Log Analysis-Based Intrusion Detection via Unsupervised Learning

Pingchuan Ma

Master of Science

School of Informatics

University of Edinburgh

2003

Abstract

Keeping networks secure has never been such an imperative task as today. Threats come from hardware failures, software flaws, tentative probing and malicious attacks. Analyzing network logs to detect suspicious activities is one form of defense. However, the sheer size of network logs makes human log analysis intractable. Furthermore, traditional intrusion detection methods based on pattern-matching techniques cannot cope with the need for faster speed to manually update those patterns.

This project aims to build an intelligent log analyzer that can detect known and unknown network intrusions automatically. Under a data mining framework, the log analyzer is trained with unsupervised learning algorithms, namely the k-means algorithm and Autoclass.

Based on these unsupervised learning algorithms, three novel intrusion detection methods are proposed and tested. Much higher detection rates are obtained with reasonable true positive rates, when compared to the best results obtained on the KDD1999 dataset. Moreover, this log analyzer is modularized so as to simplify the incorporation of new algorithms when necessary.

Acknowledgements

I would like to thank my supervisor, Qiang Shen, for helping and guiding me through this project. Thanks also go to Alexios Chouchoulas, Michelle Galea, Jeroen Keppens, Joe Halliwell for their enlightening comments and kind support.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Pingchuan Ma*)

Table of Contents

Chapter 1

Introduction

1.1 Overview

With the fast development in IT technology, it is cheaper and easier to develop and deploy computer networks of all shapes and sizes. Unfortunately, it is also cheaper and easier to probe and attack our networks. Therefore keeping our networks secure becomes vitally important.

The threats that network systems are confronting come from failures of hardware or software, tentative probing and malicious attacks from local or remote hackers. Recording network logs to monitor what has been going on is one form of defense against system failures and human attacks.

A moderate to large network tends to collect sheer size of network activities, generating huge log files, which make human inspection impossible. Traditionally, most log analyzers on the market are based on pattern-matching techniques. They compare the log entries to sets of pre-defined patterns. These sets of pre-defined patterns have to be manually updated frequently by security experts to handle all kinds of attacks they know so far. Apparently, those emerging attacks could easily out-pace the updating speed for those patterns.

From this point of view, current log analyzers are far from intelligent in that they solely rely on human intervention to operate effectively. Therefore, a more advanced log analysis tool is highly desired. It should be capable of detecting known and unknown intrusions intelligently and automatically, distinguishing normal network activities from those abnormal (very possibly malicious) ones with minimum human inputs.

1.2 Motivation

Recently, some approaches using data mining algorithms are applied to log analysis in the intrusion detection community. Those algorithms are based on supervised learning. That is to say, they are trained, instead of programmed, on data sets with labels indicating whether the instances are pre-classified as attacks or not. However, manually labelling the large volumes of network data (It is not unusual to see a network log data set larger than 1 gigabytes) is difficult and extremely expensive, due to prohibitive human labor costs. This renders supervised learning hard to apply or its performance badly impaired due to the lack of well labeled training data.

With no requirement for class labels, unsupervised learning algorithms may shed some light on this problem. Although it seems that not much work has been done in this direction, unsupervised learning algorithms can be the ideal choice because of their ability to ”grasp the true qualitative nature of a data set without class labels [15]”. When a system becomes ”familiar” with the data through unsupervised learning, it may detect ”abnormal” data when they come in. Very likely those ”abnormal” data are network attacks.

This project explores the potential of unsupervised learning, especially of Bayesian Clustering [14], in building a novel log analysis tool that is truly ”intelligent” in detecting those known and unknown network intrusions.

1.3 Project Objectives

This project aims to design and implement a system that can learn the ”normalities” of a network system and dynamically identify abnormal entries, so that they may be brought to the attention of a human auditor. The strategy and supporting objectives are described as follows:

Modularity

Modularity is desired so that newer learning algorithms could be easily added when deemed more promising.

Our system is designed in modules which communicate through dataset files in standard data format. For more details, please refer to Chapter 4.

Learning and Detection

Based on unsupervised learning, detecting abnormal activities shall be executed automatically without too much human intervention.

Thanks to the modularized design, experimenting with unsupervised learning algorithms and detection mechanisms is made easy. Autoclass [6], the implementation of Bayesian Clustering [14], is our major learning engine. The k-means algorithms, a well-known clustering algorithm, is also tried and compared with Autoclass.

Generality

The log analyzer should be as general purpose as possible. It will be very beneficial if our log analyzer can handle logs in other formats gathered from different running environments.

For this purpose, a data set driven approach is taken. This means that the Learning and Detection modules are fed with a pre-defined data set, which is automatically transformed from original logs to a representative dataset format.

1.4 Thesis Organization

The rest of this thesis follows the structure listed below

Background and Related Work

In chapter 2, a broad background regarding this project is introduced. This includes the seriousness of network intrusion problem, the format of logs, a review of today's intrusion detection methods, data mining approaches that are to be employed, some novelty detection approaches. At section 2.6, a review will be given to traditional log analyzers, data mining related work and some previous work based on text categorization and feature reduction.

Theoretical Aspects

Theoretical aspects are discussed in chapter 3. In more detail, the theoretical foundation of Autoclass [6] will be presented. Other unsupervised and supervised learning algorithms will be briefly introduced. Moreover, the basic assumptions made when designing this log analyzer and how network attacks are to be detected will also be discussed in this chapter.

System Implementation

Chapter 4 will describe the proposed system implementation. Starting with the system design philosophy, it will describe each module with a detailed specification. Those data mining packages we have incorporated in our system are briefly introduced and commented on at the end of this chapter.

Experimental results and analysis

Chapter 5 demonstrates the performance of the implemented log analyzer with regards to its unsupervised learning algorithms and intrusion detection methods. The log analyzer has been tested on a network intrusion dataset, showing promising results. Also detailed analysis of results are given thereafter.

Conclusion and future works

Chapter 6 will given conclusion, assessing the successes and limitations of log analyzer. Some future directions are also discussed.

Chapter 2

Background

Designing an intelligent network log analyzer involves with a broad range of knowledge, namely network security, data mining learning algorithms and some novelty detection approaches.

In this chapter, firstly an introduction will point out the seriousness of the network security problem. Afterwards, some conventional network intrusion detection methods are briefly discussed before data mining based approaches in the log analysis are introduced. In the next section the topic of novelty detection is covered, which links closely to the detection of network intrusions. Finally, some related work will be reviewed.

2.1 Network Intrusion

The past two decades have seen information technology growing with unprecedented speed. Computer networks of all shape and sizes are becoming ubiquitous. This trend brings the subject of network security into focus, which is, ensuring the system to behave as intended and to provide stable services.

Threats to network systems come typically from the malfunction of hardware or software, or through malicious behavior by users of software. Promptly resolving network incidents is very important, considering the huge costs of data loss and system downtime.

The abundance of computational resources makes lives of computer hackers easier. Without much effort, they can acquire detailed descriptions of system vulnerabilities and exploits to initiate attacks accordingly. Statistics from CERT® Coordination Center (CERT/CC) [4], the most influential reporting center for internet security problems, show that there was a dramatic increase of reported network incidents to CERT/CC from 1988 to 2002, as illustrated in figure 2.1. This trend is expected to continue, as the number of incidents in the first two quarters of 2003 has reached 76,404, nearly the total number of last year.

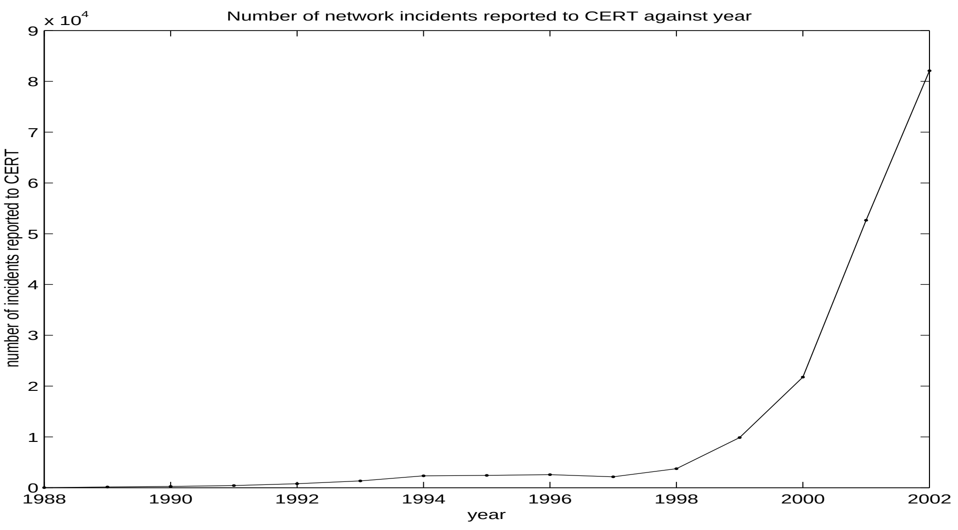


Figure 2.1: The number of incidents reported to CERT/CC from year 1988 to 2002 increased dramatically. The statistics are from http://www.cert.org/stats/cert_stats.html#incidents (Please note that this reference is from the web, which may not be authoritative enough)

2.2 Logs

To protect network systems from attacks, a common approach is to record network logs to monitor all those prominent activities. Each time a noticeable event happens in the network systems, an entry will be appended to a log file, in the form of plain text or binary format. Take web log files as an example. Every ”hit” to a web site, including requests for HTML pages as well as images, is logged as one line of text in a log file. This records information about who is visiting, where they are from and what they are doing with the web server. Below is a sample,

```
46.53.200.22 - - [ 22/Jun/2003:07:50:43 +0100 ] ”GET /themes/New_Default/newtitle-
logo.png HTTP/1.0” 200 10950 ”http://www.machine-room.org/computers/397/” ”Mozilla/4.0
(compatible; MSIE 6.0; Windows NT 5.1; .NET CLR 1.0.3705; .NET CLR 1.1.4322)”
mod_gzip: - In:- Out:- :-pct. vh:www.machine-room.org
```

Table 2.1 gives an explanation for what this line of log file tells us.

Interpretation	Content
Visitor’s IP address	46.53.200.22
Login	-
Authuser	-
Date and time	22/Jun/2003:07:50:43 +0100
Request method	GET
Request path	/themes/New_Default/newtitle-logo.png
Request protocol	HTTP/1.0
Response status	200
Response content size	10950
Referrer path	http://www.machine-room.org/computers/397/
User agent	Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR 1.0.3705; .NET CLR 1.1.4322)

Table 2.1: Interpretation for an entry in a httpd log file

An experienced network administrator may take a quick glance at those logs and realize instantly what has happened. However, it is almost impossible for a human to check those logs when the log files have accumulated thousands if not millions of entries. It is typical for a network system to collect over 1 gigabyte of data in a month. Naturally, appropriate methods are needed to remove irrelevant information and extract the most interesting. What is required, therefore, is a log analyzer intelligent enough to automatically detect those abnormal activities in the logs without too much human inputs.

2.3 Intrusion Detection Methods

There have been several log analysis tools on the market. The intrusion detection methods they have been using are categorized by [29] as follows:

Pattern Matching examines the content of network traffic (in real-time network intrusion detection systems) or log file (in log analyzers) to look for a sequence of bytes as the pattern to match. This approach is rigid but simple to implement and therefore is widely used.

Stateful Pattern Matching performs pattern matching within the context of a whole data stream instead of just looking into current atomic packets.

Protocol Decode-Based Analysis makes extensions to the stateful pattern matching method in that it tries to find out the violations against the rules that are defined by the Internet standards.

Heuristic-Based Analysis makes decisions based on pre-programmed algorithmic logic. Those algorithms are often the statistical evaluations of the network traffic content.

Anomaly Detection tries to find out anomalous actions based on the learning of its previous training experience with patterns assumed as normal.

The first four methods are widely used in industry practices. However, most of these pattern-matching based detectors can only deal with already-known intrusions that have been recognized by the security experts. Unfortunately, ill-intentioned hackers are aware of those patterns too. When new attack patterns emerge, very likely they could evade the detection by deliberately avoiding those wide publicized matching patterns. The potential damages caused by those attacks are substantial.

With regard to network attacks that become more cunning, more variant, and hence much more dangerous, it is hard to imagine that human-maintained pattern-matchers could be updated quickly enough. Data mining approaches, armed with machine learning algorithms, may come to the rescue.

2.4 Data Mining Approaches

According to the definition given by [13], ”Data Mining is the analysis of (often large) observational data sets to find unsuspected relationships and to summarize the data in novel ways that are both understandable and useful to the data owner.” During the process of data mining, many machine learning algorithms are available for choosing. Depending on whether the class labels are provided for learning, these machine learning algorithms can be classified as either supervised or unsupervised.

2.4.1 Supervised learning

Trained with data bearing class labels indicating to which subcategories they belong or what real-valued properties they have, a supervised learning algorithm tries to predict the most likely labels for new test data. There are two major subcategories for supervised learning:

Classification is to predict the class membership as one of a finite number of discrete labels.

Regression is to predict the output value as one of a potentially infinite set of real-valued points.

There are many widely used supervised classification techniques. They include but not are limited to Support Vector Machines (SVMs), Decision Trees, Neural Networks, Naive Bayes, Nearest Neighbour and Regression models. For example, based on a Naive Bayes classifier, trained with a data set with virus labels on file headers, [27] have built an automatic email filter that detects malicious Windows executables coming through the email system.

2.4.2 Unsupervised learning

In unsupervised learning, the data are not labelled, which makes it hard to tell what counts as good. [15] think that ”it is less natural, but much more revealing, to view unsupervised learning as supervised learning in which the observed data is the output and for which there is no input”. The model generating the output must either be stochastic or must have an unknown and varying input in order to avoid producing the same output every time. From this point of view, the aim of unsupervised learning could be regarded as to ”fit a generative model that gives a high likelihood to the observed data [15]”.

From the perspective of machine learning, the searching for clusters is unsupervised learning. To perform clustering is to try to discover the inner nature of the data structure as a whole, and to divide the data into groups of similarity. From the viewpoint of data mining, clustering is the partitioning of a data set into groups so that the points in the group are similar as possible to each other and as different as possible from points in other groups.

There are generally three types of clustering algorithms

Partition-based clustering

Given a predefined number of clusters, find the optimal partitions for each point. Choose the centres so as to minimize the summed distance

$$\sum_{i=1}^{n} |x_i - m_{c(i)}|^2$$

where $c(i)$ stands for the cluster to which data x_i is assigned.

The k-means algorithm is a well-known example of this kind of clustering methods. we will present it in detail in the next chapter.

Hierarchical clustering

As stated by [2], hierarchical clustering builds a cluster hierarchy. The hierarchy is a tree of clusters. Every node in the tree contains child clusters while sibling clusters share a common parent node. Depending on how the tree is formed, hierarchical clustering methods fall in two categories, agglomerative and divisive. Agglomerative methods recursively merge points while divisive methods start from a cluster of all data and then gradually split them into smaller clusters.

Probabilistic based clustering

Assume the data come from a multivariate finite mixture model with probability in the form of

$$p(x) = \sum_{k=1}^{K} \pi_k f_k(x; \theta_k)$$

where π_k is the class component prior probability and $f_k(x; \theta_k)$ is class conditional density function, θ_k is its model parameters. Use the Expectation Maximization (EM) algorithm ([1] and [8]) to find the model parameters in $f_k(x; \theta_k)$ and class component prior probability π_k from the data. Once all of them are found, assign the each data point x to the cluster to which it belongs with the highest probability $p(x)$. Autoclass [6] belongs to this category of clustering.

2.5 Novelty Detection

According to [19], novelty detection is ”the identification of new or unknown data or signal that a machine learning system is not aware of during training. It is one of the fundamental requirements of a good classification or identification system since sometimes the test data contains information about objects that were not known at the time of model training”.

Anomaly could be regarded as one kind of novelty. Normally, our classifiers are expected to give reliable results when the test data are similar to those used during training. However, the real world is totally different, when abnormal data come in, picking them out is a problem. Compared to conventional 2-class classification problem, an anomaly detection system is trained with only normal patterns and then try to predict those abnormal data based solely on the models built from normal data.

There exist a variety of methods of novelty detection that have been shown to perform well on different data sets. [19] claimed that ”there is no single best model for novelty detection and success depends not only on the type of method used but also statistical properties of data handled”. It is true, during the experimentation on different learning and detecting methods in our log analyzer, we found that some models work and some do not. Below is a brief description of those approaches given by [19].

2.5.1 Probabilistic/GMM approaches

This category of approaches is based on statistical modelling of data and then estimating whether the test data come from the same distribution that generates the training data. First estimate the density function of the training data. By assuming the training data is normal, the probability that the test data belong to that class can be computed. A threshold can then be set to signal the novelty if the probability calculated is lower than that threshold.

For Gaussian Mixture Modelling (GMM) models, the parameters of the model are chosen by maximizing the log likelihood of the training data with respect to the model. This task could be done using re-estimation techniques such as EM algorithm. However, GMM suffers from the curse of dimensionality in the sense that if the dimensionality of the data is high, a very large number of samples are needed to train the model, which makes the computation even harder.

A much simpler way is to just find the distance of test data from the class mean and set a threshold for the variance. If the test data is far away from the mean plus the variance threshold then it can be claimed to be novel.

2.5.2 Non-parametric approaches

For non-parametric methods, the overall form of the density function is estimated from the data as well as parameters of the model. Therefore non-parametric methods do not require extensive prior knowledge of the problem and do not have to make assumptions on the form of data distribution, which means that they are more flexible though much more computational demanding.

2.5.2.1 K-nearest neighbour approaches

The k-nearest neighbour algorithm is another technique for estimating the density function of data. ”This technique overcomes some of the problems of Parzen window [22] in that it does not require a smoothing parameter. Instead, the width parameter is set as a result of the position of the data point in relation to other data points by considering the k-nearest data in the training set to the test data.

For novelty detection the distribution of normal vectors is described by a small number of spherical clusters placed by the k-nearest neighbour technique. Novelty is assessed by measuring the normalised distance of a test sample from the cluster centres. [19]”

2.5.2.2 String matching approaches

String matching approaches is biologically inspired by studying how the immune system works. [11] present a method for solving the problem of distinguishing self from non-self using a change-detection algorithm which is based on the generation of T cells in the immune system. Treating training data as templates, which are represented by a string (vector of features), they could then compute some measure of dissimilarity between training and test data. The self-data is converted to binary format forming a collection S. Then a large number of random strings are generated forming a set R_0. Strings from R_0 are matched against the strings in S and those that match are eliminated.

Since perfect matching is extremely rare, the matching criterion is relaxed so as to consider only r contiguous matches in the strings. Once R_0 is created, new patterns are converted to binary and matched against R_0. If a match is found, then new pattern belongs to non-self and is rejected. The major limitation appears to be the computational difficulty of generating the initial repertoire. This method has been applied on the detection of computer virus and claimed some good results.

2.5.3 Neural network based approaches

Quite a number of different architectures of neural networks are applied to novelty detection. A neural network can detect novelty by setting a threshold on the output values of the network. Or it can calculate the Euclidean distance between output patterns and target patterns and throw those with highest distance out as the novelty. [25] have built such a system to identify legitimate users based on the commands they have entered. Their neural networks are trained with back-propagation algorithm. When their networks generate maximum activation lower than 0.5, a novelty is then declared.

2.6 Related Work

The related work will be separated into 3 sections.

2.6.1 Traditional log analyzers

Snort

Snort [28] is a successful light-weight, open-source network intrusion detector with log analyzer. With a huge rule set maintained by diligent experts, it could detect almost all the known attacks by a large rule set that is programmed within.

WebTrends Log Analyzer

Webtrends is a representative example of commercial log analyzer. It can generate many types of statistics, including

- General statistics (number of hits, page views and average length of user sessions as well as the users)
- Resources accessed
- Visitors and demographics
- Activity statistics
- Technical statistics (errors)
- Referrers and keywords
- Browsers and platforms (most used browsers and platform)

Such tools are often referred to as log analyzers, although they are only statistics reporters, and have no learning ability at all.

2.6.2 Data mining related work

General and Systematic methods for intrusion detection

[18] have developed some general and systematic methods for intrusion detection.

They have built a framework using data mining techniques to discover consistent and useful patterns of system features that describe program and user behavior, To detect anomalies and known intrusions, they have used a set of relevant system features to compute (with inductively learned) classifiers.

Two major data mining algorithms they have implemented are: the association rules algorithm and the frequent episodes algorithm, which are used to recognize intra- and inter- audit record patterns.

To meet the challenges of both efficient learning (mining) and real-time detection, they have proposed an agent-based architecture in which the learning agents continuously compute and provide the updated models, while a detection agent is equipped with a (learned and periodically updated) rule set from the remote learning agent.

”Artificial Anomalies”

In [10], the authors have proposed an algorithm to generate artificial anomalies to force the inductive learner to find out a more accurate boundary between known classes (normal connections and known intrusions) and anomalies. Their experiment on the KDD99 data set shows that the model is capable of detecting more than 77% of all unknown intrusion classes with over 50% accuracy per intrusion class. However, the way to generate anomalies is not clear.

SmartSifter

The SmartSifter [31] is an outlier detection system based on unsupervised learning of the underlying mechanism for data generation. The mechanism is based on a probabilistic model which uses a finite mixture model.

Each new input datum is examined to see how much it has deviated from a normal pattern. At the same time, an on-line learning algorithm is employed to update the model. The datum is given a score showing how many changes have happened after learning. A high score means that the datum is an outlier.

Parzen-Window Network Intrusion Detector

Yeugn and his colleagues have built a nonparametric density estimation approach [33] based on Parzen-window estimators [22] to build an intrusion detection system using normal data only. Given a data set $D = x_1, x_2, ...x_n$, where x_n is individually independently distributed examples according to p(x), the Parzen-window estimate of p(x) is

$$\hat{p}(x) = \frac{1}{n}\sum_{i=1}^{n}\delta_n(x - x_i)$$

where $\delta_n(.)$ is a kernel function with localized support and its exact form depends on n. Here Gaussian kernel functions are used. So the $\hat{p}(x)$ can be expressed as

$$\hat{p}(x) = \frac{1}{n(2\pi)^{d/2}\sigma^d}\sum_{i=1}^{n} exp\left\{-\frac{\| x - x_i \|^2}{2\sigma^2}\right\}$$

where d is the dimensionality of the feature space, σ is the variance.

Denote w_1 as normality and w_0 as anomaly or novelty. The corresponding prior probabilities are $P(w_1)$ and $P(w_0)$ and the probability density functions are $p(x|w_1)$ and $p(x|w_0)$. According to Bayes Decision Rule,
$x \in w_1$ if and only if

$$P(w_1|x) = p(x|w_1)P(w_1) > P(w_0|x) = p(x|w_0)P(w_0)$$

For a given input x, deciding whether it is anomalous or novel depends on the comparison between $p(x|w_1)$ and $\frac{p(x|w_0)P(w_0)}{P(w_1)}$, where the latter is a threshold determined by a separate set of normal data.

Their results are bench-marked as TDR (the percentage of intrusive connection in the test set that are detected as intrusions). Under this measure, they could detect 93.57% u2r and 31.17% r2l attacks while KDD Winner can only find out 26.32% u2r and 10.27% attacks.

2.6.3 Previous work on text categorization and feature reduction

Email Categorisation

In [7], the authors have investigated the applicability of rough set theory to the information retrieval and filtering domain. They have built an email categorisation application which employs the QuickReduct algorithm to reduce the dimensionality while keeping the most predictive information as much as possible.

This system is modularized as training and testing datasets splitting, keyword acquisition, rough set dimensionality reduction and classifier. Our proposed system reuses a part of the Perl code in the keyword acquisition module to create dataset from the log text.

WWW Bookmark Categorisation

In [16], the author has built a WWW Bookmark Categorisation system based on Rough Set (RS) reduction. He has demonstrated that for this domain, RS is successful in reducing the dataset with most information content retained.

He also compared the RS with another dimensionality reduction method, Entropy-Based Reduction (EBR), with the finding that EBR could also provide similar good performance in this domain.

2.7 Summary

This chapter covers a broad range of knowledge that are involved when designing log analyzer for intrusion detection. In turn, network security problems, traditional intrusion detection approaches, data mining approaches and some novelty detection techniques are introduced as well as some related work.

Chapter 3

Theoretical Aspects

This chapter will continue the discussion on how an intelligent network log analyzer can be built. In more depth and focused points, it will present the theoretical aspects behind the algorithms incorporated in this system.

The first two sections will describe two unsupervised learning algorithms, Bayesian clustering and k-means clustering. The third and fourth sections are devoted to supervised learning algorithms, Naive Bayes Classifier and Decision Trees. All of the four algorithms are employed in our log analyzer. Next two sections will introduce how the logs text is vectorized and discuss issues regarding feature selection. Finally, three intrusion detection methods are proposed.

3.1 Bayesian Clustering and Autoclass Implementation

Autoclass [6] is a Bayesian Clustering program developed by Peter Cheeseman and his colleagues at NASA [5]. It automated the process of model selection as well as the process of parameter estimation. By calculating the approximation of the marginal density of data after the integration of the parameters, Autoclass compares different models and uses Ocam Razer to favor models with less complexity.

$X = \{X_1, ..., X_I\}$	the set data instances X_i
$\vec{X}_i = \{X_{i1}, ..., X_{iK}\}$	the vector of attribute values X_ik, describing instance X_i
i	indexes instances, $i = 1, ..., I$
j	indexes classes, $j = 1, ..., J$
k	indexes attributes, $k = 1, ..., K$
l	indexes discrete attribute values, $l = 1, ..., L$
c	indicates inter-class probabilities & parameters
S	denotes the space of allowed p.d.f.'s $\vec{V}$,T
$T = T_c, T_1, ..., T_J$	denotes the abstract mathematical form of the p.d.f.
$\vec{V} = \vec{V}_c, \vec{V}_1, ..., \vec{V}_J$	denotes the set of parameter values instantiating a p.d.f.
π_j	class mixture probability, $\vec{V}_c = \{\pi_1, ..., \pi_J\}$
I	implicit information not specifically represented

Table 3.1: Notations used for describing Autoclass models, from [5]

3.1.1 Notation

Table (3.1) shows the notation used in Autoclass implementation.

3.1.2 Assumption

Autoclass assumes that all the data instances X_i are conditionally independent given the classification p.d.f. $\vec{V}, T$. Under this assumption, the class membership j is thought to be the only reason accounting for the features and there is no interaction between them. (in figure 3.1, the class membership variable is the parent node of all other children nodes) Therefore the joint data probability is the product of individual instance probabilities.

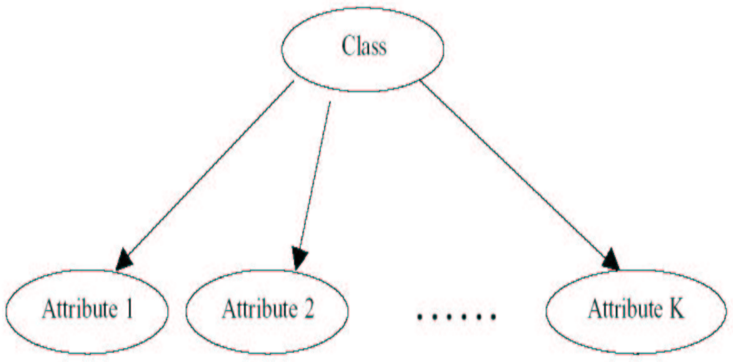


Figure 3.1: In the Bayesian approach to unsupervised classification, the goal is to find the most probable class labels given the data and prior. the fundamental model in Autoclass is the classical finite mixture model ([9] and [8])

3.1.3 Basic model

For the classical finite mixture model in Autoclass, each instance, with a probability $P(X_i \in C_j|\vec{V}_c, T_c, S, I)$, will be assigned as the member of a set of J classes C_j.

$$P(X_i \in C_j|\vec{V}_c, T_c, S, I) \equiv \pi_j \tag{3.1}$$

where the parameters $\vec{V}_c$ are a set of probabilities $\{\pi_1, \ldots, \pi_J\}$, with the constraints that $0 \leq \pi_j \leq 1$ and $\sum_j \pi_j = 1$. As the classes Autoclass have found constitute a discrete partitioning of the data, Dirichlet distribution is assumed as the prior probability distribution on the π_j:

$$P(\pi_1, \ldots, \pi_J|T_c, S, I) \equiv \frac{\Gamma(J+1)}{[\Gamma(1+1/J)]^J} \prod_j \pi_j^{\frac{1}{J}} \tag{3.2}$$

Then the class distribution function is the product of distributions which model the conditionally independent attributes k

$$P(\vec{X}_i|X_i \in C_j, \vec{V}_j, T_j, S, I) = \prod_k P(X_{ik}|X_i \in C_j, \vec{V}_{jk}, T_{jk}, S, I) \tag{3.3}$$

where $P(X_{ik}|X_i \in C_j, \vec{V}_{jk}, T_{jk}, S, I)$ are the individual attribute models which we can model as Bernoulli or Poisson distributions for nominal attributes, Gaussian distribution for real value attributes.

Hence the probability that an instance X_i with attribute values $\vec{X}_i$ comes from the class C_j is

$$P(\vec{X}_i, X_i \in C_j|\vec{V}_j, T_j, \vec{V}_c, T_c, S, I) = \prod_k \pi_j P(X_{ik}|X_i \in C_j, \vec{V}_{jk}, T_{jk}, S, I) \tag{3.4}$$

By introducing priors only on the parameters, the joint probability of the data and the parameter values are:

$$\begin{aligned} & P(X, \vec{V}|T, S, I) = P(\vec{V}|T, S, I)P(X|\vec{V}, T, S, I) \\ & = P(\vec{V}_c|T_c, S, I) \prod_{jk}[P(\vec{V}_{jk}|T_{jk}, S, I)] \prod_i [\sum_j (\pi_j \prod_k P(X_{ik}|X_i \in C_j, \vec{V}_{jk}, T_{jk}, S, I))] \end{aligned} \tag{3.5}$$

Apparently equation 3.5 is very hard to calculate directly. In Autoclass, an approximation approach is taken to address this problem. It will be discussed in section 3.1.5.2.

3.1.4 Attribute models

Under the Autoclass assumption, given each class, the likelihood of the data is a product of conditionally independent probability distribution over each single attribute or each subset of the attributes if the subset is related within. For different attribute types a few basic models are implemented in the current Autoclass version. They are listed below:

Discrete valued attributes Bernoulli distributions with uniform Dirichlet conjugate prior are modeled. For the single discrete attribute with L_k possible values, the parameters are $\vec{V}_{jk} \equiv \{q_{jk1}, \ldots, q_{jkL_k}\}$ under the constraints that $0 \leq q_{jkl} \leq 1$ and $\sum_l^{L_k} q_{jkl} = 1$ where

$$P(X_{ik} = l|X_i \in C_j, \vec{V}_{jk}, T_{jk}, S, I) \equiv q_{jkl} \tag{3.6}$$

$$P(q_{jk1}, \ldots, q_{jkL_k}|T_{jk}, S, I) \equiv \frac{\Gamma(L_k + 1)}{[\Gamma(1 + \frac{1}{L_k})]^{L_k}} \prod_{l=1}^{L_k} q_{jkl}^{\frac{1}{L_k}} \tag{3.7}$$

$$\hat{q}_{jkl} \equiv \frac{w_{jkl} + \frac{1}{L_k}}{w_j + 1} \tag{3.8}$$

Real valued location attributes Gaussian distribution is modelled. The prior on the means could be either uniform or Gaussian. For single attribute with uniform priors,

$$P(X_{ik}|X_i \in C_j, \mu_{jk}, \sigma_{jk}, T_{jk}, S, I) \equiv \frac{1}{\sqrt{2\pi}\sigma_{jk}} e^{-\frac{1}{2}\left(\frac{X_{ik}-\mu_{jk}}{\sigma_{jk}}\right)^2} \tag{3.9}$$

$$P(\mu_{jk}|T_{jk}, S, I) = \frac{1}{\mu_{k_{max}} - \mu_{k_{min}}}, \qquad \hat{\mu}_{jk} = m_{jk} \tag{3.10}$$

$$P(\sigma_{jk}|T_{jk}, S, I) = \sigma_{jk}^{-1}\left[log\frac{\sigma_{k_{max}}}{\sigma_{k_{min}}}\right]^{-1}, \qquad \hat{\sigma}_{jk}^2 = s_{jk}^2 \frac{w_j}{w_j + 1} \tag{3.11}$$

Real valued scalar attributes Log-Gaussian distribution are modelled by applying logarithm on X_{ik}

Missing values For discrete attributes, an additional attribute value marked as 'missing' is included in the model. For numerical attributes, a binary discrete probability q_{jk} is used to model the missing value and $1 - q_{jk}$ to model the known values, with which a Gaussian model conditioned on the 'known' side is used.

$$P(X_{ik} = missing|X_i \in C_j, q_{jk}, \mu_{jk}, \sigma_{jk}, T_{jk}, S, I) \equiv q_{jk} \tag{3.12}$$

$$P(X_{ik} = r|X_i \in C_j, q_{jk}, \mu_{jk}, \sigma_{jk}, T_{jk}, S, I) \equiv \frac{(1 - q_{jk})}{\sqrt{2\pi}\sigma_{jk}} e^{-\frac{1}{2}\left(\frac{r-\mu_{jk}}{\sigma_{jk}}\right)^2} \tag{3.13}$$

3.1.5 Search and evaluation

3.1.5.1 Search

Two things are sought during the searching, the MAP parameter values and the MAP model form conditional on the data. For the classification form $T = T_c, T_1, \ldots, T_J$ and data X, the MAP parameters' posterior probability distribution function is

$$P(\vec{V}|X, T, S, I) = \frac{P(X, \vec{V}|T, S, I)}{P(X|T, S, I)} = \frac{P(X, \vec{V}|T, S, I)}{\int d\vec{V} P(X, \vec{V}|T, S, I)} \tag{3.14}$$

While the MAP form T has the posterior probability distribution function:

$$P(T|X,S,I) = \frac{P(T|S,I)P(X|T,S,I)}{P(X|S,I)} \tag{3.15}$$

$P(X|S,I)$ is generally not computable. But because X is the same for all T, $P(X|S,I)$ can be dropped as a normalizing constant. It is allowed here because only the relative probabilities of models T are of interest. Therefore

$$P(T|X,S,I) \propto P(T|S,I)P(X|T,S,I) \tag{3.16}$$

where $P(T|S,I)$ is the prior probability of classification form T. As there is no special reason for us to favor this model instead of the other, we could assume this prior probability to be uniform and treat it as a single discrete probability. Then drop this term again and we will get

$$P(T|X,S,I) \propto P(X|T,S,I) \tag{3.17}$$

Hence to compare different classification form T we just need to compare $P(X|T,S,I)$.

3.1.5.2 EM algorithm

Directly optimizing over or integrate out the parameter sets $\vec{V}_{jk}$ in equation (3.5) is very hard as the product over sums requires J^I products. Without approximation, only data sets of very small size could be dealt with. Cheeseman and his colleagues addressed this problem by utilizing the mixture model assumption. When the true class memberships were known, $X_i' \in C_j$, the probability $P(X_i'|X_i' \in C_j, \vec{V}_j, T_j, S, I)$ would be zero when $X_i' \notin C_j$. If this assumption does hold, in equation (3.5) the sum over j could be simplified into a single non-zero term and then the equation (3.5) could be rewritten as

$$P(X', \vec{V}|T,S,I) = P(\vec{V}|T,S,I) \prod_j [\pi_j^{n_j} \prod_k P(X_{jk}''|\vec{V}_{jk}, T_{jk}, S, I))] \tag{3.18}$$

where n_j is the number of cases assigned to C_j, and the X_{jk}'' are sets of statistics according to attribute probability distribution function's T_{jk}, which comes from $X_i' \in C_j$.

In Autoclass, a variation of the expectation maximization(EM) algorithm ([1] and [8]) is used. An EM algorithm normally takes two steps, Expectation (E) step and Maximization (M) step.

Given the set of T_j, current MAP estimates of π_j and $\vec{V}_{jk}$, the normalized class conditional probabilities of equation (3.4) could be used to calculate the weighted assignments w_{ij} after normalized:

$$w_{ij} \equiv \frac{P(\vec{X}_i, X_i \in C_j | \vec{V}, T, S, I)}{\sum_j P(\vec{X}_i, X_i \in C_j | \vec{V}, T, S, I)} \propto \pi_j \prod_k P(X_{ik} | X_i \in C_j, \vec{V}_{jk}, T_{jk}, S, I) \tag{3.19}$$

Autoclass uses the weights to construct weighted statistics with respect to the known class case. For discrete attributes, w_{jkl} is the class weighted number of instances holding each discrete value. For a Gaussian modelled continuous attribute, the weighted statistics are the class weighted number, mean and variance:

$$w_j = \sum_i w_{ij}$$
$$m_{jk} = \frac{\sum_i w_{ij} X_{ik}}{w_j}$$
$$s_{jk}^2 = \frac{\sum_i w_{ij} (X_{ij} - m_{jk})^2}{w_j}$$

Autoclass then uses these statistics as if they stand for the known assignment statistics to re-estimate the parameters with the partitioning of equation (3.18). The newly estimated parameter set can then used to calculate the normalized probabilities. Cycling between the two steps will drive the current parameter and weight estimates towards a local maximum.

As the parameter space is normally too large to allow for a through search, Autoclass starts from pseudo-random points in the parameters space and cycles through the EM algorithm which converges to the local maximum, records the best results so far, and repeat this process for as long as it is allowed to run.

3.1.5.3 Evaluation

Given the local maxima that have been found so far during the EM algorithm, Autoclass uses some local statistics $X'' = \{w_j, X''_{jk}\}$ with:

$$P(X'' | T, S, I) \equiv \int d\vec{V} [P(\vec{V} | T, S, I) \prod_j (\pi_j^{w_j} \prod_k P(X''_{jk} | \vec{V}_{jk}, T_{jk}, S, I))] \tag{3.20}$$

$$P(X|T,S,L)^* \equiv P(X''|T,S,I)\frac{P(X|\hat{V},T,S,I)}{P(X''|\hat{V},T,S,I)} \tag{3.21}$$

where

$$\frac{P(X|\hat{V},T,S,I)}{P(X''|\hat{V},T,S,I)} = \frac{\prod_i[\sum_j(\hat{\pi}_j\prod_k P(X_{ik}|X_i \in C_j,\hat{V}_jk,T_{jk},S,I))]}{\prod_j(\hat{\pi}_j^{w_j}\prod_k P(X''_{jk}|\hat{V}_{jk},T_{jk},S,I))} \tag{3.22}$$

Autoclass approximates $P(X|T,S,I)$ with $P(X|T,S,I)^*$ holding the claiming that ”$P(X|\vec{V},T,S,I)$ and $P(X''|\vec{V},T,S,I)^*$, taken as functions of $\vec{V}$, are everywhere in the same proportion as at the MAP value $\hat{V}$”. This claim is not mathematically proved, although practically the ratio in equation (3.22) is observed to approach 1 when weights w_{ij} and parameters $\vec{V}_{jk}$ are ”mutually predictive and the weights approach indicator values”.

Moreover, Cheeseman has reported that the largest $P(X|T,S,I)^*$ can dominate the other peak integrals in the parameter sets space to a remarkable degree. ”Ratios between the two largest integrals of 10^4 to 10^9 are routine when the number of attribute values, $I \times K$, exceeds a few hundred. With a few million attribute values, the ratio may easily reach $e^{100} \approx 10^{44}$.”

Therefore, Autoclass rates the models T according to their best $P(X|T,S,I)^*$ and reports the corresponding MAP parameters $\hat{V}$. The best models that on the top list are the ones who are giving dominating marginal probability over the rest others.

3.2 K-means Clustering

The k-means algorithm needs an input to predefine the number of clusters, the k, which gives it the name. ”Means” stands for an average, the average location of all the members of a particular cluster.
Assume given n data points $D = \{x_1,...,x_n\}$, to find K clusters $\{C_1,...,C_K\}$, the following table shows the k-means algorithms.

```
initialize $m_1...m_K$ by random selection as the cluster centres
while (no termination condition is met, normally no changes in clusters $C_K$ happen)
    for $i$ = 1,...,$n$
        calculate $|x_i - m_j|^2$ for all centres
        assign data point $i$ to the closest centre
    end for
    recompute each $m_j$ as the mean of the datapoints assigned to it
end while
```

Table 3.2: K-means Algorithm

3.3 Naive Bayes Classifier

According to Principles of Data Mining [13], Naive Bayes Model assumes all the attributes are conditionally independent, given the classes labels c_k

$$p(x|c_k) = p(x_1, \ldots\ldots, x_p|c_k) = \prod_{j=1}^{p} p(x_j|c_k), \quad 1 \leq k \leq m$$

where m is the number of classes, x_j is the jth attribute . To use the model for classification we simply use the product form for the class-conditional distributions, yielding the Naive Bayes Classifier. Using Bayes theorem, the estimate of the probability that a point with measurement vector x will belong to the kth class is

$$p(c_k|x) \propto p(x|c_k)p(c_k) = p(c_k)\prod_{j=1}^{p} p(x_j|c_k) \quad 1 \leq k \leq m$$

where $p(c_k)$ is the prior probability of kth class. The Naive Bayes Classifier then assigns label k to the data point x, if its probability $p(c_k|x)$ is the higher than all the other probabilities $p(c_j|x)$, where $j \neq k$.

The reduction in the number of parameters by using the Naive Bayes model comes at a cost: we are making a very strong independence assumption. In many practical cases this conditional independence assumption may not be realistic, as some features are correlated and may be somewhat dependent on each other.

Although the independence assumption may not hold, Naive Bayes may still give relatively accurate classification performance. As [13] has given some reasons for this:

- The fact that relatively few parameters are estimated implies that the variance of the estimates will be small.
- Although the resulting probability estimates may be biased, since we are not interested in their absolute values but only in their ranked order, this may not matter.
- One of each pair of highly correlated variables may have already been discarded during feature selection.
- The decision surface from the Naive Bayes Classifier may coincide with that of the optimal classifier.

3.4 Decision Trees

Decision trees learning is one of the most widely used and practical methods for inductive inference. ”It is a method for approximating discrete-valued functions that is robust to noisy data and capable of learning disjunctive expressions. The most widely used algorithms includes ID3, ASSISTANT, C4.5 [24]. These decision tree learning methods search a completely expressive hypothesis space and thus avoid the difficulties of restricted hypothesis spaces. Their inductive bias is a preference for small trees over large trees. Learned trees can also be re-represented as sets of if-then rules to improve human readability [20]”.

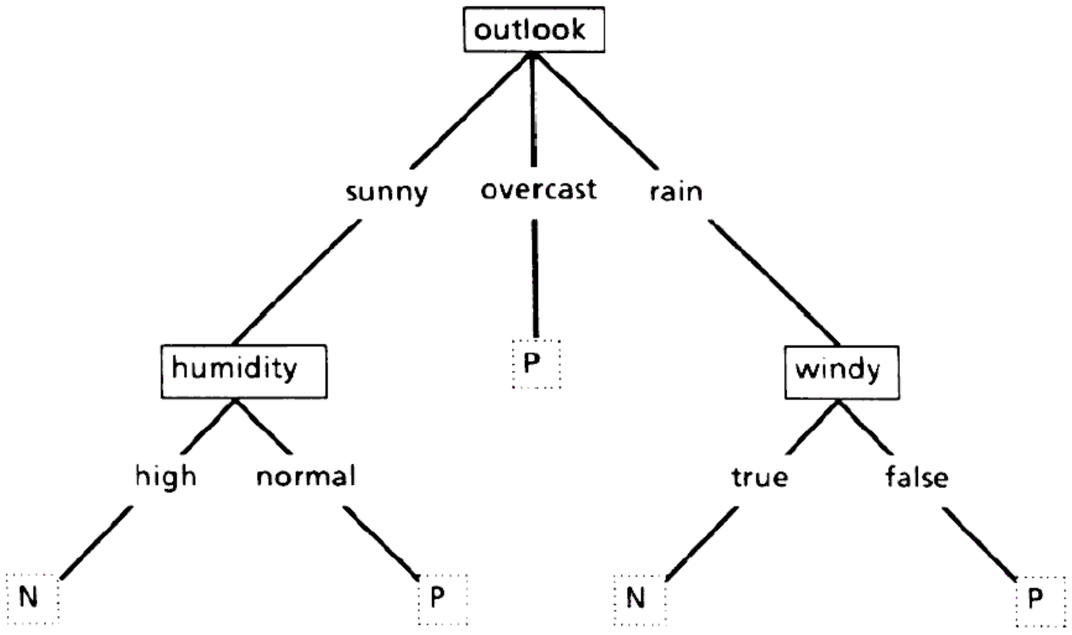


Figure 3.2: The famous decision tree example of Play Tennis.

3.5 Feature Selection

The curse of dimensionality is always a big problem for classifiers when dealing with large and relatively sparse dataset metrics in the task of text categorization. [32] have compared 5 different feature selection methods with respect to their classification performance after the feature selection. The five methods are, Document Frequency Threshold, Information Gain, χ statistic (CHI), Term Strength. Some previous projects conducted by [16] and [7] also show the promising of Rough Set Feature Reduction when doing text categorization.

However, these feature selection methods are mostly carried out requiring class labels. In other words, they are feature selection with supervised learning. When the class labels are not available or too costly to obtain, feature selection in the context of unsupervised learning is therefore highly desired.

Principle Component Analysis (PCA) [12] projects the data into lower dimensionality by keeping eigenvectors with the largest eigenvalues. This method preserves as much variation as possible in the original data but loses the semantics of original representations. [21] calculates the similarities between those features, hoping to find those features most important during clustering.

3.6 Text Vectorization

As the logs file may be in plain text format, text vectorization therefore is needed to transform them into vectors in the dataset. Below is a list of methods used by [7].

Boolean Existential Model

Assigns a weight of one if the keyword exists in the document. If the keyword is absent from the current document, a weight of zero is assigned.

Frequency

The term weight is the term's frequency in the document.

Term Frequency - Inverse Document Frequency

TF-IDF metric [26] assigns higher weights to keyword having higher frequency in the current document while not very common in the most other documents.

Fuzzy Relevance Metric (FRM)

FRM tries to determine the relevance of terms based on their places in the frequency histogram. A bell-curve is applied so that key words more frequent but not the most frequent keywords will be given more relevance.

3.7 Proposed Detection Methods

Before applying unsupervised learning algorithms on the detection of network intrusions, some assumptions are made as follows:

- The logs store almost all information about network connections so that they are representative enough.

- All the data in the training sets are normal network connections. In other words, if a network record is in the normal dataset, it must not be network intrusion.

  Just like a fundamental question a clustering algorithm needs to answer ”what is a cluster?” Before a intrusion detection system works, the question it needs to answer firstly is, ”what exactly is intrusion?”

- The data that declared abnormal are qualitatively much different from normal network connections.

- Network intrusions are the subset of the abnormal set.

After training on the normal data, the log analyzer has learned the normal profile of the network systems. The next step is to detect those abnormal network connections that look suspicious. In the implemented log analyzer, based on Autoclass and the k-means algorithm, some intrusion detection methods are proposed as follow:

Decision Boundary Capturing

This method marks those data points reside in inter-cluster decision boundaries as the abnormal data. The width of the boundaries is determined by a threshold, which is set according to the measures below:

Distance measure calculates the Euclidean distance for the test point to the cluster center. If it is larger than the threshold, it is declared abnormal.

Probability measure calculates the probability of the test point generated from the cluster. If the probability is lower than the threshold, it is declared abnormal.

Unsupervised learning and then capture via supervised labelling

This method marks all the data in some specific clusters as network intrusions. Those clusters are chosen in a supervised fashion. Given some labelled data of network attacks, the log analyzer picks the clusters that accumulate the most intrusion data , judging from a pre-set threshold.

Unsupervised learning and then capture via clusters distribution changes

This method marks all the data in some specific clusters as network intrusions. Those clusters are the clusters where the cluster distribution changes the most. This method is based on the observation that after a successful clustering, the clusters form a stable distribution regarding to their normalized sizes. When intrusions come in, the clusters distribution will be changed, which can be detected by a pre-set threshold.

3.8 Summary

This chapter introduces the theoretical aspects of the algorithms incorporated in this system. They include two unsupervised learning algorithms, Bayesian clustering and k-means clustering, and two supervised learning algorithms, Naive Bayes Classifier and Decision Trees. Issues about how the logs text are to be vectorized, how the feature selection may work or not work, and how the intrusions are to be detected are also discussed.

Chapter 4

System Implementation

This chapter describes the implementation detail of the log analyzer. First section gives the philosophy with which the design of this log analyzer is following. Later the choices of programming languages will be discussed. Then the modularized system structure will be specified. Finally, a brief introduction is given to related data mining packages that have been incorporated.

4.1 Implementation Philosophy

Experimental

This log analyzer will be an experimental framework. So we will try as much detecting methods as possible to find out which one is better. For the quickness of experimental design and modification, most of the processing jobs are written for console instead of GUI

Modularized

Modularize the whole system so that we can easily add more features as times goes on.

Quickly Developable

Incorporate tools or source code that are easy to embed and reuse. A mixture of programming languages are used here.

4.2 Programming Languages

As our system is oriented at experimenting and exploration, I didn't labor much time reinventing the wheel by write the source code all by myself. As those machine learning tools (Autoclass, Netlab, Weka and HAIG) are written in different program languages, I don't have to rewrite them all just for source code harmony. Instead, some minor modifications are made on them so that they could interact with each other in a dataset driven fashion.

C

C may be the most common programming language, devised in the 70s for the Unix Operating System and later gaining much more popularity because it offers solutions that are small size,structured,low level,efficient and very fast. Programming efforts on Autoclass are devoted to rewriting the predicting function so that Matlab readable data set could be generated during the Autoclass search.

Perl

Perl is easy to program, especially for text preprocessing. It is very quick to program and modify. Most of data set preprocessing in this project is done with perl.

Matlab

Matlab is a high-level language for analysis, visualization and development. Apart from integrated GUI editor and debugger, mathematical functions are also included. The language is both interpreted and compiled, which make it both fast enough Easy to handle dataset and do matrix operation. Most learning and detection codes in this project are written in Matlab.

4.3 Modules

Below is the description for each module in our log analyzer.

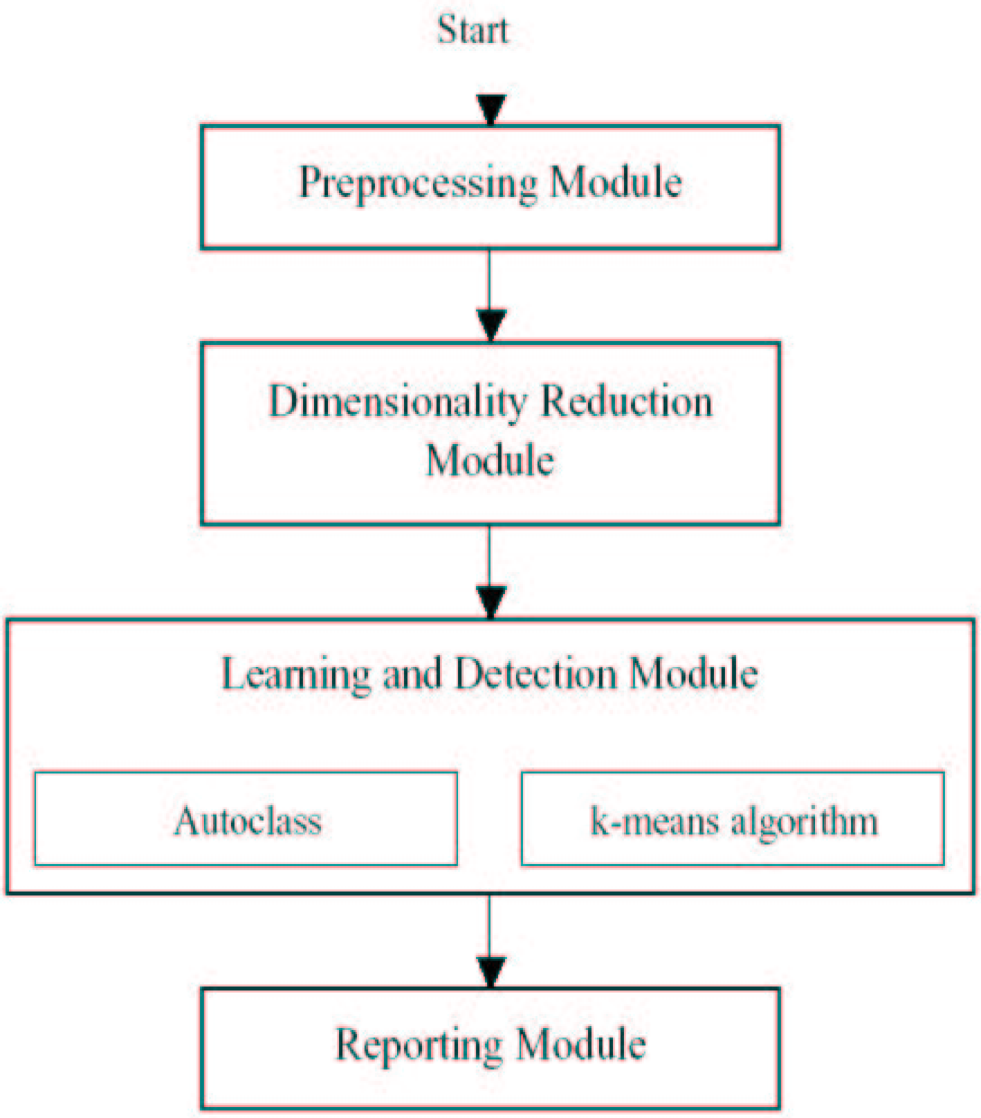


Figure 4.1: flow chart of proposed system framework

4.3.1 Preprocessing module

Input: raw logs from network

Output: The generated files are formed into the following categories

1. Autoclass input files, .hd2, .db2, .model, .s-params
2. Weka input files, with the extension of arff.
3. Matlab readable dataset file, separated by space

Function: Two main functions are required.

1. Transform the text in raw logs into dataset
2. Transform Matlab dataset file to Autoclass and Weka dataset file

Implementation: Mixture of programming languages are used to do implementation.

Written in Perl

1. Coded with network connection features
2. TF-IDF encoding

Written in Matlab

Read the Matlab dataset and write out accordingly

4.3.2 Dimensionality reduction module

Input: Matlab readable dataset file

Output: Matlab readable dataset file with dimensionality reduced

Functions: Feature Reduction and Selection

1. PCA written in Matlab
2. Manual Pick
3. Information Gain
4. χ statistics

Implementation: Mixture of methods are employed

1. PCA is Written in Matlab
2. Manual Pick is written in Perl to automatically output dataset according to user selection of the features
3. Information Gain and χ statistics feature selection are done through Weka

4.3.3 Learning and Detection module

Input: Matlab readable file and Autoclass dataset file

Output: Detection Results

Function: Two major functions:

1. Learning on the normal dataset
2. Detect network intrusions based on the learning results

Implementation: Two unsupervised learning algorithms are tried.

1. Autoclass
2. Kmeans

4.3.4 Reporting module

Input: Detection Results

Output: Statistics of detection rate and true positive rate

Function: Calculate the statistics of detection rate and true positive rate and plot the results graphically.

Implementation: Written in Matlab

4.3.5 Rule induction module

(Not implemented yet) Using decision tree rule induction to improve human readability.

4.4 Related Data Mining Packages

To reduce the programming cycle of implementation and experimentation, three external software packages are utilized, which are Autoclass, Weka and Netlab.

4.4.0.1 Autoclass

Autoclass [14] and [5] is an unsupervised classification system that seeks a maximum posterior probability classification. It is based on classical mixture model and supplemented by a Bayesian method to determine the optimal class numbers. Traditional clustering algorithms do this automatical discovery of data classes by partitioning the cases from up to down or reversely conglomerating the cases. Unlike them, Autoclass attempts to find the best class descriptions in a model space and avoids over fitting data by enforcing a tradeoff between the fit to the data and the complexity of the class descriptions.

The current version we have used in this project is Autoclass C, which is written in C, very fast in execution time. It is also very robust. Datasets with around 100,000 instances pose no challenge at all. Another nice property of Autoclass C is its flexibility in choosing searching parameters. We can preset the search time limit and request only the best few classifications Autoclass has discover so far within this time limit. This is very helpful for our initial experiments.

4.4.0.2 Weka

Weka [30] is a collection of machine learning algorithms for solving real-world data mining problems. The algorithms implemented are well designed in objected oriented fashion that they can either be applied directly or called from external Java code.

Weka package has three different interfaces: a command line interface, an Explorer GUI interface (which allows trying out different preparation, transformation and modelling algorithms on the dataset), and an Experimenter GUI interface (which allows to run different algorithms in batch and to compare the results).

Weka integrates a whole range of tools for data pre-processing, classification, regression, clustering, association rules, and visualization. This makes it really easy for trying out and comparing different classification approaches.

4.4.0.3 Netlab

Netlab is a toolbox written in Matlab functions and scripts. The most appealing thing with Netlab is that those machine learning algorithms implemented in Netlab are more up to date with the newest developments in the field. The algorithms we want to try out in our proposed detection module, such as Gaussian mixture model with EM training algorithm, K-means clustering, Self-organising map, are all included there. After little effort in modifying the source code can they be extended to serve in our log analyzer.

As Netlab is written in Matlab scripts, the extensions based it have easy access to the visualization functions provided with Matlab. That's why our reporting module is also written in Matlab, generating nice looking colored clustering plots and distribution bar graph.

4.5 Summary

In this chapter the implementation of the log analyzer is detailed. In turn the philosophies followed, the choices of programming languages, ,the modularized system structure and introductions to related data mining packages are presented.

Chapter 5

Experiments

A successful log analyzer has to identify known and unknown network attacks in the dataset. Experiments based on KDD1999 [17] will show how the log analyzer performs in capturing the known and unknown network attacks.

In detail, this chapter will describe what performance measures are, how the dataset is chosen, why the dataset poses difficulties for intrusion detection. Later, three sets of experiments are carried out to test the efficacy of the intrusion detection methods.

5.1 Experiment Design

5.1.1 Performance measures

Two measures are accepted to gauge how our log analyzer performs.

Detection Rate is the percentage of network attacks that are detected.

$$DetectionRate = \frac{NumberofNetworkAttacksDetected}{NumberofNetworkAttacks}$$

True Positive Rate is the percentage of correctly classified network attacks against the total number of data that are classified as network attacks.

$$TruePositiveRate = \frac{TruePositive}{TruePositive + FalsePositive}$$

Detection Rate indicates how successfully a log analyzer can detect attacks. True Positive Rate shows how correct a log analyzer is when making decisions.

5.1.2 Dataset choice

KDD1999 [17], the dataset experimented with is distributed by the 1998 DARPA Intrusion Detection Evaluation Program, which was prepared and managed by MIT Lincoln Lab. Lincoln labs set up an environment to acquire 9 weeks of raw TCP dump data for local-area network(LAN) simulating a typical US Air force LAN. They operated the LAN as if it were a true Air Force environment under multiple attacks.

The raw data is about 4 gigabytes of compressed binary TCP dump data from 7 weeks of network traffic. It was processed into about 5 million connection records. Similarly, the 2 weeks of test data yields around 2 million connection records.

All the data not labelled as normal connection are regarded as attacks. They fall into 4 main categories,

1. DOS: Denial of service, e.g. syn flood
2. R2L: unauthorized access from a remote machine, e.g. guessing password
3. U2R: unauthorized access to local super user privileges
4. Probing: surveillance and other probing, e.g. port scanning

The actual distribution of connection types in training and test datasets are listed in table 5.1:

connection type	training set	test set
normal	19.60%	19.48%
probe	0.83%	1.34%
dos	79.24%	73.90%
u2r	0.01%	0.07%
r2l	0.23%	5.20%

Table 5.1: Dataset Statistics

Comparing the test set with training set, the distribution of some intrusions, namely, ”u2r” and ”r2l”, has changed dramatically. This poses difficulty for some supervised learning algorithms, especially those sensitive to prior probabilities. Naive Bayes, for example, shows poor performance in the trial experiments.

Another challenge comes from the very imbalanced class distribution. ”u2r” and ”r2l”, account for only 0.07% and 5.20% of the whole test set. Those classifiers optimized for global accuracy will then tend to classify ”u2r” attacks as normal class for less global error rate and then very likely miss the chance to detect those attacks correctly.

The most difficult problem is, many attacks belonging to new sub-categories appear in the test set but not in the training set. For detectors, those attacks are totally unknown during the training. The challenges stated above are the reasons we choose this dataset as the test bed for the log analyzer to see how it deal with those difficulties, especially how it can detect those known and unknown intrusions.

5.1.3 KDD1999 winner results

The winner of KDD1999 [23] uses cost-sensitive bagged boosting of decision trees with the results in table 5.2.

predicted / actual	0	1	2	3	4	%correct
0	60262	243	78	4	6	99.5%
1	511	3471	184	0	0	83.3%
2	5299	1328	223226	0	0	97.1%
3	168	20	0	30	10	13.2%
4	14527	294	0	8	1360	8.4%
%correct	74.6%	64.8%	99.9%	71.4%	98.8%	

Table 5.2: results of KDD99 winner , table 5.3 shows attack categories

0	normal
1	probe
2	denial of service (DOS)
3	user-to-root (U2R)
4	remote-to-local (R2L)

Table 5.3: The categories of network attacks

5.2 Experiment Results and Analysis

5.2.1 Data preprocessing

5.2.1.1 Dataset transformation

Statistically, the attacks of ”u2r” and ”r2l” are of the most rare, which makes them very hard to predict. On the other hand, they are the most dangerous types. Once an attacker gains the super user right or successfully remote login, disasters of the whole system are nothing but unavoidable. Comparably, attacks of ”probe” is much more benign. Although attacks of ”dos” (denial of service) are massive in the whole original dataset, they impose less danger. This is because the nature of denial of service attacks lies in that they are trying to initiate as many as possible connections to consume the network traffics and server CPU time. This kind of attacks is too apparent for a network administrator to miss.

From the reasons above, finding out attacks of type ”u2r” and ’r2l’ is much more important than types of ”probe” and ”dos”. Hence, attacks of types ”probe” and ”dos” are thrown out. With this transformed dataset, the log analyzer can be tested with more challenging tasks.

5.2.1.2 Feature selection

This dataset has totally 43 features, including 9 basic features. The rest are all derived features extracted by domain knowledge. Those are content features and timing features. The table 5.4 lists all the basic features.

feature name	description	type
duration	length (number of seconds) of the connection	continuous
protocol_type	type of the protocol, e.g. tcp, udp, etc.	discrete
service	network service on the destination, e.g., http, telnet, etc.	discrete
src_bytes	number of data bytes from source to destination	continuous
dst_bytes	number of data bytes from destination to source	continuous
flag	normal or error status of the connection	discrete
land	1 if connection is from/to the same host/port; 0 otherwise	discrete
wrong_fragment	number of “wrong” fragments	continuous
urgent	number of urgent packets	continuous

Table 5.4: features

Because of the limitation of searching space and time, dimensionality must be reduced. The problem is, it is not easy for unsupervised feature selection. Alternately, the feature selection here is based on supervised learning methods like Information Gain and χ statistic (CHI). Appendix A prints the feature selection results given by them.

Finally, only 4 basic features of individual TCP connections are chosen. They are duration, service, src_bytes, dst_bytes. It should be noted that this choice is not guaranteed to be the best one due to the lack of proper unsupervised feature selection methods.

5.2.1.3 Creating training and test set

Some programs are written in perl to split original training and test datasets and to wipe out those attacks of types ”probe” and ”dos”. Hence the data in the training set are all network connections that have been labelled as normal. The test set varies depending on the experiments requirement and is constructed accordingly.

5.2.2 Experiment 1

This experiment aims to test whether the ”Decision boundary capturing” method will work on this dataset. Description of this detection method can be found in section 3.7.

5.2.2.1 Experiment procedures

1. Construct a new dataset with all the ”u2r” attacks and assume it as the abnormal dataset.
2. Train k-means algorithms on the normal dataset to get a model with 33 clusters. (Here it should be noted that Autoclass fails to provide meaningful decision boundary and therefore not trained in this experiment)
3. Mark those points on the inter-cluster decision boundary as network intrusions.
4. Calculate the true positive rate.

5.2.2.2 Results

Run	True Positive	Positive	True Positive Rate
1	219	59146	0.3703%
2	219	58976	0.3713%
3	220	54980	0.4001%
4	217	59396	0.3653%
5	219	54706	0.4003%
6	220	54392	0.4045%
7	218	59742	0.3649%
8	215	59295	0.3626%
9	221	54786	0.4034%
10	219	59448	0.3684%
mean	218.70	57486.7	0.3811%

Table 5.5: Capture with decision boundary has extremely low true positive rate when boundary width threshold is set as 10

5.2.2.3 Analysis

Results above show that the true positive rate is roughly the same as random guessing. It seems that, for this dataset, decision boundary cannot capture those abnormal network data.

One reason for that may be, those network intrusions are to some extent similar to some of the normal network data. This makes it difficult for those network attacks data to fall on the decision boundary. Instead, they are found condensed in some specific clusters, which is observed in later experiments.

5.2.3 Experiment 2

This experiment aims to test whether ”Unsupervised learning and then capture via supervised labelling” method will work on this dataset. Description of this detection method can be found in section 3.7.

5.2.3.1 Experiment procedures

1. Construct a new dataset with all the ”u2r” attacks and assume it as the abnormal dataset.
2. Train the k-means algorithm on the normal data to get a model of 33 clusters.
3. Train Autoclass on the normal data to get clusters. (It shall be noted that Autoclass will automatically search optimized cluster number. So no user input is needed for that)
4. Use previous labels provided to mark specific clusters accumulating enough ”u2r” attacks
5. Mark all the data in those clusters as network intrusions.
6. Set the threshold value as 5% for true positive rate.

5.2.3.2 Results

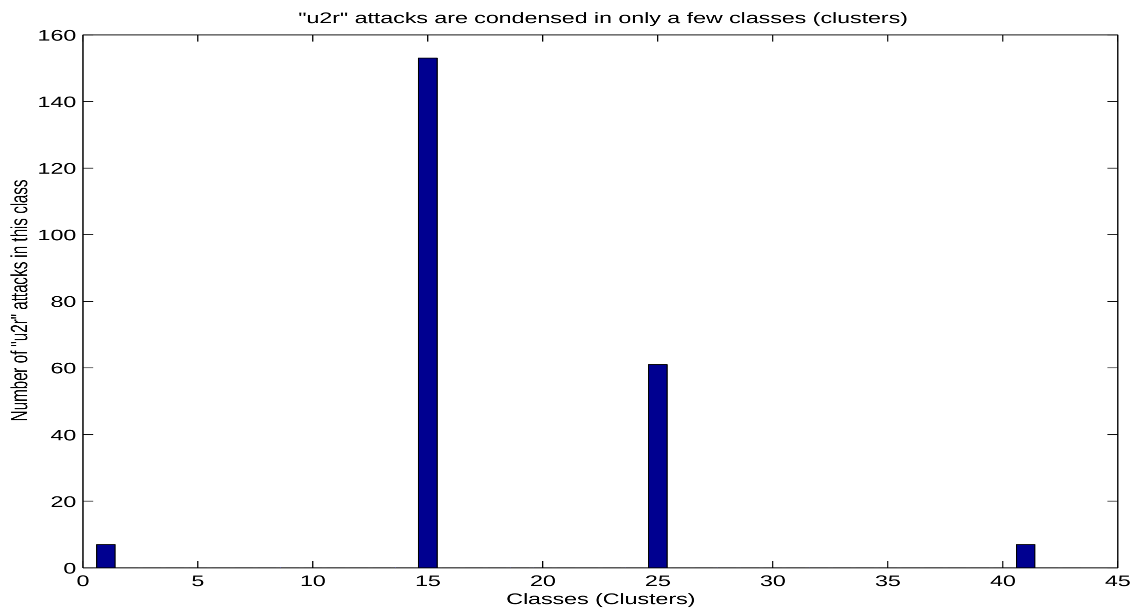


Figure 5.1: Clustering result of Autoclass in Run #10

Run	Class Numbers	True Positive	Positive	Detection Rate	True Positive Rate
1	33	154	2655	67.5439%	5.8004%
2	33	61	453	26.7544%	13.4658%
3	33	198	1360	86.8421%	14.5588%
4	33	184	2874	80.7018%	6.4022%
5	33	162	980	71.0526%	16.5306%
6	33	77	343	33.7719%	22.4490%
7	33	197	457	86.4035%	43.1072%
8	33	192	595	84.2105%	32.2689%
9	33	182	484	79.8246%	37.6033%
10	33	183	2869	80.2632%	6.3785%
mean	33	159	1307	69.74%	19.86%

Table 5.6: the k-means algorithm results

Run	Class Numbers	True Positive	Positive	Detection Rate	True Positive Rate
1	46	196	881	85.9649%	22.2474 %
2	49	208	1191	91.2281%	17.4643 %
3	43	194	800	90.6542%	24.2500 %
4	49	208	924	91.2281%	22.5108 %
5	35	208	2420	91.2281%	8.5950 %
6	43	184	1289	80.7018%	14.2746 %
7	47	211	1891	92.5439%	11.1581 %
8	39	221	3098	96.9298%	7.1336 %
9	45	199	1142	87.2807%	17.4256 %
10	43	221	2296	96.9298%	9.6254 %
mean	43.9	205	1593.2	90.47%	15.47%

Table 5.7: Autoclass results

5.2.3.3 Analysis

Both the k-means algorithm and Autoclass succeed in clustering and finding that ”u2r” attacks condensed in a limited number of classes (clusters).

As show in table 5.6, the k-means algorithm results have an average detection rate of 69.74% and true positive rate of 19.86%. For Autoclass, as illustrated in table 5.7, the average result is better, with a stable detection rate of 90.47% and true positive rate of 15.47%.

Comparably, the winner of KDD1999 can only capture 30 in 228 attacks, which is a detection rate of 13.2%. In this regard, based on no matter whether the k-means algorithm or Autoclass, our log analyzer could surely beat the winner of KDD1999. Although the winner of KDD1999 has a higher true positive rate of 71.4%, it cannot count as this rate comes with too low a detection rate. In a real-world scenario, where each ”user to root” attack is so rare (0.038%) but too lethal to miss any one of them, the detector with higher detection rate is superior.

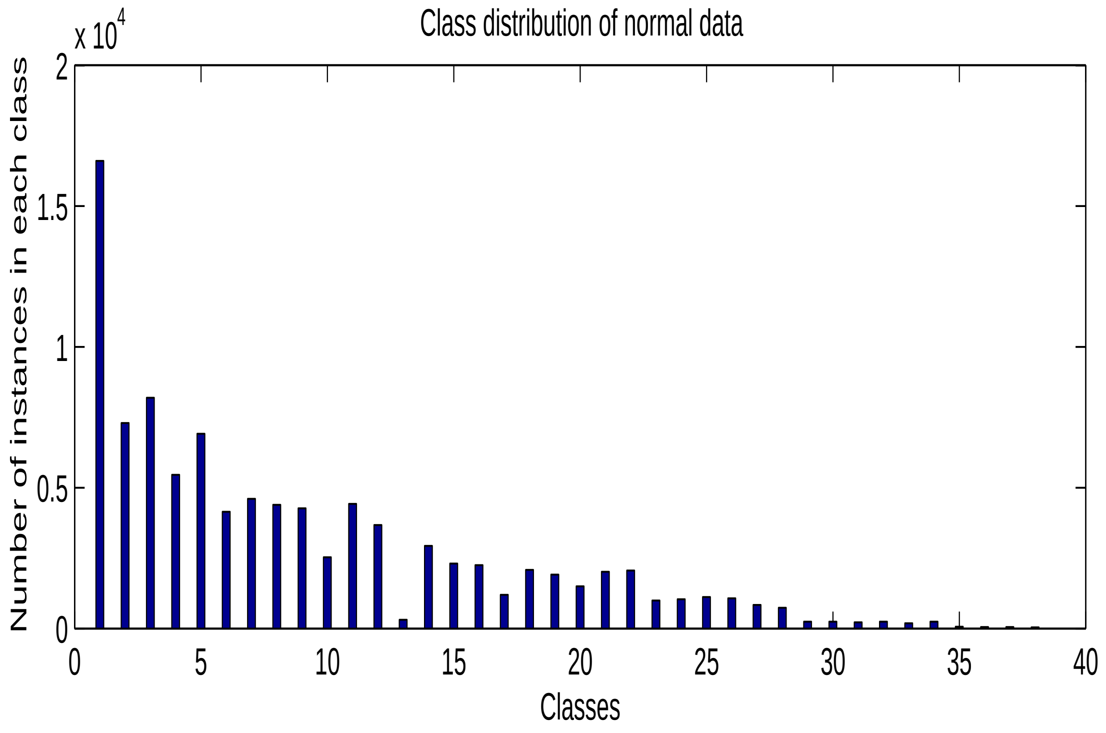
5.2.4 Experiment 3

This experiment aims to test whether ”Unsupervised learning and then capture via clusters distribution changes” method will work on this dataset. Description of this detection method can be found in section 3.7.

5.2.4.1 Experimental procedures

1. Extract ”u2r” attacks and ”r2l” attacks and put them together to form the test dataset.
2. Train Autoclass on the normal data to find clusters.
3. Use clustering results to plot the normalized class distribution.
4. Predict on the test set with the learned model.
5. Use the prediction results to plot the normalized class distribution of the test set.
6. Compare the changes of class distribution between normal dataset and test dataset.
7. Set a threshold of 0.3 to mark those clusters (classes) that have increased at least 0.3 times regarding to their normalized clusters distribution.
8. Mark all the data in those clusters (classes) as network intrusions.

5.2.4.2 Results



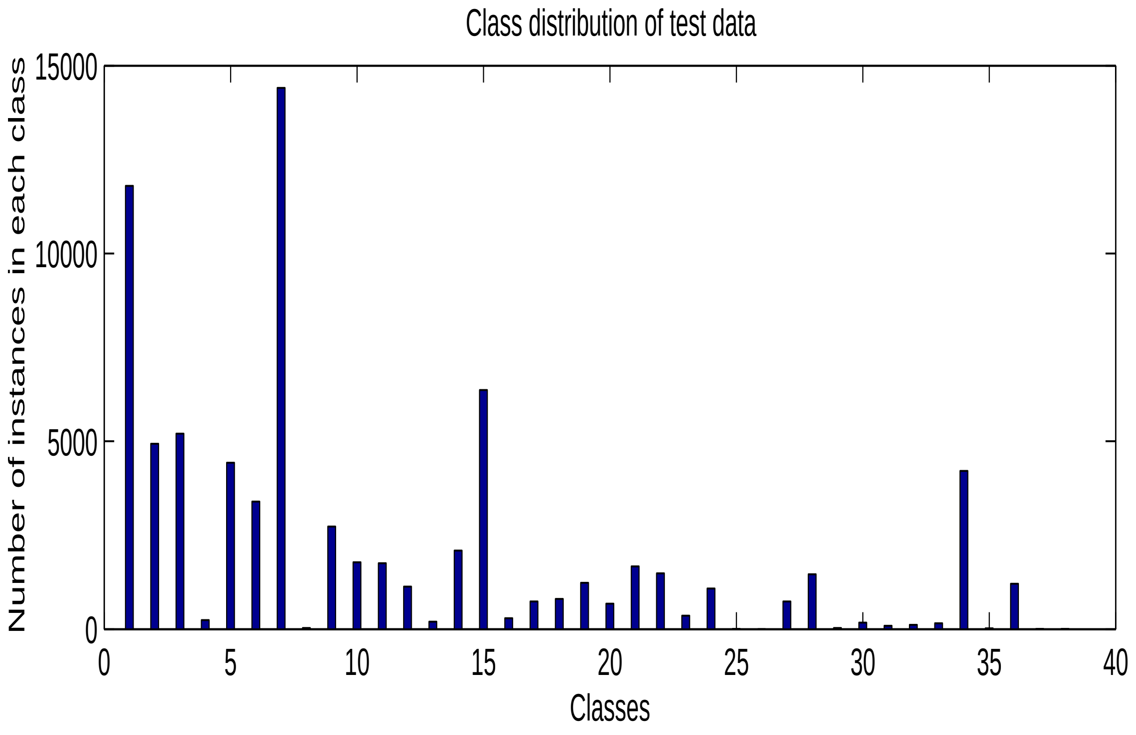
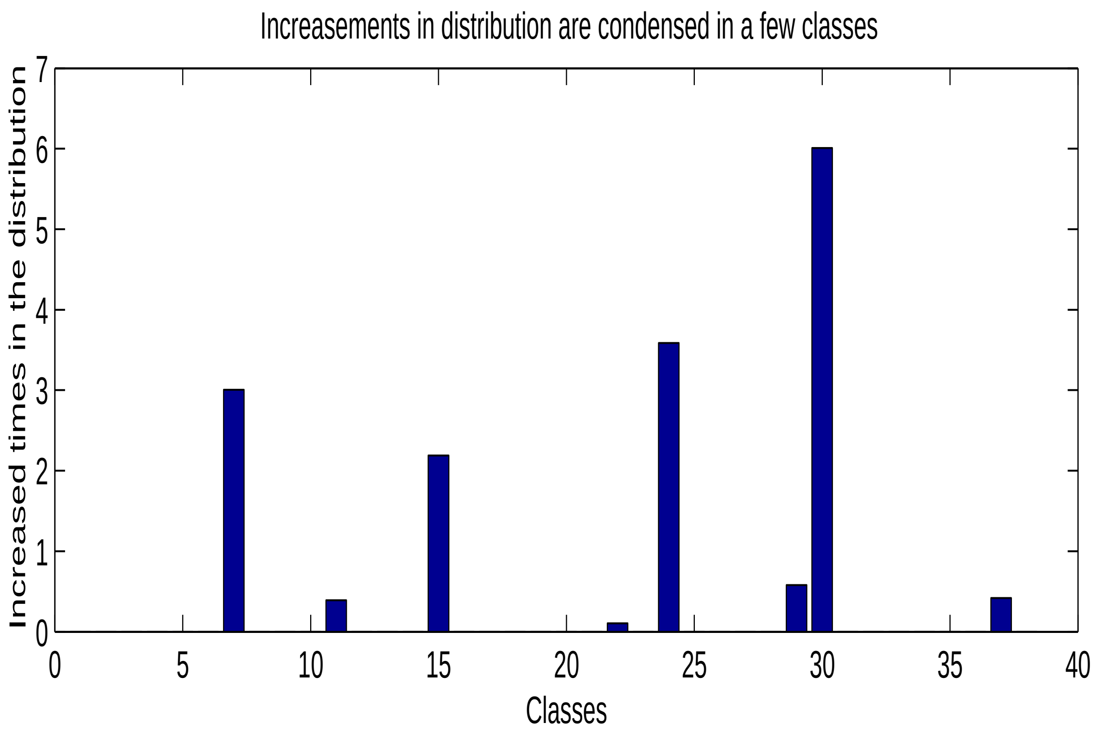


Figure 5.2: Clusters (Classes) distributions are different on the normal dataset and test dataset.



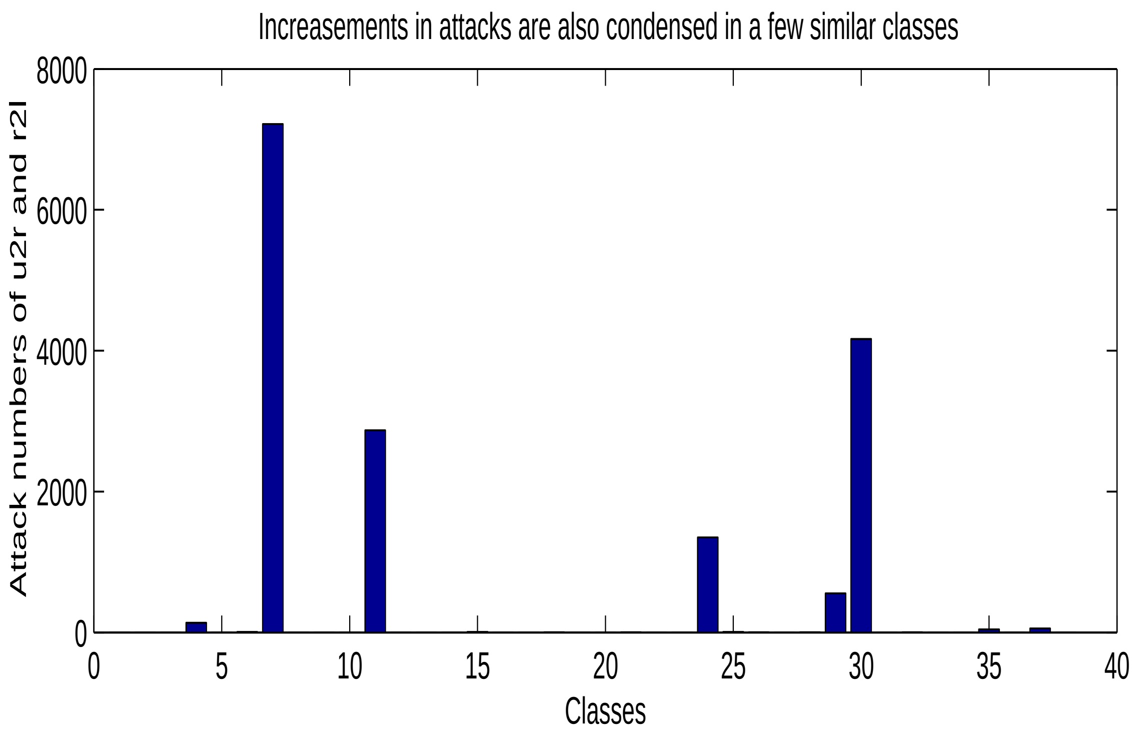


Figure 5.3: A strong correspondence between the increase in the distribution with the number of network attacks. 1. The first graph shows the increases of data distribution in this class, comparing the class distribution of the training and test dataset. The increases happen only in quite a limited classes. 2. The second figure counts the network attacks in the test set.

5.2.4.3 Analysis

Figure 5.2 shows the clusters (classes) distribution on the normal and test dataset. Moreover, as illustrated in figure 5.3, there is a strong correspondence between the increase in the class distribution and the number of network attacks. If the detecting threshold is set to 0.3, which means that the clusters (classes) that have increased over 0.3 times in the clusters (classes) distribution will be marked as clusters of network intrusions.

With this threshold, the clusters (classes) 7,11,15,24,29,30,37 are picked out as the most abnormal classes. Marking all the data in those clusters (classes) as network attacks will capture totally 16215 attacks. Thus the true positive rate is

$$\frac{16215}{31203+16215} = 34.20\%$$

More prominently, the detection rate is

$$\frac{16215}{16417} = 98.77\%$$

Comparably, the winner of KDD99 provides a solution which can capture only 13.2% u2r attacks and merely 8.4% r2l attacks.

The method of ”Unsupervised learning and then capture via distribution changes” works nicely here with very high detection rate and reasonable true positive rate. The most attractive point with this methods is that it can detect unknown attacks.

5.3 Conclusions

A series of experiments have been conducted on KDD1999 dataset. Different detection methods are tried to see how they perform in our log analyzer. Beating KDD1999 winner’s result, a very high detection rate has been obtained, although with a reasonable true positive rate.

By experimenting with two totally different unsupervised learning algorithms, both of them, the k-means algorithm and Autoclass, work well on the clustering task. The k-mean algorithm, though much simpler and less stable, sometimes does present a very good result with high detection rate or true positive rate. Based on a rigor mathematical foundation, Autoclass seems more robust in the intrusion detection.

An interesting by-product generated by Autoclass is its ”Attribute Influence Value Report”, which is given in Appendix B. This report gives a rough heuristic measure of relative influence of each attribute in differentiating the classes from the overall dataset. It may contribute to one method of unsupervised feature selection.

For the three intrusion detection methods, experiments results show that, the first method, ”Decision boundary capturing”, does not work. ”Unsupervised learning and then capture via supervised labelling” works with the help from some pre-labelled data, assuming that those pre-labelled data are representative enough to cover clusters with intrusions. This method is suitable for detecting intrusions data with limited labelled data. It may not be able to detect those unknown attacks if the clusters containing unknown attacks are not marked out by the given labelled data.

”Unsupervised learning and then capture via clusters distribution changes” works when there are dramatic changes in the whole clusters distribution. This method is nice in that no previous labelled data is needed and those unknown attacks can be captured.

It has to be noted that, the determination of the detection threshold is still not automated. For a good intrusion detection rate with reasonably high true positive rate, trade-off has to be considered by human operator.

Chapter 6

Conclusion

This project initially aimed to build a log analyzer, especially finding out proper detection methods for known and unknown network intrusions. Experiments based on the KDD1999 dataset show some encouraging results.

In this chapter, the achievements of this project will be evaluated firstly. The next subsection will discuss some of the limitations in our log analyzer. Finally, some possible extensions are discussed.

6.1 Achievements

This system is a tentative research on the possibility of applying unsupervised learning in the detection of network intrusions. Based on unsupervised learning algorithms, some novel detection methods are proposed and tested, showing a very high detection rate with a reasonable true positive rate as results, much better than the winning results of KDD1999 dataset.

It seems that not much previous work has employed unsupervised learning to detect network intrusions. By training with unsupervised learning algorithms, namely, Autoclass and the k-means algorithm, the log analyzer performs well in discovering the inherent nature of the dataset, clustering similar instances into the same classes. Two of the three detection methods that have been proposed are also shown to be very effective in detecting known and unknown network intrusions.

All initial objectives of this project have been met.

Modularity

The system is modularized so that new learning algorithms are easy to be added in and tested. This well suits the dynamics in the research community.

Learning and Detection

Experiments show that this log analyzer could successfully detect most of the attacks in KDD1999 dataset with reasonable true positive rate. This result is much better than the winner of KDD1999 regarding detection rate.

Generality

Generalization ability to apply on any other log formats is made easy under the data mining framework, by first transforming logs into standard dataset format before presenting them to the intrusion detector.

6.2 Limitations

There are still some limitations in the current version of the implemented log analyzer.

Software engineering

This system is still an experimental framework. So many scripts are not well tuned for general purpose input. Especially in the data preprocessing module, where the definition of the input dataset is pre-programmed, some codes need to be recoded for better general use if it is to work on other datasets.

Experiment completeness

Due to the lack of training examples, the efficacy of the machine learning algorithm on other datasets has not been tested. Although on email categorization, [7] have shown that RSAR+FR could successfully reduce the dimensionality while keeping enough prediction accuracy, there has not been sufficient time to apply it onto other log formats yet.

Computational cost

As mixture modelling requires a parameter estimation process, it is computational expensive for the learning. On the other hand, nonparametric density estimation, such as k-means algorithm, requires no training, but it needs a large training dataset and the testing process on the model is slow.

Normality definition

The success of our system partially depends on the definition of what is normal and what is not. Also it should be noted that network attacks instances are a subset of the abnormal set.

Threshold setting

The detection rate may be lower in a different environment, requiring different threshold for detection. Especially if the distribution is roughly at the same level, it will be very hard to set a threshold to predict the network attacks. Instead, we should rely on the training labels to see which clusters are the most representative for network attacks.

Difficulty for human interpretation

The Bayesian classification results are hard for a human reader to interpret. More comprehensibility is needed for easier human understandable results.

6.3 Future Work

Due to the time limit, several interesting ideas have not been implemented yet. However, it will be worthwhile trying them in the future.

Easier human interpretation

The clustering results are hard for a human to interpret. This poses difficulties for human experts to improve our log analyzer. A rule induction algorithm, such as a decision tree algorithm, for instance, may provide easier human interpretability.

This could be done by assigning the same class labels to those in the same cluster, then training the dataset with those new labels using a decision trees algorithm, like C4.5 [24]. The rules generated can then be presented to those domain experts for further inspection and make it easier for them to interpret the clustering results.

Real-time detection

Our log analyzer operates like a batch mode data miner, which labels all those data that look abnormal in a batch run. It is helpful for post-attack analysis. What will be more appealing is those doing real-time detection, which demands higher computational efficiency.

Two approaches may provide real-time capability in our log analyzer. One is to extract the parameters of mixture models learned so far to a real-time detector, which uses the mixture model to classify the incoming network records. As calculating the class membership probability needs only those model parameters, it is computationally very insignificant.

Another way could be utilizing the rules generated by a rule induction algorithm on the clustering results. After human expert inspection and modification, these rules could be embedded into a real-time detector. Matching the data against the rule set could be more computational expensive than the previous approach. On the other hand, it has the advantage of adding more power through its easier interpretability by human experts.

Online learning

The learning algorithms implemented here all work in off-line mode. As the environment may change gradually as time goes by, it is worthwhile investigating on how to incorporate online parameters updating into the learning.

More experiments with other logs

The experiments we have conducted are based on the network raw TCP data captured by TCPDUMP in binary format. The dataset therefore consists only of those basic connection related features. As our log analyzer aims to be as general as possible, further experiments on other kind of network logs are necessary to see whether it performs as well.

It will be especially interesting to test the log analyzer on web server logs which consist of quite a lot of url request information. Those logs have more text content than the binary logs captured by TCPDUMP. Hence this will be relevant to text categorization.

Moreover, to deal with the problem of huge dimensionality of the dataset, some feature reduction techniques such as rough set feature reduction [7], may be needed to reduce the dimensionality while keeping as much of the semantics as possible.

Neural network based approaches

Neural networks have been widely applied in supervised learning context on novelty detection projects. [3] has trained multi-layer neural networks and use the testing errors in the neural network to signal the presence of novelty or anomaly. It will be very interesting to see how it works in our log analyzer.

Self-organizing maps is also interesting for a trial as it can provide the clustering function from a totally different perspective other than those of k-means clustering or Bayesian clustering. Fortunately, our modularized framework make it very easy to add new learning algorithms and detection methods.

Meanwhile, the Netlab package has included implementations for Multi-layer neural network (MLP) and Self-organizing maps (SOM). Not too much programming jobs will be needed for that.

Unsupervised feature selection

During clustering, Autoclass [6] has generated an interesting byproduct, the weighting of features. In the ”influence value report” given by Autoclass, an ”ordered list of normalized feature influence values summed over all classes” is printed. This gives a rough heuristic measure of relative influence of each feature in differentiating the classes from the overall dataset. It may be possible to conduct unsupervised feature selection based on this ordered list by strategically keeping the most influential features and pruning those at the bottom of the list. Interested readers are referred to appendix B, where this ordered list is printed.

6.4 Summary

This chapter concludes the thesis with discussions of the achievements, limitations, and future extensions of the project. Although more tests and improvements are needed, this project has successfully produced a system that can perform log analysis-based intrusion detection, via unsupervised learning algorithms, with some novel intrusion detection methods.

Appendix A

Feature Selection

```
=== Run information ===

Evaluator:    weka.attributeSelection.InfoGainAttributeEval
Search:       weka.attributeSelection.Ranker -T
-1.7976931348623157E308 -N -1 Relation:
weka-weka.filters.ResampleFilter-B0.0-S1-Z30.0-weka.filters.ResampleFilter-B0.0
Instances:    23034 Attributes:   42 Evaluation mode:    10-fold
cross-validation



=== Attribute selection 10 fold cross-validation (stratified), seed: 1 ===

average merit      average rank  attribute
 0.405 +- 0.001      1   +- 0        3 service
 0.399 +- 0.003      2   +- 0        6 dst_bytes
 0.388 +- 0.001      3   +- 0        5 src_bytes
 0.194 +- 0.001      4   +- 0       24 srv_count
 0.153 +- 0.002      5.2 +- 0.4     33 dst_host_srv_count
 0.151 +- 0.001      5.8 +- 0.4     23 count
 0.143 +- 0.001      7   +- 0       35 dst_host_diff_srv_rate
```

```
0.14  +- 0.001      8   +- 0          1 duration
0.135 +- 0.001      9   +- 0         37 dst_host_srv_diff_host_rate
0.112 +- 0.001     10   +- 0         31 srv_diff_host_rate
0.108 +- 0.001     11   +- 0         32 dst_host_count
0.099 +- 0.001     12   +- 0         12 logged_in
0.083 +- 0.001     13   +- 0         36 dst_host_same_src_port_rate
0.069 +- 0.001     14   +- 0          2 protocol_type
0.046 +- 0.001     15   +- 0         34 dst_host_same_srv_rate
0.028 +- 0.001     16   +- 0         40 dst_host_rerror_rate
0.022 +- 0.001     17   +- 0         11 num_failed_logins
0.017 +- 0         18   +- 0         10 hot
0.011 +- 0         19   +- 0         22 is_guest_login
0.009 +- 0         20   +- 0         41 dst_host_srv_rerror_rate
0.004 +- 0         21   +- 0         38 dst_host_serror_rate
0.002 +- 0         22   +- 0         26 srv_serror_rate
0.002 +- 0         23   +- 0         25 serror_rate
0.001 +- 0         24.8 +- 0.4        4 flag
0.001 +- 0.001     25.1 +- 2.98      39 dst_host_srv_serror_rate
0.001 +- 0         26.1 +- 0.54      29 same_srv_rate
0     +- 0         27.6 +- 0.66      17 num_file_creations
0     +- 0         29.3 +- 1.79      19 num_access_files
0     +- 0         29.4 +- 0.92       8 wrong_fragment
0     +- 0         30.2 +- 1.25      21 is_host_login
0     +- 0         30.3 +- 3.44       9 urgent
0     +- 0         32.6 +- 1.28      20 num_outbound_cmds
0     +- 0         33.7 +- 0.78      27 rerror_rate
0     +- 0         33.9 +- 1.37      28 srv_rerror_rate
0     +- 0         34.7 +- 0.46      30 diff_srv_rate
0     +- 0         36.6 +- 6.73      16 num_root
0     +- 0         37   +- 0         14 root_shell
0     +- 0         37.8 +- 1.47      15 su_attempted
```

```
 0     +- 0         38   +- 0       13 num_compromised
 0     +- 0         38.1 +- 2.77    18 num_shells
 0     +- 0         39.8 +- 0.75     7 land


=== Run information ===

Evaluator:    weka.attributeSelection.ChiSquaredAttributeEval
Search:       weka.attributeSelection.Ranker -T
-1.7976931348623157E308 -N -1 Relation:
weka-weka.filters.ResampleFilter-B0.0-S1-Z30.0-weka.filters.ResampleFilter-B0.0
Instances:    23034 Attributes:   42

Evaluation mode:    10-fold cross-validation

=== Run information ===

Evaluator:    weka.attributeSelection.ChiSquaredAttributeEval
Search:       weka.attributeSelection.Ranker -T -1.7976931348623157E308 -N -1
Relation:     weka-weka.filters.ResampleFilter-B0.0-S1-Z30.0-weka.filters.Resam
Instances:    23034
Attributes:   42
Evaluation mode:    10-fold cross-validation

=== Attribute selection 10 fold cross-validation (stratified), seed: 1 ===

average merit      average rank  attribute
10423.202 +-74.353      1   +- 0        6 dst_bytes
10318.553 +-22.091      2   +- 0        3 service
10147.603 +-39.236      3   +- 0        5 src_bytes
4780.635 +-22.755      4   +- 0        1 duration
4371.23 +-63.441      5.1 +- 0.3     33 dst_host_srv_count
```

```
4265.378 +-22.027      5.9 +- 0.3     24 srv_count
4133.688 +-27.559      7   +- 0       35 dst_host_diff_srv_rate
3151.703 +-18.531      8   +- 0       23 count
2911.237 +-15.551      9   +- 0       37 dst_host_srv_diff_host_rate
2862.743 +-33.265     10   +- 0       12 logged_in
2476.222 +-18.694     11   +- 0       32 dst_host_count
2120.32 +-14.101     12   +- 0       31 srv_diff_host_rate
2064.451 +-28.34      13   +- 0        2 protocol_type
1882.648 +-17.077     14   +- 0       36 dst_host_same_src_port_rate
1523.827 +-22.773     15   +- 0       34 dst_host_same_srv_rate
1006.829 +-27.323     16   +- 0       40 dst_host_rerror_rate
744.192 +-18.064     17   +- 0       11 num_failed_logins
589.827 +-14.313     18   +- 0       10 hot
421.858 +-18.227     19   +- 0       22 is_guest_login
328.78 +-18.298     20   +- 0       41 dst_host_srv_rerror_rate
114.21 +-12.296     21   +- 0       38 dst_host_serror_rate
45.167 +- 1.12      22.5 +- 0.5     26 srv_serror_rate
44.176 +-17.671     23.7 +- 3.23    39 dst_host_srv_serror_rate
35.832 +- 2.179     24.2 +- 0.75     4 flag
35.601 +- 1.344     24.4 +- 0.49    25 serror_rate
10.006 +- 1.698     27.4 +- 0.92    21 is_host_login
10.056 +- 0.76      27.6 +- 0.8     29 same_srv_rate
 9.636 +- 3.397     27.9 +- 3.81     9 urgent
 7.135 +- 1.331     29.1 +- 0.94    19 num_access_files
 5.835 +- 0.367     30   +- 0.77    17 num_file_creations
 5.105 +- 0.282     31   +- 0.63     8 wrong_fragment
 0     +- 0         32.8 +- 1.33    27 rerror_rate
 6.062 +- 9.309     34.1 +- 5.82    16 num_root
 0     +- 0         34.1 +- 0.54    28 srv_rerror_rate
 0     +- 0         34.4 +- 1.11    30 diff_srv_rate
 0     +- 0         36.2 +- 2.27    15 su_attempted
 0     +- 0         36.6 +- 3.07    20 num_outbound_cmds
 0     +- 0         36.8 +- 2.68    14 root_shell
 0     +- 0         37.7 +- 0.9     13 num_compromised
 0     +- 0         39.3 +- 1.35    18 num_shells
 0     +- 0         40.2 +- 0.75     7 land
```

Appendix B

Normalized Feature Influence

The list below shows the normalized feature influence found by Autoclass. Probably it may serve in unsupervised feature selection.

num	description	Influence
032	Log wrong_fragment	1.000
029	Log num_compromised	0.963
028	Log root_shell	0.959
026	Log num_root	0.910
025	Log num_file_creations	0.909
027	Log su_attempted	0.885
030	Log num_failed_logins	0.870
024	Log num_access_files	0.853
023	Log is_host_login	0.829
031	Log hot	0.583
035	Log duration	0.374
034	Log src_bytes	0.128
033	Log dst_bytes	0.111
002	service	0.077
011	logged_in	0.026
001	protocol_type	0.026
006	land	0.002

003	flag	0.001
000	duration	-----
004	src_bytes	-----
005	dst_bytes	-----
007	wrong_fragment	-----
008	urgent	-----
009	hot	-----
010	num_failed_logins	-----
012	num_compromised	-----
013	root_shell	-----
014	su_attempted	-----
015	num_root	-----
016	num_file_creations	-----
017	num_shells	-----
018	num_access_files	-----
019	num_outbound_cmds	-----
020	is_host_login	-----
021	is_guest_login	-----
022	class label	-----

Bibliography

[1] N. M. Laird A. P. Dempster and D. B. Rubin. *Maximum likelihood from incomplete data via the EM algorithm*.

[2] Pavel Berkhin. Survey of clustering data mining techniques. Technical report, Accrue Software, San Jose, CA, 2002.

[3] C. Bishop. Novelty detection and neural network validation, 1994.

[4] CERT Coordination Center (CERT/CC). *CERT/CC Statistics 1988-2003*. http://www.cert.org/stats/cert_stats.html#incidents.

[5] Peter Cheeseman and John Stutz. Bayesian classification (autoclass): Theory and results. pages 153–180, 1996.

[6] Peter Cheeseman, John Stutz, and Will Taylor. World Wide Web, http://ic.arc.nasa.gov/ic/projects/bayes-group/autoclass/.

[7] Alexios Chouchoulas and Qiang Shen. A rough set approach to text classification, proceedings of the seventh international workshop on rough sets (lecture notes in artificial intelligence, no. 1711) pages=118-127 1999.

[8] A.F.M. Smith D.M. Titterington and U.E. Makov. *Statistical Analysis of Finite Mixture Distributions*. John Wiley and Sons, New York, 1985.

[9] B.S. Everitt and D.J. Hand. *Finite Mixture Distributions*. Chapman and Hall, London, New York, 1981.

[10] Wei Fan, Matthew Miller, Salvatore J. Stolfo, Wenke Lee, and Philip K. Chan. Using artificial anomalies to detect unknown and known network intrusions. In *ICDM*, pages 123–130, 2001.

[11] Stephanie Forrest, Alan S. Perelson, Lawrence Allen, and Rajesh Cherukuri. Self-nonself discrimination in a computer. In *Proceedings of the 1994 IEEE Symposium on Research in Security and Privacy*, pages 202–212, Oakland, CA, 1994. IEEE Computer Society Press.

[12] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing*. Addison-Wesley Publishing Company, 3rd edition, 1993.

[13] David Hand, Heikki Mannila, and Padhraic Smyth. Principles of data mining.

[14] Robin Hanson, John Stutz, and Peter Cheeseman. Bayesian classification theory. 1991.

[15] Geoffrey Hinton and Terrence J. Sejnowski. *Unsupervised Learning: Foundations of Neural Computation*.

[16] Richard Jenson. A rough set aided system for sorting www bookmarks.

[17] KDD-CUP-1999. http://kdd.ics.uci.edu/databases/kddcup99/task.html.

[18] Wenke Lee and Salvatore Stolfo. Data mining approaches for intrusion detection. In *Proceedings of the 7th USENIX Security Symposium*, San Antonio, TX, 1998.

[19] M. Markou and S. Singh. Novelty detection: A review, part i: Statistical approaches.

[20] Tom M. Mitchell. *Machine Learning*.

[21] P. Mitra, C.A. Murthy, and S.K. Pal. Unsupervised feature selection using feature similarity. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(4), 2002.

[22] E. Parzen. *On the estimation of a probability density function and the mode*, volume 33.

[23] Bernhard Pfahringer. *Winning the kdd99 classification cup: Bagged boosting*.

[24] Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, San Mateo, CA, 1993.

[25] Jake Ryan, Meng-Jang Lin, and Risto Miikkulainen. Intrusion detection with neural networks. In Michael I. Jordan, Michael J. Kearns, and Sara A. Solla, editors, *Advances in Neural Information Processing Systems*, volume 10. The MIT Press, 1998.

[26] G. Salton and C. Buckley. *Term-weighting approaches in automatic text retrieval*. 1988.

[27] Matthew G. Schultz, Eleazar Eskin, Erez Zadok, Manasi Bhattacharyya, and Salvatore J. Stolfo. MEF: Malicious email filter: A UNIX mail filter that detects malicious windows executables. pages 245–252.

[28] Snort. *An open source network intrusion detection system*. http://www.snort.org/.

[29] CISCO Sytems Ltd white paper. *The Science of Intrusion Detection System Attack Identification*. World Wide Web, http://www.cisco.com/en/US/products/sw/secursw/ps2113/products_white_paper09186a0080092334.shtml.

[30] Ian H. Witten and Eibe Frank. *Data Mining, Practical Machine Learning Tools and Techniques with Java Implementations*.

[31] Kenji Yamanishi, Jun ichi Takeuchi, Graham J. Williams, and Peter Milne. On-line unsupervised outlier detection using finite mixtures with discounting learning algorithms. In *Knowledge Discovery and Data Mining*, pages 320–324, 2000.

[32] Yiming Yang and Jan O. Pedersen. A comparative study on feature selection in text categorization. In Douglas H. Fisher, editor, *Proceedings of ICML-97, 14th International Conference on Machine Learning*, pages 412–420, Nashville, US, 1997. Morgan Kaufmann Publishers, San Francisco, US.

[33] Dit-Yan Yeung and Calvin Chow. Parzen-window network intrusion detectors. In *Proc. of the Sixteenth International Conference on Pattern Recognition*, volume 4, pages 385–388, August 11–15, 2002.