input**: ROCCH curve $f_{ROC}$, defined by $(n+1)$ points $P_1, \cdots P_{n+1}$, fraction $rc_{\max}$
**Result**: $(fp_\alpha, fp_\beta)$ defining a classifier $\mathcal{A}_{\alpha,\beta}$, having cost not bigger than $rc_{\max}$ instances and abstaining for the smallest number of instances.

$i_\alpha = 1, i_\beta = n+1$, $found \leftarrow$ FALSE ;
**while** *(!found)* **do**
  /* calculate coefficients for line segments $S_{i_\alpha}$ $S_{i_\beta - 1}$ */
  $A_{i_\beta - 1} \leftarrow \frac{P_{i_\beta}[tp] - P_{i_\beta - 1}[tp]}{P_{i_\beta}[fp] - P_{i_\beta - 1}[fp]}$ ;
  $B_{i_\beta - 1} \leftarrow P_{i_\beta}[tp] - A_{i_\beta - 1} P_{i_\beta}[fp]$ ;
  $A_{i_\alpha} \leftarrow \frac{P_{i_\alpha + 1}[tp] - P_{i_\alpha}[tp]}{P_{i_\alpha + 1}[fp] - P_{i_\alpha}[fp]}$ ;
  $B_{i_\alpha} \leftarrow P_{i_\alpha}[tp] - A_{i_\alpha} P_{i_\alpha}[fp]$ ;
  /* evaluate which point to advance */
  $X \leftarrow A_{i_\beta - 1} \left( A_{i_\alpha} + \frac{N}{P}(1 - \frac{c_{21}}{c_{12}}) \right) - \frac{N}{P} \frac{c_{21}}{c_{12}}$;
  **if** $X > 0$ **then**
    /* advance $fp_\alpha$ */
    **if** $rc_{BI}(P_{i_\alpha + 1}[fp], P_{i_\beta}[fp]) \geq rc_{\max}$ **then**
      $i_\alpha \leftarrow i_\alpha + 1$;
    **else**
      $fp_\beta \leftarrow P_{i_\beta}[fp]$;
      /* solve a linear eq. $rc_{BI}(fp_\alpha, fp_\beta) = rc_{\max}$ with respect to $fp_\alpha$ */
      $fp_\alpha \leftarrow \frac{P(1 - B_{i_\beta - 1} c_{12}) - (N + P(1 + B_{i_\beta - 1} - B_{i_\alpha}))rc_{\max} - fp_\beta(-(N + PA_{i_\beta - 1})rc_{\max} + PA_{i_\beta - 1} c_{12})}{(N + PA_{i_\alpha})rc_{\max} - Nc_{21}}$ ;
      $found \leftarrow$ TRUE;
    **end**
  **else if** $X < 0$ **then**
    /* advance $fp_\beta$ */
    **if** $rc_{BI}(P_{i_\alpha}[fp], P_{i_\beta - 1}[fp]) \geq rc_{\max}$ **then**
      $i_\beta \leftarrow i_\beta - 1$;
    **else**
      $fp_\alpha \leftarrow P_{i_\alpha}[fp]$;
      /* solve a linear eq. $rc_{BI}(fp_\alpha, fp_\beta) = rc_{\max}$ with respect to $fp_\beta$ */
      $fp_\beta \leftarrow \frac{P(1 - B_{i_\beta - 1} c_{12}) - (N + P(1 + B_{i_\beta - 1} - B_{i_\alpha}))rc_{\max} - fp_\alpha((N + PA_{i_\alpha})rc_{\max} - Nc_{21})}{-(N + PA_j)rc_{\max} + PA_j c_{12}}$ ;
      $found \leftarrow$ TRUE;
    **end**
  **else**
    /* can move either $i_\alpha$ or $i_\beta$ */
    (...);
  **end**
**end**

**Algorithm 7**: Algorithm for finding the optimal classifier in the bounded-improvement model for the piecewise-linear ROCCH curve.

## 6.6 Experiments

To analyze the performance of our method, we tested it on 15 well-known datasets from the UCI KDD [HB99] database: `breast-cancer`, `breast-w`, `colic`, `credit-a`, `credit-g`, `diabetes`, `heart-statlog`, `hepatitis`, `ionosphere`, `kr-vs-kp`, `labor`, `mushroom`, `sick`, `sonar`, and `vote`.

We tested our method in all three models described above. In the $\mathcal{E}_{CB}$ model, the input

data is a $2 \times 3$ cost matrix in the symmetric case ($c_{13} = c_{23}$). In $\mathcal{E}_{BA}$, we use a $2 \times 2$ cost matrix and $k_{\max}$ (the fraction of instances that the system does not classify). In $\mathcal{E}_{BI}$, we could not use a constant value of $rc_{\max}$ for all datasets because different datasets yield different ROC curves and misclassification costs. Instead we used a *fraction cost improvement* $f$ and calculated $rc_{\max}$ as follows: $rc_{\max} = (1 - f_{\min})\, rc_{\text{bin}}$, where $rc_{\text{bin}}$ is the misclassification cost of the ROC-optimal binary classifier found using (2.5). Hence the input data is also a $2 \times 2$ cost matrix and a fraction $f_{\min}$.

### 6.6.1 Constructing an Abstaining Classifier

Recall that the ROC-optimal classifier in the cost-based model is located only on the vertices of the ROCCH (Section 6.4). In the other two models, the ROC-optimal classifier uses arbitrary points on the ROCCH, most typically one point is located at the vertex and a the other one is located on a line segment computed in Algorithms 6 and 7.

Such classifiers, corresponding to points lying on the line segment, can be constructed using a weighted random selection of votes of classifiers corresponding to two adjacent vertices [Faw03]. However, our prototype uses another method, which was more stable and produced less variance than the random selection did.

A ROCCH can be considered a function $f : \tau \mapsto (fp, tp)$, where $\tau \in T$ is a set of discrete parameters, varying which, one constructs classifiers $\mathcal{C}_\tau$ corresponding to different points on the ROCCH. In our algorithm we compute an inverse function $f^{-1} : (fp, tp) \mapsto \tau$ and interpolate it using splines with a function $\hat{f^{-1}}$ defined for a continuous range of values $\tau$. Given an arbitrary point ($fp^*$, $tp^*$) on the curve, we use the function $\hat{f^{-1}}$ yielding $\tau^*$ to construct a classifier $\mathcal{C}_{\tau^*}$

### 6.6.2 Testing Methodology

The experiment for each dataset was a two-fold cross-validation repeated five times with different seed values for the pseudo-random generator (we used $5 \times 2$ cross-validation, as it has a low-level Type-I error for significance testing [Die98]). We averaged the results for these runs and calculated 95% confidence intervals, shown as error bars on each plot. In the cross-validation, we used a training set to build an abstaining classifier, which was subsequently evaluated on the testing set.

The process of building an abstaining classifier is shown in Figure 6.5. We used another two-fold cross-validation ($n = 2$) to construct a ROC curve. The cross-validation was executed five times ($m = 5$), and the resulting ROC curves were averaged (threshold averaging [Faw03]) to generate a smooth curve. Although the method is applicable for any machine-learning algorithm that satisfies the watermark condition (6.2), we used a simple Naive Bayes classifier as a base classifier, converting it to a ranker by calculating the prediction ratio $P(+ \mid x)/P(- \mid x)$.

Given the ROC curve and the input parameters (cost matrix and a value $k_{\max}$ or $f_{\min}$), the program uses the algorithms proposed to find values $\alpha$ and $\beta$ describing $\mathcal{C}_\alpha$ and $\mathcal{C}_\beta$ and the ROC-optimal classifier (in each model). These values were used to set the thresholds in a Naive Bayes classifier built using the entire training set to create $\mathcal{A}_{\alpha,\beta}$.

Such an experiment was run for every dataset and every combination of input parameters, $CR$ and $c_{13}$ ($k_{\max}$ or $f_{\min}$), thus producing multiple plots (one for each dataset), multiple series (one for each cost ratio), and multiple points (one for each value of $c_{13}$, $k_{\max}$ or $f_{\min}$).

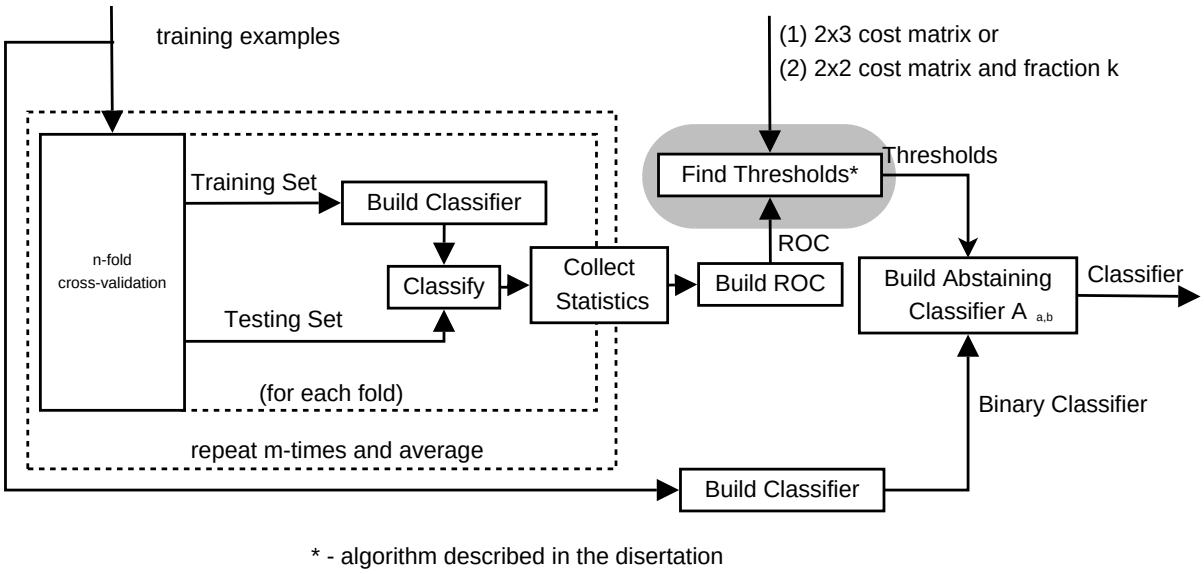



Figure 6.5: Building an abstaining classifier $\mathcal{A}_{\alpha,\beta}$.

We used three values of the cost ratio ($CR$): 0.5, 1 and 2, and four different values of $c_{13}$ (first model), $k_{\max}$: 0.1, 0.2, 0.3 and 0.5 (second model), and $f_{\min}$: 0.1, 0.2, 0.3 and 0.5 (third model), yielding 180 experiment runs ($15 \times 3 \times 4$) for each model.

We will briefly justify this choice of parameters. For the first model, we selected values of $c_{13}$ that are evenly spaced between 0 and the maximum value for a particular cost ratio (cf. (6.6)). For the other two models, we believe that, while the results will definitely be application-dependent, values of $k_{\max}$ ($f_{\min}$) that are lower than 0.1 bring too small an advantage to justify abstaining classifiers, whereas values larger than 0.5 may not be practical for real classification systems. For the $CR$s we tested the performance of our system for cost ratios close to 1.

We used Bayesian classifier from the Weka toolkit [WF00] as a machine-learning method and R [R D04] to perform numerical calculations.

### 6.6.3 Results—Cost-Based Model

Out of 180 simulations (15 datasets, four values of $c_{13}$, and three cost values), 152 are significantly better (lower $rc_{CB}$) than the corresponding optimal binary classifier (one-sided paired t-test with a significance level of 0.95). The optimal binary classifier was the same Bayesian classifier with a single threshold set using (2.5).

The results for a representative dataset are shown in Figure 6.6, and tabular results for all datasets for one cost ratio and two sample costs $c_{13}$ are shown in Table 6.2. The X-axes correspond to the cost value in a symmetric case $c_{13} = c_{23}$ (left and center panel), and the Y-axes show the relative cost improvement (left panel) and the fraction of nonclassified instances (center panel). The right panel displays the relationship between the fraction of skipped instances and the overall cost improvement. Horizontal error bars show 95% confidence intervals for the fraction of nonclassified instances, only indirectly determined by $c_{13}$.

We clearly observe that lower misclassification costs $c_{13} = c_{23}$ result in a higher number of instances being classified as "?" and higher relative cost improvement. However for different

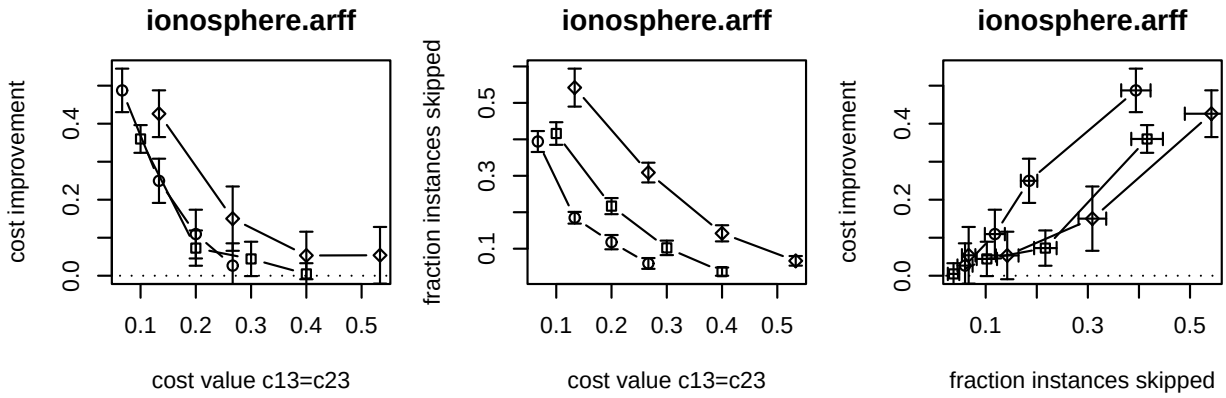



Figure 6.6: Cost-based model: Relative cost improvement and fraction of nonclassified instances for a representative dataset ($\circ : CR = 0.5$, $\square : CR = 1$, $\diamondsuit : CR = 2$).

Table 6.2: Fraction of nonclassified instances ($k$) and relative cost improvement ($f$) for a cost-based model ($CR = 1$, $c_{13} = \{0.1, 0.2\}$).

| Dataset | $c_{13} = 0.1$ | | $c_{13} = 0.2$ | |
|---|---|---|---|---|
| | $k$ | $f$ | $k$ | $f$ |
| breast-cancer | 0.97 ± 0.03 | 0.64 ± 0.01 | 0.68 ± 0.05 | 0.31 ± 0.02 |
| breast-w | 0.31 ± 0.03 | 0.16 ± 0.05 | 0.05 ± 0 | 0.13 ± 0.05 |
| colic | 0.96 ± 0.03 | 0.44 ± 0.02 | 0.27 ± 0.04 | 0.15 ± 0.03 |
| credit-a | 0.64 ± 0.01 | 0.48 ± 0.02 | 0.33 ± 0.02 | 0.22 ± 0.02 |
| credit-g | 0.84 ± 0.02 | 0.64 ± 0.01 | 0.59 ± 0.02 | 0.38 ± 0.01 |
| diabetes | 0.81 ± 0.01 | 0.64 ± 0.01 | 0.67 ± 0.02 | 0.35 ± 0.01 |
| heart-statlog | 0.76 ± 0.03 | 0.46 ± 0.01 | 0.32 ± 0.04 | 0.14 ± 0.04 |
| hepatitis | 0.46 ± 0.06 | 0.51 ± 0.03 | 0.29 ± 0.03 | 0.29 ± 0.04 |
| ionosphere | 0.42 ± 0.03 | 0.36 ± 0.04 | 0.22 ± 0.02 | 0.07 ± 0.05 |
| kr-vs-kp | 0.62 ± 0.02 | 0.46 ± 0.02 | 0.29 ± 0.01 | 0.26 ± 0.01 |
| labor | 0.65 ± 0.07 | 0.16 ± 0.13 | 0.36 ± 0.08 | -0.09 ± 0.17 |
| mushroom | 0.23 ± 0.02 | -0.08 ± 0.06 | 0.03 ± 0 | 0.22 ± 0.01 |
| sick | 0.13 ± 0 | 0.51 ± 0.03 | 0.09 ± 0 | 0.28 ± 0.05 |
| sonar | 0.93 ± 0.02 | 0.68 ± 0.02 | 0.77 ± 0.04 | 0.41 ± 0.03 |
| vote | 0.34 ± 0.04 | 0.55 ± 0.04 | 0.17 ± 0.01 | 0.39 ± 0.05 |

datasets even small differences in $c_{13}$ result in large differences of $k$ and $f$. On the other hand, for many datasets, we observe an almost linear relationship between the fraction of nonclassified instances and the relative cost improvement (right panel).

### 6.6.4 Results—Bounded Models

#### Bounded-abstention Model

Out of 180 simulations (15 datasets, four values of fractions of nonclassified instances and three cost values), 179 have a significantly lower $rc_{BA}$ than the corresponding optimal binary classifier (one-sided paired t-test with a significance level of 0.95). The optimal binary classifier is a Bayesian classifier with a single threshold.

We also observed that in most cases the resulting classifier classified the desired fraction of instances as the third class; the mean of the relative difference of $k$ ($\Delta k/k$) for all runs is 0.078 ($\sigma = 0.19$). This is particularly important as it is only indirectly determined by the two thresholds the algorithm calculates.

The results for a representative dataset are shown in Figure 6.7 and tabular results for all datasets for one cost ratio and two sample $k_{\max}$ are shown in Table 6.3. The X-axes correspond to the actual fraction of nonclassified instances and the Y-axes show the relative cost improvement (left panel) and the misclassification cost (right panel). The left panel shows the relative cost improvement as a function of the fraction of instances handled by operator $k$. The right panel shows the same data with the absolute values of $rc_{BA}$. The dashed arrows indicate the difference between an optimal binary classifier and an abstaining one.

In general, the higher the values of $k$, the higher the cost improvement; for eight datasets, namely `breast-cancer`, `credit-a`, `credit-g`, `diabetes`, `heart-statlog`, `ionosphere`, `kr-vs-kp` and `sonar`, we can observe an almost linear dependence between these variables. For four datasets (`breast-w`, `mushroom`, `sick`, `vote`) even as low an abstention as 0.1 can lead to a reduction of the misclassification cost by half (and of as much as 70% for two datasets).

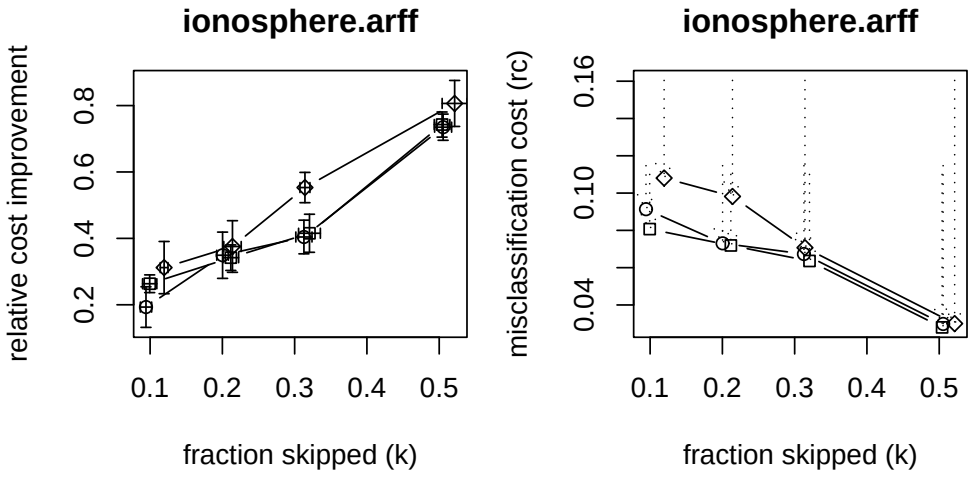



Figure 6.7: Bounded-abstention model: Relative cost improvement and the absolute cost for one representative dataset ($\circ : CR = 0.5$, $\square : CR = 1$, $\diamondsuit : CR = 2$).

### Bounded-improvement Model

This model is in fact the inverse of the preceding model, and thus we expected very similar results. The results for a representative dataset are shown in Figure 6.8, and tabular results for all datasets for one cost ratio and two sample $f_{\min}$ are shown in Table 6.6.4. The X-axes correspond to the relative cost improvement (left panel) and the misclassification cost (right panel). The Y-axes show the actual fraction of nonclassified instances. The left panel shows the fraction of instances handled by the operator as a function of the actual misclassification cost. It is interesting to compare the actual relative cost improvement $f$ and the assumed one (0.1, 0.2, 0.3, 0.5), as the former is only indirectly determined through two thresholds determined by the performance on the training set. The mean of the relative difference of $f$ ($\frac{\Delta f}{f}$) for all runs is 0.31 ($\sigma = 1.18$). The positive value of the mean shows that, on average, the system has a lower misclassification cost than required. Note that this value is higher than the corresponding difference in the preceding model. We conclude that this model is more sensitive to parameter changes than the preceding one. The right panel shows the same data

Table 6.3: Relative cost improvement ($f$) as a function of a fraction of nonclassified instances ($k_{\max}$) for a bounded-abstention model ($CR = 1$, $k_{\max} = \{0.1, 0.5\}$).

| Dataset | $k_{\max} = 0.1$ | | $k_{\max} = 0.5$ | |
|---|---|---|---|---|
| | $k$ | $f$ | $k$ | $f$ |
| breast-cancer | $0.1 \pm 0.01$ | $0.07 \pm 0.01$ | $0.53 \pm 0.01$ | $0.3 \pm 0.07$ |
| breast-w | $0.12 \pm 0.02$ | $0.58 \pm 0.06$ | $0.53 \pm 0.04$ | $1 \pm 0$ |
| colic | $0.09 \pm 0.01$ | $0.14 \pm 0.02$ | $0.48 \pm 0.01$ | $0.33 \pm 0.03$ |
| credit-a | $0.1 \pm 0$ | $0.17 \pm 0.02$ | $0.5 \pm 0.01$ | $0.55 \pm 0.03$ |
| credit-g | $0.1 \pm 0$ | $0.11 \pm 0.01$ | $0.51 \pm 0.01$ | $0.37 \pm 0.07$ |
| diabetes | $0.11 \pm 0.01$ | $0.11 \pm 0.02$ | $0.51 \pm 0.01$ | $0.41 \pm 0.03$ |
| heart-statlog | $0.11 \pm 0.01$ | $0.19 \pm 0.03$ | $0.56 \pm 0.02$ | $0.58 \pm 0.09$ |
| hepatitis | $0.13 \pm 0.01$ | $0.33 \pm 0.04$ | $0.53 \pm 0.03$ | $0.71 \pm 0.07$ |
| ionosphere | $0.1 \pm 0.01$ | $0.26 \pm 0.03$ | $0.5 \pm 0.01$ | $0.74 \pm 0.04$ |
| kr-vs-kp | $0.1 \pm 0$ | $0.25 \pm 0.01$ | $0.56 \pm 0.02$ | $0.89 \pm 0.02$ |
| labor | $0.12 \pm 0.04$ | $0.37 \pm 0.15$ | $0.58 \pm 0.05$ | $0.77 \pm 0.19$ |
| mushroom | $0.09 \pm 0.02$ | $0.71 \pm 0.01$ | $0.42 \pm 0.02$ | $1 \pm 0$ |
| sick | $0.11 \pm 0$ | $0.7 \pm 0.01$ | $0.47 \pm 0$ | $0.85 \pm 0.02$ |
| sonar | $0.13 \pm 0.01$ | $0.12 \pm 0.03$ | $0.56 \pm 0.02$ | $0.6 \pm 0.05$ |
| vote | $0.1 \pm 0.01$ | $0.46 \pm 0.04$ | $0.55 \pm 0.02$ | $0.96 \pm 0.03$ |

with the X-axis giving absolute values of costs. In addition the horizontal arrows (dashed) indicate the absolute values for the optimal binary classifier and the desired cost at the head of an arrow.

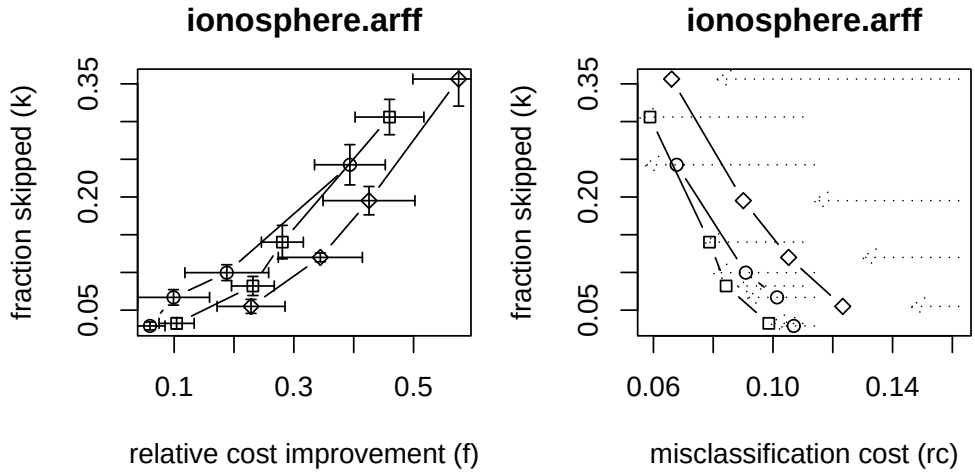



Figure 6.8: Bounded-improvement model: Fraction of nonclassified instances for a representative dataset ($\circ : CR = 0.5$, $\square : CR = 1$, $\diamondsuit : CR = 2$).

Similarly, to the preceding model, the four datasets can yield a 50% cost reduction while abstaining for approximately 10% of the instances. On the other hand, there are datasets in which even a 10% cost reduction is done at the cost of large abstention windows (e.g., 67% for `breast-cancer`). Considering much larger actual relative cost improvements than the desired one, we conclude that this model is more difficult to tune than the bounded-improvement model.

| Dataset | $f_{\min} = 0.1$ | | $f_{\min} = 0.5$ | |
|---|---|---|---|---|
| | $f$ | $k$ | $f$ | $k$ |
| breast-cancer | $0.28 \pm 0.25$ | $0.67 \pm 0.15$ | $0.66 \pm 0.24$ | $0.93 \pm 0.05$ |
| breast-w | $0.15 \pm 0.04$ | $0.03 \pm 0.03$ | $0.48 \pm 0.06$ | $0.11 \pm 0.03$ |
| colic | $0.09 \pm 0.05$ | $0.12 \pm 0.09$ | $0.43 \pm 0.11$ | $0.86 \pm 0.03$ |
| credit-a | $0.13 \pm 0.02$ | $0.06 \pm 0.01$ | $0.52 \pm 0.04$ | $0.46 \pm 0.02$ |
| credit-g | $0.14 \pm 0.05$ | $0.39 \pm 0.12$ | $0.46 \pm 0.09$ | $0.78 \pm 0.08$ |
| diabetes | $0.09 \pm 0.03$ | $0.39 \pm 0.17$ | $-0.2 \pm 0.91$ | $0.89 \pm 0.06$ |
| heart-statlog | $0.13 \pm 0.04$ | $0.07 \pm 0$ | $0.51 \pm 0.13$ | $0.62 \pm 0.08$ |
| hepatitis | $0.12 \pm 0.1$ | $0.22 \pm 0.19$ | $0.54 \pm 0.08$ | $0.49 \pm 0.18$ |
| ionosphere | $0.1 \pm 0.03$ | $0.03 \pm 0.01$ | $0.46 \pm 0.06$ | $0.31 \pm 0.02$ |
| kr-vs-kp | $0.1 \pm 0.01$ | $0.05 \pm 0.01$ | $0.5 \pm 0.01$ | $0.24 \pm 0.01$ |
| labor | $0.36 \pm 0.19$ | $0.23 \pm 0.16$ | $0.42 \pm 0.44$ | $0.59 \pm 0.16$ |
| mushroom | $0.07 \pm 0.01$ | $0 \pm 0$ | $0.48 \pm 0.02$ | $0.04 \pm 0.01$ |
| sick | $0.27 \pm 0.05$ | $0.04 \pm 0.01$ | $0.56 \pm 0.03$ | $0.07 \pm 0$ |
| sonar | $0.15 \pm 0.06$ | $0.19 \pm 0.01$ | $0.52 \pm 0.13$ | $0.74 \pm 0.05$ |
| vote | $0.13 \pm 0.04$ | $0.03 \pm 0.01$ | $0.53 \pm 0.02$ | $0.13 \pm 0.02$ |

## 6.7 Alternative Representations to ROC Curves

Our method for constructing the cost-optimal abstaining classifiers is based on ROC analysis and requires that the ROC curve of a base classifier is available. However, both machine learning as well as other domains, have also used alternative representation to ROC curves: precision-recall curves, DET-curves and cost curves. It is therefore natural to ask how our method can be applied when those alternative graphical representations are used.

### 6.7.1 Precision-Recall and ROC Curves

In information retrieval or other applications with extremely skewed class distributions precision-recall (P-R) curves are a commonly used alternative to ROC curves. Recall $r$ is defined as the fraction of relevant documents retrieved[‡] and precision $p$ is defined as the fraction of retrieved documents that are in fact relevant. In other words, $r = tp = TP/(TP + FN)$ and $p = TP/(TP + FP)$. It is therefore natural to ask how ROC and precision-recall curves are related.

**Theorem 12** *Under a constant class distribution $N/P$, a function mapping a ROC curve to a precision-recall curve is a bijection $f : f_{ROC} \leftrightarrow f_{PR}$.*

**Proof** By replacing absolute values in a confusion matrix ($FP$, $TP$, $FN$) to the corresponding rates ($fp$, $tp$, $fn$) and substituting the values of precision and recall we obtain:

$$f(fp, tp) = \left( \frac{tp}{tp + \frac{N}{P} fp}, tp \right) \wedge f^{-1}(p, r) = \left( \frac{Pr(1-p)}{Np}, r \right) \tag{6.27}$$

Note that the bijective mapping does not hold for points on the ROC space with $tp = 0 \wedge fp \neq 0$, which all map to $(0, 0)_{P-R}$. These points are uninteresting as they correspond to a worse-than-random classifier. Similarly, $(0, 0)_{ROC} \mapsto (f'_{ROC}(x)/(f'_{ROC}(x) + N/P)|_{x \to 0}, 0)_{P-R}$ so the mapping for this point is also not bijective.

[‡]The original definitions of precision and recall use a notion of *relevant documents* (positive instances) from a pool of all documents (both positive and negative instances). *Retrieved documents* refer to documents the system thinks are relevant (instances labeled as positive by the classifier).

For other points it is trivial to show using (6.27) that $f^{-1}(f(fp, tp)) = (fp, tp)$ for $(fp, tp) \in ]0, 1]\times]0, 1]$ and $f(f^{-1}(p, r)) = (p, r)$ for the domain of P-R curves, which completes the proof. □

It is easy to show that the mapping of ROC space into P-R space is not summetric along the main diagonal in the ROC space, corresponding to the "random" classifier. In fact, the diagonal is transformed to a vertical line with $p = P/(N + P)$ and all better-than-random classifiers are transformed to $[P/(N + P), 1] \times [0, 1]$ in the P-R space. Conversely, all worse-than-random classifiers, are transformed to $[0, P/(N + P)] \times [0, 1]$, further bounded by the curve $r = N/P \cdot p/(1-p)$ and all points lying above this curve are undefined in the P-R space.

Note that as P-R curves are typically used with $N \gg P$ the P-R and worse-than-random classifiers are typically not of interest, R-P curves "better utilize" the available space.

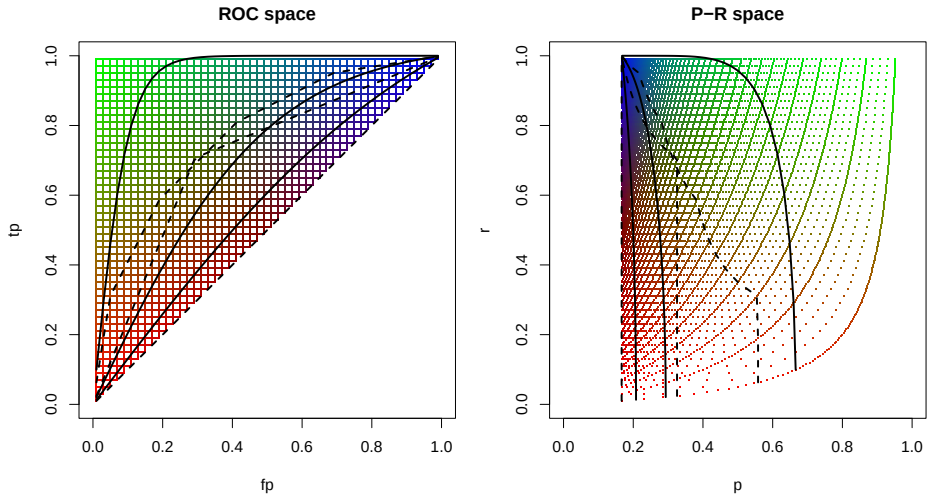



Figure 6.9: Conversion between sample ROC and P-R curves ($N/P = 5$).

There is also a number of interesting observations stemming from Theorem 12:

**Observation** If classifier $\mathcal{C}_\alpha$ dominates over $\mathcal{C}_\beta$ over the entire ROC space, it also dominates in the P-R space. As the mapping between ROC and P-R is bijective, it means that the curves can only cross each in one representation if they cross each other in the other representation. As dominating classifiers $\mathcal{C}_\alpha$ and $\mathcal{C}_\beta$ cross each other only at (0,0) and (1,1) in the ROC space, they cannot cross at any other points in the P-R space[§]. Hence, the classifiers $\mathcal{C}_\alpha$ and $\mathcal{C}_\beta$ will be also dominating in the P-R space.

**Observation** A line segment $tp = Afp + B$ in a ROC space corresponds to a segment of a rational function $r = C + D/(p - E)$, where

$$C = \frac{BN}{AP + N} \ \wedge \ D = \frac{ABPN}{(AP + N)^2} \ \wedge \ E = \frac{AP}{AP + N} \ . \tag{6.28}$$

[§]In fact, classifiers $\mathcal{C}_\alpha$ and $\mathcal{C}_\beta$ will have exactly two points of contact iff the tangents of their ROC curves at 0 are equal, as the transformation ROC↔P-R is not bijective for $(0, 0)$ in the ROC space.

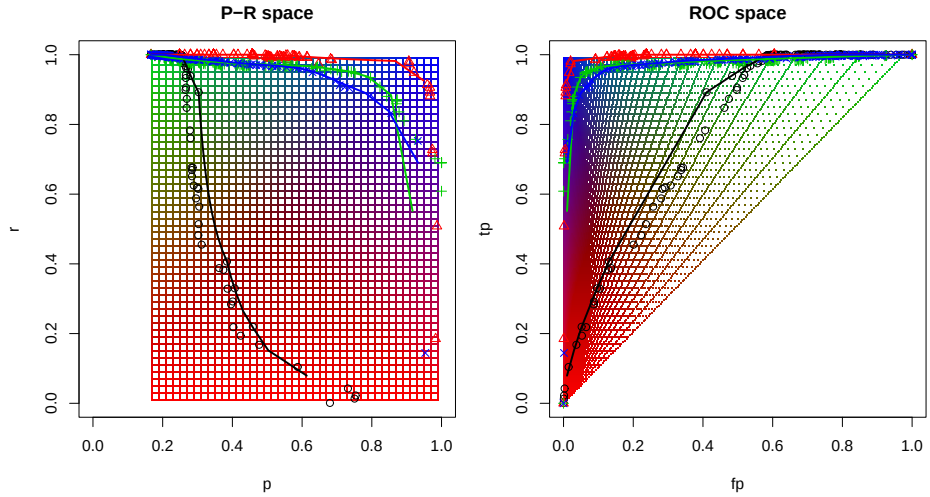



Figure 6.10: Conversion between sample P-R and ROC curves ($N/P = 5$). The ROCCH has been transferred back to the P-R curve.

This shows that the transformation of ROCCH into the P-R space consists of rational function segments.

**Observation** Convex hull ROCCH, dominating over ROC curve will also dominate in P-R space[¶]. However, the convex hull transformed into the P-R space (consisting rational function segments) is not necessarily convex.

This means that our method based on ROC curves can under constant cost distributions use P-R curves interchangeably (sample mapping is shown in Figures 6.9 and 6.10). However, unlike the ROC space, the linear approximation between two points in the DET space does not represent biased random voting of two classifiers corresponding to the two points (last observation) and therefore we cannot do such a linear approximation directly in the P-R space.

### 6.7.2 DET Curves

DET curves [MDK+97] are another representation of classifiers' performance commonly used in domains like speech recognition and also biometrics. DET curves use a simple non-linear transformation (normal deviate scale transformation) of both variables of the ROC plane. In fact, the transformation of both $fp$ and $fn$ is orthogonal and quite often DET curves are labeled in terms of $fp$ and $tp$. As the transformation used is a bijective function mapping $[0, 1] \rightarrow ]-\infty, \infty[$ the mapping between ROC space and DET space is a bijection $f : f_{ROC} \leftrightarrow f_{DET}$ as shown in (6.29), where $\Phi$ is a standard cumulative distribution function and $\Phi^{-1}$ is a quantile function.

DET curves carry the same information as ROC curves. However, DET curves are superior for comparing well performing systems (close to (0,1) in the ROC space) as this region is

[¶]This property has been independently noticed by Davis and Goadrich [DG06]

"expanded"[‖]. Conversely, the typically less-interesting region close to typically uninteresting $(0.5, 0.5)$ is "compressed" (see the density of iso-cost lines in Figure 6.11). DET curves of well-performing systems are typically close to linear [MDK+97].

$$f(fp, tp) = \left(\Phi^{-1}(fp), \Phi^{-1}(1 - tp)\right) \wedge f^{-1}(x, y) = (\Phi(x), \Phi(1 - y)) \tag{6.29}$$

However as the mapping is non-linear, lines in the ROC space are in general not mapped to lines in the DET space. This means that the convex hull in the ROC space is not necessarily convex in the DET space.

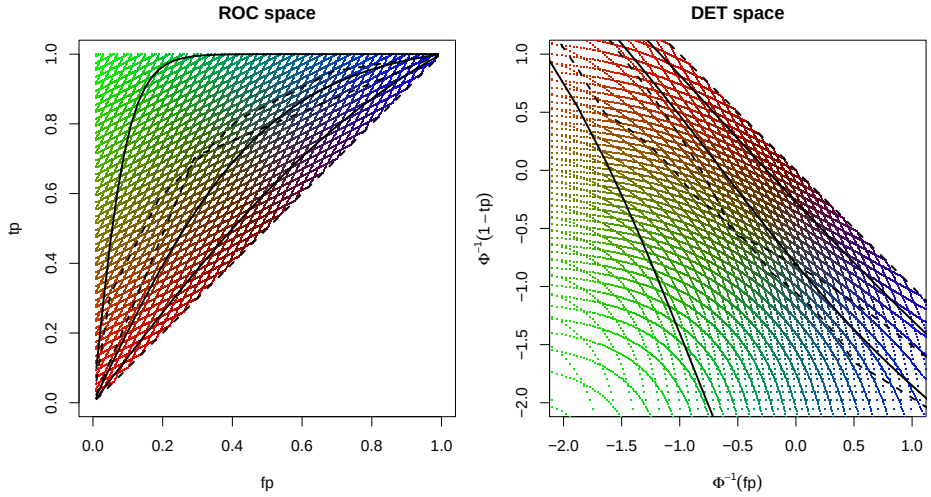



Figure 6.11: Conversion between sample ROC and DET curves. Grid shows iso-cost lines at $CR = 2$ and 0.5.

The isomorphism between ROC and DET means that our method can use DET curves interchangeably. However, unlike the ROC space, the linear approximation between two points in the DET space does not represent biased random voting of two corresponding classifiers and such an approximation cannot be used directly.

### 6.7.3 Cost Curves

Finally, cost curves [DH00] are another alternative to ROC curves, which allows to express misclassification cost directly. As shown by Drummond and Holte, there exists a point-line duality between ROC curves and cost curves, however it is currently not clear, if our method using ROC curves can be applied if only a cost curve of a classifier is given.

## 6.8 Related Work

Classifiers with reject rules were first investigated by Chow [Cho70] and further developed by Tortorella [Tor00] in the area of pattern recognition. The latter uses ROC analysis in a model corresponding to our cost-based model in a more restrictive setup ($c_{13} = c_{23}$). Our

[‖]In fact, expanded are all regions close to the four corners of ROC space.

work extends this model further and shows conditions under which a nontrivial abstaining classifier exists. We also propose two bounded models with other optimization criteria.

Cautious classifiers [FHO04] propose abstaining classifiers with a class bias $K$ and an abstention window $w$, which make them similar to our second evaluation model, in which an abstention window is defined. However, although for $w = 0$ abstention is zero and the classifier abstains for almost all instances for $w = 1$, the relationship between $w$ and the abstention is neither continuous nor linear [FHO04]. Therefore our model cannot be compared easily with cautious classifiers. Similarly, cautious classifiers require calibrated probabilities assigned to instances (otherwise the class bias might be difficult to interpret). In contrast, our model, if used with a scoring classifier, uses only information about the ordering of instances, not the absolute values of probabilities. This makes our model more general. On the other hand, cautious classifiers are more general in the sense that they can be used with a multi-class classification, whereas our model is based on ROC analysis and is only applicable to two-class classification problems.

Delegating classifiers [FFHO04] use a cascading model, in which classifiers at every level classify only a certain percentage of the instances. In this way every classifier, except for the last one, is a cautious classifier. The authors present their results with an iterative system, using up to $n - 1$ cautious classifiers.

Pazzani et al. [PMAS94] showed how different learning algorithms can be modified to increase accuracy at the cost of not classifying some of the instances, thus creating an abstaining classifier. However, this approach does not select the optimal classifier, is cost-insensitive and specific to the algorithms used.

Confirmation rule sets [GL00] are another example of classifiers that may abstain from classification. They use a special set of highly specific classification rules. The results of the classification (and whether the classifier makes the classification at all) depend on the number of rules that fired. Similarly to [PMAS94], the authors do not maximize the accuracy. Moreover, confirmation rule sets are specific to the learning algorithm used.

Active learning [LC94] minimizes the number of labeled instances by iteratively selecting a few instances to be labeled. This selection process uses an implicit abstaining classifier, where it selects instances that are lying closest to the decision boundary, however no cost-based optimization is performed.

## 6.9 Conclusions and Future Work

In this chapter we proposed a method to build a *ROC-optimal abstaining classifier* using ROC analysis. The resulting classifier minimizes the misclassification cost on instances used to build the ROC curve. Moreover, it has a low misclassification cost on other datasets from the same population as the one used to build the curve.

We defined the misclassification cost in three models: A cost-based, a bounded-abstention and a bounded-improvement model, which are relevant for numerous practical applications. All the models use only the base classifier and a ROC curve and do not require that the underlying classifier gives calibrated output probabilities, which is not always trivial [CG04, ZE01].

In the first model, we used a $2 \times 3$ cost matrix, showed the conditions under which the abstaining classifier has a nontrivial minimum cost, and presented a simple analytical solution. In the bounded model, we showed how to build the abstaining classifier assuming that no more

than a fraction $k_{\max}$ of instances is classified as the third class. Finally, in the third model, we showed how to build an abstaining classifier having a misclassification cost that is no greater than a user-defined value. In the latter two models, we showed an efficient algorithm for finding the optimal classifier. We presented an implementation and verified our method in all three models on a variety of UCI datasets.

As future work, it would be interesting to extend the experiments to include other machine-learning algorithms and also analyze the performance of our method for algorithms for which (6.2) does not hold. Our preliminary experiments showed that applying our method in such violating cases still yields good results. This is an interesting class of applications.

Future research area is on how to apply abstaining classifiers efficiently in real-world applications, also with multi-class classification.

# Chapter 7

# ALAC+—An Alert Classifier with Abstaining Classifiers

In this chapter we introduce ALAC+, a version of our alert-classification system, using abstaining classifiers. We investigate its design issues and chose appropriate abstaining classifier models. Finally, we evaluate ALAC+ on one synthetic and one real dataset in both agent and recommender modes.

## 7.1 ALAC Meets with Abstaining Classifiers

Recall that ALAC, the alert-classification system presented in Chapter 5, introduced the notion of autonomous alert processing and used the classification confidence with a statically determined threshold to determine if the system classified events "reliably". In this chapter we will replace this unreliable confidence assessment with abstaining classifiers.

Starting from our high-level problem specification in Section 1.1.3, here we introduce the third component to our utility function $\mathcal{U}$, namely the abstentions. This means that we will evaluate the system in both recommender mode (focusing on the misclassified alerts) and agent mode (focusing on misclassified alerts and the analyst's workload) in combination with abstaining classifiers.

Similarly to the base ALAC, we define our problem as follows: There is a human intrusion detection analyst $\mathcal{O}$ reviewing a sequence of intrusion detection alerts $(A_1, A_2, \ldots, A_i, \ldots)$ in the alert log $L$. The review is done by assigning one of the predefined set of classes $\{C_1, C_2, \ldots, C_n\}$ (which can be in particular two classes: true alerts and and false alerts $\{$"+", "−"$\}$) to each alert. The review is typically done sequentially and in real-time, which means that alert $A_{i+1}$ is reviewed only after alerts $(A_1, \ldots, A_i)$ has been reviewed and, at this time, alerts $(A_{i+2}, \ldots)$ are not known. The high-level goal is defined as:

| | |
|---|---|
| **Given** | – A sequence of alerts: $(A_1, A_2, \ldots, A_i, \ldots)$ in the alert log $L$,<br>– a set of classes $C = \{C_1, C_2, \ldots, C_n\}$,<br>– an intrusion detection analyst $\mathcal{O}$ sequentially and in real-time assigning classes to alerts,<br>– a utility function $\mathcal{U}$ minimizing the misclassification cost and allowing for abstentions. |
| **Find** | A classifier classifying alerts, maximizing the utility function $\mathcal{U}$ |

Unlike the base ALAC, the system can use abstaining classifiers, that is it does not have to make predictions for all alerts. Similarly, to Section 5.1, the system can work in both the agent and the recommender mode. In the agent mode, the utility function also takes into account the number of instances passed to the analyst, therefore limiting the analyst's workload. More formally, the operation of the system in the agent mode is defined as:

| | |
|---|---|
| **Given** | – A sequence of alerts: $(A_1, A_2, \ldots, A_i, \ldots)$ in the alert log $L$,<br>– a set of classes $C = \{C_1, C_2, \ldots, C_n\}$,<br>– an intrusion detection analyst $\mathcal{O}$ sequentially and in real-time assigning classes to alerts,<br>– a utility function $\mathcal{U}$ minimizing the misclassification cost and analyst's workload, allowing for abstentions. |
| **Find** | A classifier classifying alerts, maximizing the utility function $\mathcal{U}$ |

More specifically, the idea is to use abstaining classifiers (shown in Figure 7.1) as follows: If the abstaining classifier does not assign the class the alert is passed to the analyst unclassified. If the class is assigned, the behavior depends on the mode the system used is used in.

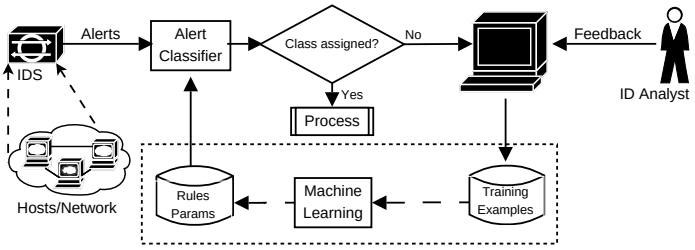



Figure 7.1: Simplified architecture of ALAC with abstaining classifiers.

In the recommender mode, all alerts with assigned class are nonetheless passed to the analyst, together with predicted labels. The advantage of using abstaining classifiers in this case it that, at the cost of skipping some alerts, the system has a much lower misclassification cost. This, with the exception of the incremental and the streaming nature of the system, is a direct application of abstaining classifiers. Algorithm 8 shows the operation of ALAC+ in this mode.

In the agent mode, if the class is assigned, the system works similarly to the baseline ALAC: alerts classified as true alerts are passed to the analyst and alerts classified as false alerts (except for a fraction $s$ of sampled instances) are discarded. Training instances used for subsequent training are the alerts that the analyst has reviewed: (i) alerts with no class assigned, (ii) alerts classified by ALAC as true alerts, and (iii) a sample $s$ of alerts classified by ALAC as false alerts. Remaining fraction $(1 - s)$ alerts classified by ALAC as false alerts are discarded and hence not used as training instances. Similarly to the original ALAC, it is also possible that there will be real attacks among these discarded alerts, which the analyst might not be aware of. In our evaluation we evaluate both types of false negatives. Algorithm 9 shows the operation of ALAC+ in this mode. In both cases, similarly to the base ALAC *goodClassifierPerformance*( $WA_{th}$) is based on the confusion matrix and calculated as a weighted accuracy with a threshold $WA_{th}$.

**Input**: a sequence of alerts $(A_1, A_2, \ldots, A_n)$, ROC curve $\mathcal{R}$, evaluation model $\mathcal{E}$ for selecting the optimal abstaining classifier
**Result**: a sequence of classified alerts $((A_1, C_{A_1}), (A_n, C_{A_n}), \ldots, (A_n, C_{A_n}))$

```
1  initialize;
   /* alerts used for the initial training                                  */
2  x ← x_0;
3  while x < n do
4  |  S_i ← subsequence(A_1, ..., A_x);
5  |  A_i ← learnUpdateAbstainingClassifier(C_{i-1}, S_i, R, ε);
6  |  while goodClassificationPerformance(WA_th) do
7  |  |  C_x ← classifyAbstaining(A_i, A_x);
8  |  |  if (C_x == "?") then
9  |  |  |  C_{A_x} ← askAnalystClassifyUnknown(A_x);
10 |  |  end
11 |  |  else
12 |  |  |  C_{A_x} ← askAnalystVerifyClassification(A_x, C_x);
13 |  |  end
14 |  |  updateAbstainingClassificationPerformance(C_x, C_{A_x});
15 |  |  x ← x + 1;
16 |  end
17 |  i ← i + 1;
18 end
```

**Algorithm 8**: ALAC+ algorithm—recommender mode.

### 7.1.1 The Problem with Rule Learners

In our evaluation of abstaining classifiers (cf. Section 6.6) we assumed that the underlying classifier $\mathcal{C}_\tau$ is a scoring classifier, which has the following two implications: (i) the ease of construction of the ROC curve [Faw03], (ii) we are sure that for two classifiers $\mathcal{C}_\alpha$ and $\mathcal{C}_\beta$ constructed from a single ranker the watermark condition (6.2) holds. The classifier used in ALAC is a rule learner, which does not have the above properties.

Addressing the first problem, rule learners are binary classifiers and do not output probabilities, which means that such a classifier produces a single point on the ROC plane. Such a point can be connected with straight lines to both trivial classifiers $(0, 0)$ and $(1, 1)$, representing biased random voting of the given classifier and trivial classifiers, classifying instances as either "+" or "−", but this does not yield an interesting ROC curve. A better alternative is to evaluate the performance of individual rules on an additional testing set and assign probabilities to those rules representing their accuracy (possibly with Laplace correction [Ces90]). This means that in general with $n$ rules we can have up to $n$ distinct points on the ROC curve.

However, we noticed in our experiments that even this method does not generate "good" curves and, therefore we resorted to another method. Recall from Section 5.4.1 that we combined the base cost-insensitive classifier with weighting [Tin98] with a weight $w$. With a single weight we obtain a single classifier so by varying weights we obtain different classifiers biased towards either positive or negative instances.

The weight combines both information about class distribution as well as the misclassification cost and makes an implicit assumption about the skew sensitivity of metrices used in the learning process, e.g., assuming that the base method produces a cost-optimal classifier

```
   Input: a sequence of alerts (A_1, A_2, ..., A_n), ROC curve R, evaluation model E for selecting
          the optimal abstaining classifier
   Result: a sequence of classified alerts ((A_1, C_{A_1}), (A_n, C_{A_n}), ..., (A_n, C_{A_n}))
 1 initialize;
   /* alerts used for the initial training                                          */
 2 x ← x_0;
 3 while x < n do
 4 |   S_i ← subsequence(A_1, ..., A_x);
 5 |   A_i ← learnUpdateAbstainingClassifier(C_{i-1}, S_i, R, ε);
 6 |   while goodClassificationPerformance(WA_th) do
 7 |   |   C_x ← classifyAbstaining(A_i, A_x);
 8 |   |   if (C_x == "?") then
 9 |   |   |   C_{A_x} ← askAnalystClassifyUnknown(A_x);
10 |   |   end
11 |   |   else
12 |   |   |   if (C_x == TRUE_ALERT) ∨ (randomSample(s)) then
13 |   |   |   |   C_{A_x} ← askAnalystVerifyClassification(A_x, C_x);
14 |   |   |   end
15 |   |   |   else
   |   |   |   |   /* e.g., discarding of false positives                          */
16 |   |   |   |   automaticProcessing() ;
17 |   |   |   |   C_{A_x} ← ∅;
18 |   |   |   end
19 |   |   end
20 |   |   updateAbstainingClassificationPerformance(C_x, C_{A_x});
21 |   |   x ← x + 1;
22 |   end
23 |   i ← i + 1;
24 end
```

**Algorithm 9**: ALAC+ algorithm—agent mode.

for a cost ratio $CR = 1$ by reweighting instances by a factor of 2, we would obtain a cost-minimizing classifier for $CR = 2$. However, as shown by Flach [Fla03], different evaluation metrics used by machine-learning algorithms have different skew sensitivities (e.g., accuracy is skew sensitive, but precision used by RIPPER for pruning is weakly skew-insensitive) and the weighting may not produce expected results. On the other hand, regardless of the skew sensitivity, the optimal classifier for a given class skew and cost distribution can be determined by means of ROC curves.

Consequently, we devised a simple method for constructing ROC curves and selecting the cost-optimal classifier for rule learners. We adaptively vary a weight $w$ and rebuild the classifier with this weight to obtain different points on the ROC plane. For each weight $w$ we ran cross-validation to obtain a threshold-averaged point $(tp, fp)$. Obviously, with $w \to 0$ we obtain a classifier classifying all instances as false (corresponding to $(0, 0)$) and, conversely, $w \to \infty$ yields $(1, 1)$. By varying the weight $w$ in this range we can obtain a complete curve. We used Algorithm 10 based on a binary search to construct the interesting fragments of the curve. Subsequently, using the obtained curve we selected the classifier that was optimal in terms of misclassification costs for a given cost ratio and class distribution.

This however, leads us to the second problem, namely, that as classifiers $\mathcal{C}_\alpha$ and $\mathcal{C}_\beta$ obtained in this way are independent, the watermark condition (6.2) can be violated. Investigat-

**Input**: a learning algorithm $\mathcal{L}$ with a parameter $w$ producing a classifier $\mathcal{C}_w$. Boundary weights $w_0$ and $w_\infty$, a desired number of points on the ROC curve $n$.
**Result**: $R$ a set of $n$ points $\{((fp, tp), w)\}$ describing the ROC curve

$(fp_0, tp_0) \leftarrow$ evaluateClassifier$(\mathcal{C}_{w_0})$;
$(fp_\infty, tp_\infty) \leftarrow$ evaluateClassifier$(\mathcal{C}_{w_\infty})$;
$R \leftarrow ((fp_0, fp_0), w_0) \cup ((fp_\infty, fp_\infty), w_\infty)$;
**while** $|R| < n$ **do**
  $S_{fp} \leftarrow$ sortbyFP$(R)$;
  $S_{tp} \leftarrow$ sortbyTP$(R)$;
  `/* find two consecutive points in a sorted sequence with the largest Euclidean distance */`
  $((fp_i, fp_i), w_i), ((fp_{i+1}, fp_{i+1}), w_{i+1}) \leftarrow$ findLargestGap$(S_{fp})$;
  $((fp_j, fp_j), w_j), ((fp_{j+1}, fp_{j+1}), w_{j+1}) \leftarrow$ findLargestGap$(S_{tp})$;
  `/* pick a pair with the larger distance */`
  $((fp_k, fp_k), w_k), ((fp_{k+1}, fp_{k+1}), w_{k+1}) \leftarrow$ pickLargerGap$(i, j)$;
  `/* pick the midpoint */`
  $w \leftarrow$ average$(w_k, w_{k+1})$;
  $(fp, tp) \leftarrow$ evaluateClassifier$(\mathcal{C}_w)$;
  `/* add the new point */`
  $R \leftarrow R \cup ((fp, tp), w)$;
**end**

**Algorithm 10**: Filling the gaps—Building ROC curves for arbitrary classifiers $C_w$.

ing this issue we found out that, in our case, the number of classifiers for which this condition is violated is extremely small (less than 1% of the total abstention). We therefore concluded that those violating cases have little impact on the efficiency of our method and decided that the classifier should abstain in this case. However, it will be interesting to investigate this property for classifiers obtained in this way.

## 7.2 ALAC+ Evaluation

The goal of ALAC+ evaluation is to test the the following hypothesis:

**Hypothesis 7.2.1** *By combining abstaining classifiers with ALAC, one can significantly reduce the number of misclassifications and the overall misclassification cost rc.*

If this hypothesis is confirmed this makes abstaining classifiers particularly attractive for intrusion detection.

### 7.2.1 Choosing Evaluation Models for ALAC+

Recall from Chapter 6 that we introduced three models for evaluating abstaining classifiers, namely, the cost-based model, the bounded-abstention model and the bounded-improvement model. While all of them could be used with ALAC, we decided to focus on the latter two, for the following two reasons: First, in our problem domain, the exact misclassification costs are not given and would have to be estimated, including the extended $2 \times 3$ cost matrix for the first model. This would make the system difficult to tune and interpret. Second, the latter two models are more intuitive and more likely to be used in real environments.

Hence, we will use the bounded-improvement (BI) and the bounded-abstention (BA) models in the evaluation. In both of these models we use a ROC curve to optimize the misclassification cost per actually classified instance, with boundary conditions. In the bounded-improvement model, the boundary condition is that the classifier should not abstain for a fraction $k_{\max}$ of instances, and in the bounded-abstention model the classifier should have the misclassification cost not larger than $rc_{\max}$ (can be also expressed as $f_{\min}$).

### 7.2.2 Setting System Parameters

As discussed in Section 5.4.1, the classification of IDS alerts is a cost-sensitive task, where false negatives are much more expensive than false positives. For such a cost-sensitive setup, given the cost matrix and the class distribution and additional parameters for the abstaining classifier evaluation model ($k_{\max}$ for BA and $f_{\min}$ for BI) we can select optimal classifiers.

#### ROC Analysis and Abstaining Classifiers

In our classification task the cost matrix is not given and can only be estimated. Machine learning typically uses Area Under Curve (AUC) in such cases, describing classifier's performance under all class distributions and all possible costs. Such an approach may not work well in this case as we do know that not all costs and class distributions are equally likely. One possible approach could be to use cost curves [DH00], which allows one to calculate the expected misclassification cost given a certain cost distribution. We consider this an interesting alternative to ROC analysis, which can be further investigated.

As a second approach one could use an estimated costs ($CR$) and obtain mode "tangible" results expressed in terms of $fp$, $fn$ and $rc$ (recall notation introduced in Section 2.2.3). In this evaluation we decided to take this second approach and to use an estimated realistic cost ratio $ICR = 50$, where false negatives are 50 times more costly than false positives. Note that this is different from the approach taken in the previous section with ALAC, where we used instance weighting with $w = 50$. In fact, $ICR$ and class distribution $N/P$ are used with ROC analysis to select an optimal classifier determined by the parameter $w$ (cf. Section 7.1.1).

To be able to compare the misclassification cost of abstaining and binary classifiers we need additional parameters for our two abstaining models. In the bounded-abstention model this parameter is $k_{\max}$, a fraction of instances which are not classified and in the bounded-improvement model the parameter is $f_{\min}$ a desired fraction cost improvement over the optimal binary classifier. While the value of these parameters are application and environment dependent we tried to select the values the model can be run with in practice: $k_{\max} = 0.1$ for the bounded-abstention model (further referred to as *BA 0.1*) and for the bounded-improvement model $f_{\min} = 0.5$ (further referred to as *BI 0.5*). The first model, BA 0.1 can be used if the environment requires that the human analyst can manually classify up to 10% of all alerts. In the second model, BI 0.5 the system would reduce the misclassification cost by 50%, compared to the optimal binary classifier.

Given these parameters we used ROCCH [PF98] to find the point on the ROC curve describing cost optimal binary classifier ($w_b$) and ran Algorithm 6 or Algorithm 7 to find corresponding abstaining classifier weights ($w_{\mathcal{C}_\alpha}$ and $w_{\mathcal{C}_\beta}$) and to calculate the performance estimates as shown in Table 7.1.

Note that in the first model, BA 0.1, where the classifier abstains for 10% of all instances, the overall misclassification cost (per classified instance) decreased by 27% for the first dataset

Table 7.1: Calculating weights $w$ for the optimal binary classifier and abstaining classifiers in the bounded abstention and the bounded improvement models.

| Dataset | Model | $w_b$ | $w_{\mathcal{C}_\alpha}$ | $w_{\mathcal{C}_\beta}$ | $rc_b$ | $rc_{\mathcal{A}_{\alpha,\beta}}$ | $f$ | $k$ |
|---|---|---|---|---|---|---|---|---|
| DARPA | BA 0.1 | 72 | 4.25 | 76 | 0.258 | 0.189 | 0.268 | 0.1 |
| Data Set B | BA 0.1 | 16 | 0.25 | 173 | 0.088 | 0.025 | 0.712 | 0.1 |
| DARPA | BI 0.5 | 72 | 0.625 | 116 | 0.258 | 0.129 | 0.5 | 0.337 |
| Data Set B | BI 0.5 | 16 | 0.25 | 64 | 0.088 | 0.044 | 0.5 | 0.067 |

and as much as 71% for the second dataset, which is a significant improvement. This is consistent with the second model, where the reduction in the misclassification cost by 50% is achieved at the cost of abstaining for 33.7% of instances for the first dataset and only 6.7% for the second dataset. We will revisit these results in Section 7.2.3, where we compare these estimates with the actual performance of the alert-classification system.

Recall that we chose $ICR = 50$ to model classifier's cost-sensitive environment. As this was only an estimate, it might be interesting to see how classifiers $\mathcal{C}_\alpha$ and $\mathcal{C}_\beta$ are affected by the choice of this parameter. To answer this question we performed a simulation calculating optimal abstaining classifiers with different misclassification costs. As shown in Figure 7.2 with increasing misclassification costs the classifier moves towards a point $(1, 1)$ a trivial binary classifier classifying all instances as true. Conversely, decreasing the misclassification cost shifts the classifier towards $(0, 0)$. Note that for all three cases we assumed that the class distribution $N/P = 5$.

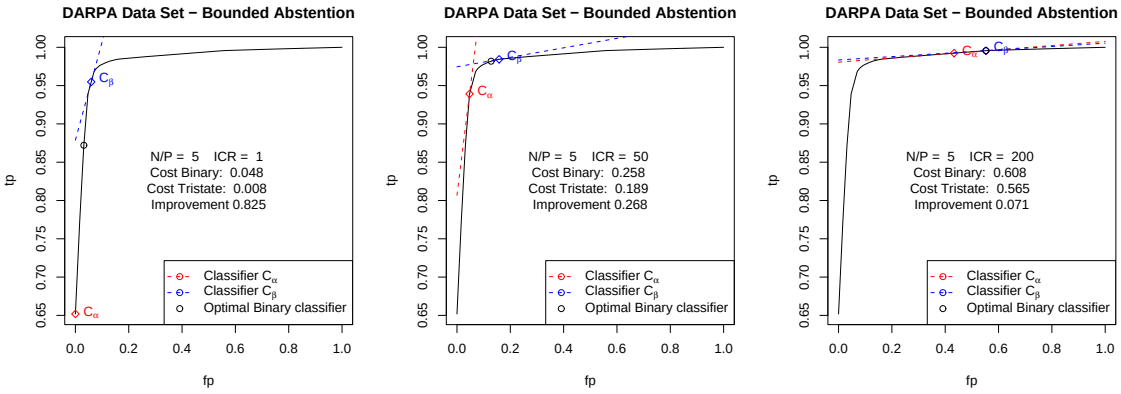



Figure 7.2: Classifiers for three different misclassification costs $ICR = 1$, $ICR = 50$ (used in the remaining experiments) and $ICR = 200$ (DARPA 1999, BA 0.1).

### Background Knowledge

Recall that in Chapter 5 we discussed three types of background knowledge and used ROC analysis to show that it is advantageous in alert classification. However, the recursive background knowledge requires special care to be handled properly. In particular, we would assume that the analyst classifies alerts strictly sequentially, which is not necessarily the case. In fact, the analyst may work with alert groups and only classify them together. In this case recursive background knowledge may lead to errors. Furthermore, non-recursive aggregates

use ad-hoc time intervals and only marginally improve the ROC curves.

Hence, for the evaluation of ALAC+ we decided to use only environmental background knowledge. Note that this is a more difficult classification task, as the cost-optimal classifier has much higher false-positive rates than in the evaluation performed in Section 5.5. We nonetheless show that ALAC+ using abstaining classifier can perform better, even with less background knowledge.

### 7.2.3 Cost Results

After setting the classifier parameters including the weights obtained in the previous section, we ran the alert-classification system ALAC+ sequentially classifying alerts as this is the environment the system will be used in. In all cases, the initial 30% of alerts was used to build an initial model. We compared the following configurations:

**Recommender (baseline)** The original ALAC in the recommender mode, where all alerts are classified and passed to the analyst. In this mode the system uses a single weight $w$ as shown in Table 7.1.

**Recommender BA 0.1/BI 0.5** ALAC+ in the recommender mode, where the system can abstain from classification. The nonclassified alerts will be passed to the analyst with no labels, hence these alerts are not taken into consideration in the confusion matrix. Note that we are calculating the misclassification cost per actually classified alert. In this mode the system uses two weights $w_{\mathcal{C}_\alpha}$ and $w_{\mathcal{C}_\beta}$.

**Agent (baseline)** The original ALAC agent mode, where the system assesses the confidence of classification based on the rule performance and, if it is above a certain threshold discards all but a fraction $s$ (set to $s = 0.3$) of randomly sampled false alerts*. These alerts are not passed to the analyst and are not used as training instances, therefore the system may discard some real alerts. The system uses a single classifier trained with a weight $w$.

**Agent (naive)** ALAC in "naive" agent mode, where the system does not assess the confidence of classification and discards all but some randomly sampled false alerts. This simplified agent is expected to perform worse than ALAC, but it is a baseline for ALAC+, where the confidence assessment is replaced with an abstaining classifier. This system is also a boundary case for ALAC+ with $\mathcal{C}_\alpha = \mathcal{C}_\beta$.

**Agent BA 0.1 / BI 0.5** This is the ALAC+ system using abstaining classifiers and automatically discarding some alerts. The system uses two weights $w_{\mathcal{C}_\alpha}$ and $w_{\mathcal{C}_\beta}$ as shown in Table 7.1.

The complete results with 95% confidence intervals for these runs for two datasets are shown in Table 7.2 (DARPA 1999) and Table 7.3 (Data Set B). For the agent mode we give two values *fn*—the number of false negatives the system calculates and $FN_r$—the total number of real false negatives, including false negatives due to discarded alerts. Note that uppercase variables $FN$, $FN_r$, $FP$ denote the absolute values, which better demonstrate the number of misclassifications made by ALAC+. For the sake of completeness, similarly to the

*Note that $s$ is not the actual fraction of discarded alerts, but the false-positive sampling rate. The actual number of discarded alerts depends on the class distribution and also the assigned confidence, cf. Algorithm 3.

evaluation of the base ALAC, Figures B.7–B.10 show the corresponding rates for a single run for both models and datasets.

Table 7.2: Misclassifications for ALAC and ALAC+ in the recommender and agent for DARPA 1999 Data Set.

| System | *FN* | $FN_r$ | *FP* | *k* | Discarded | *rc* |
|---|---|---|---|---|---|---|
| Rec. (baseln.) | 326 ± 8 | 326 ± 8 | 5268 ± 196 | 0 ± 0 | 0 ± 0 | 0.51 ± 0.01 |
| Rec. BA 0.1 | 308 ± 6 | 308 ± 6 | 1775 ± 5 | 0.06 ± 0.01 | 0 ± 0 | 0.43 ± 0.01 |
| Rec. BI 0.5 | 275 ± 8 | 275 ± 8 | 86 ± 0 | 0.18 ± 0.01 | 0 ± 0 | 0.41 ± 0.01 |
| Agent (baseln.) | 188 ± 17 | 395 ± 10 | 4471 ± 288 | 0.15 ± 0.01 | 0.36 ± 0.01 | 0.57 ± 0.02 |
| Agent (naive) | 198 ± 21 | 573 ± 66 | 4425 ± 484 | 0.023 ± 0.013 | 0.52 ± 0.02 | 0.78 ± 0.07 |
| Agent BA 0.1 | 183 ± 12 | 427 ± 18 | 1559 ± 17 | 0.08 ± 0.01 | 0.49 ± 0.01 | 0.59 ± 0.02 |
| Agent BI 0.5 | 153 ± 8 | 391 ± 14 | 51 ± 0.4 | 0.19 ± 0.01 | 0.46 ± 0.01 | 0.58 ± 0.02 |

Table 7.3: Misclassifications for ALAC and ALAC+ in the recommender and agent modes for Data Set B.

| System | *FN* | $FN_r$ | *FP* | *k* | Discarded | *rc* |
|---|---|---|---|---|---|---|
| Rec. (baseln.) | 20 ± 5 | 20 ± 5 | 1217 ± 233 | 0 ± 0 | 0 ± 0 | 0.07 ± 0.01 |
| Rec. BA 0.1 | 4.7 ± 1.0 | 4.7 ± 1.0 | 34 ± 0 | 0.10 ± 0.001 | 0 ± 0 | 0.0091 ± 0.0017 |
| Rec. BI 0.5 | 17.6 ± 1.9 | 17.6 ± 1.9 | 34 ± 0 | 0.03 ± 0.001 | 0 ± 0 | 0.029 ± 0.003 |
| Agent (baseln.) | 5.71 ± 2.0 | 18 ± 2.8 | 628 ± 20 | 0.0024 ± 0.0017 | 0.47 ± 0.007 | 0.047 ± 0.004 |
| Agent (naive) | 4.57 ± 2.3 | 18 ± 3.0 | 925 ± 228 | 0.0025 ± 0.0013 | 0.47 ± 0.007 | 0.054 ± 0.014 |
| Agent BA 0.1 | 1.85 ± 1.16 | 7.14 ± 2.47 | 23.1 ± 4.39 | 0.126 ± 0.006 | 0.39 ± 0.0 | 0.013 ± 0.004 |
| Agent BI 0.5 | 2.6 ± 1.0 | 10 ± 3.87 | 22 ± 1.75 | 0.05 ± 0.001 | 0.45 ± 0.001 | 0.017 ± 0.006 |

**Recommender Mode** Looking at the results we clearly see that for both datasets combining the recommender mode with abstaining classifiers reduces both the number of false positives and the number of false negatives. While this is to be expected (any abstention reduces the absolute number of classified instances thus also misclassifications) the rates calculated per actually classified instances are also lower.

For example, for the DARPA 1999 Data Set, the recommender mode has a comparable false-negative rate and significantly lower false-positive rate (lower by 63% for BA 0.1 and by 97% for BI 0.5, where 18% of all alerts are left unclassified). The resulting misclassification

cost per classified instance was lowered by 15-20%. For Data Set B, we observed a much better improvement than for the first dataset: with 10% of alerts left unclassified (BA 0.1) we lowered the false-negative rate by 76% and the false-positive rate by 97%, which resulted in the overall cost reduction by 87%. For BI 0.5 we achieved 50% cost reduction by abstaining for as little as 3% of all instances.

**Agent Modes** In the agent mode there are two types of errors the system makes: the ones corrected by the analyst and the ones the analyst cannot correct (missed attacks). Obviously if the system is to be useful in the agent mode, it should minimize the second types of errors.

Our results show that for the DARPA while the agent using abstaining classifiers has much lower false-positive rates than the "naive" Agent and the baseline Agent, comparable (or even higher) actual false-negative rates ($fn_r$) cancel the benefits of the system. At the end the agent using abstaining classifiers has a lower misclassification cost than the "naive" Agent, but comparable cost to the baseline agent.

For Data Set B, the agent using abstaining classifiers improves over the Agent in terms of a lower false-negative rate (up to 60%) and a false-positive rate (up to 96%) resulting in the overall cost reduction by 72% when 12.6% of the instances are left unclassified.

**Discrepancy ROC estimates and ALAC performance** Trying to explain why ALAC does not perform as well for the DARPA 1999 Data Set we observed that the expected misclassification cost obtained for the optimal classifier on the ROC curve are significantly different (0.258 based on the ROC curve and the actual recommender performance at 0.51).

This discrepancy can be attributed to the violation of *i.i.d.* assumptions in the IDS alerts. In fact, IDS alerts are not independent and identically distributed. For example, we used 30% of alerts as an initial training for the model, which turned out to contain far more false alerts than the remaining 70%. As a result, the classifier trying to optimize the misclassification cost for $N/P = 5$, did not achieve the optimal result for the more balanced distribution and overly reduced the false-positive rate (which does not contribute as much to the cost as the false-negative rate). Similarly, in the agent mode discarding of false alerts skews the dataset towards real attacks. While this is desirable for the real datasets, it should be accounted for in parameters for the abstaining classifiers.

One possible solution to this problem would be to dynamically build ROC curve describing classifier's performance during the classification process and update parameters for the abstaining classifiers accordingly. While this would increase the run-time and the complexity of our system, it should yield better results. It is an interesting research area that should be further investigated.

### 7.2.4 Conclusions

The experiments performed here confirmed that in the recommender mode ALAC+ significantly reduces the misclassification cost on both datasets. In the agent mode, ALAC+ performed significantly better in terms of misclassification costs on the second dataset and comparably on the first dataset. In all cases, the system significantly reduced the overall number of misclassifications at the price of abstentions. Note that in a cost-sensitive setup, different types of misclassifications bear different costs, hence, even a significant reduction of false negatives with comparable false positives result will not be reflected in the lower misclassification cost.

On the other hand, significantly reducing the total number of all misclassifications alleviates another problem in intrusion detection: As argued by Axelsson [Axe99], *precision* (defined as the probability that a positively classified alert is in fact positive) plays an important role in the efficacy of classification with human analysts: if an alert is classified as positive by the system but in reality has only a very small chance of being positive, the human analyst may well learn from such incidents to ignore all positive alerts classified as positive henceforth. This greatly increases the probability that real attacks will be missed. In all our experiments ALAC+, while reducing the overall cost-sensitive misclassification cost, also significantly improves the precision of classification, which makes real alerts less likely to go unnoticed. This makes it useful for the human analysts.

This confirmed our hypothesis stated in the beginning of the evaluation section that ALAC+ significantly reduces the number of misclassifications and the overall misclassification cost, which makes it particularly useful for intrusion detection.

## 7.3 Summary

In this chapter we presented ALAC+, an alert-processing system using abstaining classifiers introduced in Chapter 6. We showed how to select the runtime parameters for ALAC+ and evaluated it on one real and one synthetic dataset showing that, in many cases, ALAC+ significantly reduces the number of misclassifications and the overall misclassification cost. This makes abstaining classifiers particularly useful for the alert-classification system.

# Chapter 8

# Combining Unsupervised and Supervised Learning

In this chapter we look at unsupervised learning and investigate how it can be used to further facilitate alert management. This chapter builds upon CLARAty, an algorithm developed by Julisch [Jul03b]. The ideas presented here were also published in [PT05].

## 8.1 Why Unsupervised Learning Makes Sense

Julisch observed [Jul01, JD02, Jul03a, Jul03b] that by unsupervised clustering one can find clusters and cluster descriptions that can be interpreted by human analysts to find root causes responsible for large groups of alerts. He applied the algorithms to a set of unlabeled alerts collected by a MSSP and showed that by judiciously removing alerts covered by existing clusters, one can remove up to 70% of future alerts. In this work we take this approach further and analyze how ALAC, our classification framework, can leverage clustering techniques.

This chapter is organized as follows: In the remainder of this section we analyze two main goals of clustering of intrusion detection alerts: (i) retrospective log analysis, and (ii) supporting subsequent classification. We also analyze how clustering can take advantage of alert labels if they are known. Subsequently, in Section 8.2 we present the algorithm CLARAty [Jul03b] used for alert classification. Section 8.3 introduces an automated clustering system, and Section 8.5 introduces a two-stage alert-classification system. Finally, Sections 8.4 and 8.6 present the evaluation of the automated cluster-processing system and the two-stage classification system, respectively.

### 8.1.1 Retrospective Alert Analysis

In this setup, historical alerts are mined for patterns indicative of both benign and malicious activities. By providing concise pattern descriptions covering large groups of alerts, the human analyst can quickly analyze large groups of alerts. The goal of this analysis is threefold:

**Discovering large groups of alerts with benign root causes:** Subsequently, their root causes can be removed or IDS sensors can be tuned.

**Discovering large groups of alerts with malicious root causes:** Subsequently, the analyst can write signatures identifying previously experienced behavior.

**Quality Assurance:** With the concise "compressed" representation of redundant alerts (covered by clusters) the analyst can focus on less prevalent alerts representing events that otherwise may go unnoticed. To illustrate this with an example, if an alert log containing 100000 alerts gets clustered into 20 clusters covering in total 99000 alerts, the analyst will need to investigate only those 20 clusters and then focus on the remaining uncovered 1000 alerts. This leads to a significant workload reduction and ensures higher quality of analysis.

Note that this approach does not take advantage of the fact that it processes historical alerts, which have already been investigated and of which the incidents have been identified. In other words, each alert can be augmented with a label, describing the incident it is a member of (in case of true positives) or another value label (in case of false positives). The question that arises is how the alert-processing system should take advantage of these labels. We have investigated the following two possibilities:

**labels used in the clustering stage:** If alert labels are used by the clustering algorithm, we can ensure that the alerts with different labels are less likely to be clustered together than those with identical labels, or even ensure (by setting appropriate weights or modifying the algorithm) that they will never be clustered together. This will produce purer clusters (i.e., clusters containing only one type of labels), but can lead to problems if the initial labeling was incorrect. This way the "quality assurance" goal of unsupervised learning may not be fulfilled. Alternatively, techniques like the ones used for discovering predictive clustering rules [ŽDS05] can be used.

**labels used only in the post-clustering stage:** On the other hand, if the clustering algorithm is not aware of alert labels, it is more likely that, by investigating less pure clusters, some missclassified alerts can be corrected. However, it is also possible that clusters are overgeneralized and cover alerts with different root causes. In this case, those clusters would need to be investigated by the analyst and further split if necessary. This is a simple form of semi-supervised learning.

In either approach, the obtained clusters can be further labeled depending on the labels of alerts they contain. In particular, we can identify three types of clusters of practical relevance (Figure 8.1):

**False-Alert Clusters (FA-only clusters):** These are the clusters containing only alerts marked as false alerts. These are analogous to clusters with "benign root causes" above. However, this classification can be inferred automatically and no analyst action is required.

**True-Alert Clusters (TA-only clusters):** These are the clusters containing only alerts marked as true alerts. These clusters contain only events with malicious root causes and can be used to construct rules identifying past malicious behavior. Similarly to the previous case this classification can also be inferred automatically.

**Mixed Clusters:** These are potentially the most interesting clusters supporting quality assurance and requiring attention from the analysts. The existence of these clusters can be either attributed to overgeneralized clusters (containing various root causes) or are indicative of misclassified alerts. Investigating the latter case ensures that no malicious activities were missed.

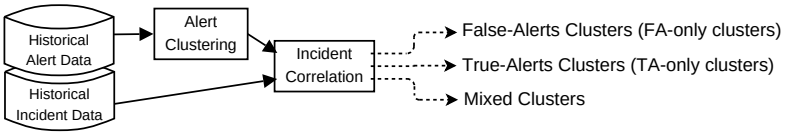



Figure 8.1: Three main types of clusters: false-alert candidates, true-alert candidates, and mixed clusters for further analysis.

### 8.1.2 Subsequent Alert Classification

Julisch showed how clustering can be used to generate human-readable cluster descriptions, which, after being reviewed, can be used to construct filtering rules, leading to up to 70% reduction in the number of subsequent alerts. Note that in this setup the clusters are being judiciously reviewed by the analysts and only the good clusters are used as filtering rules. This setup does not take advantage of the fact that the alerts are labeled. We will refer to it as *semi automated* (Figure 8.2).

The classification is done in the following steps:

1. The clustering algorithm is run on a subset of an alert log $L$ (e.g., alerts collected in the most recent week), yielding a set of clusters $\{P_1, P_2, \ldots, P_n\}$.

2. Clusters $\{P_1, P_2, \ldots, P_n\}$ are investigated by the analyst $\mathcal{O}$.

3. Reviewed clusters are used for investigating network and configuration problems (retrospective alert analysis).

4. Clusters, for which a root cause can be identified are converted to filters $\{F_1, F_2, \ldots, F_m\}$, based on the attributes of alerts they contain.

5. Filters $\{F_1, F_2, \ldots, F_m\}$ are used to remove future alerts from appearing in the console (subsequent alert classification).

6. The procedure is repeated after a predefined time has elapsed (e.g., a week) or a sufficient number of new alerts has been collected in the alert log $L$.

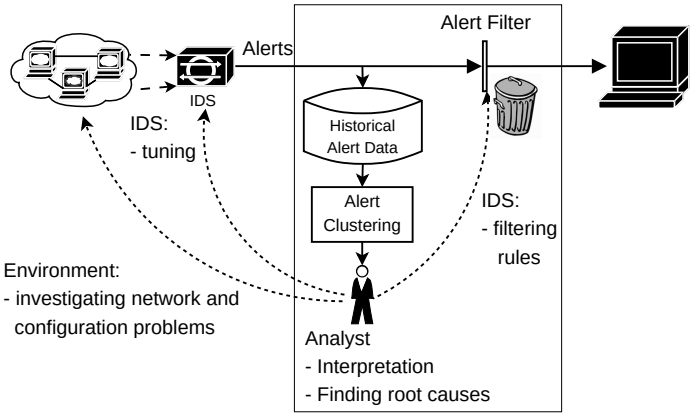



Figure 8.2: Semi-automated cluster processing.

Assuming that the alert labels are given, the above setup can be extended to take advantage of them. Similarly to the previous section, depending whether the alert labels are used in the clustering stage the system will output more or less "incident-oriented" clusters. As we want to get an unbiased evaluation of clustering quality, as well as perform the "quality assurance" function, we decided to use the alert labels only in the post-clustering stage.

In our approach we use the notion of *labeled clusters* (cf. Figure 8.1), which allows us to perform *automated cluster processing*. More specifically, we perform clustering on unlabeled data as previously and label clusters as a whole based on known attacks. False-alert clusters, that is clusters containing only alerts marked as false alerts can then be used to create filtering rules to remove subsequent alerts. Remaining alerts can be subsequently passed to the analyst to form a two-stage alert-classification system [PT05]. Note that it is possible that some of the true alerts are removed in the process, as the analyst will not review the clusters.

Alternatively, cluster information can be used to construct additional features, which will be added to the background knowledge for ALAC. The idea is that the supervised learner will be able to pick "good" cluster features in learning the classification. In Section 8.3 we will investigate both of these approaches.

## 8.2 CLARAty—Algorithm Description

In the clustering stage we use an algorithm called CLARAty [Jul03b], which is based on a modified attribute-oriented induction (AOI) [HCC92, HCC93] algorithm, to make it more suitable for alert processing.

Similarly to ALAC, CLARAty represents alerts as tuples of attribute-value pairs. Alerts can have many different attributes depending on the type of the alert, but in practice, certain basic attributes are always part of the alert, e.g., a timestamp, source and destination addresses, and a description or type of the alert.

### 8.2.1 Generalization Hierarchies

To allow the clustering of alerts in the alert space, a measure of similarity is needed. To define this measure we use our knowledge of the specific environment and general properties. This *background knowledge* is represented through *generalization hierarchies* of the important attributes of the alerts.

For example, the network topology of the specific environment can be captured in a hierarchy of IP address descriptions, which, possibly through multiple levels, generalize a specific IP address into generalized address labels. In this way, specific IP addresses could be labeled according to their role (`Workstation`, `Firewall`, `HTTPServer`), then grouped according to their network location (`Intranet`, `DMZ`, `Internet`, `Subnet1`) with a final top-level generalized address `AnyIP` (see Figure 8.3). When these classification hierarchies are not known, IP addresses can also be generalized according to the hierarchies in the addressing structure. For example, an IP address `195.176.20.45` can be generalized to the corresponding class C network: `195.176.20.0/24`, followed by the class-B generalization `195.176.0.0/16`, class-A generalization `195.0.0.0/8` and finally `AnyIP`.

Other attributes will have different generalization hierarchies, depending on the type and our interests. For example, the source and destination ports of port-oriented IP connections can be generalized into `Privileged` (0-1023) and `NonPrivileged` (1024-65535), with a top-level category of `AnyPort`. In addition, the well-known destination ports (0-1023) can

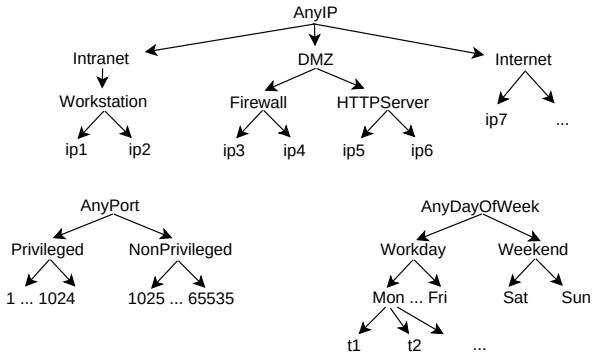



Figure 8.3: Sample generalization hierarchies for address, port and time attributes.

comprise a number of hierarchies describing their function, e.g., `httpPorts` (80, 443, 8080, 9090), `mailPorts` (25, 110, 143, 993, 995), `chatPorts` (194, 258, 531, 994). Similarly, the timestamp attribute could be generalized into `Workday` and `Weekend`, or also `OfficeHours` and `NonOfficeHours`.

Generalization hierarchies like these are static, but dynamic hierarchies are also possible, for example frequently occurring substrings in free-form string attributes (e.g., in context information), which can be generated and evaluated during the runtime of the data mining.

Generalization hierarchies are used for constructing so-called, *generalized alerts*. We will also use the notion of an alert being *covered* by the the generalized alert.

**Definition 13 (Generalized alert)** *Given a generalization hierarchy $\mathcal{G}$, a* generalized alert *$a'$ is derived from an alert $a$ by replacing at least one of its attributes: $T_1, \ldots, T_i, \ldots, T_n$ with a value from a corresponding generalization hierarchy $\mathcal{G}_i$.*

**Definition 14 (Coverage)** *Given a generalization hierarchy $\mathcal{G}$, a* generalized alert *$A'$ covers an alert $A$ iff the values of all the attributes $T_1, \ldots, T_i, \ldots, T_n$ of $A$ can be generalized to corresponding attribute values in $A'$ using $\mathcal{G}_i$.*

### 8.2.2 CLARAty Algorithm

Conceptually, CLARAty algorithm works as follows: Given a large set of alerts $L$ with attributes $T_1, \ldots, T_n$, a generalization hierarchy $\mathcal{G}_i$ for each attribute of the alerts, and parameters $k_{\min}$, CLARAty produces a sequence of clusters (generalized alerts) $\{P_1, P_2, \ldots, P_n\}$.

| | |
|---|---|
| **Given** | – A set of alerts $L$ with attributes $T_1, \ldots, T_n, \ldots$),<br>– Tree-based generalization hierarchies for each attribute $\mathcal{G}_i$,<br>– Constants $k_{\min}$, `MIN_SIZE`. |
| **Find** | A sequence of mutually exclusive clusters (patterns) $P_1, P_2, \ldots, P_k$, describing the properties of alerts, which are useful for the analyst $\mathcal{O}$. |

The clusters are derived using a *sequential covering algorithm* and are mutually disjoint (i.e., one alert can be a member of at most one cluster). However, similarly to evaluating ordered rule sets (cf. Section 5.4) an alert may be covered by more than one generalized alert if the generalized alerts are evaluated independently. CLARAty algorithm consists of the following steps:

**Data**: Set of alerts $L$, tree-based generalization hierarchies $\mathcal{G}_i$, $k_{\min}$, `MIN_SIZE`
**Result**: A sequence of clusters (generalized alerts) $P$.

```
1  ∀A ∈ L : A_count = 1 ;
2  min_size ← k_min · Σ_{A∈L} A_count;
3  C ← ∅;
4  while (min_size ≥ MIN_SIZE) do
5  |  Heuristically select an attribute T_i to generalize ;
6  |  Generalize T_i (Replace the value of attribute T_i with parent value of corresponding
   |  generalization hierarchy G_i in all alerts in L) ;
7  |  Group identical alerts A, A': A_count ← A_count + A'_count, remove A' ;
8  |  if ∃A ∈ L : A_count ≥ min_size then
   |  |  /* found the cluster                                                    */
9  |  |  P ← P ∪ {A};
   |  |  /* remove the alerts from L                                           */
10 |  |  L ← L\{A};
11 |  |  Undo generalizations in L (all actions previously taken in lines 8.2.2 and 8.2.2);
12 |  |  min_size ← k_min · Σ_{A∈L} A_count;
13 |  end
14 end
```

**Algorithm 11**: Clustering algorithm CLARAty [Jul03b].

The control parameter $k_{\min}$ determines the minimal size of the first cluster as a fraction of the total alert size and needs to be specified by the user. This parameter must be chosen carefully: if it is chosen too large, distinct root causes are merged and therefore the result will be overgeneralized. If it is chosen too small, a single root cause might be represented by multiple generalized alerts and the algorithm generates more than necessary clusters and is suboptimal as all those clusters need to be interpreted. The heuristic for selecting the attribute to be generalized (line 8.2.2 in Algorithm 11) calculates value $F_i$ for each attribute and selects the one for which the value $F_i$ is minimal. The value $F_i$ is defined as the maximum of the number of (generalized) alerts with the same values of attribute $T_i$. More formally, $F_i = \max_{v \in Dom(T_i)}(f_i(v))$, where $Dom(T_i)$ is the domain of attribute $T_i$ and $f_i(v) = \sum_{A \in L \wedge A[T_i]=v} A_{count}$.

The algorithm supports weighting implemented as the following heuristic: if all $F_i \geq min\_size$, they are weighted by weights $w_i$, i.e., $F_i \leftarrow F_i \cdot w_i$, $i = 1, \ldots, n$. This way the heuristic is biased towards selecting attributes with small associated weights.

In addition, if the generalization hierarchy is not defined for some attributes, the above algorithm is run independently for each combination of values of nongeneralized attributes. In this case, the final clustering result is a set union of results for those individual runs. To limit the number of clusters the algorithm outputs the user can provide an additional parameter `MAX_CLUSTERS`, which defines the maximum number of clusters generated. In this case, the algorithm needs to store only `MAX_CLUSTERS` biggest clusters during its execution.

This approach focuses on identifying the root causes for large groups of alerts, which typically correspond to problems in the computing infrastructure that lead to numerous false positives (with the potential exception of large-scale automated attacks). It does not look for small, stealthy attacks in the alert logs, but aims at reducing the noise in the raw alerts to make it easier to identify real attacks in the subsequent analysis.

### 8.2.3 Cluster Descriptions and Filtering

Note that in line 8.2.2, CLARAty removes all alerts covered by the cluster from $L$, which means that subsequent clusters will not try to cover those removed alerts. However, it might be the case that the newly devised generalized cluster happens to cover alerts that had already been covered. This is similar to rule-learning algorithms using a sequential covering approach, in which once covered training examples are removed from the training set.

Consequently, when the cluster descriptions are converted to filters, a new alert may be covered by more than one cluster. This raises the question as how we should account for this multiple memberships, when evaluating alert- and cluster coverage.

With both solutions having advantages and disadvantages, in our evaluation we took the "fist-match wins" approach, similar to ordered rules-sets in RIPPER, in which we register only the first cluster match in both filtering and clustering stages. This way each alert belongs to at most one cluster during the clustering stage and at most one cluster during the filtering stage.

## 8.3 Automated Cluster-Processing System

In this section we will discuss an automated cluster-processing system, a system which uses clusters generated by CLARAty, labels them and applies them to future alerts. As discussed previously there are two possible modes of "application" of this alert-processing system: (i) feature-construction mode and (ii) filtering mode.

**Feature-Construction Mode (FC)** In the first mode, the so-called *feature-construction mode* (Figure 8.4), alert cluster membership is evaluated and the matching cluster is subsequently used to construct additional features. Those features can be used by the analyst (or ALAC in a two-stage alert-classification system) to facilitate alert classification. The system works in the following steps:

1. The clustering algorithm is run on a subset of the alert log $L$ (e.g., alerts collected in the most recent week), yielding a set of clusters $\{P_1, P_2, \ldots, P_n\}$.
2. Alerts in the clusters are matched against a set of incidents $\{I\}$, based on which cluster-specific aggregates and features are calculated.
3. Each new alert $A_i$ received by the system is matched against a set of clusters $\{P_1, P_2, \ldots, P_n\}$ and, if it belongs to a cluster $P_i$, the cluster-specific features are associated with it.
4. The alert $A_i$ with extended features is forwarded to the analyst.
5. Clustering (Step 1) is performed again only after a predefined time has elapsed (e.g., a week).

We decided to add the following four features:

**ClusterID:** a unique cluster ID describing the cluster the event falls into, allowing to group events based on this cluster membership.

**IncidCount:** number of distinct incidents the cluster covers during clustering stage. Allows to distinguish incident-free clusters (not any alerts belonging to incidents) and clusters covering incidents.

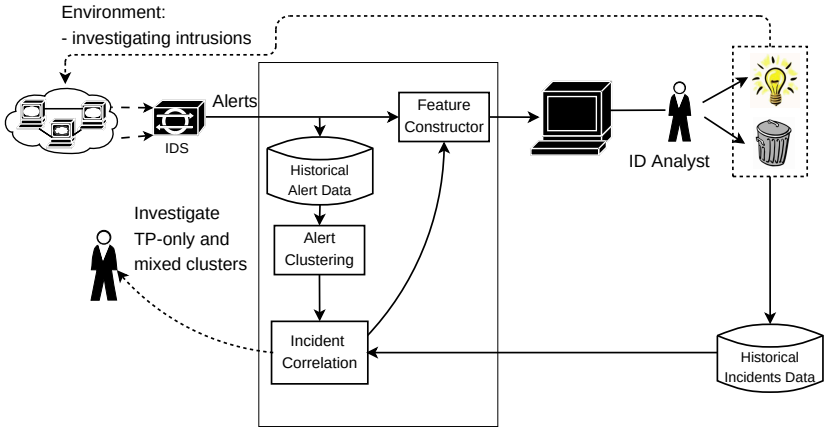



Figure 8.4: Automated cluster processing—creating features.

**IncidEventCount:** number of incident alerts the cluster covers during clustering stage.

**EventCount:** number of all events the cluster covers in the clustering stage. Allows to construct rules based on the cluster size.

Note that in this approach the analyst would have to review exactly the same number of alerts as if CLARAty was not used, however the features increase the certainty that no attacks were missed. This way the feature-construction mode is similar to the recommender mode of ALAC. It provides additional information, which allows to prioritize analyst's work and supports the classification, but does not lead to workload reduction in terms of alerts that need to be reviewed.

**Filtering Mode (FI)** In the second mode, the so-called *filtering mode* (Figure 8.5), clusters that are marked as FA-only cluster are automatically removed and subsequently not passed to the analyst. Assuming that most of the clusters will be marked as FP-only clusters and using the filtering factor of 70% given by Julisch, we can estimate that more than half of the alerts can be removed in this way.

The system works in the following steps:

1. The clustering algorithm is run on a subset of the alert log $L$ (e.g., alerts collected in the most recent week), yielding a set of clusters $\{P_1, P_2, \ldots, P_n\}$.

2. Alerts in the clusters are matched against a set of incidents $\{I\}$, based on which clusters are labeled as: FA-only, TA-only and mixed (cf. Section 8.1.1).

3. Each new alert $A_i$ received by the system is matched against a set of clusters $\{P_1, P_2, \ldots, P_n\}$ and, if it belongs to a cluster $P_i$ labeled as FA-only, the alert is discarded.

4. Otherwise, the alert $A_i$ is forwarded to the analyst unchanged.

5. Clustering (Step 1) is performed again only after a predefined time has elapsed (e.g., a week).

Obviously, all the clusters, including TA-only clusters and mixed clusters, can be independently used for retrospective alert analysis, so that the analyst can better understand

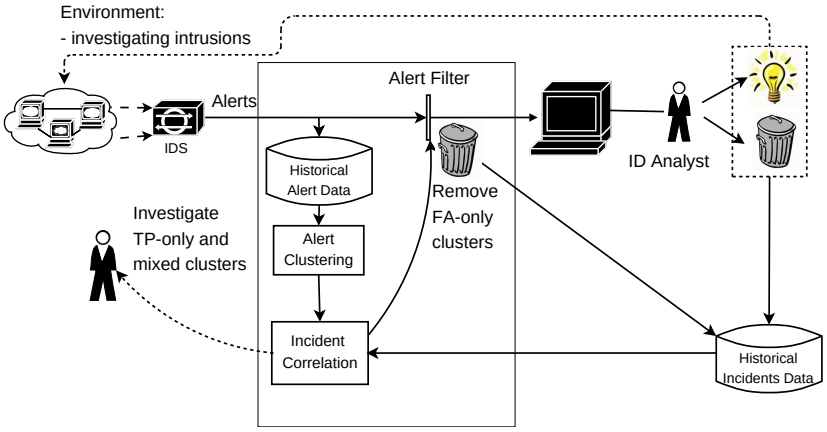



Figure 8.5: Automated cluster processing—filtering.

the characteristics of previously generated alerts and ensure the quality and consistency of alert labeling. As a safety measure, the analyst should also be able to remove some of the FA-only clusters from filtering, so that the potentially unsafe clusters will not be used or, in the extreme case, the analyst would have to approve each cluster that will be used for filtering. This, however, would turn the automated cluster-processing system into the manual one, which we want to avoid.

Our hypothesis is that this automated cluster-processing system is safe and does not discard any security related events. We will verify this hypothesis in Section 8.4, in which we evaluate this system on simulated DARPA 1999 as well as over 13.8 million real alerts.

## 8.4 CLARAty Evaluation

In this section we discuss our experiments with alert clustering, which forms the first stage of a two-stage alert-classification system. These results confirm and extend those of Julisch [Jul03b]. In particular, we validate the concept of an automated cluster-processing system (cf. Section 8.3) and present quantitative results on the safety of cluster filtering and root-cause removal. Experiments in this section are based on three types of datasets: DARPA1999 Data Set, Data Set B and 20 MSSP datasets. The goal of our experiments is to test the following hypothesis:

**Hypothesis 8.4.1** *Unsupervised learning using CLARAty and automated cluster processing framework safe and robust and can significantly reduce the future alert load.*

We will do this in the following four steps:

1. We introduce and evaluate cluster persistency showing that clusters are generally persistent over time.

2. We analyze the relationship between the numbers of clusters and the total coverage in the filtering stage, giving hints on the stopping criteria of the clustering procedure.

3. We introduce cluster precision and cluster recall and clustering precision and recall charts, providing a tool for the evaluation of cluster safety and robustness in an automated cluster-processing system.

4. Perform a case-based analysis of an automated cluster processing system using the above tools.

### 8.4.1 Evaluation Methodology

Julisch proposed the use of clustering to discover the corresponding root causes by human analysts interpreting the clusters. He also claimed that by judiciously removing those root causes one can reduce the volume of future alerts by up to 70%.

Our datasets contain only historical alerts, so we could not remove the underlying root causes to see how this will affect future alerts. Hence, similarly to Julisch, we create filters corresponding to clusters and apply them retroactively to alerts triggered after the clustering took place as shown in Figure 8.6.

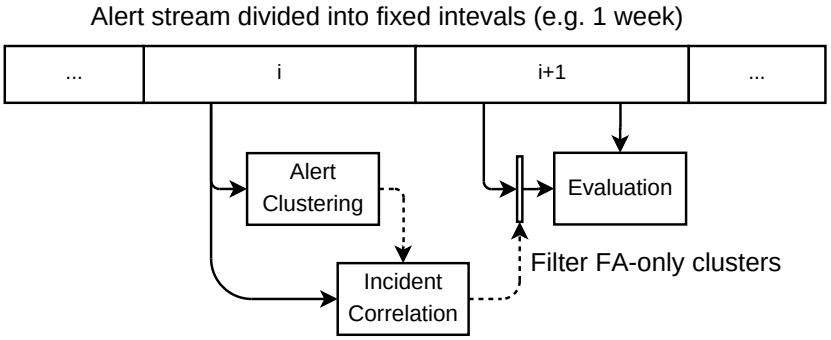



Figure 8.6: The evaluation of alert clustering and filtering.

As clustering produces clusters specific to a particular environment we cluster each of the datasets (and MSSP customers) separately. For clustering we divided the alert stream into fixed time intervals of one week. While longer or shorter time intervals could have been used depending on the environment (Julisch used a time interval of one month), this shorter time interval allows us to be more responsive to the environment changes but, on the other hand, makes it more difficult to detect rare patterns occurring over longer periods of time, e.g., on a weekly basis.

With alerts divided into weekly logs, we performed five clustering runs for DARPA 1999 Data Set and Data Set B and up to 26 runs for each customer from the MSSP dataset, each of the runs producing up to several hundred clusters.

### 8.4.2 Setting System Parameters

For clustering, alerts are represented as tuples of seven attributes, namely: timestamp, source and destination IP addresses, source and destination ports (in case of alerts related to TCP and UDP protocols), alert signature and the packet payload. For clustering we had to define the generalization hierarchies used by CLARAty:

**Source and destination IP addresses:** are generalized into classes of IP addressed (i.e., classC, classB and classA) and also into `inside`, `outside` and `DMZ` and then into `anyIP`.

**Source Port:** is generalized into `privileged` and `non-privileged`, and then into `anyPort`.

**Destination Ports:** is generalized into a number of service-oriented ports (e.g., `httpPorts`, `mailPorts`) and then into `anyPort`.

**Signature:** is generalized into `anySignature`.

Remaining non-categorical attributes, i.e., time and the packet payload are not generalized and only used after the cluster has been generated to describe its properties (e.g., hours and days of the week in case of the time and the longest-common substring in the case of the packet payload).

Recall from Section 8.2 that CLARAty uses two additional numerical parameters: $k_{\min}$ determining the minimum fraction of alerts covered by the first cluster and `MAX_CLUSTERS` determining the maximum number of clusters generated during a single run. Based on experiments not documented here we selected $k_{\min} = 0.05$ and `MAX_CLUSTERS` $= 200$. In the following sections we will investigate how to optimally choose the stopping criteria for the algorithm.

### 8.4.3 Cluster Persistency

More formally, for a given customer $\mathcal{C}$ (one of 20 MSSP customers, DARPA 1999 Data Set, or Data Set B) we perform $n$ clustering runs $\{\mathcal{R}_{\mathcal{C},i}\}$, where $i \in [1; n]$. Each run consists of running the clustering process for the time period $i$, yielding $k_{\mathcal{C},i}$ clusters $\{P_j\}$, where $j \in [1; k_{\mathcal{C},i}]$. Each cluster $P_j$ covers $coverage_c(P_j)$ alerts in the *clustering stage* $(i)$ and is subsequently used as a classifier in the subsequent *filtering stage* $(i+1)$, covering $coverage_f(P_j)$ alerts.

At the first stage, we evaluated the persistency of clusters obtained in all clustering runs. More formally, for a cluster $P_j$ we define the *cluster persistency* $P_{P_j}$ as

$$P_{P_j} = \frac{coverage_f(P_j)}{coverage_c(P_j)} \ . \tag{8.1}$$

Cluster persistency close to 1 describes persistent root causes, while values close to 0 describe ephemeral clusters, which do not have a high predictive value. Finally, values much different from 0 and 1 (either between 0 and 1 or higher than one) describe clusters in a changing environment with varying coverage.

We further define the *average cluster persistency* $P_{\mathcal{R}_{\mathcal{C},i}}$ for clustering run $\mathcal{R}_{\mathcal{C},i}$ as an arithmetic mean of cluster persistencies for a single clustering run:

$$P_{\mathcal{R}_{\mathcal{C},i}} = \frac{\sum_{j=1}^{k_{\mathcal{C},i}} \frac{coverage_f(P_j)}{coverage_c(P_j)}}{k_{\mathcal{C},i}} \ , \tag{8.2}$$

and *average cluster persistency* $P_{\mathcal{C}}$ for customer $\mathcal{C}$ for all the runs. Note that average cluster persistency does not take cluster sizes into account and a large number of small but stable clusters can largely affect $P_{\mathcal{C}}$. Hence we also calculate the overall fraction of alerts covered in both clustering and filtering stage, as shown in Table 8.1.

We also graphically visualize cluster persistency for DARPA 1999 Data Set and Data Set B in Figure. 8.7 and Figures C.1 and C.2 for the MSSP datasets. Arrows in the figures correspond to individual clusters, with the start of an arrow corresponding to the clustering stage and the end of an arrow corresponding to the filtering stage. Clusters are shown cumulatively, i.e., the actual cluster size corresponds to the difference between neighboring

Table 8.1: Cluster persistency $P_{\mathcal{C}}$ for MSSP customers, DARPA 1999 Data Set and Data Set B.

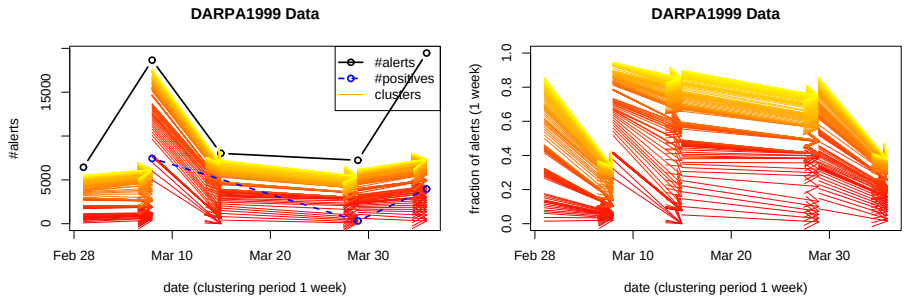
| Customer | $P_{\mathcal{C}}$ | $\pm$ | Fraction covered clustering stage $\frac{\sum_j coverage_c(P_j)}{N+P}$ | Fraction covered filtering stage $\frac{\sum_j coverage_f(P_j)}{N+P}$ |
|---|---|---|---|---|
| 3288 | 1.15 | $\pm$ 0.08 | 0.82 | 0.69 |
| 3359 | 1.07 | $\pm$ 0.05 | 0.75 | 0.50 |
| 3362 | 1.14 | $\pm$ 0.02 | 0.91 | 0.65 |
| 3363 | 1.25 | $\pm$ 0.02 | 0.93 | 0.73 |
| 3380 | 1.25 | $\pm$ 0.07 | 0.95 | 0.54 |
| 3408 | 1.05 | $\pm$ 0.07 | 0.79 | 0.62 |
| 3426 | 1.11 | $\pm$ 0.01 | 0.93 | 0.79 |
| 3473 | 1.01 | $\pm$ 0.01 | 0.91 | 0.84 |
| 3482 | 0.99 | $\pm$ 0.02 | 0.91 | 0.86 |
| 3488 | 1.83 | $\pm$ 0.12 | 0.97 | 0.69 |
| 3491 | 1.11 | $\pm$ 0.07 | 0.88 | 0.65 |
| 3520 | 1.05 | $\pm$ 0.03 | 0.90 | 0.80 |
| 3532 | 1.00 | $\pm$ 0.05 | 0.84 | 0.51 |
| 3565 | 1.08 | $\pm$ 0.02 | 0.94 | 0.85 |
| 3569 | 1.64 | $\pm$ 0.07 | 0.65 | 0.24 |
| 3590 | 2.54 | $\pm$ 0.28 | 0.95 | 0.46 |
| 3626 | 0.97 | $\pm$ 0.04 | 0.88 | 0.79 |
| 3647 | 1.06 | $\pm$ 0.01 | 0.86 | 0.73 |
| 3669 | 1.76 | $\pm$ 0.11 | 0.77 | 0.40 |
| 4043 | 3.16 | $\pm$ 0.31 | 0.94 | 0.68 |
| DARPA 1999 | 1.36 | $\pm$ 0.08 | 0.91 | 0.53 |
| Data Set B | 1.28 | $\pm$ 0.10 | 0.90 | 0.41 |

arrows. The parallel arrows correspond to the clusters with $P_{P_j} = 1$ and clusters with touching ends correspond to the ephemeral clusters, i.e., clusters describing non-recurring events.

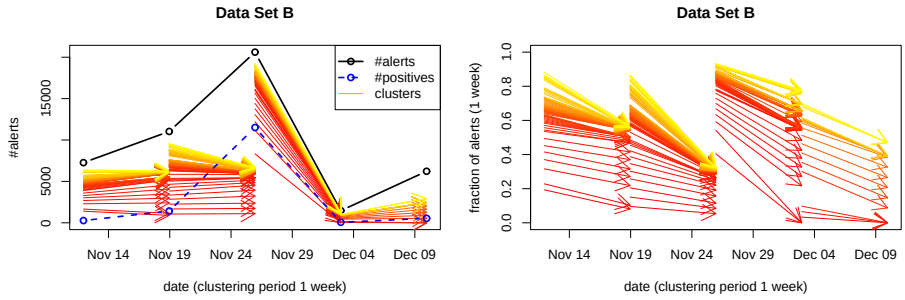
Note that the total number of alerts triggered in consecutive clustering periods can be different, however, both absolute and relative values, shown in Figure 8.7, are meaningful. On one hand, absolute values show clusters corresponding to stable root causes, i.e., root causes that do not change over time, and the resulting cluster covers the same number of alerts in both stages. On the other hand, the size of other clusters can be proportional to the total number of alerts triggered.

To illustrate this with an example, analyzing clusters shown in Figures 8.7c and 8.7d we see the first few clusters for a clustering time period Nov 19–Nov 25 correspond to the persistent root causes and have a similar representation in the following time period Nov 26–Dec 02. Conversely, in the subsequent time period, in addition to the disappearance of two big clusters (attributed to a big scanning incident), we observed a large reduction of traffic, resulting in a proportional reduction of sizes of persistent clusters (parallel arrows in Figure 8.7d).

On average, we observe that for 20 MSSP datasets, 89.6% of all alerts were covered during the clustering stage and, if those clusters were used as filters in the subsequent time period, up to 64.3% of all alerts would be covered. For the other two datasets, we observe a similar high coverage for the clustering stage and lower coverage for the filtering stage: 53% and 41% for DARPA 1999 Data Set and Data Set B, respectively. Note that these figures were




(a) DARPA 1999—absolute values (b) DARPA 1999—relative values




(c) Data Set B—absolute values (d) Data Set B—relative values

Figure 8.7: Cluster persistency for DARPA 1999 Data Set and Data Set B—relative and absolute values. Arrows show cumulative cluster coverage in the clustering (begin of an arrow) and the filtering (end of an arrow) stages for individual clustering runs.

obtained assuming that all clusters will be used in the subsequent alert filtering.

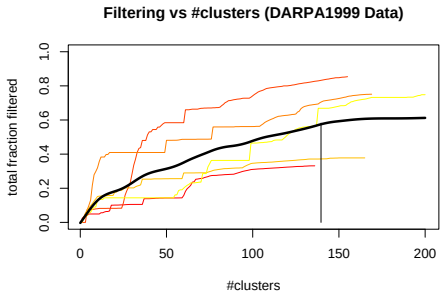
### 8.4.4 Number of Clusters and Total Coverage

In the previous section we assumed that all the clusters will be reviewed and used in the subsequent filtering. However, as the clustering algorithm outputs clusters with decreasing sizes, it is natural to ask if it makes sense to limit the output to only a certain number of clusters. In particular, we would like to answer the following questions:

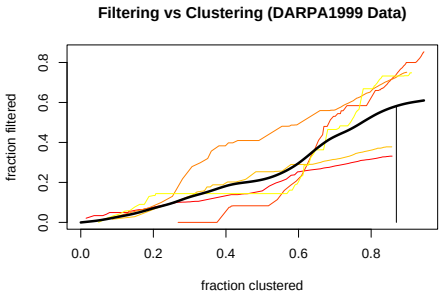
- How much does the relationship clustering coverage vs filtering coverage depend on the dataset used and whether it changes over time?
- Can the cluster coverage in the filtering stage be predicted based on the number of clusters covered?
- Can the cluster coverage in the filtering stage be predicted based on the coverage in the clustering stage?

To answer these questions we plotted the fraction of alerts covered in the filtering stage as a function of both the number of clusters learned and the coverage at the clustering stage shown in Figures 8.8, C.3 and C.4. In the figures the orange curves correspond to individual

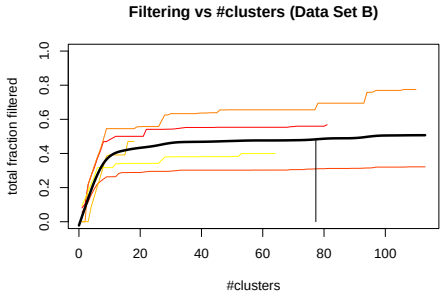
clustering runs and the thick black curve shows the average. A thin black vertical line marks the smallest argument for which the value of the target function reaches 95% of the maximum value.



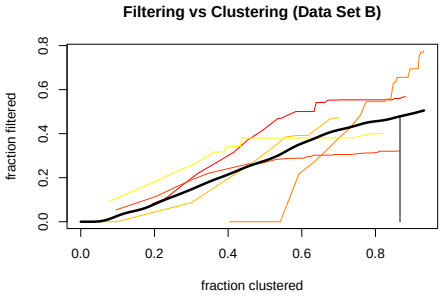

(a) DARPA 1999—fraction filtered vs #clusters




(b) DARPA 1999—fraction filtered vs fraction clustered




(c) Data Set B—fraction filtered vs #clusters




(d) Data Set B—fraction filtered vs fraction clustered

Figure 8.8: Estimating the fraction of alerts clustered and the fraction of alerts filtered as a function of the number of clusters learned. Curves correspond to individual clustering runs. Verticals line show the smallest argument for which the target function reaches 95% of its maximum value.

Interpreting Figures 8.8a, 8.8c and C.3 we see that, although the number of clusters highly depends on the dataset (ranging from 8 to the maximum of 200 clusters), the results are generally similar for consecutive clustering runs for a single dataset (however, there are a few large ephemeral clusters, which are an exception). This means that, once the relationship between clusters and the filtering coverage has been determined for the individual dataset, one can predict how many clusters need to be generated to achieve the desired filtering coverage. This answers our questions stated in the beginning of this section.

Interpreting Figures 8.8b, 8.8d and C.4 we discover a very good linear relationship ($y = 0.74(\pm 0.07)x$, $R^2 \geq 0.95$) between the average fraction of alerts covered in the clustering and the filtering stage. This means that, e.g., to achieve the filtering coverage of 60% one should stop the clustering after covering 81% of all alerts.

### 8.4.5 Automated Cluster Processing

So far we have looked at a quantitative relationship between the number of alerts covered by clusters in the clustering and the filtering stage, which was generated without taking alert labels into account. In this section we will evaluate the automated cluster-processing system introduced in Section 8.3.

More specifically, we use alert labels to label clusters into the following three categories: (i) FA-only clusters (clusters containing only alerts labeled as false alerts), (ii) TA-only clusters (clusters containing alerts only labeled as true alerts), (iii) Mixed clusters (clusters containing both types of alerts).

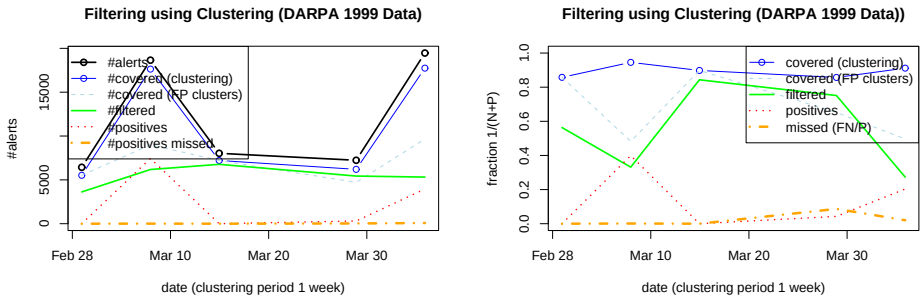
Assuming that the first group of clusters accurately identifies benign root causes, we use only these clusters to create filters in the subsequent filtering stage. Clusters labeled as "mixed" will need to be investigated by analysts and either further split into a number of homogeneous clusters (containing only one type of alerts) or some alerts will be relabeled (in case they were classified incorrectly). Finally, clusters containing only true alerts can be used to write signatures identifying particular types of attack tools or writing concise alert correlation rules. Note that this filtering scheme is conservative, as in reality some of the mixed clusters would be further split and also classified as FA-only clusters. Similarly, some of TA-only clusters may be reclassified as FA-only clusters.

For our evaluation (cf. Figure 8.6) we have the classification of all the alerts, including the future alerts, so we can evaluate how well the automated cluster-processing system would perform in practice. Figures 8.9, C.5 and C.6 show the performance of the system for all clustering runs for all datasets. The dark blue line shows the total number of alerts covered in the clustering stage, and the dotted light blue line shows only those covered alerts that are FA-only clusters. Those false alert clusters (from the preceding clustering run) converted to filters would filter the number of alerts shown by the thick green line. This may result in some true alerts being removed (orange dash-dotted line) out of all the true alerts in the given time period (dotted red line). Note that due to a small number of real incidents, we calculated the fraction of filtered true alerts as the fraction of all true alerts (not all alerts, like the remaining series).

We can clearly see that using FA-only clusters as filters only slightly reduced the filtering coverage (overall 64.3% down to 62.7%), which is expected. However, what is far more interesting is whether this automated filtering is safe, i.e., does not remove any true alerts. The quantitative answer to this question is shown in Figure 8.10, where we can see that for the DARPA 1999 Data Set, the system, reducing the number of alerts by more than a half, managed to filter a small number (about 1%) of true alerts. For Data Set B, the system yielded a slightly smaller alerts reduction, but did not remove any true alerts.

These results are strongly encouraging, as the reduction of the alerts volume by 50% is achieved at the cost of zero, or very few misclassifications. However, MSSP datasets (Figure 8.10d) show that in reality many more true alerts can be filtered out. For example, using this system for the customer 3473 would lead to almost all true alerts being removed! Similarly, another five customers, namely 3482, 3520, 3565, 3569 and 3669 would miss significantly more than 1% of all their true alerts, which is clearly unacceptable.

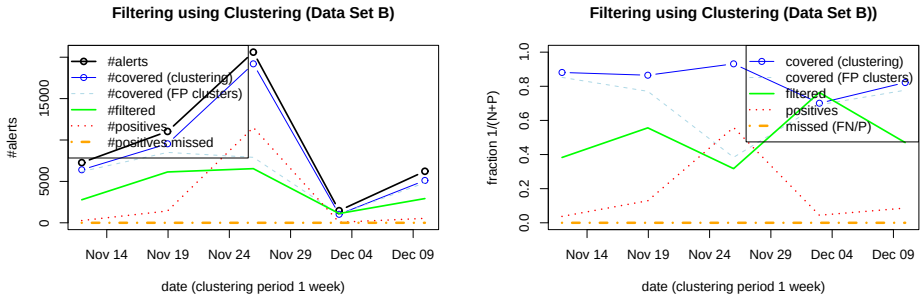
In the next section we will look at clustering precision and recall in more detail and will investigate why, in some cases, clustering can lead to the removal of true alerts.




(a) DARPA 1999—absolute values




(b) DARPA 1999—relative values




(c) Data Set B—absolute values




(d) Data Set B—relative values

Figure 8.9: Clusters as filters for DARPA 1999 Data Set and Data Set B—relative and absolute values. Missed positives in Figures 8.9b and 8.9d are calculated relative to the number of true alerts ($P$).

### 8.4.6 Cluster Precision and Recall

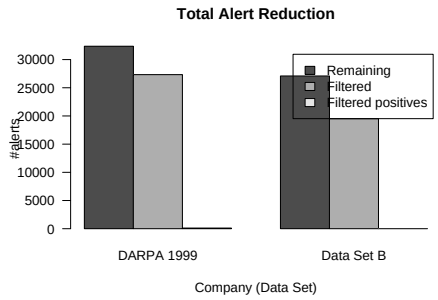
Similarly to information retrieval, we define *clustering precision* ($CP$) as the fraction of alerts covered by clusters and *clustering recall* ($CR$) as the fraction of true alerts covered by clusters*.

$$CP = \frac{|\text{false alerts covered by clusters}|}{|\text{alerts covered by clusters}|} \tag{8.3}$$

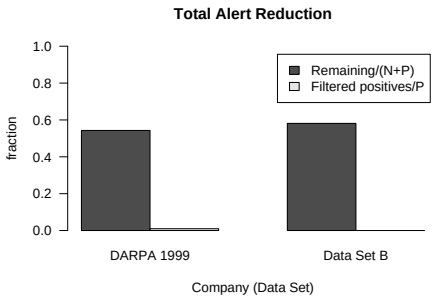
$$CR = \frac{|\text{true alerts covered by clusters}|}{|\text{true alerts}|} \tag{8.4}$$

Table 8.2 shows the clustering precision for clusters obtained in the clustering stage for all datasets. In addition to the number of clusters we also show the total fraction of alerts covered by those clusters. For example, DARPA 1999 Data Set generated 893 clusters for the entire experiment, 735 of which (covering 66% of clustered alerts) were FA-only clusters, 38 (covering 16% of alerts) were TA-only clusters and the remaining 120 clusters (covering 17% of clustered alerts) were mixed clusters. This means that only 735 clusters would be used for the filtering stage.

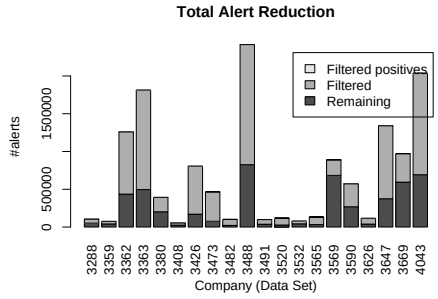
*This working definition is not entirely correct as the precision pertains to false alerts and recall pertains to true alerts. We nonetheless consider it intuitive and use it qualitatively in this section.



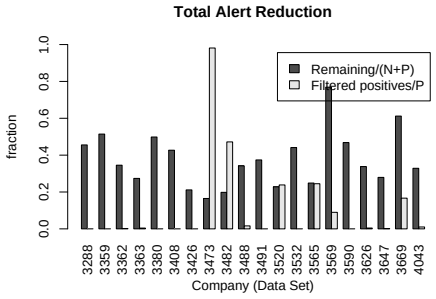

(a) Total alert reduction (DARPA 1999 and Data Set B)—absolute values




(b) Total alert reduction (DARPA1999 and Data Set B)—relative values




(c) Total alert reduction for MSSP datasets—absolute values




(d) Total alert reduction for MSSP datasets—relative values

Figure 8.10: Total alert reduction for clusters as filters for all datasets—relative and absolute values.

We observe that for DARPA 1999 Data Set and Data Set B, FA-only clusters account for a much smaller, compared to MSSP datasets, fraction of clusters (and alerts). The reason for this is that the former datasets contain much higher percentage of intrusions then the latter dataset.

Similarly, Table 8.3 shows clustering precision applied in the filtering stage, in which we used clusters generated in the clustering stage and applied them to alerts for the subsequent time interval. Similarly to the previous stage, we classified the clusters into the three groups depending on the classification of alerts they contain. Note that in this process we used all the clusters, not only FA-only clusters, which would be used by the automated cluster-processing system. Moreover, we did not include clusters that has not covered any alerts in the filtering stage, hence the cumulative number of clusters in a row is smaller than in Table 8.2.

We clearly see that, while FA-only clusters cover approximately the same fraction of alerts, both TA-only clusters and mixed clusters cover substantially less alerts than in the clustering stage. This is intuitive as benign root causes, responsible for FA-only clusters, are generally more persistent than those responsible for true alerts.

The two additional columns in Table 8.3 show the number of clusters and alerts that would be incorrectly removed in the filtering stage. Continuing with the example with DARPA 1999 Data Set, 735 FA-only clusters would be applied in the filtering stage, resulting in the removal of 115 true alerts. Those 115 are contained in 30 clusters, which means that those 30 clusters

Table 8.2: Clustering precision for the clustering stage—the cumulative number of clusters containing only false alerts, only true alerts and mixed clusters. Right columns show the distribution of the alerts among the three cluster types.

| Company | #FA-only clusters | Fraction alerts | #TA-only clusters | Fraction alerts | #Mixed clusters | Fraction alerts |
|---|---|---|---|---|---|---|
| 3288 | 724 | 0.96 | 5 | 0.00 | 94 | 0.04 |
| 3359 | 1016 | 0.97 | 4 | 0.00 | 22 | 0.03 |
| 3362 | 5182 | 1.00 | 0 | 0.00 | 70 | 0.00 |
| 3363 | 5161 | 1.00 | 10 | 0.00 | 61 | 0.00 |
| 3380 | 1632 | 0.91 | 0 | 0.00 | 30 | 0.09 |
| 3408 | 781 | 0.96 | 1 | 0.00 | 36 | 0.04 |
| 3426 | 5056 | 1.00 | 3 | 0.00 | 10 | 0.00 |
| 3473 | 4398 | 1.00 | 0 | 0.00 | 83 | 0.00 |
| 3482 | 1239 | 0.94 | 0 | 0.00 | 488 | 0.06 |
| 3488 | 5020 | 1.00 | 0 | 0.00 | 18 | 0.00 |
| 3491 | 1121 | 0.99 | 0 | 0.00 | 27 | 0.01 |
| 3520 | 906 | 0.99 | 9 | 0.01 | 6 | 0.00 |
| 3532 | 889 | 0.86 | 0 | 0.00 | 175 | 0.14 |
| 3565 | 1185 | 0.88 | 0 | 0.00 | 277 | 0.12 |
| 3569 | 4372 | 0.88 | 1 | 0.00 | 523 | 0.11 |
| 3590 | 2658 | 0.97 | 0 | 0.00 | 82 | 0.03 |
| 3626 | 387 | 1.00 | 13 | 0.00 | 30 | 0.00 |
| 3647 | 5137 | 0.99 | 14 | 0.00 | 111 | 0.01 |
| 3669 | 5135 | 0.99 | 2 | 0.00 | 127 | 0.01 |
| 4043 | 4299 | 0.85 | 0 | 0.00 | 391 | 0.15 |
| DARPA 1999 | 735 | 0.66 | 38 | 0.16 | 120 | 0.17 |
| Data Set B | 363 | 0.69 | 20 | 0.11 | 7 | 0.20 |

from the clustering stage were either too general (covering alerts with different root causes) or incorrectly classified.

Recall, that MSSP datasets and DARPA 1999 Data Sets not only contain information about the classification of alerts, but also contain information about incidents they are members of. In the case of MSSP dataset, these are incidents grouped by security analysts and, in the case of DARPA 1999 Data Set, the incidents are individual attacks launched in the simulated environment.

Table 8.4 shows the clustering recall, describing the fraction of incidents covered by clusters. We look at the following three parameters: (i) average fraction of an incident covered by clusters, (ii) total fraction of all incidents covered, and (iii) the average number of clusters covering an incident, in both clustering and filtering stage. The interpretation of the parameters is the following: The average incident coverage shows whether and how completely incidents are covered by clusters. Ideally, we want this value to be either close to 0 or close to 1. In the first case, the incidents are not covered at all (supporting the hypothesis that only false alerts are clustered). In the second case, the incident is covered completely (allowing for the derivation of rules for true alerts). Finally, the average number of clusters per incident (in the last column) shows how specific the clusters are. This value should be close to 1 as this is the case, in which clusters accurately model the intrusions.

Based on the clustering recall for both clustering and the filtering stages shown in Table 8.4, we make the following observations:

Table 8.3: Clustering precision for the filtering stage. Columns are identical to those in Table 8.2. Two additional columns show the number of clusters containing false negatives (erroneously removed alerts relating to incidents) and the absolute number of false negatives.

| Company | #FA-only clusters | Fraction alerts | #TA-only clusters | Fraction alerts | Mixed clusters | Fraction alerts | **Missed (FN) clusters** | **Missed (FN) events** |
|---|---|---|---|---|---|---|---|---|
| 3288 | 401 | 0.96 | 0 | 0.00 | 37 | 0.04 | **0** | **0** |
| 3359 | 409 | 1.00 | 0 | 0.00 | 0 | 0.00 | **0** | **0** |
| 3362 | 2816 | 1.00 | 1 | 0.00 | 0 | 0.00 | **1** | **1** |
| 3363 | 3335 | 1.00 | 0 | 0.00 | 8 | 0.00 | **4** | **9** |
| 3380 | 783 | 1.00 | 0 | 0.00 | 0 | 0.00 | **0** | **0** |
| 3408 | 496 | 1.00 | 0 | 0.00 | 0 | 0.00 | **0** | **0** |
| 3426 | 3190 | 1.00 | 0 | 0.00 | 0 | 0.00 | **0** | **0** |
| 3473 | 3075 | 1.00 | 0 | 0.00 | 83 | 0.00 | **4** | **105** |
| 3482 | 1004 | 0.92 | 0 | 0.00 | 447 | 0.08 | **28** | **469** |
| 3488 | 2771 | 1.00 | 0 | 0.00 | 6 | 0.00 | **3** | **4** |
| 3491 | 672 | 1.00 | 2 | 0.00 | 6 | 0.00 | **0** | **0** |
| 3520 | 640 | 0.99 | 3 | 0.01 | 4 | 0.00 | **2** | **317** |
| 3532 | 448 | 0.97 | 0 | 0.00 | 22 | 0.03 | **0** | **0** |
| 3565 | 995 | 0.86 | 0 | 0.00 | 277 | 0.14 | **12** | **371** |
| 3569 | 2499 | 0.87 | 1 | 0.00 | 99 | 0.13 | **24** | **1279** |
| 3590 | 1865 | 1.00 | 0 | 0.00 | 0 | 0.00 | **0** | **0** |
| 3626 | 285 | 1.00 | 0 | 0.00 | 22 | 0.00 | **1** | **2** |
| 3647 | 3729 | 0.99 | 0 | 0.00 | 26 | 0.01 | **2** | **10** |
| 3669 | 3760 | 0.99 | 0 | 0.00 | 42 | 0.01 | **21** | **123** |
| 4043 | 2464 | 1.00 | 0 | 0.00 | 38 | 0.00 | **7** | **54** |
| DARPA 1999 | 470 | 0.84 | 2 | 0.00 | 78 | 0.16 | **30** | **115** |
| Data Set B | 102 | 0.98 | 11 | 0.02 | 0 | 0.00 | **0** | **0** |

- Most of the incident events are covered in the clustering stage. The average incident coverage is lower than the fraction of incidents covered as small incidents tend not to be covered.

- Very few incidents are covered in the filtering stage, with an exception of customers 3473, 3482 and 3565, for which most of the incidents were covered. Note that these are those customers for which most of the true alerts would be filtered out. This suggests that they are covered by big overgeneralized clusters.

- The average number of clusters covering an incident is very high for some companies (e.g., 20.20 for customer 3552, 14.00 for customer 3590), suggesting that the incidents are not modeled accurately by clusters.

### 8.4.7 Clustering Precision and Recall Charts

Having analyzed clustering precision and recall, we conclude that cumulative information is sometimes insufficient to describe clustering characteristics. Moreover, we would like to investigate individual clusters that lead to the misclassifications of true alerts, to investigate what kind of alerts would have been missed and why. To this end we plotted clustering

Table 8.4: Clustering recall—clustering and filtering stage.

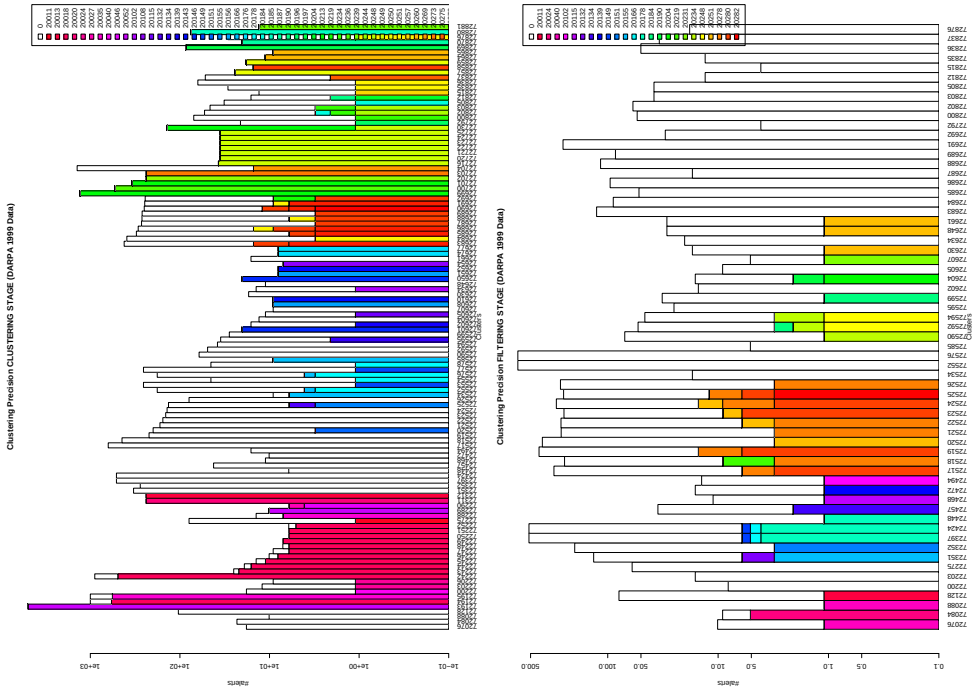
| Customer | Average incident coverage (clustering) | Fraction of all incident events covered (clustering) | Average incident coverage (filtering) | Fraction of all incident events covered (filtering) | Average #clusters covering an incident (clustering) | Average #clusters covering an incident (filtering) |
|---|---|---|---|---|---|---|
| 3288 | 0.56 | 0.50 | 0.18 | 0.36 | 6.80 | 2.50 |
| 3359 | 0.21 | 0.28 | 0.00 | 0.00 | 1.25 | 0.00 |
| 3362 | 0.53 | 0.63 | 0.00 | 0.00 | 2.48 | 0.05 |
| 3363 | 0.12 | 0.49 | 0.00 | 0.00 | 1.50 | 0.14 |
| 3380 | 0.29 | 0.71 | 0.00 | 0.00 | 2.29 | 0.00 |
| 3408 | 0.35 | 0.57 | 0.00 | 0.00 | 9.50 | 0.00 |
| 3426 | 0.12 | 0.34 | 0.00 | 0.00 | 0.89 | 0.00 |
| 3473 | 0.93 | 0.98 | 0.93 | 0.98 | 2.72 | 2.72 |
| 3482 | 0.91 | 0.93 | 0.85 | 0.92 | 9.45 | 8.74 |
| 3488 | 0.06 | 0.30 | 0.00 | 0.02 | 0.26 | 0.09 |
| 3491 | 0.58 | 0.75 | 0.12 | 0.07 | 2.00 | 0.63 |
| 3520 | 0.68 | 0.80 | 0.56 | 0.66 | 2.00 | 0.83 |
| 3532 | 0.73 | 0.87 | 0.13 | 0.06 | 20.20 | 2.40 |
| 3565 | 0.96 | 0.95 | 0.96 | 0.95 | 8.80 | 8.80 |
| 3569 | 0.48 | 0.60 | 0.18 | 0.11 | 12.41 | 2.36 |
| 3590 | 0.75 | 0.99 | 0.00 | 0.00 | 14.00 | 0.00 |
| 3626 | 0.28 | 0.73 | 0.10 | 0.16 | 4.67 | 1.83 |
| 3647 | 0.45 | 0.70 | 0.10 | 0.15 | 5.19 | 1.06 |
| 3669 | 0.64 | 0.73 | 0.27 | 0.17 | 9.43 | 3.00 |
| 4043 | 0.95 | 0.98 | 0.30 | 0.04 | 12.33 | 1.33 |
| DARPA 1999 | 0.30 | 0.92 | 0.13 | 0.01 | 0.77 | 0.35 |
| Data Set B | 0.60 | 0.93 | 0.11 | 0.03 | 5.20 | 2.20 |

precision and recall charts shown in Figures 8.11 (DARPA 1999 Data Set) and C.7–C.10 (MSSP datasets).

*Clustering precision charts* are stacked bar charts, with X-axes labeled as clusters and Y-axes labeled as the number of alerts. Bars at each cluster contain incidents (colored bars, each color represents a different incident) or non-incidents (white bars). Note that because most of the clusters contain only false alerts, both in the clustering and the filtering stage, we omitted them in the charts. This means that clusters plotted in the precision charts are only those clusters that contain at least one alert classified as true alert, at either the clustering or the filtering stage.

Similarly, *clustering recall charts* are stacked bar chars with X-axes labeled as incidents and Y-labels labeled as the number of alerts. Bars at each clusters contain clusters (marked with colored bars, each color represents a different cluster) that a given part of incident is covered with. The non-covered part of each incident is marked with white bars.

Clustering precision and recall charts provide valuable means for rapid alert analysis. For example, analyzing Figure 8.11a we immediately see that the majority of clusters are pure, i.e., contain exclusively alerts marked as incidents or non-incidents. Analyzing three of those non-pure clusters, namely clusters 72194 (cf. Figure 8.12), 72195, 72242, we discovered that the alerts that were not classified as intrusions belong in fact to a part of long-lasting portsweep attacks (training set attacks #8 and #25). This way, by analyzing clustering precision charts, we managed to detect and correct some misclassified alerts.




(a) Precision—clustering stage

(b) Precision—filtering stage

(c) Recall—clustering stage

(d) Recall—filtering stage

Figure 8.11: Clustering and filtering precision and recall for DARPA 1999 Data Set. Data shown cumulatively for all clustering runs, with FA-clusters suppressed.

```
Cluster of size 1000:
  timeDate   == hour: [15 - 16];  days of the week: THU;  days of the month: [11 - 11];
 srcIP      == 207.103.80.104 :: <1,1,1,1,1>  207.103.80.104[100.00%]
 srcWhere   == ip_other       :: <1,1,1,1,1>  ip_other[100.00%]
 srcPort    == 0              :: <1,1,1,1,1>  0[100.00%]
 dstIP      == 172.16.114.50  :: <1,1,1,1,1>  172.16.114.50[100.00%]
 dstWhere   == ip_inside      :: <1,1,1,1,1>  ip_inside[100.00%]
 dstPort    == 0              :: <1,1,1,1,1>  0[100.00%],  HGen: 0
 signature  == ICMP PING (Undefined Code!)
 context    == any
```

Figure 8.12: Cluster 72194, describing a part of a `portsweep` attack.

On the other hand, looking at clustering recall charts, we can see how accurately incidents are modeled by clusters. For example DARPA attack #8, corresponding to incident 20018 in Figure 8.11c, shows that this single incident was covered by a number of clusters, including clusters 72194 and 72242 analyzed previously. On the other hand, most of the other incidents are covered by a single cluster, or not covered at all.

We can also compare precision and recall charts for clustering and filtering stages. Recall that charts for the filtering stage (Figures 8.11d and C.10) show that most incidents are not covered by clusters, which is to be expected as incidents are rare and are unlikely to be covered by clusters from the preceding week. However, we notice, that there is a number of clusters that also cover a fraction of incidents. This is confirmed by the clustering precision chart (Figure 8.11b), which shows that there were clusters with sizes of several hundred alerts which also included some alerts belonging to incidents. This led to 30 clusters, containing 115 true alerts (cf. Table 8.3), being erroneously removed by the cluster-processing system.

### Investigating Missed Alerts for DARPA 1999 Data Set

Investigating these 115 alerts that would be mistakenly removed (together with 30 clusters they are members of) by the cluster-processing system we discover that those alerts belong to the following 24 incidents: training attacks #1, #14, #30, test attacks #2, #15, #32, #34, #39, #49, #55, #55, #66, #78, #84, #90, #104, #119, #131, #134, #148, #151, #178, #180, #182.

We further discover that in 18 of those incidents, the system not only captured all intrusions, but identified alerts incorrectly classified as malicious! In fact, 13 of those incidents, namely test attack #2, #15, #32, #34, #39, #51, #55, #84, #90, #104, #148, #151, #151, are incidents that can only be detected by a host-based IDS, not Snort, and the remaining five incidents, namely test attack #119, #131, #134, #180, #182, were not detected by Snort, because it did not have signatures for these attacks.

In all these cases, alerts incorrectly classified as true alerts were a part of background noise that happened to occur when the attacks took place and were therefore classified as belonging to an incident by our incident correlation tool. In reality, if the attacks were detected, those alerts would be also investigated by the security analyst to confirm their benign nature and this classification would not be considered incorrect. However, we know that in our case the attacks were not detected and those misclassified alerts are completely unrelated.

This leaves us with the remaining six incidents, namely training attack #1, #14, #30

and test attack #49, #66, #77, for which the system actually removed true alerts. All those incidents are scanning incidents (`ipsweep` and `NTinfoscan`) and the missed alerts are scanning alerts, which tend to have unusually high false-positive rates. In addition, analyzing the clustering recall chart for the filtering stage (Figure 8.11d) we see that none of the six incidents were covered entirely by the removed clusters, which means that those incidents would be nonetheless detected. In practice, intrusion detection is a multi-stage process (also known as multi-stage classification [Sen05]), in which after an incident has been detected, all incidents it comprises are found using link-mining (analyzing related alerts). This means that even if those alerts were removed, the incidents they belong to would not have been missed, and they are likely to be rediscovered in the forensic stage.

### Investigating Missed Alerts for MSSP Datasets

In the previous section we showed that true alerts removed by the automated alert clustering system from DARPA 1999 Data Set were, in fact, either false alerts that had been incorrectly classified as true alerts or, in the case of scanning alerts, they were only a part of a scanning incident that was extremely similar to the background traffic. In this section we will take a look at 12 MSSP customers (cf. Table 8.3) for which true alerts were removed, to support these findings and show the robustness of the automatic cluster-processing system.

For each customer, our investigation follows the following procedure:

1. List clusters that were incorrectly removed, together with incidents they cover.
2. For each incident: List all alerts constituting the incident, conclude if the incident would have been missed.
3. For each cluster: List events covered in the clustering and the filtering stage, analyze why the cluster was classified as a FA-only cluster in the clustering stage.

**Customer 3362** Missed cluster contains a `Windows LSASS RPC Overflow` alerts, which constitutes a part of `Malware Infection Incident`. As only a part of this incident is being removed, the whole incident would not have been missed.

We further investigate that the cluster was not included as a part of a `Malware Infection Incident` in the preceding week (clustering stage), although the incident itself was detected by security analysts. By retroactive cluster analysis, an appropriate rule can be written so that similar incidents are automatically detected and none of the events comprising it are missed.

**Customer 3363** Similarly to the previous case, we discover similar clusters with nine `Windows LSASS RPC Overflow` alerts, constituting a `Malware Infection Incident`, which should have been detected in the preceding week. The incident would not be missed as only part of is was removed.

Similarly to the previous case, by retroactive cluster analysis, an appropriate rule can be written so that similar malware infections are automatically detected.

**Customer 3473** Clusters that were filtered out constitute two types of alerts: `HSRP Default Password`, `SQL SSRP StackBo` classified by the analysts as `Suspected False Positive` and `Other`. We cannot determine with certainty what the actual root cause was, but the clusters

covered alerts similar to those that were not given any classification in the preceding week. Therefore we consider the alert a false alert and assume that no true alerts were missed.

**Customer 3482** For this customer, 28 clusters were removed, all of which contain alerts with only one signature `SQL_SSRP_StackBo`. These clusters were classified as a number of incidents: `Other`, `Suspicious Activity`, `Incoming Call`, `Endpoint Troubleshooting` and `Dos Denial of Service`, however, only in addition with some other alerts.

Similarly, to the previous case we cannot determine the root cause of the `SQL_SSRP_StackBo`, however it is likely to be a background noise, not constituting any security threats. For example, the clusters that were filtered out covered approximately 6000 alerts in the clustering stage and were triggered by the same signature. None of these alerts was classified as a true alert.

**Customer 3488** Removed clusters constitute a single incident comprising of four types of alerts: `Windows Registry Access`, `Windows RPC Race Condition Exploitation` and `Miscellaneous` and `TCP SYN Port Sweep` most likely indicating a worm infection. However, in the preceding week, there have been more than 100 alerts with a very similar characteristics, however, we have no record of identified incidents during that time period. If our incident data is complete, retroactive alert analysis suggests that this could be a missed incident.

**Customer 3520 and 3569** Removed clusters contain alerts classified as `Approved Vulnerability Scan`. In the preceding week, there have been identical events that did not receive any classification in the system. For consistency, both events should be either marked or not marked at all.

**Customer 3565** Removed clusters contain alerts classified as `Suspected False Positive`, however similar events in the preceding week did not receive any classification.

For consistency, both events should be either marked as incidents or not marked at all.

**Customer 3626** The cluster removes some of the events classified as `Recon - General Vulnerability Scan`. In the preceding week there has been a number of unsuccessful worm infection attempts, bearing similar characteristics, with no classification. In this case a single cluster covered events related to two different root causes: unsuccessful worm infection attempts on the external server and a vulnerability scan, also from an external address. As only a part of the incident was removed, the incident would not have been missed.

**Customer 3647** We found out the cluster contained alerts with an intent guessing IDS signature `IDS Evasive Double Encoding`, which seems to be also triggering on a misconfigured web server. In the filtering stage, the system removed events triggered by the same signature constituting an approved vulnerability scan.

**Customer 3669** Removed clusters constitute three different groups of incidents: (i) suspected false positives, (ii) approved vulnerability scans and (iii) Denial of Service attack (DoS). The approved vulnerability scan would be removed, as similar scans in the preceding week did not receive any classification. In the case of the Denial of Service attack, there have

been similar activities in the preceding week that did not receive any classification and can be a missed incident.

**Customer 4043** Removed clusters cover a number of alerts classified as `Malware Infection`, which after being removed makes the entire incident likely to be missed. However, it turns out that there were similar alerts in the preceding clustering period, namely `Windows LSASS RPC Overflow`, `Windows RPC DCOM Overflow` and `Windows_SMB_RPC_NoOp_Sled` alerts, with no classification, which suggest that there was a similar infection that has not been identified. However, closer investigation reveals that the previous malware infections were in fact identified, although not all relevant alerts were included in those incidents. By retroactive cluster analysis, an appropriate rule can be written so that similar incidents are automatically detected and none of the events comprising it are missed.

To summarize, our retroactive analysis revealed that, in two cases, namely for customers 3363 and 3488, there has likely been a worm/malware infection that was missed in the week preceding the filtering stage, thus leading to the subsequent removal of similar alerts. In another two cases, namely for customers 3362 and 4043, the incidents were detected on time, although they did not include all events comprising them, thus this would result in the subsequent removal of similar alerts in the filtering stage by CLARAty.

Another five cases of "missed" true alerts are incidents, such as approved vulnerability scans and suspected false positives that are not real security incidents. The events would be filtered out as there have been similar events in the preceding clustering period that did not receive any classification.

Finally, there are the last two cases, namely for customers 3626 and 3647, for which background traffic contained alerts similar to malicious activities. In the first case, a number of worm infection attempts, removed one event constituting an approved vulnerability scan and, in the second case, a misconfigured web server triggered an alert similar to vulnerability scanning.

### 8.4.8 Conclusions

In this section we applied CLARAty, a clustering algorithm developed by Julisch [Jul03b] to our evaluation datasets, DARPA 1999 Data Set, Data Set B and MSSP datasets, confirming that on average up to 63% of future alerts can be covered by clusters, hence allowing for a similar reduction in the number of future alerts.

We showed that the cluster filtering coverage ($coverage_f(P_j)$) can be very well predicted based on the cluster clustering coverage ($coverage_c(P_j)$), providing easy guidance on how to set the stopping criteria for the algorithm. We also introduced an automated cluster-processing system that allows the safe removal benign clusters, without the need for analyzing them manually. We defined clustering precision and recall, analyzed them quantitatively, and showed how clustering precision and recall charts can be used for fast cluster analysis.

Our initial results showed that (cf. Figures 8.10b, 8.10d and Table 8.3) unacceptably high rates of true alerts could be missed, undermining the usability of the automated cluster-processing system. We investigated the cases of those missed alerts and found out that they were due to, either false alerts being incorrectly classified as true alerts, or incidents missed in the preceding time interval. Should those events be classified correctly, the automated cluster-processing system would not remove any true alerts.

This means that the system not only captured all true alerts but also allowed us to detect inconsistencies in alert labeling and, in four cases, detected incidents that were previously missed.

We also showed how analyzing clustering precision and recall charts can be used to retroactively analyze clusters and past alerts, thus helping to assess the quality of human classification and make sure that no incidents were previously missed.

We would like to emphasize that the results here were obtained on a variety of datasets, including simulated datasets, as well as over 13.8 million real alerts analyzed by real security analysts. This shows the safety and robustness of the system and confirms its usefulness in real environments, which confirms our hypothesis stated in the beginning of the evaluation section.

## 8.5 Combining Clustering with ALAC in a Two-Stage Alert-Classification System

As proposed in [PT05], CLARAty can be cascaded with ALAC to form a two-stage alert-classification system. Essentially, the two-stage alert-classification system is a cascade of an automated cluster-processing system (Figures 8.4 and 8.5) combined with ALAC (or ALAC+).

From the problem specification such a two-stage system performs the same function as the one-stage system, and addresses the same problem definition with the utility function $\mathcal{U}$ (cf. Sections 1.2, 5.1 and 7.1), i.e., minimizing misclassification cost and analyst's workload possibly allowing for abstentions.

The system can work both in the feature-construction mode, in which clusters provided by CLARAty are used as additional features for ALAC learning, as well as in the filtering mode, in which some false positives will be removed prior to being classified by ALAC. This second mode of operation is particularly interesting, as CLARAty can effectively remove those false positives that are "easy" to classify, thus allowing ALAC to focus on the remaining ones and also reducing the "workload component" of the utility function $\mathcal{U}$. In addition, the removal of false positives in the first stage effectively changes the class distribution in favor of true alerts, which is desirable for the machine-learning techniques used in ALAC.

We will evaluate the performance of this system in both modes in the following section.

## 8.6 CLARAty and ALAC Evaluation

Having shown that if alert labels are correct, our automatic cluster-processing system using CLARAty is an efficient method of removing on average 63% of false positives, we would like to evaluate a two-stage alert-classification system using CLARAty and ALAC and test the following hypothesis:

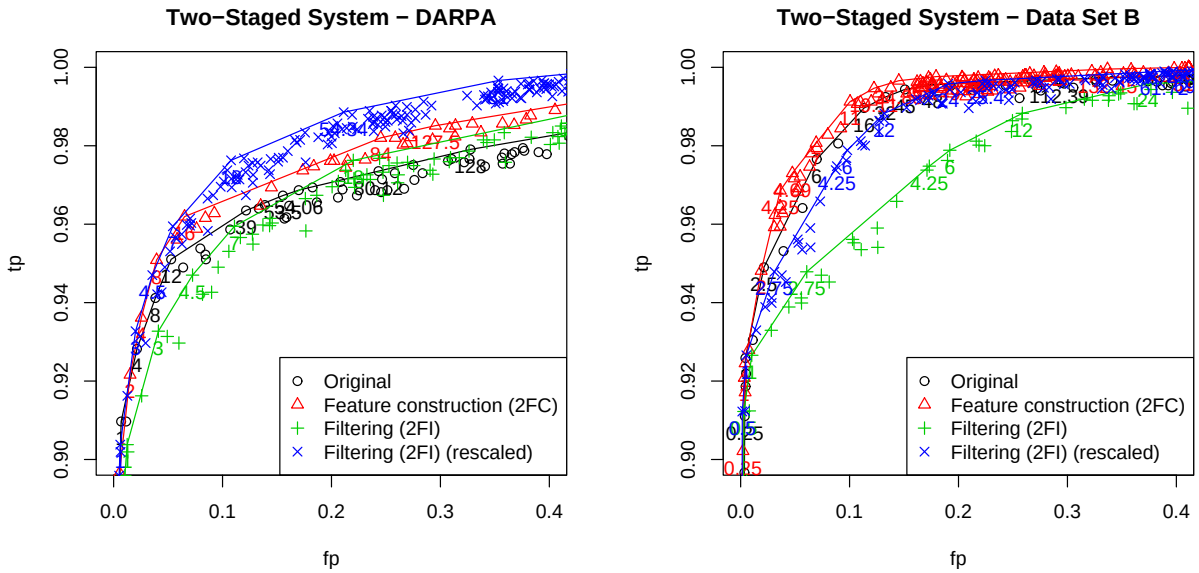
**Hypothesis 8.6.1** *A two-stage alert classification framework with CLARAty and ALAC improves the performance of ALAC in terms of the number of misclassifications and the analyst's workload, making it useful for intrusion detection.*

Recall that the two-stage alert-classification system, in which CLARAty is used at the first stage together with automated cluster labeling and ALAC is used in the second stage, has two modes depending on how the information obtained from CLARAty is used in the

subsequent stage. In the first mode, namely the *feature-construction mode* (hereafter called *2FC*) clusters are used to construct four additional features characterizing properties of the cluster the given alert belongs to. In the second mode, namely the *filtering mode* (hereafter called *2FI*) alerts belonging to clusters marked as false alerts are removed.

### 8.6.1 ROC analysis

Similarly to the evaluation of ALAC we performed an ROC analysis on a 10% stratified sample of alerts to understand the performance of a system under different cost ratios and class distributions, shown in Figure 8.13.




(a) DARPA1999 Data Set (b) Data Set B

Figure 8.13: ROC curves for two types of two-stage alert-classification systems: 2FC and 2FI, for DARPA 1999 Data Set and Data Set B.

We observe that for both datasets, the feature-construction mode performs marginally better than the standard classifier and that the filtering mode performs comparably to the original system for the DARPA 1999 Data Set and worse for Data Set B. In all the cases the differences become smaller with higher cost ratios (i.e., for classifiers with higher both true-positive rates and false-positive rates).

Note, however, that a special care needs to be taken comparing false-positive rates of systems in the 2FI mode and the remaining two modes as the number of negative examples is typically much smaller in the former. To illustrate this with an example, a classifier with a false-positive rate $fp = 0.3$ in the 2FC mode would make exactly the same number of misclassifications as a classifier with a false-positive rate $fp = 0.81$ in the 2FI mode, assuming that 63% of false positives are removed in the first stage. To illustrate this, we rescaled the original 2FI-series calculating the rates at relative to the number of instances classified at the first stage. Here we obtain that the classifier performs much better for DARPA 1999 Data set and marginally worse for Data Set B.

Similarly, to the previous two systems, obtained ROC curves can be used to select optimal

parameters, including the cost ratio for ALAC in the recommender and agent modes. For the remaining experiments we assumed $ICR = 50$ and selected identical parameters $w$ for ALAC in the agent and recommender mode as for the evaluation of ALAC+: $w = 76$ for DARPA 1999 Data Set and $w = 16$ for Data Set B (cf. Table 7.1).

### 8.6.2 DARPA 1999 Data Set

Figure 8.14 shows the performance in terms of false-negative and false-positive rates of the two-stage system in agent and recommender modes.

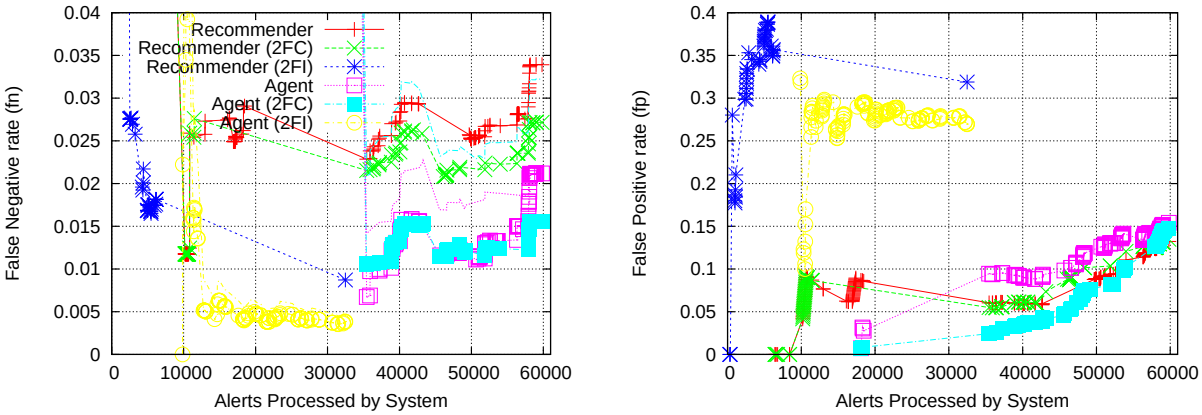
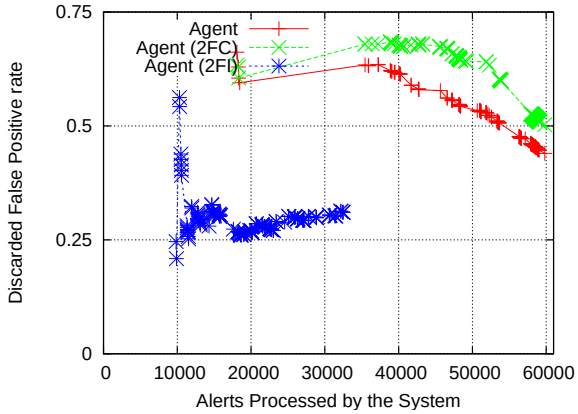



Figure 8.14: Two-stage alert-classification system: False negatives and false positives for ALAC and two-stage ALAC (2FC, 2FI) in agent and recommender modes (DARPA1999 Data Set, $ICR$ =50).

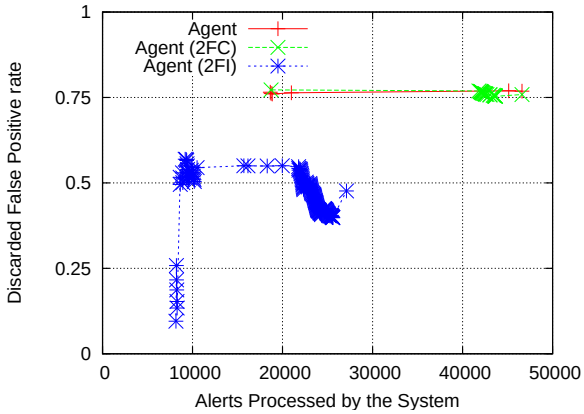
In the recommender mode, the system using the feature-construction mode has a much higher false-positive rate and smaller false-negative rate than the baseline system. In the filtering mode, the system has a comparable false-positive rate to the baseline system and significantly lower false-negative rate ($fn = 0.008$ vs. $fp = 0.034$). However, in the filtering mode, the system had removed 53% of all alerts in the filtering stage (cf. Figure 8.10b), therefore the absolute number of all false positives is only marginally higher than for the baseline system.

Similar findings can be observed for the agent mode, in which the system exhibits a much lower false-negative rate ($fn = 0.0045$ vs. $fn = 0.02$ for the baseline). The reason for this is that while filtering alerts we managed to remove a number of false alerts mistakenly classified as true alerts. Hence, the system could perform much better in terms of false negatives compared to the baseline system.

An interesting conclusion can be drawn observing the fraction of discarded alerts in the agent mode, shown in Figure 8.15a). After filtering the "easy" alerts in the first stage, the agent processes automatically less alerts, thus the rate of discarded false positives is lower. On the other hand, in the feature construction mode, the system has more features to classify alerts reliably and thus can discard more alerts compared to the baseline.




(a) DARPA1999 Data Set, $ICR$ =50



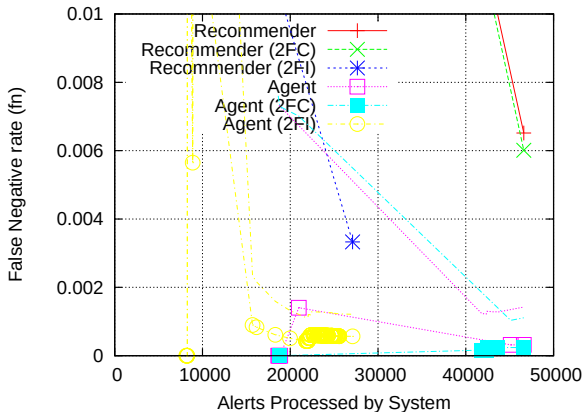

(b) Data Set B, $ICR$ =50

Figure 8.15: Two-stage alert-classification system: Number of alerts processed autonomously by ALAC and two-stage ALAC (2FC, 2FI) in agent mode.

### 8.6.3 Data Set B

For the second dataset, we observe similar results as for the DARPA 1999 Data Set, although due to much smaller absolute values of false-negatives and false-positives rates and their higher variance, the comparison of the absolute values has to be done carefully.

The feature-construction mode performs comparably to the baseline in terms of both false-positive and false-negative rates. Similarly, the filtering mode performs worse in terms of false-positive and comparably in term of false-negative rates to the baseline system. However, taking intro account that in 41% of all alerts have been discarded, the actual number of false positives is comparable. We also observe that the system in the filtering mode processes automatically less alerts (Figure 8.15b) as alerts "easy" to classify have already been discarded in the first stage.



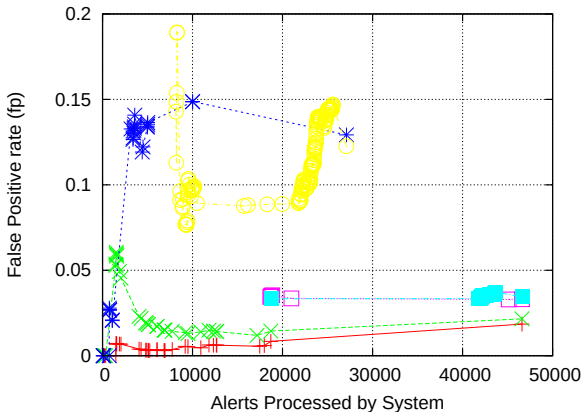




Figure 8.16: Two-stage alert-classification system: False negatives and false positives for ALAC and two-stage ALAC (2FC, 2FI) in agent and recommender modes (Data Set B, $ICR$ =50).

Based on the evaluation of the two-stage alert-classification system on two different datasets in the previous section we can conclude that the system works well in the filtering mode, lowering the number of misclassifications (both false negatives and false positives). In addition, we have shown that CLARAty and the analysis of clusters generated in the first stage can significantly reduce the numbers of alerts to be processed and, if performed systematically, reveal inconsistencies in alert labeling.

### 8.6.4 MSSP Datasets

As the last part of our evaluation we decided to apply two-stage ALAC to MSSP datasets. Recall that those datasets have two properties that make supervised learning difficult: (i) the dataset is highly skewed dataset and contains very few incidents, (ii) there are inconsistencies in the labeling. The first property was investigated in Chapter 4 and the second one was investigated while evaluating the CLARAty performance in Section 8.4.

Note that while the rarity of intrusions is intrinsic to computer security, labeling inconsistencies result from the way the incidents data was originally used. For example, after detecting that a particular machine is infected with a worm, the analyst would identify this as an incident and report it to the customer. However, subsequent alerts originating from this machine may be neither tagged as security incidents nor added to the original incidents. Consequently, semantically identical alerts receive contradicting labels, and thus make it difficult to learn an accurate classifier.

Having said this, we decided to evaluate a representative sample of MSSP alerts. We have selected a subset of MSSP customers and evaluated the system for the time period of one month. We performed the evaluation of a two-stage system in the filtering mode, in which false positive clusters labeled by CLARAty are removed prior to the classification stage. Note that this effectively changes the distribution of instances in favor of true alerts and thus makes it more similar to the datasets used previously. The statistics for those datasets before and after the first stage are shown in Table 8.5

Table 8.5: Statistics for a subset of 10 MSSP customers for a period of one month used in a two-stage ALAC experiment.

| Customer | #Alerts | #Alerts (after CLARAty) | #Positives |
|---|---|---|---|
| 3288 | 11586 | 8674 | 3293 |
| 3359 | 22954 | 15203 | 400 |
| 3362 | 115788 | 66120 | 498 |
| 3363 | 133624 | 61607 | 2080 |
| 3408 | 7694 | 5016 | 535 |
| 3426 | 93517 | 18677 | 187 |
| 3491 | 23176 | 6483 | 946 |
| 3520 | 15783 | 3935 | 825 |
| 3532 | 27230 | 16854 | 758 |
| 3647 | 142295 | 54117 | 5807 |

Prior to evaluating ALAC, similarly to the previous section, we performed ROC analysis on a stratified samples of those datasets to find optimal parameters for ALAC and also to allow us to find parameters for abstaining classifiers. We obtained the following ROC curves shown in Figure 8.17.

Investigating the reasons why the classifier would perform poorly in those three cases,

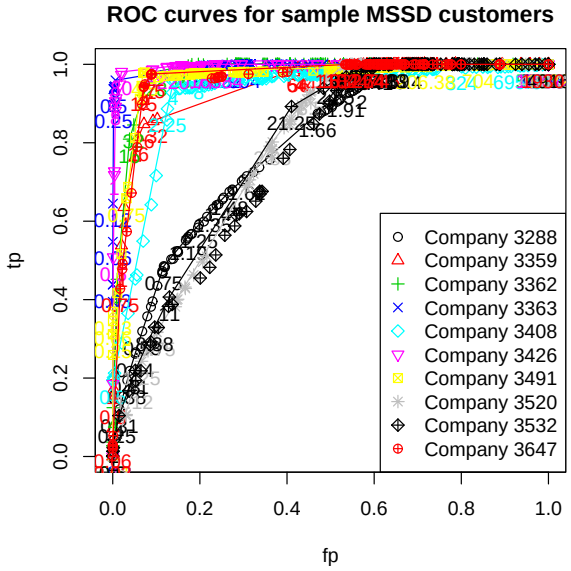



Figure 8.17: ROC curve for sample MSSP datasets.

namely for companies 3288, 3520 and 3532, we decided to estimate the fraction of inconsistently labeled alerts for those companies. In order to do so, we grouped alerts by the three key attributes: source and destination IP addresses and the signature and calculated the number of alerts marked as incidents or non-incidents in each of those groups.

Note that this is only an approximation, as alerts in the same group may receive different classification, depending on other alerts they co-occur with, or depending on the values of the remaining attributes, including the payload.

For companies with few inconsistencies, we would expect those groups to be polarized, either the events in each of those groups would be classified entirely as incidents or non-incidents. Conversely, for companies with inconsistencies, there would be a large number of groups with both values high. We will call these groups *conflicting*. We further calculated the total number of alerts in those conflicting groups and scaled them according to the total number of positive and negative events for this customer. We obtained a distinct group of customers for which the values of those inconsistently labeled groups were high: 3288, 3520, 3408, 3532, which are the four companies with the worst ROC performance. For the remaining companies the fraction of conflicting groups is smaller by at least one order of magnitude. To illustrate this with an example, for customer 3288 conflicting groups would cover 55% of false alerts and 99% of true alerts. In contrast, conflicting groups for customer 3363 cover 0.09% of false alerts and 8% of true alerts. This confirmed our hypothesis that alerts are labeled inconsistently and showed that the approximation used was good.

In our experiments we evaluated ALAC in recommender mode and ALAC+ in the BA0.1 model, in which the classifier abstains for up to 10% of all instances. The results are shown in Table 8.6.

We can see that although in most of the cases (except for the customer 3362) abstaining classifiers reduced the misclassification cost per classified example, for three customers with high inconsistency rates, namely, 3288, 3520 and 3532 the system has a very high false-positive rate in both cases. Another three customers, namely 3408, 3426 and 3647 also have very high

Table 8.6: Two-stage classification system (2FI) with MSSP datasets. The last column shows improvement ALAC+ (BI0.1) over ALAC with $ICR = 50$.

| Customer | Recommender ALAC | | Recommender ALAC+ (BA0.1) | | | Imp. |
|---|---|---|---|---|---|---|
| | *FN* (*fn*) | *FP* (*fp*) | *FN* (*fn*) | *FP* (*fp*) | k | |
| 3288 | 36 (0.01) | 3653 (0.68) | 32 (0.01) | 3229 (0.65) | 0.06 | 0.04 |
| 3359 | 131 (0.32) | 707 (0.04) | 76 (0.30) | 263 (0.02) | 0.36 | 0.11 |
| 3362 | 192 (0.38) | 1353 (0.02) | 184 (0.57) | 158 (0.005) | 0.04 | -0.97 |
| 3663 | 149 (0.07) | 2940 (0.04) | 69 (0.03) | 120 (0.002) | 0.12 | 0.60 |
| 3408 | 22 (0.04) | 1194 (0.26) | 13 (0.02) | 863 (0.26) | 0.25 | 0.13 |
| 3426 | 1 (0.005) | 16547 (0.89) | 16 (0.10) | 164 (0.018) | 0.51 | 0.88 |
| 3491 | 25 (0.026) | 872 (0.15) | 27 (0.03) | 341 (0.07) | 0.09 | 0.12 |
| 3520 | 116 (0.14) | 1405 (0.45) | 0 (0) | 1162 (0.47) | 0.21 | 0.79 |
| 3532 | 3 (0.004) | 12518 (0.77) | 3 (0.004) | 8112 (0.69) | 0.26 | 0.12 |
| 3647 | 218 (0.04) | 14678 (0.30) | 198 (0.05) | 3024 (0.08) | 0.25 | 0.31 |

false-positive rates, which were effectively reduced by ALAC+. However, the discrepancies between estimated performance based on ROC curves and the actual performance suggest that the parameters were not chosen optimally and the dynamic ROC building suggested in Section 7.2.3 should be used.

### 8.6.5 Conclusions

In Section 8.6 we evaluated the two-stage alert-classification system with CLARAty and ALAC. We evaluated two modes of the system: the feature-construction mode (2FC), in which alerts receive additional features constructed based on the clustering stage and the filtering mode (2FI), in which clusters marked as FA-only clusters are removed prior to clustering.

The feature-construction mode has the advantage that there is no risk of alerts being erroneously removed by the system, although the analyst has to review the same number of alerts as previously. Having evaluated the system of two datasets, we showed that in both cases feature construction only marginally improves the classifier's performance. This was the observation based on both ROC analysis as well as simulations with the real system.

Conversely, in the filtering mode, the analyst has only to review alerts that were not removed by the first stage of the system, which on average can lead to up to three times reduction in the alert load. However, this reduction is achieved at the cost that some true alerts are being removed. We have previously analyzed this threat and experimentally showed that such a system is generally safe and robust, however, such a system has a positive-feedback loop: once an attack has been missed, there is an increased chance that similar attacks are missed in the future.

As the first stage of the system removes alerts that are generally "easy" to classify, the resulting system has generally higher false positives rates and comparable or lower false-negative rates than the original system. The explanation for this is that the first stage removes only false positives and effectively changes the class distribution in favor of true positives. However, in our experiments, the number of false positives is comparable to the original system. This, together with the fact that on average the analyst would have to process only one third of the original alerts, makes it useful for intrusion detection and confirms the hypothesis stated at the beginning of this section.

Finally, we applied our system for the selection of MSSP datasets, and found out that

there are serious inconsistencies in alert labeling making supervised learning difficult. We nonetheless evaluated the system in the recommender mode and with abstaining classifiers (BA0.1) and obtained satisfactory results. However, discrepancies in the performance between the actual performance and ROC estimates suggest that the dynamic ROC building should be used. Note also that in the recommender mode used, all errors would be ultimately corrected by the human analyst.

## 8.7 Summary

In this chapter we discussed how unsupervised learning can be used with supervised learning in order to improve alert classification. We presented CLARAty, a clustering algorithm proposed by Julisch [Jul03b] to discover patterns in the alert stream. We discussed how CLARAty can be used in the retrospective analysis mode, in which labeled clusters are used to analyze historic alerts and the filtering mode, in which labeled clusters are used to facilitate subsequent classification.

Assuming the setting in which alerts are reviewed by the analyst and thus the labels are given, we proposed an automated cluster-processing system with CLARAty and a two-stage alert-classification system with CLARAty and ALAC. We evaluated both systems on a variety of datasets and showed that the automated cluster-processing system is safe and robust and removes 63% alerts on average. In experiments with a two-stage classification system, we obtained lower false-negative rates and comparable false-positive rates to the original system with the analyst's workload reduced by 63% on average (due to the the first stage). This confirmed that the two-stage alert-classification system is useful for intrusion detection.

# Chapter 9

# Summary, Conclusions and Future Work

## 9.1 Summary

Throughout their history, IDSs have been designed with different algorithms and detection techniques. With the technology becoming mature enough for wide-scale deployment, it has become clear that a high number of false positives, i.e., alerts not related to security problems, is a critical factor determining the cost-effectiveness and usability of IDSs.

In the effort to reduce the number of false positives, practitioners and researchers have been focusing on the following three levels with their broadening scope: (i) improving IDSs themselves, (ii) leveraging the environment, and (iii) alert postprocessing.

At the *first level*, better sensors were built, improving the detection capabilities of IDSs, lowering their false-positive rates or both. On the one hand, we observe a trend to build general sensors, with the aim of detecting the largest variety of attacks and incidents, but at the cost of some false positives. On the other hand, there is a number of highly specialized sensors aiming at detecting only particular types of attacks (e.g., [SGVS99, RZD05, VMV05]) with extremely low false-positive rates, thus allowing to fulfill the long-standing dream of intrusion detection—automated response to incidents. However, such specialized sensors offer only limited protection and have to be augmented with other sensors. At the *second level*, sensors were augmented with environment information such as vulnerability scans or OS detection (e.g., [SP01, LWS02, VMV05]), enabling a decrease of false-negative rates thanks to additional context information available to the IDSs. Finally, at the *third level*, namely alert postprocessing, alerts generated by IDS sensors are used as input for processing tools that try to improve their quality. For example, filters created using data mining can be used to remove frequently reoccurring false positives [Jul03b]. Alert correlation [CAMB02, DW01, VS01, VVCK04]) also uses the alert-postprocessing approach, although in a broader scope, with the aim of reconstructing high-level incidents from low-level alerts as well as, indirectly, reducing the number of false positives. Clearly, alert-postprocessing tools can reduce the number of false positives, but cannot detect attacks that have been missed by the underlying IDS sensor.

The approaches discussed above focus on the environment the IDSs work in; however, with a few notable exceptions (e.g., [Fan01, VS00]), they did not take advantage of the way IDSs are deployed and used in practice. In fact, in real environments most of the data collected by

IDSs are timely analyzed by the security analyst, who analyzes alerts and takes appropriate actions. The observation that in most cases the analyst analyzes alerts in real time and thus provides an implicit (or in some environments explicit) classification of alerts and also that at a given time $t$ the analyst has analyzed all previously received alerts, forms a new paradigm for the classification of intrusion detection alerts using supervised learning. This paradigm operates at the highest, the *fourth level* of processing IDS alerts by involving the human operator.

In this dissertation we presented a comprehensive approach to alert classification at this fourth level of our classification, and confirmed the hypotheses stated in Chapter 1.

**Thesis Statement**

> *(1) Using machine learning, it is possible to train classifiers of IDS alerts in the form of human-readable classification rules by observing the human analyst.*
>
> *(2) Abstaining classifiers can significantly reduce the number of misclassified alerts with acceptable abstention rate and are useful in intrusion detection.*
>
> *(3) Combining supervised and unsupervised learning in a two-stage alert-processing system forms a robust framework for alert processing.*

**(1) ALAC:** In this dissertation we have presented a novel concept of building an adaptive alert classifier based on an intrusion-detection analyst's feedback using machine-learning techniques. We discussed the issues of human feedback and background knowledge, and reviewed machine-learning techniques suitable for alert classification. We proposed two modes of operation of the system, namely, the agent mode and the recommender mode, in which the system can process some of the alerts autonomously.

To demonstrate the feasibility of this architecture, we presented a prototype implementation and evaluated its performance on synthetic as well as real intrusion data, this way showing that the first part of the thesis statement is true. We have shown that such learning is not only possible, but even yields low false-positive and false-negative rates. For example, for the DARPA Data Set, the system applied on alerts yielded low false-positive and false-negative rates ($fp = 0.025$, $fn = 0.038$). Similarly for another dataset, the system yielded an even lower false-negative rate ($fn = 0.003$), and higher, but still acceptable, false-positive rate ($fp = 0.12$). Although these figures are data dependent they show the expected performance of the system in our application.

**(2) Abstaining Classifiers:** Stemming from the observation that the analyst and ALAC form in fact a multi-stage alert-classification system, we started investigating abstaining classifiers, i.e., classifiers that can abstain from classification in some cases. While abstaining classifiers and normal classifiers cannot be compared directly (as the trade-off between non-classified instances and the misclassification cost is not defined), we introduced three different evaluation models for a particular type of abstaining classifiers, namely, the cost-based model, the bounded-abstention model and the bounded-improvement model. Given the cost ratio, the class distribution, the ROC curve and the boundary conditions, we showed how to select the classifier optimally in each of these models.

These methods can be applied to arbitrary machine-learning techniques, but we showed that by applying them to ALAC one can significantly reduce the misclassification cost, as

well as the overall number of misclassifications. We applied the two most suitable models, namely the bounded-abstention and the bounded-improvement model, to both the DARPA 1999 Data Set and Data Set B, and found that the abstaining classifier significantly lowered the false-positive and false-negative rates and thus the resulting misclassification cost even with low abstention rates. For example, for Data Set B, assuming that 10% of alerts are left unclassified, we lowered the false-negative rate by 76% and the false-positive rate by 97% thus reducing the overall misclassification cost by 87%, with a realistic cost ratio. Moreover, in all evaluation runs, we observed that the system with an abstaining classifier had a significantly lower false-positive rate, resulting in a lower number of all misclassifications.

The second fact is extremely important for the efficacy of classification involving human analysts: As argued by Axelsson [Axe99], if an alert is classified as positive by the system but in reality has only a very small chance of being positive, the human analyst may well learn from such incidents to ignore all alerts classified as positive henceforth. This greatly increases the probability that real attacks will be missed. Abstaining classifiers, by introducing an abstention class, ensure that the classifier has a high true-positive rate, without generating an excessive number of false positives. This makes such a system useful for the human analysts and confirms the second part of the thesis statement.

**(3) Two-stage Alert-Classification System:** Based on the work by Julisch [Jul03b] on clustering alerts to discover root causes, we proposed a two-stage alert-classification system, in which the alert-clustering system CLARAty is combined with ALAC to further improve classification. We investigated how unsupervised learning can take advantage of existing alert labels in order to form a robust alert-classification system. We further proposed and evaluated an automated cluster-processing system that labels clusters created by CLARAty and removes those containing only false positives.

We systematically showed how labeled clusters can be used in retrospective cluster analysis and introduced clustering-precision and clustering-recall charts, which are useful in assessing the quality of clusters, the quality of human classification, or both. Assuming that clusters are not reviewed, we showed that an automated cluster-processing system can automatically and safely reduce the number of alerts to be processed by 63% on average. This, combined with ALAC as an alert-classification system, yields an alert-classification system that has a lower false-negative rate, however, at the cost of higher false-positive rate. The last finding can be explained by the fact that alerts removed in the first stage are "easy" to classify and hence they would be also correctly classified by ALAC itself. Note, however, that the two-stage alert-classification system lowers the number of alerts to be processed by 63% on average, meaning that the analyst, assuming the same workload, can spend almost three times as much time on every alert compared with the baseline system. In addition the derived and labeled clusters can support retroactive alert analysis, providing a robust alert-processing framework. This result supports the third part of our three-part thesis statement.

## 9.2 Conclusions

Having evaluated the approach on a variety of datasets we showed that supervised alert classification can help the analyst classify the alerts and that by applying abstaining classifiers the overall misclassification cost in general, and the number of misclassified alerts in particular, can be significantly reduced. We also showed that a two-stage alert-classification system works

and that the combination of CLARAty and ALAC forms a robust alert-processing framework.

We discussed the evaluation problem in intrusion detection, in particular the lack of representative and universally reproducible datasets. We based our evaluation on one synthetic (DARPA 1999), one real (Data Set B) dataset and, partly, additional real datasets from a Managed Security Services Provider (MSSP). The latter datasets come from a real environment and have two features that make alert-classification systems such as ALAC difficult to apply: (i) rarity of intrusions and (ii) labeling inconsistencies.

Whereas the rarity of intrusions is intrinsic to computer security, labeling inconsistencies result from the way the incidents data was originally used, in which the analysts focus on detecting incidents, not on accurately classifying all alerts. Consequently, identical alerts receive contradicting labels, and thus make it difficult to learn an accurate classifier. Moreover, if the classifier correctly recognizes worm infections, those mislabeled alerts will be counted as false positives. If an alert-classification system such as ALAC is to be used in practice, analysts should understand how the system works and make an effort to label alerts consistently. To facilitate this the console should be modified to support efficient alert labeling and allow analysts to easily add alerts to already existing incidents.

We should be aware that there is some risk involved in using alert-classification systems. If the system has a good accuracy, the analyst may learn that the system is always right and not review any alerts classified as false alerts. This may result in new attacks being missed. Conversely, if the system has a high false-positive rate, the analyst may learn to ignore any alerts reported by the system or simply stop using it. In any case, the issue of human-computer interaction will need to be further investigated.

Clearly, the alert-classification system is more likely to miss new things than intrusions that have been seen before. Although it is also more likely that new types of intrusions are missed by the human analyst alone, this effect can be amplified if an alert-classification system is used. Similarly, an alert-classification system is much better in classifying automated attacks than attacks that are launched by skilled hackers. Those attacks are also more likely to be missed by an underlying IDS, although the attacker may try to trick the system into classifying real attacks as false alerts. This shows the danger associated with the use of alert-classification systems such as ALAC, which can be minimized by using the system in the recommender mode only and setting misclassification costs and abstention windows appropriately.

Thus there has always been a trade-off between true positives and false positives. First, IDSs themselves detect only some intrusions. Clearly, investigating every single network packet would achieve a higher true-positive rate, but obviously would not be feasible. Second, having human analysts analyze $n$ alerts during a shift is a trade-off. If the analyst were investigating only $n/2$ alerts, or have every alert double-checked by another analyst, this would likely increase the detection rate, but at the cost of additional resources. Finally, using an alert-classification system is also a trade-off. While we have shown that most of the alerts can be classified correctly, thus significantly reducing the analyst's workload, there are some that would inevitably be missed. It is important to be aware of these trade-offs and choose the solution that is optimal. To this end, we proposed a cost-sensitive setup, two modes of operation of the alert-classification systems and, finally the use of abstaining classifiers.

For the past 15 years, IDSs have been known to trigger many false positives and currently there is still no silver-bullet solution to this problem. The problem does and will continue to exist, but automatic alert-classification systems, such as the one proposed in this dissertation, are a step towards solving it.

## 9.3 Future Work

The work presented in this dissertation has explored the basic concepts of adaptive alert classification by showing its feasibility. It also showed areas for interesting future research.

With the assumption that an alert-classification system can only be used by the analyst if its logic can be interpreted (and corrected if necessary), we focused on rule-learning algorithms (cf. Section 5.3). While this is a laudable goal, machine learning has been recently focusing on learning techniques such as SVMs, Bayesian networks or instance-based learning, which are not interpretable. However, these techniques have other desirable properties such as supporting incremental learning or generating ranks (or even calibrated probabilities) instead of classification. In particular, supporting incremental learning and efficiently generating ROC curves would increase the efficiency of the algorithms and would allow dynamic optimization of thresholds, including the dynamic recomputation of abstaining classifiers as the learning process progresses (cf. Section 7.2.3).

In the domain of intrusion detection, alerts are not independent of each other and identically distributed, which violates the assumption the basic machine-learning algorithms used. In the current approach we tried to capture this relationship using propositionalized background knowledge; however, a more systematic approach to this problem would be to use link mining [Get03].

In our evaluation we assumed a simple binary model in which alerts were classified into true and false positives. While this is a reasonable assumption (after all the most important information is whether an alert is related to a security problem or not), the classification system would be much more useful if it could identify the type of threat or what type of false alert an event is. Extending alert classification into multi-class classification has serious implications in both practical and theoretical aspects of the system. On the practical side, with $n$-class cost-sensitive classification, up to $n(n-1)$ cost parameters would have to be estimated. Second, the human analysts would need to precisely define a set of allowable classes and how to classify them. However, our analysis of real event data shows that even with binary classification the labeling is often inconsistent. On the theoretical side, our model for abstaining classifiers is based on ROC analysis and, as multi-class ROC extensions are limited, it is inherently limited to two classes. However, extending abstaining classifiers into multiple classes would be a useful and interesting project.

Abstaining classifiers are intuitive and based on ROC analysis, commonly used in machine learning. Their main advantage is that they output a classifier that minimizes the misclassification cost and do not make assumptions that the underlying classifier outputs exact probabilities. However, there are classifiers that output accurate probability estimates, and others can be calibrated to do so (e.g., [ZE01, CG04]). In this case, the optimal abstaining classifier could be chosen using the Bayesian decision theorem in all three models we proposed. The comparison of the original abstaining classifiers and the Bayesian abstaining classifiers and also cautious classifiers [FHO04] is an interesting future work item. In this case, abstaining classifiers could easily be extended to multi-class classification.

In our system, while combining supervised and unsupervised learning, the clustering algorithm does not use alert labels, so that we can correct some misclassified labels. In reality, however, the results could be better if the clustering algorithm used the labels to induce clusters. Future work could modify the clustering algorithm to make it more similar to predictive clustering rules (e.g., [ŽDS05]) in order to leverage alert labels.

In Section 1.1.2 we showed that alert classification can be used with other methods that

reduce false positives in intrusion detection, such as leveraging the environment and other alert-postprocessing methods. Applying our methods, including an automated-cluster processing with CLARAty and ALAC with abstaining classifiers, to a multi-stage alert-correlation system (e.g., [VVCK04]) would yield an even more comprehensive alert-processing system.

Finally, in our prototype we assumed a simple user interaction, in which alerts are classified as true or false positives. Looking at the MSSP data, we observed that in reality some events are assigned into incidents, and the remaining incidents are left unassigned. While incidents can easily be converted to binary labels, the fundamental problem is that the analysts are encouraged to detect and report incidents but not to accurately label all alerts those incidents may be comprised of. This causes discrepancies in alert labels and makes the learning problem extremely difficult.

The current system uses RIPPER, a noise-tolerant algorithm, but the extent to which the system tolerates labeling errors is currently unknown. This could be evaluated by introducing artificial errors into the dataset and evaluating the results.

Future human-computer interaction research and usability studies could investigate the way supervised alert-classification systems could be integrated with the console, making the interaction as easy as possible and, at the same time, to achieve the most accurate alert labeling.

Looking at the evolution of intrusion detection systems we observed a trend from host-based to the network-based systems, which we dealt with in this dissertation, followed by the recession of the latter in favor of host-based and hybrid systems. This was caused by the problems with network-based IDSs in faster, switched network environments and the use of complex application-level protocols using end-to-end encryption. Another reason is that the majority of attacks nowadays occurs at the application level, and they are best detected by the host-based IDSs. We expect this trend to continue and see the need for researchers to develop new and improve current host-based IDSs (e.g., CSSE, our host-based IDS is a step in this direction).

On the other hand, network-based IDSs are indispensable in detecting certain policy violations and low-level attacks. Here the research challenges are to improve current IDSs at all four levels (cf. Section 1.1.2), in particular focusing on reducing the number of false positives and producing high-level semantic alerts.

We think that machine-learning and data-mining techniques for assisting alert classification and finding patterns in the alert logs will gain more significance. Systems such as ALAC, proposed in this dissertation, can significantly reduce the human analyst's workload and therefore can be very useful. The areas needing research include both the machine-learning side and the human-computer interaction side.

Intrusion detection systems have come a long way in the past 15 years, and so have the intruders. With most of the information in the modern world being processed by the systems in some way connected to computer networks, the research into intrusion detection and the application of machine learning plays an important role in the security of current and future electronic computing.

This dissertation has explored the feasibility of using supervised learning in the classification of intrusion-detection alerts, and opens multiple possibilities for future exploration and research, which may lead to the design and the development of more efficient, reliable and effective alert-management systems.

# Appendix A

# Alert Correlation

## A.1 Correlation Terminology

The definition of *intrusion* introduced in Section 1.1 given by Heady [HLMS90] is widely accepted in the scientific community. Correlation in intrusion detection, however, has no consistent definition and has been used differently by researchers and IDS vendors.

Before we try to define correlation, let's look at the dictionary [Pea00] definition shown below.

> $cor \cdot re \cdot la \cdot tion$
>
> 1. A causal, complementary, parallel, or reciprocal relationship, especially a structural, functional, or qualitative correspondence between two comparable entities: a correlation between drug abuse and crime.
> 2. Statistics. The simultaneous change in value of two numerically valued random variables: the positive correlation between cigarette smoking and the incidence of lung cancer; the negative correlation between age and normal vision.
> 3. An act of correlating or the condition of being correlated.

Statistical correlation as shown in definition (2) is not appropriate for intrusion detection, since the researchers do not model alert streams by means of random variables. Much closer is the definition (1), which is talking about *causal relationship* between comparable entities, however, this definition is not very precise.

Intuitively, we may say that correlation in intrusion detection concerns finding relationships between alerts generated by a single (or multiple) data sources and coupling this information with additional knowledge. The goal of correlation would be to present the information generated by IDS in a meaningful way, i.e., a way which can be easily understood and processed by a human analyst and helping him discover attacks and incidents.

Recall in Section 3.5 we defined terms: attack, incident, aggregation and correlation. To avoid confusion we will show how other researchers used (or defined) these terms:

**Cuppens et al.** (cf. Appendix A.2.5) use a definition of correlation similar to ours. They differentiate two types of correlation as described below:

- Explicit correlation, where it is possible to express some connection between known events. This form of knowledge has to be manually entered into the system.

- Implicit correlation is used when data analysis brings out some mappings and relations between events. Implicit correlation can be based on learning techniques and statistics.

**Dain and Cunningham** (cf. Appendix A.2.3) describe their work as *fusing* rather than correlating. The fusion process is defined as joining alerts from multiple intrusion detection systems into scenarios, revealing attacker activities.

**Debar and Wespi** (cf. Appendix A.2.1) propose an architecture with so-called aggregation and correlation components, however their definition of these terms is quite different from ours. In their understanding, *correlation* uses explicit correlation rules, manually programmed or derived automatically from configuration parameters. There can be two types of correlation relationships between events:

**duplicates** alerts considered to be related to the same event according to manually programmed criteria,

**consequences** sets of alerts appearing in a given order within a given time interval.

*Aggregation*, following correlation, is the process of grouping events together according to certain criteria to compute aggregated severity level. The goal of aggregation is to discover high-level incidents.

**Morin et al.** (cf. Appendix A.2.7) propose a formal data model for alert correlation, but only implicitly define what correlation is. The M2D2 model uses definitions of aggregation and correlation similar to the work by Cuppens et al.

**Ning et al.** (cf. Appendix A.2.4) define correlation in a broader sense, which covers both our definitions of aggregation and correlation. One of the reasons for that is that there is no separate aggregation process in their work.

**Qin and Lee** (cf. Appendix A.2.8) use the term alert correlation similarly to our definition, i.e., reconstructing incidents from alerts. Other terms used include *alert fusion*—combining multiple alerts from different IDSs into a single *hyper alert*, and *alert clustering*—the process of combining hyper alerts based on the equality of attributes (in this particular case, the equality of all alert attributes except for the time).

**Valdes and Skinner** (cf. Appendix A.2.2) define correlation in a broader sense, similarly to Ning et al. Correlation is further defined at three stages:

**event aggregation** aggregates a large number of low level events (e.g., TCP connections, audit records).

**sensor coupling** correlation utility comprehends data from different sensors, to achieve better sensitivity and false alarm suppression. Sensors can also communicate with each other.

**meta alert fusion** this allows for reconstructing scenarios of an entire attack.

Further work by Porras et al. [PFV02] extends the definition of correlation to cover a broader definition of alert stream contributors including context information (e.g., network topology, operating system, running services, vulnerability scans) and security goals.

**Valeur et al.** (cf. Appendix A.2.9) in their 10-step alert correlation system, use different terminology for each of the steps performing alert aggregation and correlation. Similarly to Qin and Lee, *alert fusion* refers to combining alerts from different sources into a single *meta-alert*.

Subsequent correlation steps, namely: *thread reconstruction*, *session reconstruction*, *focus reconstruction* and *multi-step correlation* perform subsequent grouping of meta-alerts into higher level meta-alerts using predefined heuristics (or given attacks in the last step).

Finally, *alert verification* refers to correlating alerts with vulnerability information and *impact analysis* refers to correlating incidents with the information about the criticality of targets they affect.

## A.2 Alert Correlation Systems

In this section we will discuss existing alert correlation systems, previously discussed in Section 3.5.

### A.2.1 Tivoli Aggregation and Correlation Component

Debar and Wespi [DW01] describe an algorithm and an architecture of an Aggregation and Correlation (AC) Component. The goal of the algorithm is to form groups of alerts by creating a small number of relationships. The authors define two kinds of relationship: a *correlation relationship* and an *aggregation relationship* (using a different terminology than ours in Appendix A.1).

The AC algorithm consists of three sequential steps: In the first step, alerts are verified (that they do not contain invalid information e.g., invalid timestamps) and unified (e.g., different representations of hostnames and IP addresses or port numbers and services).

The second step deals with alerts that are logically linked with each other. It uses system expert system rules to group duplicates (i.e., semantically equivalent alerts) and consequences (alerts appearing in a given order). In both cases the alerts can originate from the same or different IDSs. Note that there is a need to manually specify rules describing these grouping. This step can also be used to detect IDS failure if it is known how two IDSs respond to the same attack. If we receive an alert sequence that is different from what we expect, we can assume that the IDS probably has been broken. In some situations however, this approach may not be reliable, as IDS's behavior may differ with attack variations or may change as IDSs' signatures are being updated.

Finally, the third step groups alarms in three so called *aggregation axes*, according to the match of the following attributes: source IP, destination IP address and alert type. Given these attributes we can identify seven scenarios from the most specific with an exact match of all attributes to the most general (only one attribute matches). For example, the combination $\{SrcIP\ —AlertType\}$ will group alerts triggered by one attacker launching the same attack against different hosts in the network.

Each new alert is simultaneously added to all these seven groups, each of which has a separate sliding time window in which the alerts are counted. An alarm is raised if and only if an alert count exceeds a user-defined threshold value. In reality, sliding time windows

are being approximated by means of a weighted sum scheme, which decreases the use of resources. For each new alert, a new count value is calculated according to the following formula $count_{new} = 1 + count_{old} * 2^{\tau*(t_{old}-t_{new})}$, where $t_{new}$, $t_{old}$ are the timestamps of the new and the last alerts. The parameter $\tau$, also referred to as half-life, is a user-defined fading factor.

The authors also introduce an idea of *triggers* that can modify the severity of alert groups depending on certain criteria. The triggers can be of two types: *situation reevaluation* triggers, based on properties of a single group and *multi-situation assessment* triggers, based on properties of multiple groups, in which one correlation scenario influences an another one.

### A.2.2 Probabilistic Alert Correlation

The Probabilistic Alert Correlation (PAC) system [VS01] represents alert groups by means of so called *meta-alerts*, supporting set-valued attributes. All attributes of a meta-alert representing a group of alerts are a union of appropriate attributes. Clearly, the mapping of alarm groups to meta alarms is not injective.

The system maintains continuously updated groups of meta-alerts. For each new alert, the PAC compares the alert to all existing meta-alerts and calculates similarities between them. The new alert is merged with the most similar meta-alert if the similarity value exceeds a user-defined minimum similarity threshold. Otherwise, the alarm forms a new meta-alert.

The similarity is calculated as a weighted sum of similarities between attributes, if each of these similarities exceeds a user defined minimum similarity. To give a more precise definition, let's assume an alert has attributes $T_1, \ldots, T_n$, while $t_1, \ldots, t_n$ are minimum similarity thresholds and $w_1, \ldots, w_n$ are weights (a.k.a. expectation of similarities). Finally, attribute-wise similarity functions $sim_1(\cdot,\cdot), \ldots, sim_n(\cdot,\cdot)$ return a value between 0 (complete dissimilarity) and 1 (perfect match). The similarity between meta-alert $\mathbf{M}$ and new alert $\mathbf{X}$ is defined as follows:

$$sim(\mathbf{M},\mathbf{X}) = \begin{cases} 0 & \text{if } \exists(i):\ s_i(M[T_i], X[T_i]) \leq t_i \\ \frac{\sum_{i=1}^{n} w_i \cdot s_i(M[T_i], X[T_i])}{\sum_{i=1}^{n} w_i} & \text{otherwise.} \end{cases} \tag{A.1}$$

The authors provide little guidance how to create parameters $t_i$ and $w_i$. These parameters are also situation specific, e.g., the expectation of similarity of source IP addresses should be close to 0 for attacks for which address spoofing is likely, and much higher for other attacks.

The PAC uses intuitively appealing but completely ad hoc similarity functions $sim_i(\cdot,\cdot)$. For example, for alert classes it is defined as a matrix with values of 1 along the diagonal and off-diagonal values expressing heuristic similarities between corresponding alert types. For set-valued attributes, the similarity expresses a fraction of overlap between these sets. For IP addresses it groups addresses coming from the same subnet, and for time value attributes it has been defined as a step function that drops from 1 to 0.5 after one hour.

Since meta-alerts themselves are a valid input for PAC, the system can be cascaded, by running it recursively on its own output. Specifically, the authors use the same system to correlate individual alarms into threads, threads into security incidents and security incidents into correlated attack reports. Note that at each level, the system uses different parameter values and the authors provide very little guidance on how to set them.

### A.2.3 Alert-Stream Fusion

Similarly to the PAC, the Alert-Stream Fusion (ASF) [DC01] maintains a continuously updated list of alert groups called *scenarios*. For each new alert and each existing scenario, the system calculates the probability that the alert belongs to this scenario. The alert is added to the scenario which produces the highest probability score, if it exceeds a user-defined threshold. If all the scores are below the threshold, the alert forms a new scenario. The assignment of an alert to the scenario is final and irreversible. However, unlikely the PAC, in which the similarity is calculated based on set-valued attributes, in the ASF the probability score is a function of a new alert and *only* the last alert in the existing scenario. The other alerts are not considered.

Dain and Cunningham use predictive data-mining techniques and heuristics to learn from labeled training data and evaluate their results. The training data is obtained by manually correlating historical alarms from "attack traffic" (namely DEFCON CTF data) and assigning them into scenarios. Subsequently, the original ASF system is simulated and based on the prior scenario assignment, training examples are generated. Each training example is a tuple $training(A, X, class)$ where $A$ is the last alert from each scenario currently in the memory, $X$ is an alert being added, and $class$ is a class label which can be $merge$ or $\neg merge$. Note that if there are $n$ scenarios in the memory, adding a new alert will generate $n-1$ negative ($class = \neg merge$) and 1 positive ($class = merge$) training example if alert joins scenario, or $n$ negative training examples if the alert forms a new scenario.

The authors use several data-mining techniques in the system. In the first technique, namely a multi-layer perceptron and radial-based functions, training examples are used to adjust numerical coefficients to minimize the sum of squared errors. The second technique was a decision tree, which was learned using a CART algorithm. At the verification stage, the authors compared *confusion matrices* for each of these three algorithms and a naive algorithm joining alerts from the having the same source IP address. Out of these four, the algorithm with decision trees proved to be the best.

To summarize, in their paper the authors show a methodology how to learn correlation algorithms using labeled data. One can imagine that the similar approach could be used also for the PAC and other heuristics based on numerical parameters. The problem however is that labeling alerts and grouping them into scenarios is very labor intensive and unlikely to be done in real environments.

### A.2.4 Hyper-alert Correlation

The Hyper-alert Correlation (HAC) method [NRC01, NC02] correlates alerts using prerequisites and consequences of intrusions. It is based on the observation that most intrusions consist of many stages, with *the early stages preparing for the later ones*. For example, with a Distributed Denial of Service Attack (DDoS), the attacker has to install DDoS daemons on vulnerable hosts before launching an attack. In other words, attacker has to reach certain state before he can launch given attack, and this state is typically reached by launching some other attacks.

Intuitively, a *prerequisite* of an intrusion is a necessary condition for the intrusion to be successful, while a *consequence* of an intrusion is the outcome of an intrusion if it is successful. Note that in most cases one cannot say if an attack reported by an IDS was successful or not, therefore the consequence denotes a *possible* consequence rather than the actual one.

Nevertheless, possible consequences can be used to correlate attacks.

Assuming that for each alert reported by an IDS we have a set of prerequisites and consequences, we can correlate alerts preparing for each other, i.e., alerts whose consequences contribute to prerequisites of the next one. The HAC constructs a correlation graph $(V, E)$, in which each vertex $v_i \in V$ is an alert generated by an IDS and each directed edge $(v_i, v_j) \in E$ means that an attack $v_i$ *prepares* for the attack $v_j$.

To give a more precise definition, let's define a *hyper-alert type* $T$ as a triple $(fact, prerequisite, consequence)$, in which *fact* is a set of attribute names, each with an associated domain of values, *prerequisite* is a logical formula whose free variables are all in *fact*, and *consequences* are a set of logical formulas, such that all free variables are in *fact*. Given a hyper-alert type T, a *hyper-alert instance of type* $T$ is a set of tuples on *fact* associated with time interval $[beginTime, endTime]$ with no free variables in *prerequisite* and *consequence*. A hyper alert implies that *prerequisite* must evaluate to true and *consequence* might evaluate to true for each tuple. A hyper alert $h_1$ *prepares* for a hyper alert $h_2$ if $\exists(p \in prerequisite(h_2) \wedge C \subset consequence(h_1)) : C \ implies \ p \ \wedge \ \forall(c \in C) : c.endTime < p.beginTime$.

The goal of correlation is to help the operator analyze and understand alerts. The authors suggest that the operator interacts with the correlation graph generated by the algorithm. Considering the complexity and size of the graph, the HAC introduces three techniques, namely: adjustable graph reduction, focused analysis and graph decomposition to make the analyzed graph smaller and more comprehensible. The HAC method has been tested on the DARPA 2000 and DEFCON 8 CTF datasets, showing that HAC graphs are a good approximation of real attack strategies. It has been shown that discarding isolated (i.e., non-correlated) alerts significantly reduces the number of false positives with little degradation on the number of attacks detected.

The HAC method is intuitively appealing and captures significant relations between alerts. However it requires that prerequisites and consequences are associated with each alert generated by IDS. Since this information is not provided by IDSs vendors and often signatures are not sufficiently documented, such labeling might be a very difficult task.

### A.2.5 Cooperative Intrusion Detection Framework

The Cooperative Intrusion Detection (CID) framework [Cup01, CM02, CAMB02] integrates heterogeneous IDSs to generate more global and synthetic alerts. The CID framework cascades five processing steps through which each alert is pipelined in a strict order:

**alert base management** This steps unifies incoming alerts and stores them in a relational database. Note that the system assumes that all IDSs generate alerts compliant with the Intrusion Detection Message Exchange Format (IDMEF). IDMEF alerts are subsequently converted into a set of tuples and written to the database.

**alert clustering** Alerts are clustered into groups, which represent the same attack. The grouping is done using expert rules, which indicate which alerts should be considered *similar*. CID defines predicates and rules to define similarities at different levels: alert, entity, instance and attributes, although the authors provide little guidance on how to achieve this.

**alert merging** Similarly to the PAC (Section A.2.2) and the ASF (Section A.2.3), alert groups are incrementally constructed as alerts arrive. Each group has a so-called *global*

*alert*, which contains combined information from all alerts in the group. Global alerts are generated similarly to PAC with set-valued attributes. A new alert is compared to all global alerts in the system and added to similar groups. Note that unlike, the previously discussed approaches, an alert can be simultaneously added to more than one group. Subsequently, global alerts are being updated and if they prove to be similar, the groups can be merged. To ensure that clusters contain only alerts triggered by the same attack, there is some time period after which clusters are considered to be *stable* and cannot be further enlarged.

**alert correlation** An intruder, in order to achieve her malicious objectives, needs to perform several attacks. *Correlation* groups individual attacks and forms candidate plans which are passed to the intent recognition module. CID uses explicit and semi-explicit correlation, which will be described below.

**intent recognition** This module has not yet been described/developed. It should extrapolate candidate plans to anticipate intruder intentions. The result of the function can be used by the operator to launch appropriate counter measures.

The CID system can use both explicit and semi-explicit correlation. The explicit correlation requires that users specify, in which circumstances the two attacks are related (e.g., attacks $A$ and $B$ are related if a target node of $A$ is equal to a target node of an attack $B$). Specifying such correlation rules manually is a complex process and it is not obvious which attacks should be correlated and what conditions for correlation are.

The solution to this problem is a semi-explicit correlation, using attack specification in LAMBDA language [CO00], which defines attacks, with their preconditions and postconditions (a.k.a. consequences).

The correlation model is similar to HAC (see Section A.2.4). Attacks $A$ and $B$ are related if the postconditions of $A$ *contribute* to preconditions of $B$. CID introduces two types of correlation: *direct correlation*, in which any subsets of pre- and post-conditions are unifiable through Prolog most general unifier and *indirect correlation*, in which pre- and postconditions are unifiable through an *ontological rule*. Ontological rules are user defined rules providing correlation between predicates. For example, an ontological rule may specify that if there is a `NetBios` port open on the host it is likely to be a machine running Windows. Another technique used is *abductive correlation*, which makes the CID system work properly even if some actions are not observed. In this case, system generates "virtual alerts" which allow to correlate scenarios.

Semi-explicit correlation can be applied in real-time, however, for performance sake, system generates explicit correlation rules off-line from the attack descriptions. The output of the CID system, similarly to the HAC, are correlation graphs. The authors plan to develop an intention recognition module, which will subsequently process such graphs and recognize attacker intentions.

### A.2.6 Correlated Hacking Behavior

Correlated Hacking Behavior (CHB) Analysis [WL01] employs architecture with bidirectional communication between IDSs and centralized correlator, so that the results of correlation can be sent back to IDSs to improve their detection rate and accuracy. The architecture supports

many different correlators, specifying only output interfaces. IDSs send information using IDMEF and receive data in a publish-subscribe model as a set of attribute-value pairs.

The CHB system uses seven heuristic algorithms, each of which generates numerical values $[0, 1]$ representing ultimately ad-hoc "hacking behavior". The system uses different correlation methods, namely: linear combination, 1-Rule and Bagging Method, Native Bayer, to obtain classifier discriminating between true and false positives.

The authors train and verify the model on DARPA 1999 data. The system proves to suppress some false positives with little detection-rate degradation. Out of tested correlation methods, 1-Rule and Bagging methods with weighted votes proved to be the best (false positives reduced from 46.2% to 14.3%). One should note, however, that the size of training and test data are definitely too small to generate meaningful results (e.g., the test set contained only 13 alerts).

### A.2.7 M2D2 Formal Data Model

The M2D2 model for IDS alert correlation [MMDD02] provides concepts and relations relevant to the security of an information system. It integrates four main types of information: information system characteristics, vulnerabilities, security tools and events which are described below:

**information system characteristics** contains information about the *topology* of the system in the form of a hypergraph. It also maps hostnames to IP addresses and service names to their numerical values.

The M2D2 model also contains information about *products*, i.e., logical entities executed on the host. Product is characterized by the following attributes: vendor, name of the product, product version and type of the product in some product classification.

Products running on the host are called a *configuration*, which is formally defined as a subset of products.

**vulnerabilities** are defined as latent errors present on a host, a flaw or weakness in a system design, implementation or management that could be exploited to violate the system security policy. It might be possible that vulnerabilities do not affect a single product, but a particular configuration of products (e.g., Apache webserver running on WindowsNT computer). In M2D2 a vulnerability affects the configuration, which means that a host is vulnerable if its configuration is a subset of a vulnerable configuration. Vulnerabilities have access requirements (e.g., physical access to target, user account, network access) and consequences (e.g., DoS, information disclosure, code execution, privilege gain). Such a vulnerability set is automatically built from the ICAT vulnerability database and uses CVE/CAN notations.

**security tools** comprise both IDSs and vulnerability scanners. The first group detects vulnerabilities being exploited, the latter when they are latent. M2D2 can model both host-/network-based as well as signature-/anomaly-based IDSs.

The model specifies the range of monitored hosts as well as *operational visibility* i.e., list of alerts a security tool can raise. These alerts are mapped to vulnerabilities if such mapping is possible.

**events** are reports about the existence or ongoing exploitation attempt of a vulnerability, generated by security tools. Events can be of two types: *alerts*, reported by IDSs, and *scans* reported by vulnerability scanners. The model divides events into subclasses e.g., IP event, TCP events, UDP events, log events, and can be further extended if necessary.

The M2D2 model can be used to facilitate event aggregation and correlation. For example, based on attack and host definitions one can say whether an event reported by an IDS indicates an attack which can be successful or get a list of hosts vulnerable to a particular attack. M2D2 can also be used to aggregate similar alerts and correlate attacks based on prerequisites of intrusions.

The main contribution of M2D2 is the formal representation of security information to make security event correlation easier and more reliable. In spite of being formal, the model tries to reuse and unify existing concepts and components.

### A.2.8 Statistical Correlation Models

Qin and Lee [QL03, QL04] present an alert correlation system combining a Bayesian correlation system (naive Bayes) with a statistical correlation system using Granger Causality Test (GCT) [Gra69], a time series-based causal analysis algorithm. Based on the results of this analysis the GCT module constructs a correlation graph. The system aims at discovering new attack relationships as long as alerts of attacks have a statistical relationship. In contrast, as the structure of the network is predetermined, the Bayes-based correlation module can discover alerts that have direct "causal" relationships according to domain knowledge. The Bayesian network used is necessarily simplistic and uses unrealistic independence assumptions, however it is the only alert-correlation system to date that uses a real probabilistic approach.

The authors evaluate the system based on two artificial datasets. While the methods can be used for discovering new relationships in alerts it is currently not clear how well they perform on real datasets. The authors also report high false-causality rate, which suggests that the system can only be used by very specialized domain experts after manual tuning.

### A.2.9 Comprehensive IDS Alert Correlation

More recently, Valeur et al. [VVCK04] propose and evaluate a 10-step Comprehensive IDS Alert-Correlation (CIAC) system. The system processes alerts from multiple intrusion detection systems in the following way:

**Step 1 and 2: Normalization & Preprocessing.** In this step alerts from multiple sources are converted to a common IDMEF [CD03] format and normalized so that alerts from different IDSs are compatible.

**Step 3: Alert Fusion.** In the fusion step, alerts triggered by different IDSs are combined into a single meta-alert. The fusion is done based on the equality of attributes and timestamp proximity.

**Step 4: Alert Verification** In this step, the system verifies if the target is vulnerable to the attack reported by running a corresponding Nessus [Der03] script.

**Step 5: Thread Reconstruction.** Performs a heuristic grouping of alerts with equal source and destination IP addresses in a heuristically chosen time window. This activity groups alerts triggered by a single attacker against a single target into *meta-alerts*.

**Step 6: Session Reconstruction.** In this step, alerts from host and network-based IDSs are combined into a single meta-alert.

**Step 7: Focus Recognition.** Similarly to Step 5, it performs a heuristic grouping of meta-alerts with equal source IP addresses or destination IP addresses. This aggregates alerts triggered by a single attacker attacking many hosts, or many attackers attacking a single victim.

**Step 8: Multi-Step Correlation.** In this step high-level scenarios are discovered, e.g., "recon-breakin-escalate" and "island-hopping". Those scenarios express a domain knowledge in intrusion detection and have to be predefined beforehand.

**Step 9 and 10: Impact Analysis & Prioritization.** In these steps the correlator discovers the impact a given alert has on the infrastructure, including dependencies and assigns corresponding priorities.

The authors evaluate CIAC on five artificial and two realistic datasets and show that the system reduces the number of alerts by up to 99.6%. The architecture of the system is based on previously published ideas, however, it is the only system combining all of these ideas in such a coherent manner.

Unfortunately, the system requires significant infrastructure and integration work, which should not be underestimated. Currently, existing infrastructures do not provide such a good level of integration, e.g., in many cases even a simple correlation of alerts with vulnerabilities is not performed. Finally, the system has been tested with datasets, which have no or very few false positives. Hence, its performance in real environments is not known. Nevertheless, the paper presents one of the most comprehensive and realistic evaluations of alert correlation systems to date.

In our classification, a system like CIAC spans over levels 2 and 3 in our classification and can be cascaded with ALAC, a level-4 system.

# Appendix B

# Abstaining Classifier Evaluation Results

Figures B.1 and B.2 show the misclassification cost improvements for all datasets and illustrate the experiments described in Section 6.6.3. Similarly, Figures B.3 and B.4 show the misclassification cost improvements for all datasets and illustrate the experiments described in Section 6.6.4. Finally, Figures B.5 and B.6 are the results of the experiments described in Section 6.6.4.

Figures B.7, B.8, B.9 and B.10 illustrate false-positive rates, false-negative rates, abstention window and the fraction of discarded alerts for experiments with ALAC+ using abstaining classifiers performed in Section 7.2.

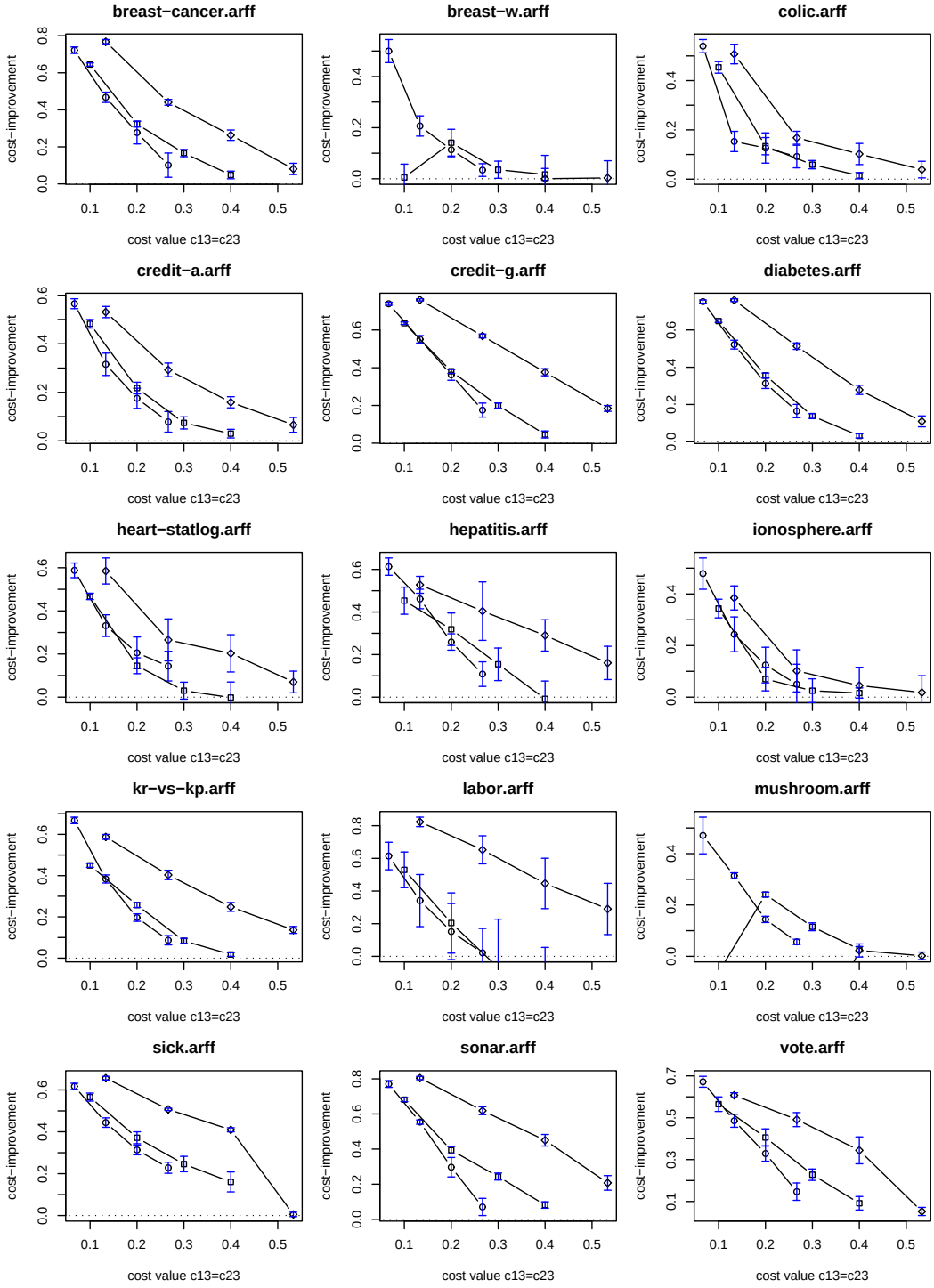



Figure B.1: Cost-Based Model: Experimental results with abstaining classifiers—relative cost improvement.

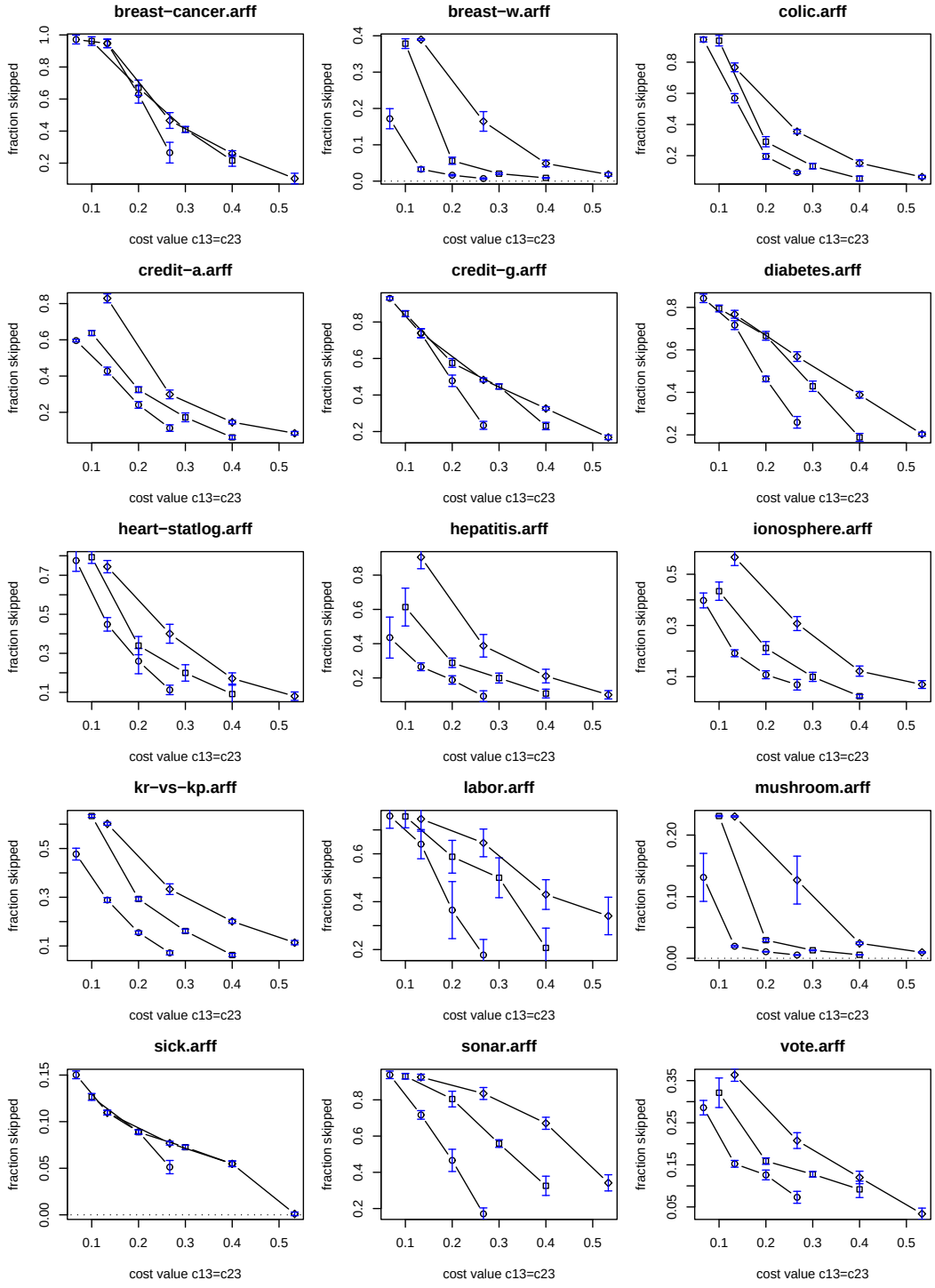



Figure B.2: Cost-based model: Experimental results with abstaining classifiers—fraction of skipped instances.

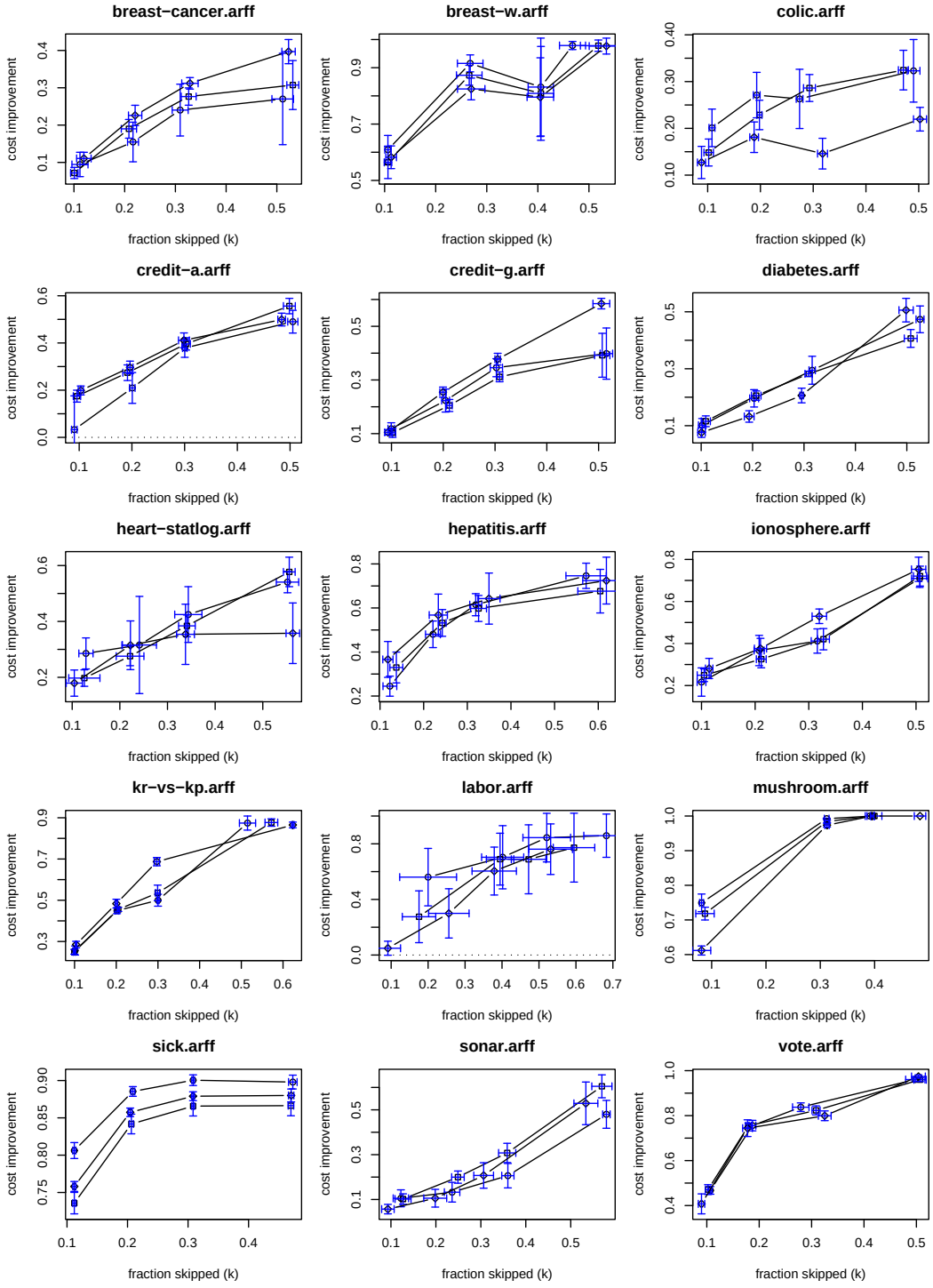



Figure B.3: Bounded model: Experimental results with abstaining classifiers—relative cost improvement.

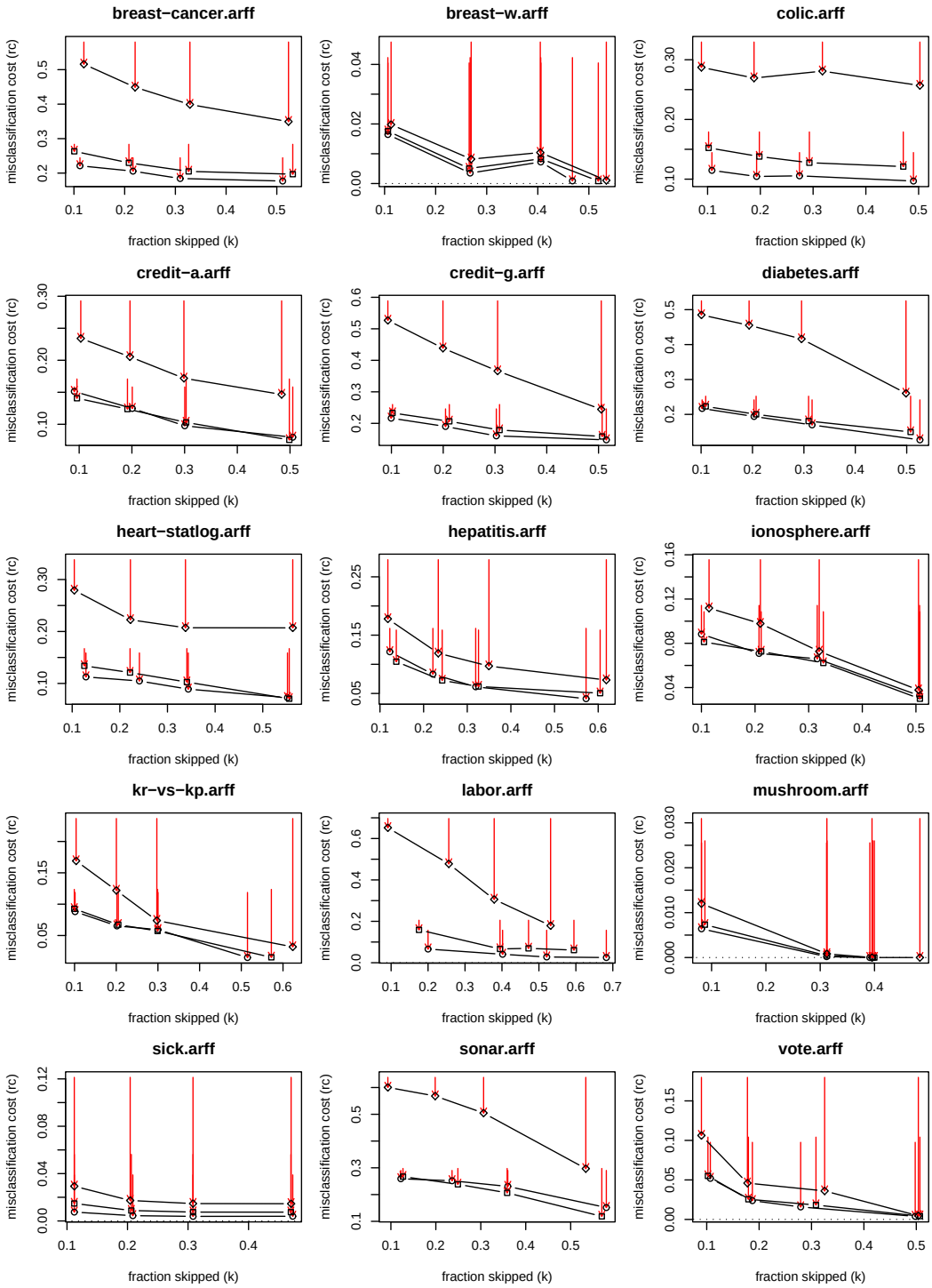



Figure B.4: Bounded model: Experimental results with abstaining classifiers—absolute cost values.

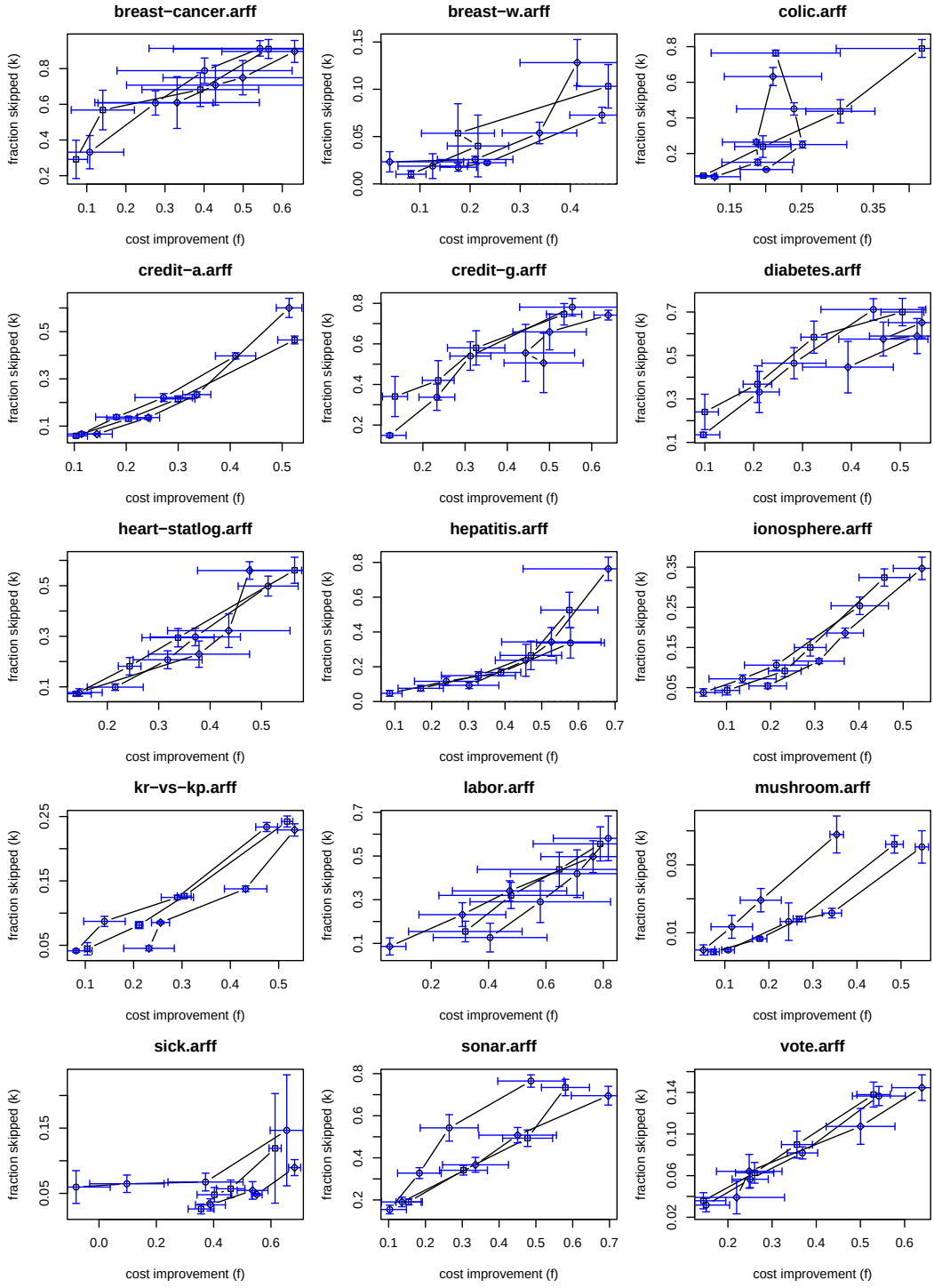



Figure B.5: Expected improvement model: Experimental results with abstaining classifiers—desired relative cost improvement vs. fraction of nonclassified instances.

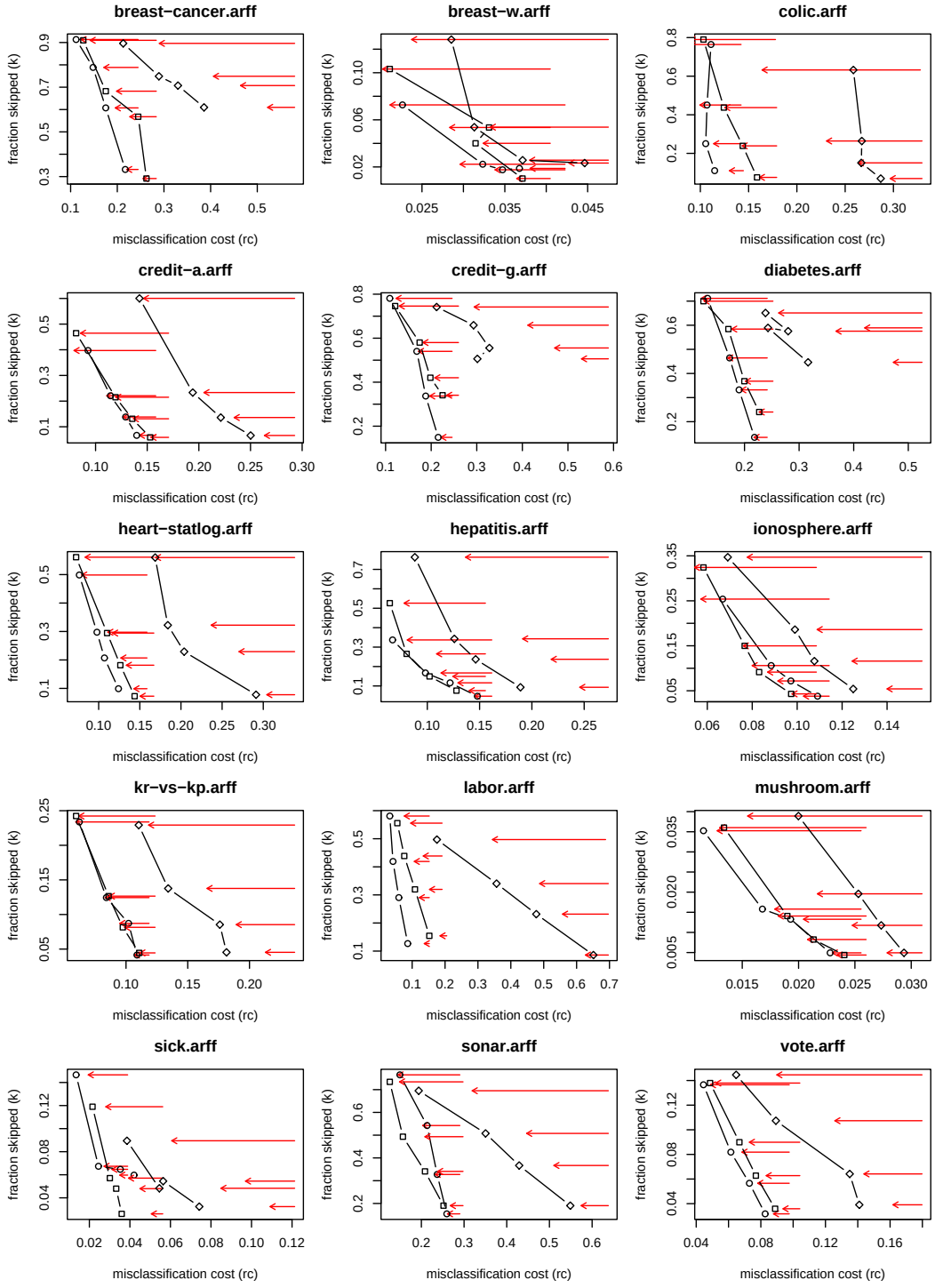



Figure B.6: Expected improvement model: Experimental results with abstaining classifiers—desired absolute cost improvement vs. fraction of nonclassified instances.

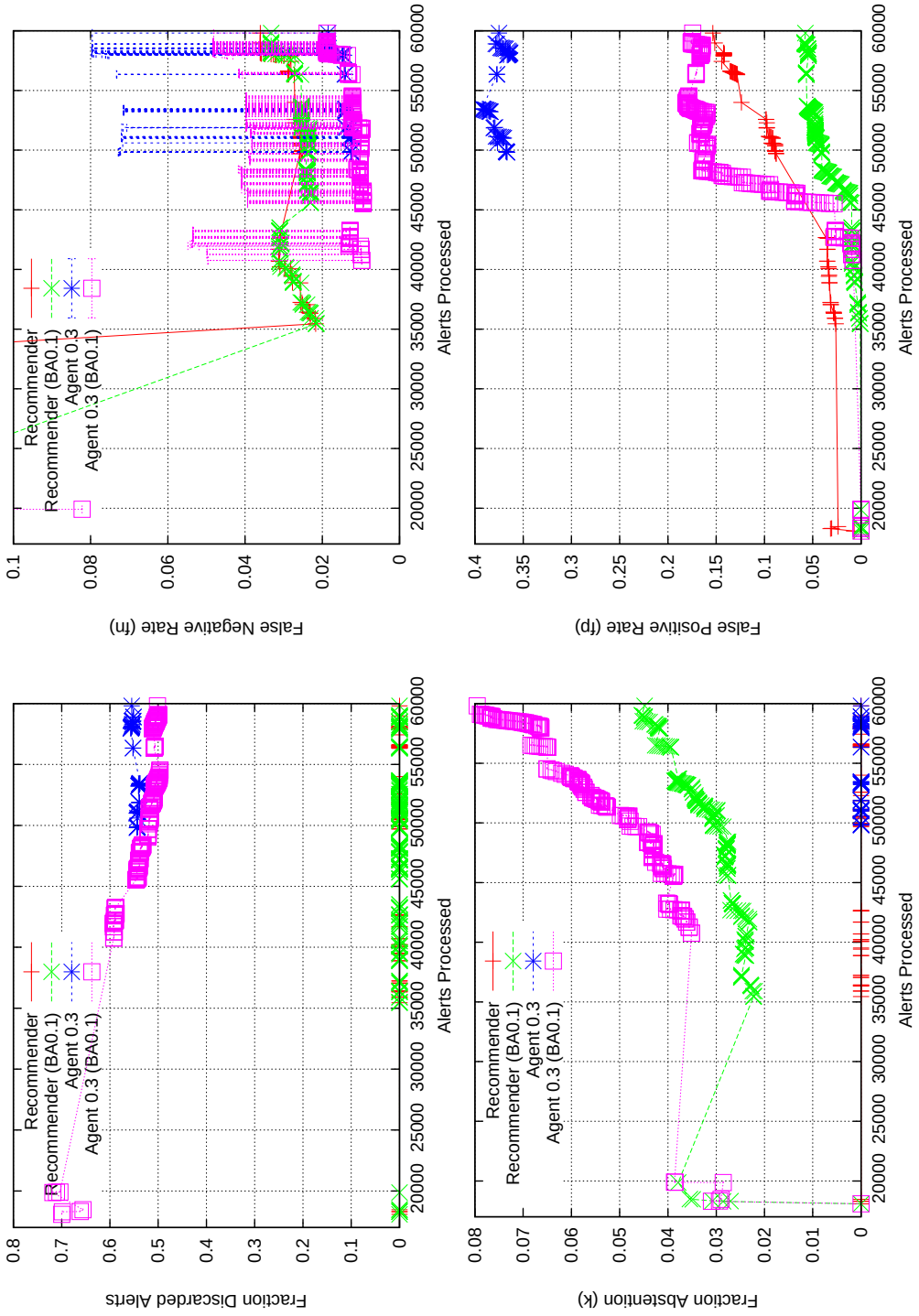



Figure B.7: ALAC+, DARPA 1999 Data Set, BA0.1: False-positive rates, false-negative rates, the abstention window and the fraction of discarded alerts in both agent and recommender modes.

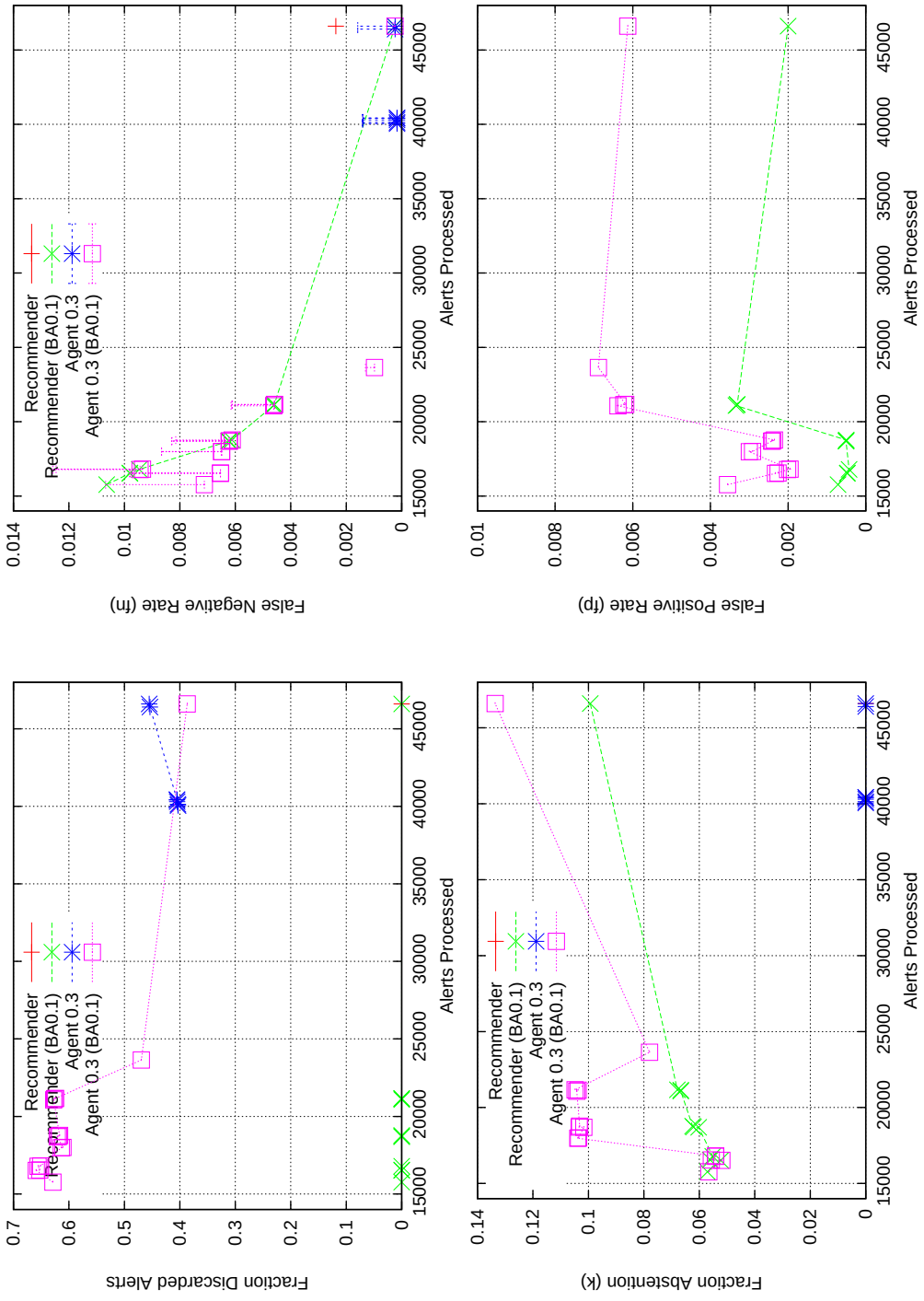



Figure B.8: ALAC+, Data Set B, BA0.1: False-positive rates, false-negative rates, the abstention window and the fraction of discarded alerts in both agent and recommender modes.

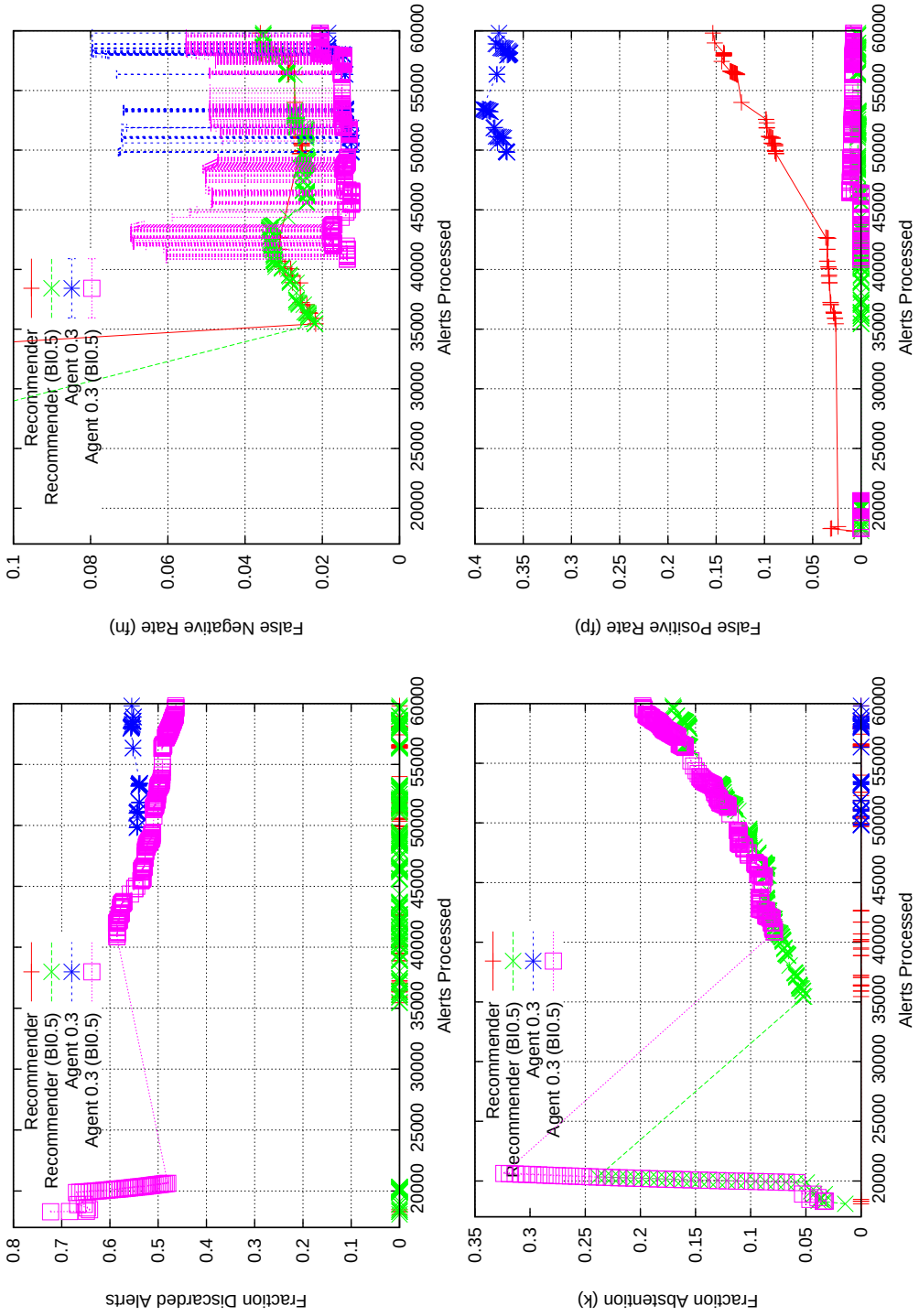



Figure B.9: ALAC+, DARPA 1999 Data Set, BI0.5: False-positive rates, false-negative rates, the abstention window and the fraction of discarded alerts in both agent and recommender modes.

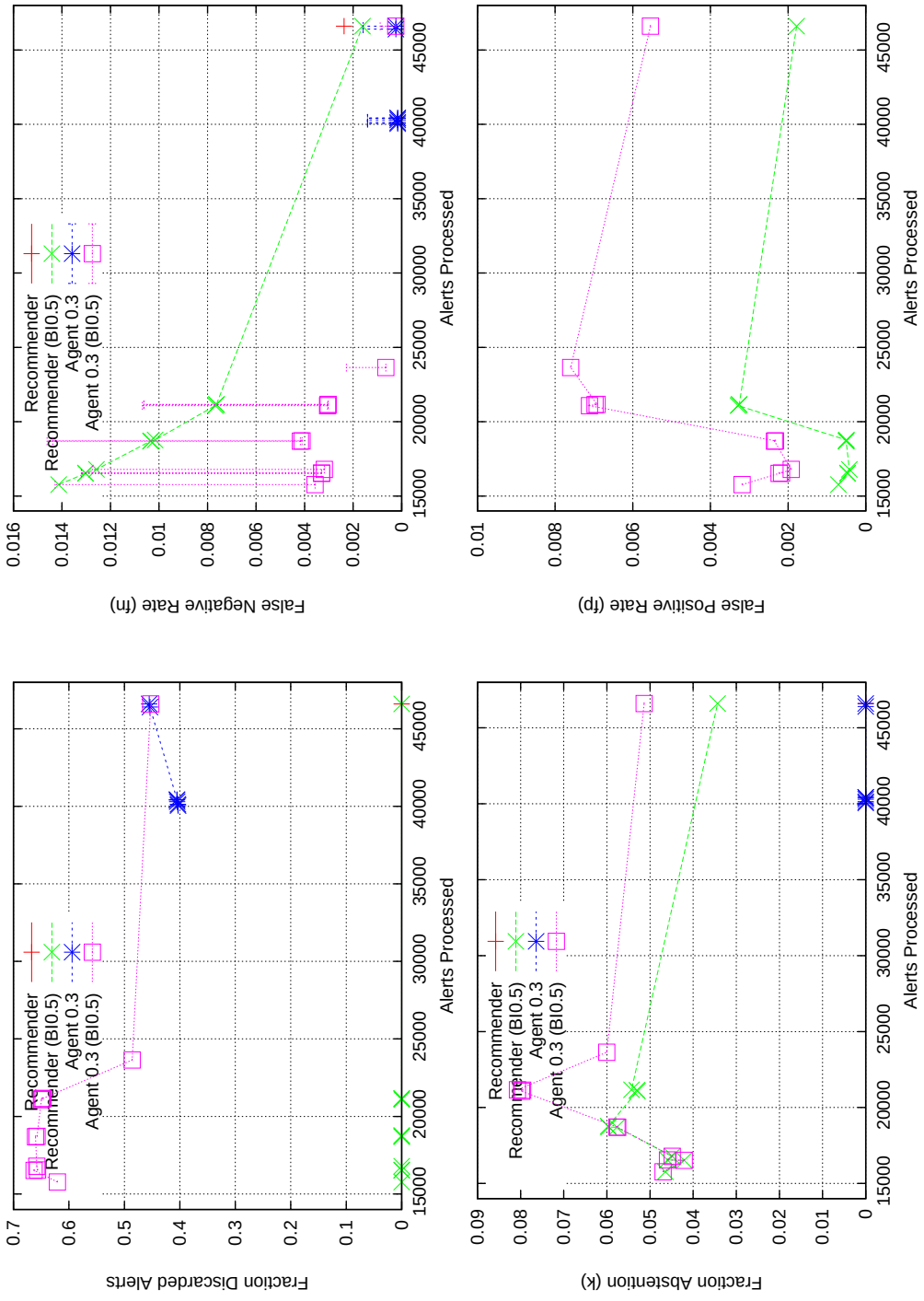



Figure B.10: ALAC+, Data Set B, BI0.5: False-positive rates, false-negative rates, the abstention window and the fraction of discarded alerts in both agent and recommender modes.

# Appendix C

# Clustering MSSP Datasets Results

In this chapter we present detailed results obtained with 20 MSSP datasets. The figures included here are discussed in and referenced from Chapter 8.

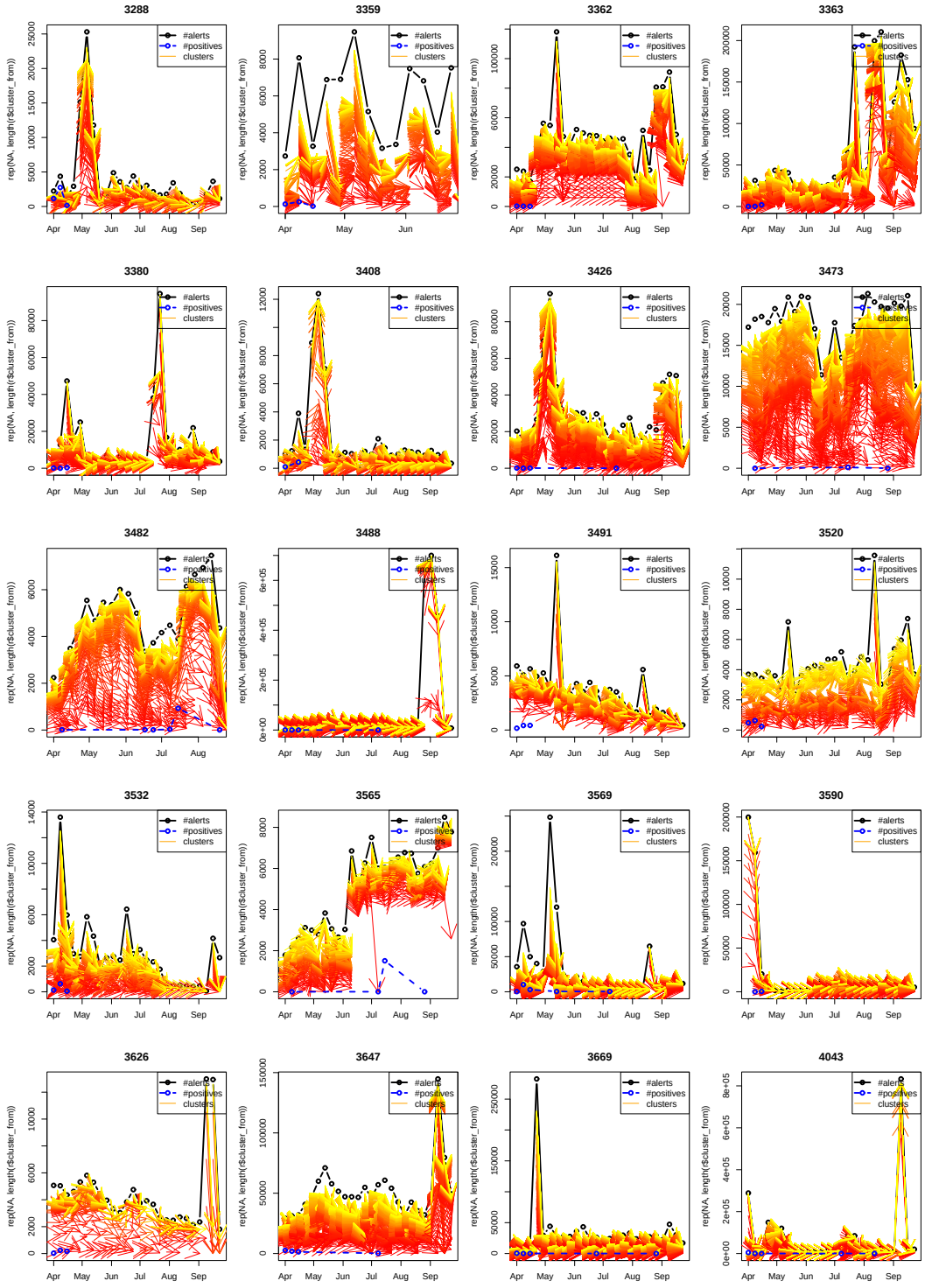



Figure C.1: Cluster persistency for 20 MSSP customers—absolute values. X and Y axes labels are the same as in Figs. 8.7a and 8.7c.

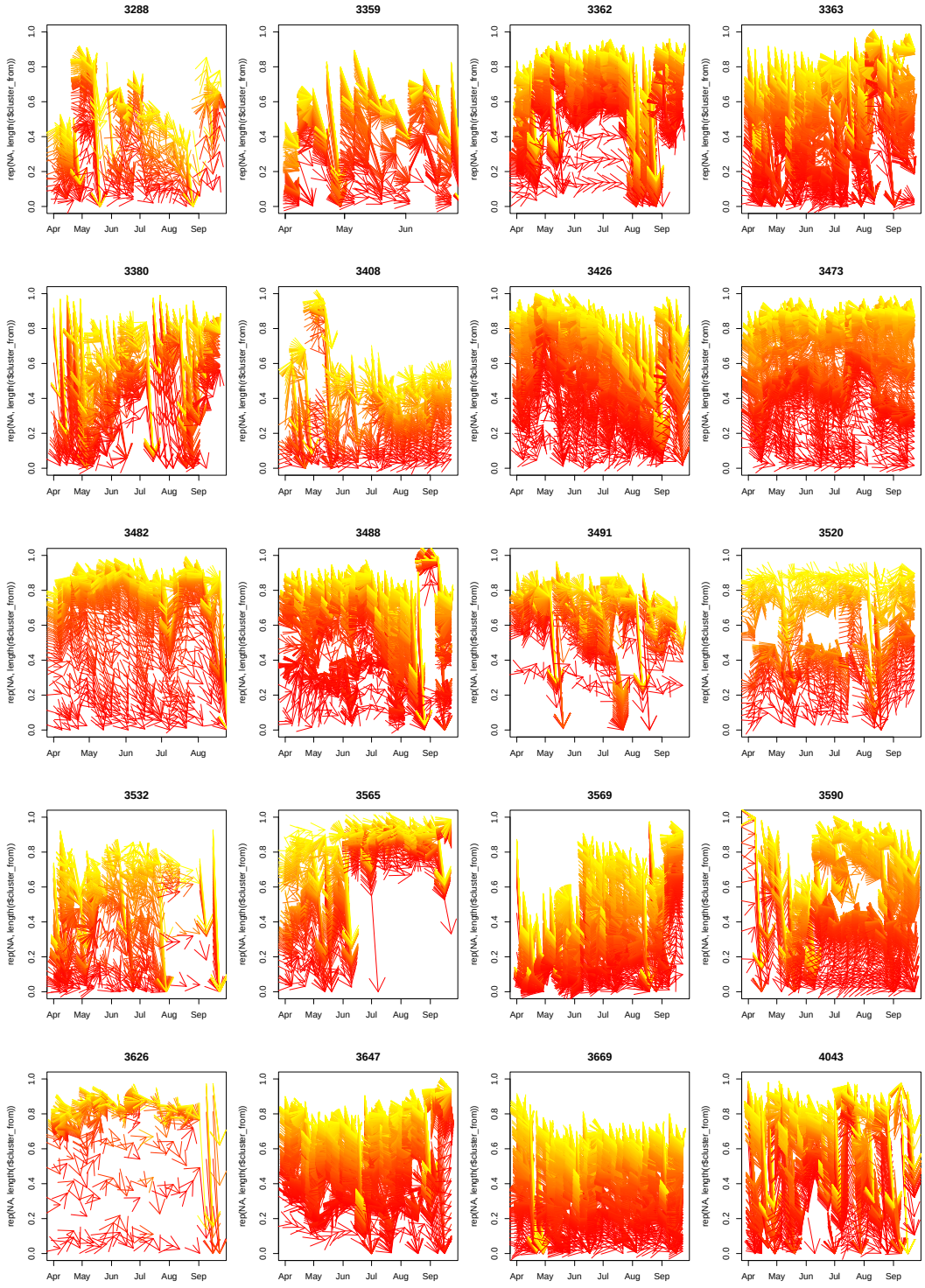



Figure C.2: Cluster persistency for 20 MSSP customers—relative values. X and Y axes labels are the same as in Figs. 8.7b and 8.7d.

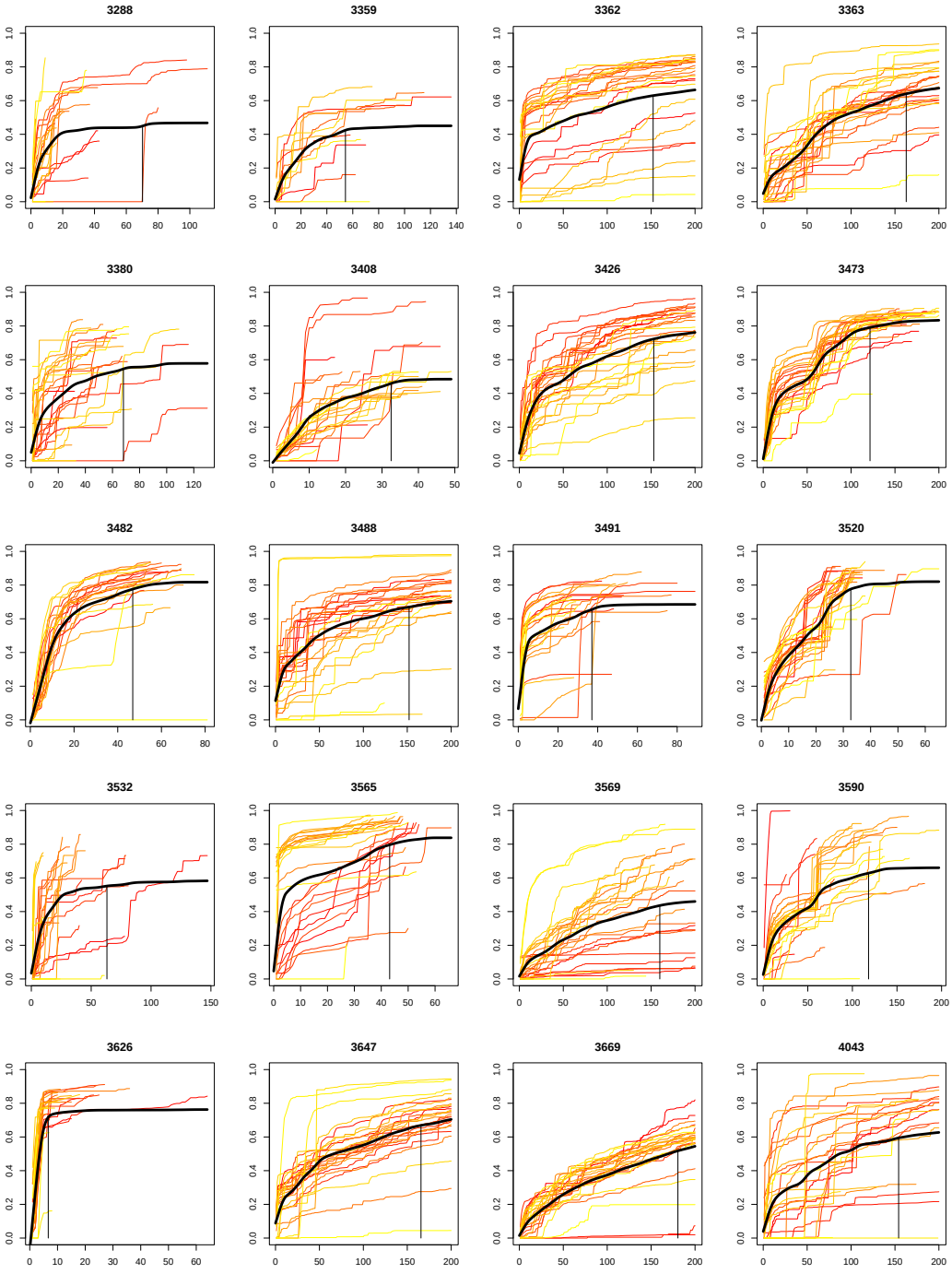



Figure C.3: Estimating the fraction of instances clustered as a function of the number of clusters learned for 20 MSSP customers. X and Y axes labels are the same as in Figs. 8.8a and 8.8c.

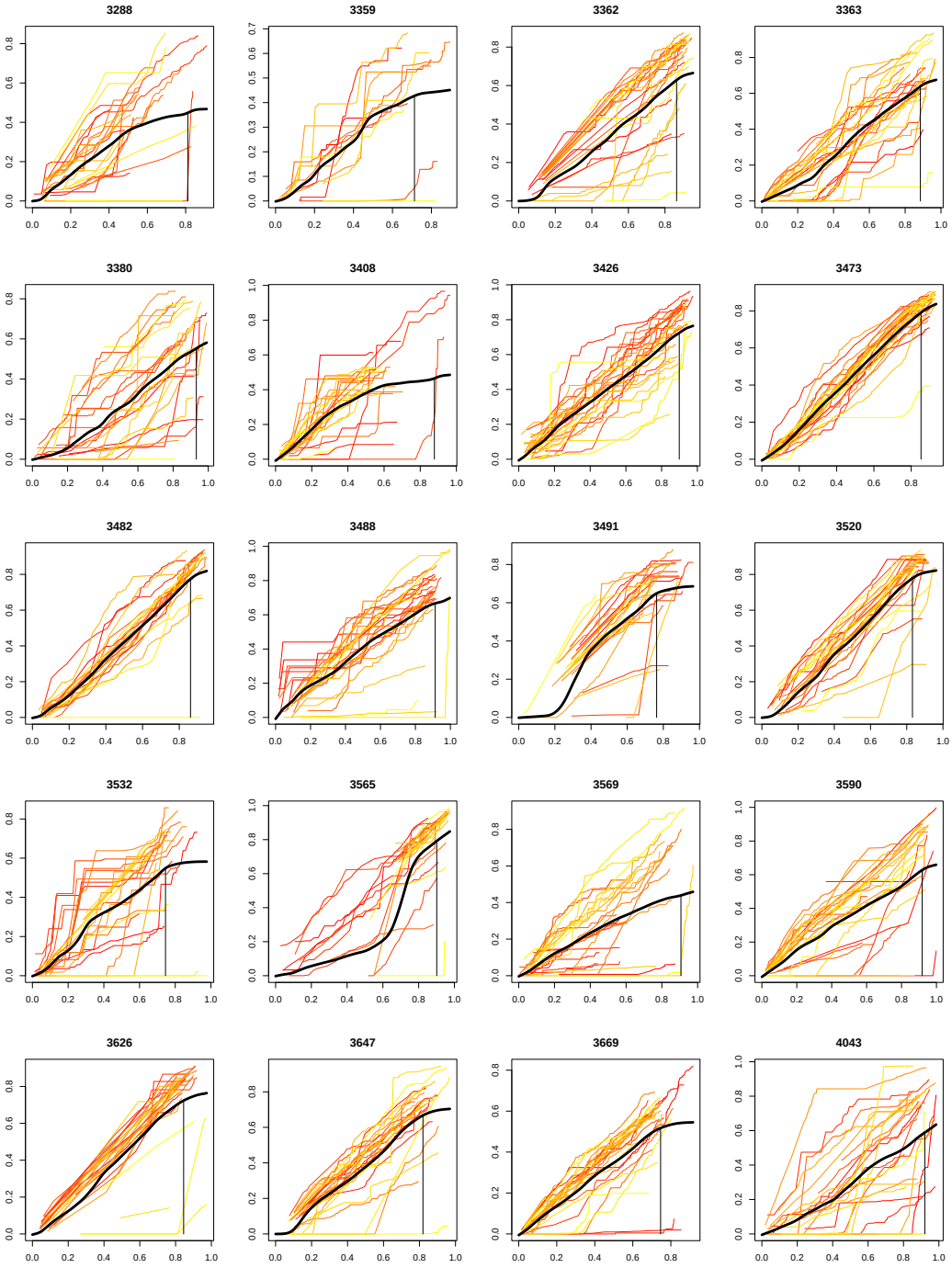



Figure C.4: Estimating the fraction of instances clustered as a function of the fraction of instances filtered for 20 MSSP customers. X and Y axes labels are the same as in Figs. 8.8b and 8.8d.

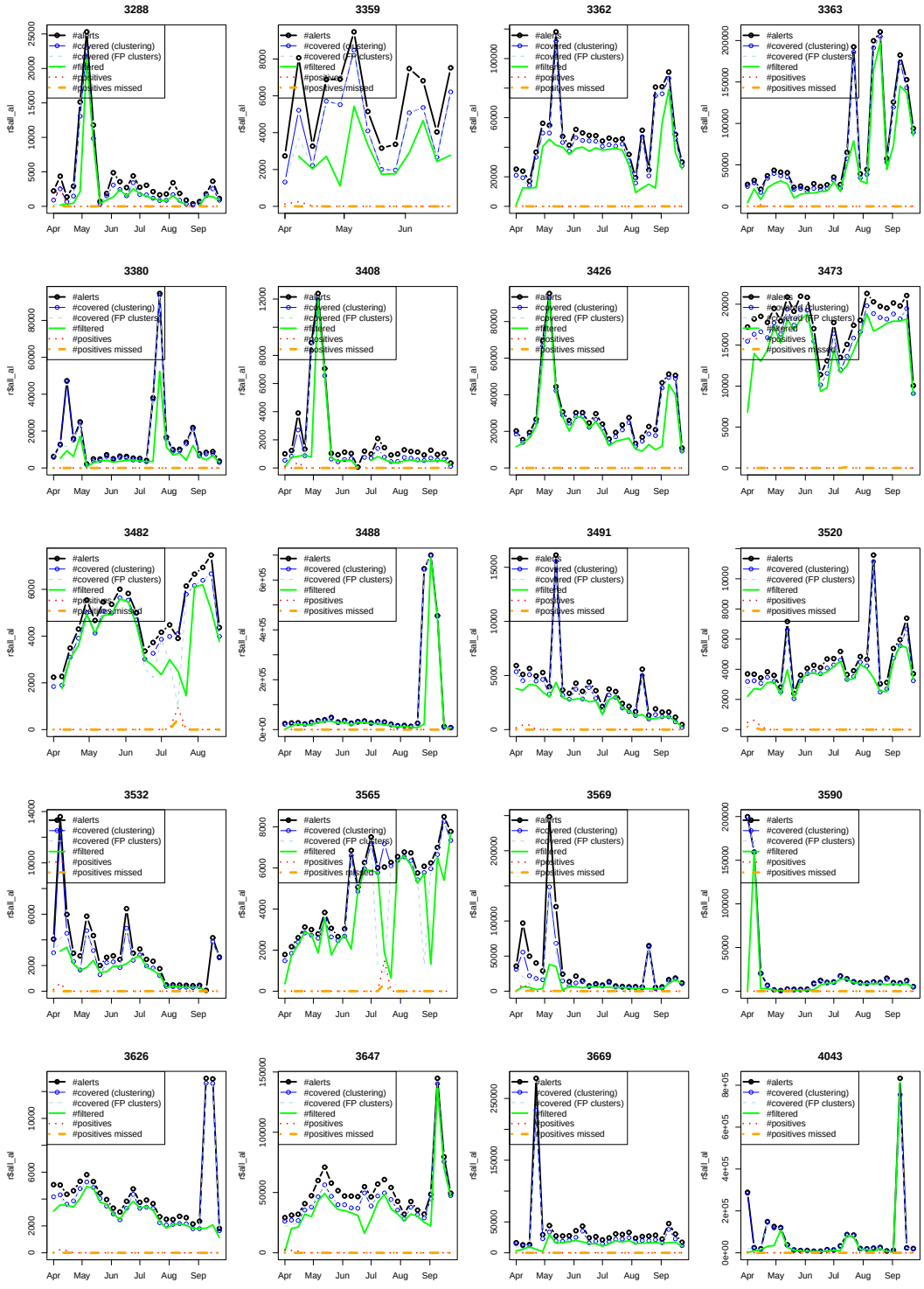



Figure C.5: Cluster filtering for 20 MSSP customers—absolute values. X and Y axes labels are the same as in Figs. 8.9a and 8.9c.

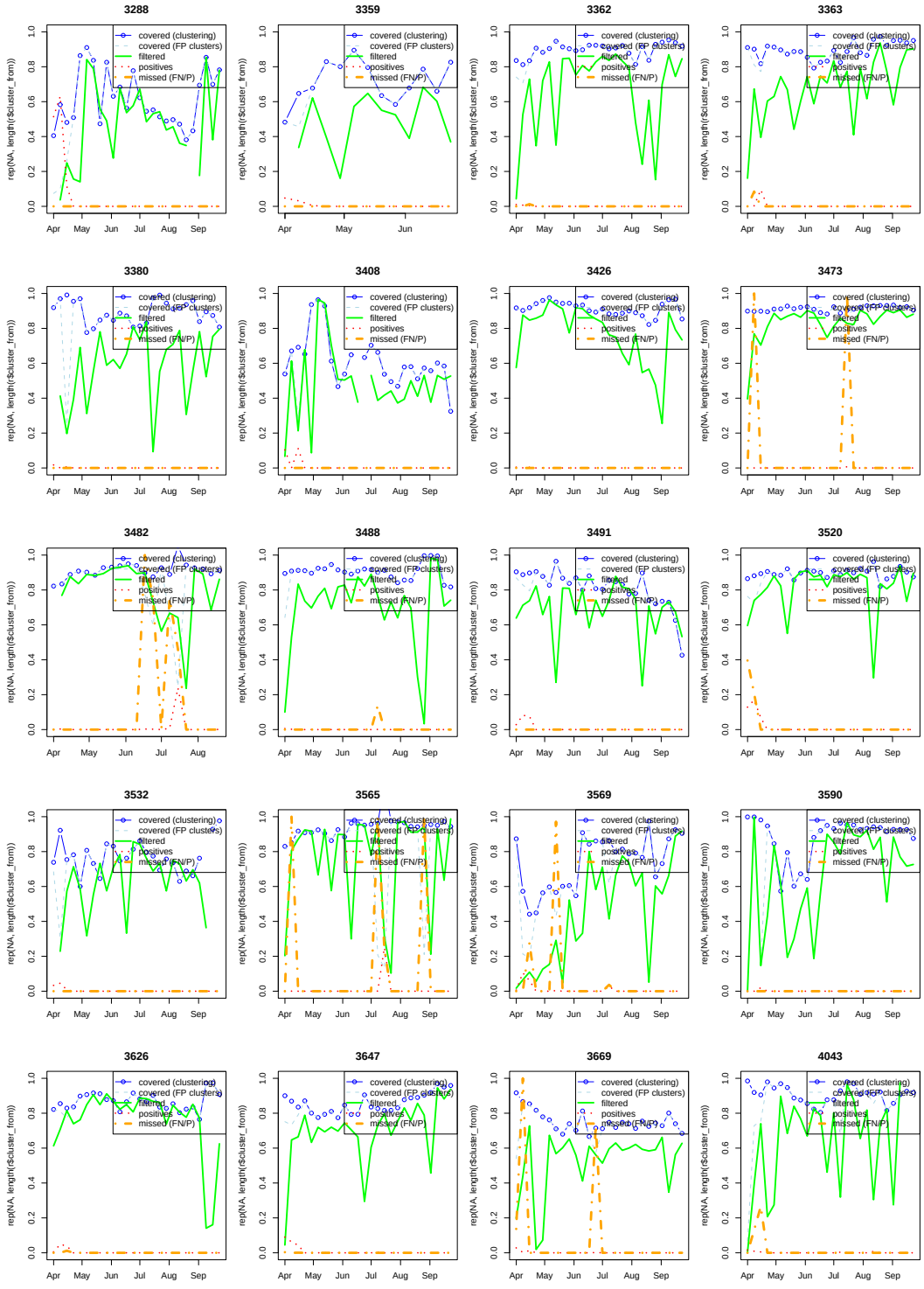



Figure C.6: Cluster filtering for 20 MSSP customers—relative values. X and Y axes labels are the same as in Figs. 8.9b and 8.9d.

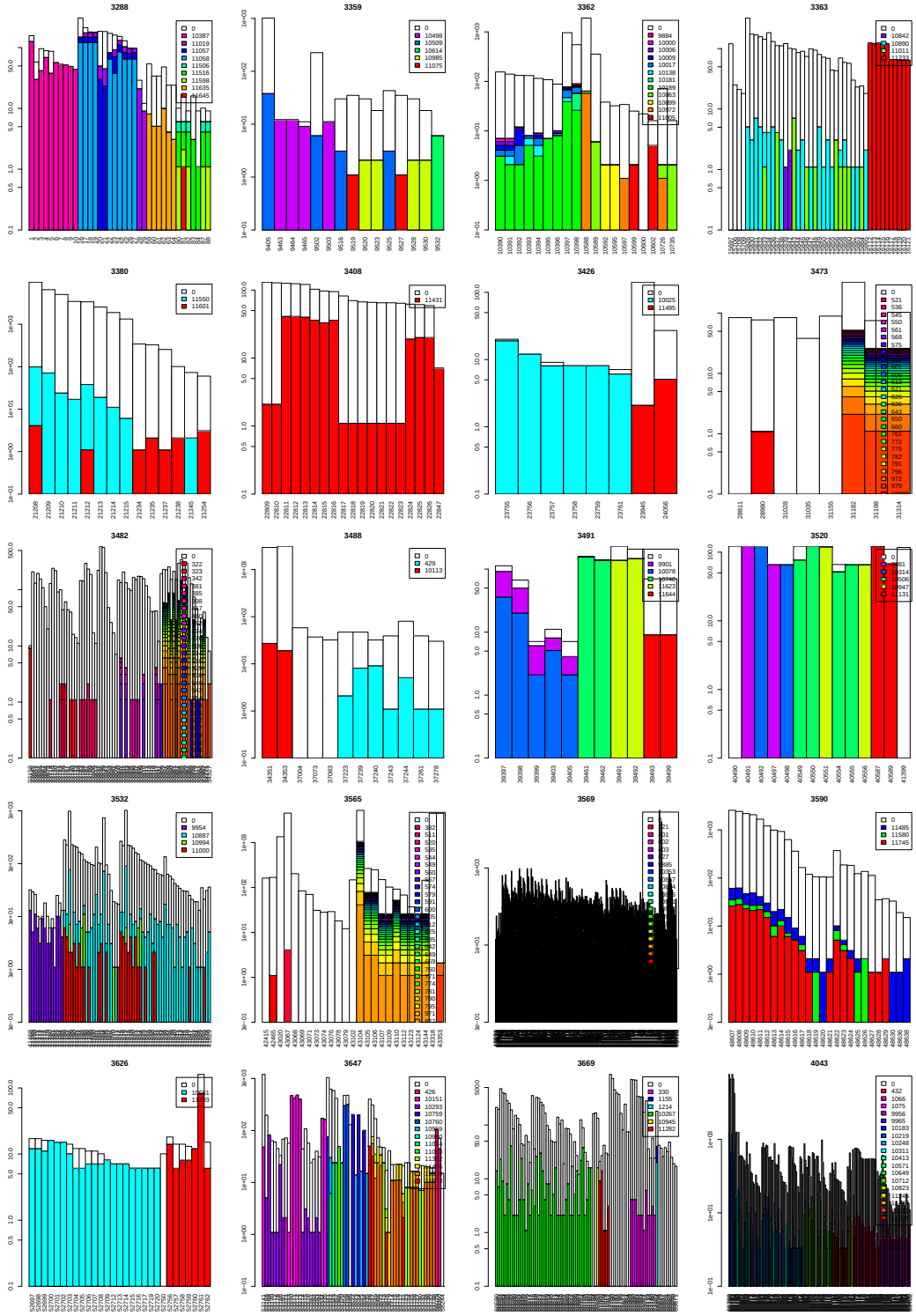



Figure C.7: Clustering precision for 20 MSSP customers—clustering stage. X and Y axes are the same as in Fig. 8.11a.

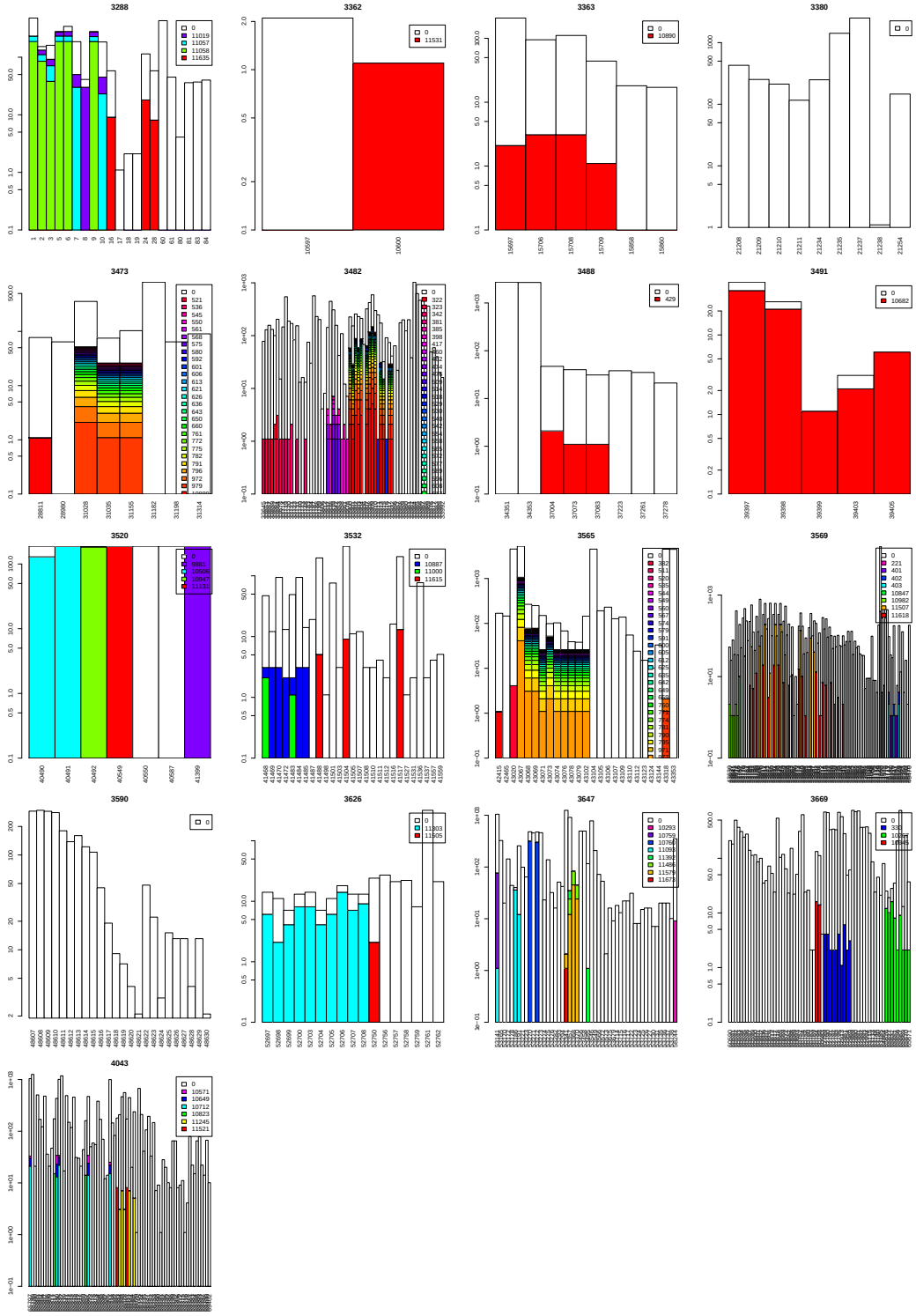



Figure C.8: Clustering precision for 20 MSSP customers—filtering stage. X and Y axes are the same as in Fig. 8.11b.

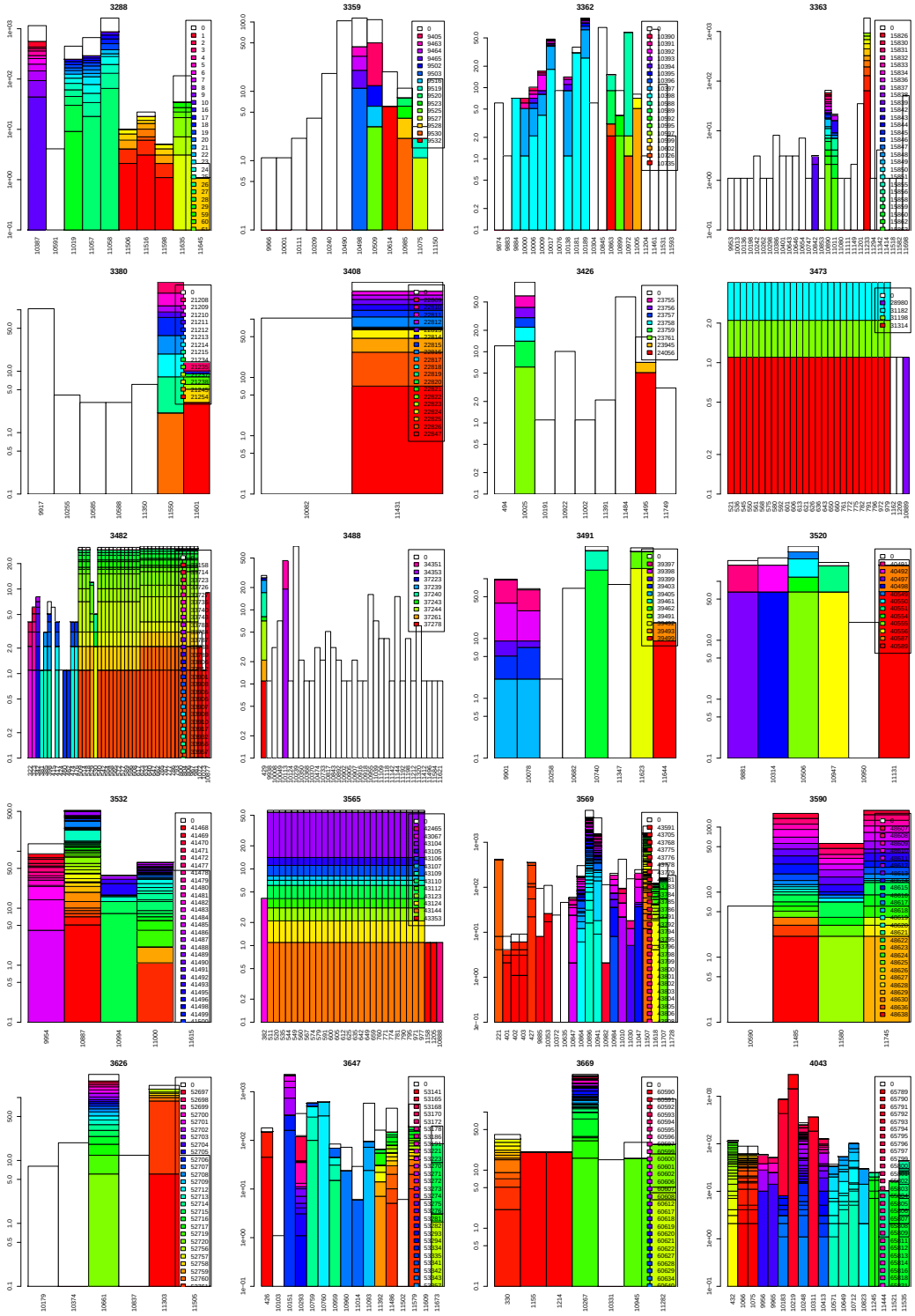



Figure C.9: Clustering recall for 20 MSSP customers—clustering stage. X and Y axes are the same as in Fig. 8.11c.

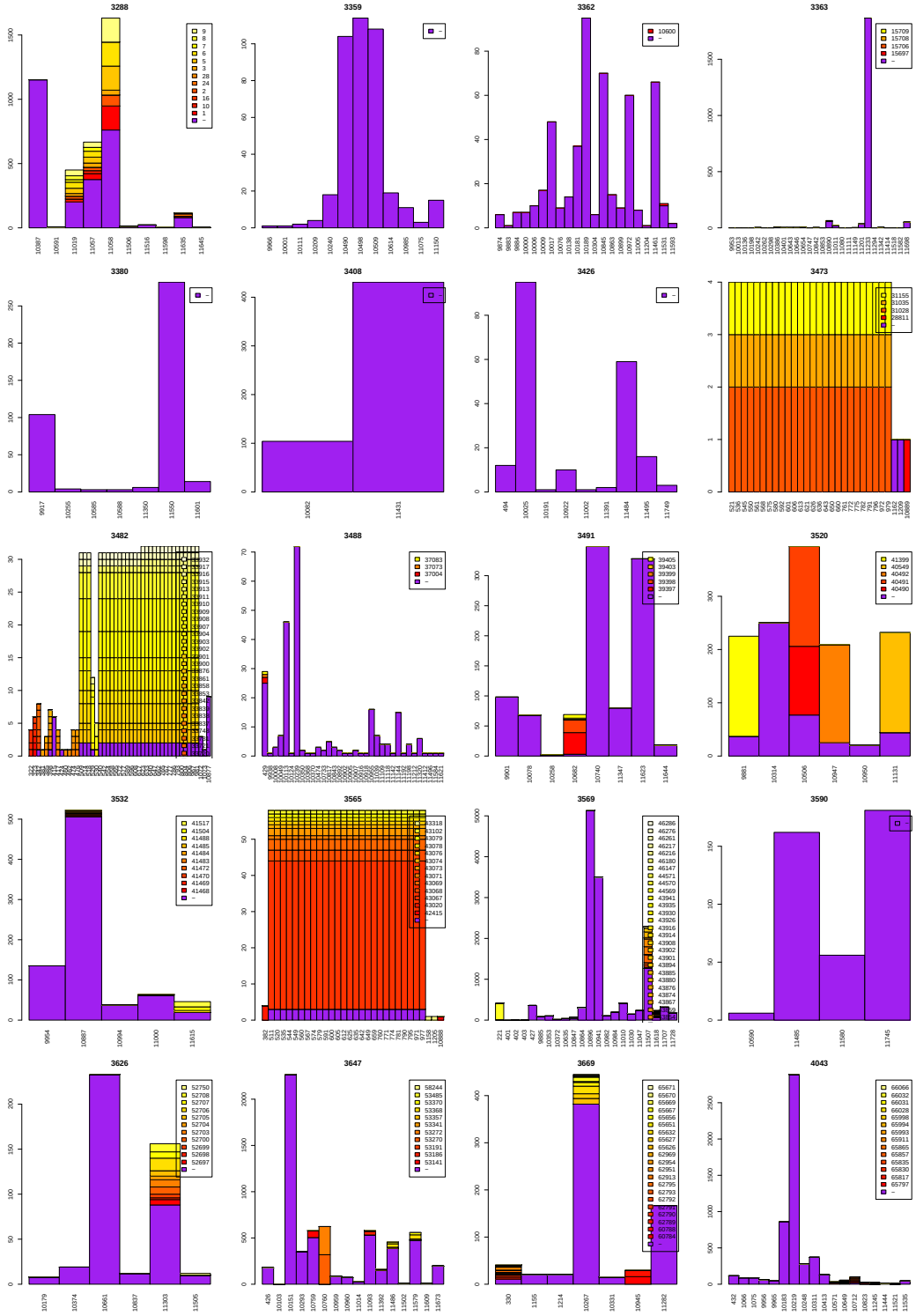



Figure C.10: Clustering recall for 20 MSSP customers—filtering stage. X and Y axes are the same as in Fig. 8.11d.

# Bibliography


[Aes] Aesop. The boy who cried "Wolf!". Folks tale. This version was taken and adapted from the web page at `http://www.rickwalton.com/folktale/bryant19.html`.

[AIS93] Rakesh Agrawal, Tomasz Imielinski, and Arun N. Swami. Mining association rules between sets of items in large databases. In *Proceedings of the ACM-SIGMOD 1993 International Conference on Management of Data*, pages 207–216, Washington, D.C., 1993.

[AKA91] David W. Aha, Dennis Kibler, and Marc K. Albert. Instance-based learning algorithms. *Machine Learning*, 6(1):37–66, 1991.

[AMS+96] Rakesh Agrawal, Heikki Mannila, Ramakrishnan Srikant, Hannu Toivonen, and A. Inkeri Verkamo. Fast discovery of association rules. In *Advances in Knowledge Discovery and Data Mining*, pages 307–328. AAAI/MIT Press, 1996.

[And80] James P. Anderson. Computer security threat monitoring and surveillance. Technical report, James P. Anderson Co., 1980.

[Axe99] Stefan Axelsson. The base-rate fallacy and its implications for the intrusion detection. In *Proceedings of the 6th ACM Conference on Computer and Communications Security*, pages 1–7, Kent Ridge Digital Labs, Singapore, 1999.

[Axe05] Stefan Axelsson. *Understanding Intrusion Detection Through Visualization*. PhD thesis, Chalmers University of Technology, 2005.

[Bel92] Steven M. Bellowin. There be dragons. In *Proceedings of the 3rd USENIX Security Symposium*, Baltimore, MD, 1992.

[Bel93] Steven M. Bellowin. Packets found on an Internet. *Computer Communications Review*, 23(3):26–31, 1993.

[BHC+00] Eric Bloedorn, Bill Hill, Alan Christiansen, Clem Skorupka, Lisa Talbot, and Jonathan Tivel. Data mining for improving intrusion detection. Technical report, MITRE Corporation, 2000.

[Blo98] Hendrik Blockeel. *Top-Down Induction of First Order Logical Decision Trees*. PhD thesis, Katholieke Universiteit Leuven, 1998.

[Bra87] Andrew P. Bradley. The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30(7):1145–1159, 1987.

[CAMB02] Frédéric Cuppens, Fabien Autrel, Alexandre Miège, and Salem Benferhat. Correlation in an intrusion detection process. In *Proceedings SÉcurité des Communications sur Internet (SECI02)*, pages 153–171, 2002.

[CD03] D. Curry and H. Debar. Intrusion detection message exchange format data model and extensible markup language (xml) document type definition. Technical report, Internet Engineering Task Force, Intrusion Detection Working Group, 2003.

[Ces90] Bojan Cestnik. Estimating probabilities: A crucial task in machine learning. In *Proceedings of the Ninth European Conference on Artificial Intelligence (ECAI-1990)*, pages 147–149, Stockholm, Sweden, 1990.

[CG04] Ira Cohen and Moises Goldszmidt. Properties and benefits of calibrated classifiers. In Jean-Francois Boulicaut, Floriana Esposito, Fosca Giannotti, and Dino Pedreschi, editors, *Proceedings of PKDD 2004: 8th European Conference on Principles and Practice of Knowledge Discovery in Databases*, volume 3202 of *Lecture Notes in Computer Science*, pages 125–136. Springer-Verlag, 2004.

[Cho70] C. Chow. On optimum recognition error and reject tradeoff. *IEEE Transactions on Information Theory*, 16(1):41–46, 1970.

[CM02] Frédéric Cuppens and Alexandre Miège. Alert correlation in a cooperative intrusion detection framework. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pages 202–215, 2002.

[CN89] Peter Clark and Tim Niblett. The CN2 induction algorithm. *Machine Learning*, 3:261–283, 1989.

[CO00] Frédéric Cuppens and Rodolphe Ortalo. LAMBDA: A language to model a database for detection of attacks. In *Recent Advances in Intrusion Detection (RAID2000)*, volume 1907 of *Lecture Notes in Computer Science*, pages 197–216. Springer-Verlag, 2000.

[Coh95] William W. Cohen. Fast effective rule induction. In Armand Prieditis and Stuart Russell, editors, *Proceedings of the 12th International Conference on Machine Learning*, pages 115–123, Tahoe City, CA, 1995. Morgan Kaufmann Publishers.

[Coh96] William W. Cohen. Learning trees and rules with set-valued features. In *AAAI/IAAI, Vol. 1*, pages 709–716, 1996.

[Com91] Comission of the European Communities. *Information Technology Security Evaluation Criteria*. Version 2.1, 1991.

[Cuf05] Andy Cuff. Talisker intrusion detection prevention systems. Web page at `http://www.networkintrusion.co.uk/ids.htm`, 2005.

[Cup01] Frédéric Cuppens. Managing alerts in a multi-intrusion detection environment. In *Proceedings 17th Annual Computer Security Applications Conference*, pages 22–31, New Orleans, 2001.

[Dar01] Dark Tangent. DEF CON 9. Web page at `http://www.defcon.org/html/defcon-9/defcon-9-post.html`, 2001.

[DC01] Olivier Dain and Robert K. Cunningham. Fusing a heterogeneous alert stream into scenarios. In *Proceedings of the 2001 ACM Workshop on Data Mining for Security Application*, pages 1–13, Philadelphia, PA, 2001.

[Den87] Dorothy E. Denning. An intrusion detection model. *IEEE Transactions on Software Engineering*, SE-13(2):222–232, 1987.

[Der03] Renaud Deraison. The Nessus Project. Web page at `http://www.nessus.org`, 2000-2003.

[DG06] Jesse Davis and Mark Goadrich. The relationship between precision-recall and ROC curves. In *Proceedings of the Twenty-Third International Conference on Machine Learning (ICML-2006)*, page (to appear), Pittsburgh, PA, 2006.

[DH00] Chris Drummond and Robert C. Holte. Explicitly representing expected cost: An alternative to ROC representation. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 198–207. ACM Press, 2000.

[Die98] Thomas G. Dietterich. Approximate statistical test for comparing supervised classification learning algorithms. *Neural Computation*, 10(7):1895–1923, 1998.

[DK03] Luc De Raedt and Kristian Kersting. Probabilistic logic learning. *ACM-SIGKDD Explorations, special issue on Multi-Relational Data Mining*, 5(1):31–48, 2003.

[Dom99] Pedro Domingos. Metacost: A general method for making classifiers cost-sensitive. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 155–164, San Diego, CA, 1999.

[DP97] P. Domingos and M. Pazzani. On the optimality of the simple Bayesian classifier under zero-one loss. *Machine Learning*, 23(2–3):103–130, 1997.

[DW01] Hervé Debar and Andreas Wespi. Aggregation and correlation of intrusion-detection alerts. In *Recent Advances in Intrusion Detection (RAID2001)*, volume 2212 of *Lecture Notes in Computer Science*, pages 85–103. Springer-Verlag, 2001.

[DWV99] Harris Drucker, Donghui Wu, and Vladimir N. Vapnik. Support vector machines for spam categorization. *IEEE Transactions on Neural Networks*, 10(5):1048–1054, 1999.

[EDQ96] Babak Esfandiari, Gilles Deflandre, and Joël Quinqueton. An interface agent for network supervision. In *Proceedings of the ECAI-96 Workshop on Intelligent Agents for Telecom Applications*, Budapest, Hungary, 1996.

[Fan01] Wei Fan. *Cost-Sensitive, Scalable and Adaptive Learning Using Ensemble-based Methods*. PhD thesis, Columbia University, 2001.

[Faw03] Tom Fawcett. ROC graphs: Notes and practical considerations for researchers (HPL-2003-4). Technical report, HP Laboratories, 2003.

[FFHO04] C. Ferri, P. Flach, and J. Hernández-Orallo. Delegating classifiers. In *Proceedings of 21th International Conference on Machine Leaning (ICML-2004)*, pages 106–110, Alberta, Canada, 2004. Omnipress.

[FHO04] C. Ferri and J. Hernández-Orallo. Cautious classifiers. In *Proceedings of ROC Analysis in Artificial Intelligence, 1st International Workshop (ROCAI-2004)*, pages 27–36, Valencia, Spain, 2004.

[Fla03] Peter Flach. The geometry of ROC space: Understanding machine learning metrics through roc isometrics. In *Proceedings of the Twentieth International Conference on Machine Learning (ICML-2003)*, pages 194–201, Washington, DC, 2003. AAAI Press.

[Fla04] Peter Flach. The many faces of roc analysis in machine learning. Tutorial at ICML-2004. Web page at `http://www.cs.bris.ac.uk/~flach/ICML04tutorial/`, 2004.

[FLSM00] Wei Fan, Wenke Lee, Salvatore J. Stolfo, and Matthew Miller. A multiple model cost-sensitive approach for intrusion detection. In *Proceedings of the ECML 2000, 11th European Conference on Machine Learning*, volume 1810 of *Lecture Notes in Computer Science*, pages 142–153, Barcelona, Spain, 2000. Springer-Verlag.

[FPSM92] W. Frawley, G. Piatetsky-Shapiro, and C. Matheus. Knowledge discovery in databases: An overview. *AI Magazine*, ?(?):213–228, 1992.

[FW05] P. A. Flach and S. Wu. Repairing concavities in ROC curves. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI'05)*, pages 702–707, Edinburgh, Scotland, 2005.

[GC00] Christophe Giraud-Carrier. A Note on the Utility of Incremental Learning. *AI Communications*, 13(4):215–223, 2000.

[Get03] Lise Getoor. Link mining: A new data mining challenge. *SIGKDD Explorations*, 5(1):84–89, 2003.

[GHH+01] Robert P. Goldman, Walter Heimerdinger, Steven A. Harp, Christopher W. Geib, Vicraj Thomas, and Robert L. Carter. Information modeling for intrusion report aggregation. In *DISCEX-2001 Conference Proceedings*, pages 46–59, Anaheim, CA, Jun 2001.

[GL00] Dragan Gamberger and Nada Lavrač. Reducing misclassification costs. In *Principles of Data Mining and Knowledge Discovery, 4th European Conference (PKDD 2000)*, volume 1910 of *Lecture Notes in Artificial Intelligence*, pages 34–43, Lyon, France, 2000. Springer Verlag.

[Gra69] Clive W. Granger. Investigating causal relationships by econometric methods and cross-spectral methods. *Econometrica*, 34:424–428, 1969.

[Gra02] Paul Graham. A plan for spam. Web page at `http://www.paulgraham.com/spam.html`, 2002.

[HB99] S. Hettich and S. D. Bay. The UCI KDD Archive. Web page at `http://kdd.ics.uci.edu`, 1999.

[HCC92] Jiawei Han, Yandong Cai, and Nick Cercone. Knowledge discovery in databases: An attribute-oriented approach. In *Proceedings 18th International Conference on Very Large Databases (VLDB)*, pages 547–559, Vancouver, Canada, Aug. 1992.

[HCC93] J. Han, Y. Cai, and N. Cercone. Data-driven discovery of quantitative rules in relational databases. *IEEE Transactions on Knowledge and Data Engineering*, 5(1):29–40, 1993.

[HGC95] D. Heckerman, D. Geiger, and D. Chickering. Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20:197–243, 1995.

[HKM+96] K. Hätönen, M. Klemettinen, H. Mannila, P. Ronkainen, and H. Toivonen. Knowledge discovery from telecommunication network alarm databases. In *Proceedings of the Twelfth International Conference on Data Engineering*, pages 115–122. IEEE Computer Society, 1996.

[HL98] John D. Howard and Thomas A. Longstaff. A common language for computer security incidents. Technical report, CERT, 1998.

[HLMS90] Richard Heady, George Luger, Arthur Maccabe, and Mark Servilla. The architecture of a network level intrusion detection system. Technical report, University of New Mexico, 1990.

[HM82] J.A. Hanley and B.J. McNeil. The meaning and use of the area under a receiver operating characteristic (ROC) curve. *Radiology*, 143(1):29–36, 1982.

[How97] John D. Howard. *An Analysis of Security Incidents on the Internet 1989–1995.* PhD thesis, Carnegie Mellon University, 1997.

[HWHM02] Guy Helmer, Johny S.K. Wong, Vasant Honavar, and Les Miller. Automated discovery of concise predictive rules for intrusion detection. *The Journal of Systems and Software*, 60(2):165–175, 2002.

[IBM02] IBM. IBM Tivoli Risk Manager. Tivoli Risk Manager User's Guide. Version 4.1, 2002.

[JD88] Anil K. Jain and Richard C. Dubes. *Algorithms for Clustering Data.* Prentice-Hall, 1988.

[JD02] Klaus Julisch and Marc Dacier. Mining Intrusion Detection Alarms for Actionable Knowledge. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 366–375, Edmonton, Alberta, Canada, 2002.

[JLM03] Van Jacobson, Craig Leres, and Steven McCanne. TCPDUMP public repository. Web page at `http://www.cpdump.org/`, 2003.

[JSW02] S. Jha, O. Sheyner, and J. Wing. Two formal analyses of attack graphs. In *Proceedings of the 15th IEEE Computer Security Foundations Workshop (CSFW 2002)*, pages 49–63, 2002.

[Jul01] Klaus Julisch. Mining Alarm Clusters to Improve Alarm Handling Efficiency. In *Proceedings 17th Annual Computer Security Applications Conference*, pages 12–21, New Orleans, LA, Dec. 2001.

[Jul03a] Klaus Julisch. Clustering intrusion detection alarms to support root cause analysis. *ACM Transactions on Information and System Security (TISSEC)*, 6(4):443–471, 2003.

[Jul03b] Klaus Julisch. *Using Root Cause Analysis to Handle Intrusion Detection Alarms*. PhD thesis, University of Dortmund, Germany, 2003.

[KCM03] Kenneth A. Kaufman, Guido Cervone, and Ryszard S. Michalski. An applicaion of Symbolic Learning to Intrusion Detection: Preliminary Results from the LUS Methodology. Reports of the Machine Learning and Inference Laboratory MLI 03-2, Machine Learning and Inference Laboratory, George Mason University, 2003.

[Kle99] Mika Klemettinen. *A Knowledge Discovery Methodology for Telecommunication Network Alarm Databases*. PhD thesis, University of Helsinki, 1999.

[KMT99] Mika Klemettinen, Heikki Mannila, and Hannu Toivonen. Rule discovery in telecommunication alarm data. *Journal of Network and Systems Management*, 7(4):395–423, 1999.

[Krs98] Ivan Victor Krsul. *Software Vulnerability Analysis*. PhD thesis, Purdue University, 1998.

[LBMC94] Carl E. Landwehr, Alan R. Bull, John P. McDermott, and William S. Choi. A taxonomy of computer program security flaws. *ACM Computing Surveys (CSUR)*, 26(3):211–254, 1994.

[LC94] David D. Lewis and Jason Catlett. Heterogeneous uncertainty sampling for supervised learning. In *Proceedings of ICML-94, 11th International Conference on Machine Learning*, pages 148–156. Morgan Kaufmann Publishers, San Francisco, CA, 1994.

[LD94] Nada Lavrač and Sašo Džeroski. *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, 1994.

[Lee99] Wenke Lee. *A Data Mining Framework for Constructing Features and Models for Intrusion Detection Systems*. PhD thesis, Columbia University, 1999.

[LFG+00] Richard P. Lippmann, David J. Fried, Isaac Graf, Joshua W. Haines, Kristopher R. Kendall, David McClung, Dan Weber, Seth E. Webster, Dan Wyschogrod, Robert K. Cunningham, and Marc A. Zissman. Evaluating intrusion detection systems: The 1998 DARPA off-line intrusion detection evaluation. In *Proceedings of the 2000 DARPA Information Survivability Conference and Exposition*, volume 1, pages 1012–1035, Hilton Head, SC, 2000.

[LFM+02] Wenke Lee, Wei Fan, Matthew Miller, Salvatore J. Stolfo, and Erez Zadok. Toward cost-sensitive modeling for intrusion detection and response. *Journal of Computer Security*, 10(1–2):5–22, 2002.

[LHF+00] Richard Lippmann, Joshua W. Haines, David J. Fried, Jonathan Korba, and Kumar Das. The 1999 DARPA off-line intrusion detection evaluation. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 34(4):579–595, 2000.

[LKD05] Niels Landwehr, Kristian Kersting, and Luc De Raedt. nFOIL: Integrating naive Bayes and FOIL. In M. Veloso and S. Kambhampat, editors, *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05)*, pages 795–800, Pittsburgh, PA, 2005.

[LLO+03] J. Levine, R. LaBella, H. Owen, D. Contis, and B. Culver. The use of honeynets to detect exploited systems across large enterprise networks. In *Proceedings of the 4th IEEE Information Assurance Workshop*, West Point, NY, 2003.

[LS98] Wenke Lee and Salvatore Stolfo. Data mining approaches for intrusion detection. In *Proceedings of the 7th USENIX Security Symposium*, San Antonio, TX, 1998.

[LWS02] Richard Lippmann, Seth Webster, and Douglas Stetson. The effect of identifying vulnerabilities and patching software on the utility of network intrusion detection. In *Recent Advances in Intrusion Detection (RAID2002)*, volume 2516 of *Lecture Notes in Computer Science*, pages 307–326. Springer-Verlag, 2002.

[MC03] Matthew V. Mahoney and Philip K. Chan. An analysis of the 1999 DARPA/Lincoln Laboratory evaluation data for network anomaly detection. In *Recent Advances in Intrusion Detection (RAID2003)*, volume 2820 of *Lecture Notes in Computer Science*, pages 220–237. Springer-Verlag, 2003.

[McH00] John McHugh. The 1998 Lincoln Laboratory IDS evaluation. A critique. In *Recent Advances in Intrusion Detection (RAID2000)*, volume 1907 of *Lecture Notes in Computer Science*, pages 145–161. Springer-Verlag, 2000.

[McH01] John McHugh. Testing intrusion detection systems: A critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln Laboratory. *ACM Transactions on Information and System Security*, 3:262–294, 2001.

[MCZH00] Stefanos Manganaris, Marvin Christensen, Dan Zerkle, and Keith Hermiz. A data mining analysis of RTID alarms. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 34(4):571–577, Oct 2000.

[MDK+97] A. Martin, G. Doddington, T. Kamm, M. Ordowski, and M. Przybocki. The DET curve in assessment of detection task performance. In *Proceedings of the European Conference on Speech Technology*, pages 1895–1898, Rhodes, Grece, 1997.

[MH02] José María and Gómez Hidalgo. Evaluating cost-sensitive unsolicited bulk email categorization. In *Proceedings of the 2002 ACM Symposium on Applied Computing*, pages 615–620. Springer-Verlag, 2002.

[MHL94] Biswanath Mukherjee, Todd L. Heberlein, and Karl N. Levitt. Network intrusion detection. *IEEE Network*, 8(3):26–41, 1994.

[Mic69] R.S. Michalski. On the quasi-minimal solution of the general covering problem. In *Proceedings of the V International Symposium on Information Processing (FCIP 69)(Switching Circuits)*, volume A3, pages 125–128, Bled, Yugoslavia, 1969.

[Mit97] Tom M. Mitchel. *Machine Learning*. Mc Graw Hill, 1997.

[MIT99] MIT Lincoln Laboratory. 1999 DARPA intrusion detection evaluation data set. Web page at `http://www.ll.mit.edu/IST/ideval/data/1999/1999_data_index.html`, 1999.

[MIT04] MITRE. Common Vulnerabilites and Exposures. Web page at `http://cve.mitre.org`, 1999–2004.

[MM95] M.A. Maloof and R.S. Michalski. A partial memory incremental learning methodology and its application to computer intrusion detection. Reports of the Machine Learning and Inference Laboratory MLI 95-2, Machine Learning and Inference Laboratory, George Mason University, 1995.

[MM02] Marcus A. Maloof and Ryszard S. Michalski. Incremental learning with partial instance memory. In *Proceedings of Foundations of Intelligent Systems: 13th International Symposium, ISMIS 2002*, volume 2366 of *Lecture Notes in Artificial Intelligence*, pages 16–27. Springer-Verlag, 2002.

[MMDD02] Benjamin Morin, Ludovic Mé, Hervé Debar, and Mireille Ducasse. M2D2: A formal data model for IDS alert correlation. In *Recent Advances in Intrusion Detection (RAID2002)*, volume 2516 of *Lecture Notes in Computer Science*, pages 115–137. Springer-Verlag, 2002.

[Moz06] Mozilla Corporation. mozilla Thunderbird. Web page at `http://www.mozilla.com/thunderbird/`, 2005–2006.

[MT96] Heikki Mannila and Hannu Toivonen. Discovering generalized episodes using minimal occurrences. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*, pages 146–151. AAAI/MIT Press, 1996.

[MTV97] Heikki Mannila, Hannu Toivonen, and A. Inkeri Verkamo. Discovery of frequent episodes in event sequences. *Data Mininig Knowledge Discovery*, 1(3):259–289, 1997.

[Mug95] Stephen Muggleton. Inverse entailment and Progol. *New Gen. Comput.*, 13:245–286, 1995.

[NC02] Peng Ning and Yun Cui. An intrusion alert correlator based on prerequisities of intrusions (TR-2002-01). Technical report, North Carolina State University, Raleigh, NC, 2002.

[NCR02a] Peng Ning, Yun Cui, and Douglas S. Reeves. Analyzing intrusion alerts via correlation. In *Recent Advances in Intrusion Detection (RAID2002)*, volume 2516 of *Lecture Notes in Computer Science*. Springer-Verlag, 2002.

[NCR02b] Peng Ning, Yun Cui, and Douglas S. Reeves. Constructing attack scenarios through correlation of intrusion alerts. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 245–254, 2002.

[NE98] Richard Nock and Babak Esfandiari. Oracles and assistants: Machine learning applied to network supervision. In *Proceedings of the 12th in Canadian Conference*

*on Artificial Intelligence*, volume 1418 of *Lecture Notes in Computer Science*, pages 86–98. Springer-Verlag, 1998.

[NIS04] NIST. ICAT Metabase. Web page at `http://icat.nist.gov/`, 2000–2004.

[NRC01] Peng Ning, Douglas S. Reeves, and Yun Cui. Correlating alerts using prerequisites of intrusions (TR-2001-13). Technical report, North Carolina State University, Raleigh, NC, 2001.

[Pax99] Vern Paxson. Bro: A system for detecting network intruders in real-time. *Computer Networks*, 31(23-24):2435–2463, 1999.

[PB88] Mark Paradies and David Busch. Root cause analysis at Savannah river plant. In *Proceedings of the IEEE Converence on Human Factors and Power Plants*, 1988.

[PDP05] Fabien Pouget, Marc Dacier, and Van Hau Pham. Leurre.Com: on the advantages of deploying a large scale distributed honeypot platform. In *ECCE'05, E-Crime and Computer Conference*, Monaco, 2005.

[Pea88] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems. Networks of Plausible Inference.* Morgan Kaufmann Publishers, 1988.

[Pea00] Joseph P. Pickett and et al., editors. *The American Heritage Dictionary of the English Language*. Boston: Houghton Mifflin Company, 2000.

[PEG97] Terry R. Payne, Peter Edwards, and Claire L. Green. Experience with Rule Induction and k-Nearest Neighbor Methods for Interface Agents that Learn. *IEEE Transactions on Knowledge and Data Engineering*, 9(2):329–335, 1997.

[PF98] Foster Provost and Tom Fawcett. Robust classification systems for imprecise environemnts. In *Proceedings of the Fifteenth National Conference on Artificial Intellignence (AAAI-98)*, pages 706–713. AAAI Press, 1998.

[PF01] Foster Provost and Tom Fawcett. Robust classification for impresice environments. *Machine Learning Journal*, 42(3):203–231, 2001.

[PFV02] Philip A. Porras, Martin W. Fong, and Alfonso Valdes. A mission-impact-based approach to INFOSEC alarm correlation. In *Recent Advances in Intrusion Detection (RAID2002)*, volume 2516 of *Lecture Notes in Computer Science*. Springer-Verlag, 2002.

[PHP04a] The PHP Group. PHP hypertext preprocessor. Web page at `http://www.php.net`, 2001–2004.

[php04b] The phpBB Group. phpBB.com. Web page at `http://www.phpbb.com`, 2001–2004.

[Pie04] Tadeusz Pietraszek. Using adaptive alert classification to reduce false positives in intrusion detection. In *Recent Advances in Intrusion Detection (RAID2004)*, volume 3324 of *Lecture Notes in Computer Science*, pages 102–124, Sophia Antipolis, France, 2004. Springer-Verlag.

[Pie05] Tadeusz Pietraszek. Optimizing abstaining classifiers using ROC analysis. In *Machine Learning, Proceedings of the Twenty-second International Conference (ICML 2005)*, pages 665–672, Bonn, Germany, 2005.

[Pie06] Tadeusz Pietraszek. On the optimization of abstaining classifiers using ROC analysis. *Machine Learning Journal*, (to appear), 2006.

[Pie07] Tadeusz Pietraszek. Classification of intrusion detection alerts using abstaining classifiers. *Intelligent Data Analysis Journal*, 11(3):(to appear), 2007.

[PMAS94] Michael J. Pazzani, Partick Murphy, Kamal Ali, and David Schulenburg. Trading off coverage for accuracy in forecasts: Applications to clinical data analysis. In *Proceedings of AAAI Symposium on AI in Medicine*, pages 106–110, Stanford, CA, 1994.

[PN98] Thomas H. Ptacek and Timothy N. Newsham. Insertion, evasion and denial of service: Eluding network intrusion detection. Technical report, Secure Networks Inc., 1998.

[Pro04] N. Provos. A virtual honeypot framework. In *Proceedings of the 11th USENIX Security Symposium*, pages 92–99, San Francisco, CA, 2004.

[PT05] Tadeusz Pietraszek and Axel Tanner. Data mining and machine learning—Towards reducing false positives in intrusion detection. *Information Security Technical Report*, 10:169–183, 2005.

[PV05] Tadeusz Pietraszek and Chris Vanden Berghe. Defending against injection attacks through context-sensitive string evaluation. In *Recent Advances in Intrusion Detection (RAID2005)*, volume 3858 of *Lecture Notes in Computer Science*, pages 124–145, Seattle, WA, 2005. Springer-Verlag.

[QCJ93] J. R. Quinlan and R. M. Cameron-Jones. Foil: A midterm report. In *Machine Learning: ECML-93, European Conference on Machine Learning, Proceedings*, volume 667, pages 3–20, 1993.

[QL03] Xinzhou Qin and Wenke Lee. Statistical causality analysis of INFOSEC alert data. In *Recent Advances in Intrusion Detection (RAID2003)*, volume 2820 of *Lecture Notes in Computer Science*, pages 73–93, Pittsburgh, PA, 2003. Springer-Verlag.

[QL04] Xinzhou Qin and Wenke Lee. Discovering novel attack strategies from INFOSEC alerts. In *Computer Security - ESORICS 2004, 9th European Symposium on Research Computer Security*, volume 3193 of *Lecture Notes in Computer Science*, pages 439–456, Sophia Antipolis, France, 2004. Springer-Verlag.

[Qui86] Ross Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[Qui93] Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufman, 1993.

[R D04] R Development Core Team. *R: A language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria, 2004. ISBN 3-900051-00-3.

[RA00] Ronald W. Ritchey and Paul Ammann. Using model checking to analyze network vulnerabilities. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P 2000)*, pages 156–165, 2000.

[Roe05] Martin Roesch. SNORT. The Open Source Network Intrusion System. Web page at `http://www.snort.org`, 1998–2005.

[RZD05] James Riordan, Diego Zamboni, and Yann Duponchel. Billy Goat, an accurate worm-detection system (revised version) (RZ 3609). Technical report, IBM Zurich Research Laboratory, 2005.

[SDHH98] M. Sahami, S. Dumais, D. Heckerman, and E. Horvitz. A Bayesian approach to filtering junk e-mail. In *AAAI Workshop on Learning for Text Categorization*, pages 55–62, Madison, WI, 1998.

[Sec04] SecurityFocus. BugTraq. Web page at `http://www.securityfocus.com/bid`, 1998–2004.

[Sen05] Ted E. Senator. Multi-stage classification. In *Proceedings of the 5th IEEE International Conference on Data Mining (ICDM 2005)*, pages 386–393, Houston, TX, 2005. IEEE Computer Society.

[SGVS99] R. Sekar, Y. Guang, S. Verma, and T. Shanbhag. A high-performance network intrusion detection system. In *ACM Conference on Computer and Communications Security*, pages 8–17, Kent Ridge Digital Labs, Singapore, 1999.

[SHJ+02] Oleg Sheyner, Joshua Haines, Somesh Jha, Richard Lippmann, and Jeannette M. Wing. Automated generation and analysis of attack graphs. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P 2002)*, pages 254–265, 2002.

[SP01] Umesh Shankar and Vern Paxson. Active mapping: Resisting NIDS evasion without altering traffic. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, pages 44–62, Oakland, CA, 2001.

[SP03] Robin Sommer and Vern Paxson. Enhancing byte-level network intrusion detection signatures with context. In *Proceedings of the 10th ACM Conference on Computer and Communication Security*, pages 262–271, Washington, DC, 2003.

[spa06a] The Apache SpamAssassin Project. Web page at `http://spamassassin.apache.org/`, 2002–2006.

[spa06b] Spam statistics. Web page at `http://spamlinks.net/stats.htm/`, 2006.

[SS04] Maheshkumar Sabhnani and Gursel Serpen. Why machine learning algorithms fail in misuse detection on KDD intrusion detection data set. *Intelligent Data Analysis*, 8(4):403–415, 2004.

[Ste92] James Stewart. *Calculus.* Brooks Cole, 1992.

[Sun95] SunSoft. *SunSHIELD Basic Security Module.* SunSoft, 1995.

[Szy00] Wisława Szymborska. *Poems New and Collected.* Harvest Books, 2000.

[Tin98] K.M. Ting. Inducing cost-sensitive trees via instance weighting. In *Proceedings of The Second European Symposium on Principles of Data Mining and Knowledge Discovery*, volume 1510 of *Lecture Notes in AI*, pages 139–147. Springer-Verlag, 1998.

[Tor00] Francesco Tortorella. An optimal reject rule for binary classifiers. In *Advances in Pattern Recognition, Joint IAPR International Workshops SSPR 2000 and SPR 2000*, volume 1876 of *Lecture Notes in Computer Science*, pages 611–620, Alicante, Spain, 2000. Springer-Verlag.

[Vap95] Vladimir N. Vapnik. *The Nature of Statistical Learning Theory*. Springer, 1995.

[Vig03] G. Vigna. Teaching hands-on network security: Testbeds and live exercises. *Journal of Information Warfare*, 3(2):8–25, 2003.

[vL02] Wim van Laer. *From Propositional to First Order Logic in Machine Learning and Data Mining. Induction of First Order Rules with ICL*. PhD thesis, Katholieke Universiteit Leuven, 2002.

[VMV05] F. Valeur, D. Mutz, and G. Vigna. A learning-based approach to the detection of SQL attacks. In *Proceedings of the Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*, Vienna, Austria, 2005.

[VS00] Alfonso Valdes and Keith Skinner. An approach to sensor correlation. In *Recent Advances in Intrusion Detection (RAID2000)*, volume 1907 of *Lecture Notes in Computer Science*. Springer-Verlag, 2000.

[VS01] Alfonso Valdes and Keith Skinner. Probabilistic alert correlation. In *Recent Advances in Intrusion Detection (RAID2001)*, volume 2212 of *Lecture Notes in Computer Science*, pages 54–68. Springer-Verlag, 2001.

[VVCK04] F. Valeur, G. Vigna, C.Kruegel, and R. Kemmerer. A comprehensive approach to intrusion detection alert correlation. *IEEE Transactions on Dependable and Secure Computing*, 1(3):146–169, 2004.

[WF00] Ian H. Witten and Eibe Frank. *Data Mining: Practical machine learning tools with Java implementations*. Morgan Kaufmann Publishers, San Francisco, CA, 2000.

[WL01] Jia Wang and Insup Lee. Measuring false-positive by automated real-time correlated hacking behavior analysis. In *Information Security 4th International Conference*, volume 2200 of *Lecture Notes in Computer Science*, pages 512–535. Springer-Verlag, 2001.

[Wol06] Wolfram Research Inc. Lagrange Multiplier—from Wolfram MathWorld. Web page at `http://mathworld.wolfram.com/LagrangeMultiplier.html`, 1999–2006.

[Zam01] Diego Zamboni. *Using Internal Sensors for Computer Intrusion Detection*. PhD thesis, Purdue University, 2001.

[ŽDS05] Bernard Ženko, Sašo Džeroski, and Jan Struyf. Learning predictive clustering rules. In Francesco Bonchi and Jean-Francois Boulicaut, editors, *Proceedings of*

*Knowledge Discovery in Inductive Databases: 4th International Workshop (KDID 2005)*, volume 3933 of *Lecture Notes in Computer Science*, pages 234–250, Porto, Portugal, 2005.

[ZE01] Bianca Zadrozny and Charles Elkan. Obtaining calibrated probability estimates from decision trees and naive Bayesian classifiers. In *Proceedings of the Eighteenth International Conference on Machine Learning (ICML-2001)*, pages 609–616, Williams College, Williamstown, MA, 2001. Morgan Kaufmann Publishers.

# Table of Symbols

**Machine Learning**

| | |
|---|---|
| $I$ | instance space $I = \{i_1, i_2, \ldots, i_n\}$, in our application pertains to alerts $\{A_i\}$, |
| $\mathcal{C}$ | classifier mapping an instance space $I$ to a class space $C$, |
| $\mathcal{L}$ | machine-learning method producing a classifier $\mathcal{C}$, |
| $\mathcal{R}$ | scoring classifier, |
| $\mathcal{C}_\tau$ | classifier constructed from a scoring classifier $\mathcal{R}$ using a threshold $\tau$, |
| "+", "−" | classes assigned by a binary classifier, |
| "+", "−", "?" | classes assigned by an abstaining binary classifier, |
| $C$ | confusion matrix, describing the performance of a classifier, |
| $TP$, $TN$, $FP$, $FN$ | true positives, true negatives, false positives and false negatives calculated from a $2 \times 2$ confusion matrix, |
| $tp$, $tn$, $fp$, $fn$ | true-positive rate, true-negative rate, false-positive rate and false-negative rate calculated from a $2 \times 2$ confusion matrix, |
| $N$, $P$ | number of negative and positive instances for binary classification, |
| $Co$ | cost matrix, modeling class-dependent misclassification costs, |
| $CR$, $ICR$ | cost ratios: the quotient of the two misclassification costs in $2 \times 2$ cost matrix $Co$, |
| $w$ | weight used with Weighting to construct a cost-sensitive classifier, |
| $f_{ROC}$, $f'_{ROC}$, $f''_{ROC}$ | ROC curve and its derivatives mapping false-positive rate ($fp$) to true-positive rate ($tp$), |
| $f_{PR}$ | precision-recall function mapping precision ($p$) to recall ($r$), |
| $f_{DET}$ | DET function, a non-linear transform of both variables of ROC function, |
| $rc$ | misclassification cost per classified example. |

**Intrusion Detection**

| | |
|---|---|
| $L$ | alert log, containing a sequence of alerts $\{A_1, A_2, \ldots, A_i, \ldots, A_n\}$. For some applications (e.g., clustering, learning a classifier) can be considered a set, |
| $A_i$ | an alert, represented as a tuple of attributes $(T_1, T_2, \ldots, T_i, \ldots, T_n)$, |
| $T_i$ | alert attribute, |
| $\mathcal{U}$ | utility function describing the utility of the system, with the follwowing three components: misclassified alerts, analyst's workload and abstentions, |
| $\mathcal{O}$ | human analyst performing the manual classification of alerts, |
| $s$ | false-positive sampling rate for ALAC and ALAC+. |

**Abstaining Classifiers, ALAC+**

| | |
|---|---|
| $\mathcal{A}$ | abstaining classifier, |
| $\mathcal{A}_{\alpha,\beta}$ | abstaining classifier constructed from two classifiers $\mathcal{C}_\alpha$ and $\mathcal{C}_\beta$, |
| $\mathcal{E}$ | evaluation model for abstaining classifiers, one of: $\mathcal{E}_{CB}$, $\mathcal{E}_{BA}$ and $\mathcal{E}_{BI}$, |
| $fp_\alpha$, $fp_\beta$ | false-positive rates of classifiers $\mathcal{C}_\alpha$ and $\mathcal{C}_\beta$ defining the abstaining classifier $\mathcal{A}_{\alpha,\beta}$, |
| $rc_{CB}$ | misclassification cost in a cost-based model, |
| $rc_B$, $rc_{BA}$, $rc_{BI}$ | misclassification cost in bounded model (bounded-improvement and bounded-abstention), calculated per actually classified instance |
| $c_{23}$, $c_{13}$ | additional components of a cost matrix in a cost-based model, |
| $k$ | abstention window, |
| $f$ | fraction improvement over the $rc$ of the optimal binary classifier, |
| $k_{\max}$ | maximum abstention window in a bounded-abstention model, |
| $rc_{\max}$ | maximum misclassification cost in a bounded-improvement model, |
| $f_{\min}$ | minimum improvement over the optimal binary classifier in a bounded-improvement model, |
| $w_b$, $w_{\mathcal{C}_\alpha}$, $w_{\mathcal{C}_\beta}$ | weights corresponding to the optimal binary classifier, classifiers $\mathcal{C}_\alpha$ and $\mathcal{C}_\beta$ (ALAC+). |

**Clustering, Two-stage Alert-Classification System**

| | |
|---|---|
| $\mathcal{G}_i$ | tree-based generalization hierarchy for attribute $T_i$, |
| $P_i$, $F_i$ | patterns $\{P_i\}$ discovered by descriptive modeling (e.g., in the case of CLARAty they would be generalized alerts), filters $\{F_i\}$ are derived from patterns $\{P_i\}$ so that they can be applied to future alerts, |
| $\mathcal{R}_{\mathcal{C},i}$ | clustering run, |
| $coverage_c(X)$ | cluster coverage in the clustering stage, |
| $coverage_f(X)$ | cluster coverage in the filtering stage, |
| $P_{P_j}$ | cluster persistency, |
| $P_{\mathcal{R}_{\mathcal{C},i}}$ | average cluster persistency for the entire clustering run $\mathcal{R}_{\mathcal{C},i}$, |
| $CP$, $CR$ | clustering precision and recall. |

# Index