

Machine Learning in Intrusion Detection

Abstract

Detection of anomalies in data is one of the fundamental machine learning tasks. Anomaly detection provides the core technology for a broad spectrum of security-centric applications. In this dissertation, we examine various aspects of anomaly based intrusion detection in computer security.

First, we present a new approach to learn program behavior for intrusion detection. Text categorization techniques are adopted to convert each process to a vector and calculate the similarity between two program activities. Then the k -Nearest Neighbor classifier is employed to classify program behavior as normal or intrusive. We demonstrate that our approach is able to effectively detect intrusive program behavior while a low false positive rate is achieved.

Second, we describe an adaptive anomaly detection framework that is designed to handle concept drift and online learning for dynamic, changing environments. Through the use of unsupervised evolving connectionist systems, normal behavior changes are efficiently accommodated while anomalous activities can still be recognized. We demonstrate the performance of our adaptive anomaly detection systems and show that the false positive rate can be significantly reduced.

Third, we study methods to efficiently estimate the generalization performance of an anomaly detector and the training size requirements. An error bound for support vector machine based anomaly detection is introduced. Inverse power-law learning curves, in turn, are used to estimate how the accuracy of the anomaly detector improves when trained with additional samples.

Finally, we present a game theoretic methodology for cost-benefit analysis and design of IDS. We use a simple two-person, nonzero-sum game to model the strategic interdependence between an IDS and an attacker. The solutions based on the game theoretic analysis integrate the cost-effectiveness and technical performance tradeoff of the IDS and identify the best defense and attack strategies.

Professor V. Rao Vemuri
Dissertation Committee Chair

Machine Learning in Intrusion Detection

By

YIHUA LIAO

B.A. (Wuhan University) 1995

M.S. (Institute of High Energy Physics, Chinese Academy of Sciences) 1998

DISSERTATION

Submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

Computer Science

in the

OFFICE OF GRADUATE STUDIES

of the

UNIVERSITY OF CALIFORNIA

DAVIS

Approved:

Committee in charge

2005

Machine Learning in Intrusion Detection

Copyright 2005
by
Yihua Liao

To Natalie

Acknowledgments

First I would like to thank my research adviser, Professor Rao Vemuri, who has been a constant source of guidance, encouragement and inspirations for the past five years. Among the many things I learned from him, his eloquent writing style, his persistency and integrity, and his passion for public service have made a deep impression on me.

I would also like to thank Professor Karl Levitt and Professor S. Felix Wu, my dissertation committee members, for providing insightful guidance and valuable comments throughout my graduate studies. I am also grateful to Professor Matt Bishop, Professor David Rocke and Professor David Woodruff, who served on my qualifying exam committee and helped shape my doctoral research.

All the members of the Computational Intelligence Lab made me feel at home. The interactions with them substantially contributed to the development of this work. I would especially like to thank Na Tang, Alejandro Pasos and Wenjie Hu for their collaborations and many useful discussions.

I am also grateful to the members of the Computer Security Lab. Through seminars and discussions, I have gained exposure to various topics in computer security and received helpful feedbacks on my research. In particular, I would like to thank Todd Heberlein, Steve templeton, Melissa Danforth, Ke Zhang, Tao Song, Tye Stallard and Patty Graves.

It has been a great pleasure to study at the Department of Computer Science of UC Davis. I sincerely thank everyone at the department for providing an intellectually stimulating environment and for their help and friendship.

I gratefully acknowledge Dr. Tom Goldring of NSA for providing the Windows NT user profiling data and Professor Philip Chan of Florida Institute of Technology for many inspiring discussions.

Financial support for this work has been provided by AFOSR grant F49620-01-1-0327 to the Center for Digital Security and the Institute of Data Analysis and Visualization (IDAV) of UC Davis, and by a Summer Research Fellowship awarded by the Graduate Studies of UC Davis.

Last, but not least, I am deeply indebted to my family members in China, who provided their unconditional love and support throughout the years and gave me strength to complete my graduate studies. This dissertation is dedicated in part to them.

Contents

| | |
|---|-------------|
| List of Figures | viii |
| List of Tables | x |
| Abstract | 1 |
| 1 Introduction | 3 |
| 1.1 General Approaches to Intrusion Detection | 4 |
| 1.2 Machine Learning Based Anomaly Detection | 6 |
| 1.3 Issues in Anomaly Detection | 7 |
| 1.3.1 Feature Selection | 7 |
| 1.3.2 Skewed Class Distribution | 8 |
| 1.3.3 Distance Metrics and Window Size | 8 |
| 1.3.4 Supervised Learning vs. Unsupervised Learning | 9 |
| 1.3.5 Concept Drift | 9 |
| 1.3.6 Generalization Performance and Training Size Requirements | 10 |
| 1.3.7 Cost-effectiveness of IDS | 10 |
| 1.4 Overview of the Dissertation | 10 |
| 2 Machine Learning Approaches to Anomaly Detection | 12 |
| 2.1 Machine Learning and Its Problem Formulations | 12 |
| 2.2 Learning Methods for Anomaly Detection | 13 |
| 2.3 Audit Data | 16 |
| 2.3.1 DARPA/KDD Datasets | 17 |
| 2.3.2 UNM System Call Data | 18 |
| 2.3.3 UNIX Command Data | 18 |
| 3 Learning Program Behavior with K-Nearest Neighbor Classifier | 21 |
| 3.1 Introduction | 21 |
| 3.2 Related Work | 22 |
| 3.3 K-Nearest Neighbor Text Categorization Method | 23 |
| 3.4 Experiments | 26 |
| 3.4.1 Dataset | 26 |
| 3.4.2 Anomaly Detection | 27 |
| 3.4.3 Anomaly Detection Combined with Signature Verification | 30 |
| 3.5 Discussion | 32 |

| | | |
|----------|---|-----------|
| 3.6 | Summary | 34 |
| 4 | Adaptive Anomaly Detection with Evolving Connectionist Systems | 35 |
| 4.1 | Introduction | 35 |
| 4.2 | Related Work | 36 |
| 4.3 | Adaptive Anomaly Detection Framework | 38 |
| 4.3.1 | Fuzzy ART | 42 |
| 4.3.2 | EFuNN | 44 |
| 4.4 | Experiments | 45 |
| 4.4.1 | Static Learning via Support Vector Machines | 46 |
| 4.4.2 | Cost Function | 46 |
| 4.4.3 | Network Intrusion Detection | 47 |
| 4.4.4 | Masquerade Detection with User Profiling Data | 52 |
| 4.5 | Discussion | 56 |
| 4.6 | Summary | 57 |
| 5 | Estimating Generalization Performance and Training Size Requirements | 58 |
| 5.1 | Introduction | 58 |
| 5.2 | Error estimate of SVM-based anomaly detection | 60 |
| 5.3 | learning curve fitting | 63 |
| 5.4 | Experiments | 64 |
| 5.4.1 | Artificial data | 64 |
| 5.4.2 | Masquerade data | 65 |
| 5.4.3 | KDD data | 66 |
| 5.4.4 | Learning curve fitting | 69 |
| 5.5 | Summary | 70 |
| 6 | Intrusion Detection and Response: A Game Theoretic Perspective | 72 |
| 6.1 | Introduction | 72 |
| 6.2 | Related Work | 74 |
| 6.3 | Review of Game Theory | 75 |
| 6.4 | Game Theoretic Modeling | 78 |
| 6.4.1 | Parameters and Cost/Payoff Factors | 78 |
| 6.4.2 | IDS vs. Attacker | 80 |
| 6.5 | Discussion | 85 |
| 6.6 | Summary | 86 |
| 7 | Conclusions and Future Directions | 87 |
| 7.1 | Conclusions | 87 |
| 7.2 | Future Directions | 88 |
| 7.2.1 | Theoretic Analysis | 88 |
| 7.2.2 | Learning for Understanding and Planning | 89 |
| 7.2.3 | Ensemble Learning | 89 |
| A | Support Vector Machine Based Anomaly Detection | 90 |
| | Bibliography | 94 |

List of Figures

| | | |
|-----|---|----|
| 1.1 | Three general approaches to intrusion detection. | 5 |
| 3.1 | Pseudo code for the k NN classifier algorithm for anomaly detection. | 28 |
| 3.2 | Performance of the k NN classifier method expressed in ROC curves for the $tf \cdot idf$ weighting method. False positive rate vs attack detection rate for $k=5, 10$ and 25 | 30 |
| 3.3 | ROC curves for $tf \cdot idf$ weighting ($k=10$) and frequency weighting ($k=15$). | 31 |
| 4.1 | (a) A simplified diagram of an evolving connectionist system for unsupervised learning. The system has n input nodes and m pattern nodes. There is a connection from each input node to every pattern node. Some connections are not shown in the figure. (b) An evolving connectionist system that has an additional fuzzy input layer. The task of the fuzzy input nodes is to transfer the input values into membership degrees. | 39 |
| 4.2 | Pseudo code for adaptive anomaly detection. | 41 |
| 4.3 | Flow graph representation of the Fuzzy ART algorithm. | 44 |
| 5.1 | A two-dimensional sphere (solid line) containing most of the data. Enclosed in a circle are four support vectors on the boundary. | 61 |
| 5.2 | Experimental results for various ν values on the artificial data. (a) Comparison of error estimate, 10-fold cross validation error and false positive rate on the testing set. (b) True positive rate vs. ν | 65 |
| 5.3 | Comparison of error estimate, leave-one-out error and false positive rate on testing data. (a) User 9; (b) User 24; (c) User 42; (d) User 43. | 67 |
| 5.4 | Experimental results for various γ values on the KDD data ($\nu = 0.001$). (a) Comparison of error estimate and 5-fold cross validation error on the training set. (b) False positive rate and false negative rate on the testing set. | 69 |
| 5.5 | Learning curves for masquerade data. (a) User 5; (b) User 6. | 70 |
| 6.1 | Extensive form for Matching Pennies. | 76 |
| 6.2 | Normal form of Matching Pennies. | 77 |
| 6.3 | Game example. | 77 |
| 6.4 | Extensive form for Game A. | 81 |
| 6.5 | Normal form of Game A. | 81 |
| 6.6 | Attacker's dominant strategy is "attack" when $\lambda > \delta/(1 - \delta)$ | 83 |

A.1 A two-dimensional sphere (solid line) containing most of the data. Enclosed in a circle are four support vectors on the boundary. 92

List of Tables

| | | |
|-----|--|----|
| 2.1 | Summary of major machine learning techniques for intrusion detection. . . | 15 |
| 3.1 | Analogy between text categorization and intrusion detection when applying the k NN classifier. | 25 |
| 3.2 | List of 50 distinct system calls that appear in the training dataset. | 27 |
| 3.3 | Attack detection rate for DARPA testing data when anomaly detection is combined with signature verification. | 32 |
| 4.1 | Numbers of normal and attack examples in <i>Exp. 1</i> and <i>Exp. 2</i> | 48 |
| 4.2 | The performance (false positive rate, hit rate and cost) of Fuzzy ART and EFuNN with the Exp. 1 data stream. Results illustrate the impact of varying ρ on their performance. | 49 |
| 4.3 | The performance of Fuzzy ART and EFuNN with the Exp. 1 data stream. Results illustrate the impact of varying β on their performance. | 50 |
| 4.4 | The performance of Fuzzy ART and EFuNN with the Exp. 1 data stream. Results illustrate the impact of varying N_{watch} and Min_{count} on their performance. | 51 |
| 4.5 | The performance of SVM, Fuzzy ART and EFuNN in <i>Exp. 2</i> | 52 |
| 4.6 | Number of login sessions and the masquerade examples. | 54 |
| 4.7 | The performance of Fuzzy ART and EFuNN in <i>Exp. 3</i> . $N_{watch} = 3$, $Min_{count} = 2$ | 54 |
| 4.8 | The performance of SVM, Fuzzy ART and EFuNN in <i>Exp. 4</i> | 55 |
| 5.1 | Experimental results for various γ values on the KDD data ($\nu = 0.001$). . . | 68 |
| 5.2 | Experimental results for various ν values on the KDD data ($\gamma = 0.01$). . . | 68 |
| 5.3 | Errors for various training sizes on the masquerade data. | 69 |
| 6.1 | List of parameters. | 79 |
| 6.2 | Numerical examples. | 85 |

Abstract

Detection of anomalies in data is one of the fundamental machine learning tasks. Anomaly detection provides the core technology for a broad spectrum of security-centric applications. In this dissertation, we examine various aspects of anomaly based intrusion detection in computer security.

First, we present a new approach to learn program behavior for intrusion detection. Text categorization techniques are adopted to convert each process to a vector and calculate the similarity between two program activities. Then the k -Nearest Neighbor classifier is employed to classify program behavior as normal or intrusive. We demonstrate that our approach is able to effectively detect intrusive program behavior while a low false positive rate is achieved.

Second, we describe an adaptive anomaly detection framework that is designed to handle concept drift and online learning for dynamic, changing environments. Through the use of unsupervised evolving connectionist systems, normal behavior changes are efficiently accommodated while anomalous activities can still be recognized. We demonstrate the performance of our adaptive anomaly detection systems and show that the false positive rate can be significantly reduced.

Third, we study methods to efficiently estimate the generalization performance of an anomaly detector and the training size requirements. An error bound for support vector machine based anomaly detection is introduced. Inverse power-law learning curves, in turn, are used to estimate how the accuracy of the anomaly detector improves when trained with additional samples.

Finally, we present a game theoretic methodology for cost-benefit analysis and design of IDS. We use a simple two-person, nonzero-sum game to model the strategic interdependence between an IDS and an attacker. The solutions based on the game theoretic analysis integrate the cost-effectiveness and technical performance tradeoff of the IDS and identify

the best defense and attack strategies.

Professor V. Rao Vemuri
Dissertation Committee Chair

Chapter 1

Introduction

Computer security vulnerabilities will always exist as long as we have flawed security policies, poorly configured computer systems, or imperfect software programs. Given the rapid increase in connectivity and accessibility of computer systems in today's society, computer intrusions and security breaches are posing serious threats to national security as well as enterprise and home user interests.

Intrusion detection is an important component of computer security mechanisms. It aims to detect computer break-ins, penetrations, and other forms of computer abuse that exploit security vulnerabilities or flaws in computer systems. Examples of these include hackers using exploit scripts to gain access to or deny the services of a computer system and insiders who misuse their privileges. An intrusion detection system (IDS) continuously monitors the standard operations of the target system and looks for known attack signatures and/or deviations in usage. Traditional security mechanisms, such as authentication, access control and information flow control, help protect systems, data, and resources. However, nothing is perfect. Even the best protected systems must be monitored to detect successful and unsuccessful attempts to breach security. This is why intrusion detection systems have become an increasingly indispensable player in the arsenal against computer misuse.

1.1 General Approaches to Intrusion Detection

IDS began in the 1980s as a promising paradigm for detecting and preventing intrusions and attacks [18]. The goals of an IDS can be summarized as follows:

- Detect a wide variety of intrusions, including intrusions from outside as well as insider attacks, both known and previously unknown attacks.
- Detect intrusions in a timely fashion.
- Present the analysis in a simple, easy-to-understand format.
- Be accurate, that is, achieve a low false alarm rate. A false positive occurs when an IDS raises an alarm when no attack is underway and a false negative occurs when an IDS fails to report an ongoing attack.

For the last two decades, three general approaches to intrusion detection have been developed, namely, anomaly detection, misuse detection and specification-based detection [8].

An anomaly detector analyzes a set of characteristics of the monitored system (or users) and identifies activities that deviate from the normal behavior, based on the assumption that such deviations may indicate that an intrusion or attack exploiting vulnerabilities has occurred (or may still be occurring). Any observable behavior of the system can be used to build a model of the normal operation of the system. Audit logs, network traffic, user commands, system calls are all common choices. Machine learning and statistical methods are usually employed to capture the system or user's normal usage pattern and classify new behavior as either normal or abnormal. Anomaly detection has the potential of detecting novel attacks, as well as variations of known attacks. However, anomaly detection suffers from the basic difficulty of defining what is "normal". When the system or user behavior varies widely, methods based on anomaly detection tend to produce many false alarms because they are not capable of discriminating between abnormal patterns triggered by an otherwise legitimate usage and those triggered by an intrusion. Some well-known anomaly detectors include NIDES [44], Wisdom & Sense [102], and EMERALD [92].

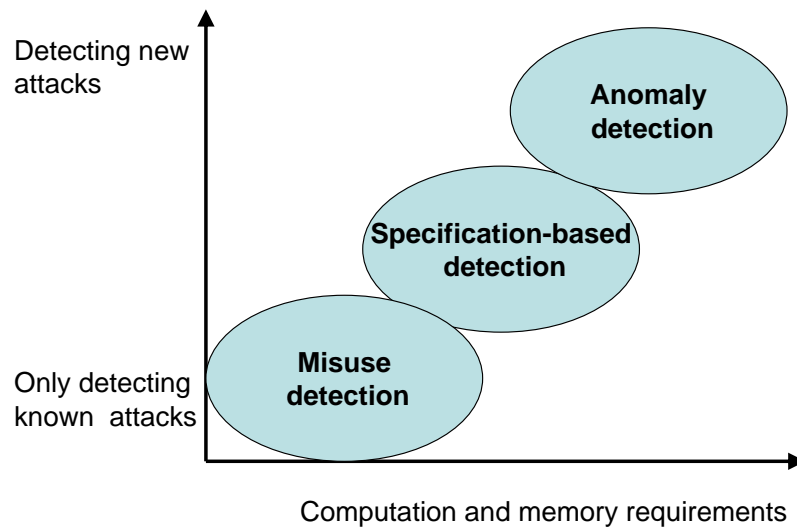


Figure 1.1: Three general approaches to intrusion detection.

In misuse detection, a user's activities are compared with the known signature patterns of attackers. Those matched are then labeled as intrusive activities. Most of today's commercial IDSEs are misuse detection based. While misuse detection can be effective in recognizing known intrusion types, it can not detect novel attacks. Research efforts on misuse detection systems include Bro [91] and NSTAT [49].

Specification-based detection [52] [101], in contrast, determines whether or not a sequence of instructions violates a specification of how a program, or system, should execute. If so, it reports a possible intrusion. Although specification-based detection has the potential for providing a very low false positive rate while detecting a wide range of attacks, it is difficult to model complex programs or systems and write security specifications for them. Figure 1.1 illustrates the difference of the three general approaches to intrusion detection in terms of their computation and memory requirements and capability to detect novel, unseen attacks. In practice, these general approaches are often combined to detect attacks more efficiently.

A detailed survey of intrusion detection research can be seen elsewhere [72] [5] [83]. While a great deal of research has been done in intrusion detection, much of it has

been based on a combination of intuition and ad hoc techniques. Current IDSeS are still plagued by excessive false alarms and poor attack detection accuracy (especially against novel attacks and insider threat).

1.2 Machine Learning Based Anomaly Detection

In this dissertation, we examine various aspects of machine learning based anomaly detection in computer security. Indeed, detection of anomalies in data is one of the fundamental machine learning tasks. Anomaly detection (also known as novelty detection, outlier detection, change detection or activity monitoring) provides the core technology for a broad spectrum of security-centric applications including fraud detection (credit cards, cell phones, etc.), crisis (e.g., epidemic or bio-terrorism) monitoring, news story monitoring, hardware fault detection, and network performance monitoring [28][77]. Although these applications may differ greatly in representation, they share significant characteristics that differentiate them from other machine learning tasks.

The goal of anomaly detection is to identify anomalous activities (i.e., rare, unusual events) in a data stream accurately and in a timely fashion. In applications such as computer security, it is also desirable to be able to explain the reason for the anomaly and propose the appropriate response actions. Due to the lack of first-principle models to describe the complex behavior of the monitored systems, learning from the audit data is necessary. Standard machine learning techniques such as classification, regression and time series analysis are useful as solution components, but they do not completely address the goal of anomaly detection, which has its own idiosyncrasies (e.g., the asymmetry of class distributions and error costs).

Over the years, many machine learning and statistical methods have been proposed for anomaly detection, including rule-based approaches [63], immunological-based approaches [30], neural nets [17] [33], instance-based approaches [59] [67], clustering methods [24] [60], probabilistic learning methods [23] [75], multi-covariance analysis [44], and so on. However, in real-world applications, anomaly based intrusion detection systems tend to give less than satisfactory performance and generate excessive false alarms.

Anomaly detection is a major application of machine learning techniques in the area of computer security. Another significant application is computer access control through user pattern recognition using biometric features, which has achieved limited success over the years (e.g., [4] [7]). Meanwhile, some ad hoc machine learning methods have been employed in other aspects of computer security such as virus recognition [51], misuse detection [55] and intrusion alert aggregation [46]. In this dissertation, we focus on anomaly detection.

1.3 Issues in Anomaly Detection

In addressing the anomaly detection problem with machine learning techniques, several difficult issues arise. These issues are common to most methods for anomaly detection.

1.3.1 Feature Selection

In anomaly detection, there are many different levels at which an IDS could monitor activities on a computer system. Anomalies may be undetectable at one level of granularity or abstraction but easy to detect at a different level. For example, a worm attack might escape detection at the level of a single host, but be detectable when the traffic of the whole network is observed and analyzed. One of the biggest challenges in anomaly detection is to choose features (i.e., attributes) that best characterize the user or system usage patterns so that intrusive behavior will be perceived while non-intrusive activities will not be classified as anomalous.

Even at a certain level of monitoring granularity, one often faces a large number of features representing the monitored object's behavior. For instance, a network connection can be characterized with numerous attributes, including basic features such as source and destination IPs, ports and protocol, and many other secondary attributes. Meanwhile, an audit trail usually consists of sequences of categorical symbols generated from a large discrete alphabet. A program may issue several hundred unique system calls. Similarly, a UNIX user's command history can contain hundreds of different commands or shell scripts. The high dimensionality of the data or the large alphabet size gives rise to a large hypothesis

search space. This, in turn, not only increases the complexity of the problem of learning normal behavior, but also can lead to large classification errors. Therefore, selecting relevant features and eliminating redundant features is vital to the effectiveness of the machine learning technique employed.

1.3.2 Skewed Class Distribution

There is a fundamental asymmetry in anomaly detection problems: normal activity is common and intrusive activity is rare. One often faces a training set consisting of a handful of attack examples and plenty of normal examples, or no attack example at all. This presents a difficult challenge to machine learning methods.

A related issue is the base-rate fallacy in intrusion detection [6]. Since intrusive activity is relatively rare, in order to achieve substantial values for the intrusion detection rate - i.e., the Bayesian probability $P(\text{Intrusion}|\text{Alarm})$, we have to achieve a very low false positive rate. This imposes a high classification accuracy requirement on IDSes.

1.3.3 Distance Metrics and Window Size

Many anomaly detection methods rely on a distance measure in the event space. The degree of suspicion attached to an instance is directly proportional to the distance of the instance from, for example, the nearest normal training examples or the center of the nearest normal clusters. Common distance metrics include Euclidean distance, Manhattan distance, Hamming distance, the vector cosine measure, and so on. A major difficulty is in the construction of a distance measure that reflects a useful metric of similarity. A poor choice of distance metrics may result in meaningless classifications. However, no rationale, except empirical analysis, seems to exist in choosing distance metrics.

The same can be said for the window size for sequential data. To learn temporal or sequential patterns of the audit data stream, a common practice is to slide a window of certain size across the audit trace and determine whether the short sequence within the sliding window is anomalous or not. Sequences of the same length (i.e., the window size) are used for training and testing. A fundamental question is how to determine the appropriate window size for anomaly detection in a systematic way instead of an ad hoc trial-and-

error fashion. Lee and Xiang [65] proposed to use several information-theoretic measures to describe the regularity of an audit dataset and determine the “best” sliding window size based on the conditional entropy and information cost measures. However, Tan and Maxion [97] demonstrated that for stide, a simple instance-based detector that merely remembers previous training sequences (no generalization capability), the appropriate window size was influenced by the length of minimal foreign sequence in the data instead of the conditional entropy. For a general learning method for anomaly detection, choosing the optimal window size awaits further investigation.

1.3.4 Supervised Learning vs. Unsupervised Learning

Many proposed anomaly detection approaches involve supervised learning, where data is manually labeled (normal or abnormal), and then provided to a learning algorithm to build the model of the normal (and abnormal) behavior. The training set is usually assumed noise-free. In practice, an IDS is operating continuously and new data is available at every time instant. Thus it may be prohibitively expensive to clearly label the audit data for training. In contrast, unsupervised learning methods work well with noisy data and do not require class labels. They can play a significant role in intrusion detection.

1.3.5 Concept Drift

Virtually all machine learning research assumes that the training sample is drawn from a stationary data source - the distribution of the data points and the phenomena to be learned are not changing with time. In a practical environment, however, system and network behaviors as well as user activities can change for bona fide reasons. For example, the amount of traffic continues to rise. System and application programs are updated frequently. The continually changing normal behavior, a problem known as *concept drift* in machine learning literature,, presents a significant challenge in anomaly detection. An effective anomaly detection system should be capable of adapting to normal behavior changes while still recognizing anomalous activities. Otherwise, large amount of false alarms would be generated if the model failed to change adaptively to accommodate the new patterns.

1.3.6 Generalization Performance and Training Size Requirements

For a given number of training samples, how significant and reliable is the performance (in terms of classification error) of the anomaly detector? In other words, to what degree can the normal behavior model learned from the finite training set be applied to yet unseen data? In addition, how much training is sufficient in order to achieve certain performance? How will the accuracy of the anomaly detector improve when trained with additional samples? These fundamental questions remain largely unanswered.

1.3.7 Cost-effectiveness of IDS

Most research efforts in intrusion detection have been devoted to improve the technical effectiveness of IDS. That is, to what degree does an IDS detect and prevent intrusions into the target system, and how good is it at reducing false positives? In practice, however, no IDS will ever be 100% accurate in detecting attacks. False positives and false negatives will be inevitably produced. Moreover, the reduction of one type of error (false positive or false negative) is usually accompanied by an increase in the other type.

Cost-effectiveness is an important, yet often overlooked aspect of IDS. When an organization makes an investment decision on a security mechanism such as IDS, risk assessment and cost-benefit analysis is essential. This includes assessment of the organization's assets and values, identification of threats and vulnerabilities, cost-benefit trade-off evaluation, and so on. The major cost factors that ought to be taken into consideration are the operational cost of IDS, the expected loss due to intrusions and the cost of manual or automatic response to an intrusion [61]. Even when the adoption of IDS technology is justifiable, the IDS operator still faces the challenge of employing the IDS properly and determining the best response strategies against various types of attacks in order to minimize the cost of maintaining the IDS while protecting the system assets.

1.4 Overview of the Dissertation

This dissertation addresses some of the aforementioned issues. Chapter 2 provides a brief introduction to the field of machine learning and a quick review of the major learning

methods that have been studied for anomaly detection.

Chapter 3 presents our approach, based on the k -Nearest Neighbor (k NN) classifier, that classifies program behavior as normal or intrusive. Program behavior is represented by frequencies of system calls, instead of short system call sequences. Each system call is treated as a word and the collection of system calls over each program execution as a document. This analogy makes it possible to bring the full spectrum of well-developed text processing methods to bear on the intrusion detection problem.

An adaptive anomaly detection framework is presented in Chapter 4. It is designed for dynamic, changing environments and capable of online learning and concept drift handling. Normal behavior is learned in an incremental, adaptive fashion, through the use of unsupervised evolving connectionist systems. We demonstrate the performance of our adaptive anomaly detection systems and show that the false positive rate can be significantly reduced.

Chapter 5 addresses two important questions in anomaly detection: how to estimate the generalization performance of an anomaly detector? And how much training is sufficient in order to achieve certain performance? An efficient generalization performance estimate, tailored to SVM-based anomaly detection, is introduced. In addition, inverse power-law learning curves are constructed to estimate training size requirements.

Chapter 6 presents a game theoretic methodology for cost-effectiveness analysis of IDS. The interactions between IDS and attackers are modeled in a game playing context to answer questions such as when an IDS is useful and how the IDS should respond to an alarm.

We summarize our conclusions and outline future directions in Chapter 7.

Chapter 2

Machine Learning Approaches to Anomaly Detection

2.1 Machine Learning and Its Problem Formulations

Learning is the process of estimating an unknown input-output dependency, i.e., a model (or hypothesis) of a system using a limited number of observations. In many learning scenarios, it has become popular to use the computer to learn the correct model based on examples of observed behavior. When computers are used to implement these learning algorithms, the discipline is machine learning. In machine learning research, a set of training data is used to search (or build or learn) for a model (or a hypothesis) in a space of possible models (hypotheses). Given some training data, machine learning is equivalent to searching the hypothesis space (or model space) for the best possible hypothesis that describes the observed training data. Evidently, a well-defined learning problem requires a well-formulated problem, a performance metric and a source of training experience.

Depending on how the learning task is formulated in terms of the inputs that drive learning and the manner in which the learned knowledge is utilized, we can divide the machine learning tasks into three broad formulations [21]:

- Learning for classification and regression. This is the most common formulation of machine learning. Classification involves assigning a test case to one of a finite set

of classes, whereas regression instead predicts the case's value on some continuous variable or attribute. There exist a variety of well-established methods for classification and regression, including decision trees, rule induction, neural networks, support vector machines, nearest neighbor approaches and probabilistic methods.

- Learning for acting and planning. It addresses learning of knowledge for selecting actions or plans for an agent to carry out. One well known example is reinforcement learning, where the agent typically carries out an action and receives some reward that indicates the desirability of the resulting states.
- Learning for interpretation and understanding. It focuses on learning knowledge that lets one interpret and understand situations or events. Approaches of this formulation attempt to interpret observations in a more constructive way than simple classification, by combining a number of separate knowledge elements to explain them, a process often referred to as abduction.

In the context of anomaly detection, it is straightforward to cast it as a classification problem (with two classes: normal and abnormal). Indeed most existing approaches proposed for anomaly detection employ some learning mechanisms to capture the monitored object's normal usage patterns and classify new behavior as either normal or abnormal. However, one could also formulate it as a problem of understanding and explaining anomalous behavior. Yet another option would be the formulation that focuses on learning for selecting appropriate responses in the presence of intrusions. Each of these three formulations suggests different approaches to the anomaly detection task, and requires different input data and prior knowledge.

2.2 Learning Methods for Anomaly Detection

In this section, we review the major learning methods that have been proposed for intrusion detection. Although they all represent "learning for classification," they do fall into two broad categories:

- A generative (also known as profiling) approach builds a model solely based on normal training examples and evaluates each testing instance to see how well it fits the model. Activities that deviate significantly from normal trigger an alarm.
- A discriminative approach attempts to learn the distinction between the normal and abnormal classes. Both normal and attack examples are used in training. New activities are classified as either normal or abnormal.

A generative approach aims to define the perimeter of “normal” and thus tends to generate more false alarms. However, it is robust over noisy training data. In contrast, a discriminative approach may give better classification performance if clean, labeled training data is available from both classes. Usually normal examples are common and attack examples are rare. It is common to have a training set with skewed class distributions - much more normal examples than attack examples.

Within each broad class of intrusion detection approaches, there exist many learning techniques that differ in the knowledge representations and their specific algorithms for using and learning that knowledge. Common knowledge representations include rules, decision trees, linear and non-linear functions (including neural networks and support vector machines (SVM)), instance libraries, probabilistic summaries, and so on. Table 2.1 summarizes the major techniques that have been used for intrusion detection, some of which can be employed in both generative and discriminative manners.

It is non-trivial to compare these techniques against each other as each has its own strengths and weaknesses, let alone their different knowledge representations. In general, rule-based approaches can provide rules that are easy to understand, but they require expensive training process as many neural nets based learning methods do. On the other hand, the cost of classifying a new instance is high for instance-based and immunological-based approaches since the new instance is often compared with a large corpus of normal or abnormal instances. Methods based on clustering and probability density estimation approaches usually require a larger number of samples than other approaches.

Table 2.1: Summary of major machine learning techniques for intrusion detection.

| Style | Knowledge representation | Learning algorithm examples | Features |
|----------------------------------|---|--|---|
| Rule-based | Classification rules that characterize normal (and intrusive) | RIPPER [63] [38][106], time-based inductive learning [100] | Concise and intuitive rules, easy to understand and inspect |
| Immunological-based | Self (or non-self) model representing the set of normal (or anomalous) instances. | Negative selection, positive selection [30] [26] | Suitable for distributed processing |
| Neural nets | Linear/non-linear classification functions | Recurrent networks [33] SVM [86][40] | High classification accuracy |
| Instance-based | Instance library | k -nearest neighbor [59][67] | No training involved, classification cost can be high when library is large |
| Clustering and outlier detection | Clusters in input (or feature) space | k -nearest neighbor, one-class SVM [24], density based local outliers [60] | Unsupervised learning, no class label required |
| Probabilistic learning | Probability summaries | Markov models [106][57], Mixture model [23], Bayesian networks [54] | Robust over noisy data, require large data |

2.3 Audit Data

In order to detect intrusions, some source of information in which the intrusion is manifest must be observed and some analysis that can reveal the intrusion must be performed. Systems that obtain the data to analyze from the operating system or applications subject to attack are called host based. The events audited by operating systems usually include the use of identification and authentication mechanisms (login etc.), file opens and program executions, deletion of objects, administrative actions, and other security relevant events. The audit trail should be protected from unauthorized access or tampering. Most modern operating systems have such basic auditing capabilities. Windows NT and Solaris are examples that support the so-called C2-level security audit. The Solaris BSM audit mechanism [84], in particular, provides the ability to collect detailed security relevant data at the system call level. Most host-based systems collect data continuously as the system is operating. The substantial amount of auditing, however, could impact the host system performance and require large storage space. In addition, some intrusions may not directly manifest themselves in the audit trail.

Alternatively, network based intrusion detection systems observe the network traffic that goes to and from the monitored system(s) and look for signs of intrusions in that data. The advantage of network based data collection is that a single sensor, properly placed, can monitor a number of hosts and can look for attacks that target multiple hosts. However, the rapidly increasing network data rates and encrypted connections are the major challenges for network monitoring. Depending on the type of information that is used for intrusion detection, we can further distinguish between traffic and application models. Systems that use traffic models monitor the flow of network packets. The source and destination IP addresses and port numbers are used to determine the features such as the number of total connection arrivals in a certain period of time, the inter-arrival time between packets or the number of packets to/from a certain host. These features, in turn, are used to model the normal traffic and detect attacks such as port scans or denial-of-service attacks. In contrast, the application (or service) model attempts incorporate application specific knowledge of the network services (e.g., HTTP, DNS, FTP) to detect more sophis-

licated attacks. The packet header as well as the application payload information is used to establish the normal traffic model for each service.

In order to facilitate quantitative evaluations of IDS performance and comparisons of different intrusion detection methods, researchers have realized the need for standardized datasets. Described below are several widely cited and publicly available host-based and network-based datasets for research purposes. To our knowledge, there are some privately owned audit datasets, including Windows NT user profiling data and network traffic data, which are not available in the public domain.

2.3.1 DARPA/KDD Datasets

Sponsored by the Department of Defense Advanced Research Projects Agency (DARPA), the MIT Lincoln Laboratory conducted the most comprehensive evaluations of research IDSes in 1998 and 1999 [69][70]. In these evaluations, researchers were given sensor data in the form of sniffed network traffic (TCPdump), Solaris BSM audit data, Windows NT audit data and file-system snapshots and asked to identify the intrusions that had been carried out during the data collection period. The evaluation effort used a testbed, which generated live background traffic similar to that on an Air Force base containing hundreds of users on thousands of hosts. During the 1999 evaluation, more than 200 instances of 58 attack types (including stealthy and novel attacks) were embedded in seven weeks of training data and two weeks of test data. Automated attacks were launched against three UNIX victim machines (SunOS, Solaris, Linux), Windows NT hosts, and a router in the presence of background traffic. Attack categories include DoS, probe, remote-to-local, and user-to-super-user attacks. The DARPA evaluations resulted in the development of an intrusion detection corpus that includes weeks of background traffic and host audit logs, and hundreds of labeled and documented attacks [16]. Part of this corpus, the preprocessed TCPdump data consisting of 41 features, was used for the 1999 KDD Cup contest [22], held at the fifth ACM International Conference on Knowledge Discovery and Data Mining. The DARPA corpus, especially the preprocessed KDD dataset portion, has been used extensively by researchers.

The DARPA evaluations have been criticized for their design and execution, how-

ever. As pointed out in [81], the flaws in the DARPA evaluations include failures to appropriately validate the background data (especially with respect to its ability to cause false alarms), the lack of an appropriate unit of analysis for reporting false alarms and the use of questionable or inappropriate data analysis and presentation techniques. Nevertheless, it is still possible to mix the well-behaved DARPA data with real-world data and conduct meaningful intrusion detection analysis [76].

2.3.2 UNM System Call Data

In a ground-breaking study, Forrest et al. [30] discovered that the short sequences of system calls made by a UNIX program during its normal execution are very consistent. More importantly, the sequences are different from the sequences of its abnormal, exploited executions as well as the executions of other programs. Therefore a concise database consisting of these normal sequences can be used as the “self” definition of the normal behavior of a program, and as the basis to detect anomalies (i.e., “non-self”). A number of follow-on studies, for example, [62][106] [33], attempted alternative models with the system call sequences including classification rules, neural nets, hidden Markov model, variable-length patterns, etc..

Forrest’s group has collected several data sets of system calls executed by active processes and made them publicly available at University of New Mexico [87]. These datasets include different kinds of programs (e.g., programs that run as daemons and those that do not), programs that vary widely in their size and complexity, and different kinds of intrusions (buffer overflows, symbolic link attacks, and Trojan programs). Only programs that run with privilege are included, because misuse of these programs has the greatest potential for harm to the system.

2.3.3 UNIX Command Data

User profiling has been considered as an important technique for detecting an insider or masqueraders misuse of information systems. The underlying assumption is that hostile activity is unusual activity that will manifest as significant excursions from normal user profiles. A user profile contains information that characterizes a system users behavior,

such as commands issued, files normally accessed, resource usage, periods of time that user is normally logged in, keystroke patterns, and a wide variety of other attributes. A popular choice has been the user commands. Lane and Brodley [59] [57] modeled truncated UNIX command data (no arguments) for intrusion detection using instance-based models and hidden Markov models. Two different user populations were used in their study. The first group comprised eight different UNIX users at Purdue University, monitored over the course of more than two years. This set of UNIX command data does not appear to be publicly available now. The second group is a subset of the 168 users monitored by Saul Greenberg at the University of Calgary. The original Greenberg data, documented in [35], comprised full command line entries from 168 volunteer users of the UNIX csh system. The data are further split into four groups: 55 novice users, 36 experienced users, 52 computer-scientist users, and 25 non-programmer users, all of whom were affiliated with the University of Calgary (Canada) as students, faculty, researchers or staff. This user command dataset is available for research use on request.

Schonlau and his colleagues applied a number of techniques, including Bayes 1-Step Markov, Hybrid Multi-Step Markov, IPAM, and so forth [94], to the same UNIX command dataset. The data, available online at <http://www.schonlau.net/>, contains user IDs and command names only (no arguments). This limitation was imposed for privacy reasons. The first 15,000 commands for each of about 70 users were recorded over a period of several months. Some users generated 15,000 commands in a few days; others took a few months. Some commands not explicitly typed by the user (e.g., those generated by shell files or scripts) were also included, as were names of executable programs. To evaluate the intrusion detection performance, Schonlau et al. [94] randomly selected 50 users out of the 70 from whom data were collected to serve as intrusion targets. The remaining 20 users were used as masqueraders and their commands were interspersed into the data of the 50 intrusion targets. Each users data was decomposed into 150 blocks of 100 commands each. The first 50 blocks (5000 commands) of all users, free of contamination by masqueraders, were kept aside as training data. The last 10,000 (masquerader-injected with certain probability) commands were used as testing data for each user. Maxion and Townsend [80] suggested that command line data alone, without arguments, is not enough to profile users. More

recently, using the Greenberg data as the testbed, Maxion [79] demonstrated that the command data, enriched with command-line flags and arguments, facilitated masquerade detection with a significant reduction in the overall cost of errors, compared to truncated user command data.

Chapter 3

Learning Program Behavior with K-Nearest Neighbor Classifier

3.1 Introduction

Intrusions most often occur when programs are misused. In Unix, intruders usually gain super-user status by exploiting privileged programs. A program profile can be generated by monitoring the program execution and capturing the system calls associated with the program. Compared to user behavior profiles, program profiles are more stable over time because the range of program behavior is more limited. Furthermore, it would be more difficult for attackers to perform intrusive activities without revealing their tracks in the execution logs. Therefore program profiles provide concise and stable tracks for intrusion detection. However, almost all the research in learning program behavior has used short sequences of system calls as the observable, and generated a large individual database of system call sequences for each program. A program's normal behavior is characterized by its local ordering of system calls, and deviations from their local patterns are regarded as violations of an executing program. It is still a tedious and costly approach because system and application programs are constantly updated, and it is difficult to build profiles for all of them.

In this chapter, we discuss a new technique for learning program behavior in intru-

sion detection. Our approach employs the k -Nearest Neighbor (k NN) classifier to categorize each new program behavior into either normal or intrusive class. The frequencies of system calls used by a program, instead of their local ordering, are used to characterize the program's behavior. Each system call is treated as a “word”, and each process, i.e., program execution, as a “document”. Then the k NN algorithm, which has been successful in text categorization applications [109], can be easily adapted to intrusion detection. Since there is no need to build a profile for each program and check every sequence during the new program execution, the amount of calculation involved is largely reduced.

The rest of this chapter is organized as follows. Section 2 surveys related work. We explain the analogy between text categorization and intrusion detection and the k NN classifier in Section 3. Section 4 describes our experiments with the 1998 DARPA data, and Section 5 contains further discussions. Finally, we summarize our conclusions in Section 6.

3.2 Related Work

Ko et al. at UC Davis first proposed to specify the intended behavior of some privileged programs (setuid root programs and daemons in Unix) using a program policy specification language [52]. During the program execution, any violation of the specified behavior was considered “misuse”. The main limitation of this method is the difficulty of determining the intended behavior and writing security specifications for all monitored programs. Nevertheless, this research opened the door of modeling program behavior for intrusion detection.

Forrest's group at the University of New Mexico introduced the idea of using short sequences of system calls issued by running programs as the discriminator for intrusion detection [30]. Normal behavior was defined in terms of short sequences of system calls of a certain length in a running Unix process, and a separate database of normal behavior was built for each process of interest. This work was extended with various classification schemes such as artificial immune systems [29], rule learning [62] and Hidden Markov Model (HMM) [106]. Ghosh and others [33] employed artificial neural network techniques to learn program behavior profiles with system call sequences for the 1998 DARPA BSM data. More than

150 program profiles were established. For each program, a neural network was trained and used for anomaly detection. Their Elman recurrent neural networks were able to detect 77.3% of all intrusions with no false positives, and 100% of all attacks with about 10% miss-classified normal sessions.

Unlike most researchers who concentrated on building individual program profiles, Asaka et al. [3] introduced a method based on discriminant analysis. Without examining all system calls, an intrusion detection decision was made by analyzing only 11 system calls in a running program and calculating the program's Mahalanobis' distances to normal and intrusion groups of the training data. There were 4 instances that were misclassified out of 42 samples. Due to its small size of sample data, however, the feasibility of this approach still needs to be established.

Our approach treats the system calls differently. Instead of looking at the local ordering of the system calls, our method uses the frequencies of system calls to characterize program behavior. Using the text processing metaphor, each system call is treated as a "word" in a long document and the set of system calls generated by a process is treated as the "document". This analogy makes it possible to bring the full spectrum of well-developed text processing methods [1] to bear on the intrusion detection problem. One such method is the k -nearest neighbor classification method.

3.3 K-Nearest Neighbor Text Categorization Method

Text categorization is the process of grouping text documents into one or more predefined categories based on their content. A number of statistical classification and machine learning techniques have been applied to text categorization, including regression models, Bayesian classifiers, decision trees, nearest neighbor classifiers, neural networks, and support vector machines [1].

The first step in text categorization is to transform documents, which typically are strings of characters, into a representation suitable for the learning algorithm and the classification task. The most commonly used document representation is the so-called vector space model. In this model, each document is represented by a vector of words. A word-

by-document matrix \mathbf{A} is used for a collection of documents, where each entry represents the occurrence of a word in a document, i.e., $\mathbf{A} = (a_{ij})$, where a_{ij} is the weight of word i in document j . There are several ways of determining the weight a_{ij} . Let f_{ij} be the frequency of word i in document j , N the number of documents in the collection, M the number of distinct words in the collection, and n_i the total number of times word i occurs in the whole collection. The simplest approach is Boolean weighting, which sets the weight a_{ij} to 1 if the word occurs in the document and 0 otherwise. Another simple approach uses the frequency of the word in the document, i.e.,

$$a_{ij} = f_{ij} \quad (3.1)$$

A more common weighting approach is the so-called *tf · idf* (term frequency - inverse document frequency) weighting:

$$a_{ij} = f_{ij} \times \log \left(\frac{N}{n_i} \right) \quad (3.2)$$

A slight variation [56] of the *tf · idf* weighting, which takes into account that documents may be of different lengths, is the following:

$$a_{ij} = \frac{f_{ij}}{\sqrt{\sum_{l=1}^M f_{lj}^2}} \times \log \left(\frac{N}{n_i} \right) \quad (3.3)$$

For matrix \mathbf{A} , the number of rows corresponds to the number of words M in the document collection. There could be hundreds of thousands of different words. In order to reduce the high dimensionality, stop-word (frequent word that carries no information) removal, word stemming (suffix removal) and additional dimensionality reduction techniques, feature selection or re-parameterization [1], are usually employed.

To classify a class-unknown document X , the k -Nearest Neighbor classifier algorithm ranks the document's neighbors among the training document vectors, and uses the class labels of the k most similar neighbors to predict the class of the new document. The classes of these neighbors are weighted using the similarity of each neighbor to X , where similarity is measured by Euclidean distance or the cosine value between two document vectors. The cosine similarity is defined as follows:

$$sim(X, D_j) = \frac{\sum_{t_i \in (X \cap D_j)} x_i \times d_{ij}}{\|X\|_2 \times \|D_j\|_2} \quad (3.4)$$

Table 3.1: Analogy between text categorization and intrusion detection when applying the k NN classifier.

| Terms | Text categorization | Intrusion Detection |
|----------|--|--|
| N | total number of documents | total number of processes |
| M | total number of distinct words | total number of distinct system calls |
| n_i | number of times i th word occurs | number of times i th system call was issued |
| f_{ij} | frequency of i th word in document j | frequency of i th system call in process j |
| D_j | j th training document | j th training process |
| X | test document | test process |

where X is the test document, represented as a vector; D_j is the j th training document; t_i is a word shared by X and D_j ; x_i is the weight of word t_i in X ; d_{ij} is the weight of word t_i in document D_j ; $\|X\|_2 = \sqrt{x_1^2 + x_2^2 + x_3^2 + \dots}$ is the norm of X , and $\|D_j\|_2$ is the norm of D_j . A cutoff threshold is needed to assign the new document to a known class.

The k NN classifier is based on the assumption that the classification of an instance is most similar to the classification of other instances that are nearby in the vector space. Compared to other text categorization methods such as Bayesian classifier, k NN does not rely on prior probabilities, and it is computationally efficient. The main computation is the sorting of training documents in order to find the k nearest neighbors for the test document.

We seek to draw an analogy between a text document and the sequence of all system calls issued by a process, i.e., program execution. The occurrences of system calls can be used to characterize program behavior and transform each process into a vector. Furthermore, it is assumed that processes belonging to the same class will cluster together in the vector space. Then it is straightforward to adapt text categorization techniques to modeling program behavior. Table 3.1 illustrates the similarity in some respects between text categorization and intrusion detection when applying the k NN classifier.

There are some advantages to applying text categorization methods to intrusion detection. First and foremost, the size of the system-call vocabulary is very limited. There are less than 100 distinct system calls in the DARPA BSM data, while a typical text categorization problem could have over 15000 unique words [1]. Thus the dimension of the word-by-document matrix \mathbf{A} is significantly reduced, and it is not necessary to apply

any dimensionality reduction techniques. Second, we can consider intrusion detection as a binary categorization problem, which makes adapting text categorization methods very straightforward.

3.4 Experiments

3.4.1 Dataset

We applied the k -Nearest Neighbor classifier to the 1998 DARPA data. The 1998 DARPA Intrusion Detection System Evaluation program provides a large sample of computer attacks embedded in normal background traffic[16]. The *TCPDUMP* and *BSM* audit data were collected on a network that simulated the network traffic of an Air Force Local Area Network. The audit logs contain seven weeks of training data and two weeks of testing data. There were 38 types of network-based attacks and several realistic intrusion scenarios conducted in the midst of normal background data.

We used the Basic Security Module (BSM) audit data collected from a victim Solaris machine inside the simulation network. The BSM audit logs contain information on system calls produced by programs running on the Solaris machine. See [84] for a detailed description of BSM events. We only recorded the names of system calls. Other attributes of BSM events, such as arguments to the system call, object path and attribute, return value, etc., were not used here, although they could be valuable for other methods.

The DARPA data was labeled with session numbers. Each session corresponds to a TCP/IP connection between two computers. Individual sessions can be programmatically extracted from the BSM audit data. Each session consists of one or more processes. A complete ordered list of system calls is generated for every process. A sample system call list is shown below. The first system call issued by Process 994 was *close*, *execve* was the next, then *open*, *mmap*, *open* and so on. The process ended with the system call *exit*.

Process ID: 994

Table 3.2: List of 50 distinct system calls that appear in the training dataset.

| | | | | | | | |
|----------|-----------|---------|----------|---------|----------|---------|-----------|
| access | audit | auditon | chdir | chmod | chown | close | creat |
| execve | exit | fchdir | fchown | fcntl | fork | fork1 | getaudit |
| getmsg | ioctl | kill | link | login | logout | lstat | mementl |
| mkdir | mmap | munmap | nice | open | pathconf | pipe | putmsg |
| readlink | rename | rmdir | setaudit | setegid | seteuid | setgid | setgroups |
| setpgrp | setrlimit | setuid | stat | statvfs | su | sysinfo | unlink |
| utime | vfork | | | | | | |

close *execve* *open* *mmap* *open* *mmap* *mmap* *munmap* *mmap*
mmap *close* *open* *mmap* *close* *open* *mmap* *mmap* *munmap*
mmap *close* *close* *munmap* *open* *ioctl* *access* *chown* *ioctl*
access *chmod* *close* *close* *close* *close* *close* *exit*

The numbers of occurrences of individual system calls during the execution of a process were counted. Then text weighting techniques were used to transform the process into a vector. We used two weighting methods, frequency weighting defined by (3.1) and *tf* · *idf* weighting defined by (3.3), to encode the processes.

During our off-line data analysis, our dataset included system calls executed by all processes except the processes of the Solaris operating system such as the *inetd* and shells, which usually spanned several audit log files.

3.4.2 Anomaly Detection

First we implemented intrusion detection solely based on normal program behavior. In order to ensure that all possible normal program behaviors are included, a large training dataset is preferred for anomaly detection. On the other hand, a large training dataset means large overhead in using a learning algorithm to model program behavior. There are 5 simulation days that were free of attacks during the seven-week training period. We arbitrarily picked 4 of them for training, and used the fifth one for testing. Our training normal dataset consists of 606 distinct processes running on the victim Solaris machine during these 4 simulation days. There are 50 distinct system calls observed from the training dataset, which means each process is transformed into a vector of size 50. Table 3.2 lists all the 50 system calls.

Once we have the training dataset for normal behavior, the *k*NN text categorization

```

build the training normal dataset  $D$ ;
for each process  $X$  in the test data do
    if  $X$  has an unknown system call then
         $X$  is abnormal;
    else then
        for each process  $D_j$  in training data do
            calculate  $sim(X, D_j)$ ;
            if  $sim(X, D_j)$  equals 1.0 then
                 $X$  is normal; exit;
        find  $k$  biggest scores of  $sim(X, D)$ ;
        calculate  $sim\_avg$  for  $k$ -nearest neighbors;
        if  $sim\_avg$  is greater than  $threshold$  then
             $X$  is normal;
        else then
             $X$  is abnormal;

```

Figure 3.1: Pseudo code for the k NN classifier algorithm for anomaly detection.

method can be easily adapted for anomaly detection. We scan the test audit data and extract the system call sequence for each new process. The new process is also transformed to a vector with the same weighting method. Then the similarity between the new process and each process in the training normal process dataset is calculated using Equation 3.4. If the similarity score of one training normal process is equal to 1, which means the system call frequencies of the new process and the training process match perfectly, then the new process would be classified as a normal process immediately. Otherwise, the similarity scores are sorted and the k nearest neighbors are chosen to determine whether the new program execution is normal or not. We calculate the average similarity value of the k nearest neighbors (with highest similarity scores) and set a threshold. Only when the average similarity value is above the threshold, is the new process considered normal. The pseudo code for the adapted k NN algorithm is presented in Figure 3.1.

In intrusion detection, the Receiver Operating Characteristic (ROC) curve is usually used to measure the performance of the method. The ROC curve is a plot of intrusion detection accuracy against the false positive probability. It can be obtained by varying the detection threshold. We formed a test dataset to evaluate the performance of the k NN classifier algorithm. The BSM data of the third day of the seventh training week was chosen as part of the test dataset (none of the training processes was from this day). There was no attack launched on this day. It contains 412 sessions and 5285 normal processes (We did not require the test processes to be distinct in order to count false alarms for one day). The rest of the test dataset consists of 55 intrusive sessions chosen from the seven-week DARPA training data. There are 35 clear or stealthy attack instances included in these intrusive sessions (some attacks involve multiple sessions), representing all types of attacks and intrusion scenarios in the seven-week training data. Stealthy attacks attempt to hide perpetrator’s actions from someone who is monitoring the system, or the intrusion detection system. Some duplicate attack sessions of the types *eject* and *warezclient* were skipped and not included in the test dataset. When a process is categorized as abnormal, the session that the process is associated with is classified as an attack session. The intrusion detection accuracy is calculated as the rate of detected attacks. Each attacks counts as one detection, even with multiple sessions. Unlike the groups who participated in the 1998 DARPA Intrusion Detection Evaluation program [69], we define our false positive probability as the rate of mis-classified processes, instead of mis-classified sessions.

The performance of the k NN classifier algorithm also depends on the value of k , the number of nearest neighbors of the test process. Usually the optimal value of k is empirically determined. We varied k ’s value from 5 to 25. Figure 3.2 shows the ROC curves for 3 different k values when the processes are transformed with the *tf·idf* weighting method. For this particular dataset, $k=10$ is a better choice than other values in that the attack detection rate reaches 100% faster. For $k=10$, the k NN classifier algorithm can detect 10 of the 35 attacks with zero false positive rate. And the detection rate reaches 100% rapidly when the threshold is raised to 0.72 and the false positive rate remains as low as 0.44% (23 false alarms out of 5285 normal processes) for the whole simulation day.

We also employed the frequency weighting method to transform the processes of

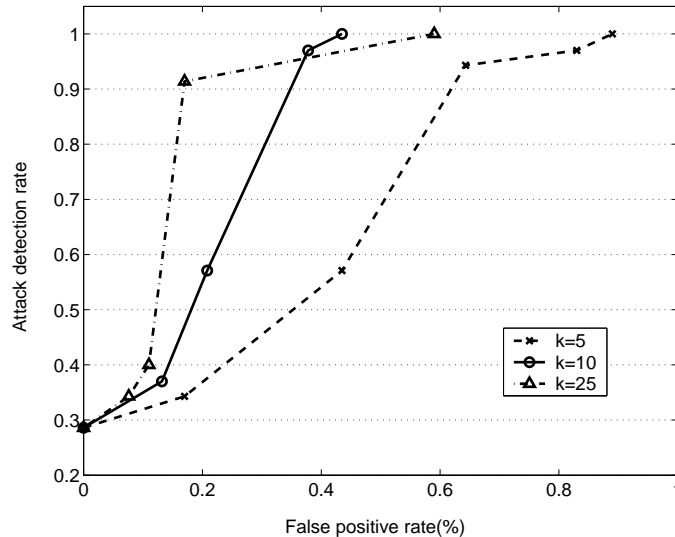


Figure 3.2: Performance of the k NN classifier method expressed in ROC curves for the $tf \cdot idf$ weighting method. False positive rate vs attack detection rate for $k=5$, 10 and 25.

the same training and test datasets. Similarly, for the frequency weighting method, $k=15$ provides the lowest false positive rate 0.87% (46 false alarms out of 5285 normal processes) when the attack detection rate reaches 100% with the threshold value of 0.99. The reason for the high threshold value is that some attack instances are very similar to the normal processes with the frequency weight. A comparison of two different weighting methods is shown in Figure 3.3. while the frequency weighting method offers a desirable high attack detection rate (86%) at zero false positives, the $tf \cdot idf$ weighting method provides lower false positive rate at 100% attack detection rate. It appears that the $tf \cdot idf$ weighting can make process vectors of two classes more distinguishable than the frequency weighting. Therefore, a lower threshold value is needed, and better false positive rate can be achieved with the $tf \cdot idf$ weighting method.

3.4.3 Anomaly Detection Combined with Signature Verification

We have just shown that the k NN classifier algorithm can be implemented for effective abnormality detection. The overall running time of the k NN method is $O(N)$, where N is the number of processes in the training dataset (usually k is a small constant). When N is large, this method could still be computationally expensive for some real-time intrusion detection systems. In order to detect attacks more effectively, the k NN anomaly detection

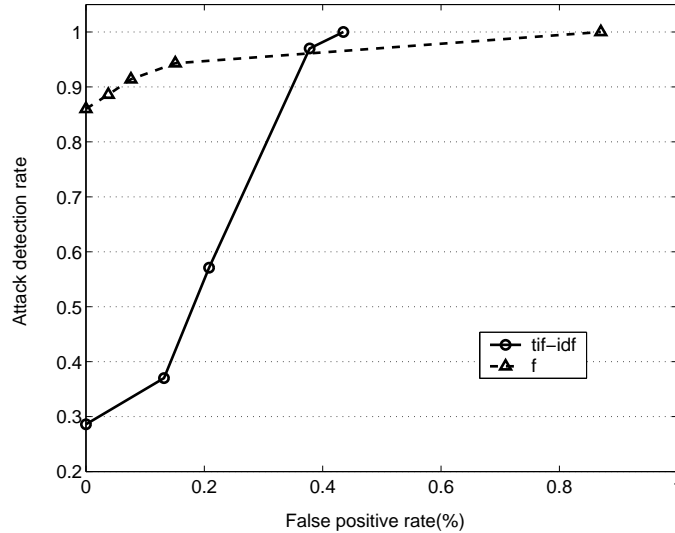


Figure 3.3: ROC curves for $tf \cdot idf$ weighting ($k=10$) and frequency weighting ($k=15$).

can be easily integrated with signature verification. The malicious program behavior can be encoded into the training set of the classifier. After carefully studying the 35 attack instances within the seven-week DARPA training data, we generated a dataset of 19 intrusive processes. This intrusion dataset covers most attack types of the DARPA training data. It includes the most clearly malicious processes, including *ejectexploit*, *formatexploit*, *ffbeexploit* [50] and so on.

For the improved k NN algorithm, the training dataset includes 606 normal processes as well as the 19 aforementioned intrusive processes. The 606 normal processes are the same as the ones in subsection 4.2. Each new test process is compared to intrusive processes first. Whenever there is a perfect match, i.e., the cosine similarity is equal to 1.0, the new process is labeled as intrusive behavior (one could also check for near matches). Otherwise, the abnormal detection procedure in Figure 3.1 is performed. Due to the small amount of the intrusive processes in the training dataset, this modification of the algorithm only causes minor additional calculation for normal testing processes.

The performance of the improved k NN classifier algorithm was evaluated with 24 attacks within the two-week DARPA testing audit data. The DARPA testing data contains some known attacks as well as novel ones. Some duplicate instances of the *eject* attack were not included in the test dataset. The false positive rate was evaluated with the same

Table 3.3: Attack detection rate for DARPA testing data when anomaly detection is combined with signature verification.

| Attack | Instances | Detected | Detection rate |
|---------------|-----------|----------|----------------|
| Known attacks | 16 | 16 | 100% |
| Novel attacks | 8 | 6 | 75% |
| Total | 24 | 22 | 91.7% |

5285 testing normal processes as described in Section 4.2. Table 3.3 presents the the attack detection accuracy for the *tf·idf* weighting ($k=10$ and *threshold* = 0.8) and the frequency weighting ($k = 15$ and *threshold* = 0.99). For the *tf·idf* weighting method, the false positive rate is 0.59% (31 false alarms) when the threshold is adjusted to 0.8. For the frequency weighting, the false positive rate remains at 0.87% with the threshold of 0.99.

The two missed attack instances were a new denial of service attack, called *process table*. They matched with one of training normal processes exactly, which made it impossible for the k NN algorithm to detect. The *process table* attack was implemented by establishing connections to the telnet port of the victim machine every 4 seconds and exhausting its process table so that no new process could be launched [50]. Since this attack consists of abuse of a perfectly legal action, it didn't show any abnormality when we analyzed individual processes. Characterized by an unusually large number of connections active on a particular port, this denial of service attack, however, could be easily identified by other intrusion detection methods.

Among the other 22 detected attacks, eight were captured with signature verification for the *tf·idf* weighting and two for the frequency weighting.

3.5 Discussion

The RSTCORP group [33] gave good performance during the evaluation of the 1998 DARPA BSM data [69]. A neural network was trained for each program. Their Elman recurrent neural networks were able to detect 77.3% of all intrusions with no false positives, and 100% of all attacks with about 10% miss-classified normal sessions. which means 40 to 50 false positive alarms for a typical simulation day with 500 sessions. Their test data

consisted of 139 normal sessions and 22 intrusive sessions. Since different test datasets were used, it is difficult to compare the performance of our k NN classifier with that of the Elman networks. In spite of that, the k NN classifier avoids the time-consuming training process, and more importantly, bypasses the need to learn individual program profiles separately. Thus the cost of learning program behavior is significantly decreased.

Unlike the $tf \cdot idf$ weighting, the frequency-weighting method assigns the number of occurrences of a system call during the process execution to a vector entry. Each process vector does not carry any information on other processes. A new training process could be easily added to the training dataset without changing the weights of the existing training samples. Therefore the frequency-weighting method makes the k NN classifier method more suitable for dynamic environments that requires frequent updates of the training data.

In our current implementation, we used all the system calls to represent program behavior. The dimension of process vectors, and hence the classification cost, can be further reduced by only using the most influential system calls.

Our approach is predicated on the following properties: the frequencies of system calls issued by a program appear consistently across its normal executions and unseen system calls will be executed or unusual frequencies of the invoked system calls will appear when the program is exploited. We believe these properties hold true for many programs. However, if an intrusion does not reveal any anomaly in the frequencies of system calls, our method would miss it. For example, attacks that consist of abuse of perfectly normal processes such as *process table* would not be identified by the k NN classifier.

With the k NN classifier method, each process is classified when it terminates. We point out that it could still be suitable for real-time intrusion detection. Each intrusive attack is usually conducted within one or more sessions, and every session contains several processes. Since the k NN classifier method monitors the execution of each process, it is very likely that an attack can be detected while it is in operation. However, it is possible that an attacker can avoid being detected by not letting the process exit. Nonetheless, the k NN classifier could be integrated with other methods [15] that utilize the ordering information of system calls for performance enhancement.

3.6 Summary

In this chapter we have presented a new algorithm based on the k -Nearest Neighbor classifier method for modeling program behavior in intrusion detection. Our experiments with the 1998 DARPA BSM audit data have shown that this approach is able to effectively detect intrusive program behavior. Compared to other methods using short system call sequences, the k NN classifier doesn't have to build separate profiles of short system call sequences for different programs, thus the calculation involved with classifying new program behavior is largely reduced. Our results also show that a low false positive rate can be achieved. While this result may not hold against a more sophisticated dataset, text categorization techniques appear to be well applicable to the domain of intrusion detection.

Further research is needed to investigate the reliability and scaling properties of the k NN classifier method. Other directions of future work include:

- How to select the most relevant system calls for classification?
- Quantitative comparison between the k NN classifier and other machine learning methods.
- Mixed modeling of program behavior using both local ordering of system calls and system call frequencies.

Chapter 4

Adaptive Anomaly Detection with Evolving Connectionist Systems

4.1 Introduction

In order to be effective in a practical environment, anomaly detection systems have to be capable of online learning and concept drift handling. In this chapter, we present an adaptive anomaly detection framework that is suitable for dynamic, changing environments. Our framework employs unsupervised evolving connectionist systems to learn system, network or user behavior in an online, adaptive fashion without *a priori* knowledge of the underlying non-stationary data distributions. Normal behavior changes are efficiently accommodated while anomalous activities can still be identified.

Adaptive learning and evolving connectionist systems are an active area of artificial intelligence research. Evolving connectionist systems are artificial neural networks that resemble the human cognitive information processing models. They are stable enough to retain patterns learned from previously observed data while being flexible enough to learn new patterns from new incoming data. Due to their self-organizing and adaptive nature, they provide powerful tools for modeling evolving processes and knowledge discovery [48].

Our adaptive anomaly detection framework performs one-pass clustering of the input data stream that represents a monitored subject's behavior patterns. Each new

incoming instance is assigned to one of the three states: *normal*, *uncertain* and *anomalous*. Two different alarm levels are defined to reduce the risk of false alarming. We evaluated our adaptive anomaly detection systems, based on the Fuzzy Adaptive Resonance Theory (Fuzzy ART) [11] and Evolving Fuzzy Neural Networks (EFuNN) [47], over two types of datasets, the KDD Cup 1999 network data [22] and Windows NT user profiling data. Our experiments show that both evolving connectionist systems are able to adapt to user or network normal behavior changes and at the same time detect anomalous activities. Compared to support vector machines (SVM) based static learning, our adaptive anomaly detection systems significantly reduced the false alarm rate.

The rest of this chapter is organized as follows. In Section 2 we review some related work on adaptive anomaly detection. Section 3 presents our adaptive framework and a brief introduction to Fuzzy ART and EFuNN. Section 4 details our experiments with the KDD Cup 1999 network data and the Windows NT user profiling data. Section 5 provides further discussions and Section 6 summarizes our conclusions and future work.

4.2 Related Work

To handle concept drift and non-stationary data distributions, a common practice is to forget out-of-data statistics of the data and favor recent events using a decay or aging factor. For example, NIDES [44] compares a user's short-term behavior to the user's long-term behavior. The user profiles keep statistics such as frequency table, means and covariance, which are constantly aged by multiplying them by an exponential decay factors. This method of aging creates a moving time window for the profile data, so that the new behavior is only compared to the most recently observed behaviors that fall into the time window. Similarly, SmartSifter [107][108] employs discounting algorithms to gradually fade the effect of past examples. Mahoney and Chan [75] took training decay to the extreme by discarding all events before the most recent occurrence. There is one theoretical and one practical problem with this aging or time window approach. Theoretically, no justification has been provided for the assumption that a user's behavior changes gradually. Notwithstanding this theoretical gap, the decay factor is usually chosen in an *ad-hoc* manner. By

contrast, our evolving connectionist systems are able to adapt to normal behavior changes without losing earlier information.

There were a few other previous efforts on adaptive intrusion detection. Teng et al. [100] proposed a time-based inductive learning approach to perform adaptive real time anomaly detection. Sequential rules were generated dynamically to adapt to changes in a user's behavior. Lane and Brodley [58] proposed a nearest neighbor classifiers based online learning scheme and examined the issues of incremental updating of system parameters and instance selection. Finite mixture models were employed in [25] to generate adaptive probabilistic models and detect anomalies within a dataset. Fan [27] used ensembles of classification models to adapt existing models in order to detect newly established patterns. Hossain and Bridges [39] proposed a fuzzy association rule mining architecture for adaptive anomaly detection.

Compared to previous statistical and rule-learning based adaptive anomaly detection systems, our framework does not require *a priori* knowledge of the underlying data distributions. Through the use of evolving connectionist systems, it provides efficient adaptation to new patterns in a dynamic environment. Unlike other neural networks that have been applied to intrusion detection (e.g., [17] [33]) as “black boxes,” our evolving connectionist systems can provide knowledge (i.e., the weight vectors) to “explain” the learned normal behavior patterns.

Our approach also falls into the category of unsupervised anomaly detection [24] [60] as it does not require the knowledge of data labels. However, our algorithms assign each instance into a cluster in an online, adaptive mode. No distinction between training and testing has to be made. Therefore the period of system initialization during which all behaviors are assumed normal is not necessary.

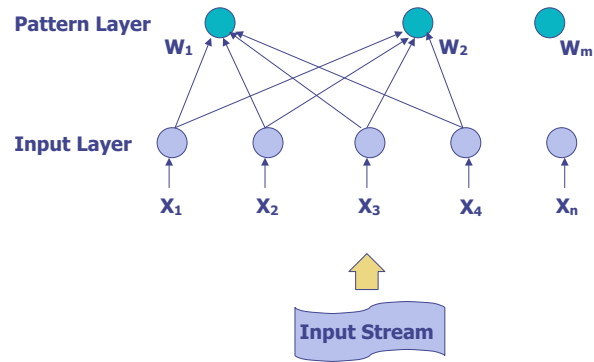
Another research project closely related to ours is ADMIT [95], which uses semi-incremental clustering techniques to create user profiles. Different types of alarms are also introduced.

4.3 Adaptive Anomaly Detection Framework

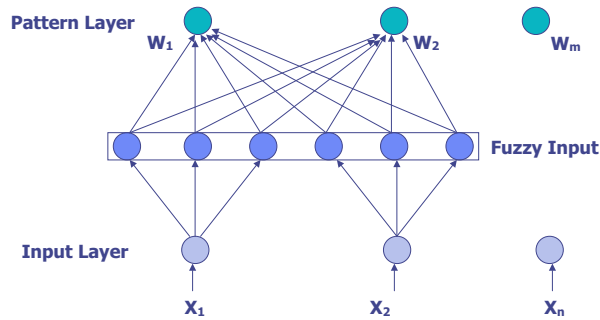
In addressing the problem of adaptive anomaly detection two fundamental questions arise: (a) How to generate a model or profile that can concisely describe a subject's normal behavior, and more importantly, can it be updated efficiently to accommodate new behavior patterns? (b) How to select instances to update the model without introducing noise and incorporating abnormal patterns as normal? Our adaptive anomaly detection framework addresses these issues through the use of online unsupervised learning methods, under the assumption that normal instances cluster together in the input space, whereas the anomalous activities correspond to outliers that lie in sparse regions of the input space. Our framework is general in that the underlying clustering method can be any online unsupervised evolving connectionist system and it can be used for different types of audit data. Without loss of generality, we assume the audit data that is continuously fed into the adaptive anomaly detection system has been transformed into a stream of input vectors after pre-processing, where the input features describe the monitored subject's behavior.

The evolving connectionist systems are designed for modeling evolving processes. They operate continuously in time and adapt their structure and functionality through a continuous interaction with the environment [48]. They can learn in unsupervised, supervised or reinforcement learning modes. The online unsupervised evolving connectionist systems provide one-pass clustering of an input data stream, where there is no predefined number of different clusters that the data belong to.

A simplified diagram of an evolving connectionist system for online unsupervised learning is given in Figure 4.1(a) (some systems such as EFuNN may have an additional fuzzy input layer, shown in Figure 4.1(b), which represents the fuzzy quantization of the original inputs with the use of membership functions [43]). A typical unsupervised evolving connectionist system consists of two layers of nodes: an *input* layer that reads the input vectors into the system continuously, and a *pattern* layer (or cluster layer) representing previously learned patterns. Each pattern node corresponds to a cluster in the input space. Each cluster, in turn, is represented by a weight vector. Then the subject's normal behavior profile is conveniently described as a set of weight vectors that represent the clustering of



(a)



(b)

Figure 4.1: (a) A simplified diagram of an evolving connectionist system for unsupervised learning. The system has n input nodes and m pattern nodes. There is a connection from each input node to every pattern node. Some connections are not shown in the figure. (b) An evolving connectionist system that has an additional fuzzy input layer. The task of the fuzzy input nodes is to transfer the input values into membership degrees.

the previous audit data.

A distance measure has to be defined to measure the mismatch between a new instance (i.e., a new input vector) and existing patterns. Based on the distance measure, the system either assigns an input vector to one of the existing patterns and updates the pattern weight vector to accommodate the new input, or otherwise creates a new pattern node for the input. The details of clustering vary with different evolving connectionist systems.

In order to reduce the risk of false alarms (classifying normal instances as abnormal), we define three states of behavior patterns (i.e., the pattern nodes of the evolving connectionist system): *normal*, *uncertain* and *anomalous*. Accordingly, each instance is labeled as either *normal*, *uncertain* or *anomalous*. In addition, the alarm is differentiated into two levels: *Level 1 alarm* and *Level 2 alarm*, representing different degrees of anomaly. As illustrated in Figure 4.2, a new instance is assigned to one of the existing *normal* patterns and labeled *normal* if the similarity between the input vector and the *normal* pattern is above a threshold (the *vigilance* parameter). Otherwise, it is *uncertain*. The *uncertain* instance is either assigned to one of the existing *uncertain* patterns if it is close enough to that *uncertain* pattern, or becomes the only member of a new *uncertain* pattern. A *Level 1 alarm* is triggered whenever a new *uncertain* pattern is created as the new instance is different from all the learned patterns and thus deserves special attention. At this point, some preliminary security measures need to be taken. However, one can not draw a final conclusion yet. The new instance can be truly anomalous or merely the beginning of a new normal behavior pattern, which will be determined by the subsequent instances. After the processing of a certain number (the N_{watch} parameter) of the subsequent instances in the same manner, if the number of members of an *uncertain* pattern reaches a threshold value (the Min_{count} parameter), the *uncertain* pattern becomes a *normal* pattern and the labels of all its members are changed from *uncertain* to *normal*. This indicates that a new behavior pattern has been developed and incorporated into the subject's normal behavior profile as enough instances have shown the same pattern. On the other hand, after N_{watch} subsequent instances, any *uncertain* pattern with less than Min_{count} members will be destroyed and all its members are labeled *anomalous*. This will make sure that anomalous

patterns, corresponding to the sparse regions in the input space, will not be included into the normal profile. A *Level 2 alarm* is issued when an instance is labeled *anomalous* and further response actions are expected.

1. Assign the first instance to a new uncertain pattern and label it *uncertain*.
2. Consider the next instance.
 - (a) If the similarity between this instance and one of the existing *normal* patterns is above *vigilance*, assign it to the *normal* pattern and label it *normal*.
 - (b) If the instance is close enough to one of the existing *uncertain* patterns, assign it to the *uncertain* pattern and label it *uncertain*.
 - (c) Otherwise, the instance becomes the only member of a new *uncertain* pattern. A *Level 1 alarm* is triggered.
 - (d) Increase the age of each existing *uncertain* pattern by 1.
 - (e) for each *uncertain* pattern whose age has reached N_{watch} , if it has more than Min_{count} members, it becomes a *normal* pattern and change the labels of all its members from *uncertain* to *normal*. Otherwise, the *uncertain* pattern is destroyed and all its members are labeled *anomalous*. *Level 2 alarms* are issued.
3. Repeat step 2 to process subsequent instances.

Figure 4.2: Pseudo code for adaptive anomaly detection.

The main tunable parameters of an adaptive anomaly detection system are summarized as follows:

- *Vigilance* ρ . This threshold controls the degree of mismatch between new instances and existing patterns that the system can tolerate.
- *Learning rate* β . It determines how fast the system should adapt to a new instance when it is assigned to a pattern.
- N_{watch} . It is the period that the system will wait before making a decision on a newly

created *uncertain* pattern.

- *Mincount*, the minimum number of members that an *uncertain* pattern should have in order to be recognized as a *normal* pattern.

Our framework does not require *a priori* knowledge of the number of input features. When a new input feature is presented, the system simply adds a new input node to the input layer and connections from this newly created input node to the existing pattern nodes. This can be very important when the features that describe a subject's behavior grow over time and can't be foreseen in a dynamic environment. Similarly, accommodation of a new pattern is efficiently realized by creating a new pattern node and adding connections from input nodes to this new pattern node. The rest of the structure remains the same.

With the framework, the learned normal profile is expressed as a set of weight vectors representing the coordinates of the cluster centers in the input space. These weight vectors can be interpreted as a knowledge presentation that can be used to describe the subject's behavior patterns, and thus they can facilitate understanding of the subject's behavior. The weight vectors are stored in the long term memory of the connectionist systems. Since new instances are compared to all previously learned patterns, recurring activities would be recognized easily.

While the underlying clustering method of the adaptive anomaly detection framework can be any unsupervised evolving connectionist system, Fuzzy ART and the unsupervised learning version of EFuNN are adapted for anomaly detection in this paper. Both of them are conceptually simple and computationally fast. Furthermore, they cope well with fuzzy data, and the fuzzy distance measures help to smooth the abrupt separation of normality and abnormality of a subject's behavior. Below is a brief introduction to Fuzzy ART and EFuNN.

4.3.1 Fuzzy ART

Fuzzy ART [11] is a member of the Adaptive Resonance Theory (ART) neural network family [10]. It incorporates computations from fuzzy set theory [43] into the ART 1 neural network. It is capable of fast stable unsupervised category learning and pattern

recognition in response to arbitrary input sequences.

Fuzzy ART clusters input vectors into patterns based on two separate distance criteria, *match* and *choice*. For input vector X and pattern j , the *match* function is defined by

$$S_j(X) = \frac{|X \wedge W_j|}{|X|}, \quad (4.1)$$

where W_j is the weight vector associated with pattern j . Here, the fuzzy AND operator \wedge is defined by

$$(X \wedge Y)_i \equiv \min(x_i, y_i), \quad (4.2)$$

and the norm $|\cdot|$ is defined by

$$|X| \equiv \sum_i |x_i|. \quad (4.3)$$

The *choice* function is defined by

$$T_j(X) = \frac{|X \wedge W_j|}{\alpha + |W_j|}, \quad (4.4)$$

where α is a small constant.

For each input vector X , Fuzzy ART assigns it to the pattern j that maximizes $T_j(X)$ while satisfying $S_j(X) \geq \rho$, where ρ is the *vigilance* parameter, $0 \leq \rho \leq 1$. The weight vector W_j is then updated according to the equation

$$W_j^{(new)} = \beta(X \wedge W_j^{(old)}) + (1 - \beta)W_j^{(old)}, \quad (4.5)$$

where β is the *learning rate* parameter, $0 \leq \beta \leq 1$. If no such pattern can be found, a new pattern node is created. This procedure is illustrated in Figure 4.3.

In order to avoid the pattern proliferation problem, Fuzzy ART uses a *complement coding* technique to normalize the inputs. The complement of vector X , denoted by X^c , is defined by

$$(X^c)_i \equiv 1 - x_i. \quad (4.6)$$

For an n -dimensional original input X , the complemented coded input X' to the Fuzzy ART system is the $2n$ -dimensional vector

$$X' = (X, X^c) \equiv (x_1, x_2, \dots, x_n, x_1^c, x_2^c, \dots, x_n^c). \quad (4.7)$$

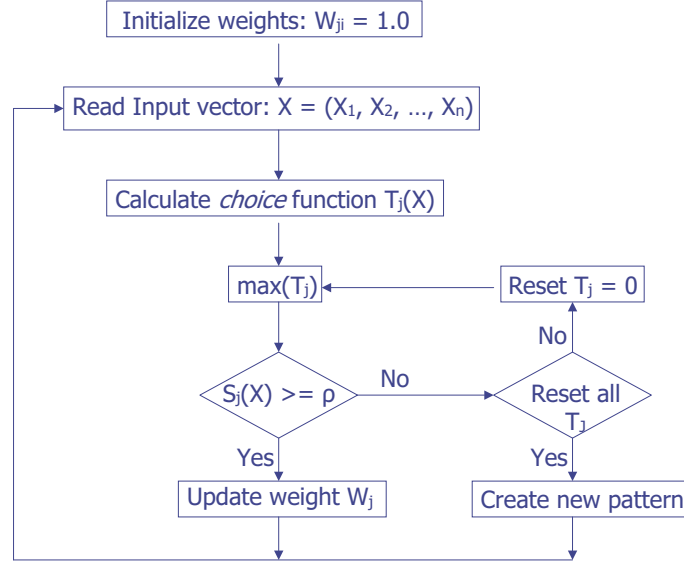


Figure 4.3: Flow graph representation of the Fuzzy ART algorithm.

4.3.2 EFuNN

EFuNN is one of the evolving connectionist systems developed by Kasabov [47] that is capable of modeling evolving processes through incremental, online learning. It has been successfully applied to bio-informatics, speech and image recognition [48]. The original EFuNN has a five-layer structure. Here we only use its first three layers for unsupervised learning (Figure 4.1(b)).

The fuzzy input layer transfers the original input values into membership degrees with a membership function attached to the fuzzy input nodes. The membership function can be triangular, Gaussian, and so on. The number and the type of the membership function can be dynamically modified during the evolving process. In this research we used the triangular membership function.

Unlike Fuzzy ART, EFuNN groups input vectors into patterns based on one distance measure only, the local normalized fuzzy distance between a fuzzy input vector X_f and a weight vector W_j associated with pattern j , which is defined by

$$D(X_f, W_j) = \frac{|X_f - W_j|}{|X_f + W_j|}, \quad (4.8)$$

where $|\cdot|$ denotes the same vector norm defined in Fuzzy ART. The local normalized fuzzy

distance between any two fuzzy membership vectors is within the range of $[0, 1]$.

The rest of the clustering algorithm of EFuNN is very similar to that of Fuzzy ART. When a new input vector X is presented, EFuNN calculates the corresponding fuzzy input vector X_f and evaluates the normalized fuzzy distance between X_f and the existing pattern weight vectors. The activation of the pattern node layer A is then calculated. The activation of a single pattern node j is defined by

$$A(j) = f(D(X_f, W_j)), \quad (4.9)$$

where f can be a simple linear function, for example, $A(j) = 1 - D(X_f, W_j)$. EFuNN finds the closest pattern node j to the fuzzy input vector that has the highest activation value $A(j)$. If $A(j) \geq \rho$, where ρ is the *vigilance* parameter (the original EFuNN paper named it *sensitivity threshold* [47]), the new input is assigned to the j th pattern and the weight vector W_j is updated according to the following vector operation:

$$W_j^{(new)} = W_j^{(old)} + \beta(X_f - W_j^{(old)}), \quad (4.10)$$

where β is the *learning rate*. Otherwise, a new pattern node is created to accommodate the current instance X .

The parameters ρ and β can be static, or they can be self-adjustable while the structure of EFuNN evolves. They can hold the same values for all the patterns, or they can be pattern-specific so that the pattern node that has more instance members will change less when it accommodates a new instance. In our early implementation, all the pattern nodes share the same static ρ and β values.

4.4 Experiments

In this section we describe some experiments. The emphasis of the experiments is on the understanding of how Fuzzy ART and EFuNN based adaptive anomaly detection systems work in practice. One objective of our experiments is to observe the influence of variability of the tunable parameters on the performance of an anomaly detection system. Another objective of the experiments is to compare SVM based static learning and evolving connectionist system based adaptive learning.

4.4.1 Static Learning via Support Vector Machines

The details on support vector machine based anomaly detection are presented in Appendix A. In our experiments, we used SVM to demonstrate the weakness of static learning and the importance of adaptive learning. SVM was employed to learn a model (i.e., support vectors) that fits the training dataset. The model was then tested on the testing dataset without any update (thus it is static learning). SVM is optimal when the data are independent and identically distributed (i.i.d.). If there was concept drift between the training dataset and the testing dataset, SVM would generate classification errors. Adaptive learning can adapt to concept changes incrementally and learn new patterns when new testing instances are presented to the learning system. Therefore the classification accuracy is improved.

In our research, we used LIBSVM (version 2.35) [13], an integrated tool for SVM classification and regression.

4.4.2 Cost Function

To facilitate performance comparison among different methods, we used the cost function:

$$Cost = (1 - hit\ rate) + \gamma * false\ positive\ rate, \quad (4.11)$$

where the hit rate is the rate of detected intrusions (attacks or masquerades), the false positive rate is the probability that a normal instance is classified as *anomalous*, and the parameter γ represents the relative cost difference between a false alarm and a miss. There is no obvious way to determine the value of γ , since the cost of a false alarm as well as the cost of a miss will vary from one environment to another. Here we set the γ value to 6, which was used in [80], while other values are certainly applicable. Varying the tunable parameters' values results in different hit rates and false positive rates, and, subsequently, different cost values. The lower cost, the better performance an intrusion detection system has.

4.4.3 Network Intrusion Detection

We conducted a series of experiments on a subset of the dataset KDD Cup 1999 [22] prepared for network intrusion detection. Many methods have been tested with this popular dataset for supervised intrusion detection. The data labels were usually used for training the learning systems. Our evolving connectionist systems, however, do not rely on the data labels. They build network connection patterns incrementally in an online unsupervised learning mode. Therefore they are not directly comparable to previously proposed supervised learning methods.

The 1999 KDD Cup network traffic data are connection-based. Each data record, described by 7 symbolic attributes and 34 continuous attributes, corresponds to a TCP/IP connection between two IP addresses. In addition, a label is provided indicating whether the record is normal or it belongs one of the four attack types (*Probe*, *DoS*, *U2R* and *R2L*). The symbolic attributes that have two possible values (e.g., *logged_in*) were represented by a binary entry with the value of 0 or 1. For symbolic attributes that have more than two possible categorical values, we used multiple entries to encode them in the vector representation, one entry for each possible value. The entry corresponding to the category value has a value of 1 while the other entries are set to 0. The attribute *service* has 41 types, and we further classified them into $\{http, smtp, ftp, ftp_data, others\}$ to reduce the vector dimensions. The resulting feature vectors have a total of 57 dimensions.

Since different continuous attributes were measured on very different scales, the effect of some attributes might be completely dwarfed by others that have larger scales. Therefore we scaled the attributes to the range of $[0, 1]$ by calculating:

$$X_i = \frac{v_i - \min(v_i)}{\max(v_i) - \min(v_i)}, \quad (4.12)$$

where v_i is the actual value of attribute i , and the maximum and minimum are taken over the whole dataset. However, we are aware that this scaling technique would not work if the maximum and minimum values are not known *a priori*.

We formed a subset of the original dataset consisting of 97277 normal connections and 9199 attacks by randomly sampling. We then conducted two experiments with this subset. The first experiment (*Exp. 1*) was designed to test our evolving connectionist

Table 4.1: Numbers of normal and attack examples in *Exp. 1* and *Exp. 2*.

| <i>Exp. 1</i> | | <i>Exp. 2</i> | | |
|---------------|---------|-----------------|----------------|---------|
| normal | attacks | training normal | testing normal | attacks |
| 97277 | 998 | 38910 | 58367 | 580 |

systems. In the data stream of *Exp. 1*, the attack examples randomly drawn from the 9199 attacks were inserted into the 97277 normal examples with a 1% probability. Fuzzy ART and EFuNN were employed to model the network connections on the fly from an empty set of normal patterns and detect the intrusions in the data stream. For the second experiment (*Exp. 2*), the training dataset and testing dataset were formed to compare the performance between static learning and adaptive learning. The first 40% of the 97277 normal examples were used for training, and the rest for testing. The testing dataset also included attacks interspersed into the normal examples with the probability of 1%. The model learned from the training examples was applied to the testing dataset. The model remained unchanged during the testing process for static learning, while it was updated continuously for adaptive learning methods. Table 4.1 lists the numbers of normal and attack examples in *Exp. 1* and *Exp. 2*.

Effectiveness of Varying Vigilance

The *vigilance* parameter ρ controls the degree of mismatch between new instances and previously learned patterns. The greater the value of *vigilance*, the more similar the instances ought to be in order to be assigned to a pattern. We studied the effect of varying ρ while keeping the values of other parameters fixed. Table 4.2 presents the results when ρ 's value was varied from 0.9 to 0.99 with the data stream of *Exp. 1*. The *learning rate* parameter β was set to 0.1, N_{watch} was 8 and Min_{count} was 4. The false positive rate was calculated as the percentage of normal instances that were labeled *anomalous* out of the 97277 normal examples. Similarly, the hit rate was the percentage of detected attacks (i.e., labeled *anomalous*) out of the 998 attacks.

The results show that the false positive rate increases monotonically as the vigilance threshold is raised. This is due to the fact that more normal instances are classified as

Table 4.2: The performance (false positive rate, hit rate and cost) of Fuzzy ART and EFuNN with the Exp. 1 data stream. Results illustrate the impact of varying ρ on their performance.

| ρ | Fuzzy ART | | | EFuNN | | |
|--------|---------------------|--------------|--------------|---------------------|--------------|--------------|
| | false positive rate | hit rate | cost | false positive rate | hit rate | cost |
| 0.90 | 1.82% | 79.8% | 0.311 | 0.259% | 33.4% | 0.682 |
| 0.91 | 2.07% | 73.6% | 0.389 | 0.340% | 37.2% | 0.649 |
| 0.92 | 2.06% | 66.3% | 0.460 | 0.421% | 39.3% | 0.632 |
| 0.93 | 2.35% | 86.3% | 0.278 | 0.573% | 66.4% | 0.370 |
| 0.94 | 2.31% | 66.9% | 0.469 | 0.823% | 57.4% | 0.475 |
| 0.95 | 3.13% | 66.6% | 0.521 | 1.29% | 74.9% | 0.328 |
| 0.96 | 3.33% | 64.4% | 0.556 | 1.97% | 90.0% | 0.218 |
| 0.97 | 4.42% | 89.7% | 0.369 | 3.30% | 91.7% | 0.281 |
| 0.98 | 5.81% | 93.2% | 0.417 | 6.99% | 98.7% | 0.433 |
| 0.99 | 8.84% | 98.1% | 0.549 | 18.3% | 99.6% | 1.10 |

uncertain and then *anomalous* when the value of ρ increases. Meanwhile, it is interesting to note that the hit rate oscillates at lower ρ values, and then approaches to 100% as ρ is raised nearer to 1.0. Ideally, the hit rate should increase monotonically as well. Its oscillation may suggest the abnormality of the data. The cost of Fuzzy ART reaches the lowest value at $\rho = 0.93$ with a false positive rate of 2.35% and hit rate of 86.3%. For EFuNN, the lowest cost is obtained at $\rho = 0.96$ while the hit rate is 90% and the false positive rate is as low as 1.97%.

Effectiveness of Varying Learning Rate

The *learning rate* parameter β determines how fast the system should adapt to new instances in order to accommodate them. A higher value of β places more weight to the new instance when it is assigned to a pattern and less weight to existing members of the pattern. We evaluated the performance of Fuzzy ART and EFuNN with the *Exp. 1* data stream by widely varying the *learning rate*. The results are described in Table 4.3. The *vigilance* parameter was set to 0.93 for Fuzzy ART and 0.96 for EFuNN respectively since they provided the lowest cost when the effectiveness of varying vigilance was studied. N_{watch} was set to 8 and Min_{count} was 4.

It is interesting to note that for the *Exp. 1* dataset, $\beta = 0.1$ appears to be the

Table 4.3: The performance of Fuzzy ART and EFuNN with the Exp. 1 data stream. Results illustrate the impact of varying β on their performance.

| β | Fuzzy ART | | | EFuNN | | |
|---------|---------------------|--------------|--------------|---------------------|--------------|--------------|
| | false positive rate | hit rate | cost | false positive rate | hit rate | cost |
| 0.001 | 0.256% | 24.9% | 0.766 | 2.34% | 76.4% | 0.377 |
| 0.01 | 0.675% | 54.7% | 0.493 | 2.20% | 88.3% | 0.249 |
| 0.1 | 2.35% | 86.3% | 0.278 | 1.97% | 90.0% | 0.218 |
| 0.3 | 3.17% | 68.6% | 0.504 | 1.69% | 77.3% | 0.329 |
| 0.5 | 3.44% | 71.0% | 0.496 | 1.64% | 72.7% | 0.371 |
| 0.7 | 3.58% | 70.1% | 0.513 | 1.60% | 77.4% | 0.322 |
| 0.9 | 3.53% | 79.0% | 0.369 | 1.47% | 76.1% | 0.327 |
| 1.0 | 3.23% | 67.8% | 0.515 | 1.55% | 74.9% | 0.343 |

best choice for both Fuzzy ART and EFuNN in terms of the cost. Higher β values provide relatively stable false positive rates and hit rates. For Fuzzy ART, lower β values ($\beta = 0.01$ or 0.001) cause much lower false positive rates as well as lower hit rates. For EFuNN, however, the false positive rate gets even higher at lower β values while the hit rate declines slightly.

Effectiveness of Varying N_{watch} and Min_{count}

N_{watch} and Min_{count} are two other important parameters for an adaptive anomaly detection system. N_{watch} represents the delay the system will experience before it evaluates a newly created *uncertain* pattern. If it is too long, there is a risk that an anomalous instance can not be handled in a timely manner. If it is too short, large amount of false alarms may be generated. Min_{count} is the minimum number of members that an *uncertain* pattern ought to have before it is changed to *normal*. We empirically studied the effect of varying N_{watch} and Min_{count} on the performance of Fuzzy ART and EFuNN. Different values of N_{watch} and Min_{count} and the corresponding results are described in Table 4.4. The *vigilance* parameter was set to 0.93 for Fuzzy ART and 0.96 for EFuNN, and the *learning rate* was 0.1 for both of them.

The results show that $N_{watch} = 8$ and $Min_{count} = 4$ is a better choice than others for Fuzzy ART as it provides the lowest cost. Similarly, $N_{watch} = 4$ and $Min_{count} = 2$ gives the best performance for EFuNN. The hit rate of EFuNN is higher and more stable than

Table 4.4: The performance of Fuzzy ART and EFuNN with the Exp. 1 data stream. Results illustrate the impact of varying N_{watch} and Min_{count} on their performance.

| N_{watch} | Min_{count} | Fuzzy ART | | | EFuNN | | |
|-------------|---------------|---------------------|--------------|--------------|---------------------|--------------|--------------|
| | | false positive rate | hit rate | cost | false positive rate | hit rate | cost |
| 4 | 2 | 1.71% | 74.2% | 0.360 | 1.53% | 88.9% | 0.203 |
| 8 | 4 | 2.35% | 86.3% | 0.278 | 1.97% | 90.0% | 0.218 |
| 12 | 4 | 1.77% | 77.1% | 0.335 | 1.65% | 89.4% | 0.205 |
| 12 | 6 | 4.03% | 70.0% | 0.542 | 3.66% | 92.9% | 0.291 |
| 12 | 8 | 6.04% | 95.4% | 0.408 | 5.04% | 95.7% | 0.345 |
| 16 | 6 | 2.97% | 61.1% | 0.567 | 2.95% | 92.9% | 0.248 |
| 16 | 8 | 4.95% | 72.0% | 0.577 | 4.19% | 94.3% | 0.309 |
| 16 | 10 | 6.77% | 84.2% | 0.564 | 6.41% | 96.3% | 0.422 |

that of Fuzzy ART as the values of N_{watch} and Min_{count} change. It indicates that given the distance measure of EFuNN, the attacks are more distinguishable among the normal instances.

Static Learning vs. Adaptive Learning

We compared Fuzzy ART and EFuNN with SVM using the *Exp. 2* datasets. During the training process, Fuzzy ART and EFuNN assumed every pattern was normal and no instance was discarded. During the testing process, however, the task of Fuzzy ART and EFuNN became twofold: evolving their structure to accommodate new patterns and detecting anomalous instances. For simplicity, we set N_{watch} to 8 and Min_{count} to 4. We then varied the *vigilance* parameter's value from 0.9 to 0.99, and the *learning rate*'s value from 0.01 to 0.9. The parameter settings that provide the lowest cost for Fuzzy ART and EFuNN are shown in Table 4.5.

The SVM model learned from the one-class training dataset was applied to the testing dataset. Common types of kernel functions used in SVM include linear, radial basis and polynomial functions. In our experiments, we found the radial basis kernel performed better than other kernel functions for one-class learning. The parameter ν [93], which controls the number of support vectors and errors, was determined by cross validation with the training data sets.

Table 4.5: The performance of SVM, Fuzzy ART and EFuNN in *Exp. 2*.

| | SVM | Fuzzy ART | EFuNN |
|---------------------|-------|-----------|--------|
| ρ | | 0.93 | 0.96 |
| β | | 0.2 | 0.01 |
| false positive rate | 12.4% | 2.98% | 0.884% |
| hit rate | 90.7% | 94.0% | 85.0% |
| cost | 0.836 | 0.239 | 0.203 |

Table 4.5 compares the performance of SVM, Fuzzy ART and EFuNN. SVM was able to detect 90% of the attacks in the testing dataset. However, the false positive rate was as high as 12.4%, which indicates the presence of concept drift between the training dataset and the testing dataset. Compared to SVM, Fuzzy ART and EFuNN generated significantly less false alarms. Fuzzy ART was the best in terms of hit rate, whereas EFuNN gave the lowest cost.

4.4.4 Masquerade Detection with User Profiling Data

Dataset Descriptions

We obtained a set of Windows NT user profiling data from an NSA officer. The data was collected for 20 users on 21 different hosts in a real-world government agency environment (a single user might have worked on multiple hosts). During the raw data collection, a tool was developed to query the Windows NT process table periodically (2 to 3 times per second) and collect all the process information of each user’s login session. Processes that were not related to user identification were filtered out during the pre-processing. The processes that correspond to the windows the user activated are of special interest to us because they represent the programs the user was running. The accumulated CPU time was calculated for each of these window-associated processes from a user’s login to logout, which reflects the workload the user performed during this login session. For processes that have the same process name, their CPU times were added together. Then a CPU time vector can be formed for each login session, where the value of each entry is the percentage of CPU time consumed by a unique process during this login session. There are 105 processes contained in this dataset, for example, *netscape*, *explorer*, *outlook*, *msoffice*

and so on, while each individual user has his or her own process “vocabulary.”

In addition to the CPU information, we also included the login time in the input vectors. A user’s login time was categorized as *early morning* (before 7 AM), *morning* (between 7 AM and 12 PM), *afternoon* (between 12 PM and 6 PM) or *evening* (later than 6 PM). Since the login time has four possible values, we have four entries in the input vectors corresponding to the login time. Therefore each login session is encoded as a vector that contains CPU time percentages consumed by the user as well as four additional entries that represent the user’s login time.

We selected the 7 users that have the most login sessions to serve as our masquerade targets. We then used the remaining 13 users as masqueraders and inserted their data into the data of the 7 users. Two experiments were conducted with the dataset, similar to the ones with the KDD Cup 1999 dataset. The first experiment (*Exp. 3*) was designed to test our evolving connectionist systems. The masquerade examples randomly drawn from the 13 users were embedded into the 7 users’ login sessions with a 3% probability. In the second experiment (*Exp. 4*), in order to compare the performance of SVM, Fuzzy ART and EFuNN, the 7 normal users’ login sessions were split into two parts. The first half of the login sessions were used for training the learning systems, and the remaining for testing. Then the masquerade examples were inserted in the testing datasets with a probability of 6%. The values of the masquerade probability were chosen so that there was at least one masquerade example in each user’s testing dataset for both experiments. Table 4.6 shows the numbers of the 7 users’ login sessions and the corresponding masquerade examples for each experiment. The user IDs are user identification numbers inherited from the original dataset. We built learning models for each of the 7 individual users to see if they could identify the masquerade examples hiding in their testing datasets.

Results

For Fuzzy ART and EFuNN, when a testing instance is labeled *anomalous* and a *Level 2 alarm* is generated, it is either a true positive (i.e., a hit) if the instance represents a masquerade example, or a false positive if the instance is the user’s own login session. We varied *vigilance* ρ ’s value from 0.89 to 0.99 and *learning rate* β ’s values from 0.1 to 1.0,

Table 4.6: Number of login sessions and the masquerade examples.

| UserID | <i>Exp. 3</i> | | <i>Exp. 4</i> | | |
|--------|---------------|-------------|---------------|-----------------------|-------------|
| | self sessions | masquerades | training | testing self sessions | masquerades |
| 1 | 184 | 7 | 92 | 92 | 7 |
| 2 | 54 | 1 | 27 | 27 | 2 |
| 4 | 45 | 2 | 22 | 23 | 2 |
| 6 | 55 | 2 | 27 | 28 | 3 |
| 7 | 50 | 2 | 25 | 25 | 2 |
| 14 | 58 | 2 | 29 | 29 | 3 |
| 19 | 87 | 6 | 43 | 44 | 6 |
| Total | 527 | 22 | 259 | 268 | 25 |

Table 4.7: The performance of Fuzzy ART and EFuNN in *Exp. 3*. $N_{watch} = 3$, $Min_{count} = 2$.

| UserID | Fuzzy ART | | | | EFuNN | | | |
|---------|--|---------|-----------------|------|--|---------|-----------------|------|
| | ρ | β | false positives | hits | ρ | β | false positives | hits |
| 1 | 0.95 | 0.4 | 5 | 3 | 0.94 | 0.2 | 7 | 5 |
| 2 | 0.92 | 0.8 | 6 | 1 | 0.91 | 0.8 | 5 | 1 |
| 4 | 0.90 | 0.4 | 1 | 1 | 0.89 | 0.2 | 2 | 2 |
| 6 | 0.91 | 0.4 | 1 | 2 | 0.90 | 0.4 | 0 | 2 |
| 7 | 0.90 | 0.6 | 4 | 1 | 0.90 | 0.2 | 0 | 1 |
| 14 | 0.93 | 0.8 | 7 | 1 | 0.90 | 0.4 | 2 | 1 |
| 19 | 0.91 | 1.0 | 0 | 6 | 0.89 | 1.0 | 2 | 6 |
| Total | | | 24 | 15 | | | 18 | 18 |
| Overall | false positive rate = $24/527 = 4.55\%$ hit rate = $15/22 = 68.2\%$ Cost = 0.591 | | | | false positive rate = $18/527 = 3.42\%$ hit rate = $18/22 = 81.8\%$ Cost = 0.387 | | | |

respectively. The choice of N_{watch} and Min_{count} depends on the profiled individual user. For simplicity, we set N_{watch} to 3 and Min_{count} to 2 for all users, which were found to be an acceptable compromise. For each of the 7 users, we report the parameter settings (ρ and β) that give the lowest cost in Table 4.7 for *Exp. 3* and Table 4.8 for *Exp. 4*.

In *Exp. 3*, both Fuzzy ART and EFuNN were able to model user behavior starting from an empty set of normal patterns and still recognize the majority of the masquerade instances, while the false positive rate was under 5%. EFuNN performed slightly better than Fuzzy ART because EFuNN provided higher hit rate and lower false positive rate and thus lower overall cost.

Table 4.8: The performance of SVM, Fuzzy ART and EFuNN in *Exp. 4*.

| UserID | SVM | | Fuzzy ART | | | | EFuNN | | | |
|---------|---|------|--|---------|-----------------|------|---|---------|-----------------|------|
| | false positives | hits | ρ | β | false positives | hits | ρ | β | false positives | hits |
| 1 | 11 | 6 | 0.92 | 1.0 | 6 | 7 | 0.96 | 0.1 | 2 | 6 |
| 2 | 7 | 1 | 0.91 | 1.0 | 0 | 1 | 0.90 | 1.0 | 0 | 1 |
| 4 | 3 | 2 | 0.91 | 0.6 | 0 | 2 | 0.91 | 0.8 | 0 | 1 |
| 6 | 3 | 2 | 0.89 | 0.8 | 1 | 2 | 0.94 | 0.6 | 1 | 3 |
| 7 | 8 | 2 | 0.89 | 0.6 | 1 | 1 | 0.90 | 0.6 | 1 | 1 |
| 14 | 6 | 1 | 0.91 | 1.0 | 2 | 1 | 0.91 | 0.4 | 3 | 1 |
| 19 | 4 | 6 | 0.91 | 0.4 | 0 | 6 | 0.92 | 0.2 | 0 | 6 |
| Total | 42 | 20 | | | 10 | 20 | | | 7 | 19 |
| Overall | false positive rate = $42/268 = 15.7\%$ hit rate = $20/25 = 80.0\%$ Cost = 1.14 | | false positive rate = $10/268 = 3.73\%$ hit rate = $20/25 = 80.0\%$ Cost = 0.424 | | | | false positive rate = $7/268 = 2.61\%$ hit rate = $19/25 = 76.0\%$ Cost = 0.397 | | | |

Table 4.8 shows the performance comparison of static learning with SVM and adaptive learning with Fuzzy ART and EFuNN in *Exp. 4*. The parameter ν of SVM was again determined by cross validation with the training data sets. SVM generated 42 false alarms (15.7% false positive rate) due to the concept drift between the training dataset and the testing dataset. The false positive rates of Fuzzy ART and EFuNN were significantly less because they were able to adapt to user behavior changes incrementally, while the hit rates were comparable to that of SVM. Therefore the overall cost of Fuzzy ART and EFuNN was largely reduced.

4.5 Discussion

Our approach assumes that the number of normal instances vastly outnumbers the number of anomalies, and the anomalous activities appear as outliers in the data. This approach would miss the attacks or masquerades if the underlying assumptions do not hold. For example, some DoS attacks would not be identified by our adaptive anomaly detection systems. Nevertheless, our anomaly detection framework can be easily extended to incorporate signature detection. Previously learned patterns can be labeled in such a way that certain patterns may generate an alert no matter how frequently they are observed, while other patterns do not trigger an alarm even if they are rarely seen [103].

With our adaptive anomaly detection framework, it is possible that one can deliberately cover his malicious activities by slowly changing his behavior patterns without triggering a *level2 alarm*. However, a *level1 alarm* is issued whenever a new pattern is being formed. It is then the security analyst's responsibility to identify the user's intent in order to distinguish malicious from non-malicious anomalies, which is beyond the scope of this paper. We also note that in case a continuing abnormal activity occurs, large amount of *level1 alarms* may be raised and the security analyst can still get overwhelmed.

4.6 Summary

This chapter has presented a new adaptive anomaly detection framework through the use of evolving connectionist systems. A subject's normal behavior is learned in the online unsupervised mode. The performance of two adaptive anomaly detection systems, based on Fuzzy ART and EFuNN, was empirically tested with the KDD Cup 1999 network data and the user profiling data. The experiments have shown that our adaptive anomaly detection systems are able to adapt to user or network behavior changes while still recognizing anomalous activities. Compared to the SVM based static learning, the adaptive anomaly detection methods can significantly reduce the false alarms.

In order to make an adaptive anomaly detection system scalable, it might be necessary to prune or aggregate pattern nodes as the system evolves, which is a significant issue for future work. Other issues include exploring automated determination of the parameters and comparing more evolving connectionist systems, such as evolving self-organizing maps.

Chapter 5

Estimating Generalization Performance and Training Size Requirements

5.1 Introduction

In anomaly detection, the following questions are fundamental:

- For a given number of samples, how significant and reliable is the performance (in terms of classification error) of an anomaly detector? In other words, to what degree can the normal behavior model learned from the finite training set be applied to yet unseen data?
- How much training is sufficient in order to achieve certain performance? How will the accuracy of the anomaly detector improve when trained with additional samples?

These two questions arise in almost every machine learning task. In machine learning, the problem of estimating how well a learning method can generalize and predict future data based on the given training examples is called generalization performance analysis, or statistical stability analysis. Statistical learning theory estimates generalization bounds through the uniform convergence analysis based on the use of VC dimension [104], or more

recently, Rademacher Complexity [53][96]. While the convergence estimation provides valuable insights in model selection and parameter tuning, the theoretical generalization bounds or error estimates tend to be rather loose and therefore are of little practical use. Other empirical methods, such as cross-validation and bootstrapping, usually give good error estimates, but are computationally expensive [45].

The closely related problem is how the error rate might decrease as more training data becomes available. It amounts to developing a model to compute how fast the learning method “learns” as a function of training dataset size. A natural approach is to build a *learning curve* for a given learning method and dataset to fit the empirical error rate as a function of training size. In particular, the inverse-power-law learning curve, where the error rate drops according to an inverse power law, appears to be universal and is observed for many classifiers and types of datasets [14]. In addition to its empirical prevalence, the inverse-power-law model, in some cases, can be also derived with a statistical mechanics approach to learning (e.g., [20] [90]).

This chapter addresses these two issues with a particular learning method - Support Vector Machines (SVM). We introduce an efficient generalization performance estimate tailored to SVM-based anomaly detection. In anomaly detection, due to the lack of anomalous examples, it is not possible to provide a reliable estimate of the true positive rate, i.e., probability of fault detection (also known as hit rate). However, the false positive rate, i.e., probability of false alarms, is bounded under certain assumptions and can be estimated efficiently without re-sampling and retraining. We demonstrate that with such estimate, it is then possible to select different models and different learning parameters, and identify the changes of normal behavior patterns, i.e., the *concept drift* problem. Furthermore, we fit inverse power-law models to construct empirical learning curves for anomaly detection. It provides the basis for estimating training size requirements and deciding whether a certain minimum performance can be achieved or not. We test our methodology on a variety of artificial and real-world datasets.

5.2 Error estimate of SVM-based anomaly detection

Virtually all machine learning research assumes that the training sample is drawn from a stationary data source, that is, the distribution of data points and the patterns to be learned are not changing with time. SVM is no exception. Given a set of training examples generated independently and identically distributed from an unknown distribution, the generalization error bound for SVM-based anomaly detection can be derived through the use of VC dimension [93] or Rademacher Complexity [96]. Although these error bounds set a theoretical limit on the probability that a new test object drawn from the same underlying distribution lies outside the optimal hypersphere, they are very loose and thus not useful in practice. Joachims [45] proposed the so-called ξ_α -estimator for the generalization performance of a standard SVM classifier. This is still a loose estimate and not directly applicable to SVM-based anomaly detection.

Here we introduce a simple and effective error estimate tailored to the SVM-based anomaly detection problem, which was first briefly explored in [99]. The idea is based on the leave-one-out (loo) estimation. The loo estimator excludes one sample from a training dataset of size N , constructs a classifier using the remaining $N - 1$ samples, and then tests the classifier on the sample left out. This procedure is repeated for all training examples. The total number of misclassifications divided by N is the loo estimate of the generalization error. Lunts and Brailovsky [73] showed that the loo estimator gives an almost unbiased estimate of the expected error. However, the computational cost of the loo estimator is high as the learner is invoked N times. This is prohibitively expensive for large N or complicated learning methods. To reduce the running time, a common practice is to randomly partition the training set into k parts and exclude one of the k samples each time, a scheme known as k -fold cross validation. Cross validation is probably the most popular method for estimating the generalization performance of a classifier, although it has a larger bias than loo. Loo, essentially a N -fold cross validation, is approximately unbiased but has larger variance.

The details on SVM-based anomaly detection are described in Appendix A. The key idea of the error estimate for SVM-based anomaly detection is to count the number of training examples that can cause an error in leave-one-out testing. For a given training

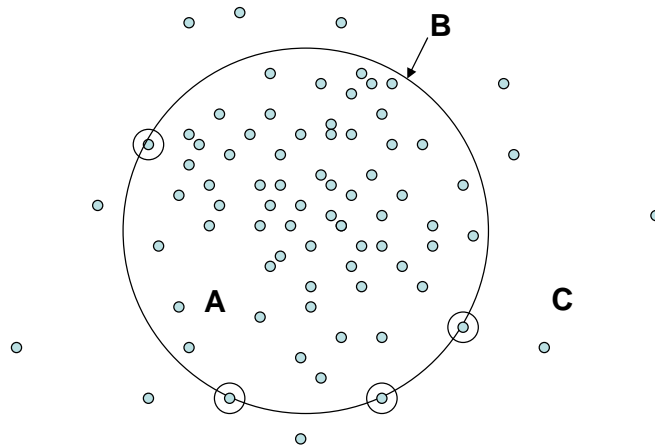


Figure 5.1: A two-dimensional sphere (solid line) containing most of the data. Enclosed in a circle are four support vectors on the boundary.

set S of N examples, the resulting hypersphere divides the N objects into three distinct regions as shown in Figure 5.1:

- Region A, the interior of the hypersphere. For data points lying in this region, $\alpha_i = 0$.
- Region B, the boundary of the hypersphere. Data points on the hypersphere are unbounded support vectors with $0 < \alpha_i < 1/\nu N$.
- Region C, the space outside the hypersphere. Data points lying in this region are bounded support vectors with the α_i value of $1/\nu N$.

Before we examine data points in these regions in turn for the leave-one-out estimation, the concept of *essential support vectors* [104] needs to be introduced. Although the optimal hypersphere is unique, the expansion of the center of the hypersphere $\mathbf{c} = \sum \alpha_i \mathbf{x}_i$ is not unique. There might be more support vectors than are necessary. The support vectors that appear in all possible expansions are essential support vectors. While it is not trivial to identify all the essential support vectors for a practical problem, they play an important role in determining the loo error. Here we assume N is large enough and $\nu N \simeq \nu(N - 1)$ so that the upper bound on the number of outliers and the lower bound on the number of support vectors remain the same when one training example is left out.

1. Region A: since the internal points have zero α_i value, they do not make any contribution in defining the center and radius of the optimal hypersphere. When a point lying inside the hypersphere is left out of training, the same hypersphere solution will be obtained when training with the remaining $N - 1$ examples. During testing this object will therefore still fall inside the hypersphere. Thus it does not produce an error for the loo estimation.
2. Region B: When a non-essential support vector on the boundary is excluded during the loo process, the same hypersphere is achieved using the remaining support vectors in the expansion. Therefore it will be recognized correctly and no loo error will be generated. When an essential support vector on the boundary is left out, however, a smaller hypersphere is obtained. This essential support vector, lying outside the new hypersphere, will cause a loo error.
3. Region C: the bounded support vectors are already outside the hypersphere. When they are excluded from training, no matter whether they are essential support vectors or not, they will fall outside the hypersphere again and thus cause a loo error.

In summary, the leave-one-out error is the sum of the number of essential support vectors on the boundary (N_{eSV}) and the number of bounded support vectors (N_{bSV}) divided by N :

$$E_{loo} = \frac{N_{eSV} + N_{bSV}}{N} \leq \frac{N_{SV}}{N},$$

where N_{SV} is the number of all support vectors. The inequality holds when not all support vectors on the boundary are essential support vectors. Since the loo error is the unbiased estimate of the expected error, we have:

$$E[Error] \leq \frac{N_{SV}}{N}.$$

The derivation above shows that the fraction of support vectors is a clear indication of the generalization performance of the SVM-based anomaly detection. This error estimate does not require expensive re-sampling and retraining and is available immediately after the hypersphere is found.

The generalization error can also be interpreted as an estimate of the false positive rate. Therefore, for a given training data set of normal examples, the expected false positive rate for future unseen data is no more than the fraction of support vectors, provided that the testing data is generated from the same underlying distribution as the training data. The error estimate can guide the model selection and learning parameter tuning. In addition, it can be used to detect concept drift when there is a large discrepancy between the expected false positive rate and the actual false positive rate.

5.3 learning curve fitting

The generalization performance analysis addresses the question of how confident we are of our error estimate for a given training set. Yet it does not answer the question of how the error rate drops as the training size increases. Learning curves address this latter problem. They estimate the empirical error rate as a function of training set size for a given classifier and dataset. A learning curve is usually well characterized by an inverse power law:

$$e(n) = an^{-\alpha} + b.$$

Where $e(n)$ is the expected error rate for training size n , a is the learning rate, α is the decay rate, and the Bayes error b is the minimum error rate achievable [19]. The parameter values of a , α and b will change for a particular learning method and dataset. According to this model, as the training size increases, the error rate will asymptotically approach b . The inverse-power-law model has been successfully used to fit learning curves in many applications (e.g., [85]).

Anomaly detectors usually suffer from high false positive rate when normal behavior varies widely. For a system that has less regular normal behavior, more training data is required in order to reduce false alarms and achieve certain performance. The inverse-power-law learning curve fitting provides the basis for estimating training size requirements and determining whether a minimum false positive rate can be obtained.

Fitting a learning curve amounts to the following optimization problem:

$$\min_{a, \alpha, b} \sum_{j=1}^M [an_j^{-\alpha} + b - e(n_j)]^2,$$

subject to $a, \alpha, b \geq 0$. It can be solved with standard nonlinear programming techniques. $e(n_j)$ can be estimated using the efficient technique shown in Section III or more complicated estimates such as cross validation.

5.4 Experiments

In this section, we evaluate our error estimate and fit learning curves for artificial and real-world datasets. In our research, we used LIBSVM (version 2.6) [13], an integrated tool for SVM classification and regression as well as one-class SVM.

5.4.1 Artificial data

We used pseudo random numbers to generate normal Gaussian datasets and a mixture of three outlier groups. The training set consists of 30000 three-dimensional data points, generated with a Gaussian probability density $p(y|\mu, \Lambda)$, where $\mu = (0, 0, 0)$ and

$$\Lambda = \begin{pmatrix} 1.2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

The testing dataset contains 30000 normal instances generated from the same distribution and 10000 outliers generated by a Gaussian mixture:

$$p(y) = c_1 p(y|\mu_1, \Lambda_1) + c_2 p(y|\mu_2, \Lambda_2) + c_3 p(y|\mu_3, \Lambda_3),$$

where $c_1 = 0.3$, $c_2 = 0.3$, $c_3 = 0.4$, $\mu_1 = (3.5, 0, 0)$, $\mu_2 = (3.5, 2, 0)$, $\mu_3 = (9, -3, 0)$, and $\Lambda_1 = \Lambda_2 = \Lambda_3 = 0.2I$ (I is the identity matrix).

Gaussian kernel function was used for this dataset. γ , the parameter of the Gaussian kernel, was set to 0.001. We varied ν 's value to observe the effectiveness of the error estimate as well as the influence of ν on the performance of the anomaly detector. Figure 5.2(a) compares the fraction of support vectors as the error estimate, the 10-fold cross validation error on the training set and the false positive rate on the testing set when varying

ν from 0.001 to 0.1. The results show that these three quantities coincide at every different value of ν . As ν increases, the volume of the hypersphere shrinks in order to accommodate more outliers in the training set. We can see that the error rate gets higher for both the training and testing sets. This is not surprising as the training and testing datasets are generated from the same underlying distribution. Meanwhile, Figure 5.2(b) shows the true positive rate on the testing set, i.e., the outlier detection rate, improves, as expected, when ν value increases and the hypersphere gets smaller.

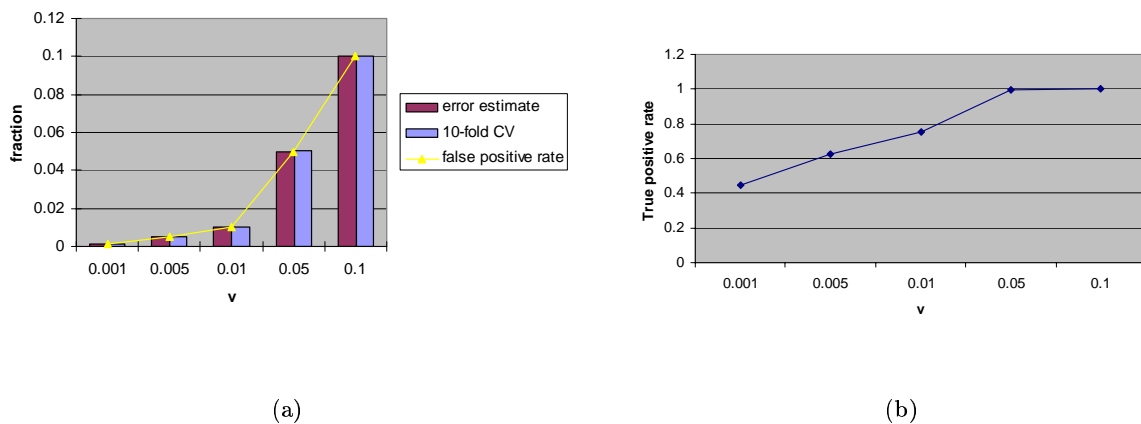


Figure 5.2: Experimental results for various ν values on the artificial data. (a) Comparison of error estimate, 10-fold cross validation error and false positive rate on the testing set. (b) True positive rate vs. ν .

5.4.2 Masquerade data

We chose a UNIX command dataset that is often used for masquerade studies [94] [80] [88]. The data, available at <http://www.schonlau.net/>, consists of 50 users' UNIX command sequences. Each individual user's data has 150 blocks of 100 commands each. The first 50 blocks are noise-free and used for training. The testing set, starting from block 51 to block 150, may contain command blocks from masqueraders. Many methods have been tested on this popular dataset for masquerade detection. The emphasis of our experiments, however, is on demonstrating the effectiveness of the error estimate.

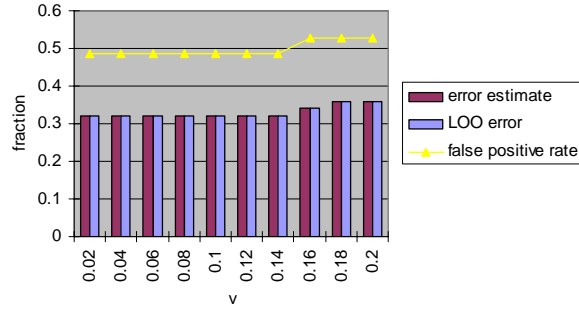
We selected the four users that have the most masquerade blocks: User 9, User 24, User 42 and User 43. The k -spectrum kernel [66] was used to map the command sequences

into the feature space (with $k = 2$). Figure 5.3 compares the error estimate, i.e., the fraction of support vectors, the leave-one-out error and false positive rate on the testing data when ν varies from 0.02 to 0.2. We can see that the error estimate is equal to the loo error in most cases, and only slightly larger than the loo error for the rest. This shows that the fraction of support vectors effectively predicts the loo error without expensive retraining. As ν increases, the error estimate and the loo error get higher. The similar pattern can be seen with the false positive rate on the testing set. It is interesting to note, however, that the testing false positive rate is much higher than the training error for User 9, User 42 and User 43. The large discrepancy indicates the drift of user command patterns. This suggests that the error estimate can also be used to detect concept drift in anomaly detection. The true positive rate, i.e., the masquerade detection rate, is always 100% for these four users.

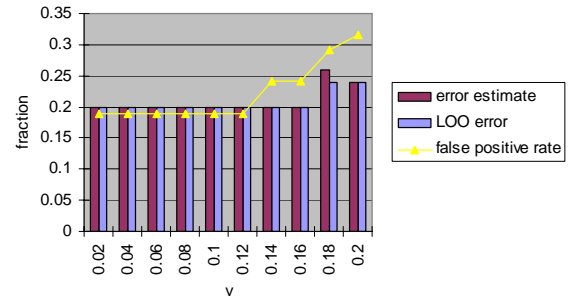
5.4.3 KDD data

We also conducted experiments on a subset of the KDD Cup 1999 [22] data prepared for network intrusion detection. The 1999 KDD Cup network traffic data are connection-based. Each data record, described by 7 symbolic attributes and 34 continuous attributes, corresponds to a TCP/IP connection between two IP addresses. In addition, a label is provided indicating whether the record is normal or it belongs one of the four attack types (*Probe*, *DoS*, *U2R* and *R2L*). The symbolic attributes that have two possible values (e.g., *logged_in*) were represented by a binary entry with the value of 0 or 1. For symbolic attributes that have more than two possible categorical values, we used multiple entries to encode them in the vector representation, one entry for each possible value. The entry corresponding to the category value has a value of 1 while the other entries are set to 0. The attribute *service* has 41 types, and we further classified them into $\{http, smtp, ftp, ftp_data, others\}$ to reduce the vector dimensions. The resulting feature vectors have a total of 57 dimensions. Since different continuous attributes were measured on very different scales, the effect of some attributes might be completely dwarfed by others that have larger scales. Therefore we scaled the attributes to the range of $[0, 1]$.

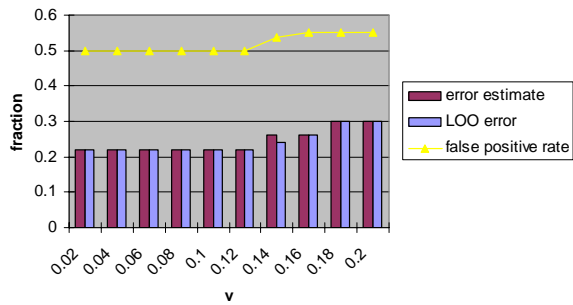
We formed a subset of the original dataset consisting of 97277 normal connections and 9199 attacks by randomly sampling. We used 50% of the normal instances for training.



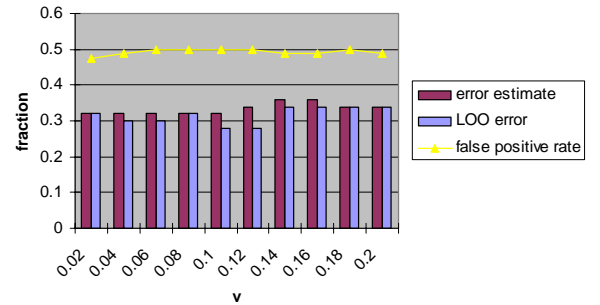
(a)



(b)



(c)



(d)

Figure 5.3: Comparison of error estimate, leave-one-out error and false positive rate on testing data. (a) User 9; (b) User 24; (c) User 42; (d) User 43.

The testing set consists of the remaining normal examples and all the attacks.

Gaussian kernel function was used for this dataset. We varied the values of ν and γ to observe their influence on the performance. Table 5.1 and Figure 5.4 present the results when γ varies from 0.001 to 0.1 and ν is set to 0.001. As can be seen in Figure 5.4(a), the error estimate and the 5-fold cross validation error are close to each other with minor difference. This is reasonable since the 5-fold cross validation is subject to large variance. Note that γ represents the inverse of the Gaussian kernel width. With increasing γ , the Gaussian kernel width gets smaller and the boundary of the hypersphere becomes less smooth [99]. As a result, we can see a slight increase of the number of support vectors and thus the error estimate. Meanwhile, the false negative rate, i.e., the probability of missing intrusions, drops slightly. We can also see that the false positive rate on the testing set is one magnitude larger than the expected error on the training set, indicating significant concept drift between the training and testing datasets.

Table 5.1: Experimental results for various γ values on the KDD data ($\nu = 0.001$).

| γ | N_{SV} | error estimate | 5-fold CV error | false positive rate | false negative rate |
|----------|----------|----------------|-----------------|---------------------|---------------------|
| 0.0001 | 51 | 0.105% | 0.15% | 8.543% | 18.643% |
| 0.001 | 52 | 0.107% | 0.115% | 9.145% | 17.38% |
| 0.01 | 53 | 0.109% | 0.105% | 9.38% | 16.13% |
| 0.1 | 60 | 0.123% | 0.152% | 8.62% | 14.523% |

We also varied ν 's value from 0.0001 to 0.1 while γ was set to 0.01. As shown in Table 5.2, ν has a much stronger influence on the performance of the anomaly detector than γ . As ν increases, the intrusion detection rate improves significantly at the cost of increasing false alarms. However, the error estimate remains a close estimate of the cross validation error.

Table 5.2: Experimental results for various ν values on the KDD data ($\gamma = 0.01$).

| ν | N_{SV} | error estimate | 5-fold CV error | false positive rate | false negative |
|--------|----------|----------------|-----------------|---------------------|----------------|
| 0.0001 | 12 | 0.02467% | 0.0247% | 1.057% | 30.49% |
| 0.001 | 53 | 0.109% | 0.105% | 9.38% | 16.13% |
| 0.01 | 490 | 1.007% | 0.995% | 14.08% | 10.26% |
| 0.1 | 4866 | 10% | 9.25% | 27.8% | 0.75% |

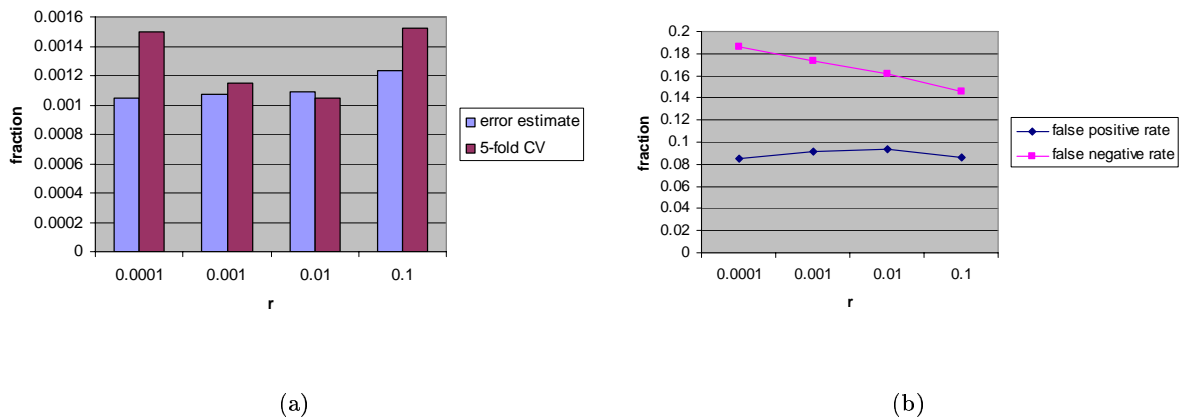


Figure 5.4: Experimental results for various γ values on the KDD data ($\nu = 0.001$). (a) Comparison of error estimate and 5-fold cross validation error on the training set. (b) False positive rate and false negative rate on the testing set.

5.4.4 Learning curve fitting

We used the same masquerade dataset to fit learning curves. We randomly picked two users that do not have masquerade command blocks: User 5 and User 6. The training size was increased gradually from 50 to 150. For each training size, ν 's value was varied from 0.01 to 0.2 and the value that gave the lowest error estimate was chosen. The spectrum kernel was used again with $k = 2$. Table 5.3 presents the error estimates for different training sizes.

Table 5.3: Errors for various training sizes on the masquerade data.

| training size | User 5 error estimate | User 6 error estimate |
|---------------|-----------------------|-----------------------|
| 50 | 26% | 10% |
| 70 | 18.57% | 8.57% |
| 90 | 15.56% | 4.44% |
| 110 | 14.54% | 5.45% |
| 130 | 12.31% | 4.62% |
| 150 | 10.67% | 4% |

Figure 5.5 shows the learning curves for User 5 and User 6. The estimated inverse-power-law models for User 5 and User 6 are:

$$User5 : Error(n) = 5.3642n^{-0.7801} + 0.0003,$$

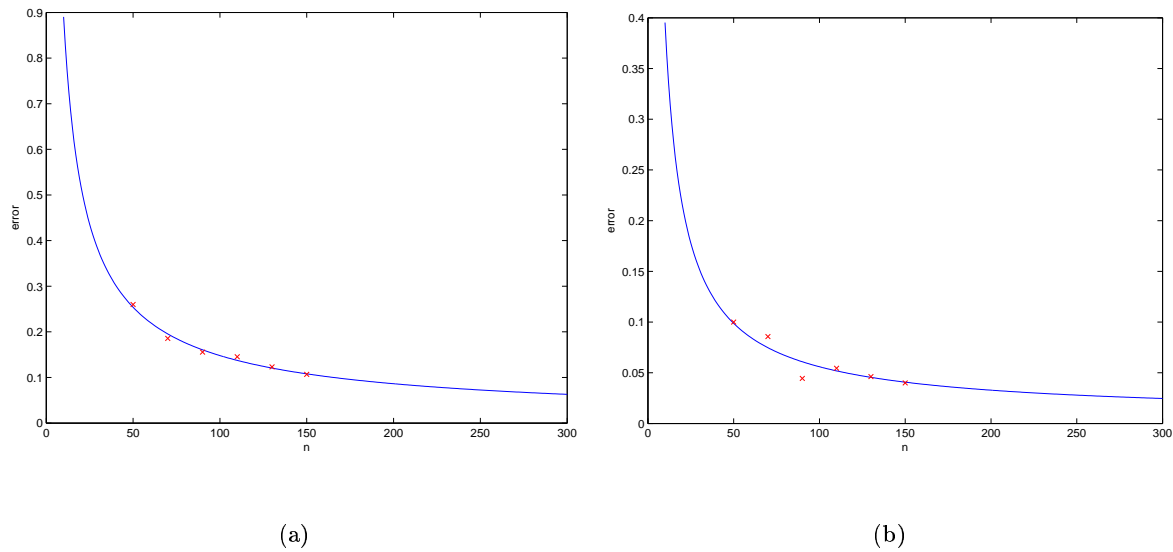


Figure 5.5: Learning curves for masquerade data. (a) User 5; (b) User 6.

$$User6 : Error(n) = 3.0247n^{-0.8902} + 0.00058.$$

According to these learning curve functions, in order to achieve the 1% false positive rate, an often used target for anomaly detectors, the training size has to be 3280 and 654 for User 5 and User 6, respectively. This indicates User 6's behavior is more regular and easier to predict, and therefore the anomaly detector can reach 1% false positive rate with much less training examples.

5.5 Summary

In this chapter, we presented a new and efficient error estimate for support vector machine based anomaly detection systems. The fraction of support vectors sets the upper bound for the generalization performance, i.e., the expected false positive rate for a given training dataset. The effective error estimate can guide the model selection and learning parameter tuning as well as detect concept drift of normal behavior patterns. Furthermore, we proposed to use inverse power law to model learning curves. This provides the basis for estimating the training size requirements for anomaly detection. We also demonstrated the effectiveness of these estimates with various artificial and real-world data.

In addition to more comprehensive experiments and evaluations, one direction of future work is to study how to quantitatively characterize concept drift with the error estimate and incorporate it into an adaptive anomaly detector that handles concept drift. Another direction is to extend our estimates with learning methods other than support vector machines.

Chapter 6

Intrusion Detection and Response: A Game Theoretic Perspective

6.1 Introduction

In May 2003, the Gartner Information Security Hype Cycle report declared that intrusion detection systems (IDSes) are “a market failure” and will be obsolete by 2005 due to the problems such as excessive false positives and false negatives, high operational cost and taxing incident-response process [36]. The report has stirred fierce debate within the IDS vendor as well as research communities. While it is debatable whether the Gartner’s prediction for IDS is short-sighted or not, it is clear that cost-effectiveness will be one of the deciding factors in IDS’ future.

IDS began in the 1980s as a promising paradigm for detecting hackers and malicious insiders that exploit security vulnerabilities or flaws in computer systems [82]. For the last two decades, most research efforts have been devoted to improve the technical effectiveness of IDS. That is, to what degree does an IDS detect and prevent intrusions into the target system, and how good is it at reducing false positives? In practice, however, no IDS will ever be 100% accurate in detecting attacks. False positives and false negatives will be inevitably produced. Moreover, the reduction of one type of error (false positive or false negative) is usually accompanied by an increase in the other type.

Cost-effectiveness is an important, yet often overlooked aspect of IDS. When an organization makes an investment decision on a security mechanism such as IDS, risk assessment and cost-benefit analysis is essential. This includes assessment of the organization's assets and values, identification of threats and vulnerabilities, cost-benefit trade-off evaluation, and so on. The major cost factors that ought to be taken into consideration are the operational cost of IDS, the expected loss due to intrusions and the cost of manual or automatic response to an intrusion [61]. Even when the adoption of IDS technology is justifiable, the IDS operator still faces the challenge of employing the IDS properly and determining the best response strategies against various types of attacks in order to minimize the cost of maintaining the IDS while protecting the system assets.

Our research aims to provide a game theoretic methodology for analysis and design of IDS and improve the effectiveness of IDS technology by modeling the interaction between IDS and attackers in a game playing context. Game theory offers a natural setup for adversarial situations, where multiple players with different objectives compete and interact with each other. As a powerful strategic decision making tool, game theory has been applied in many fields, including economics, political science, etc. [31]. The game theoretic modeling of security systems such as IDS makes it possible to bring the full spectrum of well-developed game theory techniques to bear on the information security problems.

In this chapter, we use a simple two-person, nonzero-sum game to model and analyze the IDS and attacker behavior in a general environment. Attacking and defending of the protected system are formalized in terms of a set of strategies for attacker and IDS, respectively, whereas risk and objectives are formalized in terms of payoff (or utility) functions. Each player strives to maximize his payoff function by selecting a feasible strategy. The solutions based on the game theoretic analysis naturally integrate the cost-effectiveness and technical performance trade-off of the IDS and identify the "best" defense and attack strategies. Specifically, our results provide valuable insights in answering the following fundamental questions:

- Under what condition would an attack most likely occur?
- When is an IDS useful? When does its use become counterproductive?

- When an IDS is deemed useful, what should be its technical specification?
- What's the best response strategy when the IDS raises an alarm? Ignore it or respond to it?
- If the IDS operator can only respond to a subset of the alarms, what percentage is optimal?

The rest of the chapter is organized as follows. In Section 2 we review some related work. Section 3 is a brief introduction to game theory. Section 4 describes details of our game model. In Section 5, we discuss the extensions of our method as well as the challenges of game theoretic modeling of security systems. Section 6 presents conclusions.

6.2 Related Work

The economics of information security is an emerging and growing research area [78] [89]. For example, Gordon and Loeb [34] presented an economic model that determines the optimal amount to invest to protect a given set of information assets. Theagwara [42] examined the effect of implementation methods, management methods and intrusion detection policy on the return of investment.

The application of game theory to the domain of computer security has recently been a topic of interest. Lye and Wing [74] constructed a stochastic game to model the interactions between an attacker and the administrator in a typical network environment. Liu and Zang [71] presented a general incentive-based method to model attacker's intent, objectives and strategies (AIOS) and employed game theoretic techniques to infer AIOS. Different game models (stochastic or Bayesian games) were proposed based on the accuracy of intrusion detection and the correlation among attack actions. Alpcan and Basar [2] argued that game theory can provide the much needed mathematical framework for analysis, decision and control processes for information security and intrusion detection. They designed a security warning system for distributed IDS using Shapley values. A two-person finite game was used to model security attacks in a multiple sensor environment. Cavusoglu and Raghunathan [12] took a game theoretic approach to determine the optimal configura-

tion (detection and false positive rates) of an IDS and compared it with the decision theory approach. While the game models in this paper are similar to those in [2] and [12], our focus is on the modeling of general IDS and its insights in IDS' cost-effectiveness. In addition, our incentive-based payoff functions more accurately represent the interactions between the IDS and attacker.

Decision theory is often employed to facilitate risk management and cost-benefit analysis [32] [41]. For example, Gaffney and Ulvila [32] used a decision analysis to evaluate and configure IDS. Decision theory assigns prior probabilities (usually fixed and exogenous) to handle the uncertainty of the environment (e.g., the prior probability of an intrusion). As pointed out in [12], decision theory based approaches are inadequate for IDS problems because they don't allow the defense system's decisions to influence the attacker's behavior. In contrast, game theory brings the attacker into the model and thus makes itself more attractive for handling the strategic interdependence between the IDS and attacker.

6.3 Review of Game Theory

A *game* is a formal representation of a situation in which a number of individuals with different objectives compete and interact with each other. In general, a game consists of the following elements:

- *Players*: who are involved?
- *Rules*: who moves when? What do they know when they move? What can they do (i.e., what strategies do they have)?
- *Outcomes*: for each possible set of actions by the players, what is the outcome of the game?
- *Payoffs*: what are the players' utility functions over the possible outcomes?

A classic example is the game of *Matching Pennies*. In this game, player 1 puts a penny down, either heads up or tails up. Player 2, not knowing player 1's choice, also puts a penny down. If the two pennies match (either both heads up or both tails up), player

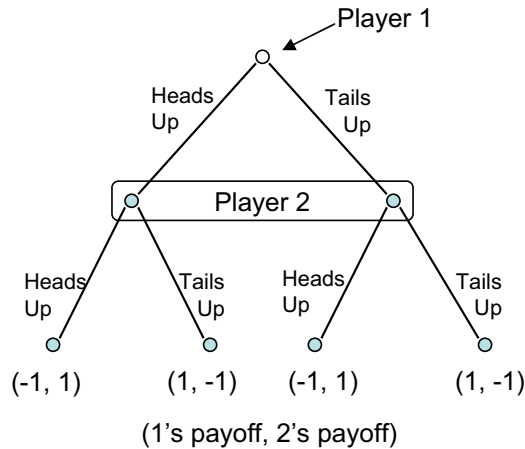


Figure 6.1: Extensive form for Matching Pennies.

1 pays 1 dollar to player 2. Otherwise, player 2 pays 1 dollar to player 1. The Matching Pennies game can be represented in the *extensive form* depicted in Figure 6.1. Due to its treelike structure, the extensive form is also known as a *game tree*. Matching Pennies is a two-player zero-sum game, in which case the sum of the payoffs is always zero. In general, however, most games of interest are non-zero-sum.

A central concept of game theory is the notion of a player's *strategy*. A strategy is a complete contingent plan, or decision rule, that specifies how the player will act in every possible circumstance in which he might be called upon to move. For example, in Matching Pennies, both players have two possible strategies: play heads (H) or tails (T). Then the game can be represented in terms of strategies and their associated payoffs. This representation, often depicted as a matrix, is known as the *normal* (or *strategic form*). The normal form of Matching Pennies is presented in Figure 6.2. Each row of the matrix represents a strategy of player 1, and each column a strategy of player 2. Within each cell, the first entry is player 1's payoff for the corresponding strategy profile; the second is player 2's.

Up to this point, we have assumed that players make their strategic decisions with certainty. However, a player could randomize when faced with a choice. We now call the deterministic strategy a *pure strategy*. By contrast, a *mixed strategy* for a player is simply a probability distribution over his pure strategies. For example, a mixed strategy for player 1

| | | |
|---|-------|-------|
| | H | T |
| H | -1, 1 | 1, -1 |
| T | 1, -1 | -1, 1 |

Figure 6.2: Normal form of Matching Pennies.

| | | |
|---|------|------|
| | L | R |
| U | 5, 1 | 6, 2 |
| M | 8, 4 | 3, 6 |
| D | 9, 6 | 2, 8 |

Figure 6.3: Game example.

in Matching Pennies is to play heads with the probability of 30%, and tails of 70%, instead of playing heads or tails all the time.

It is not so obvious to predict how each player should play Matching Pennies in order to maximize his own payoff. Consider the game illustrated in Figure 6.3 instead. In this game, player 1 has three pure strategies (U, M and D) and player 2 has two pure strategies (L and R). Note that, no matter how player 1 plays, R gives player 2 a strictly higher payoff than L does. In formal language, strategy L is *strictly dominated*. Thus, a “rational” player 2, who uses only those strategies that are best responses to some beliefs he might have about the strategies of his opponent, should not play L. Furthermore, if player 1 knows that player 2 will not play L, then U is a better choice than M or D. This process of elimination is called *iterated dominance*. It reduces the strategy sets of the players and thus simplifies the game.

Unfortunately, most games (e.g., Matching Pennies) are not solvable by iterated dominance. The *Nash equilibrium* solution provides the optimal response to other players’ strategies for each player. In a Nash equilibrium, none of the players has an incentive to deviate unilaterally from the equilibrium as long as the other players don’t deviate. It can be proved that every finite game has a mixed strategy Nash Equilibrium. Solving the

Nash Equilibrium for a 2×2 matrix game is trivial, although it can be costly for higher-dimensional matrix games [31]. The Nash Equilibrium for Matching Pennies is the mixed strategy in which each player randomizes between his two pure strategies, assigning equal probability to each.

So far we have assumed that players know all relevant information about each other, including the structure of the game and payoffs that each receives. Such games are known as games of *complete information*. However, this assumption may be invalid in practice. How to weaken or entirely dispense with this assumption and solve games of *incomplete information* has been a challenging research topic in game theory. One widely used approach is to transform incomplete information about players into *imperfect* information about nature's moves. A game of this sort is known as a *Bayesian game*.

6.4 Game Theoretic Modeling

We use a two-person non-zero-sum game model to formulate the strategic interdependence between a general IDS and an attacker. The IDS can be host-based or network-based in an organizational environment ¹. The organization can range from small enterprises to government agencies. Intrusions are identified through anomaly detection, misuse detection or hybrid techniques. The attacker can be a skillful intruder from outside, a malicious insider, or even a script kiddie.

Before we delve into the game modeling details, we shall introduce the parameters and identify the cost and payoff factors related to both players of the game.

6.4.1 Parameters and Cost/Payoff Factors

Table 6.1 summarizes the parameters used in our models. The parameters are always positive.

The cost and payoff factors associated with an IDS are:

- Operational cost (*OC*). This includes the cost of purchasing the IDS, the amount

¹Our model can be easily extended to the case when there are multiple IDSes within the organization. The game formulation will remain the same.

Table 6.1: List of parameters.

| notation | description |
|----------|--|
| α | false alarm rate, the probability of an alarm, given no intrusion |
| β | false positive rate, the probability of no alarm, given an intrusion |
| δ | intrusion detection rate, $\delta = 1 - \beta$ |
| ψ | probability of an attacker conducting an attack |
| ρ | probability of responding to an alarm |
| OC | IDS operational cost |
| DC | damage cost of an attack |
| RC | IDS response cost |
| RD | reward (i.e., payoff) of IDS for responding to an attack |
| LC | attacker's cost of launching an attack |
| PC | penalty to the attacker when the attack is detected |
| BA | benefit to the attacker when the attack is undetected |

of time and computing resources needed to process the audit data stream and the personnel cost involved in administering and maintaining the IDS. Considering the voluminous audit data an IDS processes, the average operational cost for each audit event (the unit of analysis) should be nominal.

- Damage cost (DC), the amount of damage to the organization by an attack when IDS is not available or ineffective. It can be measured by the degradation of a set of security measurements associated with the organization's security metrics [71]. Different types of attacks can incur various levels of damage. Here we use DC to represent the expected damage cost by a generic attack.
- Response cost (RC), the cost of acting upon an IDS alarm. Depending on the type of response mechanisms being used, the response cost includes the computing resources for terminating the offending connection or session, the downtime needed to repair and patch the computer systems, the labor cost of the response team, and so on.
- RD , the reward to the organization for responding to an attack. It can be measured by the improvement of the organization's security metrics after the response to the attack. In other words, RD is the potential damage cost caused by the attack if it went unnoticed otherwise ².

²For organizations such as law enforcement agencies, there is additional value for catching the attackers.

Similarly, from an attacker's point of view, the cost and payoff factors include:

- LC , the cost for an attacker to launch an attack. It is the amount of time and resources needed to conduct the attack, which includes searching for system vulnerabilities, designing malicious code to exploit the vulnerabilities, etc.. It is reasonable for a vigilant security officer to assume that the cost of attack LC is small.
- Penalty cost (PC). This characterizes the risk for the attacker to be traced-back and punished. Quantitatively, it is the product of the probability of the attacker being held accountable and the penalty to the attacker when he is caught.
- BA , the benefit to the attacker when the attack is undetected. We use the attacker's incentive, quantified as the organization's damage cost DC , to represent BA (i.e., $BA = DC$), although it may not be the benefit he receives directly from the attack [71].

In this section we assume the values of the parameters are common knowledge known to both players of the game. Section 5 shows how we can weaken this assumption and design the game of incomplete information. We further assume that OC and LC are much less than the other cost and payoff factors.

6.4.2 IDS vs. Attacker

The extensive form of the game is illustrated in Figure 6.4. The attacker's strategy is either to attack or not to attack the targeted organization. Accordingly, the organization may simply choose to have or not to have an IDS to defend against the attacker. Due to the imperfect technical performance of the IDS, it is possible that the IDS can raise a false alarm when there is no attack or generate no alarm when an attack is occurring. The corresponding probabilities and payoffs are shown in the shaded area in Figure 6.4. In this game we assume the IDS operator responds to every alarm the IDS generates (i.e., $\rho = 1$).

Figure 6.5 presents the normal form of the game. The payoffs for both players are determined as follows. If the attacker decides not to commit an attack, he receives no payoff. In contrast, the the organization has to pay for the operational cost (OC) along

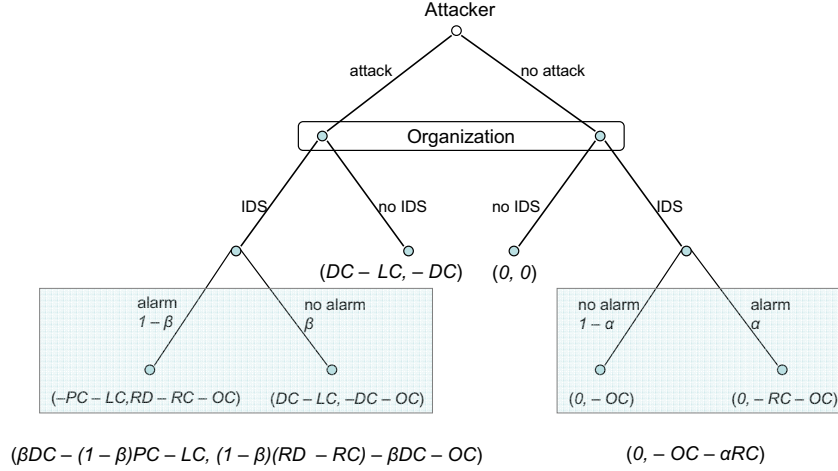


Figure 6.4: Extensive form for Game A.

| | IDS | No IDS |
|-----------|--|----------------------------------|
| Attack | $A_{11} = \beta DC - (1 - \beta) PC - LC, B_{11} = (1 - \beta)(RD - RC) - \beta DC - OC$ | $A_{12} = DC - LC, B_{12} = -DC$ |
| No Attack | $A_{21} = 0, B_{21} = -OC - \alpha RC$ | $A_{22} = 0, B_{22} = 0$ |

Figure 6.5: Normal form of Game A.

with the cost of false alarms (αRC) if an IDS is employed. When the attacker conducts an attack while the IDS is not in place or does not generate an alarm, the organization's loss is DC , whereas the attacker's payoff is the difference between BA and LC , which is the same as $DC - LC$. If the IDS successfully detects an attack, the organization gains RD at the cost of RC and OC . Meanwhile, the attacker bears the expected penalty cost PC in addition to LC ³. The organization's expected payoffs when having an IDS is determined by taking the sum of products of probabilities and payoffs for two scenarios (alarm or no alarm).

We shall consider the attacker's strategies first. As shown in Figure 6.5, the attacker's payoff $A_{12} = DC - LC$ is greater than $A_{22} = 0$, based on our assumption that $LC \ll DC$. "Attack" would be the attacker's dominant strategy if A_{11} is also greater than

³Our model can be extended to accommodate the case in which the attack is partially in progress when the IDS raises an alarm and the organization only recovers a portion of the damage.

$A_{21} = 0$. That is,

$$\begin{aligned} A_{11} &= \beta DC - (1 - \beta)PC - LC \\ &\simeq \beta DC - (1 - \beta)PC \\ &> 0 \end{aligned}$$

This is equivalent to

$$\frac{DC}{PC} > \frac{1 - \beta}{\beta},$$

or

$$\frac{DC}{PC} > \frac{\delta}{1 - \delta}.$$

We define $\lambda = DC/PC$, which essentially represents the benefit-to-risk ratio for the attacker.

Remark 1 If the attacker's benefit-to-risk ratio λ is greater than $\delta/(1 - \delta)$, "attack" is his dominant strategy. In other words, "no attack" is the strictly dominated strategy. A rational attacker would always choose to attack the organization and thus maximize his payoff regardless of the organization's decision.

This result is not surprising. Intuitively, the greater potential damage cost or the lower risk would motivate the attacker more to commit the attack. On the other hand, the higher the attack detection rate, the more risk to the attacker and the less likely the attack occurring. This is illustrated in Figure 6.6. For a fixed λ value, an IDS with the intrusion detection rate less than $\delta_{threshold} = \lambda/(1 + \lambda)$ ($\lambda > \frac{\delta}{1 - \delta}$ is equivalent to $\delta < \frac{\lambda}{1 + \lambda}$) would not play a deterrent role for the attacker at all. For example, when $\lambda = 10$, $\delta_{threshold} = 0.91$. Even when $\lambda = 5$, $\delta_{threshold}$ is still as high as 0.83. On the other hand, for a fixed δ value, a malicious attacker who wants to maximize his payoff would rather conduct an attack with $\lambda > \delta/(1 - \delta)$ than do nothing! This implies that the effective way to discourage the occurrence of an attack is to not only improve the attack detection rate but also increase the punishment for the attackers. In addition, it is interesting to note that false alarms cost nothing to the attacker. Therefore α does not come into play when determining the attacker's dominant strategy.

We next examine the organization's strategies. The negative B_{21} is always less than B_{22} . If B_{11} is also less than B_{12} , the organization's dominant strategy would be "no

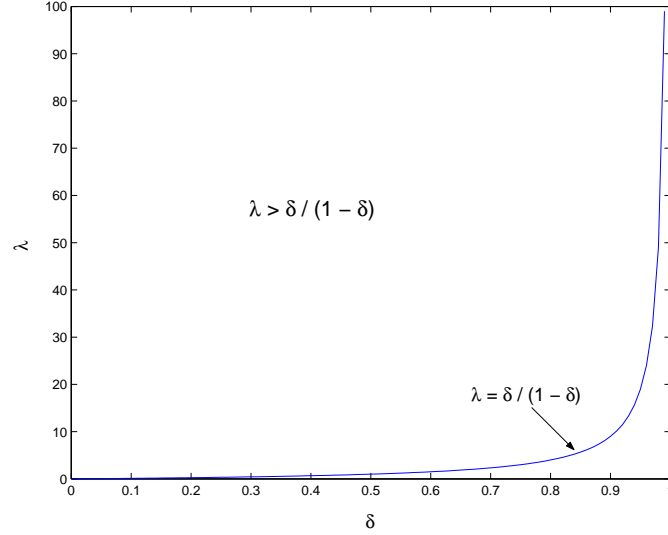


Figure 6.6: Attacker's dominant strategy is "attack" when $\lambda > \delta / (1 - \delta)$.

IDS". Therefore we have

$$\begin{aligned}
 B_{12} - B_{11} &= -DC - (1 - \beta)(RD - RC) + \beta DC + OC \\
 &\simeq (1 - \beta)(RC - DC - RD) \\
 &> 0
 \end{aligned}$$

Clearly, B_{12} is greater than B_{11} if and only if $RC - DC - RD > 0$ (OC is relatively small and neglected and β is always less than 1).

Remark 2 Having an IDS becomes counterproductive when $RC > DC + RD$. It is more cost-effective for the organization not to use the IDS due to its high response cost.

Note that RC is compared to the sum of DC and RD (instead of DC alone). As long as $RC \leq DC + RD$, it is beneficial for the organization to employ an IDS. A similar analysis of the IDS' decisions upon an alarm (respond or not respond) would reveal the following:

Remark 3 When an IDS is deployed and it generates an alarm, it is more cost-effective to ignore the alarm and not to respond to it if $RC > DC + RD$.

In case there is no dominant pure strategy for neither player, we need to examine the Nash Equilibrium of the game. Let ψ and $1 - \psi$ be the probabilities for attacker's

strategies “attack” and “no attack”, respectively. Also let q and $1 - q$ be the probabilities for strategies “having IDS” and “no IDS” of the organization. In practice, an probability distribution over an organization’s pure strategies can be interpreted as the extent to which the organization needs an IDS or the percentage of the time that the IDS should be available.

The Nash Equilibrium solution of the game is as follows:

$$\psi^* = \frac{\alpha RC}{\delta(DC + RD - RC) + \alpha RC},$$

$$q^* = \frac{DC}{\delta(PC + DC)} = \frac{\lambda}{\delta(1 + \lambda)}.$$

The organization’s mixed strategy $(q^*, 1 - q^*)$ is the best response to the attacker’s strategies. In fact, if the IDS is available with the probability of q^* , the attacker’s expected payoff will be 0, whether he attacks or not. Similarly, if the probability for the attacker to attack the organization is ψ^* , the organization’s expected cost is

$$V^* = -\frac{\alpha DC RC}{\delta(DC + RD - RC) + \alpha RC},$$

whatever its defense strategy is.

Remark 4 When an IDS is useful, the organization’s best response to the attacker’s strategies is to employ the IDS with the probability of $\lambda/[\delta(1 + \lambda)]$. This way, the attacker’s expected payoff will be 0, whatever he does.

It may seem counter-intuitive that q^* is proportional to $1/\delta$ and it increases monotonically with increasing λ . However, δ is expected to have a value greater than $\delta_{threshold} = \lambda/(1 + \lambda)$. The higher benefit-to-risk ratio for the attacker, the better the IDS ought to be in terms of intrusion detection rate, and the more it is inclined for the organization to have the IDS in order to catch the attacker and reduce his payoff. On the other hand, ψ^* increases with increasing false alarm cost of the IDS or decreasing δ , which implies an attack is more likely to happen if the IDS is less accurate.

Alternatively, q can be interpreted as the probability of responding to an alarm when an IDS is employed (i.e., ρ). Therefore the optimal value of ρ is q^* .

Remark 5 The optimal probability of responding to an IDS alarm is $\lambda/[\delta(1 + \lambda)]$.

Table 6.2 lists a set of numerical examples of λ , δ , $\delta_{threshold}$ and q^* values.

Table 6.2: Numerical examples.

| λ | δ | $\delta_{threshold} = \lambda/(1 + \lambda)$ | $q^* = \lambda/[\delta(1 + \lambda)]$ |
|-----------|----------|--|---------------------------------------|
| 0.1 | 80% | 9.1% | 11.4% |
| 1 | 80% | 50% | 62.5% |
| 5 | 85% | 83.3% | 98.0% |
| 10 | 95% | 90.9% | 95.7% |

6.5 Discussion

The game theoretic methodology of cost-benefit analysis is not limited to IDS. In fact, it can be easily extended to any security mechanism. It is important to bring adversarial attackers into the security models. The game theoretic formulation makes it possible to understand an attacker's intent and strategies from the attacker's perspective, which has been an often neglected facet of computer security research.

Our game model assumes complete information of the IDS and attackers. However, this assumption is somewhat unrealistic in practice. In particular, it is very difficult to estimate an attacker's payoff values. Significant research effort is needed to address the issue of accurate quantification of the attacker's payoff functions. Nevertheless, a qualitative game theoretic analysis can still bring unique and valuable insights to the understanding of the attacker's behavior and the decision making of security systems. Meanwhile, Bayesian games can be used to accommodate the uncertainty of the payoffs and handle the case of incomplete information. For instance, an attacker can be classified into three types: a skillful intruder from outside, a malicious insider, or a script kiddie. A prior probability is assigned to each type and the payoff values are identified for each sub-game associated with each type. Then the transformed game can be analyzed with standard techniques. Similarly, different attack types, such as denial of services, port scanning, masquerading, and so on, can be incorporated into the game model as well.

Finally, the interaction between attackers and security systems can be viewed as a repeated game. Both players can improve with the experience of playing the repeated games. How to incorporate game theory with learning is another important issue for our future work.

6.6 Summary

Cost-effectiveness is an important aspect of intrusion detection systems. In this chapter, we have presented a game theoretic methodology for cost-benefit analysis and design of IDS. A simple two-person game was used to model the strategic interdependence between a general IDS and an attacker. The solutions based on the game theoretic analysis naturally integrate the cost-effectiveness and technical performance tradeoff of the IDS and provide valuable insights in intrusion detection and response.

Our game theoretic methodology can be applied to the cost-benefit analysis of any security mechanism. The main difficulty is the quantification of the players' payoffs. Our future work includes Bayesian game modeling of security systems and learning of repeated games.

Chapter 7

Conclusions and Future Directions

7.1 Conclusions

In this dissertation, we examined the fundamental issues involved in anomaly detection and presented our solutions to some of the issues. Specifically, we:

- presented a new approach to learn program behavior for intrusion detection. Text categorization techniques were adopted to convert each process to a vector and calculate the similarity between two program activities. Then the k -Nearest Neighbor classifier was employed to classify program behavior as normal or intrusive. We demonstrated that our approach is able to effectively detect intrusive program behavior while a low false positive rate is achieved.
- designed an adaptive anomaly detection framework that is able to handle concept drift and online learning for dynamic, changing environments. Through the use of unsupervised evolving connectionist systems, normal behavior changes are efficiently accommodated while anomalous activities can still be recognized. We demonstrated the performance of our adaptive anomaly detection systems and showed that the false positive rate can be significantly reduced.
- introduced an efficient error estimate for support vector machine based anomaly detection and proposed to use inverse power-law learning curves to estimate training size requirements.

- presented a game theoretic methodology for cost-effectiveness analysis of IDS. A two-person, nonzero-sum game was used to model the strategic interdependence between an IDS and an attacker. The solutions based on the game theoretic analysis provide valuable insights in answering questions such as when an IDS is useful and how the IDS should respond to an alarm.

7.2 Future Directions

Over the past two decades, machine learning has played a significant role in building anomaly detection models for detecting previously unknown attacks. Yet the current state-of-the-art of intrusion detection systems is still primitive and much remains to be explored. The complexity of intrusion detection problems of today presents new research challenges in the field of machine learning.

7.2.1 Theoretic Analysis

Like many other fields, intrusion detection has been based on a combination of intuition and ad hoc techniques. There is a lack of underlying theoretic analysis in this field.

First of all, anomaly detection assumes that intrusive activities are distinct from normal activities and deviation from normal behaviors by users or programs are indications of intrusions. Although the assumption is intuitively appealing, there has been little theoretical support. Therefore, there is a need to address the soundness and completeness of anomaly detection methods. In other words, what types of intrusions can and can not be detected by anomaly detection? What are the power and limitations of a machine learning based anomaly detection system? Is it possible to distinguish anomalies related to intrusions from those related to other factors? Despite a few previous attempts [37] [98] [105], these questions remain largely open.

Second, for a particular environment, what features, metrics, machine learning techniques provide the best performance to model the normal behavior of the environment? How efficient is the learning system in time, space and data? How to systematically in-

corporate domain knowledge in anomaly detection? Computational learning theory, the theoretical underpinning of machine learning, might shed light on these fundamental questions.

Last, an equally important issue is to develop some theoretical understanding of intrusive behaviors. Attackers are constantly looking for new way to attack. Is it possible to get an abstract view of various intrusions or even predict the next wave of attacks?

7.2.2 Learning for Understanding and Planning

Intrusion prevention, detection and response are three general tasks of security officers. Once an intrusion is detected, proper response should be evoked to recover and prevent future attacks. The current state-of-art of IDSes, mostly based on classification techniques, provides little insight on the attacker's intent (especially in case of sophisticated attacks). Therefore intrusion response heavily relies on manual forensic analysis. Machine learning techniques that aim to interpret observations, create situation awareness, and plan automated response can play a significant role in the progress of the IDS field.

7.2.3 Ensemble Learning

One method of improving the intrusion detection accuracy is to combine the outputs of different classifiers, a strategy known as ensemble learning. In practice, there are many different types of intrusions, and different methods are needed to detect them using multiple and diverse sensors. Combining the evidence from multiple classifiers can effectively improve the accuracy and trustworthiness of IDSes. There are various ways of constructing an ensemble of classifiers. For example, one can divide the original training dataset into subsets and different classifiers are trained with each subset. This strategy works well when the hypothesis learned is fairly sensitive to small changes in training data. Another general approach is to manipulate the input features in a similar fashion. Furthermore, it is possible to build an ensemble of classifiers each using a different learning algorithm. Learning algorithms using radically different principles probably may produce very different classifiers. The key is to correlate the outputs of these classifiers (for example, using the majority voting rule).

Appendix A

Support Vector Machine Based Anomaly Detection

Support Vector Machine is a relatively new and state-of-the-art classification method pioneered by Vapnik [104]. It is based on the so-called *structural risk minimization* principle, which minimizes an upper bound on the generalization error. The method performs a mapping from the input space to a higher-dimensional feature space through the use of a kernel function. It separates the data in the feature space by means of a maximum margin hyperplane.

Inspired by SVM, Tax and Duin [99] proposed to use a spherically shaped boundary, which is the smallest hypersphere containing most of the training data points, to describe the data. The hypersphere, also known as Support Vector Data Description (SVDD), represents the characterization of the training set and can be used for anomaly detection. Naturally, the data points that fall outside the hypersphere are regarded as outliers (i.e., anomalies).

To begin, let us assume that we are given a training set $S = \{\mathbf{x}_i\}, i = 1, 2, \dots, N$, where each vector \mathbf{x}_i represents a data point or an object. The hypersphere enclosing most of S is characterized by center \mathbf{c} and radius R . One could require the hypersphere to contain all the data points in S instead. However, it would not be a robust approach in the presence of noise in the training data.

To allow the possibility of outliers in \mathcal{S} , the distance from \mathbf{x}_i to the center \mathbf{c} should not be strictly smaller than R , but larger distances should be penalized. Therefore we introduce the so-called slack variables ξ_i , where ξ_i is zero for points inside the hypersphere and measures the degree to which the distance squared from \mathbf{c} exceeds R^2 for points outside. Then finding the smallest hypersphere (\mathbf{c}, R) amounts to the following minimization problem:

$$\min_{\mathbf{c}, R, \xi} R^2 + \frac{1}{\nu N} \sum_{i=1}^N \xi_i,$$

with constraints that all data are within a slightly larger hypersphere here:

$$\|\mathbf{x}_i - \mathbf{c}\|^2 \leq R^2 + \xi_i, \quad \xi_i \geq 0, \quad i = 1, 2, \dots, N.$$

Here parameter $\nu \in (0, 1)$ controls the trade-off between the volume of the hypersphere and errors. Its meaning will be clearer later.

If we introduce a Lagrangian and rewrite the original optimization in terms of the Lagrange multipliers α_i , we obtain the dual problem:

$$\min_{\alpha} \sum_{i,j=1}^N \alpha_i \alpha_j (\mathbf{x}_i \cdot \mathbf{x}_j) - \sum_{i=1}^N \alpha_i (\mathbf{x}_i \cdot \mathbf{x}_i)$$

subject to

$$\sum_{i=1}^N \alpha_i = 1, \quad 0 \leq \alpha_i \leq \frac{1}{\nu N}, \quad i = 1, 2, \dots, N.$$

The solution of the dual problem gives a set of α_i . When an object \mathbf{x}_i satisfies $\|\mathbf{x}_i - \mathbf{c}\|^2 < R^2$, the corresponding α_i will be zero. For objects satisfying $\|\mathbf{x}_i - \mathbf{c}\|^2 = R^2 + \xi_i$, $\alpha_i > 0$:

$$\begin{aligned} \|\mathbf{x}_i - \mathbf{c}\|^2 < R^2 &\Rightarrow \alpha_i = 0 \\ \|\mathbf{x}_i - \mathbf{c}\|^2 = R^2 &\Rightarrow 0 < \alpha_i < \frac{1}{\nu N} \\ \|\mathbf{x}_i - \mathbf{c}\|^2 > R^2 &\Rightarrow \alpha_i = \frac{1}{\nu N} \end{aligned}$$

The center of the optimal hypersphere is simply a linear combination of the objects:

$$\mathbf{c} = \sum_{i=1}^N \alpha_i \mathbf{x}_i.$$

Only objects \mathbf{x}_i with $\alpha_i > 0$ are needed to define the hypersphere and these objects are therefore called *support vectors*. By definition, R is the distance from the center of the optimal hypersphere \mathbf{c} to any of the support vectors on the boundary known as the *unbounded*

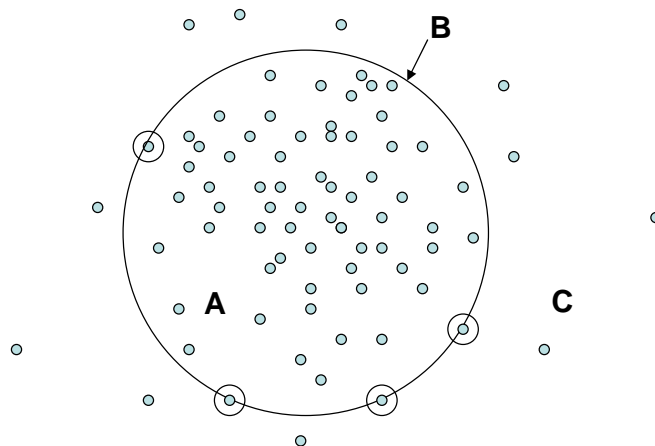


Figure A.1: A two-dimensional sphere (solid line) containing most of the data. Enclosed in a circle are four support vectors on the boundary.

support vectors. Support vectors that fall outside the hypersphere with $\alpha_i = 1/\nu N$, the *bounded support vectors*, are excluded. Therefore:

$$R^2 = (\mathbf{x}_k \cdot \mathbf{x}_k) - 2 \sum_i \alpha_i (\mathbf{x}_i \cdot \mathbf{x}_k) + \sum_{i,j} \alpha_i \alpha_j (\mathbf{x}_i \cdot \mathbf{x}_j)$$

for any \mathbf{x}_k with $0 < \alpha_i < 1/\nu N$. To test an object \mathbf{z} , its distance to the center of the optimal hypersphere is calculated and compared with the radius R . It is considered anomalous if the distance is greater than R . Figure A.1 illustrates the sphere and support vectors for a two-dimensional data set.

Instead of finding the smallest hypersphere in the original input space, we can perform a mapping ϕ from the input space to a feature space such that the inner product in the image of ϕ can be computed by evaluating some simple kernel:

$$K(\mathbf{x}, \mathbf{y}) = (\phi(\mathbf{x}), \phi(\mathbf{y})),$$

such as the Gaussian kernel

$$K(\mathbf{x}, \mathbf{y}) = e^{-\gamma \|\mathbf{x} - \mathbf{y}\|^2}.$$

Schölkopf et al. [93] took an alternative approach, also known as one-class SVM, to adapting the SVM paradigm to the anomaly detection problem. A hyperplane is placed such that it separates the dataset from the origin with maximal margin. Although this is not a

close boundary around the data, it gives equivalent solutions when the data is preprocessed to have unit norm or the kernel $K(\mathbf{x}, \mathbf{y})$ depends on only $\mathbf{x} - \mathbf{y}$ so that $K(\mathbf{x}, \mathbf{x})$ is constant. One-class SVM has been used for computer intrusion detection [24].

It can be shown [96] that finding the smallest hypersphere containing most of the data is a convex optimization problem. This ensures a unique hypersphere solution that can be found efficiently. Furthermore, ν is an upper bound on the fraction of outliers, that is, training points lying outside the optimal hypersphere. Meanwhile, ν is a lower bound on the fraction of support vectors. In other words, there are at most νN training points outside the hypersphere, while at least νN of the training points do not lie in the interior of the hypersphere and they serve as the support vectors.

Bibliography

- [1] K. Aas and L. Eikvil. Text categorisation: A survey, <http://citeseer.nj.nec.com/aas99text.html>, 1999.
- [2] T. Alpcan and T. Basar. A game theoretic approach to decision and analysis in network intrusion detection. In *Proceedings of 42nd IEEE Conference on Decision and Control*, Maui, HI, December 2003.
- [3] M. Asaka, T. Onabuta, T. Inoue, S. Okazawa, and S. Goto. A new intrusion detection method based on discriminant analysis. *IEEE TRANS. INF. & SYST.*, E84-D(5):570–577, 2001.
- [4] J. Ashbourn. *Biometrics: Advanced Identity Verification*. Springer-Verlag, London, 2000.
- [5] S. Axelsson. Research in intrusion-detection systems: A survey. Technical Report 98–17, Department of Computer Engineering, Chalmers University of Technology, Goteborg, Sweden, December 1998.
- [6] S. Axelsson. The base-rate fallacy and the difficulty of intrusion detection. *ACM Trans. Inf. Syst. Secur.*, 3(3):186–205, 2000.
- [7] F. Bergadano, D. Gunetti, and C. Picardi. User authentication through keystroke dynamics. *ACM Trans. Inf. Syst. Secur.*, 5(4):367–397, 2002.
- [8] M. Bishop. *Computer Security: Art and Science*. Addison-Wesley, 2002.
- [9] J. Cannady. Next generation intrusion detection: Autonomous reinforcement learning of network attacks. In *Proceedings of the 23rd National Information Systems Security Conference*, 2000.
- [10] G. A. Carpenter and S. Grossberg, editors. *Pattern Recognition by Self-Organizing Neural Networks*. MIT Press, Cambridge, MA, 1991.
- [11] G. A. Carpenter, S. Grossberg, and D. B. Rosen. Fuzzy art: Fast stable learning and categorization of analog patterns by an adaptive resonance system. *Neural Networks*, 4:759–771, 1991.
- [12] H. Cavusoglu and S. Raghunathan. Configuration of intrusion detection systems: A comparison of decision and game theoretic approaches. In *Proceedings of International Conference on Information Systems (ICIS)*, Seattle, WA, December 2003.
- [13] C.-C. Chang and C.-J. Lin. LIBSVM: a library for support vector machines, <http://www.csie.ntu.edu.tw/~cjlin/libsvm>, 2001.

- [14] C. Cortes, L. Jackel, S. Solla, V. Vapnik, and S. Denker. Asymptotic values and rates of convergence. In *Advances in Neural Information Processing Systems VI*, San Francisco, CA, 1994. Morgan Kaufmann.
- [15] M. Damashek. Gauging similarity with n-grams: Language-independent categorization of text. *Science*, 267:843–848, February 1995.
- [16] DARPA Intrusion Detection datasets. <http://www.ll.mit.edu/ist/ideval/data/>.
- [17] H. Debar, M. Becker, and D. Siboni. A neural network component for an intrusion detection system. In *Proceedings of the 1992 IEEE Symposium on Security and Privacy*, page 240. IEEE Computer Society, 1992.
- [18] D. E. Denning. An intrusion-detection model. *IEEE Trans. Softw. Eng.*, 13(2):222–232, 1987.
- [19] L. Devroye, L. Györfi, and G. Lugosi. *A Probabilistic Theory of Pattern Recognition*. Springer Verlag, New York, 1996.
- [20] R. Dietrich, M. Opper, and H. Sompolinsky. Support vectors and statistical mechanics. In A. J. Smola and P. J. Bartlett, editors, *Advances in Large Margin Classifiers*, pages 359–368. MIT Press, Cambridge, MA, 2000.
- [21] T. Dietterich and P. Langley. Machine learning for cognitive networks: Technology assessment and research challenges. draft of may 11, 2003, <http://web.engr.oregonstate.edu/tgd/kp/dl-report.pdf>.
- [22] The Fifth International Knowledge Discovery and Data Mining Tools Competition (KDD Cup 1999) Data. <http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html>, 1999.
- [23] E. Eskin. Anomaly detection over noisy data using learned probability distributions. In *Proceedings 17th International Conference on Machine Learning*, pages 255–262. Morgan Kaufmann, San Francisco, CA, 2000.
- [24] E. Eskin, A. Arnold, M. Prerau, L. Portnoy, and S. Stolfo. A geometric framework for unsupervised anomaly detection: Detecting intrusions in unlabeled data. In D. Barbara and S. Jajodia, editors, *Applications of Data Mining in Computer Security*. Kluwer, 2002.
- [25] E. Eskin, M. Miller, Z.-D. Zhong, G. Yi, W. Lee, and S. Stolfo. Adaptive model generation for intrusion detection systems. In *Proceedings of the ACMCCS Workshop on Intrusion Detection and Prevention*, Athens, Greece, 2000.
- [26] F. Esponda, S. Forrest, and P. Helman. A formal framework for positive and negative detection. *IEEE Transactions on Systems, Man, and Cybernetics*, 34(1):357–373, 2004.
- [27] W. Fan. *Cost-sensitive, scalable and adaptive learning using ensemble-based Methods*. PhD thesis, Columbia University, February 2001.

- [28] T. Fawcett and F. Provost. Activity monitoring: Noticing interesting changes in behavior. In Chaudhuri and Madigan, editors, *Proceedings on the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 53–62, San Diego, CA, 1999.
- [29] S. Forrest, S. A. Hofmeyr, and A. Somayaji. Computer immunology. *Communications of the ACM*, 40:88–96, 1997.
- [30] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff. A sense of self for unix processes. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, page 120. IEEE Computer Society, 1996.
- [31] D. Fudenberg and J. Tirole. *Game Theory*. MIT, 1991.
- [32] J. Gaffney and J. Ulvila. Evaluation of intrusion detectors: A decision theory approach. In *Proceedings of 2001 IEEE Symposium on Security and Privacy*, pages 50–61, Los Alamitos, CA, 2001.
- [33] A. K. Ghosh and A. Schwartzbard. A study in using neural networks for anomaly and misuse detection. In *Proceedings of the Eighth USENIX Security Symposium*, 1999.
- [34] L. A. Gordon and M. P. Loeb. The economics of information security investment. *ACM Trans. Inf. Syst. Secur.*, 5(4):438–457, 2002.
- [35] S. Greenberg. Using unix: Collected traces of 168 users. Technical Report 88/333/45, Department of Computer Science, University of Calgary, Calgary, Canada, 1998.
- [36] Gartner Group. Hype cycle for information security, <http://www.gartner.com/pages/story.php.id.8789.s.8.jsp>, 2003.
- [37] P. Helman and G. Liepins. Statistical foundations of audit trail analysis for the detection of computer misuse. *IEEE Trans. Software Engineering*, 19(9):886–901, September 1993.
- [38] G. Helmer, J. S. K. Wong, V. G. Honavar, and L. Miller. Automated discovery of concise predictive rules for intrusion detection. *J. Syst. Softw.*, 60(3):165–175, 2002.
- [39] M. Hossain and S. M. Bridges. A framework for an adaptive intrusion detection system with data mining. In *Proceedings of the 13th Annual Canadian Information Technology Security Symposium*, Ottawa, Canada, June 2001.
- [40] W. Hu, Y. Liao, and V. R. Vemuri. Robust support vector machines for anomaly detection in computer security. In *Proceedings of international Conference of Machine Learning and Applications*, Los Angeles, CA, 2003.
- [41] J. Rowe I. Balepin, S. Maltsev and K. Levitt. Using specification-based intrusion detection for automated response. In *Proceedings of 6th International Symposium, RAID 2003, Recent Advances in Intrusion Detection*, Pittsburgh, PA, 2003.
- [42] C. Iheagwara. The effect of intrusion detection management methods on the return on investment. *Computers & Security*, 23(3):213–228, 2004.
- [43] J.-S. R. Jang, C.-T. Sun, and E. Mizutani. *Neuro-Fuzzy and Soft Computing: A computational approach to learning and machine intelligence*. Prentice Hall, 1997.

- [44] H. S. Javitz and A. Valdes. The nides statistical component description and justification. Technical report, Computer Science Laboratory, SRI International, Menlo Park, CA, March 1994.
- [45] T. Joachims. Estimating the generalization performance of a SVM efficiently. In Pat Langley, editor, *Proceedings of ICML-00, 17th International Conference on Machine Learning*, pages 431–438, San Francisco, CA, 2000. Morgan Kaufmann.
- [46] K. Julisch. Clustering intrusion detection alarms to support root cause analysis. *ACM Trans. Inf. Syst. Secur.*, 6(4):443–471, 2003.
- [47] N. Kasabov. Evolving fuzzy neural networks for supervised/unsupervised on-line, knowledge-based learning. *IEEE Trans. on Man, Machine and Cybernetics–Part B: Cybernetics*, 31(6):902–918, December 2001.
- [48] N. Kasabov. *Evolving Connectionist Systems: Methods and Applications in Bioinformatics, Brain Study and Intelligence Machines*. Springer, 2002.
- [49] R. A. Kemmerer. NSTAT: A model-based real-time network intrusion detection system. Technical Report TRCS97-18, 17 1998.
- [50] K. Kendall. A database of computer attacks for the evaluation of intrusion detection systems, 1998.
- [51] J. O. Kephart, G. B. Sorkin, W. C. Arnold, D. M. Chess, G. J. Tesauro, and S. R. White. Biologically inspired defenses against computer viruses. In Morgan Kaufmann, editor, *Proceedings of IJCAI*, San Francisco, 1995.
- [52] C. Ko, G. Fink, and K. Levitt. Automated detection of vulnerabilities in privileged programs by execution monitoring. In *Proceedings of 10th Annual Computer Security Applications Conference*, pages 134–144, 1994.
- [53] V. Koltchinskii and D. Panchenko. Rademacher processes and bounding the risk of function learning. In Evarist Gine, David M. Mason, and Jon A. Wellner, editors, *High Dimensional Probability II*, pages 443–459, 2000.
- [54] C. Kruegel, D. Mutz, W. Robertson, and F. Valeur. Bayesian event classification for intrusion detection. In *Proceedings of Annual Computer Security Applications Conference*, Las Vegas, NV, 2003.
- [55] C. Kruegel and T. Toth. Using decision trees to improve signature-based intrusion detection. In *Proceedings of Recent Advances in Intrusion Detection (RAID 2003)*, 2003.
- [56] J. T.-Y. Kwok. Automatic text categorization using support vector machine. In *Proceedings of International Conference on Neural Information Processing*, pages 347–351, 1998.
- [57] T. Lane. *Machine Learning Techniques for the computer security domain of anomaly detection*. PhD thesis, Purdue University, West Lafayette, IN, 2000.
- [58] T. Lane and C. E. Brodley. Approaches to online learning and concept drift for user identification in computer security. In *Knowledge Discovery and Data Mining*, pages 259–263, 1998.

- [59] T. Lane and C. E. Brodley. Temporal sequence learning and data reduction for anomaly detection. *ACM Trans. Inf. Syst. Secur.*, 2(3):295–331, 1999.
- [60] A. Lazarevic, L. Ertöz, V. Kumar, A. Ozgur, and J. Srivastava. A comparative study of anomaly detection schemes in network intrusion detection. In *Proceedings of the 3rd SIAM International Conference on Data Mining*, pages 25–36, San Francisco, CA, 2003.
- [61] W. Lee, W. Fan, M. Miller, S. J. Stolfo, and E. Zadok. Toward cost-sensitive modeling for intrusion detection and response. *J. Comput. Secur.*, 10(1-2):5–22, 2002.
- [62] W. Lee, S. Stolfo, and P. Chan. Learning patterns from unix process execution traces for intrusion detection. In *Proceedings of the AAAI97 workshop on AI methods in Fraud and risk management*, pages 50–56, July 1997.
- [63] W. Lee and S. J. Stolfo. A framework for constructing features and models for intrusion detection systems. *ACM Trans. Inf. Syst. Secur.*, 3(4):227–261, 2000.
- [64] W. Lee, S. J. Stolfo, and K. W. Mok. Adaptive intrusion detection: A data mining approach. *Artif. Intell. Rev.*, 14(6):533–567, 2000.
- [65] W. Lee and D. Xiang. Information-theoretic measures for anomaly detection. In *Proc. of the 2001 IEEE Symposium on Security and Privacy*, pages 130–143, May 2001.
- [66] C. Leslie, E. Eskin, A. Cohen, J. Weston, and W. S. Noble. The spectrum kernel: A string kernel for svm protein classification. In *Proceedings of the Pacific Symposium on Biocomputing 2002 (PSB 2002)*, pages 564–575, January 2002.
- [67] Y. Liao and V. R. Vemuri. Use of k-nearest neighbor classifier for intrusion detection. *Computers and Security*, 21(5):439–448, 2002.
- [68] Y. Liao and V. R. Vemuri. Using text categorization techniques for intrusion detection. In *Proceedings of the 11th USENIX Security Symposium*, pages 51–59. USENIX Association, 2002.
- [69] R. Lippmann, D. Fried, I. Graf, J. Haines, K. Kendall, D. McClung, D. Webber, S. Webster, D. Wyszograd, R. Cunningham, and M. Zissan. Evaluating intrusion detection systems: the 1998 darpa off-line intrusion detection evaluation. In *Proceedings of the DARPA Information Survivability Conference and Exposition*, pages 12–26, Los Alamitos, CA, 2000. IEEE Computer Society Press.
- [70] R. Lippmann, J. W. Haines, D. J. Fried, J. Korba, and K. Das. Analysis and results of the 1999 darpa off-line intrusion detection evaluation. In *RAID '00: Proceedings of the Third International Workshop on Recent Advances in Intrusion Detection*, pages 162–182, London, UK, 2000. Springer-Verlag.
- [71] P. Liu and W. Zang. Incentive-based modeling and inference of attacker intent, objectives, and strategies. In *Proceedings of ACM Conference on Computer and Communications Security (CCS' 03)*, pages 179–189, 2003.
- [72] T. F. Lunt. Detecting intruders in computer systems. In *Proceedings of Conference on Auditing and Computer Technology*, 1993.

- [73] A. Lunts and V. Brailovsky. Evaluation of attributes obtained in statistical decision rules. *Engineering Cybernetics*, 3:98–109, 1967.
- [74] K. Lye and J. Wing. Game strategies in network security. In *Proceedings of 15th IEEE Workshop on Foundations of Computer Security (FCS '02)*, Copenhagen, Denmark, July 2002.
- [75] M. V. Mahoney and P. K. Chan. Learning nonstationary models of normal network traffic for detecting novel attacks. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 376–385. ACM Press, 2002.
- [76] M. V. Mahoney and P. K. Chan. An analysis of the 1999 darpa/lincoln laboratory evaluation data for network anomaly detection. In *Proceedings of the 6th Intl. Symp. Recent Advances in Intrusion Detection (RAID)*, 2003.
- [77] M. Markou and S. Singh. Novelty detection: a review. *Signal Processing*, 83:2481–2521, 2003.
- [78] K. Matsuura. Information security and economics in computer networks: An interdisciplinary survey and a proposal of integrated optimization of investment. In *Proceedings of the 9th International Conference of Computing in Economics and Finance (CEF 2003)*, Seattle, WA, July 2003.
- [79] R. A. Maxion. Masquerade detection using enriched command lines. In *Proc. Int. Conf. Dependable Systems and Networks*, San Francisco, CA, 2003.
- [80] R. A. Maxion and T. N. Townsend. Masquerade detection using truncated command lines. In *Proceedings of International Conference on Dependable Systems & Networks*, pages 219–228, Washington DC, June 2002.
- [81] J. McHugh. Testing intrusion detection systems: A critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory. *ACM Trans. Information and System Security*, 3(4):262–294, 2000.
- [82] J. McHugh. Intrusion and intrusion detection. *International Journal of Information Security*, 1(1):14–35, 2001.
- [83] L. M'e and C. Michel. Intrusion detection: A bibliography. Technical Report SSIR-2001-01, Sup'elec, Rennes, France, September 2001.
- [84] Sun Microsystems. *SunShield Basic Security Module Guide*. 1995.
- [85] S. Mukherjee, P. Tamayo, S. Rogers, R. Rifkin, A. Engle, C. Campbell, T. R. Golub, and J. P. Mesirov. Estimating dataset size requirements for classifying dna microarray data. *Journal of Computational Biology*, 10:119–142, November 2003.
- [86] S. Mukkamala, G. Janowski, and A. H. Sung. Intrusion detection using neural networks and support vector machines. In *Proceedings of Hybrid Information Systems Advances in Soft Computing*, pages 121–138. Springer Verlag, 2001.
- [87] University of New Mexico Computer Immune System Data Sets. <http://www.cs.unm.edu/%7eimmsec/data-sets.htm>.

- [88] M. Oka, Y. Oyama, H. Abe, and K. Kato. Anomaly detection using layered networks based on eigen co-occurrence matrix. In *Proceedings of 7th International Symposium on Recent Advances in Intrusion Detection (RAID 2004)*, pages 223–237, Sophia Antipolis, French Riviera, France, 2004.
- [89] The Third Annual Workshop on Economics and Information Security (WEIS04). <http://www.dtc.umn.edu/weis2004/>.
- [90] M. Opper and W. Kinzel. *Statistical Mechanics of Generalisation in Models of Neural Networks*. Springer Verlag, Heidelberg, 1995.
- [91] V. Paxson. Bro: a system for detecting network intruders in real-time. *Computer Networks (Amsterdam, Netherlands: 1999)*, 31(23–24):2435–2463, 1999.
- [92] P. A. Porras and P. G. Neumann. EMERALD: Event monitoring enabling responses to anomalous live disturbances. In *Proc. 20th NIST-NCSC National Information Systems Security Conference*, pages 353–365, 1997.
- [93] B. Scholkopf, J. C. Platt, J. Schawe-Taylor, A. J. Smola, and R. C. Williamson. Estimating the support of a high-dimensional distribution. *Neural Computation*, 13(7):1443–1471, 2001.
- [94] M. Schonlau, W. DuMouchel, W.-H. Ju, A. F. Karr, M. Theus, and Y. Vardi. Computer intrusion: Detecting masquerades. *Statistical Science*, 16(1):33–38, November 2000.
- [95] K. Sequeira and M. J. Zaki. Admit: Anomaly-based data mining for intrusions. In *Proceedings of SIGKDD*, pages 386–395, 2002.
- [96] J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.
- [97] K. M. C. Tan and R. A. Maxion. "why 6?" defining the operational limits of stide, an anomaly-based intrusion detector. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, page 188. IEEE Computer Society, 2002.
- [98] K. MC Tan, K. S. Killourhy, and R. A. Maxion. Undermining an anomaly-based intrusion detection system using common exploits. In *Proceedings of 5th International Symposium, RAID 2002, Recent Advances in Intrusion Detection*, pages 54–73, 2002.
- [99] D. M.J. Tax and R. P.W. Duin. Support vector data description. *Machine Learning*, 54:45–66, 2004.
- [100] H. S. Teng, K. Chen, and S. C. Lu. Adaptive real-time anomaly detection using inductively generated sequential patterns. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 278–284, Oakland, CA, May 1990.
- [101] P. Uppuluri and R. Sekar. Experiences with specification-based intrusion detection. In *Proceedings of Recent Advances in Intrusion Detection (RAID 2001)*, pages 172–189, 2001.
- [102] H.S. Vaccaro and G.E. Liepins. Detection of anomalous computer session activity. In *Proceedings of the 1989 IEEE Symposium on Security and Privacy*, pages 280–289. IEEE Computer Society, 1989.

- [103] A. Valdes. Detecting novel scans through pattern anomaly detection. In *Proceedings of DARPA Information Survivability Conference and Exposition*, pages 140–151, Washington, DC, April 2003.
- [104] V. Vapnik. *Statistical Learning Theory*. Wiley, New York, 1998.
- [105] D. Wagner and P. Soto. Mimicry attacks on host-based intrusion detection systems. In *ACM Conference on Computer and Communications Security*, pages 255–264, 2002.
- [106] C. Warrender, S. Forrest, and B. Pearlmutter. Detecting intrusions using system calls: Alternative data models. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 133–145, Oakland, CA, May 1999.
- [107] K. Yamanishi, J. Takeuchi, and G. Williams. On-line unsupervised outlier detection using finite mixtures with discounting learning algorithms. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 320–324, Boston, MA, August 2000.
- [108] K. Yamanishi, J. Takeuchi, and G. Williams. On-line unsupervised outlier detection using finite mixtures with discounting learning algorithms. *Data Mining and Knowledge Discovery*, 8:275–300, 2004.
- [109] Y. Yang. Expert network: Effective and efficient learning from human decisions in text categorization and retrieval. In *Proceedings of 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'94)*, pages 13–22, 1994.