INTRUSION DETECTION USING THE SUPPORT VECTOR MACHINE ENHANCED WITH A FEATURE-WEIGHT KERNEL

A Thesis

Submitted to the Faculty of Graduate Studies and Research

In Partial Fulfillment of the Requirements

for the Degree of

Master of Science

in

Computer Science

University of Regina

By

Songlun Zhao

Regina, Saskatchewan

August 2007

Abstract

With the popularization of the Internet and local networks, malicious attacks and intrusion events to computer systems are growing. The design and implementation of intrusion detection systems are becoming extremely important in helping to maintain proper network security. Support Vector Machines (SVM) as a classic pattern recognition tool, have been widely used in intrusion detection. However, conventional SVM methods do not involve the different characteristics of the features of data sets. This thesis proposes a new SVM model enhanced with a weighted kernel function based on features of the training data for intrusion detection. Rough set theory is used to perform the feature ranking and selection tasks of the enhanced SVM model in order to take advantage of both SVM and rough set theory. Based on the feature ranks resulting from Rough Set theory, a new algorithm is developed to calculate the feature weights in a kernel function for the enhanced SVM model. The new model is tested with two data sets namely, the KDD CUP 1999 dataset and the system call trace dataset from the University of New Mexico. When compared with the result from conventional SVM, the test provides evidence that the proposed model outperforms conventional SVM in precision, computation time, and false negative rate.

Acknowledgments

I would like to express my deep and sincere gratitude to my co-supervisors Dr. Jingtao Yao and Dr. Lawrence V. Saxton, for their patience, guidance and encouragement in carrying out this thesis. It was a great pleasure to me to conduct this thesis under their supervision.

I thankfully acknowledge Dr. Lisa Fan, for her valuable commentary and suggestions regarding my thesis.

Especially, I would like to give my thanks to my dear wife Jia Zhang, my parents, and friends for their constant love, encouragement and moral support.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

The Internet is used extensively nowadays. According to Internet World Stats [45], over a billion people use the Internet as of 2007. From 2000 to 2007, the number of Internet users has grown 214%. Along with the growing number of Internet users, the number of intrusions to computer systems through the network is increasing, and the type of attacks is getting more sophisticated [35]. Additionally, the risk of being attacked by intruders is getting higher for home and corporate computers. Network security has become a serious issue for both personal and corporate users.

Network security, in general, includes anti-spam and anti-virus, authorization, authentication and audit, firewalls, and intrusion detection. Anti-spam focuses on helping end-users reject or block email spam. Anti-virus techniques usually use software to identify, block and eliminate computer viruses or malicious software. Authorization, authentication and audit are used to protect computer resources, such as files, data, computer programs and devices from illegal accessing. The basic functionality

of firewalls is to separate computer networks into different zones. Each zone can have different security policies applied to prevent possible malicious actions to the computer systems and resources. Intrusion detection uses an Intrusion Detection System to detect possible malicious actions that bypass the firewalls and authentication. Intrusion detection is therefore considered as a second defense line to the computer systems and resources.

Most Intrusion Detection Systems(IDS) are software defense systems, which detect and repel hostile activities in networks [3]. An IDS is used to monitor the events occurring in a computer system or network, analyze the system events, detect suspected activities, and raise an alarm if an intrusion is detected. An IDS should be able to detect different kinds of malicious network actions and computer usages that cannot be prevented passively by a conventional firewall with its security policies. An IDS is more difficult to be implemented because the intrusion techniques keep changing, and getting more and more intelligent, which requires an IDS to be flexible and self-adjustable.

The key of an IDS is to discriminate hostile actions from normal user actions. Because the new work intrusion is complex and changing rapidly, artificial intelligence is used in the IDS, as a powerful tool dealing with intelligent behavior, learning and adaptation in systems. Various artificial intelligence and soft computing techniques have been used for analyzing the network attacking behaviors and detecting possible intrusions, such as association rules [31], fuzzy logic [31], genetic algorithms [34, 52], hidden Markov model [42], decision trees [25], Bayesian networks [25] and neural networks [36]. Among all these methods, Support Vector Machine (SVM) has been proved as an effective technique due to its nature of good generalization, and the

ability of overcoming the problem of high-dimensionality data [7, 51].

First introduced by Vapnik in 1995 [51], SVM is a modern computational learning method based on statistical learning theory. In SVM, data points in the original feature space are transformed to a high dimension feature space which has typically higher dimensions than the original feature space. In this high dimension space, an optimal hyperplane is determined by maximizing the distance from the hyperplane to the closest data points from two different classes. As a pattern recognition and classification technique, SVM has been used in a lot of different application areas. Osuna *et al.* [40] used SVM for human face recognition. Guyon *et al.* [15] used SVM to classify cancer genes. Leslie *et al.* [32] developed a special SVM kernel to classify different proteins. Rueda *et al.* [43] used SVM to analyze the currency crisis in economics.

1.2 Motivation

Although having better performance compared to some other traditional AI techniques, the conventional SVM could still be improved on intrusion detection in some aspects.

One possible improvement could be done would be increasing the speed and accuracy of SVM by feature reduction and feature selection. Let N_s be the number of Support Vectors (SVs), l be the number of training data samples, and d_L be the dimension of the input data. The computational complexity of SVM's training process is $O(N_s^3 + N_s l + N_s d_L l)$, and the complexity of intrusion detection process is $O(N_s d_L)$ [7]. From these two computational complexities we find that reducing data

features can increase both the training and test speed of SVM. In addition to the speed increasing, the feature selection can also increase the accuracy of intrusion detection systems. In the dataset of real-life intrusion detection, some features are redundant or less important than other features [24]. The redundant features make it harder to detect possible intrusion patterns [30].

Generally, feature selection methods search through the subsets of features and try to find the best subset which still contains enough information for intrusion detection. Kira *et al.* presented a features selection algorithm based on the Euclidean distance between data points [23]. Ben-Hur *et al.* used principal component analysis to find the subsets of features on gene expression data [6]. Other machine learning and artificial intelligence techniques, such as Bayesian networks, decision trees, and neural networks have also been used for feature selection [13]. In this study we use rough sets as our tool to do the feature selection. One unique characteristic of rough sets is that rough sets can generate multiple feature subsets or multiple reducts at one time.

Introduced by Zdzislaw Pawlak in the early 1980s, the theory of rough sets is a mathematical tool to deal with vagueness and uncertainty [41]. Rough sets use two sets, the lower and upper approximation sets to approximate the given set. The lower approximation set comprises all objects classified with certainty as belonging to the given set. The upper approximation set comprises objects which are possibly classified as belonging to the given set. A rough set is any set expressed in terms of its upper and lower approximations. Rough sets theory has proved its advantages on feature analysis and feature selection [16, 17, 55].

In this thesis, a SVM model enhanced with a feature-weights kernel will be presented. First, the rough sets technique is used to calculate feature reducts and feature

ranks for a feature-weight kernel. Second, an algorithm is developed to calculate the value of each feature weight from the feature reducts. Third, the feature weights are applied to the conventional SVM kernel function to form the feature-weights kernel and SVM model. Lastly the SVM model enhanced with the feature-weight kernel is used to test two intrusion detection datasets, namely the KDD dataset from KDD CUP 1999 and the UNM system call trace dataset from the University of New Mexico.

1.3 Outline

This thesis is organized as follows. In Chapter 2 we introduce the theories and concepts relating to this study, including the Intrusion Detection System, SVM and the theory of rough sets. In Chapter 3 we present the details of the enhanced SVM model including the new kernel function, related algorithms, theorems and theorem proving, and the implementation of the enhanced SVM model. In Chapter 4 we report the experiments including processes, results, analysis and discussions about the results. In Chapter 5 we conclude the whole study and give some possible perspectives.

Chapter 2

Background Information

In this chapter we provide some detailed reviews of three related works to this research. They are Intrusion Detection System (IDS), Support Vector Machine (SVM) and the theory of rough sets. In the first section we provide a brief introduction to IDS, including its definition, formation, architecture and some IDS using artificial intelligence (AI) techniques. In the second section we introduce the theory of SVM. We focus on the theoretical advantages of SVM that make it be a good tool for Intrusion Detection and the SVM kernel function. In the third section we provide the basic concept of feature selection and feature ranking, rough sets, as well as the usage of rough sets on feature selection.

2.1 Overview of Intrusion Detection Systems

2.1.1 Basic Concepts of Intrusion Detection Systems

As we mentioned in Section 1.1, most Intrusion Detection Systems are software defense systems. The major functions of an IDS are monitoring the events occurring

in a computer system or network, analyzing the system events, detecting suspected activities, and raising an alarm if an intrusion is detected. A typical IDS can be divided into three functional components [3]: an information source, an analysis engine and a decision maker. These three components can be applied on one single computer, or more commonly be applied on three or more different computers. Therefore the whole IDS can be a host system on one computer or be a distributed system on a local network or even across the Internet. Figure 2.1 shows the relations between these three components, the Internet and the protected systems.

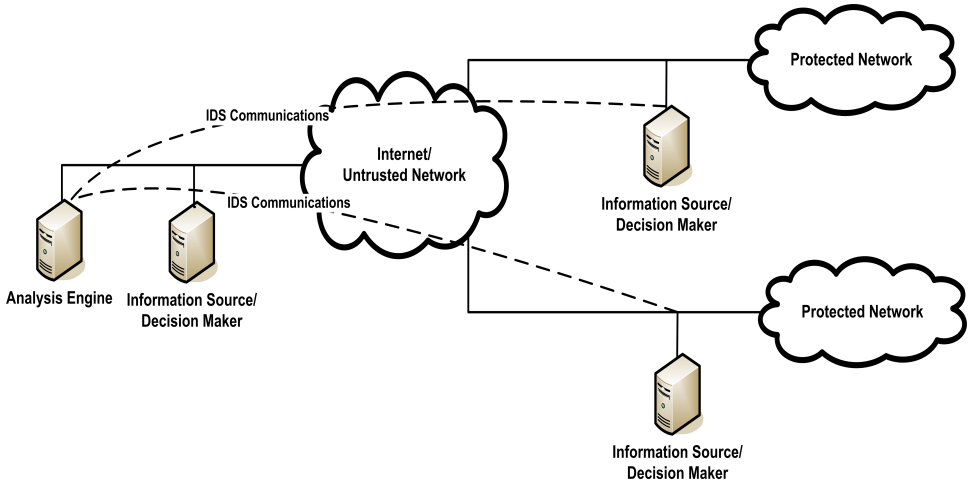

Figure 2.1: Relations between IDS components, Internet and protected system

Of the IDS, the first component, the information source is used to monitor the events occurring in a computer system or network. In the duration of monitoring, the information source provides a stream of event records for analysis. This component is working as an event generator. It senses and monitors different data sources, and generates event data that are well formatted and suitable for an analysis engine to do further analysis. Based on where their data come from, the data sources can be divided into three categories: the first one is the data source related to operating

systems, such as system calls and system logs. An example is the UNM dataset used in this study [39]. The second category is the network traffic monitors generating raw network packets. One example of this kind of data is the KDD dataset used in this study [10]. The third category is data collectors of different applications running on the network systems. The sensors of information source could be attached to operating systems, new work devices and/or application software. The information source captures all information using sensors, which are software processes continually scanning the different data sources. After capturing data, the information source executes a simple formatting, and sends the formatted data to the analysis engine.

The second component, the analysis engine is a key part of an IDS. An IDS relies on the analysis engine to find the signs of intrusions. All the artificial intelligence techniques can be applied to this component. The analysis engine analyzes, filters the information coming from the information source, and discards any irrelevant data in the information, thereby detecting suspicious activities. The analysis engine usually uses a detection policy database for analyzing. Depending on different intrusion detection approaches and techniques, there could be attack signatures, normal behavior profiles and necessary parameters (for example, thresholds) in the detection policy database.

The last component, the decision maker, applies some rules to the outcomes of the analysis engine, and decides what reactions should be done based on the outcomes of the analysis engine. The major reason for using the decision maker is to increase the usability of an IDS.

Figure 2.2 shows the details of the three components of an IDS and the data flows between these components.

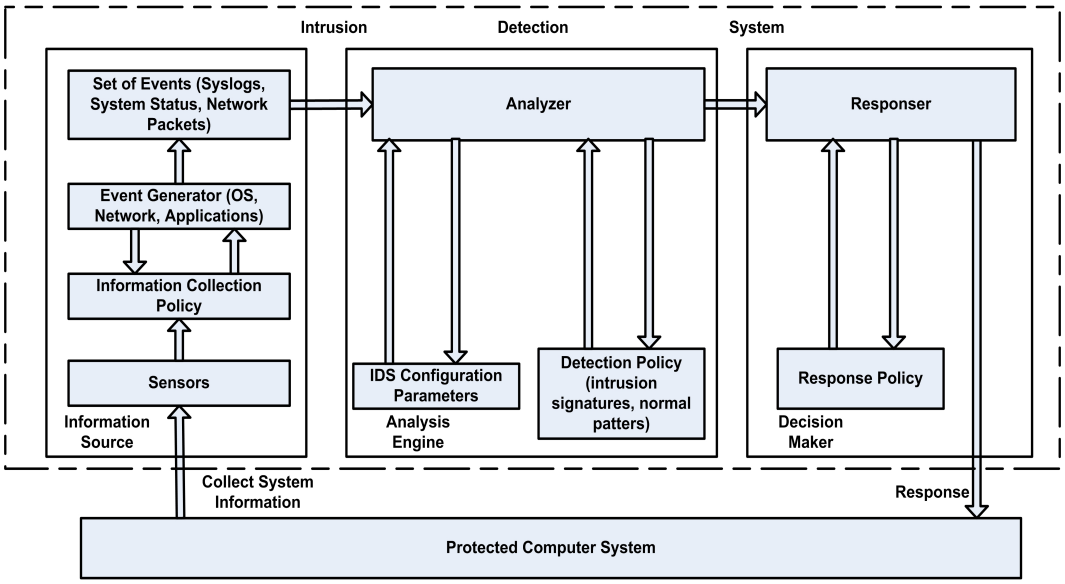

Figure 2.2: Component details of an IDS

Classified by the detection principle, there are two basic approaches used to detect intrusions. The first approach is called misuse detection. An IDS using this approach detects intrusion events which follow known patterns [3]. The patterns may describe a suspect set of sequences of actions or describe a unique feature of an intrusion. For example, a pattern could be a static bit string, which presents a specific virus. In practice, an expert system can be used as a misuse IDS. The knowledge of past intrusions can be encoded as rules by human experts or computers. These rules reflect the ordered sequence of actions that comprise one or multiple intrusion scenarios. Then the expert system uses the rules to detect the known intrusion activities. Kumar and Ilgun*et al.* [27, 19] proposed misuse detection systems based on the rule matching. In their models, the known intrusion signatures are encoded as rules, and are matched against the audit data introduced by the analysis engine. The primary limitation of misuse detection is that it cannot detect possible novel intrusions, i.e., events that

never happened or captured in the system. Further, it is highly time consuming to specify the intrusion patterns for a misuse IDS.

To overcome limitations of misuse detection, the second approach called anomaly detection is proposed. An anomaly detection based system analyzes the event data of a training system, and recognizes the patterns of activities that appear to be normal [3]. If a test event lies outside of the patterns, it is reported as a possible intrusion. From this point of view, the anomaly intrusion detection can be considered as a classification technology. We first use some training data to train a model, and get a discriminant function or a classifier. Then we use this discriminant function to test and classify new coming data. In this article we focus on anomaly intrusion detection. From the viewpoint of classification, the major work of building an anomaly IDS is to build a classifier which can classify normal event data and intrusion event data from an original dataset. This classification process consists of two main steps, which are training the parameters of a classifier from a training dataset and using this classifier to classify a test dataset.

2.1.2 Intrusion Detection Systems Using AI Techniques

Many AI techniques have been used in IDS. One of the first well-known IDS is MIDAS [44]. MIDAS is built as an expert system. Human experts analyze the logs of computer systems and networks, recognize some rules detecting intrusions and store the rules in the database of the expert system. The authors of MIDAS realized the large workload of rule recognition so they introduced a mechanism that automatically analyzes the data, recognizes and populates new rules or updates the old rules in the database of the expert system. MIDAS is still a misuse system, though the authors

improved the rule generation mechanism.

Another popular IDS is called NIDES [2]. As a comprehensive IDS, NIDES implements both misuse and anomaly detection approaches. NIDES implements anomaly detection using "Profiles". These profiles are actually patterns presenting the normal system activities. The IDS monitors and analyzes all kinds of data, such as CPU usage, command usage, and network activities, then generates profiles. The profiles in NIDES are usually updated once a day to reflect new changes. Besides the automatically updating, the system administrators can manually add extra information to the profiles, for example certain date or users information. Including human interference makes the NIDES have the ability of a misuse system.

From the introduction of the two IDS systems, we can see that both misuse and anomaly detection approach in IDS need to analyze data and recognize some new rules or patterns. The main challenge of an IDS is to find the patterns efficiently and accurately, which relies on the detection techniques applied to the analysis engine. Since the intrusion detection can be considered as a classification or pattern recognition issue, many artificial intelligence and soft computing techniques have been used to analyze data and find the patterns of normal behaviors or intrusions. Qiao *et al.* [42] presented a detection method by using a hidden Markov model to analyze the system call traces from the UNM dataset. Lee *et al.* [31] established an anomaly detection model that integrated association rules and frequency episodes with fuzzy logic to produce patterns for intrusion detection. Mohajeran *et al.* [36] developed an anomaly IDS that combines neural networks and fuzzy logic to analyze the KDD dataset. Wang *et al.* [52] applied genetic algorithms to optimize the membership function for mining fuzzy association rules from intrusion data.

2.1.3 Intrusion Detection Systems Using SVM

Among all the AI and software computing techniques, SVM is one of the most popular methods used for intrusion detection. SVM could be used in three different ways in the intrusion detection process. First, SVM could be used directly to find the pattern of normal activities of a computer system. Laskov *et al.* [28] implemented the incremental SVM for network traffic monitoring. Tian *et al.* [48] presented an anomaly detection method on a sequence-based dataset by using one-class SVM. Zanni *et al.* [57] developed a parallel software for training large scale SVM. Deng *et al.*[11] used SVM to detect intrusions on wireless networks . Yao *et al.* [54] presented a rough sets enhanced SVM model for intrusion detection on both sequence based and feature based datasets. Second, SVM could be used to find the important features of data. Sung *et al.* [46] used SVM and neural network to rank the features of an intrusion detection dataset. Third, SVM could cooperate with other AI and software computing techniques to solve the intrusion detection problems. Chen *et al.* [9] used genetic algorithms to analyze and optimize the data, then used SVM to classify the optimized data. Tsang *et al.* [50] presented a computational geometry algorithm adopting SVM kernel function to increase the speed of SVM on large scale intrusion dataset.

Compared to other AI and software computing techniques, SVM is one of the best methods in terms of accuracy and speed. Lazarevic *et al.* [29, 12] used SVM, neural networks, density based local outlier and Mahalanobis-distance based outlier to analyze unlabeled intrusion detection data, and discovered that SVM was the best in terms of finding new intrusions. The detection rate of SVM is 7% higher than the rate

of neural networks, 11% higher than the rate of density based local outlier, and 32% higher than the rate of Mahalanobis-distance based outlier. Mukkamala *et al.* [37, 38] applied SVM to identify important features and detect intrusions. They concluded that SVM outperformed neural networks in the respect of scalability, training and test time, and detection accuracy. SVM can train with a larger number of patterns, while neural networks take a long time to train or fail to converge at all when the number of patterns gets large. The detection speed of SVM is much faster than the detection speed of neural networks. The accuracy of SVM is also higher than the accuracy of neural networks.

SVM is effective in intrusion detection mostly due to its nature of good generalization and the ability of overcoming the problem of high-dimensionality data [7, 51]. In the following section, we introduce SVM and analyze why SVM has good generalization ability and good handling on high-dimensionality data.

2.2 Overview of Support Vector Machine

In this section, we briefly explain the basic concept of SVM and the kernel functions used in SVM, then we provide some theory explanations for why SVM has good generalization ability and ability to handle very high dimensionality which are the major advantages of using SVM for intrusion detection.

2.2.1 Basic Concepts of SVM

From Section 2.1.2 we know that intrusion detection process can be considered as a pattern recognition and classification process. A general two-class classification

problem can be presented as: given l training data samples $(\vec{x_1}, y_1)..(\vec{x_i}, y_i)...(\vec{x_l}, y_l)$, find a discriminant function $f(\vec{x})$ such that $y = f(\vec{x})$, where $y_i = \{+1, -1\}$ is the class label for sample data point $\vec{x_i}$.

When the data is linear separable, the discriminant function is a optimal hyperplane which takes the form [7]

$$\vec{w} \cdot \vec{x} - b = 0, \tag{2.1}$$

where $\vec{w}$ is normal to the hyperplane, $|b|/||\vec{w}||$ is the perpendicular distance from the hyperplane to the origin, and $||\vec{w}||$ is the Euclidean norm of $\vec{w}$. All the data satisfy the constraints

$$\vec{w} \cdot \vec{x_i} - b >= 1 \quad for\ y_i = 1; \qquad \vec{w} \cdot \vec{x_i} - b <= 1 \quad for\ y_i = -1. \tag{2.2}$$

We could define two hyperplanes which are parallel to the optimal hyperplane. These two hyperplanes take the form [7]

$$\vec{w} \cdot \vec{x} - b = -1; \qquad \vec{w} \cdot \vec{x} - b = 1. \tag{2.3}$$

The data points that lay on these two hyperplanes are the data points closest to the optimal hyperplane within their own classes. These data points are called Support Vectors (SVs). This is where the name Support Vector Machine comes from.

The optimal hyperplane is determined by maximizing its margin to different classes of data points, which means to find the two hyperplanes that maximize the distance to each other. This distance is $2/||\vec{w}||$. So the problem of finding the optimal hyperplane is transferred to minimize $||\vec{w}||$ under the above two constraints.

The final linear discriminant function is given as [7]

$$f(\vec{x}) = sgn(\sum_{i=1}^{l} \alpha_i y_i (\vec{x_i} \cdot \vec{x} - b)), \tag{2.4}$$

where l is the number of support vectors, $y_i \in \{-1, +1\}$ is the label associated with the training data, $0 \leq \alpha_i \leq C,\ C > 0$ is a constant, $\vec{x_i}$ is the support vectors found from the training process.

In the case that data points are not linear separable, SVM transforms the data points to another higher dimensional space in which the data points will be linear separable. If we define the transforming as

$$\Phi : R^d \rightarrowtail H, \tag{2.5}$$

the linear discriminant in space H or the nonlinear discriminant function in space R^d is given as

$$f(\vec{x}) = sgn(\sum_{i=1}^{l} \alpha_i y_i (\Phi(\vec{x_i}) \cdot \Phi(\vec{x}) - b). \tag{2.6}$$

2.2.2 Kernel Functions of SVM

To directly solve Φ in Equation 2.6 is difficult. SVM uses the "kernel trick" to solve the problem. The kernel trick uses a kernel function K such as

$$K(\vec{x_i}, \vec{x_j}) = \Phi(\vec{x_i}) \cdot \Phi(\vec{x_j}) \tag{2.7}$$

to generate a nonlinear discriminant function. We only need to use K in SVM and never need to explicitly know what Φ is.

The kernel function K exists if and only if the Mercer's Condition is satisfied [7]: there exists a mapping Φ and a kernel expansion $K(\vec{x_i}, \vec{x_j}) = \Phi(\vec{x_i}) \cdot \Phi(\vec{x_j})$ if and only

if, for any $g(\vec{x})$ such that

$$\int g(\vec{x})^2 d(\vec{x}) \quad is\ finite, \tag{2.8}$$

then

$$\int K(\vec{x_i}, \vec{x}) g(\vec{x_i}) g(\vec{x_j}) d(\vec{x_i}) d(\vec{x_j}) \geq 0. \tag{2.9}$$

The final format of nonlinear discriminant function of SVM is given as [7]

$$f(\vec{x}) = sgn(\sum_{i=1}^{l} \alpha_i y_i K(\vec{x_i}, \vec{x}) - b). \tag{2.10}$$

2.2.3 Generalization Ability of SVM

In this section we discuss Structural Risk Minimization which makes SVM have a good generalization ability.

Suppose y is the class label of data point $\vec{x}$, $f(\vec{x})$ is the discriminant function. The classification error can be measured as [7]

$$E(y, f(\vec{x})) = \begin{cases} 0 & \text{if y=}f(\vec{x}) \\ 1 & \text{otherwise} \end{cases} \tag{2.11}$$

If there is a discriminant function with general adjustable parameter set λ, given a general classification problem, the accuracy of this function can be measured by the expectation of test error, given as [7]

$$R(\lambda) = \int E(y, f(\vec{x}, \lambda)) dP(\vec{x}, y). \tag{2.12}$$

$R(\lambda)$ is called the actual risk. $P(\vec{x}, y)$ is an unknown probability distribution, which is not available for most classification cases. $f(\vec{x}, \lambda)$ is a possible mapping from $\vec{x}$ to y. λ is the set of adjustable parameters in the mapping. $R(\lambda)$ is the quantity

which we are ultimately interested in. Because $P(\vec{x}, y)$ is hard to know, we must use a empirical risk $R_{emp}(\lambda)$ to estimate $R(\lambda)$ as

$$R_{emp}(\lambda) = \frac{1}{l}\sum_{i=1}^{n} E(y, f(\vec{x}, \lambda)), \tag{2.13}$$

where l is the number of training data samples.

Some of the classification training algorithms implement Empirical Risk Minimization (ERM), i.e minimize $R_{emp}(\lambda)$ using Maximum Likelihood estimation for the parameter λ. But an algorithm using only ERM can result in overfitting, which can well classify training data samples, but has poor accuracy in testing data samples. In SVM, Vapnik used Structural Risk Minimization (SRM) [51] to find the learning machine that has a good trade-off of low $R_{emp}(\lambda)$ and good generalization ability.

According to SRM, choose some η such that $0 \leq \eta \leq 1$, the following bound holds between $R(\lambda)$ and $R_{emp}(\lambda)$ [51] given as

$$R(\lambda) \leq R_{emp}(\lambda) + \sqrt{\frac{h(\log(2l/h) + 1) - \log(\eta/4)}{l}}, \tag{2.14}$$

where h is a non-negative integer called the Vapnik Chervonenkis (VC) dimension. VC dimension decided the classification capacity of a classifier. The higher h means higher classification capacity [7]. If its capacity is too high, which means the classifier has the overfitting problem. The second term on the right hand side is called VC Confidence. The whole right hand side is called the bound on actual risk $R(\lambda)$. From Equation 2.14 we can see that to minimize $R(\lambda)$, one has to minimize the bound, which is equal to minimize $R_{emp}(\lambda)$ and VC Confidence simultaneously. We can prove that given a fixed, sufficiently small η, VC Confidence is monotonic in VC dimension h [7].

We can see from Equation 2.13 that $R_{emp}(\lambda)$ depends on a particular function in the function set $f(\vec{x}, \lambda)$. Because VC dimension h is an integer, VC confidence depends on a chosen subset of function set $f(\vec{x}, \lambda)$. With different value of h, a "structure" is introduced by dividing the entire function set into nested subsets. For each subset, we must be able either to compute h or to get a bound on h itself. SRM then consists of finding that subset of functions which minimize the bound on $R(\lambda)$. This can be done by training a series of discriminant functions $f(\vec{x}, \lambda)$, one for each subset with a certain h, where for a given subset the goal of training is to minimize $R_{emp}(\lambda)$. One then takes that trained machine in the series whose sum of $R_{emp}(\lambda)$ and VC confidence is minimal.

SVM's training is based on SRM. By minimizing VC confidence, SVM guarantees the VC dimension h is not too high, therefore avoiding the overfitting problem.

2.2.4 High Dimensionality Handling of SVM

From Equation 2.6 we can see that when it does non-linear classification, the SVM depend on the data only through the function $\Phi(\vec{x_i}) \cdot \Phi(\vec{x})$. If we replace $\Phi(\vec{x_i}) \cdot \Phi(\vec{x})$ by $K(\vec{x_i}, \vec{x_j})$ everywhere in the training algorithm, the algorithm will produce a SVM which is in a very high dimension d_H. Furthermore, comparing the linear SVM Equation 2.4 and the non-linear SVM Equation 2.10, training SVM on the transformed data takes roughly the same amount of time as training SVM on the un-transformed data. This means we could build a non-linear classifier with the similar amount of time we build a linear classifier. So the training time is linear to the dimension of data. A simple analysis shows the computational complexity of SVM algorithms. Let N_s be the number of Support Vectors (SVs), l be the number of

training data samples, and d_L be the dimension of the input data. The computation complexity of SVM's training process is $O(N_s^3 + N_s l + N_s d_L l)$, and the complexity of intrusion detection process is $O(N_s M)$, where M is the number of operations to evaluate the kernel [7]. For RBF kernel, M is $O(d_L)$ and the complexity of detection process for SVM with RBF kernel is $O(N_s d_L)$ [7].

2.3 Rough Sets and Reducts

Rough sets theory has several advantages that makes it a good choice for feature selection and reduction. First, rough sets theory uses a clear mathematical manner to discover hidden patterns in data, and the redundancies and dependencies between the data features [41]. Normal feature selection methods simply select some subset of the features set and do experiments against the feature subset. The feature selection result is more experimental instead of mathematical. Rough sets use lower and upper approximations, and decision tables as the mathematical table to find the feature reducts. We consider that rough sets are more elegant in feature selection. Second, normal feature selection methods only find one feature subset or reduct. Rough sets can find out all of the possible reducts. We want to use as many reducts as we have to evaluate the importance of the features. We consider that the multiple reducts have a better generalization ability to present the data than a single reduct.

Because of the above advantages of rough sets, in this study we use rough sets and reducts as tools to analyze the importance of features of a dataset. In this section we briefly introduce the basic concepts of rough sets and reducts.

Originally introduced by Zdzislaw Pawlak in the early 1980s [41], rough sets are

a very general technique for approximation of a given set. As we introduced in Section 1.2, rough sets use two sets, the lower and upper approximation sets to approximate a given set. The lower approximation set comprises all objects classified with certainty as belonging to the given set. The upper approximation set comprises objects which are possibly classified as belonging to the given set. A rough set is any set expressed in terms of its upper and lower approximations. We could explain rough set using the following algebraic notions [56]: Let U denote the universe, the equivalence relation $\Re$ partitions U into disjoint subsets. Given an arbitrary set $X \subseteq U$, it may be impossible to describe X precisely. We can approximate X using a pair of lower and upper approximation sets, the lower approximation as

$$\underline{apr}(X) = \bigcup_{[x]_{\Re} \subseteq X} [x]_{\Re}, \tag{2.15}$$

and the upper approximation as

$$\overline{apr}(X) = \bigcup_{[x]_{\Re} \cap X \neq \emptyset} [x]_{\Re}, \tag{2.16}$$

where

$$[x]_{\Re} = \{y | x \Re y\} \tag{2.17}$$

is the equivalence class containing x.

The main problems that rough set deal with are feature reduction, discovery of data dependency, estimation of data importance, and approximate classification of data [26, 41].

Reducts, in rough set theory, is standard for the subsets of the full list of features containing no loss of quality and accuracy of approximation. A reduct is the minimal feature set having the same classification information of the whole feature set [26].

A general hill climbing algorithm to calculate reducts is showed in algorithm 1:

Algorithm 1: Reducts Calculation

Input : Full feature list F, data class C
Output: Reduct R

```
1  R ← {};
2  while γ(R, D) ≠ γ(C, D) do
3  |  T ← R;
4  |  foreach x ∈ (C − R) do
5  |  |  if  γ(R ∪ {x}, D) > γ(T, D) then
6  |  |  |  T ← R ∪ {x};
7  |  |  end
8  |  end
9  |  R ← T;
10 end
```

In practice, there are many artificial intelligence and statistics techniques, such as genetic algorithms [53] and heuristic search [4] that are used to calculate rough set reducts. In this study, we use Rough Set Exploration System (RSES) [5] as the tool to calculate Reducts. RSES is a Java implementation of the rough sets toolkit, which can calculate reducts, simplify data table, or classify data. RSES can use either Genetic Algorithm or exhaustive search to calculate reducts. We use it to deal with both big and small datasets.

Chapter 3

SVM Model with a Feature-weight Kernel

In this chapter we provide the details of the SVM model enhanced with a new feature-weight kernel. In the first section we introduce the enhancements of the new model compared to the conventional SVM model, and describe the formation and the structure of the model. In the second section we introduce the new feature-weight SVM kernel. In the third section, we theoretically prove the eligibility of the feature-weight kernel in our SVM model. Finally, we introduce our implementation of the enhanced SVM model.

3.1 SVM Model Enhanced with a New Kernel

As we mentioned in Section 2.2.2, the kernel function is the key to map SVM from two-dimension space to high dimension space. The function decides the shape of the finial discriminant function in the data space. It is a natural thought to change the shape of the discriminant function to make it more suitable for the certain type of data. This means improving SVM by changing the SVM kernel function. Amari *et al.* [1] introduced a magnification factor into the SVM kernel. Generated from

the training data, the magnification factor can adjust the spatial resolution around the SVM discrimination hyper planes. Tan *et al.* [47] adds some coefficients and parameters to the kernel function to extend the generalization power of the SVM. The coefficients and parameters are generated by minimizing the upper bound of the Vapnik Chervonenkis (VC) dimension. Tian *et al.* [48] applied dynamic programming and bit-parallelism techniques to efficient computation of SVM kernel and adapted the SVM kernel for anomaly detection of short sequences of system calls. This anomaly detection is part of the intrusion detection. Leslie *et al.* [32] introduced several new families of string kernels designed for use with SVM for classification of protein sequence data. Suykens *et al.* [20] enhanced least squares (LS) SVM with weights associated in training process to overcome the two potential drawbacks of LS-SVM. These two drawbacks are losing generalization ability and low accuracy for data with un-Gaussian distribution.

In this study, we propose a SVM model enhanced with a new feature-weight kernel. We improve the conventional SVM kernel by adding feature weights to it. The feature weights are calculated by analyzing training data with the rough sets technique. After the calculation, the feature weights are integrated into the conventional SVM kernel in such a way that the new kernel still satisfies Mercer's Condition. From our experiments we discover that, when used in the intrusion detection area, the SVM with the new kernel function is faster and more accurate than the conventional SVM with common kernels (linear, polynomial, radial basis function, etc.).

The major targets of the enhanced SVM model are:

- Reduce the redundant features to increase the speed of intrusion detection.

- Apply information of features onto the SVM classification process to increase the accuracy of intrusion detection.

In real-life datasets, some of the features are redundant or less important than other features [24]. Feature reduction is feasible in practice for two main reasons: 1) Many of the features in a real-life dataset are correlated to each other [24]. For example, when the voltage is fixed, the power and current in an electric system are inherently linked. 2) Some features may carry low signal-to-noise ratios. Other features may simply be unrelated to the task, so their values are noise. Before the feature reduction is conducted, these noisy features' relations to the application domain are unknown. For example in this study, the System Call dataset contains sequence data. After mapping system call sequences to feature-value pairs, we will see that the dataset becomes a high dimensional and extremely sparse matrix. In the after-mapping System Call dataset, most of the features are unrelated to our task.

Although SVM uses kernel techniques to overcome the problem of high dimension, in Section 2.2.4 we showed that the dimension of data is still linear to the computational complexity of SVM. From Equation 2.10 we can see that the information of features are integrated into SVM through its kernel function, so we could enhance the conventional SVM by adding a feature reduction ability into SVM kernel function.

The second enhancement that we want to apply to SVM is to apply features information into SVM through the SVM kernel function. Various SVM kernel functions are proposed for users to choose for different applications [7, 21]. Some of the popular kernel functions are the linear function $K(\vec{x_i}, \vec{x}) = \vec{x_i} \cdot \vec{x}$, polynomial function $K(\vec{x_i}, \vec{x}) = (s(\vec{x_i} \cdot \vec{x}) + c)^d$, sigmoid function $K(\vec{x_i}, \vec{x}) = \tanh(s(\vec{x_i} \cdot \vec{x}) + c)$, and radial basis function (RBF) $K(\vec{x_i}, \vec{x}) = e^{-||\vec{x_i} - \vec{x}||^2 \cdot \gamma}$. The kernel functions decide the shape

of the hyperplane in the data space, so we need to choose different kernels for different datasets with different data distributions. However, all of these kernel functions do not involve the differences between the features of data. Actually, from the general SVM kernel function format $K(\vec{x_i}, \vec{x})$, we can see that, in the conventional SVM, all features of the training or test datasets are treated equally. Treating all features equally may not be efficient and it may affect the accuracy of SVM. Based on the SVM's mechanical torque analogy [7], considering the case in which the data point are in a two dimensional space R^2, the i'th support vector generates a force $F_i = \alpha_i y_i \vec{I}$ on a stiff sheet lying along the decision hyperplane. The total torques generated by all support vectors satisfies the condition of mechanical equilibrium as

$$\sum Torques = \sum_i S_i \bigwedge (\alpha_i y_i \vec{I}) = \vec{I} \bigwedge \vec{w} = 0, \tag{3.18}$$

where $\vec{I}$ is the unit vector in the direct $\vec{w}$, which is perpendicular to the decision hyperplane. The normal to the hyperplane $\vec{w} = \sum_{i=1}^{N_s} \alpha_i y_i \vec{x}_i$. The decision hyperplane is decided by Equation 3.18. From Equation 3.18, it is a natural thought that we can put some factors or coefficients on each feature of the data to increase or decrease the torque point to the direction of this feature. Those important features would contribute a higher value of torque, which should increase the accuracy of SVM on intrusion detection, while still maintaining a good generalization ability. We call the coefficients applied to the features as feature weights. To implement this thought, we enhance the conventional SVM kernel with feature weights.

3.2 The SVM Kernel Enhanced with Feature Weights

3.2.1 Feature Weights Calculation

In Section 2.3, we introduced rough sets technique and its advantage for feature selection and intrusion detection. In this section, we present an algorithm based on rough set and reducts to calculate the weights in the enhanced SVM kernel function.

We propose that the feature weights should have the following attributes:

- The weights should include information for both feature discrimination and feature reduction.

  Because we want the new SVM kernel to be faster and more accurate than the conventional SVM kernel, the feature weights should improve the SVM kernel in feature discrimination and feature reduction.

- The weights calculation should be independent to any particular SVM kernels.

  As we mentioned in Section 3.1, different datasets and data distributions need different SVM kernels to classify them. If the weights depend on a certain type of kernel, the use of the weights are restricted.

- The weights calculation should be fully automatic.

  We need to use the feature weights in intrusion detection without any human interference. An IDS system needs to be run continually to monitor and protect computer networks and systems. As we mentioned in Section 2.1.1, the intrusions keeps changing. The SVM kernel and the associated feature weights are

part of the patterns for detection intrusions. Updating patterns is also a continual work. It would be a huge workload for people if an IDS system requires human to update the patterns.

Because the feature weighs requires the above attributes, we develop a feature weights calculation algorithm with the following basic principles:

- If a feature is not in any reducts then the weight of this feature is 0.
- The more times a feature appears in the reducts, the more important this feature is.
- The fewer the number of features in a reduct, the more important the features appear in this reduct. If a reduct has only one feature, the feature belonging to this reduct is the most important.

Based on the above principles, we propose an algorithm as depicted in Algorithm 2 that adopts rough set theory to rank features and calculate feature weights.

The input of the algorithm is the training dataset. The output of the algorithm is a vector $\vec{w}$. The dimension of the vector equals to the number of the features of the training dataset. Line **1** uses RSES Rough Set Exploration System (RSES) [5] to calculate the reducts from the training dataset and put the reducts in set R. Line **2** gets the number of features in the training dataset D. Line **3** gets the number of reducts in the reduct set R. Lines **5-7** initialize the output vector $\vec{w}$. Lines **9-17** repeat to calculate the value of each feature w_i in the output vector $\vec{w}$. Because $m > 0$ is always true, $w_i > 0$ is always true. Guaranteeing w_i is always greater than zero is very important. It makes the SVM kernel with feature weights $\vec{w}$ satisfies Mercer's

Algorithm 2: Feature Weights Calculation

Input : Dataset D.
Output: A weight vector $\vec{w}$.

```
1  R ← Find out all the reducts of D using rough sets;
2  N_feature ← number of features in D;
3  N_reduct ← number of reducts in R;
4  //Initialize the weight of each feature.
5  for (i ← 0 to N_feature) do
6  |  w_i ← 0;
7  end
8  // Calculate the weight of each feature.
9  for (i ← 0 to N_feature) do
10 |  for (j ← 0 to N_reduct) do
11 |  |  if (feature i in the j^th reduct R_j) then
12 |  |  |  m ← number of features in R_j;
13 |  |  |  w_i ← w_i + 1/m;
14 |  |  end
15 |  end
16 |  Scale w_i into the interval [0, 100];
17 end
18 Return w⃗
```

Condition, which guarantees the eligibility of the new SVM kernel [7]. We will prove this conclusion in Section 3.3. Line **16** scale the result into interval 0 - 100.

We conclude some advantages of the algorithm are the following:

- Instead of simply ranking the importance of different features, this algorithm calculates the actual value of the feature importance, which makes it possible to bring the feature importance into a SVM kernel function.

- This value of importance is calculated from multiple reducts generated from the rough sets algorithm. With more than one reduct, the feature weights reflect more information of the dataset and is more stable compared to generating feature weights from only one reduct [4].

- Feature ranking and selection are done in the same process. Algorithm 2 removes the redundant features and calculates the feature weights at the same time. This makes it possible that the enhanced SVM model has both fast processing speed and better accuracy than the conventional SVM. After the feature ranking process, we consider those features with a 0 weight as the least important features, and thus delete them.

3.2.2 The SVM Feature-weight Kernel Function

The final SVM kernel we proposed is enhanced in both feature discrimination and feature reduction. This kernel calculates the importance of different features and reduces redundant features at the same time. A nonlinear discriminant function with feature weights is formulated as

$$f(\vec{x}) = sgn(\sum_{i=1}^{l} \alpha_i y_i K(\vec{w}\vec{x_i}, \vec{w}\vec{x}) + b), \tag{3.19}$$

where $\vec{w}$ is the output vector of algorithm 2. When using the new kernel function, we form this vector $\vec{w}$ as a diagonal matrix as

$$\begin{pmatrix} w_0 & 0 & ... & 0 \\ 0 & w_1 & ... & 0 \\ ... & ... & ... & ... \\ 0 & ... & ... & w_l \end{pmatrix}. \tag{3.20}$$

For convenience, we still call $\vec{w}$ as a vector in the rest part of this article. The value of each element w_i in $\vec{w}$ is the weight of feature i. $\vec{w}$ enhances the SVM kernel function with the feature's information. Besides, if w_i equals to 0, that means feature i is redundant and we could remove the feature when using the discriminant function to

classify data. The $\vec{w}$ improves the conventional SVM kernel function on both feature discrimination and feature reduction.

3.3 Eligibility of The SVM Feature-Weight Kernel

In this section, we prove the eligibility of the new SVM kernel with Feature-Weight. From Section 2.2.1 we know that all SVM kernel functions, including the enhanced SVM kernel function, have to satisfy Mercer's Condition. That means we need to prove that $K(\vec{x_i}, \vec{x}) = \Phi(\vec{x_i}) \cdot \Phi(\vec{x})$ exists if and only if, for any $g(x)$ such that

$$\int g(\vec{x})^2 d(\vec{x}) \quad is\ finite \tag{3.21}$$

then

$$\int K(\vec{x_i}, \vec{x}) g(\vec{x_i}) g(\vec{x_j}) d(\vec{x_i}) d(\vec{x_j}) \geq 0. \tag{3.22}$$

In this section we prove that the two frequently used kernels, linear and RBF SVM kernels, can be enhanced with weights and satisfy Mercer's Condition. Because the weight-enhanced kernel function $K(\vec{w}\vec{x_i}, \vec{w}\vec{x})$ is applied to a known SVM kernel function $K(\vec{x_i}, \vec{x})$, here we only need to prove that if

$$\int K(\vec{x_i}, \vec{x}) g(\vec{x_i}) g(\vec{x}) d(\vec{x_i}) d(\vec{x}) \geq 0 \tag{3.23}$$

then

$$\int K(\vec{w}\vec{x_i}, \vec{w}\vec{x}) g(\vec{x_i}) g(\vec{x}) d(\vec{x_i}) d(\vec{x}) \geq 0. \tag{3.24}$$

First we prove two preparing theorems.

Theorem 3.1. *If kernels satisfy Mercer's Condition, the sum of these kernels also satisfies Mercer's Condition. That is if*

$$K(\vec{x_i}, \vec{x}) \tag{3.25}$$

satisfies Mercer's Condition, then

$$\sum_{i=1}^{l} c_i K(\vec{x_i}, \vec{x}) \tag{3.26}$$

also satisfies Mercer's Condition. Where $c_i \geq 0$ is a nonnegative constant.

Proof. Because $K(\vec{x_i}, \vec{x})$ is kernel function, from Equation 3.22 we have

$$\int K(\vec{x_i}, \vec{x}) g(\vec{x_i}) g(\vec{x_j}) d(\vec{x_i}) d(\vec{x_j}) \geq 0.$$

Since $c_i \geq 0$, we have

$$\int c_i K(\vec{x_i}, \vec{x}) g(\vec{x_i}) g(\vec{x}) d(\vec{x_i}) d(\vec{x}) \geq 0,$$

and

$$\int \sum_{i=1}^{l} c_i K(\vec{x_i}, \vec{x}) g(\vec{x_i}) g(\vec{x}) d(\vec{x_i}) d(\vec{x}) \geq 0.$$

Therefore

$$\sum_{i=1}^{l} c_i K(\vec{x_i}, \vec{x})$$

satisfies Mercer's Condition. □

Theorem 3.2. *If kernels satisfy Mercer's Condition, the product of these kernels also satisfies Mercer's Condition. That is if*

$$K(\vec{x_i}, \vec{x}) \tag{3.27}$$

satisfies Mercer's Condition, then

$$\prod_{i=1}^{l} c_i K(\vec{x_i}, \vec{x}) \tag{3.28}$$

also satisfies Mercer's Condition, where $c_i \geq 0$ *is a nonnegative constant.*

Proof. Because $K(\vec{x_i}, \vec{x})$ is kernel function, from Equation 3.22 we have

$$\int K(\vec{x_i}, \vec{x}) g(\vec{x_i}) g(\vec{x_j}) d(\vec{x_i}) d(\vec{x_j}) \geq 0.$$

Since $c_i \geq 0$, we have

$$\int c_i K(\vec{x_i}, \vec{x}) g(\vec{x_i}) g(\vec{x}) d(\vec{x_i}) d(\vec{x}) \geq 0,$$

and

$$\int \prod_{i=1}^{l} c_i K(\vec{x_i}, \vec{x}) g(\vec{x_i}) g(\vec{x}) d(\vec{x_i}) d(\vec{x}) \geq 0.$$

Therefore

$$\prod_{i=1}^{l} c_i K(\vec{x_i}, \vec{x})$$

satisfies Mercer's Condition. □

Based on the theorem 3.1 and theorem 3.2, we can prove that both feature-weight enhanced linear kernel and RBF kernel satisfy Mercer's Condition.

Theorem 3.3. *Weight-enhanced liner kernel function*

$$K(\vec{w}\vec{x_i}, \vec{w}\vec{x}) = \vec{w}(\vec{x_i} \cdot \vec{x}) \tag{3.29}$$

satisfies Mercer's Condition.

Proof. From Equation 3.20 we have the weight-enhanced linear kernel function

$$\vec{w}(\vec{x_i} \cdot \vec{x}) = \begin{pmatrix} w_0 & 0 & ... & 0 \\ 0 & w_1 & ... & 0 \\ ... & ... & ... & ... \\ 0 & ... & ... & w_l \end{pmatrix} (\vec{x_i} \cdot \vec{x}) = w_0(\vec{x_i} \cdot \vec{x}) + w_1(\vec{x_i} \cdot \vec{x}) + ... + w_l(\vec{x_i} \cdot \vec{x}).$$

From Alg. 2 we know that $w_i \geq 0$. Because $(\vec{x_i} \cdot \vec{x})$ satisfies Mercer's Condition, based on theorem 3.1, the sum of $(\vec{x_i} \cdot \vec{x})$ also satisfies Mercer's Condition. □

Theorem 3.4. *Weight-enhanced RBF kernel function*

$$K(\vec{w}\vec{x_i}, \vec{w}\vec{x}) = e^{-||\vec{w}(\vec{x_i} - \vec{x})||^2 \cdot \gamma} \tag{3.30}$$

satisfies Mercer's Condition.

Proof. From Equation 3.20 we have the weight-enhanced RBF kernel function

$$e^{-||\vec{w}(\vec{x_i} - \vec{x})||^2 \cdot \gamma} = e^{-|| \begin{pmatrix} w_0 & 0 & ... & 0 \\ 0 & w_1 & ... & 0 \\ ... & ... & ... & ... \\ 0 & ... & ... & w_l \end{pmatrix} (\vec{x_i} - \vec{x})||^2 \cdot \gamma}. \tag{3.31}$$

In Equation 3.31

$$-|| \begin{pmatrix} w_0 & 0 & ... & 0 \\ 0 & w_1 & ... & 0 \\ ... & ... & ... & ... \\ 0 & ... & ... & w_l \end{pmatrix} (\vec{x_i} - \vec{x})||^2 \cdot \gamma$$

can be presented as a polynomial

$$a_1||(\vec{x_i} - \vec{x})||^2 \cdot \gamma + ... + a_l||(\vec{x_i} - \vec{x})||^2 \cdot \gamma,$$

and the weight-enhanced RBF kernel function can be presented as

$$e^{-a_1||(\vec{x_i}-\vec{x})||^2\cdot\gamma}\cdot\ldots\cdot e^{-a_l||(\vec{x_i}-\vec{x})||^2\cdot\gamma}, \tag{3.32}$$

where $a_i \geq 0$. Based on theorem 3.2 we know that Equation 3.32 satisfies Mercer's Condition. Therefore the weight-enhanced RBF kernel function satisfies Mercer's Condition. □

Similarly we could also prove that if $d\ mod\ 2 = 0$, the weighted polynomial function $K(\vec{w}\vec{x_i}, \vec{w}\vec{x}) = (s\vec{w}(\vec{x_i}\cdot\vec{x}) + c)^d$ satisfies Mercer's Condition. In our experiments, the discriminant function with feature weight-enhanced RBF kernel

$$f(\vec{x}) = sgn(\sum_{i=1}^{l}\alpha_i y_i e^{-||\vec{w}(\vec{x_i}-\vec{x})||^2\cdot\gamma} + b) \tag{3.33}$$

is used.

3.4 Implementation of The SVM Model Enhanced with a Feature-weight Kernel

There are two popular implementations of the conventional SVM. The first one is SVMLight developed by Joachims [21], which is a light weight C implementation of the SVM algorithm. The SVMLight has been used in bioinformatics [18], text classification [49], intrusion detection [11] and many other areas. The second one is LIBSVM developed by Chang and Lin [8], which has different interfaces for different programming languages such as C, C++, Java and Perl etc.

To implement the SVM model with a feature weight kernel, we choose SVMLight as our prototype. The major reason we use SVMLight is that SVMLight is simple, and

it separates the kernel function of SVM into an independent module. It is easier for us to replace the conventional SVM kernel function with the feature weights enhanced SVM kernel function.

The modification on SVMLight for the implementation of the enhanced SVM model as following,

- Write a C program implementing algorithm 2. This program calculates feature weights from the rough set reducts and forms a diagonal matrix. This matrix represents the future weight vector $\vec{w}$ in Equation 3.19. The program is integrated into SVMLight as an independent module

- Modify the data preprocessing module of SVMLight to remove the redundant features. If the weight of a feature is equal to 0, we consider this feature is redundant, then we remove the feature from the dataset.

- Modify the SVMLight kernel module, change the kernel function into format as Equation 3.19.

The flowchart of the modified SVMLight program is shown in Figure 3.1

As we can see from Figure 3.1, the user interface of the enhanced SVM model implementation is not different from the SVMLight. But the internal process of the implementation has been enhanced with feature weights calculation and feature reduction. The users need to input parameters, training data and test data. The training data and the parameters are used to train the conventional SVM kernel. Besides, the training data is also input into the feature weight calculation algorithm to generate the feature weights. The new SVM kernel is generated by applying feature weights to the training out of the conventional SVM kernel. During the test process,

the test data are trimmed by a data preprocessing process which is applied the feature weights. After the process, all the redundant features are removed from the test data. Finally, the new SVM kernel enhanced with feature weight will classify the trimmed data to get the final results.

As we mentioned in Section 3.1, various kernel functions can be used on the conventional SVM model. SVMLight implements all of the linear function, the polynomial function, the sigmoid function, and the radial basis function. Because the enhancement is independent to the kernel functions, after our modification, the new SVMLight kernel module applies feature weights to all these kernel functions. The new kernel functions applied feature weights are that the linear function

$$K(\vec{w}\vec{x_i}, \vec{w}\vec{x}) = \vec{w}(\vec{x_i} \cdot \vec{x}), \tag{3.34}$$

the polynomial function

$$K(\vec{w}\vec{x_i}, \vec{w}\vec{x}) = (s\vec{w}(\vec{x_i} \cdot \vec{x}) + c)^d, \tag{3.35}$$

the sigmoid function

$$K(\vec{w}\vec{x_i}, \vec{w}\vec{x}) = \tanh(s\vec{w}(\vec{x_i} \cdot \vec{x}) + c), \tag{3.36}$$

and the RBF function

$$K(\vec{w}\vec{x_i}, \vec{w}\vec{x}) = e^{-||\vec{w}(\vec{x_i} - \vec{x})||^2 \cdot \gamma}. \tag{3.37}$$

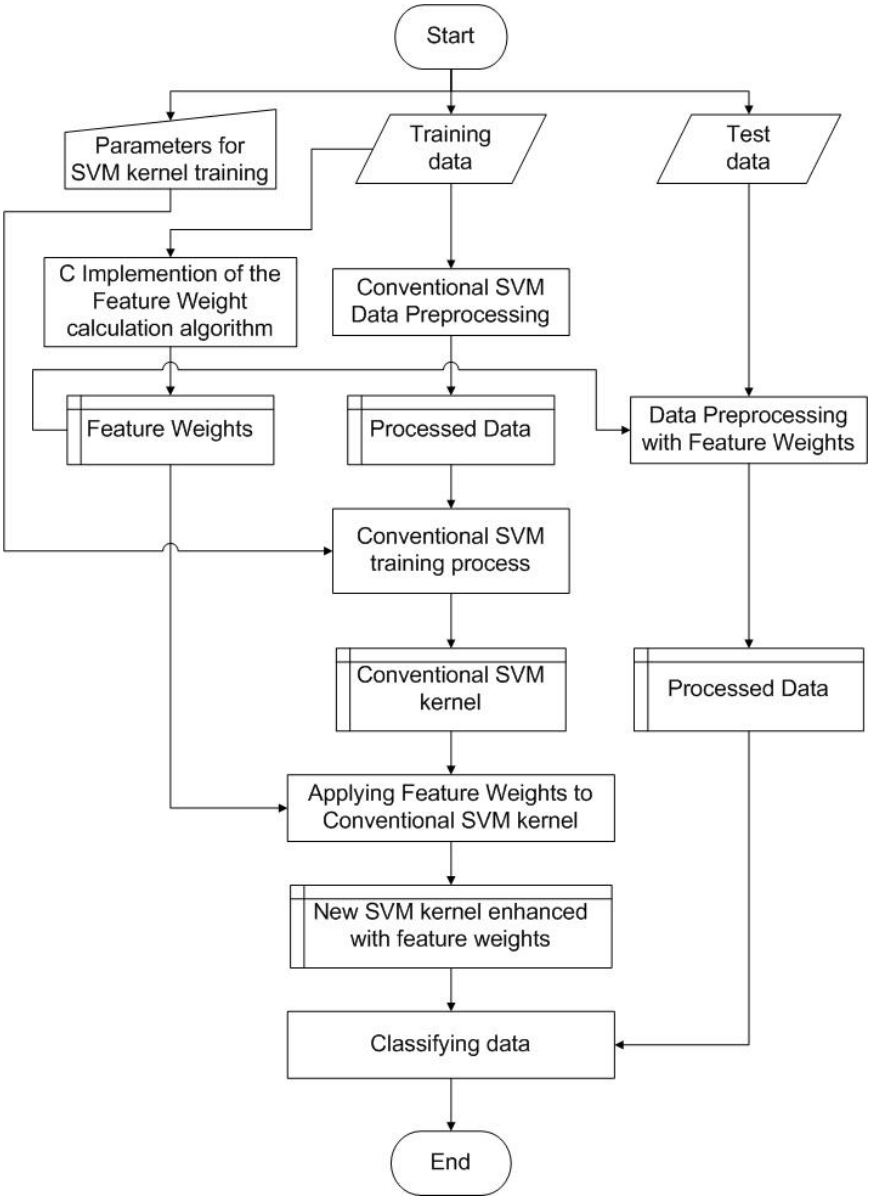

Figure 3.1: Flowchart of the modified SVMLight

Chapter 4

Experiments

In this chapter, we provide the details of the experiments carried out about using the Support Vector Machine with the feature-weight kernel for intrusion detection. In the first section we describe the performance measures that we used for the experiment. In the second section we present a common procedure using the enhanced SVM model in an IDS. In the third section and the fourth section, we present two experiments on the enhanced SVM model using two different datasets. The first experiment uses the UNM dataset [39], which includes System Call sequence data. The second experiment uses the KDD dataset [10], which presents basic network communication data. Network communication data and system call data are the most used data in intrusion detection and these data provide good samples of real life intrusions. We believe that the experiments on these two datasets could show that the enhanced SVM model can be used in various intrusion detection scenarios. Each experiment will be presented in the following sequence: dataset description, experiment procedures, experiment results, discussion and the comparison to the conventional SVM model. In the last section we summarize the experiments.

4.1 IDS Performance Measures

Speed measure

The speed of an intrusion detection system depends on the host computer on which it is running, the algorithm and the actual software implementation of the model.

To evaluate the speed of our model as objectively as possible, we run all the tests on a same computer with a Pentium IV 2.66 GHz CPU and 512 M RAM. Because software implementation of our model is based on SVMLight [21], when we compare our model with the conventional SVM, we use SVMLight as the conventional SVM software implementation and perform all the experiments on it.

Accuracy measure

We use measures[14] adapted from general measures used in information retrieval, to evaluate the accuracy of an SVM model. The measures are showed in table 4.1.

In table 4.1, for a certain dataset, the total number of records $=A+B+C+D$, the number of intrusion records $= A+C$, the number of normal records $= B+D$, the number of records that are classified as intrusion $= A+B$, and the number of records that are classified as normal $= C+D$.

Based on the above notations, we have Precision=$\frac{A}{A+B}$, Recall=$\frac{A}{A+C}$, False Negative Rate=$\frac{C}{A+C}$, False Positives Rate=$\frac{B}{B+D}$. Correspondingly, for an intrusion detection system, we have four evaluation factors, which are precision, recall, false positive rate and false negative rate. A false positive occurs when the system classifies an action as an intrusion while it is a legitimate action. In table 4.1, false positive = B. A false negative occurs when an intrusion action has occurred but the system

Table 4.1: Intrusion detection performance measure

	Intrusion	Not Intrusion
Detected as Intrusion	A	B
Not Detected	C	D

considers it as a non-intrusive behavior. In table 4.1, false negative = C.

A good intrusion detection system has a high precision and a high recall, as well as a lower positive rate and a lower false negative rate. The consideration of both precision and false negative rate is very important. In practice, normal data usually significantly outnumbers the intrusion data. In this situation, only measuring precision of an intrusion detection system is misleading. A poor intrusion detection system may still have a high Precision with a high false negative rate. In our experiments, we will measure the precision and false negative rate at the same time.

4.2 Procedures for Using The Enhanced SVM Model on Intrusion Detection

4.2.1 Training Process of The Conventional SVM

Before using the enhanced SVM model, we need to know how to train the conventional SVM. The basic procedure of training the conventional SVM is as following:

- Transform data to the format of SVMlight.

  We use SVMlight as our SVM software. SVMlight requires data point to be represented as a vector of real numbers. So if there is a categorical feature in KDD data set, we have to convert them into numeric data. Here we use a simple

single number to represent a categorical feature. For example, if a feature has 5 categories, the possible value of this feature is 1-5.

The SVMlight only accepts a text file as input. In the text file each data point on one line. Each line starts with the class label of this data point followed by feature-value pairs indicating the position where this data point is located in the data space. For example the KDD data set has 41 features, so we have 41 feature-value pairs on each line.

- Conduct simple normalization on the data.

  Since our enhanced model is designed to fit different data sets, this model should be able to handle a data set from an arbitrary distribution. Because the features in KDD data set represent totally different physical meanings, the actual values of different features vary. Scale data values could avoid the features in greater numeric ranges dominate features in small numeric ranges.

  For example: consider two set of data points

  $(1, 3, 4), (1, 4, 3)$;

  and $(900, 1000, 700), (1000, 1100, 800)$.

  When we use SVM under the Euclidean metric, the squared distance between the first set of data points in the first set will be $(1-1)^2+(3-4)^2+(4-3)^2 = 1$, while it will be $30,000$ between the second set of data points. So if we do not conduct normalization, it is possible that the second set of data points will be misclassified.

  We scale data values of all features into a interval $[-1, +1]$.

- Choose a kernel function and tune the parameters of this function.

  Various SVM kernel functions are proposed for users to choose for different applications [7, 21]. Some common kernel functions are linear function, polynomial function, sigmoid function and radial basis function (RBF). Here we choose RBF as our kernel function. As we mentioned in Section 2.2.1, when SVM calculates the decision hyper plane, it uses the kernel function to transform data to represent patterns in a high dimension which is typically higher than the original feature space. Unlike other linear kernel function, RBF can handle the case when the relation between class labels and features is nonlinear [22, 33]. Besides, linear kernel is a special case of RBF and sigmoid kernel behaves like RBF for certain parameters [22, 33].

- Use cross-validation to find the best parameter C and γ.

  There are two parameters need to be found when we use RBF kernel. It is not known beforehand which C and γ are the best. The values of C and γ affects the accuracy of SVM classifier as well as the number of support vectors generated which affects the time of classification. The goal of cross-validation is to identify good C and γ values so the SVM classifier can accurately predict unknown data within a time duration as short as possible.

  One apparent advantage of using cross-validation is to prevent the overfitting problem. For example on the KDD dataset, 10% of the original data is formed as a training data set. From this training data set we generate 5 smaller training sets by randomly choosing records from the original training set. Each smaller training set has 50,000 records. Then we conduct a 5-fold cross-validation, after

we divide the training set into 5 subsets of equal size. Sequentially one subset is tested using the classifier trained on the remaining 4 subsets.

- Use the best parameter C and γ to train the whole training set.

  After choosing the best C and γ using cross-validation, we use these two parameters to train SVM and generated a SVM classifier. The SVM classifier is the final conventional one to which we will apply our enhanced model.

4.2.2 Common Procedures for Using The Enhanced SVM Model

From the introduction of intrusion detection system in Section 2.1.1, we already know the common architecture of an intrusion detection system. In this section we introduce a standard procedure for using the enhanced SVM model on intrusion detection. This procedure is also applied to all of our experiments in this study. The flowchat of the procedure is shown in Figure 4.1

The first step is the data preprocessing for SVM. SVM only process data in the feature-value format. For the sequence data such as UNM dataset [39], we need to convert the data to feature-value format. Besides the data conversion, due to the good generalization of SVM, we can also reduce the number of training samples if the training set is large, normalize data or do other data preprocessing to make the data prepared for training and test processing.

The second step is to form a sample set for rough set feature weights calculation. Rough set does not require a large number of samples to analyze the features of a dataset. As long as the sample set well represents the original data, users could use a

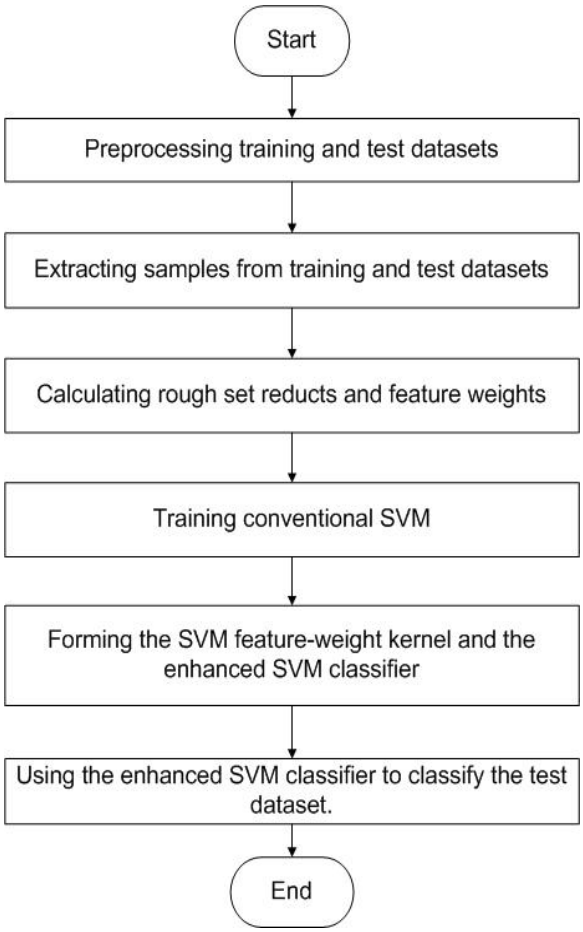

Figure 4.1: Enhanced SVM model for intrusion detection

small sample set in the enhanced model. One common method is to randomly select a small amount of samples from the training dataset.

The third step is to calculate the rough set reducts and feature weights. From Section 3.3 we know that users could use any algorithm to do this job as long as the feature weights are larger than zero. In our experiment, we use Rough Set Exploration System (RSES) [5] and the algorithm 2 as the rough set implementation to calculate the reducts from a small sample set and calculate the feature weights from the reducts. RSES includes a discretion function, so it can handle both continuous and discrete

data.

The fourth step is to train the SVM. Every kernel function used in SVM has some parameters. The training process will decide the actual value of these parameters. For example for polynomial function $(s\vec{w}(\vec{x_i} \cdot \vec{x}) + c)^d$, we need to decide the value of c; for RBF function $e^{-||\vec{w}(\vec{x_i}-\vec{x})||^2 \cdot \gamma}$ we need to decide the value of γ. These kernel functions and their parameters define the shape of the discrimination hyperplane in the data space. The enhanced SVM model does not change the value of these parameters, instead it uses the same training process as the conventional SVM to get these values. Users could use all the standard SVM training methods in this step to find the most suitable kernel function and parameter values of the kernel function. An example of the standard training methods is using cross validation to compare results of different kernel functions and using data normalization technique to process data. The details of the training process is presented in Section 4.2.1. Because the redundant features are removed from the training set in the enhanced SVM model, the training process is accelerated.

On the fifth step, after receiving the suitable conventional SVM discriminant function, we apply the feature weights to the kernel function and form an enhanced SVM discriminant function.

Finally, we use the enhanced SVM discriminant function to classify the test dataset and detect possible intrusions. Because the enhanced SVM model remove redundant features in the test dataset, the classification process could be faster than the process using the conventional SVM while having better accuracy.

4.3 Experiment on UNM Dataset

We conduct our experiment on the UNM dataset to test the performance of the enhanced SVM model on sequence based intrusion data.

4.3.1 Dataset Descriptions

The UNM dataset [39] includes a series of sequence-based data collected for the Computer Immune Systems project of the Computer Science department, University of New Mexico. The whole dataset consists of several smaller subsets generated by different UNIX system processes, which are *sendmail* dataset, *lpr* dataset, and *xlock* dataset, etc. Each subset has a number of UNIX system calls executed by an active UNIX system process. All UNIX systems have a set of system processes, such as *sendmail*, *lpr*, *xlock*, *named*, *login*, *ps*, *inetd* and *stide*. Every time a user accesses one of these UNIX system processes, the process will generate a series of UNIX system calls. These system calls form a trace. Each trace is the list of system calls issued by a single UNIX system process from the beginning of its execution to its end. Each trace is one record in a subset of the UNM dataset.

The details of UNM dataset are shown in Table 4.2:

In the experiment, we use the *lpr* subset and the *stide* subset in the UNM dataset as our experiment datasets because these two datasets have more traces than other datasets.

The data in the *lpr* dataset were collected over a period of 3 months, from one computer running SunOS 4.1.4 at the CS department at UNM. This dataset consists of 4,298 normal traces and 1,001 intrusion traces. There are totally 182 different

Table 4.2: UNM dataset details

System Process	Intrusions	Normal data	
	Number of traces	Number of traces	Number of system calls
lpr	1001	4,298	2,027,468
named	2	27	9,230,572
xlock	2	72	16,937,816
login	9	12	8,894
ps	26	24	6,144
inetd	31	3	541
stide	105	13,726	15,618,237
sendmail	-	71,760	44,500,219

system calls appearing in this dataset. Intruders use *lpr* to replace the contents of an arbitrary file with those of another. This attack exploits the fact that older versions of *lpr* use only 1000 different names for printer queue files, and they do not remove the old queue files before reusing them.

The data in the *stide* dataset were collected from a UNM computer running the Linux 2.0.35 kernel. 13,726 traces of normal data and 105 traces of intrusion data were collected. The intrusion against the *stide* process is a denial-of-service attack that affects any running program requesting memory.

4.3.2 Experiment Procedures

The procedure of our experiments are based on the standard procedures for using the enhanced SVM model shown in Figure 4.1.

Preprocessing training and test datasets

We need to convert the original data format to the feature-value format. The SVM only processes data in feature-value format, but there is no obvious feature-value pairs in the UNM dataset. Before using our enhanced SVM model to classify the data, we have to use a mapping method to convert the dataset to feature-value format. From Section 4.3.1 we know that in the UNM dataset, each trace is a sequence of system calls. The length of a trace is arbitrary. Let us say T is a trace, the number of system calls in T is L. We use a sliding window of length l to move along T from its beginning to the end, then we obtain a set of system call segments with length l, and we have L segments for trace T. We define each unique segment as a feature, the value of a feature is the count of the number of times the corresponding segment occurs in a trace. In this way, we convert a trace with length of L into a record with L feature-value pairs. During the conversion, the number of possible segments or features is huge. For example, on the *lpr* dataset, we have 182 different types of system calls. So the dimension of feature space is 182^l. In our experiment we use a window of length 4. The possible feature space is a $182^4 = 1,097,199,376$ dimensional space. Noting that the feature vectors corresponding to a trace are extremely sparse, using a conversion algorithm shown in Algorithm 3, we scan the whole *lpr* dataset and just keep the features that really occurs. After being processed, the final feature space of the *lpr* dataset has 467 features in it. The final feature space of the *stide* dataset has 273 features in it.

Algorithm 3: UNM dataset conversion

Input : UNM Dataset D_{in} in system call trace format.
Output: UNM dataset D_{out} in feature-value pair format.

```
1  l ← window length;
2  // convert a trace to a feature-value record
3  foreach trace T in D_in do
4      L ← length of T;
5      F ← empty feature set;
6      V ← empty value set;
7      for (j ← 0 to L) do
8          f ← T_j...T_{j+l};
9          if (f ∈ F) then
10             if f = F_i then
11                 V_i = V_i + 1
12             end
13         end
14         else
15             F = F ⋃ {f}
16             j ← number of element in F − 1;
17             // j starts from 0
18             V_j = V_j + 1
19         end
20     end
21     for (i ← 0 to number of element in F) do
22         // R is a record in D_out
23         R ← (F_i, V_i);
24     end
25     D_out = D_out ⋃ {R}
26 end
```

Extracting samples from training and test datasets

From Section 2.2.3 we know that SVM has good generalization ability. We could use less data for training and test. From table 4.2 we know that in the UNM dataset, *lpr* dataset has 5,299 records and *stide* has 13,831 records. We randomly select 2,000 records from each of the dataset as the sample set for training. The training process

includes feature weights calculation and the conventional SVM kernel training. We generate three test sets from each of the *lpr* and *stide* UNM datasets. Each test set has $2,000$ randomly selected records.

Calculating rough set reducts and feature weights

We use Algorithm 2 to calculate the weight of each feature and delete redundant features from datasets. After being processed, the number of features of the *lpr* dataset is narrowed down from 467 to 9, and the number of feature of stride dataset is narrowed down from 273 to 2.

Training the conventional SVM

We use the process explained in Section 4.2.1 to train the conventional SVM. We need to choose a kernel function and tune the parameters of this function before we start the training process. In this experiment we choose RBF as our kernel function. As we mentioned in Section 2.2.1, when SVM calculate the decision hyper plane, it uses a kernel function to transform data to represent patterns in a high dimension which is typically higher than the original feature space. Unlike other linear kernel functions, RBF can handle the case when the relation between class labels and features is nonlinear [22, 33]. Besides, linear kernel is a special case of BRF and sigmoid kernel behaves like RBF for certain parameters [22, 33]. Based on previous research, we choose $\gamma = 10^{-6}$ for RBF kernel $e^{-||\vec{x_i}-\vec{x}||^2 \cdot \gamma}$ [55].

Forming the SVM feature-weight kernel and SVM classifier

By applying the feature weights to the conventional SVM kernel, we build a decision function to classify the test data. During the classification process, the enhanced SVM model trims the test dataset and deletes all of the feature-value pairs, if the feature in the pairs has a weight equal to 0. The volumes of both the *lpr* and *stide* datasets are extremely reduced. The *lpr* data has been reduced 98% from its original size and the *stide* data has been reduced 99% from its original size.

4.3.3 Results and Analysis

LPR dataset

Experimental results for the UNM *lpr* datasets are presented in Table 4.3. The test runs against three test datasets. For each dataset, first we use the conventional SVM to classify it, then use the enhanced SVM to classify the same dataset.

The conventional SVM uses all features in the dataset, the total number of features is 467. The enhanced SVM uses only 9 features in the dataset. In terms of the number of features, the enhanced SVM has significant 98% improvement. The CPU-seconds used by the conventional SVM is from 1.59 to 1.62. The CPU-seconds used by the enhanced SVM is from 0.25 to 0.25. The speed of the enhanced SVM is over 80% faster than the conventional SVM. Both the conventional and the enhanced SVM do not find any normal data as intrusions. The precisions of both SVMs on all the three datasets are 100%. Both the conventional and the enhanced SVM find out all the intrusions. The false negative rate of both SVMs on all the three datasets are 0%

From Section 4.3.1 we know that the *lpr* dataset has only one type of intrusion.

The mapping method we used generates the test dataset as a sparse matrix. Among all of the 467 features in this sparse matrix, only a few features have useful information. In this scenario the feature reduction function of the enhanced SVM works very well. Because the enhanced SVM kernel uses fewer features than the conventional SVM kernel, the enhanced SVM is much faster than the conventional SVM. This speed increase is consistent with the computational complexity analysis in Section 2.2.4. The average speed increase is over 80%, and the average feature reduction improvement is over 90%. The speed increase does not achieve the same level as the feature reduction. This is because the enhanced SVM kernel function needs an extra calculation of dot product between feature weights and data points, as we presented in Equation 3.19 and Equation 3.31. The accuracy test result for *lpr* dataset is unusual. Both the conventional SVM and the enhanced SVM detect the intrusion perfectly with 100% precision and 0% false *negative*. We consider the reason is that the *lpr* dataset is simple, and has only one type of intrusion.

The STIDE Dataset

Experimental results for the UNM *stide* datasets are presented in Table 4.4. Same as the test on *lpr* datasets, this test runs against three test datasets with $2,000$ records in each of the dataset. For each dataset, we use the conventional SVM to classify it, and use the enhanced SVM to classify the same dataset.

The conventional SVM uses all features in the dataset. The total number of features is 273. The enhanced SVM uses only 2 features in the dataset. In terms of the number of features, the enhanced SVM has 99% improvement. The CPU-seconds used by the conventional SVM is from 0.21 to 0.27. The CPU-seconds used by the

Table 4.3: Comparisons of the experimental results on the UNM *lpr* dataset

	N_{record}	$N_{feature}$	CPU-sec	Precision(%)	False Neg(%)
test set 1					
Conventional SVM	2×10^3	467	1.62	100	0
Enhanced SVM	2×10^3	9	0.28	100	0
Improvement		**98%**	**83%**		
test set 2					
Conventional SVM	2×10^3	467	1.71	100	0
Enhanced SVM	2×10^3	9	0.29	100	0
Improvement		**98%**	**83%**		
test set 3					
Conventional SVM	2×10^3	467	1.59	100	0
Enhanced SVM	2×10^3	9	0.25	100	0
Improvement		**98%**	**84%**		

enhanced SVM is from 0.02 to 0.04. The speed of the enhanced SVM is over 80% faster than the conventional SVM. The precisions of the conventional SVM are from 98.25% to 98.40%. The precisions of the enhanced SVM are from 99.02% to 99.63%. The conventional SVM missed some intrusions. Its false negative rate is 5%. The enhanced SVM find out all the intrusions. Its false negative rate is 0%.

Same as on the UNM *lpr* dataset, the speed increase of the enhanced SVM is consistent to the computational complexity analysis in Section 2.2.4. Different from the test result on the UNM *lpr* dataset, the accuracy result for the conventional SVM is not consistent on the three test datasets. The false negative rate is 2.5% on the second test set, and it is changed to 10% on the third test set. Further, the precision and false negative rate are not changed in the same direction. While the precision is increased from the second to the third test set, the false negative rate is decreased from the third to the second test set. The accuracy result for the enhanced SVM

does not have the same problem as the conventional SVM. The enhanced SVM find out all the intrusions.

We believe the reason of this phenomenon is because there are some noisy features that affects the performance of the conventional SVM. These noisy features do not allow the conventional SVM discriminate some intrusion records. Because we restrict intrusion records to only count 1.5% of whole dataset, the number of intrusion records is small in the *stide* dataset. If even just a few intrusion records are not classified correctly, it affects the false negative rate significantly. Because the number of misclassified intrusion records is small, the precision is not affected too much. This explains why the precision and false negative rate are not changed consistently. The enhanced SVM removes the noisy features and classifies all the intrusion records correctly. Therefore the enhanced SVM outperforms the conventional SVM on both precision and false negative rate. Because the enhanced SVM also removes a lot of redundant features from the test data, the classification speed of the enhanced SVM is much faster than the conventional SVM.

Because the experiments on both UNM datasets use the same mapping technique, data density on both of the test datasets is sparse. The sparing datasets allow the enhanced SVM to remove many redundant features and significantly increase the speed. Furthermore, the enhanced SVM is able to remove noisy features, so its accuracy performance on the UNM *stide* dataset is better than the conventional SVM.

Table 4.4: Comparisons of the experimental results on the UNM *stide* dataset

	N_{record}	$N_{feature}$	CPU-sec	Precision(%)	False Neg(%)
test set 1					
Conventional SVM	2×10^3	273	0.21	98.40	5
Enhanced SVM	2×10^3	2	0.04	99.02	0
Improvement		**99%**	**83%**	**0.8%**	**100%**
test set 2					
Conventional SVM	2×10^3	273	0.27	98.25	2.5
Enhanced SVM	2×10^3	2	0.02	99.67	0
Improvement		**99%**	**83%**	**1.42%**	**100%**
test set 3					
Conventional SVM	2×10^3	273	0.25	98.37	10
Enhanced SVM	2×10^3	2	0.02	99.63	0
Improvement		**99%**	**84%**	**1.25%**	**100%**

4.4 Experiment on KDD Dataset

In this experiment, we use the KDD (Data Mining and Knowledge Discovery) Cup 1999 data [10]. The KDD data is relatively more complex, and it has more data compared to the UNM dataset. We want to study the performance of the enhanced SVM on a more sophisticated dateset by conducting experiment on the KDD dataset.

4.4.1 Dataset Descriptions

The KDD dataset contains a wide variety of intrusions simulated in a military network environment. The data of this dataset are collected raw network packets and are independent to an operating system or an application. Each record in this dataset has been attached a label identifying to which class the record belongs to. All labels are assumed to be correct. The KDD dataset is more complex than the

UNM dataset. Instead of having only one type of intrusion, the KDD dataset has four intrusion categories and one normal category. Twenty-four types of attacks fall into four main categories which are denial-of-service (DOS), e.g., SYN flood, unauthorized access from a remote machine (R2L), e.g., guessing password, unauthorized access to local superuser (root) privileges (U2R), e.g., various "buffer overflow" attacks, and surveillance and other probing, e.g., port scanning. It has 41 features as shown in Table 4.5. In the training dataset, each record has a class label which identifies whether this record is an intrusion data record or not. The KDD dataset contains more data than the UNM dataset. The volume of the dataset is 744 MB including 4,940,000 connection records. 10% of the original data is formed as training dataset with a label identifying whether the record is an intrusion and to which intrusion types the record belongs. In this paper, we only discuss binary classification. Therefore, instead of considering four intrusion categories and one normal category, we treat all intrusion records as one category and all normal records in another category.

Compared with the UNM dataset, the KDD dataset has different characters. At first, the volume of the KDD dataset is much higher than the UNM dataset. This requires us to use more training sets while trying to keep each training set still small.

Second, the each feature of the KDD dataset has a real-life meaning as shown in Table 4.5. Because most of the features are independent of each other, although the number of features is smaller than UNM dataset, the dataset is highly non-linear. It is harder to find redundant features and remove them from the feature space. This could affect the accuracy of intrusion detection. We have to make more effort to find the appropriate SVM kernel and parameters.

Third, the KDD dataset is more complicated. The types of attacks is 24, which is

Table 4.5: Feature list of KDD dataset
Type C=continuous D=discrete

#	Feature Name	Description	Type
1	Duration	Length (number of seconds) of the connection.	C
2	Protocol	Type of the protocol, e.g. tcp, udp.	D
3	Service	Network service on the destination, e.g. http, telnet, ftp.	D
4	Flag	Normal or error status of the connection.	D
5	src_bytes	number of data bytes from source to destination.	C
6	dst_bytes	number of data bytes from destination to source.	C
7	Land	1 if connection is from/to the same host/port; 0 otherwise	D
8	wrong_fragment	number of "wrong" fragments.	C
9	Urgent	number of urgent packets.	C
10	Hot	number of "hot" indicators.	C
11	num_failed_logins	number of failed login attempts.	C
12	logged in	1 if successfully logged in; 0 otherwise.	D
13	num_compromised	number of compromised conditions.	C
14	root_shell	1 if root shell is obtained; 0 otherwise.	D
15	su_attempted	1 if "su root" command attempted; 0 otherwise.	D
16	num_root	number of "root" accesses.	C
17	num_file_creations	number of file creation operations.	C
18	num_shells	number of shell prompts.	C
19	num_access_files	number of operations on access control files.	C
20	num_outbound_cmds	number of outbound commands in an ftp session.	C
21	is_host_login	1 if the login belongs to the "hot" list; 0 otherwise.	D
22	is_guest_login	1 if the login is a "guest" login; 0 otherwise.	D
23	Count	number of connections to the same host as the current one during past two seconds.	C
24	srv_count	number of connections to the same service as the current connection in the past two seconds.	C
25	serror_rate	% of connections that have "SYN" errors.	C
26	srv_serror_rate	% of connections that have "SYN" errors.	C
27	rerror_rate	% of connections that have "REJ" errors.	C
28	srv_rerror_rate	% of connections that have "REJ" errors.	C
29	same_srv_rate	% of connections to the same service.	C
30	diff_srv_rate	% of connections to different services.	C
31	srv_diff_host_rate	% of connections to different hosts.	C
32	dst_host_count		C
33	dst_host_srv_count		C
34	dst_host_same_srv_rate		C
35	dst_host_diff_srv_rate		C
36	dst_host_same_src_port_rate		C
37	dst_host_srv_diff_host_rate		C
38	dst_host_serror_rate		C
39	dst_host_srv_serror_rate		C
40	dst_host_rerror_rate		C
41	dst_host_srv_rerror_rate		C

much more than the UNM dataset which has only one type of attack on each subset. Besides, the dataset has both continuous and discrete data. These complications increases the difficulty of intrusion detection.

Lastly, the KDD dataset is in the feature-value pair format. So we do not need to convert the data as we did for the UNM dataset.

4.4.2 Experiment Procedures

The experiment procedure on the KDD dataset is almost the same as the procedure on UNM dataset. The preprocessing step on the KDD dataset does not need to convert the data format because the dataset is already in feature-value format. The sample extracting step selects more data than the samples from the UNM dataset. Because the data volume in the KDD dataset is much larger, the test dataset needs to be larger as well, so we generate three test sets with 5×10^4 records in each set for testing. The feature weight calculating step is the same as the steps on the UNM dataset. The conventional SVM training step is more complex than the step on the UNM dataset because the KDD dataset is much more complex. The final test step is the same as the step on UNM dataset.

4.4.3 Results and Analysis

In Section 4.2.1 we explained that the RBF kernel can handle the case when the relation between class labels and features is nonlinear [22, 33]. In the experiment on the KDD dataset, we choose RBF kernel function as the kernel for the conventional SVM. γ is the only parameter we need to train for RBF kernel function.

Table 4.6: Training results of conventional SVM with different values of gamma

Training Result	Exp1	Exp2	Exp3
Number of training records	5×10^4	5×10^4	5×10^4
Number of features	41	41	41
kernel	RBF	RBF	RBF
Value of γ	10^{-3}	10^{-6}	10^{-8}
Number of generated SVs	6,498	1,868	1,057

Training result of the conventional SVM on KDD99 Dataset

Because the KDD99 dataset is more complex than the UNM dataset, training the conventional SVM is harder. Here we present the training result of the conventional SVM. Table 4.6 shows the training results from the conventional SVM using different values of γ. All the three experiments use the same training set which consists of 5×10^4 randomly chosen data records from the original training set of $494,020$ records. The table shows that the different sets of support vectors generated by using different values of γ. When $\gamma = 10^{-3}$, the conventional SVM generates $6,498$ support vectors. When $\gamma = 10^{-6}$, the conventional SVM generates over $1,868$ support vectors. When $\gamma = 10^{-8}$, the conventional SVM generates over $1,057$ support vectors. It is observed that the larger value of γ results in a larger number of support vectors generated and the smaller value of γ results in a small number of support vectors.

Table 4.7 shows three different test results. All the tests use the same test dataset with $10,000$ records. Each test is applied to a trained conventional SVM kernel shown in Table 4.6. Experiment 1 uses a RBF kernel with $\gamma = 10^{-3}$. The precision

of experiment 1 is 99.56% which is the highest among the three tests. The number of false negatives is 7 which is the lowest among the three tests . The CPU-seconds of experiment 1 is 49.53 which is the longest among three tests. Experiment 2 uses a RBF kernel with $\gamma = 10^{-6}$. The precision of experiment 2 is 99.37% which is in the middle among the three tests. The number of false negatives is 11 which is also in the middle among the three tests. The CPU-seconds of experiment 2 is 11.34 which is in the middle among the three tests. Experiment 3 uses RBF kernel with $\gamma = 10^{-8}$. The precision of experiment 2 is 97.89% which is the lowest among the three tests. The number of false negatives is 35 which is highest among the three tests. The CPU-seconds of experiment 3 is 8.32 which is the shortest among three tests.

From these results we conclude that a larger number of rules results in higher detection accuracy and higher computation costs. In experiment 1, the accuracy is the highest. Experiments 2 and 3 have similar precision, false negative rate and test time. This result is quite reasonable. From the form of SVM RBF classifier and analysis in Section 2.2.4 we know the computational complexity of test process is $O(mn)$ with m is the number of rules, n is the number of features in the data space. Therefore, the test time is increased along with the increasing of the number of support vectors. Another phenomenon we noticed is that experiment 1 has a much larger number of support vectors and much longer test time than experiment 2 and 3. The accuracy of these three experiments are on the same level. From this result, we choose $\gamma = 10^{-6}$ as the final parameters for RBF SVM decision function.

Table 4.7: Test results of conventional SVM with different values of gamma

Test Result	Exp1	Exp2	Exp3
Number of test records	10,000	10,000	10,000
Number of features	41	41	41
Value of *gamma*	10^{-3}	10^{-6}	10^{-8}
Number of generated SVs	6,498	1,868	1,057
Number of misclassifications	44	63	211
Precision	99.56%	99.37%	97.89%
Number of False Negative	7	11	35
CPU-second	49.53	11.34	8.32

Result comparison of the conventional and enhanced SVM on KDD99 Dataset

Table 4.8 shows the results comparison between the conventional SVM and the enhanced SVM for the KDD dataset. The conventional SVM uses all 41 features in the dataset. The enhanced SVM uses only 13 features in the dataset. In terms of the number of features, the enhanced SVM has 68% improvement. The CPU-seconds used by the conventional SVM is from 222 to 230. The CPU-seconds used by the enhanced SVM is from 75 to 78. The speed of the enhanced SVM is 65% to 66% faster than the conventional SVM. The precisions of the conventional SVM are from 99.82% to 99.88%. The precisions of the enhanced SVM are from 99.86% to 99.91%. The false negative rate of the conventional SVM is from 7.45% to 7.69%. The false negative rate of the enhanced SVM is from 5.49% to 6.91%.

The performance of both the conventional SVM and the enhanced SVM are consistent on the three experiments. Both SVMs have the best accuracy and the longest running time on test set 3, and the worst accuracy and the shortest running time on

Table 4.8: Comparisons of the experimental results on the KDD dataset

	N_{record}	$N_{feature}$	CPU-sec	Precision(%)	False Neg(%)
test set 1					
Conventional SVM	5×10^4	41	222.28	99.82	7.69
Enhanced SVM	5×10^4	13	75.63	99.86	6.39
Improvement		**68.0%**	**66.0%**	**0.4%**	**16.9%**
test set 2					
Conventional SVM	5×10^4	41	227.03	99.80	8.25
Enhanced SVM	5×10^4	13	78.93	99.85	6.91
Improvement		**68.0%**	**65.0%**	**0.5%**	**16.2%**
test set 3					
Conventional SVM	5×10^4	41	230.27	99.88	7.45
Enhanced SVM	5×10^4	13	77.85	99.91	5.49
Improvement		**68.0%**	**66.0%**	**0.3%**	**26.3%**

test set 2. The improvements of performance are consistent on all of the three test sets. On all of the three test sets the enhanced SVM model outperforms the conventional SVM in all three measures, namely, precision, false negative rate and CPU time for the KDD dataset. This suggests that the enhanced SVM model has a good generalization ability. The speed improvement of the enhanced SVM model is consistent to the the reducing of the number of features. The improvement for precision is 0.4% on average, which is moderate. We consider there is not significant precision improvement because the precision of the conventional SVM is already very high. It is hard for the enhanced SVM to improve the result in terms of the precision.The improvement of the enhanced SVM on false negative rate is significant, which means the feature weights helps the SVM classifier find out more intrusions.

4.5 Summary

We performed our experiments on two datasets with different characteristics. The UNM dataset is an example of low data density, simple intrusion datasets. The KDD dataset has a high data density, and includes multiple intrusion types, which makes this dataset more complex than the UNM dataset. The performance of the enhanced SVM is improved on both datasets, although the improvements emphasize on different aspects. From the analysis of the experiment results and comparison of the results on different datasets, we have the following findings:

- Compared to the conventional SVM, the enhanced SVM can significantly increase the speed on UNM dataset, which is a low data density and simple intrusion dataset.

  We use a mapping method to convert the UNM dataset to feature-value format. The UNM dataset in feature-value format has low data density. From the experiment result we can see that, for the enhanced SVM, intrusions in the UNM dataset can be classified from relatively less features compared to the KDD dataset. This causes the enhanced SVM to remove a lot of redundant features. The results show that the speed improvement of the enhanced SVM is caused by its feature reduction ability. A significant feature reduction makes the enhanced SVM much faster than the conventional SVM.

  Each subset of UNM dataset is generated by a single type of intrusion, so the UNM dataset is relatively simple. The conventional SVM can classify the dataset very well, so the enhanced SVM does not have a significant improvement in terms of accuracy.

- Compared to the conventional SVM, the enhanced SVM can increase the accuracy on the KDD dataset, which is a high data density, complex intrusion dataset.

  The KDD dataset consists of 41 features. Each feature has a real-life meaning. The enhanced SVM does not have significant feature reduction on this dataset, so the speed increase is moderate. But the enhanced SVM significantly increased the intrusion detection accuracy on both precision and false negative rate. We believe that it is because the enhanced SVM includes the feature weights information.

Chapter 5

Conclusion and Perspective

We propose an enhanced SVM model for intrusion detection. To develop this enhanced model, at first we used rough sets technique to calculate the reducts and ranks of the features of intrusion detection data, then we developed an algorithm to calculate feature weights based on the reducts and form a new kernel functions enhanced with feature weights. The experiment results proved that the proposed new model is effective on both the UNM dataset and the KDD dataset. On the KDD dataset, although the precision levels of both the conventional SVM and the new SVM model are about the same, the false negative rates of the new model are lower than the conventional SVM model. In addition, the time used to detect an intrusion of the new model is much less than the conventional SVM. On the UNM dataset, both the conventional SVM and the new model performed perfectly in terms of accuracy. However, the new SVM model still has some advantages, i.e., the running time is much less as fewer number of features are used for classification.

5.1 Summary of Contributions

In this thesis, we use SVM to find the normal patterns of computer system activities and detect intrusions. We try to improve the conventional SVM on intrusion detection in the aspects of feature discrimination and feature reduction using rough set. The major contributions are following,

- Propose a SVM model enhanced with a new feature-weight kernel. The feature-weight kernel enhanced SVM in both the feature discrimination and feature reduction. By applying rough sets theory to the training process of SVM and adding feature weights to the SVM kernel function, the enhanced SVM model has the capability to quantitatively evaluate the importance of each feature of intrusion detection datasets, distinguish important features from unimportant features and remove redundant features. A new algorithm is proposed and implemented to calculate the weights in the feature-weight kernel.

- Implement the enhanced SVM model and examine the model by doing experiment on a serial of well-known intrusion detection datasets.

5.2 Future Research

The current research focuses on using the enhanced SVM model on intrusion detection application. The SVM model itself is a general technique used on a lot of different application areas. The feature weights enhanced kernel is not related to any particular application, so we could explore the usage of the enhanced SVM on other application areas.

In addition, research is also possible regarding the different ways of applying feature weights to different SVM kernels and the different ways to calculate the feature weights.

Finally, the mathematical proof of the reason that enhanced SVM mode has good performance also needs future research.

References

[1] S. Amari and S. Wu. Improving support vector machine classifiers by modifying kernel functions. *Neural Networks*, 12(6):783–789, 1999.

[2] D. Anderson, T. Lunt, H. Javits, A. Tamaru, and A. Valdes. Detecting unusual program behavior using the statistical components of NIDES. Technical report, Computer Science Laboratory, SRI International, 1995.

[3] R. G. Bace. *Intrusion Detection*. Macmillan Technical Publishing, 2000.

[4] J. G. Bazan. Dynamic reducts and statistical inference. In *Proc. of International Conference of Information Processing and Management of Uncertainty*, pages 1147–1152, 1996.

[5] J.G. Bazan, M.S. Szczuka, and J. Wroblewski. A new version of rough set exploration system. In *Proc. of the 3rd International Conference on Rough Sets and Current Trends in Computing*, pages 397–404, 2002. Software available at http://logic.mimuw.edu.pl/∼rses/.

[6] A. Ben-Hur and I. Guyon. Detecting stable clusters using principal component analysis. *Journal of Machine Learning Research*, 3:1183–1208, 2003.

[7] C. Burge. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2:121–167, 1998.

[8] C. Chang and C. Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at http://www.csie.ntu.edu.tw/∼cjlin/libsvm.

[9] R. Chen, J. Chen, T. Chen, and C. Heieh. Building an intrusion detection system based on support vector machine and genetic algorithm. In *Proc. of Advances in Neural Networks*, pages 1055–1077, 2005.

[10] KDD cup 1999. http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.

[11] H. Deng, Q. Zeng, and D.P. Agrawal. Svm-based intrusion detection system for wireless ad hoc networks. In *Proc. of Vehicular Technology Conference*, pages 2147–2151, 2003.

[12] L. Ertoz, E. Eilertson, A. Lazarevic, P. Tan, J. Srivastava, V. Kumar, and P. Dokas. *MINDS - Minnesota Intrusion Detection System*. MIT Press, 2004.

[13] G. Forman. An extensive empirical study of feature selection metrics for text classification. *Journal of Machine Learning Research*, 3:1289–1306, 2003.

[14] W.B. Frakes, R. Baeza-Yates, and B.Y. Ricardo. *Information Retrieval: Data Structures and Algorithms*. Prentice-Hall, 1992.

[15] I. Guyon, J. Westonl, S. Barnhill, and V. Vapnik. Gene selection for cancer classification using support vector machines. *Machine Learning*, 46:389–422, 2002.

[16] J.C. Han, X. Hu, and T.Y. Lin. Feature subset selection based on relative dependency between attributes. *Lecture Notes in Computer Science*, 3066:176–185, 2004.

[17] K. Hu, Y. Lu, and C. Shi. Feature ranking in rough sets. *AI Communications*, 16:41–50, 2003.

[18] S. Hua and Z. Sun. Support vector machine approach for protein subcellular localization prediction. *Bioinformatics*, 17:721–728, 2001.

[19] K. Ilgun, R.A. Kemmerer, and P.A. Porras. State transition analysis: A rule-based intrusion detection approach. *IEEE Transaction on Software Engineering*, 7:181–199, 1995.

[20] L. Lukas J. Suykens, J. De Brabanter and J. Vandewalle. Weighted least squares support vector machines: robustness and sparse approximation. *Nerocomputing*, 48:85–105, 2002.

[21] T. Joachims. *Making large-Scale SVM Learning Practical. Advances in Kernel Methods - Support Vector Learning.* MIT-Press, 1992.

[22] S. Keerthi and C. J. Lin. Asymptotic behavior of support vector machine with gaussian kernel. *Neural Computation*, 15(7):1667–1689, 2003.

[23] K. Kira and L.A. Rendell. The feature selection problem:traditional methods and new algorithm. In *Proc. of the 9th National Conference on Artificial Intelligence*, pages 129–134, 1992.

[24] R. Kohavi and G.H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1-2):273–324, 1997.

[25] J.Z. Kolter and M.A. Maloof. Learning to detect and classify malicious executables in the wild. *Journal of Machine Learning Research*, 7:2721–2744, 2006.

[26] J. Komorowski, L. Polkowski, and Skowron. *Rough sets: a tutorial*. Springer-Verlag, 1998.

[27] S. Kumar and E. Spafford. A software architecture to support misuse intrusion detection. In *Proc. of the 18th National Information Security Conference*, pages 194–204, 1995.

[28] P. Laskov, C. Gehl, S. Kruger, and K.R. Muller. Incremental support vector learning: analysis, implementation and applications. *Journal of Machine Learning Research*, 7:1909–1936, 2006.

[29] A. Lazarevic, L. Ertoz, A. Ozgur, J. Srivastava, and V. Kumar. A comparative study of anomaly detection schemes in network intrusion detection. In *Proc. of the 3rd SIAM Conference on Data Mining*, 2003.

[30] W. Lee and S.J. Stolfo. A framework for constructing features and models for intrusion detection systems. *Information System Security*, 3(4):227–261, 2000.

[31] W. Lee, S.J. Stolfo, and W. Kui. Adaptive intrusion detection: A data mining approach. *Artificial Intellegence Review*, 14(6):533–567, 2000.

[32] C. Leslie and R. Kuang. Fast string kernels using inexact matching for protein sequences. *Journal of Machine Learning Research*, 5:1435–1455, 2004.

[33] H. T. Lin and C. J. Lin. A study on sigmoid kernels for svm and the training of non-psd kernels by smo-type methods. Technical report, National Taiwan University, 2003.

[34] J. Luo and S. Bridges. Mining fuzzy association rules and fuzzy frequency episodes for intrusion detection. *International Journal of Intelligent Systems*, 15(8):687–703, 2000.

[35] J. McHugh, A.M. Christie, and J. Allen. Defending yourself: The role of intrusion detection systems. *IEEE Software*, 17(5):42–51, 2003.

[36] M. Mohajerani, A. Moeini, and M. Kianie. A neuro-fuzzy intrusion detection system. In *Proc. of the 10th IEEE International Conference on Electronics, Circuits and Systems*, pages 348–351, 2003.

[37] S. Mukkamala and A.H. Sung. Performance based feature identification for intrusion detection using support vector machines. In *Proc. of the 2nd International Conference on Hybrid Intelligent Systems*, pages 351–364, 2002.

[38] S. Mukkamala, A.H. Sung, and A. Abraham. Intrusion detection using ensemble of soft computing paradigms. In *Proc. of the 3rd International Conference on Intelligent Systems Design and Applications*, pages 239–248, 2003.

[39] University of New Mexico. Sequence-based Intrusion Detection: http://www.cs.unm.edu/∼immsec/systemcalls.htm.

[40] E. Osuna, R. Freund, and F. Girosit. Training support vector machines: an application to face detection. In *Proc. of Computer Vision and Pattern Recognition*, pages 130–136, 1997.

[41] Z. Pawlak, J. Grzymala-Busse, R. Slowinski, and W. Ziarko. Rough set. *IEEE Network*, 38(11):89–95, 1995.

[42] Y. Qiao, X.W. Xin, Y. Bin, and S. Ge. Anomaly intrusion detection method based on HMM. *Electronics Letters*, 38(13):663–664, 2002.

[43] I. Rueda, F.A. Arciniegas, and M.J. Embrechts. Svm sensitivity analysis: An application to currency crises aftermaths. *IEEE Transactions on systems and cybernetics part A - systems and humans*, 34(3):387–398, 2004.

[44] M. Sebring, E. Shellhouse, M. Hanna, and R. Whitehurst. Expert systems in intrusion detection: A case study. In *Proc. of the 11th International Conference of Computer Security*, pages 17–20, 1998.

[45] Internet World Stats. http://www.internetworldstats.com/stats.htm.

[46] A.H. Sung and S. Mukkamala. Identifying important features for intrusion detection using support vector machines and neural networks. In *Proc. of Applications and Internet*, pages 209–216, 2003.

[47] Y. Tan and Jun. Wang. A support vector machine with a hybrid kernel and minimal vc dimension. *IEEE Transaction on Knowledge and Data Engineering*, 4(16):385–395, 2004.

[48] S.F. Tian, S. Mu, and C. Yin. Sequence-similarity kernels for svms to detect anomalies in system calls. *Nerocomputing*, 70(4-6):859–866, 2007.

[49] S. Tong and D. Koller. Support vector machine active learning with applications to text classification. *Journal of Machine Learning Research*, 2:45–46, 2002.

[50] I.W. Tsang, J.T. Kwok, and P.M. Cheung. Core vector machines: Fast svm training on very large data sets. *Journal of Machine Learning Research*, 6:363–392, 2005.

[51] V.N. Vapnik. *The Nature of Statistical Learning Theory*. Springer, 1995.

[52] W.D. Wang and S. Bridges. Genetic algorithm optimization of membership functions for mining fuzzy association rules. In *Proc. of the 7th International Conference on Fuzzy Theory & Technology*, pages 131–134, 2000.

[53] J. Wroblewski. Genetic algorithms in decomposition and classification problems. *Rough Sets in Knowledge Discovery*, 38(2):471–487, 1999.

[54] J.T. Yao, S.L. Zhao, and L. Fan. An enhanced support vector machine model for intrusion detection. In *Proc. of International Conference on Rough Sets and Knowledge Technology*, pages 538–543, 2006.

[55] J.T. Yao, S.L. Zhao, and L.V. Saxton. A study on fuzzy intrusion detection. In *Proc. of Data Mining, Intrusion Detection, Information Assurance, and Data Networks Security*, pages 23–30, 2005.

[56] Y.Y. Yao, S.K.M. Wong, and T.Y. Lin. *Rough Sets and Data Mining: Analysis for Imprecise Data*. Kluwer Academic Publishers, 1997.

[57] L. Zanni, T. Serafini, and G. Zanghirati. Parallel software for training large scale support vector machines on multiprocessor systems. *Journal of Machine Learning Research*, 7:1467–1492, 2006.