A NEW DISTRIBUTED FRAMEWORK FOR CYBER ATTACK DETECTION AND CLASSIFICATION

By

SANDEEP GUTTA

Bachelor of Engineering in Electronics and
Communication Engineering
Andhra University
Visakhapatnam, Andhra Pradesh, India
2008

Submitted to the Faculty of the
Graduate College of
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
December, 2011

COPYRIGHT ©

By

SANDEEP GUTTA

December, 2011

A NEW DISTRIBUTED FRAMEWORK FOR CYBER ATTACK DETECTION AND CLASSIFICATION

Thesis Approved:

Dr. Qi Cheng

Thesis Advisor

Dr. Martin T. Hagan

Dr. Weihua Sheng

Dr. Sheryl A. Tucker

Dean of the Graduate College

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

This work has been supported in part by the Center for Telecommunications and Network Security (CTANS).

CHAPTER 1

INTRODUCTION

With the enormous growth of cyber activity and Internet penetration, the number of cyber attacks has increased significantly over the last decade. Detection of these attacks has always been a major concern for many governments and organizations all over the world. Some of the recent well-known cyber attacks include Nimda attack, SQL Slammer attack, July 2009 attacks, and Operation Aurora. The Operation Aurora cyber attacks were launched against major organizations like Google, Yahoo, Adobe Systems, Morgan Stanley, Dow Chemical Company, etc. in the second half of 2009. Recently in April 2011, a series of cyber attacks were launched against Sony's PlayStation Network which made the network go offline for about 24 days. In May 2011, cyber attacks were launched against Citibank and the account information of about 1% of its 21 million North American credit card customers was stolen. Such cyber attacks incur great expenses and trouble to several organizations. In April 2009, the Pentagon announced that more than $100 million were spent on repairing the damage from the cyber attacks over the past six months. In May 2011, Sony Corporation declared that the cost of its PlayStation Network attack was about $171 million.

Moreover, the computing systems and networks in the critical sectors like military, banking and finance, telecommunications, transportation, medical etc. are vulnerable to these growing cyber threats. Military and financial organizations are always the top target for attackers. The protection of the sensitive data from cyber attacks in the critical sectors like these, is now the topmost priority for government and other

organizations. Nowadays, politically motivated cyber attacks are also increasing in number. The Economist describes the cyberwarfare as the fifth domain of warfare after land, sea, air, and space. In May 2010, a new agency called the United States Cyber Command (USCYBERCOM) was established by the U.S. government to exclusively protect and defend its military networks. Similar agencies were set up by other countries as well to protect their digital infrastructure.

1.1 Taxonomy of Cyber Attacks

A *cyber attack* (or *intrusion*) can be defined as a series of malicious computer activities that threaten and compromise the security and integrity of a computer/network system. The cyber attacks disrupt the normal operation of a computer system, and may illegally access or destroy the information in the computer systems. Most of the time, a cyber attack is launched through the data stream on the computer networks. The classification of cyber attacks is helpful for learning the behavior of different attacks, which may be used in the design of cyber attack detection systems. In general, cyber attacks can be broadly classified into the following four categories:

1. **Denial of Service (DoS) Attacks**: The Denial of Service (DoS) attacks are those that make a computer resource (e.g., a Web server) unavailable to the actual legitimate users. The most common form of the DoS attack involves making the computer resource too busy and fully loaded with lots of unnecessary requests, so that the actual users cannot use it. There are many variants of DoS attacks including TCP-SYN Flood, ICMP/UDP Flood, Smurf, Ping of Death, Teardrop, Mailbomb, Apache2.

   Denial of Service (DoS) attacks are the most common cyber attacks. The most popular variant of the DoS attack is the Distributed Denial of Service (DDoS) attack. As the name itself suggests, DDoS attacks are launched in a distributed

fashion. Most often in the case of DoS attacks, the victim's computing resource is more powerful than the attacker's computing resource. In such cases, the DoS attacker launches the attack using several multiple intermediate hosts (generally called bots).

2. **Remote to Local (R2L) Attacks**: In this type of attacks, an attacker tries to illegally gain local access to a computer system, by sending some packets to the system over a network. Some common ways in which this is accomplished is guessing passwords through the brute-force dictionary method, FTP Write, etc.

3. **User to Root (U2R) Attacks**: In this class of attacks, an attacker with a normal user privileges illegally tries to gain the root access (administrator privileges) to a computer system. One common way in which this is done is by using the buffer overflow methods.

4. **Probing Attacks**: In this type of attacks, an attacker scans a network/computer to find possible vulnerabilities through which the attacker can exploit the system. This is like some sort of surveillance on the system. One common way in which this is done is through port scanning. By scanning the different ports of a computer system, the attacker can get the information about the open ports, services running, what the hosts in a network are up to, and various other sensitive details like the IP address, MAC address, firewall rules, etc. Some examples of probing attacks are IPSweep, NMap, MScan, Satan, SAINT.

There are other attacks such as the Man-in-the-middle (MITM) attacks, Social Engineering attacks, etc. However, these attacks can be prevented by following some personal security measures. For example, MITM attack is a sort of active eavesdropping in which the attacker hears the conversation between the victims. This is generally done by establishing separate and independent connections with each

victim, and relaying messages between them without them even knowing it. This MITM attack can be prevented by using strong encryption schemes like the Secure Shell (SSH) protocol, which prevents the attacker from reading the messages.

1.2 Motivation

Cyber attacks have become a major threat nowadays. The cost of damage from a cyber attack is very huge. Protecting the computer networks from these cyber attacks has become the topmost priority. Thus, the problem of cyber attack detection is of great importance. There is an urgent need for an efficient cyber attack detection system, which can accurately detect the cyber attacks in time so that proper countering actions can be taken to protect the vital cyber infrastructure. Most of the current cyber attack detection systems suffer from two main issues:

1. High computational complexity
2. Low detection accuracy

There is generally a trade-off between these two. In general, the cyber attack detection system with high detection accuracy suffers from high computational complexity. One of the main challenges in designing a cyber attack detection system is to decrease the overall computational complexity of the system without any decrease in its detection accuracy. The focus of this thesis is to address this challenge. In this thesis, a new cyber attack detection system is proposed which has relatively less computational complexity and high detection accuracy.

1.3 Thesis Organization

In this chapter, different types of cyber attacks were introduced. The behavior of different attacks is presented in detail.

In Chapter 2, different cyber attack detection systems are introduced. The relative merits and demerits of those cyber attack detection systems are discussed. Then, the proposed distributed cyber attack detection system is described in detail.

In Chapter 3, the support vector machines which are used as the binary classifiers in the proposed cyber attack detection system are presented. The support vector machines are generally the best supervised classifiers, and yield the best classification performance. The support vector machines are discussed in detail in this chapter.

In Chapter 4, several fast training approaches to train the support vector machine are discussed. Though the support vector machines are the best classifiers, their training procedure has high computational complexity. This often limits the use of support vector machines in the real-world applications involving huge data (for e.g., cyber attack detection). In this chapter, several existing fast training approaches for support vector machines are presented. Finally, a new fast and efficient training approach for support vector machines is proposed which reduces the support vector machine training complexity without having significant degradation in the classification performance of the support vector machine.

In Chapter 5, the fusion rules used in the proposed distributed cyber attack detection system are discussed. Effective fusion rules are proposed using the Dempster-Shafer theory of evidence.

In Chapter 6, the experimental results are provided. Conclusions are provided in Chapter 7.

CHAPTER 2

CYBER ATTACK DETECTION SYSTEMS

The *cyber attack detection system*, also referred to as the *intrusion detection system (IDS)*, continuously monitors the computer/network system trying to identify the cyber attacks while they are going on a computer/network system. Once an attack is detected, the cyber attack detection system alerts the corresponding security professional who then takes a necessary action. The design of the cyber attack detection system is generally based on the basic assumption that the cyber attack activities are different from the normal activities and hence can be detected [1].

Generally the cyber attack detection systems or intrusion detection systems (IDSs) are classified into two main categories:

1. Signature-based intrusion detection systems

2. Anomaly-based intrusion detection systems

The signature-based intrusion detection system is based on the prior knowledge of known attack signatures. In this approach, an activity is classified as an attack if it matches with an already known attack signature. Therefore the performance of this approach is greatly limited by the signature database available. This approach cannot detect novel attacks, which are quite different from the known attacks. Moreover, defining signatures for all the known attacks is in general difficult. There is a great need for expert knowledge to create the attack signatures. Any error in the attack signature definitions may lead to a large number of missed detections. The common techniques adopted include pattern matching, and rule-based techniques. Some best

examples of signature-based intrusion detection systems are Snort [2], Bro [3], and Prelude [4].

In the anomaly-based intrusion detection system, an assumption that a cyber attack will always show some deviations from normal patterns is made. First, a normal profile of the system is developed. Then, all the activities that do not match with this normal profile are considered to be attacks. The major drawback in this approach is the difficulty in accurately modeling the normal behavior of the system, which can be highly dynamic. Furthermore, the assumption that the abnormal behavior is only due to attacks is another limitation. To summarize, though the anomaly-based intrusion detection system may sometimes detect novel attacks, the biggest drawback is the lack of accurate profiling of the normal behaviors of a system. This is in general difficult, and this may lead to a very large number of false alarms.

If we observe the conventional intrusion detection approaches mentioned above, it is clear that both of them completely rely on explicit pattern matching techniques. That is, all the activities matching with the known attack signatures are considered to be attacks in the signature-based detection approach, and all the activities not matching with the system's normal profile are considered to be attacks in the anomaly-based detection approach. It is clear that they perform good when the attacks are somewhat similar and regular. But the attacks in general are not regular. Novel attacks are being developed day by day. And the conventional detection approaches cannot adapt to the new attacks that are being developed. Therefore, there is a need for an adaptive intrusion detection approach. The above mentioned difficulties lead researchers to apply the pattern classification techniques to the problem of intrusion detection, where the normal and attack models with appropriate decision rules are automatically learned by the system. Hence, these systems are adaptive to new unforeseen attacks. The main advantage of using these pattern classification techniques is their ability to generalize, learn and adapt.

The intrusion detection systems (IDSs) can also be classified into the following categories based on the type of data they collect:

1. **Host-based intrusion detection systems**: These IDSs monitor the individual host systems in a network. They often collect a number of system level details like system calls, application logs, incoming and outgoing network events, system file changes, etc. An example of a host-based intrusion detection system is OSSEC [5].

2. **Network-based intrusion detection systems**: A network-based IDS monitors the whole network for signs of an ongoing attack. It examines the overall network traffic and observes different hosts as well. These network-based IDSs access the network traffic through a network tap, or by connecting to a network switch configured for port mirroring. An example of a network-based IDS is the well known Snort [2]. Another open-source network-based IDS is the Bro [3]. An example of commercial network-based IDS is the McAfee Network Security Platform [6].

3. **Application-based intrusion detection systems**: These IDSs monitor only a specific application by collecting the corresponding necessary data. They use external sensors that capture the data exchanged between the monitored application and other third party entities with which the application interacts.

4. **Hybrid intrusion detection systems**: These IDSs are a combination of two or more of the above described IDSs.

2.1 Proposed Cyber Attack Detection System

In this thesis, a new network-based cyber attack detection system (intrusion detection system) is proposed. The proposed cyber attack detection system is designed in a

distributed fashion, using multiple sources of information (sensors). The sensing technology has greatly improved in recent times, and a wide variety of advanced sensors are now available which can collect a lot of information from the network. Each sensor is a source of information which independently operates and collects different types of data from the network. Using a variety of multiple sensors, we can have a complete view of the network and hence the cyber attacks can be detected more accurately.

The data collected from all the sensors in the system can be processed in two ways: centralized approach and decentralized (or distributed) approach. In the centralized processing approach, each sensor transmits its entire data to a central unit. The central unit receives the data from all the sensors and processes it to generate a final decision. This traditional centralized processing approach has high computational complexity, requires huge bandwidth, and is practically inefficient. On the other hand, the distributed processing approach is more efficient and has relatively less computational complexity. In this distributed approach, each sensor first process its data and generates a local decision. All the local decisions from all the sensors are then transmitted to the central unit (generally referred to as the fusion center), which then generates the final decision based on all the available local decisions. Clearly, this distributed approach is computationally more efficient and requires very less bandwidth. Moreover with the recent advancements in the field of sensing, currently available sensors all have computing capabilities. Thus we propose a new distributed cyber attack detection system which is robust and efficient than the traditional ones.

Let us assume there are $L-1$ known types of cyber attacks. Thus there are a total of L classes including the normal class. Let there be M sensors, which are distributed wide across the network observing different aspects of the network under consideration. Each sensor processes the observed data and makes a local decision regarding the network condition. The local decision of the sensor S_j is u_j. All

these local decisions are then transmitted to the fusion center, which generates the final decision about the state of the network using the available local decisions. The proposed distributed cyber attack detection system is shown in Figure 2.1.

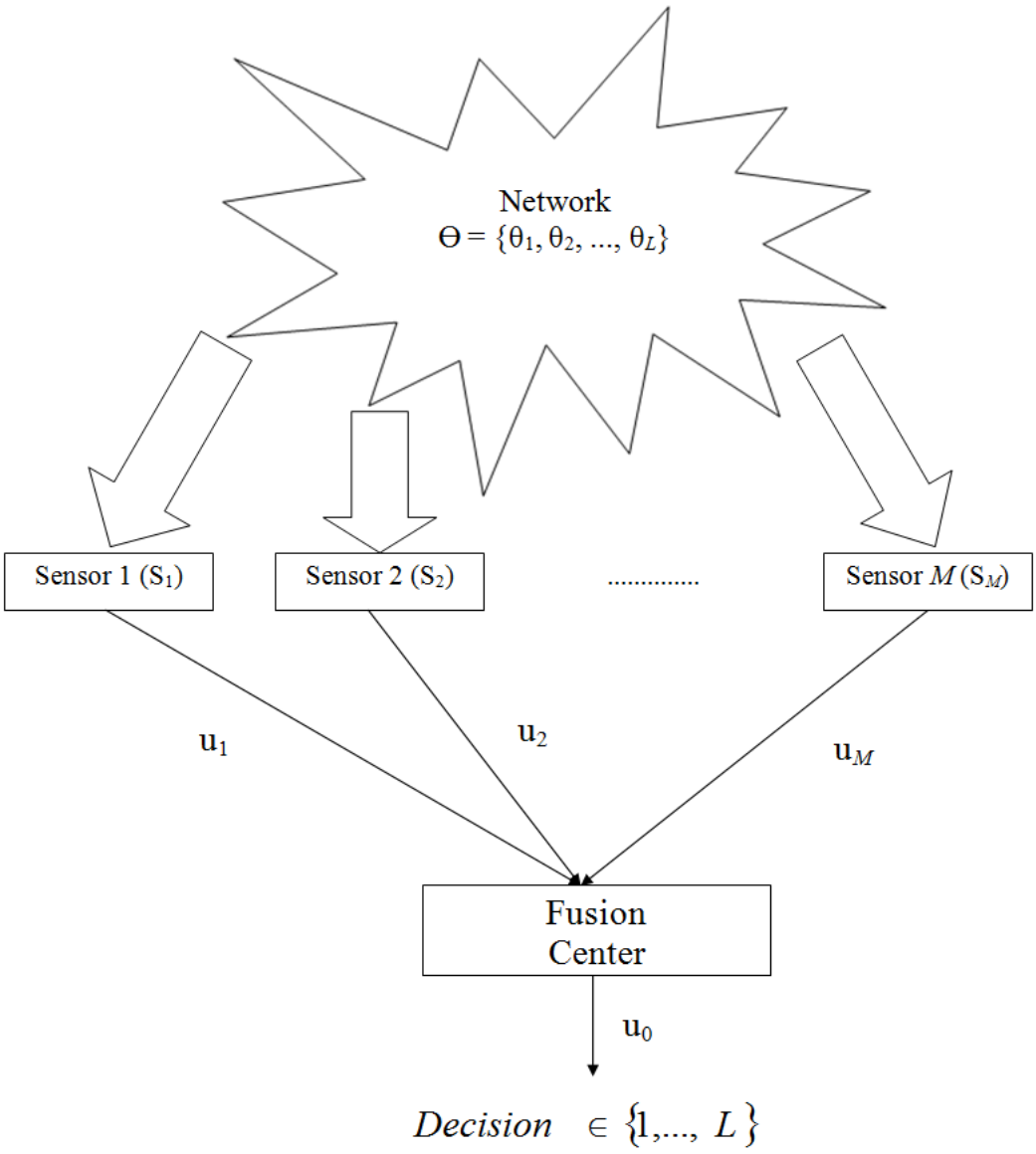

Figure 2.1: Proposed distributed cyber attack detection system.

At each sensor, there are L binary classifiers with each classifier distinguishing one class from the rest. Let $g_i^j(\cdot)$ denote the binary classifier i at the sensor j. The binary classifier $g_i^j(\cdot)$ at the sensor j classifies the observed data record into either "belonging to class i" or "not belonging to class i." The proposed local fusion rule

at each sensor generates the corresponding local decision based on the outputs of the L binary classifiers. This is clearly shown in Figure 2.2.

The binary classifiers in the proposed system are designed using the machine learning (statistical learning) approach. In general, we assume that we initially have data (generally referred to as the training data) of different types of cyber attacks available. All the binary classifiers are initially trained on this available training data. Gradually over time, when the data about the new types of cyber attacks becomes available, the training data is updated and the classifiers are retrained using the updated training data. In order to decrease the computational complexity of the classifier training process, a new fast and efficient training method is proposed in this thesis. Effective fusion rules, at the sensors and the fusion center, are proposed using the Dempster-Shafer theory of evidence.

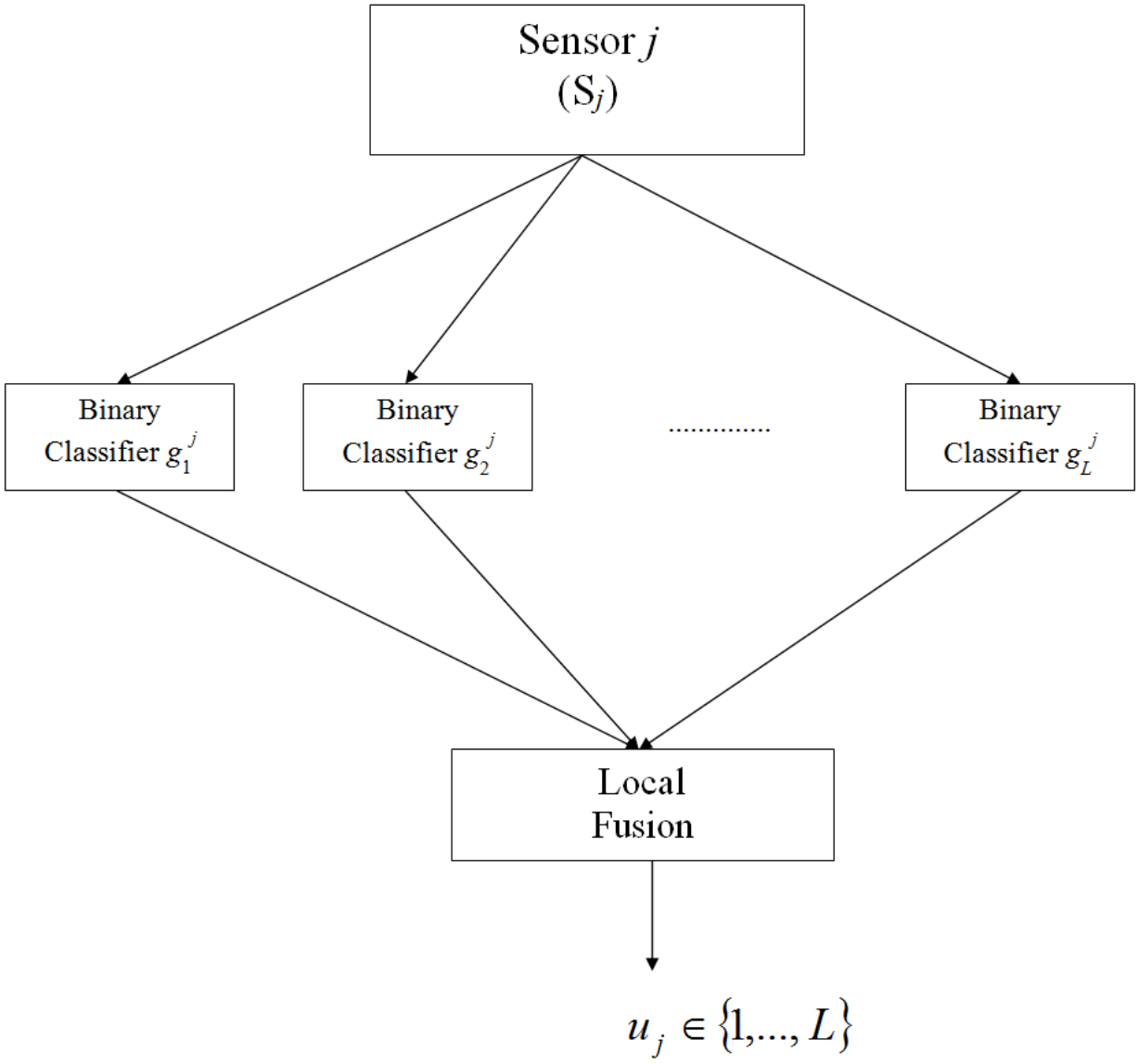

Figure 2.2: Local decision at each sensor in the system.

CHAPTER 3

SUPPORT VECTOR MACHINES

In this chapter, we describe in detail the support vector machine (SVM), which is used as the binary classifier in our proposed cyber attack detection system. Support vector machine (SVM) is generally considered to be the best off-the-shelf supervised classifier. The SVM tends to find a decision boundary (hyperplane) between the two classes, which lies at a maximum distance from both the classes. The main advantage of the support vector machines (SVMs) is that the parameters are found by solving a convex optimization problem, which makes the solution globally optimal.

The support vector machine (SVM) is a sparse kernel method. Kernel methods are a special class of pattern classification techniques, in which the decisions (or predictions) are made using the entire training data, or a subset of it. The support vector machine (SVM) is a kernel method with sparse solution. The SVM decisions (or predictions) are made using only few training data points.

The problem of supervised binary classification can be formulated as follows. The training set $\mathbf{T} = \{(\mathbf{x}_i, y_i); i = 1, ..., n\}$ is given. The training sample $\mathbf{x}_i$ has d features, i.e., $\mathbf{x}_i \in \mathbb{R}^d$. Thus each training sample $\mathbf{x}_i$ is a point in $\mathbb{R}^d$ space. The label of the sample $\mathbf{x}_i$ is $y_i \in \{-1, +1\}$ corresponding to two different classes. The SVM classifier design is to learn a function $y = f(\mathbf{x})$ which not only classifies the training data accurately but also generalizes well to the new samples (samples with unknown labels). Mathematically, the SVM classifier can be defined as

$$y = f(\mathbf{x}) = \text{sgn}(\mathbf{w}^T \Phi(\mathbf{x}) + b), \tag{3.1}$$

where $\mathbf{w}$ and b are the parameters of the SVM, and $\Phi(\cdot)$ denotes a nonlinear feature

space mapping which transforms the original d-dimensional input space into some higher dimensional feature (Hilbert) space. From (3.1), it is clear that the SVM parameters $\mathbf{w}$ and b define a linear hyperplane in the feature space corresponding to the nonlinear mapping $\Phi(\cdot)$. The optimal values of the SVM parameters $\mathbf{w}$ and b are found by solving a convex optimization problem which maximizes the margin between the two classes.

3.1 Margins and Maximum Margin Classifiers

First, we discuss the concept of margins. Figure 3.1 shows the decision boundary between two classes. The parameter vector $\mathbf{w}$ is orthogonal to the decision hyperplane. Consider a training sample (data point) at A. The geometric margin (or simply margin) of the decision boundary with respect to this training sample A is the distance of the point A to the decision boundary. This margin is represented by the line segment AB in Figure 3.1. Mathematically, the margin of the decision boundary represented by the parameters $(\mathbf{w}, b)$ with respect to a training sample $(\mathbf{x}_i, y_i)$ is defined as

$$\gamma_i = y_i \left(\left(\frac{\mathbf{w}}{\|\mathbf{w}\|} \right)^T \mathbf{x}_i + \frac{b}{\|\mathbf{w}\|} \right). \tag{3.2}$$

Using the notion of margins, the optimal classifier with good generalization is the one which maximizes the margin with respect to the entire training set $\mathbf{T} = \{(\mathbf{x}_i, y_i); i = 1, ..., n\}$. Such classifiers that separate the training samples with the maximum margin (large gap) are called *maximum margin classifiers*. The support vector machine (SVM) is one such maximum margin classifier. Given a training set $\mathbf{T} = \{(\mathbf{x}_i, y_i); i = 1, ..., n\}$, the support vector machine (SVM) tries to solve the following optimization problem:

$$\begin{aligned} \max_{\mathbf{w},b} \quad & \gamma \\ \text{s.t.} \quad & y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq \gamma, \; i = 1, ..., n \\ & \|\mathbf{w}\| = 1. \end{aligned} \tag{3.3}$$

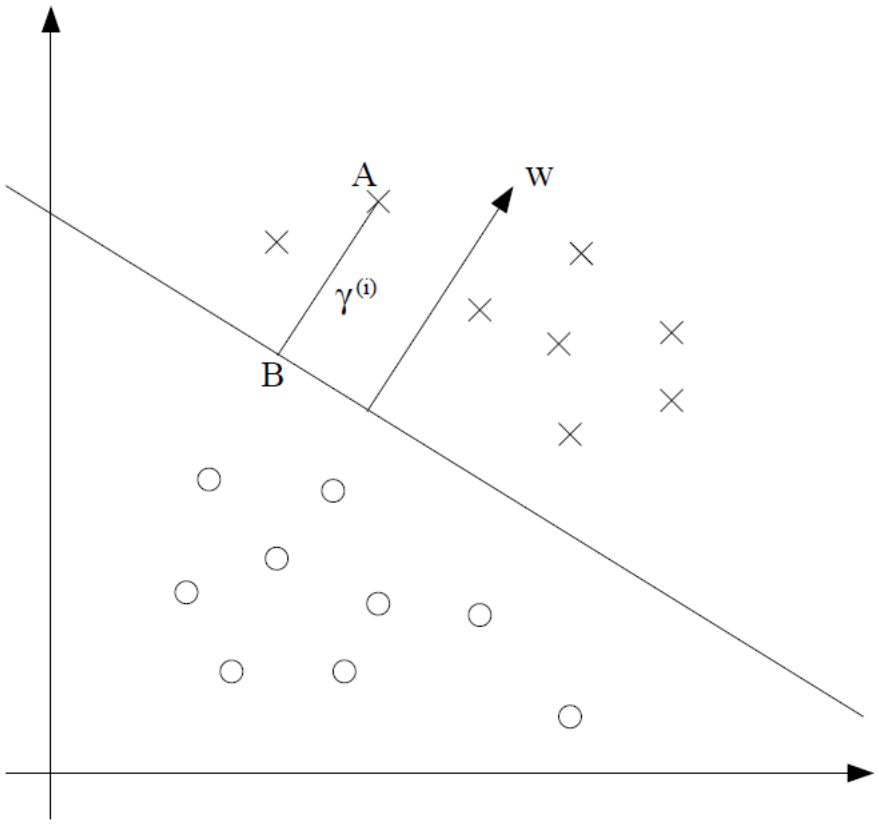

Figure 3.1: Geometric margins.

In the above optimization problem we are trying to maximize the margin γ, subject to the constraint that every training sample has the margin at least γ. The solution of the optimization problem (3.3) gives the hyperplane $(\mathbf{w}, b)$ with the largest possible margin with respect to the entire training set.

The above optimization problem (3.3) cannot be solved directly, as the constraint $\|\mathbf{w}\| = 1$ is a nonconvex one. We can try to embed the constraint $\|\mathbf{w}\| = 1$ into the objective function, which results in the following optimization problem:

$$\begin{aligned} \max_{\mathbf{w},b} \quad & \frac{\hat{\gamma}}{\|\mathbf{w}\|} \\ \text{s.t.} \quad & y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq \hat{\gamma},\ i = 1, ..., n \end{aligned} \tag{3.4}$$

Solving this optimization problem (3.4) is also in general difficult as the objective function in (3.4) is a nonconvex one. To transform this optimization problem (3.4)

into a convex one, we make use of the fact that scaling the parameters $(\mathbf{w}, b)$ does not affect the geometric margin. This is clear from the equation (3.2). We scale the parameters $(\mathbf{w}, b)$ by some factor such that $\hat{\gamma} = 1$. Now the objective function of the optimization problem (3.4) becomes $\frac{1}{\|\mathbf{w}\|}$. Moreover, maximizing $\frac{1}{\|\mathbf{w}\|}$ is equivalent to minimizing $\|\mathbf{w}\|^2$. So we now have the following optimization problem:

$$\begin{aligned} \min_{\mathbf{w},b} \quad & \frac{1}{2} \|\mathbf{w}\|^2 \\ \text{s.t.} \quad & y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1, \; i = 1, ..., n \end{aligned} \tag{3.5}$$

The scaling factor $\frac{1}{2}$ in the objective function is introduced for mathematical convenience. This optimization problem (3.5) is a convex one. More precisely, the optimization problem (3.5) is a quadratic programming (QP) problem, which tries to minimize a quadratic function subject to a set of linear inequality constraints.

In practice, the dual problem of the above problem is solved. The Lagrangian of the above problem (3.5) is

$$L(\mathbf{w}, b, \boldsymbol{\lambda}) = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^{n} \lambda_i (y_i(\mathbf{w}^T \mathbf{x}_i + b) - 1), \tag{3.6}$$

where $\boldsymbol{\lambda} = \{\lambda_i \geq 0, i = 1, ..., n\}$ are the Lagrange multipliers. The Lagrange dual function (or simply dual function) is

$$\begin{aligned} D(\boldsymbol{\lambda}) &= \inf_{\mathbf{w},b} L(\mathbf{w}, b, \boldsymbol{\lambda}) \\ &= \inf_{\mathbf{w},b} \left(\frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^{n} \lambda_i (y_i(\mathbf{w}^T \mathbf{x}_i + b) - 1) \right). \end{aligned} \tag{3.7}$$

The Lagrange dual function is the minimum value of the Lagrangian over the parameters $(\mathbf{w}, b)$. To find the dual function, we first need to set the derivatives of the Lagrangian $L(\mathbf{w}, b, \boldsymbol{\lambda})$ with respect to $\mathbf{w}$ and b equal to zero.

$$\nabla_{\mathbf{w}} L(\mathbf{w}, b, \boldsymbol{\lambda}) = \mathbf{w} - \sum_{i=1}^{n} \lambda_i y_i \mathbf{x}_i = 0, \tag{3.8a}$$

$$\frac{\partial}{\partial b} L(\mathbf{w}, b, \boldsymbol{\lambda}) = \sum_{i=1}^{n} \lambda_i y_i = 0. \tag{3.8b}$$

From (3.8a), we have

$$\mathbf{w} = \sum_{i=1}^{n} \lambda_i y_i \mathbf{x}_i. \tag{3.9}$$

Plugging the value of $\mathbf{w}$ from (3.9) into the Lagrangian (3.6), we get

$$L(b, \boldsymbol{\lambda}) = \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j \lambda_i \lambda_j (\mathbf{x}_i)^T \mathbf{x}_j - b \sum_{i=1}^{n} \lambda_i y_i. \tag{3.10}$$

From (3.8b), the last term on the RHS of the above equation is zero. Finally, the Lagrange dual function is

$$D(\boldsymbol{\lambda}) = \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j \lambda_i \lambda_j (\mathbf{x}_i)^T \mathbf{x}_j. \tag{3.11}$$

The Lagrange dual function (3.11) gives lower bounds on the optimal value of the original primal problem (3.5). If $p^\star$ is the optimal value of the primal optimization problem (3.5), then for any $\boldsymbol{\lambda} \geq 0$ we have

$$D(\boldsymbol{\lambda}) \leq p^\star. \tag{3.12}$$

It is clear that the Lagrange dual function $D(\boldsymbol{\lambda})$ gives lower bound on the optimal value $p^\star$ of the primal optimization problem (3.5) for every $\boldsymbol{\lambda} \geq 0$. The best lower bound that can be given by the dual function can be found by the following optimization problem, which is called the Lagrange dual problem (or simply dual problem).

$$\begin{aligned} \max_{\boldsymbol{\lambda}} \quad & D(\boldsymbol{\lambda}) \\ \text{s.t.} \quad & \boldsymbol{\lambda} \geq 0. \end{aligned} \tag{3.13}$$

The dual problem of our original optimization problem (3.5) is

$$\begin{aligned} \max_{\boldsymbol{\lambda}} \quad & D(\boldsymbol{\lambda}) = \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j \lambda_i \lambda_j (\mathbf{x}_i)^T \mathbf{x}_j \\ \text{s.t.} \quad & \lambda_i \geq 0, \; i = 1, ..., n \\ & \sum_{i=1}^{n} \lambda_i y_i = 0. \end{aligned} \tag{3.14}$$

The above dual problem (3.14) is also a quadratic programming (QP) problem, which tries to minimize a quadratic function subject to a set of inequality and equality

constraints. Let $d^\star$ denote the optimal value of the above dual problem (3.14). Thus, by definition, we always have $d^\star \leq p^\star$. The nonnegative quantity $p^\star - d^\star$ is called the duality gap. In order to actually solve the dual problem (3.14) in place of the original primal problem (3.5), we need to have zero duality gap, i.e., $d^\star = p^\star$. In order to have zero duality gap, the primal optimal points $(\mathbf{w}^\star, b^\star)$ and the dual optimal point $\boldsymbol{\lambda}^\star$ must satisfy the Karush-Kuhn-Tucker (KKT) conditions:

$$y_i(\mathbf{w}^{\star T}\mathbf{x}_i + b^\star) \geq 1,\ i = 1, ..., n \tag{3.15a}$$

$$\lambda_i^\star \geq 0,\ i = 1, ..., n \tag{3.15b}$$

$$\lambda_i^\star(1 - y_i(\mathbf{w}^{\star T}\mathbf{x}_i + b^\star)) = 0,\ i = 1, ..., n \tag{3.15c}$$

$$\nabla_{(\mathbf{w},b)} L(\mathbf{w}^\star, b^\star, \boldsymbol{\lambda}^\star) = 0. \tag{3.15d}$$

These KKT conditions hold in our case here. The KKT condition (3.15c) is an important one, which is generally referred to as the KKT dual complementarity condition. It states that, for every data point $\mathbf{x}_i$, either the corresponding $\lambda_i = 0$ or $y_i(\mathbf{w}^{\star T}\mathbf{x}_i + b^\star) = 1$. Thus we can solve the dual problem (3.14) instead of the original primal problem (3.5). The various algorithms used to solve the dual problem (3.14) are discussed later in this chapter.

Once the dual problem (3.14) is solved, we get the dual optimal point $\boldsymbol{\lambda}^\star$. Having found the dual optimal point $\boldsymbol{\lambda}^\star$, we can find the primal optimal point $\mathbf{w}^\star$ using (3.9). The optimal value $b^\star$ can be calculated using (3.9) and the KKT dual complementarity condition (3.15c). In order to classify new samples using the trained SVM $(\mathbf{w}^\star, b^\star)$, we just need to evaluate the sign of the quantity $f(\mathbf{x}) = \mathbf{w}^{\star T}\mathbf{x} + b^\star$. Using (3.9), we

have

$$
\begin{aligned}
f(\mathbf{x}) &= \mathbf{w}^{\star T}\mathbf{x} + b^{\star} \\
&= \left(\sum_{i=1}^{n} \lambda_i^{\star} y_i \mathbf{x}_i \right)^T \mathbf{x} + b^{\star} \\
&= \sum_{i=1}^{n} \lambda_i^{\star} y_i \left(\mathbf{x}_i\right)^T \mathbf{x} + b^{\star} \\
&= \sum_{i=1}^{n} \lambda_i^{\star} y_i \left\langle \mathbf{x}_i, \mathbf{x} \right\rangle + b^{\star},
\end{aligned}
\tag{3.16}
$$

where $\langle \mathbf{x}_i, \mathbf{x} \rangle$ represents the inner product between the points $\mathbf{x}_i$ and $\mathbf{x}$. The optimal value $b^{\star}$ in the above equation can be directly calculated using (3.9) and the KKT dual complementarity condition (3.15c). The value $b^{\star}$ also depends only on the Lagrange multipliers $\{\lambda_i\}$, training data points $\{\mathbf{x}_i\}$, and the training data labels $\{y_i\}$. It is clear from (3.16) that once the dual optimal point $\boldsymbol{\lambda}^{\star}$ is found by solving the dual problem (3.14), the new data points can be classified directly using (3.16). There is no need to explicitly calculate the primal optimal point $(\mathbf{w}^{\star}, b^{\star})$ for future predictions about new data points. From (3.16), it is clear that the quantity $f(\mathbf{x})$ mainly depends on the inner product between the new point $\mathbf{x}$ and the points in the training set. From the the KKT dual complementarity condition (3.15c) and (3.16), it is clear that only few data points have nonzero λ_i. Such points in the training set are called *support vectors*, and hence the name **support vector machine (SVM)**. All the training data points except support vectors have $\lambda_i = 0$, and thus does not play any role in forming the decision boundary and making predictions for new data points.

3.2 Nonlinear Feature Space Mapping and Kernel Trick

From the above discussion, it clear that the support vector machine (SVM) is a maximum margin classifier which tries to separate the linearly separable data. Most of the time in real-world situations, the data is not linearly separable in the original d-dimensional input space $\boldsymbol{I}$. However, the data can be linearly separable in some higher

dimensional feature space (Hilbert space) $\boldsymbol{H}$. Hence, the original input space $\boldsymbol{I}$ is transformed into a higher dimensional feature space $\boldsymbol{H}$ through a general nonlinear feature mapping $\Phi(\cdot)$. The SVM is applied in this new higher dimensional space. In order to do this, we need to replace $\mathbf{x}$ everywhere in the SVM algorithm with $\Phi(\mathbf{x})$. Since the SVM algorithm can be expressed entirely using the inner product $\langle \mathbf{x}_p, \mathbf{x}_q \rangle$, we need to replace all such inner products with $\langle \Phi(\mathbf{x}_p), \Phi(\mathbf{x}_q) \rangle$. This new inner product can be explicitly defined using a kernel function. For a given nonlinear feature mapping $\Phi(\cdot)$, the corresponding kernel function is defined as

$$K(\mathbf{x}_p, \mathbf{x}_q) = \langle \Phi(\mathbf{x}_p), \Phi(\mathbf{x}_q) \rangle = (\Phi(\mathbf{x}_p))^T \Phi(\mathbf{x}_q). \tag{3.17}$$

Thus, we can just replace the inner product $\langle \mathbf{x}_p, \mathbf{x}_q \rangle$ everywhere in the SVM algorithm by the corresponding kernel function $K(\mathbf{x}_p, \mathbf{x}_q)$.

Given the nonlinear feature mapping $\Phi(\cdot)$, the corresponding kernel $K(\cdot, \cdot)$ can be easily calculated. But, sometimes determining the nonlinear feature mapping $\Phi(\cdot)$ is very difficult, especially in very high-dimensional cases. However most of the time, we can efficiently construct the corresponding kernel function $K(\cdot, \cdot)$ directly without even having to find the feature mapping $\Phi(\cdot)$ explicitly. This is a huge advantage. However, we need to make sure that the constructed function is a valid kernel, i.e., the constructed kernel function should correspond to a scalar product in some (high-dimensional) feature space. One way to do this is to expand the chosen kernel function and identify the corresponding mapping $\Phi(\cdot)$. This may be difficult in some situations. A more simple way to check whether a function is a valid kernel or not is to use the *Mercer's condition*.

For a function $K(\cdot, \cdot)$ to be valid kernel, corresponding to some nonlinear feature mapping $\Phi(\cdot)$, it needs to satisfy the Mercer's condition: Given a data set $\{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$, $K : \mathbb{R}^d \times \mathbb{R}^d \mapsto \mathbb{R}$ is a valid kernel if the corresponding kernel matrix $\mathbf{K}$ is symmetric positive-semidefinite.

Thus, we can simply replace the inner product between the data points $\langle \mathbf{x}_p, \mathbf{x}_q \rangle$

in the SVM algorithm with the kernel function $K(\mathbf{x}_p, \mathbf{x}_q)$ corresponding to a higher dimensional feature space, where the data is linearly separable, and apply the SVM algorithm in the new feature space. This is generally referred to as the *kernel trick*, which can be used for any classifier learning algorithms that can be explicitly expressed in terms of inner products between the data points.

Some examples of the kernels include:

- Linear kernel:

$$K(\mathbf{x}_p, \mathbf{x}_q) = \left(\mathbf{x}_p^T \mathbf{x}_q\right). \tag{3.18}$$

  This is the simplest kernel corresponding to the feature mapping $\Phi(\mathbf{x}) = \mathbf{x}$.

- Polynomial kernel:

$$K(\mathbf{x}_p, \mathbf{x}_q) = \left(\mathbf{x}_p^T \mathbf{x}_q\right)^d, \tag{3.19}$$

  where d is the degree.

- Gaussian kernel:

$$K(\mathbf{x}_p, \mathbf{x}_q) = \exp\left(-\frac{\|\mathbf{x}_p - \mathbf{x}_q\|^2}{2\sigma^2}\right). \tag{3.20}$$

  The Gaussian kernel (3.20) corresponds to an infinite dimensional feature mapping. The Gaussian kernel is also referred to as the radial basis function kernel (RBF kernel).

- Hyberbolic tangent kernel:

$$K(\mathbf{x}_p, \mathbf{x}_q) = \tanh(a\left(\mathbf{x}_p^T \mathbf{x}_q\right) + b), \tag{3.21}$$

  for some $a > 0$ and $b < 0$.

Also new kernels can be constructed from the old kernels using the following properties. Given valid kernels $K_1(\cdot, \cdot)$ and $K_2(\cdot, \cdot)$, the following kernels will also be valid:

- $K(\mathbf{x}_p, \mathbf{x}_q) = K_1(\mathbf{x}_p, \mathbf{x}_q) + K_2(\mathbf{x}_p, \mathbf{x}_q)$.
- $K(\mathbf{x}_p, \mathbf{x}_q) = K_1(\mathbf{x}_p, \mathbf{x}_q)K_2(\mathbf{x}_p, \mathbf{x}_q)$.
- $K(\mathbf{x}_p, \mathbf{x}_q) = cK_1(\mathbf{x}_p, \mathbf{x}_q)$, where c is a constant.
- $K(\mathbf{x}_p, \mathbf{x}_q) = \exp(K_1(\mathbf{x}_p, \mathbf{x}_q))$.
- $K(\mathbf{x}_p, \mathbf{x}_q) = f(K_1(\mathbf{x}_p, \mathbf{x}_q))$, where $f(\cdot)$ is a polynomial with nonnegative coeffecients.
- $K(\mathbf{x}_p, \mathbf{x}_q) = f(\mathbf{x}_p)K_1(\mathbf{x}_p, \mathbf{x}_q)f(\mathbf{x}_q)$, where $f(\cdot)$ is any function.

3.3 Regularization and Soft Margins

Till now, we have assumed that the training data is linearly separable either in the original input space or in some higher dimensional feature space. However in some real-world situations, this may not be the case. The data cannot be linearly separable, even in the higher dimensional spaces. This is due to the fact that the underlying true class distributions, that generate the data, may have a significant overlap. In such cases, trying to exactly separate the training data may lead to overfitting and hence poor generalization. In such cases, we need to allow the SVM to misclassify some of the training points for good generalization. Also sometimes, there may be some extreme outliers in the data. The original SVM algorithm, which tries to exactly separate the training data, will be extremely sensitive to such outliers. This is clearly illustrated in the Figure 3.2. In such cases as well, we need to allow the misclassification of some of the training points.

In case of linearly separable classes, the original SVM uses an error function that gives an infinite error when a training sample is misclassified and zero error when it is correctly classified, which is optimized with respect to the SVM parameters to maximize the margin [7]. Now we can modify this error function so that the data

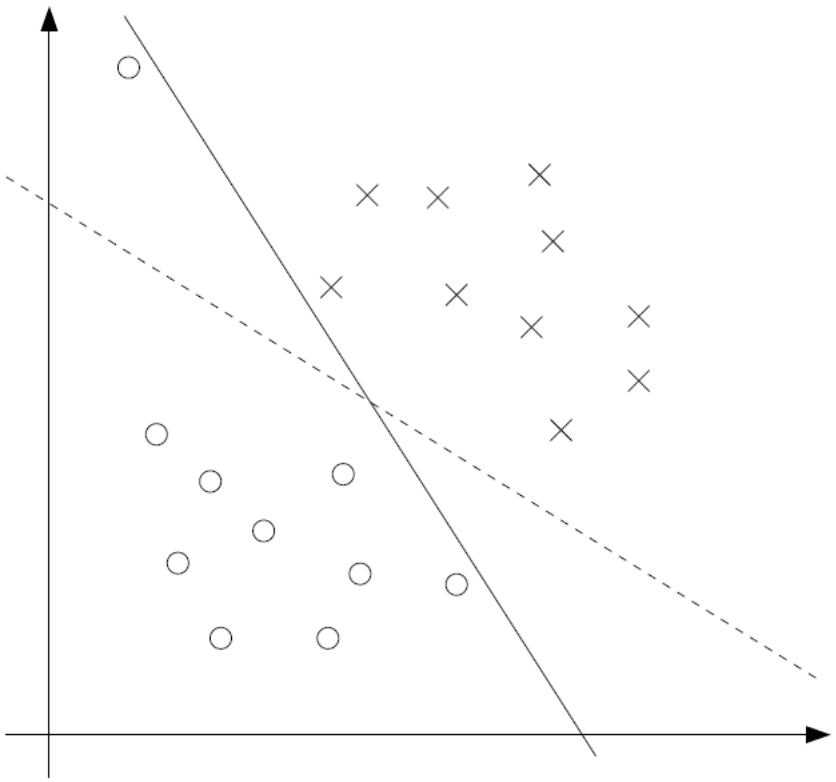

Figure 3.2: Sensitivity of the original SVM (with hard margin) to outliers.

points are allowed to be on the wrong side of the decision boundary with a penalty that increases with the distance from the decision boundary. The original SVM optimization problem (3.5) can be reformulated using L_1 regularization as

$$\begin{aligned} \min_{\mathbf{w},b} \quad & \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\xi_i \\ \text{s.t.} \quad & y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i,\ i = 1, ..., n \\ & \xi_i \geq 0,\ i = 1, ..., n \end{aligned} \tag{3.22}$$

where the quantities $\xi_i, i = 1, ..., n$ are called slack variables. They are defined as follows:

- For the data points that are correctly classified, and which lie on or outside the correct side SVM margin boundary, $\xi_i = 0$.

- For the data points that are correctly classified, and which lie within the correct

side SVM margin, $0 < \xi_i < 1$.

- For the data points that lie exactly on the decision boundary, $\xi_i = 1$.
- For the data points that are misclassified (i.e., those which lie on the wrong side of the decision boundary), $\xi_i > 1$.

Thus, for every data point $\mathbf{x}_i$ that is misclassified, the objective function in (3.22) is penalized by the quantity $C\xi_i$. Hence the parameter $C > 0$ controls the trade-off between the twin objectives of maximizing the margin and minimizing the number of data points that are allowed to be misclassified. The parameter C can also be interpreted as analogous to a regularization parameter which controls the trade-off between minimizing the number of classification errors and controlling the model complexity. Thus the SVM is relaxed to include some misclassifications, which leads to a soft margin instead of a hard margin.

As before, the dual problem of the above problem (3.22) is solved in practice. The Lagrangian of the above problem (3.22) is

$$L(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\lambda}, \boldsymbol{\nu}) = \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^{n} \xi_i - \sum_{i=1}^{n} \lambda_i (y_i(\mathbf{w}^T \mathbf{x}_i + b) - 1 + \xi_i) - \sum_{i=1}^{n} \nu_i \xi_i, \tag{3.23}$$

where $\boldsymbol{\lambda} = \{\lambda_i \geq 0, i = 1, ..., n\}$ and $\boldsymbol{\nu} = \{\nu_i \geq 0, i = 1, ..., n\}$ are the Lagrange multipliers. The dual problem is given by

$$\begin{aligned} \max_{\boldsymbol{\lambda}} \quad & D(\boldsymbol{\lambda}) = \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j \lambda_i \lambda_j (\mathbf{x}_i)^T \mathbf{x}_j \\ \text{s.t.} \quad & 0 \leq \lambda_i \leq C, \; i = 1, ..., n \\ & \sum_{i=1}^{n} \lambda_i y_i = 0. \end{aligned} \tag{3.24}$$

The above dual problem (3.24) is similar to the earlier dual problem (3.14) with the exception of the constraint on the Lagrange multipliers $\boldsymbol{\lambda} = \{\lambda_i, i = 1, ..., n\}$. The kernel trick can be applied here, and the inner product in the above dual problem is replaced by the kernel $K(\mathbf{x}_i, \mathbf{x}_j)$.

3.4 Solving the SVM Dual Problem

We now describe the algorithms used to solve the SVM dual problem (3.24). The SVM dual problem (3.24) is a quadratic programming (QP) problem, which tries to minimize a quadratic function subject to a set of inequality and equality constraints. Though the support vector machine (SVM) is a sparse kernel algorithm which uses relatively few basis functions (as defined by the support vectors) to make predictions for new data, the training/learning algorithm of the SVM uses the entire training data. Thus, some efficient algorithms are required for solving the SVM dual problem.

The dual problem (3.24) is a quadratic programming (QP) problem. The objective function in (3.24) is a quadratic one, which implies that any local optimal point is also a global optimal point. Directly solving the quadratic programming (QP) problem is often difficult and infeasible as the computational and memory requirements are prohibitively high. The kernel matrix $\mathbf{K} \in \mathbb{R}^{n \times n}$ itself takes huge memory space. When the size of the training data (n) is large, which is often the case in many real-world applications, the SVM runs out of memory. Thus, some practical approaches need to used in solving the SVM dual quadratic programming (QP) problem.

One such method is the chunking method [8]. In the chunking method, the actual QP problem is broken down into a series of smaller QP problems, which identify the non-zero Lagrange multipliers. The basic idea in this method is to scale down the size of the kernel matrix by discarding all those elements which correspond to zero Lagrange multipliers, and finally the scaled down QP problem is solved. Though this method scales down the size of the problem, the computational complexity of this method is still high for large training datasets as the method still involves several matrix operations, and computation of gradients of the dual function. Decomposition methods [9] also try to solve a series of series of smaller QP problems. The only difference is that each of these smaller QP problems is of a fixed size. These methods also suffer from the same disadvantages, and are not suitable for large training

datasets.

In general, these methods are found to have computational complexity $O(n^3)$, where n is the size of the training dataset. These methods clearly does not scale well with the training data. For arbitrarily large training datasets, these methods are not suitable. In the next chapter, some fast and efficient training algorithms are presented which can be applied in applications involving very large datasets.

3.5 Summary

The support vector machine (SVM) is a maximum margin classifier which tries to separate the data of two classes with maximum possible margin. Since the SVM generates a linear hyperplane (decision boundary), in order to generalize to nonlinearly separable data, the kernel trick is employed to transform the input space to a high dimensional space, where the data becomes linearly separable. Thus theoretically, the support vector machine (SVM) has two main phases:

1. The d-dimensional original input space $\boldsymbol{I}$ is transformed into a higher dimensional feature space (Hilbert space) $\boldsymbol{H}$ through a general nonlinear mapping $\Phi(\cdot)$. Usually, the kernel trick is used which does not require the explicit computation of the nonlinear mapping $\Phi(\cdot)$.

2. The separating hyperplane (decision boundary), with maximum possible margin, is then constructed in the high dimensional feature space $\boldsymbol{H}$. This maximum margin decision hyperplane is obtained by solving the SVM dual quadratic programming problem.

The support vector machine (SVM), in general, generates hard classification decisions for the new inputs. That is, the SVM decides whether the new input belongs to one class or the other. The SVM does not generate probabilistic outputs. However, in some situations, we are more interested in the probabilistic outputs instead of hard

decisions. To address this issue, Platt proposed a post-processing approach in which a logistic sigmoid is fitted to the outputs of an already trained support vector machine [10]. This approach is described in detail in Appendix A.

CHAPTER 4

FAST AND EFFICIENT SVM TRAINING APPROACHES

Support vector machines (SVMs) are generally considered to be the best off-the-shelf supervised classifiers. However, the main drawback of support vector machines is that the training procedure has a very high computational complexity. This high computational complexity often limits the real-time implementation of support vector machines, especially in applications involving very large datasets like cyber attack detection. In this chapter, we first discuss some of the existing approaches which try to reduce the SVM training complexity, and then present our proposed fast training approach which further reduces the SVM training complexity without significantly degrading the classification performance of the SVM.

4.1 Existing Fast SVM Training Approaches

In [11], [12], greedy approximation of the kernel matrix and low-rank kernel representation are proposed, respectively. The problem with these approximation-based approaches is that they have a significant impact on the classification performance [13]. There other approaches which try to solve the quadratic programming (QP) problem more efficiently. One such method is the chunking method [8], which was described in the previous chapter. As mentioned before, this method is found to have complexity $O(n^3)$ [14], where n is the size of the training set, and hence is not very efficient for training SVM on large datasets. Another popular approach is the sequential minimal optimization (SMO) [14]. The sequential minimal optimization (SMO) algorithm is basically a coordinate ascent algorithm. The SMO algorithm is a

simpler and more efficient algorithm for solving the QP problem. The SMO basically takes the previous chunking concept to an extreme limit. The SMO algorithm breaks the original QP problem into a series of smaller QP subproblems. At every step, the SMO solves a smaller optimization problem of finding the optimal values of two Lagrange multipliers, and updates the SVM accordingly. The main advantage of the SMO lies in the fact that solving for two Lagrange multipliers at each step is done analytically, thus avoiding the matrix computations and standard QP calculations on the whole. In general, the SMO method is found to have complexity $O(n^2)$ [14], where n is the size of the training dataset. There are some other approaches which try to avoid the quadratic program in the SVM algorithm [15]. However they still involve the kernel trick which has the high complexity, and their performance greatly depends on factors such as random sampling, selection of the hyperparameter values, and stopping criteria.

On the other hand, there are improved training approaches which do not require any type of approximation in the SVM algorithm, but rather focus on the appropriate training data selection for SVMs [16]. In [16], a reduced SVM (RSVM) is proposed based on a simple random sampling. Tong and Koller propose an active training approach where the SVM learner (classifier) sequentially selects the training samples based on certain criterion [17]. Among the proposed three methods in [17], MaxMin and Ratio methods are computationally expensive as they tend to train the SVM multiple times in each iteration. The Simple method, though relatively faster, is more unstable and performed poorly on the Newsgroup data. Its complexity greatly depends on the price of each query. Since the decision boundary of an SVM depends only on a small subset of the training data (support vectors), fast training can be achieved by identifying and selecting training samples that are support vectors. This idea has been explored in [18], [19], [13], [20], [21], [22]. In [18], Abe and Inoue extract the samples close to the boundary using the Mahalanobis distance measure, which

generally performs well when the data belonging to each class is clustered together and when there is a minimal overlap between the data of different classes. In [19], Shin and Cho propose a method which selects training samples near the boundary using the neighborhood properties of samples. Their method is based on the k-nearest neighbor (k-NN) algorithm, and tends to be computationally expensive in case of large high-dimensional data. Though they used a faster selective k-NN spanning approach, the performance of their method greatly depends on k, whose value was randomly chosen in different situations. In [13], Li *et al.* propose a method based on edge detection whose performance depends on prefixed parameters including k (in k-means clustering) and m (the number of neighbors). The method proposed by Lyhyaoui *et al.* [20] also depends on some additional parameters that need to be set beforehand. In [21], the training sample selection is done using the k-means clustering technique. The performance of their method greatly depends on k, whose value is randomly chosen for different cases. Fuzzy clustering based training data selection is proposed in [22]. The proposed fuzzy clustering method works well when the exact number of clusters in the data is known beforehand. Clearly, the performance of most of these methods depends on certain key parameters which need to be fixed beforehand. These methods generally perform well if the preset parameter values exactly capture the nature of the training data. However, determining these key parameters is generally a non-trivial task, especially in high-dimensional spaces.

4.2 A New Fast SVM Training Approach

We propose a new fast training method in which there is no need of presetting any parameters. The basic idea is to reduce the size of the training data by retaining only the most informative samples that are likely to become support vectors. First, we detect the clusters present in the data. We use the nonparametric Bayesian approach for clustering which does not require any preset parameters. After forming the clusters,

we propose an efficient sample elimination approach based on logistic regression. The proposed sample elimination approach takes two important factors into account: relative position of a sample within the cluster and the relative position of a sample with respect to the other class data. By doing this, the SVM training complexity is shown to be much reduced without significantly degrading the classification performance of the SVM.

In the proposed fast training approach, our goal is to scale down the training set by retaining only the most relevant (most informative) samples and removing the least relevant samples. The main issue here is to determine which samples are the most informative and which are not. It is known that the samples which are close to the decision boundary are more important to form the boundary than the ones which are far away. In other words, the samples which are close to the boundary tend to be more informative for SVMs. Eventually such samples tend to become the support vectors. We make use of this fact to design our fast training approach. First, we accurately detect the clusters present in the training data. In each cluster, the outward samples which are closer to the other class samples, tend to be more informative than the inward samples. We then detect the most informative samples in each cluster, and retain them in the training set. All the other (least informative) samples are safely eliminated from the training set.

4.2.1 Training Data Clustering

First, we need to accurately form multiple clusters for each class present in the training data. One way to do this is to use any existing clustering based technique to form the clusters. However, this simple and heuristic approach may not perform well. This is because most of the existing clustering techniques tend to form spherical clusters only. But the true clusters in the data need not always be spherical. They can exist in any arbitrary shape. We need to accurately model the true shapes of clusters present in

the data, in order to accurately detect the samples which are well inside the clusters. In this thesis, a more sophisticated method for this task is proposed.

We define a cluster, in a more general way, to be the convex hull of a set of data points, which is the set of all convex combinations of the points. In other words, it is the smallest convex set that contains all the points. Thus, the cluster of p points $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_p \in \mathbb{R}^d$ is defined as

$$\begin{aligned} &\mathbf{conv}\{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_p\} \\ &\quad =\{\lambda_1\mathbf{x}_1+\lambda_2\mathbf{x}_2+...+\lambda_p\mathbf{x}_p \mid \lambda_i \geq 0,\, i = 1, ..., p,\, \lambda_1+\lambda_2+...+\lambda_p = 1\}. \end{aligned} \tag{4.1}$$

This is shown in Figure 4.1. In order to determine the center of a cluster, we need to calculate the center of the convex hull. Since the convex hull of a set of points is a polyhedron defined by its vertices, finding the exact center of this polyhedron is in general difficult, especially in high-dimensional spaces. Hence, we approximate the cluster (convex hull) using the *Löwner-John ellipsoid*, which is the minimum volume ellipsoid containing the cluster. There are many nice properties of ellipsoids. They are generally considered to be an universal geometric approximator of convex sets as they have sufficient degrees of freedom. Furthermore, ellipsoidal approximation is invariant under affine coordinate transformations. Finding this Löwner-John ellipsoid can be formulated as a convex optimization problem. The minimum volume ellipsoid containing the finite set of points $\{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_p\}$ is the same as the minimum volume ellipsoid containing the polyhedron $\mathbf{conv}\{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_p\}$. The Löwner-John ellipsoid approximating a cluster (convex hull) is shown in Figure 4.2.

Generally an ellipsoid in $\mathbb{R}^d$ space is defined as

$$\mathbb{E}(\mathbf{c}, \mathbf{P}) = \{\mathbf{x} \in \mathbb{R}^d | (\mathbf{x} - \mathbf{c})^T \mathbf{P}^{-1} (\mathbf{x} - \mathbf{c}) \leq 1\}, \tag{4.2}$$

where $\mathbf{c} \in \mathbb{R}^d$ is the center of the ellipsoid and $\mathbf{P} \in \mathbb{R}^{d \times d}$ is a positive definite matrix which defines the shape and size of the ellipsoid. Using the ellipsoidal (general

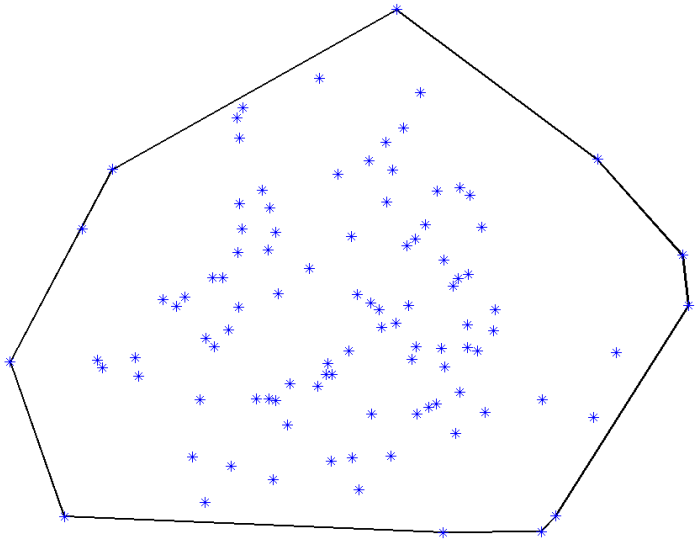

Figure 4.1: Cluster defined as the convex hull of the data points.

Euclidean) norm $\|\cdot\|_{\mathbf{P}}$, the above definition is equivalent to

$$\mathbb{E}(\mathbf{c}, \mathbf{P}) = \{\mathbf{x} \in \mathbb{R}^d | \, \|\mathbf{x} - \mathbf{c}\|_{\mathbf{P}} \leq 1\}, \tag{4.3}$$

which represents the ellipsoid as a unit ball about the center $\mathbf{c}$ in the corresponding ellipsoidal norm $\|\cdot\|_{\mathbf{P}}$. The volume of the ellipsoid $\mathbb{E}(\mathbf{c}, \mathbf{P})$ is given by

$$\text{vol}(\mathbb{E}(\mathbf{c}, \mathbf{P})) = \sqrt{\det \mathbf{P}} \, \frac{\pi^{d/2}}{\Gamma(d/2 + 1)}, \tag{4.4}$$

where $\Gamma(\cdot)$ is the gamma function. The volume of the ellipsoid mainly depends on the determinant of the matrix $\mathbf{P}$. Using this definition of ellipsoid, the Löwner-John ellipsoid containing the points $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_p \in \mathbb{R}^d$ can be computed by solving the

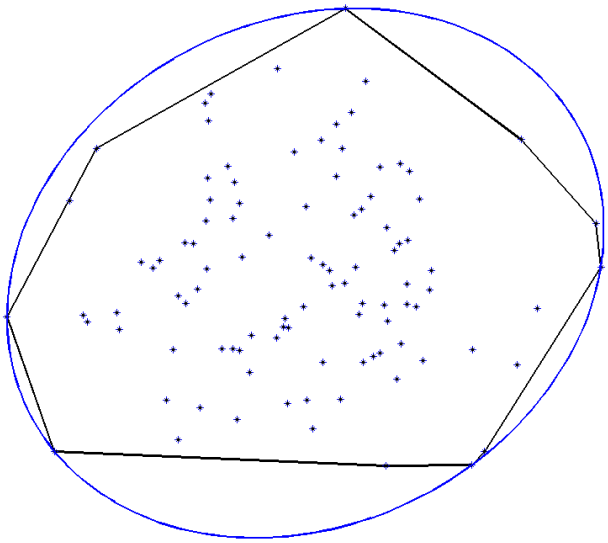

Figure 4.2: Löwner-John ellipsoid.

following optimization problem:

$$\begin{aligned} \text{minimize} \quad & \sqrt{\det \mathbf{P}} \\ \text{subject to} \quad & (\mathbf{x}_i - \mathbf{c})^T \mathbf{P}^{-1} (\mathbf{x}_i - \mathbf{c}) \leq 1, \; i = 1, ..., p \\ & \mathbf{P} \succ 0, \end{aligned} \tag{4.5}$$

where $\mathbf{c} \in \mathbb{R}^d$ and $\mathbf{P} \in \mathbb{R}^{d \times d}$ are the variables. The above problem (4.5) is a nonconvex one. To convert this into a convex problem, we need to parametrize the ellipsoid as

$$\mathbb{E}(\mathbf{A}, \mathbf{b}) = \{\mathbf{x} \in \mathbb{R}^d | \, \|\mathbf{A}\mathbf{x} + \mathbf{b}\|_2 \leq 1\}, \tag{4.6}$$

which is the inverse image of an Euclidean unit ball under affine mapping [23]. Here $\mathbf{A} \in \mathbb{R}^{d \times d}$ is a positive definite matrix, and the volume of the ellipsoid is now proportional to the determinant of $\mathbf{A}^{-1}$. The new ellipsoid formulation (4.6) can always be transformed back into the original formulation (4.2) using the following change of

variables:

$$\begin{aligned} \mathbf{c} &= -\mathbf{A}^{-1}\mathbf{b}, \\ \mathbf{P} &= \mathbf{A}^{-2}. \end{aligned} \tag{4.7}$$

Using this new definition of ellipsoid, computing the Löwner-John ellipsoid containing the points $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_p \in \mathbb{R}^d$ can be formulated as

$$\begin{aligned} &\text{minimize} && \log\det \mathbf{A}^{-1} \\ &\text{subject to} && \|\mathbf{A}\mathbf{x}_i + \mathbf{b}\|_2 \leq 1,\ i = 1, ..., p \end{aligned} \tag{4.8}$$

where $\mathbf{A} \in \mathbb{R}^{d\times d}$ and $\mathbf{b} \in \mathbb{R}^d$ are the variables. The above problem (4.8) is a convex optimization problem with the implicit constraint $\mathbf{A} \succ 0$. The convex optimization problem (4.8) can be efficiently solved by the available interior-point methods.

To find the clusters present in the data, we use the nonparametric Bayesian clustering method which does not require any preset sensitive parameters such as the number of clusters. Let $\mathbf{X} = \{\mathbf{x}_1, ..., \mathbf{x}_N\}$, $\mathbf{x}_n \in \mathbb{R}^d$ be the training data of one class. First, we model this training data $\mathbf{X} = \{\mathbf{x}_1, ..., \mathbf{x}_N\}$, $\mathbf{x}_n \in \mathbb{R}^d$, by a mixture of K Gaussian components (clusters) given by

$$\begin{aligned} p(\mathbf{x}_n|\boldsymbol{\pi}, \boldsymbol{\mu}, \mathbf{P}) &= \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}_n|\mu_k, P_k^{-1}), \\ p(\mathbf{X}|\boldsymbol{\pi}, \boldsymbol{\mu}, \mathbf{P}) &= \prod_{n=1}^{N} p(\mathbf{x}_n) = \prod_{n=1}^{N}\sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}_n|\mu_k, P_k^{-1}), \end{aligned} \tag{4.9}$$

where $P_k = \Sigma_k^{-1}$ is the precision matrix, $\pi_k (k = 1, ..., K)$ are the mixing parameters which must be positive and sum to 1.

We now introduce a latent random variable z_n corresponding to each data point $\mathbf{x}_n$. The variable z_n can take any of the K discrete values $\{1, ..., K\}$ and thus indicates the Gaussian distribution (cluster) to which the data point $\mathbf{x}_n$ belongs. We reformulate the above mixture model (4.9) using these latent variables. The distribution over z_n

is given by

$$\begin{aligned} p(z_n = k) &= \pi_k, \\ p(z_n) &= \prod_{k=1}^{K} {\pi_k}^{\mathbf{1}(z_n=k)}. \end{aligned} \tag{4.10}$$

The conditional distribution of $\mathbf{x}_n$ given z_n is given by

$$\begin{aligned} p(\mathbf{x}_n|z_n = k) &= \mathcal{N}(\mathbf{x}_n|\mu_k, P_k^{-1}), \\ p(\mathbf{x}_n|z_n) &= \prod_{k=1}^{K} \mathcal{N}(\mathbf{x}_n|\mu_k, P_k^{-1})^{\mathbf{1}(z_n=k)}. \end{aligned} \tag{4.11}$$

Calculating the marginal distribution of $\mathbf{x}_n$ from (4.10) and (4.11) gives the original mixture model (4.9). Considering the whole dataset $\mathbf{X} = \{\mathbf{x}_1, ..., \mathbf{x}_N\}$, we have the latent variables $\mathbf{z} = \{z_1, ..., z_N\}$. We now have

$$p(\mathbf{z}|\boldsymbol{\pi}) = \prod_{n=1}^{N} \prod_{k=1}^{K} {\pi_k}^{\mathbf{1}(z_n,k)}, \tag{4.12a}$$

$$p(\mathbf{X}|\mathbf{z}, \boldsymbol{\mu}, \mathbf{P}) = \prod_{n=1}^{N} \prod_{k=1}^{K} \mathcal{N}(\mathbf{x}_n|\mu_k, P_k^{-1})^{\mathbf{1}(z_n=k)}. \tag{4.12b}$$

For mathematical convenience, we place conjugate priors over the parameters $\boldsymbol{\pi}$, $\boldsymbol{\mu}$ and $\mathbf{P}$.

$$\begin{aligned} p(\boldsymbol{\pi}) = p(\pi_1, ..., \pi_K) &\sim \text{Dirichlet}\left(\frac{\alpha_0}{K}, ..., \frac{\alpha_0}{K}\right), \\ p(P_k) &\sim \mathcal{W}(m_1, V), \\ p(\mu_k) &\sim \mathcal{N}(m_2, R^{-1}), \end{aligned} \tag{4.13}$$

where $\mathcal{W}(m_1, V)$ is the Wishart distribution and $\{\alpha_0, m_1, V, m_2, R\}$ are the hyperparameters.

The conditional posteriors for the parameters can be obtained from the priors in (4.13) and the likelihood in (4.9). The conditional posterior for the means is given by

$$p(\mu_k|\mathbf{X}, \mathbf{z}, P_k, m_2, R) \sim \mathcal{N}\left(\frac{\overline{\mathbf{x}}_k n_k P_k + m_2 R}{n_k P_k + R}, \frac{1}{n_k P_k + R}\right), \tag{4.14}$$

where n_k is the number of data points that belong to component (cluster) k, $\overline{\mathbf{x}}_k$ is the mean of these points [24]. Similarly, the conditional posterior for the precisions

is given by

$$p(P_k|\mathbf{X}, \mathbf{z}, \mu_k, m_1, V) \sim \mathcal{W}\left(m_1 + n_k, (m_1 + n_k)\left[m_1 V + \sum_{n|z_n=k} (\mathbf{x}_n - \mu_k)^T(\mathbf{x}_n - \mu_k)\right]^{-1}\right). \tag{4.15}$$

The mixing parameters $\pi_k (k = 1, ..., K)$ have the symmetric Dirichlet prior with parameter α_0/K.

$$p(\boldsymbol{\pi}) = p(\pi_1, ..., \pi_K) \sim \text{Dirichlet}\left(\frac{\alpha_0}{K}, ..., \frac{\alpha_0}{K}\right) = \frac{\Gamma(\alpha_0)}{\Gamma(\alpha_0/K)^K} \prod_{k=1}^{K} \pi_k^{\alpha_0/(K-1)}. \tag{4.16}$$

Given the mixing parameters $\pi_k (k = 1, ..., K)$, the latent variables $\mathbf{z}$ have the multinomial distribution.

$$\begin{aligned} p(\mathbf{z}|\boldsymbol{\pi}) &= \prod_{n=1}^{N} \prod_{k=1}^{K} {\pi_k}^{\mathbf{1}(z_n=k)}, \\ p(\mathbf{z}|\boldsymbol{\pi}) &= \prod_{k=1}^{K} {\pi_k}^{n_k},\ n_k = \sum_{n=1}^{N} \mathbf{1}(z_n = k). \end{aligned} \tag{4.17}$$

Integrating out the mixing parameters using the Dirichlet integral, we get the prior on $\mathbf{z}$ directly as

$$p(\mathbf{z}) = p(\mathbf{z}|\alpha_0) = \frac{\Gamma(\alpha_0)}{\Gamma(N + \alpha_0)} \prod_{k=1}^{K} \frac{\Gamma(n_k + \alpha_0/K)}{\Gamma(\alpha_0/K)}. \tag{4.18}$$

Writing the conditional prior (suitable for Gibbs sampling), we have

$$P(z_n = k|\mathbf{z}_{-n}, \alpha_0) = \frac{n_{-n,k} + \alpha_0/K}{N - 1 + \alpha_0}, \tag{4.19}$$

where subscript $-n$ denotes all indices except n and $n_{-n,k}$ is the number of data points (excluding $\mathbf{x}_n$) associated with the component (cluster) k. As we do not know the number of clusters (K) beforehand, we take the limit $K \to \infty$ on the conditional prior in (4.19). This yields the following conditional prior:

$$P(z_n = k|\mathbf{z}_{-n}, \alpha_0) = \frac{n_{-n,k}}{N - 1 + \alpha_0}, \tag{4.20a}$$

$$P(z_n \neq z_m \forall m \in \{-n\}|\mathbf{z}_{-n}, \alpha_0) = \frac{\alpha_0}{N - 1 + \alpha_0}. \tag{4.20b}$$

This conditional prior is often interpreted as the Chinese Restaurant Process (CRP).

From the Bayes' theorem, the posterior of the latent variables $\mathbf{z}$ is given by

$$p(\mathbf{z}|\mathbf{X}) \propto p(\mathbf{X}|\mathbf{z})p(\mathbf{z}). \tag{4.21}$$

From the conditional priors in (4.20a)-(4.20b) and the likelihood in (4.12b), we get the conditional posteriors for the latent variables $\mathbf{z}$:

$$P(z_n = k|\mathbf{z}_{-n}, \mathbf{x}_n, \alpha_0, \mu_k, P_k) \propto \frac{n_{-n,k}}{N-1+\alpha_0}\mathcal{N}(\mathbf{x}_n|\mu_k, P_k^{-1}), \tag{4.22a}$$

$$\begin{aligned} &P(z_n \neq z_m \forall m \in \{-n\}|\mathbf{z}_{-n}, \mathbf{x}_n, \alpha_0, m_1, V, m_2, R) \\ &\propto \frac{\alpha_0}{N-1+\alpha_0}\int p(\mathbf{x}_n|\mu_k, P_k)p(\mu_k, P_k|m_1, V, m_2, R)d\mu_k dP_k, \\ &\propto \frac{\alpha_0}{N-1+\alpha_0}\int \mathcal{N}(\mathbf{x}_n|\mu_k, P_k^{-1})p(\mu_k|m_2, R)p(P_k|m_1, V)d\mu_k dP_k. \end{aligned} \tag{4.22b}$$

We use a Gibbs sampling method [25] and repeatedly sample from the posterior (4.22a)-(4.22b):

- For all k, sample μ_k and P_k according to the equations (4.14)-(4.15). Calculate the RHS of (4.22a).
- Draw 50 samples of μ_k and P_k from their priors in (4.13) to approximately calculate the integral on the RHS of (4.22b). And then calculate the RHS of (4.22b).
- Draw a sample for $\mathbf{z}$ from the multinomial distribution with the above calculated event probabilities.
- Remove a component (cluster) when it becomes empty. A new component (cluster) is added when a new unrepresented component from (4.22b) is chosen.
- We initially start with a single cluster and preset hyperparameters.
- Stopping criterion: Stop when the components are neither removed nor added for five consecutive iterations.

The convergence and the accuracy of the above sampling method depends mainly on the hyperparameter α_0. If α_0 is too low, we may get less number clusters. On the other hand, if it is too high, we may get more number of clusters. We empirically set $\alpha_0 = \frac{N}{100}$. The remaining hyperparameters are set according to the hyperpriors suggested in [24] [1]. After the convergence, the components obtained are considered to be the clusters present in the data. The Löwner-John ellipsoid for each cluster is then calculated by (4.8). This clustering process is done for the data of each class separately.

4.2.2 Training Data Elimination

With the above clustering algorithm, every cluster in the training data is accurately modeled by its Löwner-John ellipsoid. We now make use of the fact that the samples which are close to the decision boundary are more informative than the ones which are far away. The relative informativeness of a sample is determined by the following two factors:

1. Relative position of the sample in the cluster.

2. Relative position of the sample with respect to the other class data.

In other words, the samples which are well inside the clusters and far away from the other class data are considered to be least informative and hence can be eliminated. All such samples which are considered to be least informative among each cluster are eliminated from the training set. Only the most informative samples stay in the training set. The probability that a sample $\mathbf{x}_n$ stays in the training set is given by a

[1] The presetting of these parameters has minimal effect on the final SVM classification performance.

logistic regression model:

$$P_{stay}(\mathbf{x}_n) = \frac{e^G}{1+e^G},$$

where

$$\begin{aligned} G &= \alpha_1(2v_1 - 1) + \alpha_2(2v_2 - 1), \\ v_1 &= \|\mathbf{x}_n - \mathbf{c}\|_{\mathbf{P}}, \\ v_2 &= \frac{\left\|\sum_{j=1}^{k_{oth}} \mathbf{z}_{ij}\right\|}{\|\mathbf{z}_{max}\|}, \end{aligned} \tag{4.23}$$

where α_1 and α_2 are the parameters of the logistic regression model, $\mathbf{c}$ and $\|\cdot\|_{\mathbf{P}}$ are respectively the center and the ellipsoidal norm of the cluster to which the sample $\mathbf{x}_n$ belongs, k_{oth} is the number of other class clusters, $\mathbf{z}_{ij}$ is the force vector which represents the closeness of the sample $\mathbf{x}_n$ to the other class cluster j, $\mathbf{z}_{max}$ is the maximum force vector of all the samples within the cluster to which the sample $\mathbf{x}_n$ belongs. The force vector between the sample $\mathbf{x}_n$ and the cluster j is given by

$$\mathbf{z}_{ij} = \frac{\mathbf{c}_j - \mathbf{x}_n}{\|\mathbf{c}_j - \mathbf{x}_n\|_2^2}, \tag{4.24}$$

where $\mathbf{c}_j$ is the center of the cluster j. The numerator of v_2 gives the magnitude of the resultant force between the sample $\mathbf{x}_i$ and the other class cluster centers, and the denominator is a normalization factor. Infact the indepenent variables v_1 and v_2 in the logistic model are measures of the aforementioned factors. The above defined variables vary in the interval [0,1]. So the logistic model using the above variables directly will not span the entire probability space of [0,1]. Hence we first apply the affine transformation $(\cdot) \mapsto 2(\cdot) - 1$ to the variables.

The logistic model parameters are empirically chosen to be $\alpha_1 = 10$ and $\alpha_2 = 20$. The values of these parameters govern the final probability of stay (P_{stay}) values. As we intend to make hard decisions about the sample selection, the probability of stay (P_{stay}) values need to be either 0 or 1. Hence, relatively higher values were given to the parameters. The corresponding sigmoid curve is shown in Figure 4.3. The values

of these parameters can be adjusted to make the sigmoid curve more smoother, and more probabilistic decisions can be made about the sample selection as required. Also, we gave relatively more importance to the second variable v_2 which measures the closeness of the sample to the other class data. We retain all the training samples whose probability of stay (P_{stay}) is equal to 1. All the other samples are eliminated from the training set. Finally the support vector machine is trained using this new training set which contains only the most informative training samples.

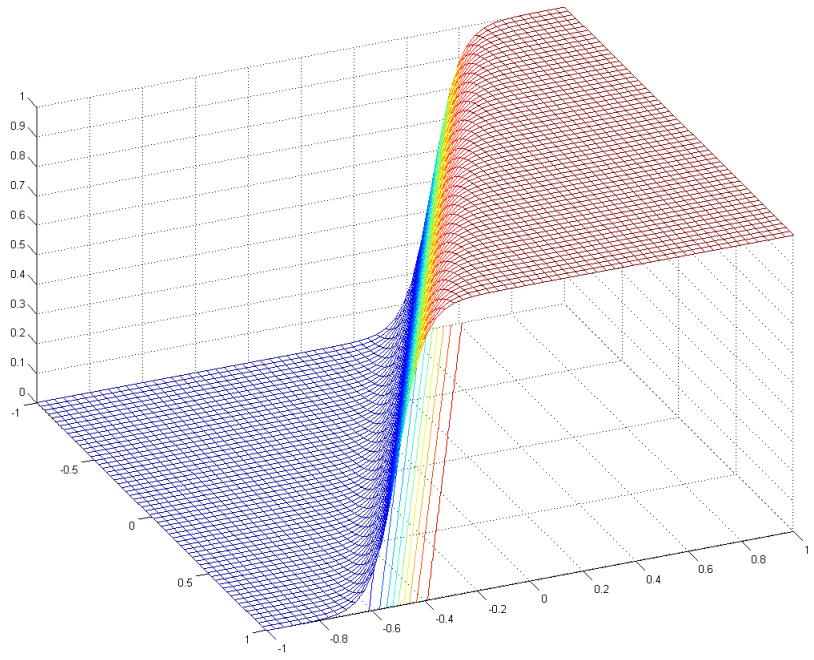

Figure 4.3: Logistic sigmoid curve of the training sample elimination model.

4.3 Complexity and Performance Analysis

Computing the Löwner-John ellipsoid is the key to the proposed training sample selection method. The Löwner-John ellipsoid for each cluster in the data is computed by solving the convex optimization problem (4.8). The problem (4.8) is a second-order cone programming (SOCP) problem which can be efficiently solved using the available

interior-point methods. In this work, we have used the self-dual minimization method whose complexity is about $O(\sqrt{d}p)$, where p is the number of points and d is the dimension [26]. However, the Löwner-John ellipsoid is usually determined by at most $\frac{d^2+3d}{2}$ points out of the p points [27]. Thus, the complexity of computing the Löwner-John ellipsoid can be further reduced by intelligently selecting these points. For this purpose, we use the active-set strategy based on Sample Covariance Initialization (SCI) proposed by [28]:

1. Define an initial active set $\mathbf{X}_a^0 = \{\mathbf{x}_1, ..., \mathbf{x}_L\}$ such that the affine hull of $\mathbf{x}_1, ..., \mathbf{x}_L$ spans the space $\mathbb{R}^d$. The Sample Covariance Initialization (SCI) scheme in [28] is used.

2. *(ith iteration)* Solve (4.8) for the active set $\mathbf{X}_a^i$. Let $(\mathbf{c}^i, \mathbf{P}^i)$ be the solution.

3. If $d_n^i = \|\mathbf{x}_n - \mathbf{c}^i\|_{\mathbf{P}^i} \leq 1$ for $n \in \mathbf{X} \backslash \mathbf{X}_a^i$, stop. Otherwise, proceed to the next step.

4. Update the active set to $\mathbf{X}_a^{i+1}$:

   - *(Adding points to the active set)* $\mathbf{X}_a^{i+1} = \mathbf{X}_a^i \cup \Delta\mathbf{X}_a$. We intend to add the points $\mathbf{x}_n \notin \mathbf{X}_a^i$ whose distance from current center $\mathbf{c}^i$ in the ellipsoidal norm $\|\cdot\|_{\mathbf{P}^i}$ $(d_n^i) \geq 1$ and largest. To further reduce the redundancy, we intend to add the points that are well spread around the current ellipsoid $\mathbb{E}(\mathbf{c}^i, \mathbf{P}^i)$. For this purpose, we gradually consider the points in the descending order of d_n^i, and add a point $\mathbf{x}_n$ to $\Delta\mathbf{X}_a$ if

   $$\sum_{j \in \Delta\mathbf{X}_a} (\mathbf{x}_n - \mathbf{c}^i)^T P^{i-1} (\mathbf{x}_j - \mathbf{c}^i) < 0. \tag{4.25}$$

   - *(Removing the points that are no longer necessary)* We delete all the points $\mathbf{x}_n$ whose $d_n^i < 0.99$.

   Go to step 2.

Using this active-set strategy, the complexity in computing the Löwner-John ellipsoid is greatly reduced.

For the analysis purpose, the proposed fast training method is divided into the following four phases where most of the computation happens:

1. Clustering the data.
2. Computing the Löwner-John ellipsoids for each cluster.
3. Training sample elimination.
4. Training the SVM using the new training set.

For the nonparametric Bayesian clustering approach, we first evaluate the complexity of each iteration. The total amount of computation involved in each iteration can be typically divided into the following parts:

- Sampling the mean vectors and precision matrices from normal and Wishart distributions respectively.
- Calculating the posterior probabilities for each sample.
- Sampling the latent variables $\mathbf{z}$ from the multinomial distribution.

The complexity of sampling from normal and Wishart distributions is on the order of $O(d)$, where d is the dimension of the data. The complexity of calculating the posterior probabilities for n samples is $O(n)$. The complexity of sampling the latent variables from multinomial distribution is $O(n)$. Thus the complexity of each iteration in the nonparametric Bayesian clustering is $O(n)$. Generally, the total number of iterations is much smaller than the size of the training data. Thus, the overall complexity of the nonparametric Bayesian clustering approach can be approximated to $O(n)$. After clustering, computing the Löwner-John ellipsoids for the clusters has the complexity $O(n)$ at most. The sample elimination approach involves calculating the probability

of stay for each sample, and hence has the complexity $O(n)$. After reducing the training set to m ($m \ll n$) training samples, the SVM is trained using this new training set. The complexity of this phase is $O(m^2)$, as we use the SMO method. Thus the total complexity of our SVM training algorithm is at most

$$O(n) + O(n) + O(n) + O(m^2), \tag{4.26}$$

whereas the complexity of conventional SVM training (using the SMO method) is $O(n^2)$. With $m \ll n$, this clearly demonstrates the reduction in the computational complexity achieved by using the proposed fast training approach.

Our main objective in designing the fast training approach is to decrease the computational complexity of SVM training without having huge degradation in the classification performance of the SVM. This is clearly taken care of in the design of our fast training approach, where we retain only the most relevant samples that are likely to become support vectors, thus greatly reducing the size of the training set and the SVM training complexity.

CHAPTER 5

FUSION RULES BASED ON DEMPSTER-SHAFER THEORY

In this chapter, we present the data fusion rules which are used at each sensor and the fusion center of the proposed cyber attack detection system. The fusion rules are designed using the Dempster-Shafer theory of evidence. The Dempster-Shafer theory can be considered as a generalized version of the traditional probability theory. The Dempster-Shafer theory is a result of the work by Arthur P. Dempster [29] and Glenn Shafer [30].

The Dempster-Shafer theory is both a theory of evidence and a theory of probable reasoning. It exactly quantifies the amount of evidence available from a source. Also, the Dempster-Shafer theory effectively combines evidence from different sources and defines the degree of belief based on the total evidence available from different sources.

Probability theory is the most widely used mathematical framework to represent uncertainty. Generally, uncertainty can be of two types:

1. **Aleatoric Uncertainty (Objective or Stochastic Uncertainty):** It is the uncertainty due to the random behavior of a system. This type of uncertainty can be described using the idea of chances in general.

2. **Epistemic Uncertainty (Subjective Uncertainty):** It is the uncertainty due to the lack of information or knowledge about a system. In other words, it is the uncertainty due to ignorance. Generally, it can be described using the idea of beliefs.

The traditional probability theory is based on the theory of chance, which is generally

used to describe the aleatoric uncertainties. The probability theory, when applied in aleatoric situations, is generally referred to as the objective (or frequency) probability theory. Hence, the traditional probability theory can handle the aleatoric uncertainty very well. However, the traditional probability theory has been directly extended to characterize the epistemic uncertainty as well (through the Bayesian theory). In the Bayesian theory of probability, the probability is defined and interpreted as the degree of belief in a particular proposition (or hypothesis) on the basis of the available evidence. Thus, the probability $P(A)$ represents the degree of belief in the proposition A based on the available evidence. And the probability distribution (probability density function or probability mass function) is used to represent the amount of available evidence.

The (Bayesian) probability theory cannot accurately characterize the epistemic uncertainty. The basic idea in the Bayesian theory that the degrees of belief always tend to be like the objective probabilities (chances) in their mathematical structure, is not true in general. The degrees of belief do not, in general, have the same mathematical properties of objective probabilities (chances). However, the Bayesian theory explicitly forces that the degrees of belief (represented by the Bayesian probabilities) must obey all the mathematical rules of the chances (objective probabilities). The Bayesian theory directly adopts the three basic axioms of (objective) probability to the Bayesian probability (which are the degrees of belief in the Bayesian theory). This makes the Bayesian theory too restrictive and less flexible in modeling epistemic uncertainty (or ignorance). The mathematical rule that makes the Bayesian probability theory too restrictive is the third axiom of probability:

- **Additivity axiom of probability:** The probability of the union of mutually exclusive (disjoint) events must be equal to the sum of the probabilities of the individual events. Let $E_1, E_2, ..., E_N$ be the mutually exclusive events of the sample space Ω. Then, according to the additivity axiom of the probability

theory,

$$P(E_1 \cup E_2 \cup ... \cup E_N) = P(E_1) + P(E_2) + ... + P(E_N) = \sum_{i=1}^{N} P(E_i). \quad (5.1)$$

This rule is not appropriate for modeling the epistemic uncertainty. Due to this assumption, the Bayesian probability theory cannot exactly represent the ignorance. For example, let $E \in \Omega$ represent a proposition and the complement $\overline{E} \in \Omega$ represent the negation of the proposition. Since $E \cup \overline{E} = \Omega$, from the above rule of additivity, the Bayesian probabilities which represent the degrees of belief must satisfy condition: $P(E) + P(\overline{E}) = 1$. This implies that the $P(E)$ cannot be low unless the $P(\overline{E})$ is sufficiently high. Thus, the main difficulty of the Bayesian probability theory is that it cannot distinguish between the lack of belief and disbelief. The Bayesian probability theory does not allow to withhold belief from a hypothesis without giving it to the negation of the hypothesis. This leads to the Laplace's principle of insufficient reason in the Bayesian theory, which restricts that the complete ignorance is always modeled by the uniform probability distribution. Hence the (Bayesian) probability theory cannot accurately characterize the epistemic uncertainty. A more general and flexible mathematical framework is required to accurately model the epistemic uncertainty (or ignorance).

The Dempster-Shafer theory is a more generalized mathematical framework that can be applied to accurately characterize the epistemic uncertainty. In a finite discrete space, the Dempster-Shafer theory can be considered as a generalization of the probability theory, where the degree of beliefs (represented by probabilities in the probability theory) are assigned to sets of events besides individual events. The (Bayesian) probability theory is a special case of the Dempster-Shafer theory. The greater flexibility of the Dempster-Shafer theory is valuable and essential for an adequate representation of evidence and probable reasoning [30].

5.1 Basics of Dempster-Shafer Theory

Let $\Theta = \{\theta_1, \theta_2, ..., \theta_N\}$ be the set of all possible hypotheses (or propositions). The set Θ is called the *frame of discernment*. Let 2^Θ be the power set of Θ, i.e., the set of all possible subsets of Θ. A real function $m : 2^\Theta \mapsto [0, 1]$ is called a *basic probability assignment (BPA)* if

$$\begin{aligned} m(\emptyset) &= 0, \\ \text{and } \sum_{A \in 2^\Theta} m(A) &= 1. \end{aligned} \tag{5.2}$$

The basic probability assignment (BPA) function is sometimes referred to as the *mass function*. The quantity $m(A)$ is called the *basic probability number* of A (or sometimes called the *mass number*).

The basic probability number of A can be interpreted as the measure of belief that is committed exactly to A. The above conditions reflect the facts that no belief should be committed to the empty set $\emptyset$ and the total belief must be equal to 1. For a given set A, the basic probability number $m(A)$ represents belief that the true hypothesis lies in the set A. In other words, $m(A)$ represents the proportion of the total available evidence that supports the claim that the true hypothesis lies in the set A but not in any particular subset of A. Thus in other words, the BPA $m(\cdot)$ is a measure of evidence. For singletons $A = \{\theta_i\}$, the basic probability number $m(A)$ represents our confidence that the hypothesis θ_i is true. For non-singletons $A \neq \{\theta_i\}$, the basic probability number $m(A)$ represents our ignorance, as we are not exactly sure which hypothesis in the set A is actually true. The important thing to note is that the basic probability number $m(A)$ makes claims only about the set A. It does not make any additional claims about the subsets of A. Any additional evidence or information about the subsets of A should be represented by another BPA function.

For a given set A, $m(A) + m(\overline{A}) \leq 1$, where $\overline{A}$ is the complement of A. When the inequality holds, the amount of belief assigned neither to A nor to $\overline{A}$ represents the

degree of ignorance. In general, the BPA $m(\cdot)$ is not same as the classical probability. However in some special cases, the BPA $m(\cdot)$ can be equivalent to the classical probability. For example, for a given frame of discernment Θ, $m(A) \neq 0$ for all $A = \{\theta_i\}$ and $m(A) = 0$ for all $A \neq \{\theta_i\}$, the BPA $m(\cdot)$ is equivalent to the classical probability.

Given a BPA $m(\cdot)$, a real function $Bel : 2^{\Theta} \mapsto [0, 1]$ called a *belief function* is defined as follows:

$$\text{Belief of a set } B, \quad Bel(B) = \sum_{A \subseteq B} m(A). \tag{5.3}$$

The belief of a set B is defined as the sum of the basic probability numbers of all the subsets of B. Thus, the belief function $Bel(\cdot)$ represents the total belief that is committed to a particular hypothesis (or proposition). The BPA function $m(\cdot)$ represents the belief that is committed to a particular hypothesis (or proposition), not the total belief. The BPA $m(\cdot)$ can be considered as a generalization of the probability distribution (probability density function or probability mass function), whereas the belief function $Bel(\cdot)$ can be considered as a generalization of the probability function. For singletons $A = \{\theta_i\}$, $Bel(A) = m(A)$. Given a subset A of the frame of discernment Θ, it is called a *focal element* of a belief function if $m(A) > 0$. The union of all the focal elements of a belief function is called the *core* of the belief function.

Given a BPA $m(\cdot)$, a real function $Pl : 2^{\Theta} \mapsto [0, 1]$ called a *plausibility function* is defined as follows:

$$\text{Plausibility of a set } B, \quad Pl(B) = \sum_{A \cap B \neq \emptyset} m(A). \tag{5.4}$$

The plausibility of a set B is defined as the sum of all the basic probability numbers of the sets that intersect B. The plausibility function can be defined using the belief function as follows:

$$\text{Plausibility of a set } B, \quad Pl(B) = 1 - Bel(\overline{B}), \tag{5.5}$$

where $\overline{B}$ is the complement of B. This clearly implies that the amount of belief not assigned to $\overline{B}$ is not automatically assigned to B, like in the probability theory. This

belief (not assigned to $\overline{B}$) makes B more probable or plausible and is represented by the plausibility of B, $Pl(B)$.

The Dempster-Shafer framework uses these two measures, belief $Bel(\cdot)$ and plausibility $Pl(\cdot)$, to deal with the epistemic uncertainty. These two measures, belief $Bel(\cdot)$ and plausibility $Pl(\cdot)$, generally form the lower and upper bounds of the classical probability measure. That is, the classical probability of an event lies within the lower and upper bounds of belief and plausibility. Given a set A,

$$Bel(A) \leq P(A) \leq Pl(A). \tag{5.6}$$

For this reason, the belief and the plausibility values are sometimes called the *lower* and *upper probabilities*.

5.2 Dempster's Rule of Combination

Another main advantage in the Dempster-Shafer theory is the ability to combine the evidences from difference sources using the Dempster's rule of combination. Dempster's rule of combination provides a way for effectively changing our (prior) beliefs in the light of new evidence. The Dempster's rule of combination deals symmetrically with the new and the old evidence, unlike the Bayesian theory which represents the new evidence as a proposition and conditions the prior (Bayesian) belief on that proposition using the Bayes' rule of conditioning. There is no symmetry in dealing the new and the old evidence in the Bayesian theory. Also in the Bayesian theory, when combining new and old evidence, an assumption that the new evidence always establishes a single proposition with certainty is made. This further restricts the Bayesian theory in effectively combining different evidences. On the other hand, the Dempster's rule of combination in the Dempster-Shafer theory treats both the new and the old evidence equally, and provides a method to effectively combine them without making any restrictive assumptions.

Let $m_1(\cdot)$ and $m_2(\cdot)$ be the two basic probability assignments (BPAs) on the frame of discernment Θ corresponding to two independent sources of evidence. Then from the Dempster's rule of combination, we can fuse those two basic probability assignments (BPAs) into a single new BPA $m_{12}(\cdot)$ as

$$\begin{aligned} m_{12} &= m_1 \oplus m_2, \\ m_{12}(A) &= \frac{\sum_{B \cap C = A} m_1(B) m_2(C)}{1 - \sum_{B \cap C = \emptyset} m_1(B) m_2(C)}. \end{aligned} \tag{5.7}$$

The operator $\oplus$ represents the orthogonal sum.

The Dempster's rule of combination is illustrated geometrically in the following figures. Let $m_1(\cdot)$ be the basic probability assignment (BPA) over a frame of discernment Θ, and $m_2(\cdot)$ be another independent basic probability assignment (BPA) over the same frame of discernment Θ. Let $A_1, ..., A_k$ be the focal elements of $m_1(\cdot)$ and $B_1, ..., B_l$ be the focal elements of $m_2(\cdot)$. The corresponding BPA values of these focal elements can be depicted as segments of the line segment from 0 to 1, as shown in Figure 5.1. Figure 5.2 shows the unit square obtained by orthogonally combining the individual line segments m_1 and m_2. The unit square represents the total basic probability mass. The BPA $m_1(\cdot)$ commits the vertical stripes to its focal elements, and the BPA $m_2(\cdot)$ commits the horizontal stripes to its focal elements. Now the Figure 5.2 shows the total combined belief (the shaded rectangle) that is committed exactly to $A_i \cap B_j$, which is given by the quantity $m_1(A_i)m_2(B_j)$. Similarly the total combined belief of every rectangle in the Figure 5.2 can be calculated. Now a set A in general may have one or more than one of these rectangles in it, and hence the total combined probability mass that is exactly committed to A is given by $\sum_{B \cap C = A} m_1(B)m_2(C)$. Since some of the rectangles in the unit square may always lie in the empty set $\emptyset$, some belief is always assigned to the empty set $\emptyset$. In order to avoid this, all such rectangles are discarded and the basic probability masses of the remaining rectangles need to be increased by the factor $\left(1 - \sum_{B \cap C = \emptyset} m_1(B)m_2(C)\right)^{-1}$, so that the total probability mass will be equal to 1.

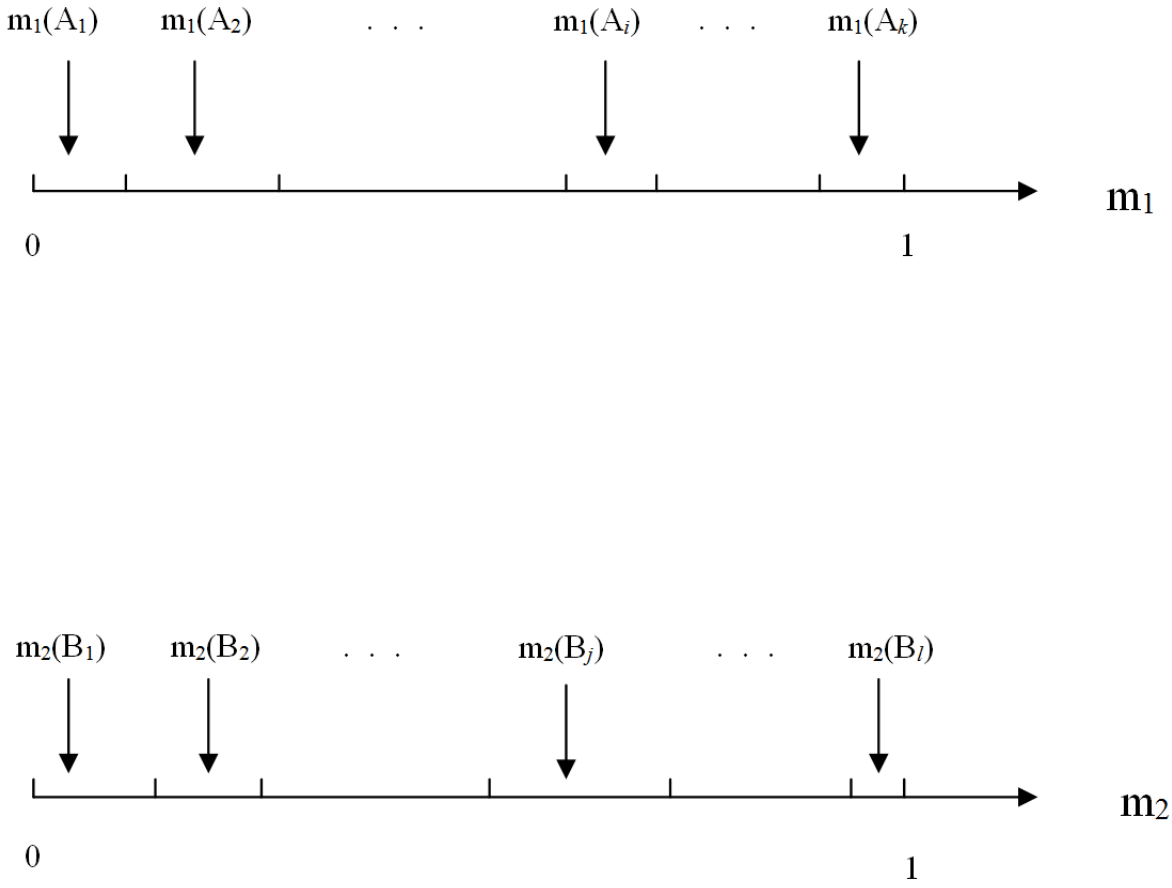

Figure 5.1: Basic probability assignments (BPAs) of two independent sources of evidence.

The Dempster's rule of combination (5.7) can be generalized to combine the basic probability assignments (BPAs) from multiple independent sources of evidence.

$$m = m_1 \oplus m_2 \oplus ... \oplus m_L. \tag{5.8}$$

5.3 Local Fusion Rule at Each Sensor

At each sensor, we have a binary classifier (SVM) trained for each class. For the ith SVM $g_i(\cdot)$ at a sensor, which decides whether a sample $\mathbf{x}$ belongs to class i or not, the output of the SVM is $y = \text{sgn}(f_i(\mathbf{x}))$, where $f_i(\mathbf{x})$ is the corresponding SVM decision function. Clearly this SVM function $f_i(\mathbf{x})$ is evidence available at the ith SVM $g_i(\cdot)$. Hence the BPA function can be defined based on this SVM function.

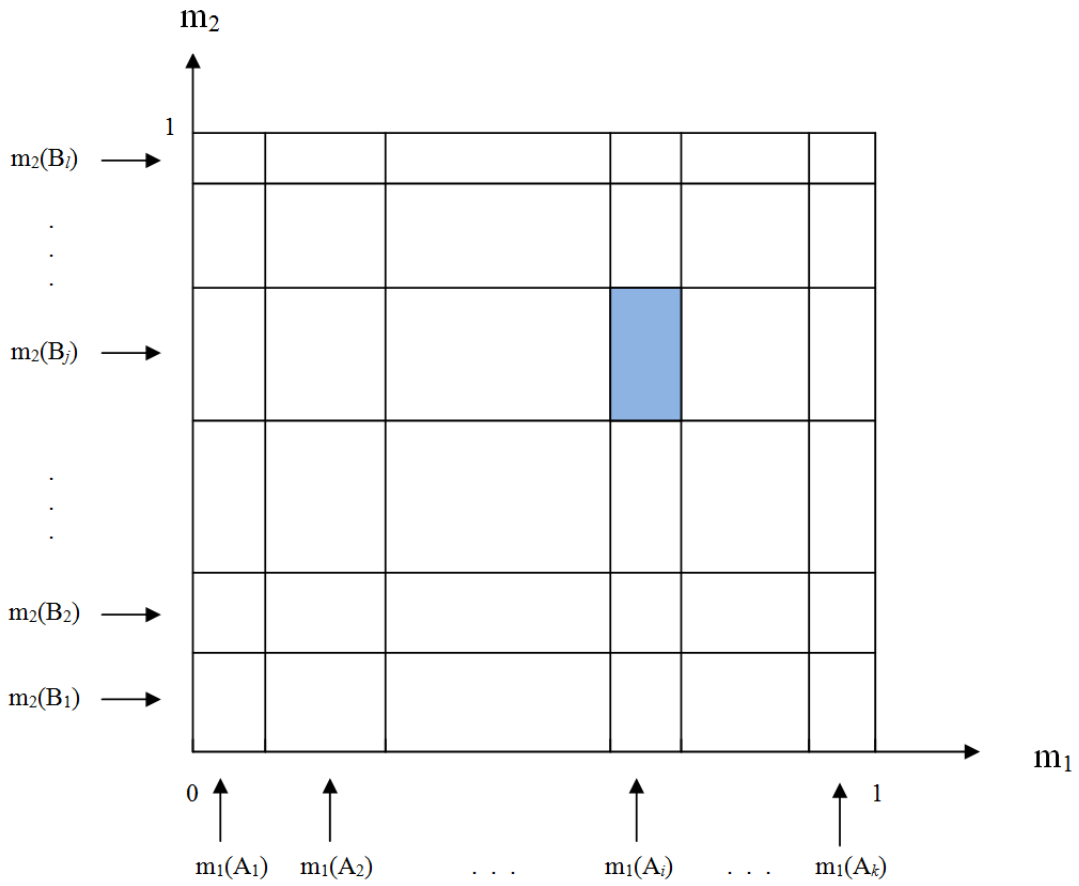

Figure 5.2: Dempster's rule of combination of basic probability assignments (BPAs) of two independent sources of evidence.

The BPA function $m_i(\cdot)$ for the ith SVM $g_i(\cdot)$ is defined as:

$$m_i(A) \begin{cases} = P(y=1|f_i(\mathbf{x})) = P_i, & \text{if } A = \{\theta_i\} \\ = 1 - P(y=1|f_i(\mathbf{x})) = 1 - P_i, & \text{if } A = \Theta \backslash \{\theta_i\} \\ = 0, & \text{otherwise} \end{cases} \tag{5.9}$$

where $P_i = P(y=1|f_i(\mathbf{x}))$ is the posterior probability of class i given the ith SVM output function $f_i(\mathbf{x})$. This probability represents the total amount of evidence that supports the hypothesis θ_i (sample $\mathbf{x}$ belongs to class i). All the remaining evidence does not support any other hypothesis in particular, and hence is assigned to all the remaining hypotheses $\Theta \backslash \{\theta_i\}$. The posterior probability $P(y=1|f_i(\mathbf{x}))$ for support

vector machines can be calculated in different ways. One such method of calculating the posterior probabilities is presented in Appendix A.

After calculating the posterior probabilities $P(y = 1|f_i(\mathbf{x}))$, the BPAs corresponding to the different SVMs at each sensor can be fused using the Dempster's rule of combination (5.8). The new combined BPA at each sensor $m(\cdot)$ is

$$m(A) \begin{cases} = \frac{(1-P_1)(1-P_2)...(1-P_{i-1})P_i(1-P_{i+1})...(1-P_L)}{R}, & \text{if } A = \{\theta_i\} \\ = 0, & \text{otherwise} \end{cases} \tag{5.10}$$

where $i = 1, ..., L$ and R is a normalization factor which satisfies the condition $\sum_{A \in 2^\Theta} m(A) = 1$. Based on this new BPA $m(\cdot)$, the belief function $Bel(\cdot)$ at each sensor can be defined as

$$Bel(A) = \frac{(1 - P_1)(1 - P_2)...(1 - P_{i-1})P_i(1 - P_{i+1})...(1 - P_L)}{R}, \text{ if } A = \{\theta_i\} \tag{5.11}$$

where $i = 1, ..., L$. For singletons, the belief function is same as the BPA.

Finally at each sensor, a sample $\mathbf{x}$ is classified into the class with the highest belief. The local decision at sensor S_j is given by

$$u_j = \arg\max_{i=1,...,L} Bel(\{\theta_i\}). \tag{5.12}$$

Clearly from (5.11), for $1 \leq i, j \leq L$, we have $Bel(\{\theta_i\}) \geq Bel(\{\theta_j\})$ iff $P_i \geq P_j$. Hence the local decision at sensor S_j is given by

$$u_j = \arg\max_{i=1,...,L} Bel(\{\theta_i\}) = \arg\max_{i=1,...,L} P_i. \tag{5.13}$$

5.4 Global Fusion Rule at the Fusion Center

Each sensor transmits its local decision u_j and the corresponding posterior probability P_i to the fusion center. Based on the available local decisions, the fusion center takes the final (global) decision regarding the state of the network. The fusion center takes

the final decision u_0 using the majority voting rule. The final decision u_0 at the fusion center, given all the local decisions $\{u_j, j = 1, ..., M\}$, is given by

$$\begin{aligned} & u_0 = u_l, \\ & \text{where} \\ & l = \arg\max_{j=1,...,M} \sum_{i=1}^{M} \mathbf{1}(u_j = u_i), \text{ and } i \neq j, \end{aligned} \tag{5.14}$$

and $\mathbf{1}(H) = 1$ when condition H is true and $\mathbf{1}(H) = 0$ when condition H is false.

In case of tie between the sensors, the final decision is made based on the corresponding posterior probability value. This posterior probability value plays the role of a confidence score of the corresponding local decision. In case of tie between the sensors, the final decision u_0 at the fusion center is given by

$$\begin{aligned} & u_0 = u_l, \\ & \text{where} \\ & l = \arg\max_{j} \ P_j, \end{aligned} \tag{5.15}$$

and P_j is the posterior probability corresponding to the local decision u_j.

CHAPTER 6

EXPERIMENTAL RESULTS

In this chapter, the proposed distributed cyber attack detection system is evaluated on the widely used 1999 KDD intrusion detection dataset [31]. The results of both the training phase and the decision making phase are presented.

6.1 Cyber Attacks Dataset

The 1999 KDD intrusion detection dataset [31] is the popular publicly available cyber attacks dataset. The 1999 KDD intrusion detection dataset is a version of the 1998 DARPA Intrusion Detection Evaluation Program data, which is developed by MIT Lincoln Laboratory. In the 1998 DARPA Intrusion Detection Evaluation Program by the MIT Lincoln Laboratory, the normal and attack data was recreated on a private network using real hosts, live attacks, and live background traffic.

The testbed used by the MIT Lincoln Laboratory is shown in Figure 6.1 [32]. The testbed simulates network traffic similar to the one seen between a Air Force base and the Internet. Custom software is used to simulate hundreds of different types of users like programmers, managers, workers, system administrators, attackers, etc., running the common UNIX applications. The user operations include typical day-to-day operations like sending and receiving email, browsing the Web, sending and receiving files using FTP, remote logging in using Telnet to work, chatting using Internet Relay Chat (IRC), etc. The custom software also allows the small number of hosts present to appear as thousands of hosts with different IP addresses. Large amount of network traffic is generated using different network services and protocols.

The total proportion and variability in the generated network traffic are similar to that of a typical Air Force base. All the attacks are launched from the outside of the simulated Air Force base.

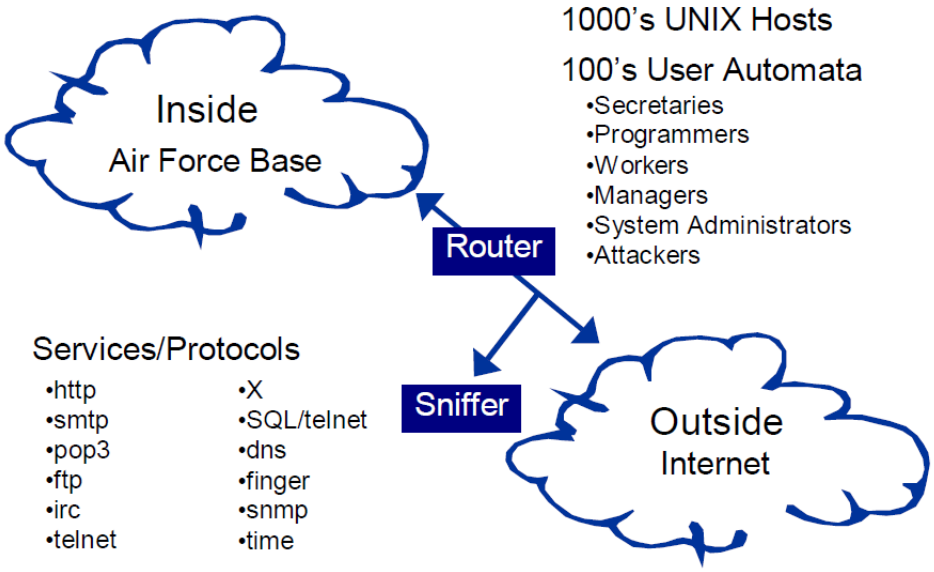

Figure 6.1: The 1998 DARPA intrusion detection evaluation testbed.

In Figure 6.2 [32], a more detailed block diagram of the testbed is shown. The simulated Air Force base contains three UNIX machines (Linux 2.0.27, SunOS 4.1.4, Sun Solaris 2.5.1) which are the frequent victims of the attacks, and a gateway to hundreds of other computers and workstations. The outside part shown in the figure contains several workstations and Web servers, which simulate the Internet. A sniffer is used to capture the traffic. The total data used in the evaluation program is collected from the following three sources:

1. Network sniffing data from the sniffer.

2. Sun Solaris Basic Security Module (BSM) audit data from the Solaris host.

3. Disk dump data from the three UNIX victim machines.

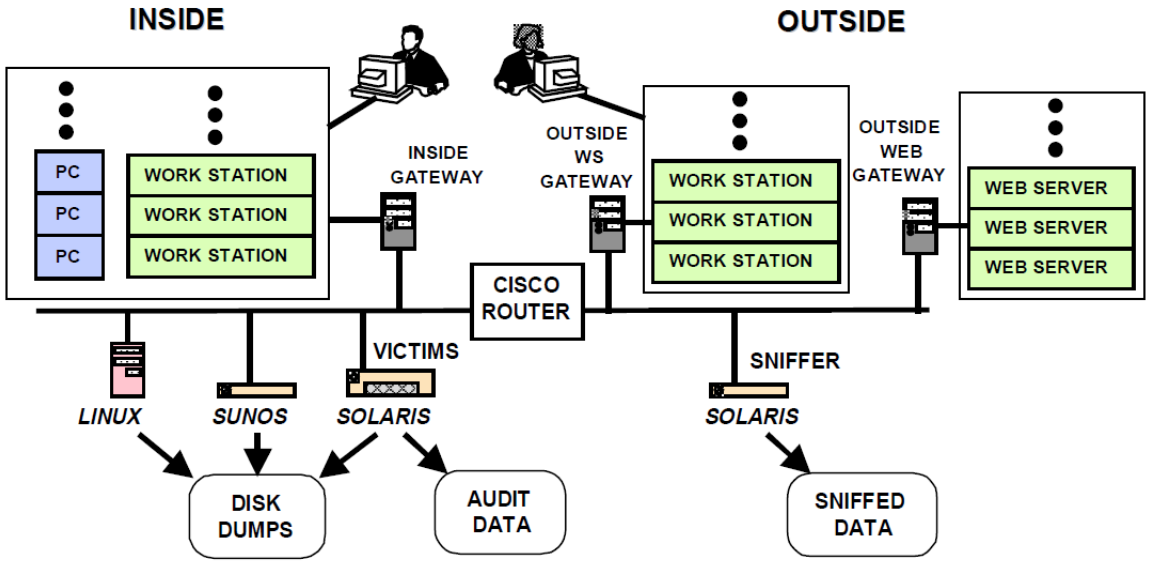

Figure 6.2: Detailed block diagram of the 1998 DARPA evaluation testbed.

A large amount of background traffic is generated between the inside PCs and workstations and the outside workstations and Web servers. A lot of user operations are performed on the three victim machines inside from the outside using various network protocols like Telnet. The three gateway machines used contain some operating system kernel modifications, which together with the custom software allows the small number of hosts present to appear as thousands of hosts with different IP addresses. The contents of the network traffic for some services like SMTP, FTP, and HTTP are ensured to be statistically similar to that of a live network traffic. For instance, the email message contents are created using the statistical bigrams frequencies preserving the one-word and two-word sequence statistics from a sample of 10,000 actual email messages from different computer professionals, which are initially filtered using a 40,000 word dictionary to remove names and other sensitive private information. Some of other email messages are the actual messages which are taken from a variety of public domain mail list servers. The FTP file transfers content is also generated using the similar approaches.

The content of the web servers is generated using a custom web automaton run on the real Internet. The custom web automaton used is initially programmed to visit several thousands of websites with a frequency proportional to the website's popularity, and to visit some random number of links at each website before moving to another website. Thus, it generated a large database of public domain website content, which is then used in the testbed. As the testbed is not connected to the Internet for security reasons, this automaton process ensures that the outside part of the testbed simulates the real Internet. During the experiments, the browsing automata accessed different web pages on the web servers through the outside web gateway. Again, custom software is used in the outside web gateway to emulate thousands of websites. Also, custom software is used to generate user automata which simulates some simple scenarios like the one in which there are different users typing at the keyboards. Actual humans performed more complex tasks like upgrading the software, changing passwords, remotely accessing the programs, some system administration tasks, sending and receiving email, etc.

A total of 32 different cyber attacks were generated in the evaluation program. All the attacks fall into the following four categories:

1. Denial of Service (DoS) attacks
2. Remote to Local (R2L) attacks
3. User to Root (U2R) attacks
4. Probe attacks

The generation of the attacks is a complex process. First the attack mechanism was studied and analyzed, and then a working attack is developed on the testbed. The working attack developed is further analyzed and tuned to determine the software and/or services required to run the attack. This analysis is also used to develop

new stealthy versions of the attack. Novel attacks were generated using the network weaknesses in the developed testbed. The novel (unseen) attacks were used only in the test data, to facilitate the evaluation of the cyber attack detection system's ability in detecting the new never-before-seen attacks.

A total of nine weeks of raw data is collected and used as the training data. The raw training data was about 4 GB of compressed binary TCP dump data of network traffic. This raw data is finally processed into five million connection records. Similarly, two weeks of data is collected and used as the test data. The final processed test data contains around two million connection records. A connection is defined as "a sequence of TCP packets starting and ending at some well defined times, between which data flows to and from a source IP address to a target IP address under some well defined protocol." [31] Each connection is labeled as either normal, or as an attack, with exactly one specific attack type. Each connection record is about 100 bytes in size.

The 1999 KDD intrusion detection dataset [31] is a version of this 1998 DARPA Intrusion Detection Evaluation Program data. Each record (connection record) in the KDD dataset has 41 features. All the features come from three different sources (or sensors). All the 41 features are divided into three different groups corresponding to each sensor. The different types of cyber attacks in the KDD dataset are listed in the Table 6.1. The first column (Old) lists all the attack types that are present in the training data. The second column (New) list all the new (novel) attacks that are present only in the test data but not in the training data. All the attacks are grouped into the four general cyber attack categories. We evaluate the proposed distributed cyber attack detection system on this KDD dataset. However recently, a statistical analysis of the KDD dataset exposed some issues with the original KDD dataset, which highly affects the performance of the evaluated systems [33]. The main issues found were:

- A number of duplicate records are present in the test dataset, which makes the evaluation biased towards some detection systems.

- The training set includes huge number of redundant records of some attack types.

To eliminate these issues, a new dataset NSL-KDD is proposed [33]. The NSL-KDD dataset consists of selected records of the entire KDD dataset and does not suffer from the above mentioned problems. We evaluated the proposed cyber attack detection system on this NSL-KDD dataset.

The NSL-KDD dataset contains 125,973 records in the training set, and 22,544 records in the test set. Initially 10% of the training set is set aside as the validation set, and the remaining data is used for training the classifiers (support vector machines).

6.2 Training Phase

As mentioned before, we have a total of three sensors. As we try to classify the cyber attacks into the four general categories (DoS, R2L, U2R, and Probe), we have a total of five classes including the normal class. Thus the proposed cyber attack detection system has a total of fifteen binary classifiers (support vector machines), with five binary classifiers at each sensor. In the training phase, all the binary classifiers (support vector machines) under all the sensors are trained on the training set. Support vector machines with the Gaussian kernel are used. The Gaussian kernel function is given by

$$K(\mathbf{x}_p, \mathbf{x}_q) = \exp\left(-\frac{\|\mathbf{x}_p - \mathbf{x}_q\|^2}{2\sigma^2}\right). \tag{6.1}$$

The Gaussian kernel (6.1) corresponds to an infinite dimensional feature mapping. Hence the resulting feature space is a Hilbert space of infinite dimensions.

The support vector machine finally has two parameters which need to be preset: the regularization parameter C and the Gaussian kernel parameter σ. The choice of

these parameters really depends on the problem at hand and the corresponding data. In order to pick the best parameter values for each support vector machine, we adopt a model selection approach. We adopt the simple and effective grid-search using cross-validation. Several pairs of the parameters (C, σ) are used and the one with the best cross-validation accuracy is finally picked for the support vector machine. The grid-search is done using the following values of C and σ:

- $C = 0.125, 0.5, 2, 8, 32, 128, 512.$
- $\sigma = 0.25, 0.5, 1, 2, 4, 8, 16.$

Generally this grid-search based model selection is time consuming. But with our proposed fast training approach, this is not a problem as the model selection is done using the new training set which contains relatively few samples. This further highlights the advantage of the proposed fast training approach.

All the 15 support vector machines are trained on the training set using the proposed fast training approach. For comparison, the support vector machines were also trained in the normal way (i.e., by using the entire training set). The total training time for all the 15 support vector machines using both the approaches is given in Table 6.2. From the table, it is clear that the proposed fast SVM training approach greatly reduces the computational complexity of training the support vector machines (SVMs).

6.3 Decision Making Phase

Once all the binary classifiers (support vector machines) are trained, the posterior class probabilities need to be estimated. As mentioned before, this is done as explained in the Appendix A. As suggested, the unused part of the training set and the validation set are used to fit the sigmoid. The Newton's method with backtracking

line search is used for solving the unconstrained optimization problem. This is done for all of the support vector machines (SVMs).

After obtaining the posterior probability model for each SVM, the local and the final decisions are made using the proposed fusion rules. The following tables show the final confusion matrices on the NSL-KDD test set. Table 6.3 shows the confusion matrix of the proposed distributed cyber attack detection system on the NSL-KDD test set (using only the old attack types). Table 6.4 shows the confusion matrix of the proposed distributed cyber attack detection system, using the SVMs which are trained normally using the entire training set, on the NSL-KDD test set (using only the old attack types). From the tables, it is clear that the proposed cyber attack detection system (with the SVMs trained using the proposed fast training approach) is more effective in detecting the cyber attacks than the system (with the SVMs trained normally). This is due to the fact that in the case of SVMs trained using the proposed fast training approach, more training data is available to fit the sigmoid of the SVM posterior probability model, as only a subset of the whole training data is used to actually train the SVMs. All the remaining training data along with the validation data is available for fitting the sigmoid of the SVM posterior probability model. Whereas in the case of SVMs trained normally, all the training data is used for training the SVMs, and only the validation data is left for fitting the sigmoid of the SVM posterior probability model. This results in the poor posterior probability models for the SVMs which are trained normally. Thus, the proposed fast SVM training approach efficiently uses the available training data.

Also the proposed cyber attack detection system is evaluated on the NSL-KDD test set using the old as well as the new attacks. The Table 6.5 shows the confusion matrix of the proposed cyber attack detection system on the NSL-KDD test set containing both old and new (novel) attacks. From the table, it is clear that the proposed cyber attack detection system performs well on detecting the novel attacks.

Actually, the proposed cyber attack detection system can detect all the new attacks which are statistically similar or close to the old attacks. The proposed system cannot accurately detect the new attacks which are completely different from the old attacks.

The detection accuracies for R2L and U2R attacks are very low when compared to others, because of the less training data available for these types of attacks. The R2L and U2R cyber attacks occur rarely in general. So, we have less training data available for them.

Table 6.1: Types of cyber attacks in the 1999 KDD intrusion detection dataset.

Attack Category	Old	New
Denial of Service (DoS)	back land neptune pod smurf teardrop 7 write	udp storm process table mailbomb apache2
Remote to Local (R2L)	ftp write guess password imap multihop phf spy warezclient warezmaster	snmpgetattack sendmail xlock xsnoop httptunnel named
User to Root (U2R)	buffer overflow loadmodule perl rootkit	sqlattack xterm snmpguess worm ps
Probe	ip sweep nmap port sweep satan	saint mscan

Table 6.2: Total training time for the support vector machines.

SVM	Training Time using the proposed fast training approach (in min.)	Training Time using the normal training approach (in min.)
SVM (Normal, Sensor 1)	69	108
SVM (DoS, Sensor 1)	66	112
SVM (R2L, Sensor 1)	36	56
SVM (U2R, Sensor 1)	28	49
SVM (Probe, Sensor 1)	51	89
SVM (Normal, Sensor 2)	61	105
SVM (DoS, Sensor 2)	58	104
SVM (R2L, Sensor 2)	35	62
SVM (U2R, Sensor 2)	24	46
SVM (Probe, Sensor 2)	45	87
SVM (Normal, Sensor 3)	56	106
SVM (DoS, Sensor 3)	49	99
SVM (R2L, Sensor 3)	31	51
SVM (U2R, Sensor 3)	20	38
SVM (Probe, Sensor 3)	41	79

Table 6.3: Confusion matrix of the proposed cyber attack detection system on the NSL-KDD test set (with only old attacks).

Predicted → Actual ↓	Normal	DoS	R2L	U2R	Probe	Correct Detection Rate
Normal	9430	88	41	26	126	97.11%
DoS	119	5441	22	19	140	94.77%
R2L	852	147	796	15	389	36.20%
U2R	8	3	2	19	5	51.35%
Probe	106	19	27	12	942	85.17%

Table 6.4: Confusion matrix of the proposed cyber attack detection system (using SVMs which are trained normally) on the NSL-KDD test set (with only old attacks).

Predicted → Actual ↓	Normal	DoS	R2L	U2R	Probe	Correct Detection Rate
Normal	9412	112	29	23	135	96.92%
DoS	167	5341	45	34	154	93.03%
R2L	1293	96	551	17	242	25.06%
U2R	12	6	3	10	6	27.03%
Probe	165	55	41	12	833	75.32%

Table 6.5: Confusion matrix of the proposed cyber attack detection system on the NSL-KDD test set (containing both old and new attacks).

Predicted → Actual ↓	Normal	DoS	R2L	U2R	Probe	Correct Detection Rate
Normal	9430	88	41	26	126	97.11%
DoS	369	6654	60	34	341	89.22%
R2L	1022	287	801	25	419	31.36%
U2R	248	77	14	24	37	6.00%
Probe	318	121	43	27	1912	78.98%

CHAPTER 7

CONCLUSIONS

In this thesis, a new distributed cyber attack detection system is proposed. The proposed cyber attack detection system has relatively less computational complexity and high detection accuracy. The proposed system uses multiple sensors across the network. At each sensor, local processing is done and a local decision is generated. All the local decisions are then sent to the fusion center, where the final decision regarding the state of the network is generated.

Support vector machines are used as the supervised binary classifiers at each sensor. Though the support vector machines are the best supervised classifiers, their training is often a computationally intensive process. A new fast and efficient training approach is proposed which greatly reduces the computational complexity of the support vector machine training process without significantly affecting the classification performance.

Finally, the fusion rules are designed using the Dempster-Shafer framework, which is a more generalized mathematical framework. The main advantage of using the Dempster-Shafer framework is that it exactly characterizes the amount of evidence available at each sensor. The Dempster's rule of combination effectively combines the evidence from the multiple sensors (sources of evidence).

BIBLIOGRAPHY

[1] D. E. Denning, "An intrusion-detection model," *IEEE Transactions on Software Engineering*, vol. 13, pp. 222–232, 1987.

[2] Sourcefire, *Snort*. http://www.snort.org/.

[3] V. Paxson, *The Bro Network Security Monitor*. http://www.bro-ids.org/.

[4] Y. Vandoorselaere, *Prelude Intrusion Detection System*. http://prelude-ids.org/.

[5] TrendMicro, *OSSEC*. http://www.ossec.net/.

[6] McAfee, *McAfee Network Security Platform*. http://www.mcafee.com/.

[7] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[8] V. N. Vapnik, *Estimation of Dependencies Based on Empirical Data*. Springer, 1982.

[9] E. Osuna, R. Freund, and F. Girosi, "Support vector machines: Training and applications," *MIT AI Memo AIM-1602*, 1997.

[10] J. C. Platt, "Probabilistic outputs for support vector machines and comparison to regularized likelihood methods," *Advances in Large Margin Classifiers*, 2000.

[11] A. Smola and B. Schölkopf, "Sparse greedy matrix approximation for machine learning," *Proceedings of the Seventeenth International Conference on Machine Learning*, pp. 911–918, June 2000.

[12] S. Fine and K. Scheinberg, "Efficient svm training using low-rank kernel representations," *Journal of Machine Learning Research*, vol. 2, pp. 243–264, 2001.

[13] B. Li, Q. Wang, and J. Hu, "A fast svm training method for very large datasets," *Proceedings of International Joint Conference on Neural Networks*, June 2009.

[14] J. C. Platt, "Sequential minimal optimization: A fast algorithm for training support vector machines," *Advances in Kernel Methods - Support Vector Learning*, 1998.

[15] 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*, 2005.

[16] Y. J. Lee and O. L. Mangasarian, "Rsvm: Reduced support vector machines," *Proceedings of the First SIAM International Conference on Data Mining*, 2001.

[17] S. Tong and D. Koller, "Support vector machine active learning with applications to text classification," *Journal of Machine Learning Research*, 2001.

[18] S. Abe and T. Inoue, "Fast training of support vector machines by extracting boundary data," *Proceedings of the International Conference on Artificial Neural Networks*, 2001.

[19] H. Shin and S. Cho, "Fast pattern selection for support vector classifiers," *Lecture Notes in Computer Science, Springer*, 2003.

[20] A. Lyhyaoui, M. Martinez, I. Mora, M. Vazquez, J. L. Sancho, and A. R. Figueiras-Vidal, "Sample selection via clustering to construct support vector-like classifiers," *IEEE Transactions on Neural Networks*, vol. 10, no. 6, pp. 1474–1481, 1999.

[21] R. Koggalage and S. Halgamuge, "Reducing the number of training samples for fast support vector machine classification," *Neural Information Processing - Letters and Reviews*, vol. 2, no. 3, pp. 57–65, 2004.

[22] J. Cervantes, X. Li, and W. Yu, "Support vector machine classification based on fuzzy clustering for large data sets," *Lecture Notes in Computer Science, Springer*, vol. 4293, 2006.

[23] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.

[24] C. E. Rasmussen, "The infinite gaussian mixture model," *Advances in Neural Information Processing Systems*, pp. 554–560, 2000.

[25] R. M. Neal, "Markov chain sampling methods for dirichlet process mixture models," *Journal of Computational and Graphical Statistics*, vol. 9, no. 2, pp. 249–265, 2000.

[26] Y. Ye, M. J. Todd, and S. Mizuno, "An $o(\sqrt{n}l)$-iteration homogeneous and self-dual linear programming algorithm," *Mathematics of Operations Research*, 1994.

[27] F. John, "Extremum problems with inequalities as subsidiary conditions," *Studies and Essays Presented to R. Courant on his 60th Birthday*, pp. 187–204, 1948.

[28] P. Sun and R. M. Freund, "Computation of minimum-volume covering ellipsoids," *INFORMS Operations Research*, vol. 52, no. 5, pp. 690–706, 2004.

[29] A. P. Dempster, "A generalization of bayesian inference," *Journal of the Royal Statistical Society, Series B (Methodological)*, vol. 30, no. 2, pp. 205–247, 1968.

[30] G. Shafer, *A Mathematical Theory of Evidence*. Princeton University Press, 1976.

[31] S. Hettich and S. D. Bay, *The UCI KDD Archive*. University of California Irvine, Department of Information and Computer Science, 1999.

[32] R. P. Lippmann, D. J. Fried, I. Graf, J. W. Haines, K. R. Kendall, D. McClung, D. Weber, S. E. Webster, D. Wyschogrod, R. K. Cunningham, and M. A. Zissman, "Evaluating intrusion detection systems: the 1998 darpa off-line intrusion

detection evaluation," *Proceedings of the 2000 DARPA Information Survivability Conference and Exposition*, vol. 2, 2000.

[33] M. Tavallaee, E. Bagheri, W. Lu, and A. Ghorbani, "A detailed analysis of the kdd cup 99 data set," *Proceedings of the IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA)*, 2009.

[34] H.-T. Lin, C.-J. Lin, and R. C. Weng, "A note on platt's probabilistic outputs for support vector machines," *Machine Learning*, vol. 68, no. 3, pp. 267–276, 2007.

APPENDIX A

PROBABILISTIC OUTPUTS OF SUPPORT VECTOR MACHINES

The support vector machine (SVM) does not provide probabilistic outputs. It just makes hard decisions (predictions) about the new inputs, i.e., the support vector machine (SVM) just gives a hard decisions whether the new input belongs to one class or the other. Though these hard decision are based on the value of the SVM function $f(\mathbf{x})$, this value is an uncalibrated value and is not a probability. Posterior probabilities, which gives the probability of a new sample belonging to a particular class, are extremely useful in many practical applications. The posterior probabilities are also very useful in situations when a particular SVM makes a small portion of the overall decision, and the final classification decision is based on combining all such individual SVM decisions.

Platt proposes a post-processing approach to calculate the posterior probabilities, after training the support vector machine [10]. The proposed approach is a post-processing step, and hence can be applied on already trained support vector machines (SVMs). The basic idea is to fit a logistic sigmoid function to the outputs of an already trained support vector machine. The proposed sigmoid model is

$$P(y = +1|f(\mathbf{x})) = \frac{1}{1 + \exp(Af(\mathbf{x}) + B)}, \tag{A.1}$$

where A and B are the parameters of the model. The logistic sigmoid model (A.1) is equivalent to the assumption that the output of the support vector machine is proportional to the log-odds of the positive data sample. The parameters A and B of the model (A.1) are found by minimizing the negative log-likelihood of the training

data:

$$\begin{aligned} &\min_{A,B} \quad -\sum_{i=1}^{q} t_i \log(p_i) + (1-t_i)\log(1-p_i), \\ &\text{where} \\ &t_i = \frac{1+y_i}{2}, \\ &p_i = \frac{1}{1+\exp(Af(\mathbf{x}_i)+B)}. \end{aligned} \tag{A.2}$$

The training set used above is $\mathbf{R} = \{(f(\mathbf{x}_i), y_i); i = 1, ..., q\}$. The above optimization problem (A.2) is an unconstrained one. The targets which are actually used in the above optimization problem (A.2) are $t_i \in \{0, 1\}$. However, more effective targets can be derived. These new targets t_i represent the probability of the positive and negative labels. The new targets are actually the MAP estimates for the target probabilities of positive and negative samples, which are given by

$$t_i \begin{cases} = \frac{N_+ + 1}{N_+ + 2}, & \text{if } y_i = +1 \\ = \frac{1}{N_- + 2}, & \text{if } y_i = -1 \end{cases} \tag{A.3}$$

where N_+ and N_- are the number of positive and negative labels respectively. Thus, the actual optimization problem that is solved is

$$\begin{aligned} &\min_{A,B} \quad -\sum_{i=1}^{q} t_i \log(p_i) + (1-t_i)\log(1-p_i), \\ &\text{where} \\ &t_i \begin{cases} = \frac{N_+ + 1}{N_+ + 2}, & \text{if } y_i = +1 \\ = \frac{1}{N_- + 2}, & \text{if } y_i = -1 \end{cases} \qquad p_i = \frac{1}{1+\exp(Af(\mathbf{x}_i)+B)}, \end{aligned} \tag{A.4}$$

where N_+ and N_- are the number of positive and negative labels respectively.

The training set used in the above optimization problem (A.4), $\mathbf{R} = \{(f(\mathbf{x}_i), y_i); i = 1, ..., q\}$, needs to be different from the actual training set used for training the SVM. This is due to the fact that using the same training set often leads to disastrously biased fits. In [10], Platt proposes two different methods for deriving an unbiased

training set: hold-out set method and cross-validation method. Here in our case, the support vector machine (SVM) is trained on a subset of the actual training set. Hence, we can use the other unused training set to fit the sigmoid. However, using the entire unused training set at once may sometimes lead to wrong estimates of the target probabilities, especially when one class dominates the other class. To eliminate this difficulty, we divide the unused training set into three parts, and the trained SVM is evaluated each of these three parts separately. Finally, the union of all these three different sets of SVM outputs forms the new unbiased training set which is used to fit the sigmoid.

Platt proposes a Levenberg-Marquardt (LM) algorithm to solve the optimization problem (A.4). However, it is found that the proposed approach suffers from some severe drawbacks [34]. In [34], Lin *et al.* propose a Newton's method with backtracking line search for solving the optimization problem (A.4), which is found to be more efficient. We adopt the Newton's method with backtracking line search for solving the unconstrained optimization problem (A.4).

VITA

Sandeep Gutta

Candidate for the Degree of

Master of Science

Thesis: A NEW DISTRIBUTED FRAMEWORK FOR CYBER ATTACK DETECTION AND CLASSIFICATION

Major Field: Electrical Engineering

Biographical:

Personal Data:

Born in Kakinada, Andhra Pradesh, India on September 10, 1987.

Education:

Received the B.E. degree from Andhra University, Visakhapatnam, Andhra Pradesh, India, in Electronics and Communication Engineering in 2008.

Completed the requirements for the degree of Master of Science with a major in Electrical Engineering, Oklahoma State University, Stillwater, OK, in December 2011.

Experience:

Worked as Research Assistant at the Statistical Signal Processing Laboratory, Oklahoma State University, Stillwater, OK, from August 2009 to December 2011.

Professional Memberships:

Institute of Electrical and Electronics Engineers (IEEE)
Golden Key International Honour Society

Name: Sandeep Gutta Date of Degree: December, 2011

Institution: Oklahoma State University Location: Stillwater, Oklahoma

Title of Study: A NEW DISTRIBUTED FRAMEWORK FOR CYBER ATTACK DETECTION AND CLASSIFICATION

Pages in Study: 77 Candidate for the Degree of Master of Science

Major Field: Electrical Engineering

With the fast growing cyber activity day by day, the threat from cyber attacks has increased enormously. The timely detection of these cyber attacks has been a major concern to many governments and organizations all over the world. A number of cyber attack detection systems have been developed in the past decade. However, most of them tend to suffer from two main issues: high computational complexity and low detection accuracy. In this thesis, a new distributed framework is proposed for cyber attack detection. Besides detecting the attacks, the proposed system also classifies the attacks into different categories so that corresponding proper counteraction can be taken in time. The proposed system uses multiple sensors which are deployed at various parts of the network, thus providing a complete view of the network. The traditional centralized processing approach, in which all the sensors transmit their entire data to a central decision making unit, has high computational complexity and requires huge bandwidth. Hence, the proposed system employs distributed processing, where each sensor processes the observed data and generates a local decision. All the local decisions from all the sensors are then transmitted to the fusion center, which generates a final decision based on all the available local decisions. At each sensor, multiple supervised binary classifiers are employed. Support vector machines, which are one of the best, are used as the classifiers. A new fast and efficient training approach for support vector machines is proposed, which greatly reduces the computational complexity of training the support vector machines without significantly affecting the classification performance. Effective fusion rules, at each sensor and at the fusion center, are proposed using the Dempster-Shafer theory. The proposed cyber attack detection system is evaluated using the popular 1999 KDD intrusion detection dataset, which is a version of the 1998 DARPA intrusion detection evaluation program data.

ADVISOR'S APPROVAL: Dr. Qi Cheng