ADDIS ABABA UNIVERSITY

SCHOOL OF GRADUATE STUDIES
FACULTY OF TECHNOLOGY

INTRUSION DETECTION SYSTEM USING VISUALIZATION AND INTEGRATION TECHNIQUE

BY

SISAY DENBOBA

August, 2006

ADDIS ABABA UNIVERSITY

SCHOOL OF GRADUATE STUDIES
FACULTY OF TECHNOLOGY

INTRUSION DETECTION SYSTEM USING VISUALIZATION AND INTEGRATION TECHNIQUE

A thesis submitted to the School of Graduate Studies of Addis Ababa University in partial fulfillment for the Degree of Master of Science in Computer Engineering

By

SISAY DENBOBA

Advisor

Dr. Kumudha Raimond

Addis Ababa

August 2006

ADDIS ABABA UNIVERSITY

SCHOOL OF GRADUATE STUDIES
FACULTY OF TECHNOLOGY

INTRUSION DETECTION SYSTEM USING VISUALIZATION AND INTEGRATION TECHNIQUE

BY

SISAY DENBOBA

Approval by Board of Examiners

Dr. Getachew Biru	_____
Chairman, Department of Electrical and Computer Engineering	Signature
Dr. Kumudha Raimond	_____
Advisor	Signature
Mr. Manochj V.N.V	_____
External Examiner	Signature
Dr. D. P. Roy	_____
Internal Examiner	Signature
_____	_____
Internal Examiner	Signature

ACKNOWLEDGMENT

First and foremost, I offer my thanks to GOD, who made this work possible.

I express my deepest gratitude to my advisor Dr. Kumudha Raimond for her advice, and suggestions that greatly enriched this thesis.

I have to express my sincere appreciation to the staff of Electrical and Computer Engineering and Information Science departments for their encouragement and concern.

Finally, my heartfelt appreciation and indebtedness goes to my family, my fiancé and my friends for their love, patience, support and encouragement throughout my study.

CONTENT

List of Tables

List of Figures

Glossary of Acronyms

DM: Data Mining

KDD: Knowledge Discovery and Data Mining

GA: Genetic Algorithm

SOM: Self Organized Map

DoS: Denial of Service

R2L: Access from a remote machine

U2R: Access to local super user

ANN: Artificial Neural Network

BP: back propagation algorithm

MLP: Multi Layer Perceptron

RBF: Radial Basis Function

IDS: Intrusion Detection Systems

DB: Davies-Bouldin index or validity index

NSGA: Non-dominated Sorting Genetic Algorithm

LAN: Local Area Network

HIDS: Host Intrusion Detection Systems

NIDS: Network Intrusion Detection Systems

ABSTRACT

Intrusion detection is an area of ever increasing importance. Currently existing Intrusion Detection Systems (IDS) lack visualization and false alarms detection capabilities. Researchers have proposed integrated systems which may reduce the percentage of false alarms. This work addresses the above stated problems by integrating Self-Organized Map (SOM) with Genetic Algorithm (GA) so as to minimize the false alarms and also to provide visualization capability to the new IDS. SOM is an unsupervised Artificial Neural Network (ANN) learning algorithm that attempts to visualize a large dataset in compact representation. GA is an evolutionary computing type of artificial intelligence algorithm, which is better for optimization, feature selection and clustering problems. The performance of the model is measured using Knowledge Discovery and Data Mining (KDD) Cup 99 dataset, which was prepared for The Third International Knowledge Discovery and Data Mining (DM) Tools Competition for researchers who work on intrusion detection. The work also includes GA based feature selection to further improve the performance of the model. The result shows 94.3 % of intrusion detection rate with 2.93% of false alarm rate.

.

CHAPTER ONE

INTRODUCTION

1.1 Background

These days, many important activities are computerized. Money handling, electronic commerce and Internet banking services are very common. Machineries such as power plants and medical equipments operate automatically by computers. Companies rely on computers for their business processes such as keeping track of stock, automatic ordering, producing documents and storing business secrets. People communicate to a great extent through computer systems by electronic mails, electronic newspapers and web sites. Even though such computerization has multitude of advantages and such communication brings a revolution in the ability to exchange information, it also provides a greater opportunity for disruption and sabotage of data.

The nature of computer crime has changed over the years as the technology has changed and the opportunities for crime have increased. Thrill-seeking adolescent hackers are still common. Creating viruses and cracking systems are their daily activities to abuse the computerized systems. Because of this reason, computer security is necessary to protect vulnerable systems.

A common view of computer security is that the threat comes from a vast group of malicious hackers. The focus of many computer security mechanisms is to keep track of outsiders through physical and technical measures. While the threat from outsiders is indeed as great as generally believed, the malicious insiders with approved access to the system are an even greater threat. Basically, these computer threats can be divided as denial of service and unauthorized access from a remote machine, unauthorized access from local machine and probing.

Even though it is impossible to build a completely secure system, many solutions are proposed and implemented to solve the security problems. Using cryptographic method to secure password, firewall to block attacks and IDS to detect attacks are some of the solutions.

IDS are used to help computer systems to deal with attacks. This goal is accomplished by collecting information from a variety of system and network sources and then analyzing the information for symptoms of security problems. Researchers have proposed different types of intrusion detection methods such as misuse and anomaly type, network-based and host-

based. But these methods have their own drawbacks such as high percentage of false alarm and limitation in visualizing the various attacks. False alarm is normal connection which considered as attack by the system. In recent years, the researchers have proposed new methodologies to overcome the drawbacks of currently existing IDS by integrating the previously developed intrusion detection methods.

1.2 Statement of the Problem

Even if different types of intrusion detection methods are implemented using statistical, DM and ANN models, there is still a lot of false alarm problem. Moreover, since network traffic contains very huge amount of data, visualization method will be useful to observe the result. However, the current IDS lack the above mentioned feature and further suffer from large percentage of false alarm. So, the current work address these problems and also tries to improve the performance of IDS by integrating SOM for the purpose of data reduction and visualization and GA for clustering the various attacks. Moreover GA based feature selection approach also is used in this work to remove unwanted noises from the dataset.

1.3 Objective

The main objective of this work is to develop an IDS prototype that uses integrated approach in order to solve false alarms and visualization problem. Specifically, it includes the following specific objectives.

- Pre-processing the KDD dataset to use it as training and testing data

- Implementing feature selection algorithm to remove noisy attributes from the dataset
- Develop an IDS prototype by integrating SOM and GA.
- Test the model

1.4 Methodology

- Simulating and preparing data to test IDS is out of scope of this thesis. So, in this work, the well structured KDD CUP 99 dataset is used to train and test the model.
- Data preprocessing is another task that should be done before using the data. Textual details are represented using numbers and numerical data are normalized.
- To improve the performance of the model, selecting the best features from the dataset is another crucial work. GA model is used to select the best features and to remove the noisy features.
- Finally, the IDS is developed by integrating SOM with GA. All the above algorithms are developed using Java Programming language.

1.5 Thesis Outline

The following section covers literature survey; this is to review the published and unpublished works, journals and books and to gather any information relevant to the research work. Section three and four gives background knowledge about IDS, SOM and GA. It introduces about computer security, ANN models and IDS in more detail. The methodology of feature selection, visualization and clustering are described in detail in section five. The implementation and the results are demonstrated in section six. Conclusions and future work are presented in section seven.

CHAPTER TWO

LITERATURE SURVEY

2.1 Introduction

A number of researches have been done work on intrusion detection. Since it is difficult to cover all the previous research works, only some of the papers are compiled in this part. Many researchers developed IDS by using different approaches. Most agree that the central part of IDS is the detector module and its underling principle of operation. Hence, the literature survey is divided into three parts: the first part deals with general detection methodologies like anomaly and misuse detection approaches, the second part covers recently proposed solutions such as visualization and combined or integrated approaches. Finally, the experience of using SOM, GA and DM independently for intrusion detection is covered in part three.

2.2 Misuse and Anomaly Detection

2.2.1 Misuse detection

Misuse detection is the process of attempting to identify instances of network attacks by comparing current activity against the expected actions of an intruder. Misuse detection is primarily done using some forms of pattern matching. The simpler form of pattern matching is string matching [10]. It is easy for the administrator to add his own patterns for events he wishes to notify about. Other misuse detection systems use a "bag of system calls" representation for intrusion detection of system call sequences. The representation of system call sequences is effective and often performs better than those approaches that use foreign contiguous subsequences for detecting intrusive behaviors of compromised processes [16].

Expert systems have been used in several IDS, for example in [11] and [12]. The underlying idea is to create a knowledge base by talking to experts in the area. The rules in

the expert system are often in the form of if-then structure, which are used to derive relevant facts from the log data. It is possible to create more complex "scenarios" using an expert system than using simple string matching, and it is probably easier to encode knowledge about misuse. Both papers include descriptions of special languages that are designed to make it easier to write pattern-matching rules.

In addition to simple pattern matching and expert system, some works use ANN models to apply misuse detection approach. ANN provides the potential to identify and classify network activity based on limited, incomplete, and nonlinear data sources. In [13], an approach was used to the process of misuse detection that utilizes the analytical strengths of neural networks.

DM is another popular method used for intrusion detection [14]. The paper uses DM to extract useful patterns from network traffic and creates intrusion models. This method is interesting since it automates the creation of detection rules somewhat. The authors claim that their rules become more general and produce fewer false alarms than "hand-coded" rules. In addition, DM can also be used for anomaly detection.

One of the largest challenges for misuse intrusion detection tools is to be able to generalize from previously observed behaviors (normal and malicious) to recognize similar future behavior [17]. Anomaly detection is a solution for this.

2.2.2 Anomaly detection

Anomaly detection assumes that intrusions are highly correlated to abnormal behavior exhibited by either a user or the system. Anomaly detection can be done in many ways. Statistical techniques, neural networks, and genetic programming are some of the methods that have been implemented for intrusion detection. The best-described statistical anomaly detection component is the one used in [15]. This is also one of the most advanced anomaly detection systems. These authors calculate statistics for a large number of parameters in the system. The parameters are updated daily to adapt to the changes in behavior. Each audit

record that is compared to the profiles is given a score value that represents its similarity to the profile parameters. A threshold value can be set to report only those events that have a higher score value.

Some works generalize anomaly and misuse detections. Most IDS lack the ability to generalize from previously observed attacks to detect even slight variations of known attacks. Paper [17] describes process-based intrusion detection approach that provides the ability to generalize from previously observed behavior to recognize future unseen behavior. The approach employs ANN, and can be used for both anomaly detection in order to detect novel attacks and misuse detection in order to detect known attacks and even variations of known attacks.

Most anomaly detection methods are more or less self-learning and can be trained to recognize normal behavior. Paper [18] used another approach that he calls specification-based detection. The intended security-relevant behavior of programs is manually specified using a formal method. Deviations from this specification are then detected. This may be very useful for monitoring security critical programs.

2.3 Visualization and Integrated Approach

2.3.1 Visualization Approach

Paper [19] presents a new approach to detection. They used an unsupervised neural network to reduce the dimensions of the input data. SOM was used to project network events on a space appropriate for visualization. The output was visualized in a grid where the units of the map were portrayed as squares. Squares of different sizes and colors represent the number of events mapped in the unit and the weight of the selected attributes. They focus on visualization for use in intrusion detection. While the prototype they developed is still in its infancy, they have nonetheless shown that the visual paradigm for analyzing network activity is a powerful approach for anybody confronted with the monitoring of activity-based information.

In paper [20], they showed the applicability of visualization approach for large-scale network intrusion detections. They explore the applicability of visualization techniques in conjunction with a well-known intrusion detection system for the detection and analysis of misuse of computer system connected to the internet. The visualization techniques will allow users to identify the behavior of user connection to the system and identify those whose intentions are unwelcome.

To ensure the normal operation of a large computer network system, another common practice is to constantly collect system logs and analyze the network activities for detecting anomalies. Most of the analysis methods in use today are highly automate due to the enormous size of the collected data. Conventional automated methods are largely based on statistical modeling, and some employ machine learning. In paper [21], they show interactive visualization as an alternative and effective data exploration method for understanding the complex behavior of computer network systems. They describe three log-file analysis applications, and demonstrate how the use of visualization-centered tools can lead to the discovery of flaws and intruders in the network systems.

2.3.2 Integrated Approach

Combining methods may give better convergence, and may make the detection more effective. The question is what methods to combine and how.

Paper [22] combines two neural network algorithms and presents an IDS based on SOM and Resilient Propagation Neural Network for visualizing and classifying intrusion and normal patterns. They introduced a cluster matching equation for finding associated principal components in component planes. They used KDD Cup 99 dataset for training and testing their prototype. Their approach achieved about 90% of detection rate and 5% of false alarm rate.

Another paper [23] combines three different methods in one system. DM is used to find indicators. A way to combine methods is to use several layers of filtering. For example, it may be very useful to analyze alarms by collecting different scenarios, by removing redundant alarms, or by getting some more information about the incident. Even if there is a room to further improve the performance of IDS by integrating different types of models, there are only the above stated few works available in this area. Most visualization approaches also in infancy stage.

2.4 Self Organized Map and Genetic Algorithm

In this section, the use of SOM and GA for the purpose of intrusion detection is revised from the previous research works.

2.4.1 Self Organized Map

As discussed on misuse and anomaly detection, many papers use neural networks to detect intrusions. Multi Layer Perceptron (MLP), Radial Basis Function (RBF), Adaptive Resonance Theory and SOM are some of the algorithms that have been used by different researchers.

Especially, SOM has been used in many papers for clustering and visualizing intrusions. Paper [25] describes results concerning the robustness and generalization capabilities of machine learning methods in creating network profiles based on network traffic measurements generated by real-time TCPTrace. This data is used to train a SOM model. The SOM is an unsupervised non-linear ANN model. An empirical evaluation of the approach demonstrated that it can differentiate the normal traffic and abnormal ones.
In the following section, the use of SOM and GA for intrusion detection work is revised from previous research works.

2.4.2 Genetic Algorithm

GA also been used by different researchers for intrusion detection. Basically GA is for optimization problem, but it can also be used for clustering and feature selection problems. A brief overview of the IDS using GA and related detection techniques is presented in [26]. Parameters and evolution process for GA are discussed in detail. Unlike other implementations of the same problem, this implementation considers both temporal and spatial information of network connections in encoding the network connection information into rules in IDS.

GA is famous for optimization problem and is also used for classification [27]. This paper describes the design of genetic classifier-based IDS, which can provide active detection and automated responses during intrusions. It is designed to be a sense and response system that can monitor various activities on the network. In particular, it simultaneously monitors networked computers' activities at different levels (such as user level, system level, process level and packet level) and use a genetic classifier system in order to determine a specific action to be taken in case of any security violation.

GA can also be used to select a subset of input features for decision tree classifiers, with a goal of increasing the detection rate and decreasing the false alarm rate in network intrusion detection [28]. They used the KDDCUP 99 dataset to train and test the decision tree classifiers. The experiments show that the resulting decision trees can have better performance than those built with all available features.

Paper [29] proposes and surveys genetic implementations of algorithms for selection and partitioning of attributes in large-scale concept learning problems. The purpose of this approach is to improve upon existing search-based algorithms (or wrappers) in terms of training efficiency. Several GA implementations of alternative (search-based and knowledge-based) attribute synthesis algorithms are surveyed, and their application to large-scale concept learning problems is addressed.

The problem of feature selection is an important real-world problem that involves multiple objectives to be simultaneously optimized. In order to tackle this problem, paper [30] proposes a multi-objective GA for feature selection based on the wrapper approach. The algorithm's main goal is to find the best subset of features that minimizes both the error rate and the size of the tree discovered by a classification algorithm, namely C4.5, using the Pareto dominance concept.

2.5 KDD CUP Data

KDD CUP 99 is the dataset prepared for The Third International Knowledge Discovery and Data Mining Tools Competition, which was held in conjunction with KDD-99 the Fifth International Conference on Knowledge Discovery and Data Mining [1]. The competition task was to build a network intrusion detector, a predictive model capable of distinguishing between "bad" connections, called intrusions or attacks, and "good" normal connections. This database contains a standard set of data to be audited, which includes a wide variety of intrusions simulated in a military network environment.

The 1998 DARPA Intrusion Detection Evaluation Program was prepared and managed by MIT Lincoln Labs. The objective was to survey and evaluate research in intrusion detection. A standard set of data to be audited, which includes a wide variety of intrusions simulated in a military network environment, was provided. The 1999 KDD intrusion detection contest uses a version of this dataset.

Lincoln Labs set up an environment to acquire nine weeks of raw TCP dump data for a local-area network (LAN) simulating a typical U.S. Air Force LAN. They operated the LAN as if it were a true Air Force environment, but peppered it with multiple attacks.

The raw training data was about four gigabytes of compressed binary TCP dump data from seven weeks of network traffic. This was about five million connection records. Similarly, the two weeks of test data yielded around two million connection records.

A connection is 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 protocols. Each connection is labeled as either normal, or as an attack, with exactly one specific attack type. Each connection record consists of about 100 bytes.

Attacks fall into four main categories:

- DOS: denial-of-service, e.g. syn flood;
- R2L: unauthorized access from a remote machine, e.g. guessing password;
- U2R: unauthorized access to local super user (root) privileges, e.g., various "buffer overflow" attacks;
- Probing: surveillance and other probing, e.g., port scanning.

It is important to note that the test data is not from the same probability distribution as the training data, and it includes specific attack types which are not in the training data. This makes the task more realistic. Some intrusion experts believe that most novel attacks are variants of known attacks and the "signature" of known attacks can be sufficient to catch novel variants. The datasets contain a total of 22 training attack types, with an additional 17 different types in the test data and normal connection in both dataset(refer on Appendix I, table B)

The distribution of the data in each attack type category of the KDD dataset is shown in the table 2.1. Different types of attacks in each category are presented in Appendix I, table B.

Table 2.1 Distribution of attack types in KDD dataset

No	Category	Amount in number
1	Normal	97, 278
2	DOS	391, 458
3	Probe	4, 107
4	R2L	1, 126
5	U2L	52

KDD CUP dataset used by many researchers to train, test and evaluate their IDS models [22] [28]. Paper [31] evaluates performance of a comprehensive set of pattern recognition and machine learning algorithms on four attack categories as found in the KDD 1999 Cup intrusion detection dataset.

2.6 Conclusion

Even though IDS is a very wide area, which is difficult to cover in this short literature survey, the researcher tries to cover main research areas done on intrusion detection. From the survey, one can clearly identify the gap in the research area to contribute further. As briefly described in the introduction part, this work is to fill the gap in minimizing false alarms and add visualization capability in IDS. The work uses integration and visualization technique to satisfy the above key points.

CHAPTER THREE

INTRUSION DETECTION

3.1 Computer Threats and Security

There are two extremes: absolute security and absolute access. The closest to an absolutely secure machine is one that is unplugged from the network but it is not useful in this state. On the contrary, a machine with absolute access is extremely convenient to use, but is not practical because of network threats. So, every organization needs to decide for itself where between the two extremes of total security and total access. A policy needs to articulate this, and then define how that will be enforced in practice and everything that is done in the name of security, then, must enforce that policy uniformly.

A threat is an unwanted (deliberate or accidental) event that may result in harm to an asset. Often, a threat is exploiting a known vulnerability. There are different types of threats to networked computers. One can generalize the type of threats into denial of service and unauthorized access categories.

3.1.1 Denial-of-Service

DoS (Denial-of-Service) attacks are probably the nastiest, and most difficult to address. These are the most horrible, because they're very easy to launch, difficult (sometimes impossible) to track, and it isn't easy to refuse the requests of the attacker.

The principle of a DoS attack is simple: send more requests to the machine than it can handle. There are toolkits available in the underground community that make this a simple matter by running a program and telling it which host to blast with requests. The attacker's program simply makes a connection on some service port, perhaps forging the packet's header information that says where the packet came from, and then dropping the connection. If the host is able to answer 20 requests per second, and the attacker sends 50

requests per second, obviously the host will be unable to service all of the attacker's requests, much less any legitimate requests.

Such attacks were fairly common in late 1996 and early 1997, but are now becoming less popular.

Some of the approaches that can be followed to reduce the risk of being stung by a DoS attack include

- Not running your visible-to-the-world servers at a level too close to capacity
- Using packet filtering to prevent obviously forged packets from entering into your network address space.
- Keeping up-to-date on security-related patches for your hosts' operating systems.

3.1.2 Unauthorized Access

Unauthorized access is a very high-level term that can refer to a number of different sorts of attacks. The goal of these attacks is to access some resources that your machine should not provide the attacker. For example, a host might be a web server, and should provide anyone with requested web pages. However, that host should not provide command shell access without being sure that the person making such a request is someone who should get it, such as a local administrator. This kind of attack can be from local user who need to abuse administrative privilege or it can also be from remote users. Unauthorized access can have any one of the following behaviors:

Executing Commands Illicitly

It's obviously undesirable for an unknown person to be able to execute commands on your server machines. There are two main classifications of the severity of this problem: normal user access, and administrator access. A normal user can do a number of things on a system (such as read files, mail them to other people, etc.) that an attacker should not be able to do. This might, then, be all the access that an attacker needs. On the other hand, an attacker

might wish to make configuration changes to a host (perhaps changing its IP address, putting a start-up script in place to cause the machine to shut down every time it's started or something similar). In this case, the attacker will need to gain administrator's privileges on the host.

Confidentiality Breaches

Certain information (such as personal record, company information, credit card details and others), would cause damage once gone into the hands of a competitor, an enemy, or the public. In this case, it's possible that a normal user's account on the machine can be enough to cause damage, while many of the perpetrators of this sort of break-in can be merely thrill-seekers interested in nothing or more malicious sort of break-in.

Destructive Behavior

Among the destructive sorts of break-ins and attacks, there are two major categories (data diddling and data destruction).

Data diddling is a destructive sort of break-in and attack. It is likely the worst sort, since the break-in might not be immediately obvious. Perhaps It is toying with the numbers in your spreadsheets, or changing the dates in your projections and plans. Maybe It is changing the account numbers for the auto-deposit of certain paychecks. One day when one notices the discrepancy in the system, trying to track the problem down will certainly be difficult.

Data Destruction is a destructive sort of break-in and attack where some of the perpetrate attacks like to delete things. In this case, the impact on your computing capability and consequently your business can be nothing less than if a fire or other disaster caused your computing equipment to be completely destroyed.

3.2 Intrusion detection

Intrusion detection can be defined as a process of monitoring the events that occur in a computer system or network and analyzing for any intrusions considering confidentiality, integrity and availability of a computer system or other network infrastructure security properties. An intrusion causes the confidentiality violation, if it allows intruders to access system information without authorization (password authentication). If an intruder changes the system date or any data residing on or passing through the system causes the integrity violation. An availability violation will occur if an intrusion keeps an authorized user from accessing the particular service or system resource when he needs it. The major task of intrusion detection is to gather data that may contain evidences of intrusions from target system processing and analyzing the data to identify the potential intrusions and generate the specified responses for every intrusion either manually or automatically.

3.2.1 Intrusion Detection Systems (IDS)

An intrusion detection system (IDS) monitors network traffic for suspicious activity and alerts the system or network administrator. In some cases, the IDS may also respond to anomalous or malicious traffic by taking action such as blocking the user or source IP address from accessing the network.

As soon as someone discovers new computer security vulnerability, horde of crackers start knocking at the doors of computers worldwide to see if they can penetrate their defenses. Many sites employ a combination of border router firewalls and host-based packet filters and wrappers to protect themselves, but what if the vulnerability is in the very mechanism that's used to secure a service? How can system’s administrator know that their machines are under attack and/or have been compromised? The best way to catch the crackers is to use IDS.

3.2.2 Need for IDS

A computer system should provide confidentiality, integrity and assurance against different type of attacks. However, due to increased connectivity (especially on the Internet), and the vast spectrum of financial possibilities that are opening up, more and more systems are subject to attack by intruders. These subversion attempts to exploit flaws in the operating system as well as in application programs and have resulted in spectacular event like the Internet Worm incident of 1988.

The presence of IDS allows preventing subversion by building a completely secure system. We could, for example, require all users to identify and authenticate themselves; we could protect data by various cryptographic methods and very tight access control mechanisms. However this is not really feasible because:

- In practice, it is not possible to build a completely secure system [2].
- Cryptographic methods have their own problems. Passwords can be cracked, users can lose their passwords, and entire crypto-systems can be broken.
- Even a truly secure system is vulnerable to abuse by insiders who misuse their privileges.
- It has been seen that the relationship between the level of access control and user efficiency is an inverse one, which means that the stricter the mechanisms, the lower the efficiency becomes.

So that using cryptograph method or simply putting a firewall could not enough to solve problem of vulnerabilities. If there are attacks on a system, we would like to detect them as soon as possible and take appropriate action. This is essentially what an IDS does. An IDS does not usually take preventive measures when an attack is detected; it is a reactive rather than a pro-active agent. It plays the role of an informant rather than a police officer.

3.2.3 IDS flavors

IDS come in a variety of "flavors" and approach with the goal of detecting suspicious traffic in different ways. There are two basic types of IDS, Network based and host based IDS.

Network Intrusion Detection Systems (NIDS)

NIDS are placed at a strategic point or points within the network to monitor traffic to and from all devices on the network. Ideally you would scan all inbound and outbound traffic; however doing so might create a bottleneck that would impair the overall speed of the network.
This type of IDS is much more cost effective than Host IDS since several hosts can be monitored with one IDS, which also gives a better overall picture of the threat situation.

Another advantage is that the NIDS are separate entities in the network and can be tightly secured. Therefore, they are less prone to be compromised.

A downside to NIDS is the need for very efficient packet scanning, so that no threats are missed going through the network. With the increasing bandwidth and Internet usage on everyday basis, the amount of network traffic is making it tough for the IDS to keep up.

The other main disadvantage is that the hackers have discovered numerous methods to hide malicious network traffic; these methods are also referred to as IDS evasion techniques.

Host Intrusion Detection Systems (HIDS)

HIDS are run on individual hosts or devices on the network. A HIDS monitor the inbound and outbound packets only from the device and will alert the user or administrator of suspicious activity when detected.

Typical measurements a HIDS will monitor are logs, error messages, service and application rights and resource usage on a host.

In comparison with NIDS, HIDS are more accurate at detecting genuine intrusions because of their relatively lower rate of false-positives. Another advantage of HIDS over NIDS is that with the extended knowledge of the host it monitors, it can detect attacks, which may seem like a normal traffic to NIDS.

The disadvantages are significant however, as they reside on the monitored host. This limits the view of the IDS on the network topology, and it cannot detect attacks on other hosts. The result is that every host needs to have a HIDS, which in most cases would not be feasible and in some cases impossible.

3.2.4 Detection Methods

There are IDS that detect intrusions based on looking for specific signatures of known threats similar to the way anti-virus software typically detects and protects against malware. Also there are IDS that detect based on comparing traffic patterns against a baseline and looking for anomalies. The detection methods are divided into two categories as signature based and anomaly based intrusion detection model.

Signature Based

A signature based IDS, also called misuse detection, will monitor packets on the network and compare them against a database of signatures or attributes from known malicious threats. This is similar to the way most antivirus software detects malware. The issue is that there will be a lag between a new threat being discovered and the signature for detecting that threat being applied to your IDS. During that lag time your IDS would be unable to detect the new threat.

Anomaly Based

An IDS which is anomaly based will monitor network traffic and compare it against an established baseline. The baseline will identify what is “normal” for that network - what sort of bandwidth is generally used, what protocols are used, which port and device are generally gets connected - and alert the administrator or user when traffic is detected as anomalous, or significantly different from the baseline.

Behavior-based anomaly detection compares a profile of all allowed application behavior to actual traffic. Any deviation from the profile is flagged as a potential attack. It is commonly referred to as a positive security model because it seeks only to identify all "known good" behaviors and assumes that everything else is bad. Behavior anomaly detection has the potential to detect attacks of all kind – including "unknown" attacks on custom code.

Behavior anomaly detection can also lead to a high rate of false positives. For example, after a behavior profile is created, an application developer may change the application (a new URL, new parameter, etc.) without notifying the security team. In this case, behavior-based anomaly detection wrongly identifies access to these new parameters as potential attacks. Given the extreme complexity and dynamics of enterprise Web applications, the use of behavior anomaly detection as the sole basis for blocking attacks in real time is difficult without continuous tuning. This is why some systems combines anomaly detection

with signature detection and correlates all irregularities over time to accurately detect attacks without manual configuration or tuning.

3.2.5 Response Methods

There are two other categories such as Passive and Reactive type with respect to the response of IDS.

Passive IDS

A passive IDS simply detects and alerts. When suspicious or malicious traffic is detected an alert is generated and sent to the administrator or user and it is up to them to take action to block the activity in some way.

Reactive IDS

Reactive IDS will not only detect suspicious or malicious traffic and alert the administrator, but will take pre-defined proactive actions to respond to the threat. Typically this means that the IDS will block any further network traffic from the source IP address or user.

CHAPTER FOUR

SELF ORGANIZED MAP AND GENETIC ALGORITHM

4.1 Neural Network

An ANN is an information-processing paradigm that is inspired by the way biological nervous system, such as the brain, process information. The key element of this paradigm is the novel structure of the information processing system. It is composed of a large number of highly interconnected processing elements (neurons) working in unison to solve specific problems. ANNs, like people, learn by example. An ANN is configured for a specific application, such as pattern recognition or data classification, through a learning process. Learning in biological systems involves adjustments to the synaptic connections that exist between the neurons. This is true in ANNs as well.

ANN, with their remarkable ability to derive meaning from complicated or imprecise data, can be used to extract patterns and detect trends that are too complex to be noticed by either humans or other computer techniques. A trained neural network can be thought of as an “expert” in the category of information it has been given to analyze. This expert can then be used to provide projections given new situations of interest and answer "what if" questions. Other advantages include:

1. Adaptive learning: An ability to learn how to do tasks based on the data given for training or initial experience.

2. Self-Organization: An ANN can create its own organization or representation of the information it receives during learning time.

3. Real Time Operation: ANN computations may be carried out in parallel, and special hardware devices are being designed and manufactured which take advantage of this capability.

4. Fault Tolerance via Redundant Information Coding: Partial destruction of a network leads to the corresponding degradation of performance. However, some network capabilities may be retained even with major network damage.

ANN or shortly neural networks have been quite promising in offering solutions to problems, where traditional models have failed or are very complicated to build. Due to the non-linear nature of the neural networks, they are able to express much more complex phenomena than some linear modeling techniques.

Kohonen divides ANN into three categories [7]:

- Signal transfer networks
- State transition networks
- Competitive learning networks

In signal transfer networks, the input signal is transformed into an output signal. The signal traverses the network and undergoes a signal transformation of some kind. The network has a set of pre-defined basis functions, which are parametrized. The learning in these networks corresponds to changing parameters of these basis functions. Some examples of signal transfer networks are the MLP networks that are taught with error back propagation algorithm (BP) and RBF networks.

In state transition networks, the dynamic behavior of the network is essential. Given an input, the network converges to a stable state, which, hopefully, is a solution to a problem presented to it. Examples are Hopfield networks and Boltzmann machines.

In competitive learning networks, or self-organizing networks, all the neurons of the network receive the same input. The cells have lateral competition and the one with most activity “wins”. Learning is based on the concept of winner neurons. A representative example of a network based on competitive learning is SOM. The monograph by Kohonen [7] is the most complete book about this particular network model.

Learning in ANN is done in terms of adaptation of the network parameters. Network parameters are changed according to pre-defined equations called the learning rules. The learning rules may be derived from pre-defined error measures or may be inspired by biological systems. An example of an error measure in a network based on supervised learning could be the squared error between the output of the model and the desired output. This requires knowledge of the desired value for a given input. Learning rules are written so that the iterative learning process minimizes the error measure. Minimization might be performed by gradient descent optimization methods, for instance. In the course of learning, the residual between the model output and the desired output decreases and the model learns the relation between the input and the output.

Basically, there are two types of neural networks according to the learning strategies; supervised and unsupervised. Supervised learning which incorporates an external teacher, so that each output unit is told what its desired response to input signals ought to be. During the learning process, global information may be required. Unsupervised learning uses no external teacher and is based upon only local information. It is also referred to as self-organization, in the sense that it self-organizes data presented to the network and detects their emergent collective properties.

Next section focuses on a particular neural network model, SOM, which is used in this thesis work.

4.2 Self Organized Map

The SOM is one of the most popular neural network models. It belongs to the category of competitive learning networks. The SOM is based on unsupervised learning, which means that no human intervention is needed during the learning and that little need to be known about the characteristics of the input data. We could, for example, use the SOM for clustering data without knowing the class memberships of the input data.

The SOM algorithm provides a topology preserving mapping from the high dimensional space to map units. Map units, or neurons, usually form a two-dimensional lattice and thus the mapping is a mapping from high dimensional space onto a plane. The property of topology preserving means that the mapping preserves the relative distance between the points. Points that are near to each other in the input space are mapped to nearby map units in the SOM. The SOM can thus serves as a cluster analyzing tool of high-dimensional data. Also, the SOM has the capability to generalize. Generalizing capability means that the network can recognize or characterize inputs it has never encountered before. A new input is assimilated with the map unit to the winner node.

The way that SOMs go about in organizing themselves is by computation for representation of the samples. Neurons are also allowed to change themselves by learning to become more like samples in hopes of winning the next competition. It is this selection and learning process that makes the weights organize themselves into a map representing similarities.

So, with these two components (the sample and weight vectors), how can one order the weight vectors in such a way that they will represent the similarities of the sample vectors? This is accomplished by using the very simple algorithm shown here.

Initialize Map
For t from 0 to 1
    Randomly select a sample
    Get best matching unit
    Scale neighbors
    Increase t a small amount
End for

The first step in constructing a SOM is to initialize the weight vectors. Then select a sample vector randomly and search the map of weight vectors to find which weight best represents that sample. Since each weight vector has a location, it also has neighboring

weights that are closer to it. The weight vector that is chosen is rewarded by being able to become more like that randomly selected sample vector. In addition to this reward, the neighbors of that weight are also rewarded by being able to become more like the chosen sample vector. And then time (t) increase with a small amount and continue the process, because the number of neighbors and how much each weight can learn decreases over time. This whole process is then repeated a large number of times. Generaly the tasks in SOM algorithm can be divided in to the following three phases:

Initializing the Weights

There are a number of ways to initialize the weight vectors. The first one is just giving each weight vector random values for its data. Unfortunately, calculating SOMs is very computationally expensive, so there are some variants of initializing the weights so that samples that you know for a fact are not similar start off far away. This way you need less iteration to produce a good map and can save yourself some time.

Get Best Matching Unit

This is a very simple step, just go through all the weight vectors and calculate the distance from each weight to the chosen sample vector. The weight with the shortest distance is the winner. If there are more than one with the same distance, then the winning weight is chosen randomly among the weights with the shortest distance. There are a number of different ways for determining what distance actually means mathematically. The most common method is to use the Euclidean distance:

$$\sqrt{\sum_{i=0}^{n}(x_i - w_i)^2} \qquad 4.1$$

Where xi is the data value, w_i is the weight vector, I represents each attributes runs from o to n. n is the number of attributes (features).

Scale Neighbors

There are actually two parts to scale the neighboring weights: determining which weights are considered as neighbors and how much each weight can become more like the sample vector.

The neighbors of a winning weight can be determined using a number of different methods. Some use concentric squares, others hexagons and others use a Gaussian function where every point with a value above zero is considered a neighbor (fig 4.1).

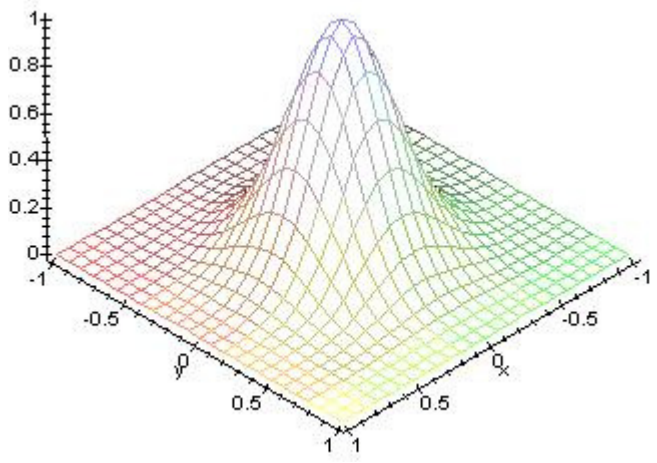

Fig 4.1 Plot of Gaussian Function

$$f := (x, y) \rightarrow e^{(-6.66667\sqrt{x^2+y^2})} \qquad 4.2$$

As mentioned previously, the amount of neighbors decreases over time. This is done so that samples can first move to an area where they will probably be. This process is similar to coarse adjustment followed by fine-tuning. The function used to decrease the radius of influence doesn't really matter as long as it decreases. Most of the time a linear function is used.

As the time progresses, the base goes towards the center, so there are less neighbors as time progresses. The initial radius is set really high, some value near the width or height of the map.

The second part to scaling the neighbors is the learning function. The winning weight is rewarded with becoming more like the sample vector. The neighbors also become more like the sample vector. An attribute of this learning process is that farther away the neighbor is from the winning vector, the less it learns. The rate at which a weight can learn decreases and can also be set to whatever you want. The new weight can be calculated:

$$W_{t+1} = W_t \times (1-t) + I_t \times t \qquad 4.3$$

Where Wt+1 is the new value, Wt is the current value and I is the sample vector , t is the time

In the first iteration, the best matching unit will get a t of 1 for its learning function, so the weight will then come out of this process with the same exact value as the randomly selected sample.

Just like the amount of neighbors, the amount a weight can learn also decreases with time. On the first iteration, the winning weight becomes the sample vector since t has a full range from 0 to 1. Then as time progresses, the winning weight becomes slightly more like the sample where the maximum value of t decreases. The rate at which a weight can learn falls of linearly. To depict this visually, in the previous plot, the amount a weight can learn is equivalent to how high the bump is at their location. As time progresses, the height of the bump will decrease. Adding this function to the neighborhood function will result in the height of the bump going down while the base of the bump shrinks.

So, once a weight vector is determined as winner, the neighbors of that weight vector are found and each of those neighbors' in addition to the winner's weight vectors are changed to become more like the sample vector.

The second algorithm that has been used in this paper is GA. The next section explains the basis of GA.

4.3 Genetic algorithm

GA are a part of evolutionary computing, which is a rapidly growing area of artificial intelligence.

As you can guess, GA is inspired by Darwin's theory of evolution. Simply said, problems are solved by an evolutionary process resulting in a best (fittest) solution (survivor) - in other words, the solution is evolved.

Rechenberg introduced this evolutionary computing in 1960s in his work as “Evolution strategies”. His idea was then further developed by other researchers. GA were invented by John Holland and developed by him and his students and colleagues. This lead to Holland's book titled “Adoption in Natural and Artificial Systems”, published in 1975.

In 1992 John Koza has used GA to evolve programs to perform certain tasks. He called his method as “genetic programming” (GP).

4.3.1 Biological Background

Chromosome

All living organisms consist of cells. In each cell there is the same set of chromosomes. Chromosomes are strings of DNA and serve as a model for the whole organism. A chromosome consists of genes, blocks of DNA. Each gene encodes a particular protein. Basically, it can be said that each gene encodes a trait, for example color of eyes. Possible settings for a trait (e.g. blue, brown) are called alleles. Each gene has its own position in the chromosome. This position is called locus.

Complete set of genetic material (all chromosomes) is called genome. Particular set of genes in genome is called genotype. The genotype is with later development after birth base for the organism's phenotype, its physical and mental characteristics, such as eye color, intelligence etc.

Reproduction

During reproduction, recombination (or crossover) first occurs. Genes from parents combine to form a whole new chromosome. The newly created offspring can then be mutated. Mutation means that the elements of DNA are a bit changed. These changes are mainly caused by errors in copying genes from parents.

The fitness of an organism is measured by success of the organism in its life (survival).

4.3.2 Genetic Algorithm

GA is inspired by Darwin's theory of evolution. Solution to a problem solved by genetic algorithms uses an evolutionary process (it is evolved).

GA begins with a set of solutions (represented by chromosomes) called population. Solutions from one population are taken and used to form a new population. This is motivated by a hope, that the new population will be better than the old one. Solutions which are then selected to form new solutions (offspring) are according to their fitness - the more suitable they are the more chances they have to reproduce.

This is repeated until some condition (for example, number of populations or improvement of the best solution) is satisfied.

The following is an outline of the basic GA

1. **[Start]** Generate random population of *n* chromosomes (suitable solutions for the problem)
2. **[Fitness]** Evaluate the fitness *f(x)* of each chromosome *x* in the population
3. **[New population]** Create a new population by repeating the following steps until the new population is complete
    1. **[Selection]** Select two parent chromosomes from a population according to their fitness (the better fitness, the bigger chance to get selected)
    2. **[Crossover]** With a crossover probability, cross over the parents to form new offspring (children). If no crossover was performed, offspring is the exact copy of parents.
    3. **[Mutation]** With a mutation probability, mutate new offspring at each locus (position in chromosome).
    4. **[Accepting]** Place new offspring in the new population
4. **[Replace]** Use new generated population for a further run of the algorithm
5. **[Test]** If the end condition is satisfied, **stop**, and return the best solution in current population
6. **[Loop]** Else, Go to step **2**

As can be seen, the outline of the Basic GA is very general. There are many parameters and settings that can be implemented differently in various problems.

The first question to ask is how to create chromosomes and what type of encoding to choose. It is then that one address Crossover and Mutation; the two basic operators of GA. Encoding, crossover and mutation are introduced below.

The next question is how to select parents for crossover. This can be done in many ways, but the main idea is to select the better parents (best survivors) in the hope that the better parents will produce better offspring.

One may think that generating populations from only two parents may cause to loose the best chromosome from the last population. This is true, and so elitism is often used. This means, that at least one of a generation's best solution is copied without changes to a new population, so the best solution can survive to the succeeding generation.

4.3.3 Operators of GA

As one can see from the genetic algorithm outline, the crossover and mutation are the most important parts of the genetic algorithm. Mainly these two operators influence the performance of the model. Before applying the two operators, the chromosome should be encoded first.

Encoding of a Chromosome

A chromosome should in some way contain information about solution that it represents. The most used way of encoding is a binary string. A chromosome then could look like this:

Chromosome 1 1101100100110110

Chromosome 2 1101111000011110

Each chromosome is represented by a binary string. Each bit in the string can represent some characteristics of the solution. Another possibility is that the whole string can represent a number.

Of course, there are many other ways of encoding. The encoding depends mainly on the solved problem. For example, one can encode directly integer or real numbers; sometimes it is useful to encode some permutations and so on.

Crossover

After we have decided what encoding we will use, we can proceed to crossover operation. Crossover operates on selected genes from parent chromosomes and creates new offspring. The simplest way how to do that is to choose randomly some crossover point and copy everything before this point from the first parent and then copy everything after the crossover point from the other parent.

Crossover can be illustrated as follows. Where | is the crossover point.

Chromosome 1 11011 | 00100110110

Chromosome 2 11011 | 11000011110

Offspring 1 11011 | 11000011110

Offspring 2 11011 | 00100110110

There are other ways to make crossover, for example we can choose more crossover points. Crossover can be quite complicated and depends mainly on the encoding of chromosomes. Specific crossover made for a specific problem can improve performance of the genetic algorithm.

Mutation

After a crossover is performed, mutation takes place. Mutation is intended to prevent falling of all solutions in the population into a local optimum of the solved problem. Mutation operation randomly changes the offspring resulted from crossover. In case of binary encoding we can switch a few randomly chosen bits from 1 to 0 or from 0 to 1. Mutation can be then illustrated as follows:

Original offspring 1 1101111000011110

Original offspring 2 1101100100110110

Mutated offspring 1 1100111000011110

Mutated offspring 2 1101101100110110

Like crossover, the technique of mutation depends mainly on the encoding of chromosomes. For example when we are encoding permutations, mutation could be performed as an exchange of two genes.

4.3.4 Parameters of GA

Crossover probability, mutation probability and population size are the basic parameters of GA.

Crossover probability: how often crossover will be performed. If there is no crossover, offspring are exact copies of parents. If there is crossover, offspring are made from parts of both parent's chromosome. If crossover probability is 100%, then all offspring are made by crossover. If it is 0%, whole new generation is made from exact copies of chromosomes from old population.

Crossover is made in hope that new chromosomes will contain good parts of old chromosomes and therefore the new chromosomes will be better. However, it is good to leave some part of old population survives to next generation.

Mutation probability: how often parts of chromosome will be mutated. If there is no mutation, offspring are generated immediately after crossover (or directly copied) without any change. If mutation is performed, one or more parts of a chromosome are changed. If mutation probability is 100%, whole chromosome is changed, if it is 0%, nothing is changed.

Mutation generally prevents the GA from falling into local extremes. Mutation should not occur very often, because then GA will in fact change to random search.

Population size: how many chromosomes are in population (in one generation). If there are too few chromosomes, GA has few possibilities to perform crossover and only a small part of search space is explored. On the other hand, if there are too many chromosomes, GA slows down. Research shows that after some limit (which depends mainly on encoding and the problem) it is not useful to use very large populations because it does not solve the problem faster than moderate sized populations.

4.3.5 Population Selection

Chromosomes are selected from the population to be parents for crossover. The problem is how to select these chromosomes. According to Darwin's theory of evolution the best ones survive to create new offspring. There are many methods in selecting the best chromosomes. Examples are roulette wheel selection, rank selection, steady state selection and some others.

Roulette Wheel Selection

Parents are selected according to their fitness. The better the chromosomes are, the more chances to be selected they have. Imagine a roulette wheel where all the chromosomes in the population are placed. The size of the section in the roulette wheel is proportional to the value of the fitness function of every chromosome - the bigger the value is, the larger the section is.

A marble is thrown in the roulette wheel and the chromosome where it stops is selected. Clearly, the chromosomes with bigger fitness value will be selected more times.

Rank Selection

The previous type of selection will have problems when there are big differences between the fitness values. For example, if the best chromosome fitness is 90% of the sum of all fitnesses then the other chromosomes will have very few chances to get selected.

Rank selection ranks the population first and then every chromosome receives fitness value determined by this ranking. The worst will have the fitness 1, the second worst 2 etc. and the best will have fitness N (number of chromosomes in population).

Now all the chromosomes have a chance to be selected. However this method can lead to slower convergence, because the best chromosomes do not differ so much from other ones.

Steady-State Selection

This is not a particular method of selecting parents. The main idea of this type of selection from the new population is that a big part of chromosomes can survive to next generation.

The steady-state selection GA works in the following way. In every generation, a few good (with higher fitness) chromosomes are selected for creating new offspring. Then some bad (with lower fitness) chromosomes are removed and the new offspring is placed in their place. The rest of population survives to new generation.

Elitism

When creating a new population by crossover and mutation, we have a big chance, that we will loose the best chromosome. Elitism is the name of the method that first copies the best chromosome (or few best chromosomes) to the new population. Elitism can rapidly increase the performance of GA, because it prevents a loss of the best found solution.

CHAPTER FIVE

MODEL BUILDING

5.1 The Proposed system Model

The task of developing IDS involves the following tasks; Data preparation, input preprocessing, feature selection and intrusion detection. The data preparation task includes data collection. Input pre-processing task covers data transformation, data normalization and data preparation for testing, training and validation. The extracted features can either be important attributes which are useful for decision making or it can also be a set of attribute which generates noise on the detector. In order to filter unwanted attributes, a feature selection module is used. After selecting the best features, the next task will be detection of intrusions. The intrusion detection model is an integrated type of detector which integrates SOM with GA. The intrusion detection block diagram is shown in figure 5.1. Each of the models is explained below.

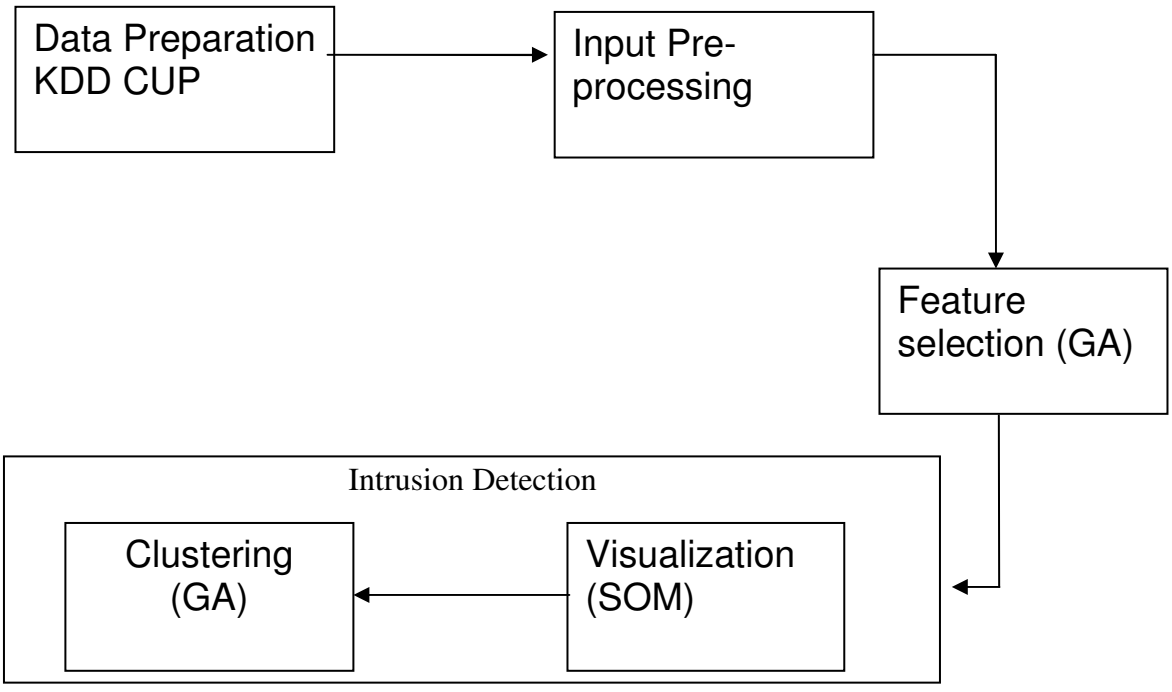

Fig 5.1 Block diagram of the proposed intrusion detection system

5.2 Data Preparation

The data preparation task needs high effort to collect different scenario data to evaluate the system. Since, it is difficult to simulate and collect different intrusion data, the current work used KDD dataset which is a full-fledged intrusion data, which was developed to survey and evaluate research works in intrusion detection. Further details and the characteristics of KDD dataset are described in detail in section 2.5.

5.3 Input Preprocessing

Once the most appropriate raw input data has been selected, it must be preprocessed; otherwise, the neural network will not produce accurate results. The decisions made in this phase of development are critical to the performance of a network [3].

Transformation and normalization are two widely used preprocessing methods. Transformation involves manipulating textual data to numeric value, while normalization is performed on each data input to distribute the data evenly and scale it into an acceptable range for the network. Knowledge of the domain is important in choosing the appropriate preprocessing method to highlight underlying features in the data, which can increase the network's ability to learn the association between inputs and outputs.

This benefits the network in two ways. First, it has been given useful information at a reasonable level of detail; and second, by smoothing the data, the noise entering the network has been reduced. This is important because noise can obscure the underlying relationships. The only disadvantage is that worthwhile information might be lost in an effort to reduce the noise, but this tradeoff always exists when attempting to smooth the noisy data.

In normalizing the data, the goal is to ensure that the statistical distribution of values for each net input and output is roughly uniform. In addition, the values should be scaled to match the range of the input data. This means that along with any other transformations

performed on network inputs, each input should be normalized as well. In this work, linear scaling of data is used. At the very least, data must be scaled into the range used by the input data in the neural network. This is typically the range from zero to one. A linear scaling requires that the minimum and maximum values associated with each attribute to be found. Let's call these values as Dmin and Dmax respectively. The input range required for the network must also be determined. Let's assume that the input range is from Imin to Imax. The formula for transforming each data value D to an input value I is:

$$I = \mathrm{I}_{\min} + \frac{(\mathrm{I}_{\max} - \mathrm{I}_{\min}) * (\mathrm{D} - \mathrm{D}_{\min})}{(\mathrm{D}_{\max} - \mathrm{D}_{\min})} \qquad 5.1$$

5.4 Feature Selection using Genetic Algorithm

The choice of features to represent the patterns affects several aspects of the pattern recognition problem such as accuracy, required learning time, and the necessary number of samples. In this way, the selection of the best discriminative features plays an important role when constructing classifiers. However, this is not a trivial task especially when dealing with lot of features. In order to choose a subset of the original features by eliminating irrelevant and redundant ones, automated feature selection algorithm has been used.

In this work, unsupervised GA based feature selection method is used. The objective of unsupervised feature selection method is to search for a subset of features that best uncovers “natural” groupings (clusters) according to some criterion. This is a difficult task because the clusters have to be defined to find the subset of features that maximize the performance criterion. The problem is made more difficult when the number of clusters is unknown beforehand which happens in most real-life situations.

The feature selection approach often combines different optimization objectives into a single objective function. The main drawback of this kind of strategy lies in the difficulty of exploring different possibilities of trade-offs among objectives.

In order to overcome this kind of problem, some authors [5] proposed the use of a multi-objective GA to perform feature selection. They used number of features and a validity index to measure the quality of the clusters as multi-criterion optimization function. GA offers a particularly attractive approach to solve this kind of problem since they are generally quite effective in rapid global search of large, nonlinear, and poorly understood spaces.

Non-dominated Sorting Genetic Algorithm (NSGA) method is used in this work. This method is proposed by Srinivas and Deb in [9] which deals with multi-objective optimization. The objective is to find a set of non-dominant solutions which contains the more discriminate features and pertinent number of clusters. Two criteria are used to guide the search: minimization of the number of features and minimization of a validity index that measures the quality of clusters.

Let there be n objects, N_i, where i=1,…,n. Suppose each object is characterized by p feature values. In other words, an object is represented by a p-dimensional vector. Hence, there are $N_i = (N_{ij})$, where i=1, …,n; j = 1, … , p. Suppose n objects are classified into k clusters, C_a, where a = 1, … ,k. Each C_a contains some objects. Let Z_a be the centers of C_a respectively. If $Z_a = (Z_{aj})$, where a=1,…k; j=1,…,p then the centroid calculated using

$$Z_a = \frac{\sum_i N_i}{|C_a|}, N_i \in C_a \qquad 5.2$$

As stated before, two criteria are used: minimization of a validity index and minimization of the number of features. In order to measure the quality of clusters, the within-cluster scatter and between-cluster separation have been widely used by various researchers. Some researchers used two objective functions to compute these measurements. Some have combined the two measurements into one using Davies-Bouldin (DB) index (called validity index) proposed by Davies et al in [2]. To make such indices suitable for this problem, they must be normalized by the number of selected features. This is due to the fact that they are based on geometric distance metrics and are therefore not directly applicable here because they are biased by the dimensionality of the space, which is variable in feature selection problems. The normalized DB index is considered in this experiment.

DB index is a function of the ratio of the sum of within-cluster scatter to between-cluster separation. The scatter within the i^{th} cluster is computed as follows

$$S_i = \frac{1}{|C_i|} \sum_{N \in C_i} \{ \| N - Z_i \| \} \qquad 5.3$$

and the distance between clusters C_i and C_j is defined as

$$d_{ij} = \{ \| Z_i - Z_j \| \} \qquad 5.4$$

S_i is the average Euclidean distance of the vectors N in cluster C_i with respect to its centroid Z_i. d_{ij} is the Euclidean distance between the centroids Z_i and Z_j of the clusters C_i and C_j respectively. Subsequently, the maximum ration of with in cluster and between clusters, R_i is computed

$$R_i = \max_{j, j \neq i} \{ \frac{S_i + S_j}{d_{ij}} \} \qquad 5.5$$

The DB index is then defined as

$$I_{DB} = \frac{1}{D}\frac{1}{K}\sum_{i=1}^{K} R_i \qquad 5.6$$

Where K corresponds to the number of selected clusters and D is the number of selected features. The objective is to achieve proper clustering by minimizing the DB index.

DB Index is one of the objective functions to select best attributes for clustering problem. The second objective function is minimizing the features. These two objective functions are used to evaluate the fitness function for populations in each generation. But, it is difficult to compare chromosomes using two or more objectives. One method is using weighting to get single objective function, but it is difficult to select the weightings and consider the two objective functions properly. The second method is NSGA. It is used in this work.

A general multi-objective optimization problem consists of a number of objectives and is associated with a number of inequality and equality constraints. Solution to a multi-objective optimization problem can be expressed mathematically in terms of non dominated points, i.e., a solution is dominant over another only if it has superior performance in all criteria. A solution is said to be Pareto-optimal if it cannot be dominated by any other solution available in the search space. NSGA feature selection algorithm works in the following way;

1. Generate random population of *n* chromosomes. Each chromosome is a string of zero's and one's. A bit value of 1 in the chromosome represents that the corresponding feature is included in the specified subset and a value of 0 indicates that the corresponding feature is not included in the subset.

2. Evaluate the fitness *f(x)* of each chromosome *x* in the population using NSGA. As described above, the GA uses DB Index and number of features

as a fitness function. These two fitness values rank the chromosomes using NSGA. The flowchart for NSGA shown below in fig 5.2.

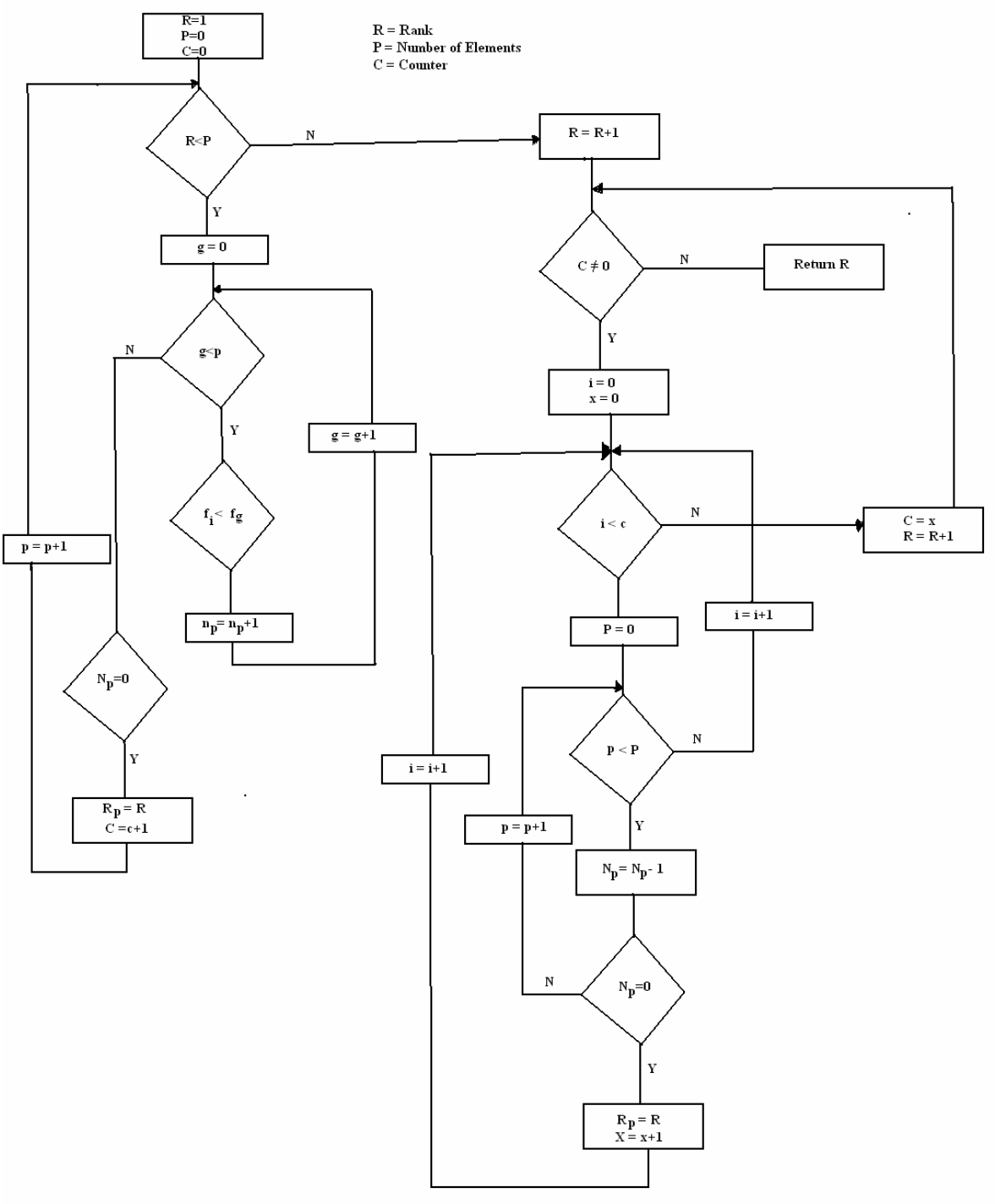

Fig 5.2 Flow chart of NSGA

Where Rp is new fitness rank, Np is level counter and i, x, g and p is loop counters.

3. Create a new population by repeating the following steps until the new population is complete

    1. Select two parent chromosomes from a population according to their fitness. The better fitness, the bigger chance to be selected.

    2. Random point is selected between zero to number of features to fix the point of crossover. With a crossover probability, crossover the parents by swapping the chromosome string of 0 to p of one with p to number of feature of the other and vise versa to form new offspring (children). If no crossover was performed, offspring is the exact copy of parents.

    3. For each allele of the chromosome with a mutation probability mutate applied by flipping the bit.

    4. Place new offspring in the new population

7. Replace the old with the new generation by keeping only the best chromosome to satisfy elitism with elitism rate.

8. Go to step 2

5.5 Self-Organized Map Algorithm

The SOM is an unsupervised neural network model for analyzing and visualizing high dimensional data. It maps non-linear statistical relationships between high-dimensional input data into simple geometric relationships on a regular, usually two-dimensional grid.

SOM is a mapping from the ensemble of data vector $\{x_1, x_2, \ldots, x_n\}$ to an ordinary two dimensional array of neurons N_i. Neurons are arranged in rectangular configuration. Each

neuron in the lattice corresponds to a reference vector that is characterized by p features. The number of reference vectors will depend on how large a lattice the researcher wants. In this work, fifty by fifty lattice i.e. 2500 nodes are used. The algorithm establishes a local order among the set of reference vectors. In this way, the closeness between two reference vectors is solely dependent on how close the corresponding nodes are on the two dimensional array.

Let there be *N* neurons in two dimensional grids. Suppose each neuron is characterized by *p* feature values. There is also training data X that characterized by p feature values. Hence, there are X_{ij} *and* N_{ij} , where i=1, …,n; j = 1, … , p. The SOM algorithm works in the following way:

Initialization: Initializing the map is the first step. Each neuron is initialized by taking sample randomly from input vectors.

Best Match Unit: For each training data, the distance between the data sample X and each neuron N is calculated. The shortest distance is selected as a winner, p. A neighborhood set of array points surrounding neuron p is denoted as Np; in practice, Gaussian function is used to determine the neighborhood. The neighborhood set decrease with increase in number of iterations and with decrease in the value of 't', where 't' is calculated as
t = 1- 1/ number of iterations. So, Np is represented as Np(t) in this work. The above stated idea is represented in Fig 5.3, where $t_1 > t_2 > t_3$

Learning process: Defining $M_i(t)$ to be the position of the i^{th} weight vector at iteration t. Based on the input vector X, the closest weight vector is identified as M_p. The weight vectors of the winner and the corresponding neighborhoods are adjusted according to equation 5.7.

$$\mathrm{M_i}(t+1) = \mathrm{M_i}(t) + (\mathrm{X} - \mathrm{M_i}(t)) * \tau(p,i,i) \qquad 5.7$$

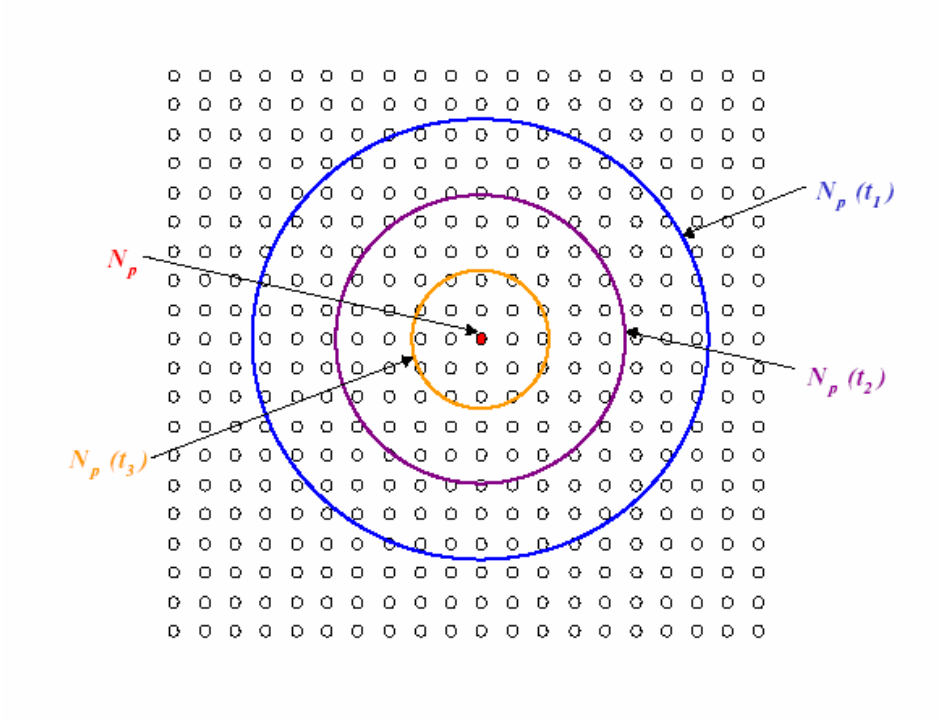

Fig 5.3 Determine neighborhood of the winner node using Gaussian function

The quantity (X - $M_i(t)$) is the component-wise difference between the input vector and the i^{th} weight vector and $\tau(p,i,t)$ is a neighborhood function acting on the array of neighbouring nodes. Neighbourhood function, $\tau(p,i,t)$ uses Gussian function. It is calculated using equation 5.8, where d is the distance from the winner vector to M_p. t is iteration time that controls the learning rate of SOM

$$\tau(p,i,t) = \frac{e^{(-d\ /0.15)}}{4t+1} \qquad 5.8$$

5.6 Clustering using Genetic Algorithm

In this section, a novel GA for the clustering problem is described, although there have been some researchers proposing different GAs for clustering. It is a modification of Krishna and Murty's algorithm [18]. The notations used in this algorithm are described first. Let there be n neurons, N_i, where i=1,…,n. Suppose, each neuron is characterized by p feature values. In other words, a neuron is represented by a p-dimensional vector. Hence, there are $Ni = (N_{ij})$, where i=1, …,n; j = 1, … , p. Suppose n objects are classified into m clusters, Ca, where a = 1, … ,m. Each C_a contains some neurons. Let S_a is the center of C_a. If $S_a = (s_{a1}, s_{a2}, \ldots, s_{ap})$, then S_a is calculated as

$$S_a = \frac{\sum_i N_i}{|C_a|}, N_i \in C_a \qquad 5.9$$

Where the summation is over all i's such that $N_i \in C_a$ (*i.e.* the number of neurons in C_a).

Initial Step: In the initial step, a population of M strings is generated. All the strings are of length m. Each string is represented as $\{I_1, I_2, \ldots I_m\}$. The I_i, which is randomly generated from N_i where i = 1, … , n is defined as the initial center for each cluster.

The Procedures of Generating Clusters: After generating the chromosomes of initial step, let $I = \{I_1, I_2, \ldots, I_m\}$ be one such chromosome. So, let the initial centers of clusters $S_a = I_a$ for a = 1, 2, … , m. The generation of the clusters proceeds as follows:

Step 1: The neurons in $\{N_1, N_2, \ldots, N_n\}$ are taken one by one and the distance between the taken neuron N_i and each S_a is calculated using the following distance function

$$d(N_i, S_a) = \left(\sum_{q=1}^{p} (N_{iq} - S_{aq})^2 \right)^{1/2} \qquad 5.10$$

Where $N_i = (N_{i1}, N_{i2}, \ldots, N_{ip})$ and $S_a = (S_{a1}, S_{a2}, \ldots, S_{ap})$. $Ni \in C_a$ if $d(N_i, S_a) < d(N_i, S_k)$ for all k such that $1 < k < m$ and $k \neq a$. After all neurons in $\{N_1, N_2, \ldots, N_n\}$ have been considered, the centers $S_1, S_2, \ldots, S_m$ of m clusters $C_1, C_2, \ldots, C_m$ are computed with Equation 5.9.

Step 2: Assign each neurons N_i, $i = 1, \ldots, n$ to cluster C_a, $a = 1, \ldots, m$ again. If $d(N_i, S_a) < d(N_i, S_k)$ then $N_i \in C_a$ for all k such that $1 < k < m$ and $k \neq a$.

Step 3: The new cluster centers are recomputed when all neurons are assigned to different clusters. The new cluster centers are calculated with Equation 5.9. If the objects in the clusters do not change anymore, the process of generating clusters terminates; otherwise go back to Step 2. The clusters are generated after terminating the generating process. The fitness function F of each chromosome used in this research to evaluate the capability of clustering is as follows:

$$F = f \times \left(\frac{1}{SSW} \right) \qquad 5.11$$

$$SSW = \sum_{N_i \in C_a} d(N_i, S_a)^2 \qquad 5.12$$

Where $d(N_i, S_a)$ is Euclidean distance defined in equation 5.10, and $f = 0$ if there is only one neuron in one cluster; otherwise, $f = 1$. It is expected to collect some neurons in the cluster.

In our experiments, there are few cases in which there is only one object in one cluster, but (1/SSW) of those few cases is large enough to be selected as the final solution.
To avoid this problem, the f is used.

Reproduction Phase: After the calculation of f for each chromosome in the population, the reproduction phase is implemented.

Crossover Phase: A random number r from uniform distribution within [1, m] is generated to decide which piece of the string is to be interchanged. If there any two genes of the chromosome are the same after interchanging, the crossover process repeats until genes of the chromosome are different. For each selected pair of chromosomes, the crossover operation is done with probability p_c, which is the so-called crossover rate.

Mutation Phase: In the mutation phase, the chromosomes in the population will be chosen within probability p_m, which is called the mutation rate. If one chromosome is to mutate, a random number r from uniform distribution within [1, m] is generated to determine which gene will mutate. The mutated gene will be changed to a random number from uniform distribution within [1, n]. If there are any two genes of the chromosome the same after mutating, the mutation process repeats until all genes of the chromosome are different.

Selection Phase and Stopping Criterion: The set of steps from an initial population to a new population is called a *generation.* The aim of the selection phase is to carry the best chromosomes from the previous generation to the next. The best half of the chromosomes from the initial and the new population are copied to the next generation in the selection phase. There is no stopping criterion that ensures the convergence of GA to an optimal situation [8, 24]. In this research, a fixed number of iterations are executed and the best chromosome is selected as the optimal solution.

CHAPTER SIX

IMPLEMENTATION AND RESULTS

6.1 Introduction

This chapter discusses about implementation results like input processing, feature selection and visualization. It finally discusses the result of the IDS in terms of false positive and detection rates.

6.2 Implementation

The algorithms used for input preprocessing, feature selection, visualization and clustering are discussed in detail in the previous chapter. The algorithms are implemented using Java programs (See Appendix II, III, IV and V for the code). KDD dataset is used to train and test the different algorithms used in this work and the results are presented here below.

6.2.1 Input Preprocessing

Input preprocessing includes data transformation and normalization. The KDD dataset has forty one input attributes and one output attribute (refer to Appendix I, table A) of which four are of text type. The textual data of the KDD dataset are protocol type, service, flag and attack type. These textual data are numerated to generate the necessary pattern. Numericals of each textual attribute are presented in Appendix I in table C, D, E and F.

In the data normalization phase, the numerical values of the attributes are adjusted to be within certain range. In this work, the values of each attribute are normalized between zero and one using the formula 5.1.The original KDD dataset before pre-processing the input looks like as shown in table 6.1.

Table 6.1 Sample of the orginal KDD dataset

0,tcp,http,SF,181,5450,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,8,8,0.00,0.00,0.00,0.00,1.00,0.00,0.00,9,9,1.00,0.00,0.11,0.00,0.00,0.00,0.00,0.00,normal.

After performing transformation and normalization, it is converted to uniform numeric data range between zero and one as shown in table 6.2.

Table 6.2 Sample dataset after transformation and normalization

0,1,0.1384615,0,0.0000015,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,1,0,0,1,1,1,0,1,0,0,0,0,0,1

6.2.2 Data Preparation

Then, in the data preparation phase, the KDD testing and training datasets are used in this work. The training set is comprised of 61.36% of the data and the remaining 38.63% is used for testing the system. Both the datasets contain different type of attacks. The testing dataset includes new type of attacks (see Appendix I table B). The available number of data in KDD dataset under each of the attack type category is given in the table 2.1.

6.2.3 Feature Selection Result

The genetic feature selection algorithm is used to generate a binary string that has forty one input attributes. For the representation, ‘1’ is used to indicate important features and ‘0’ is used for representing attributes that will not be used for training and testing the system.

The initial set of chromosomes can be selected randomly depending on the size of the population or all alleles of each chromosome in the initial population can be assigned ‘1’. In this implementation, both scenarios are checked and the same result observed. The sample for both types of scenarios of initial chromosome is presented below in table 6.3.

Table 6.3 Different scenarios of GA initial chromosome

Scenarios	Initial chromosome
Random	10001101101110000110111110110110111110110
All one's	11111111111111111111111111111111111111111

As clearly indicated in chapter five, for the GA's input, a random population will first be selected and then the required reproduction operators such as crossover and mutation are applied on the generation. The parameters used in GA are given in table 6.4 and adopted from [34].

The crossover applied is single point crossover and the point of crossover is selected randomly. In one generation, two chromosomes crossover with the probability of $p_{c,}$ crossover probability. After crossover, one of the allele of a chromosome may mutate to new value with the probability of $p_{m,}$ mutation probability. The allele is selected randomly from the forty-one zero-one string and toggle the original value. These GA operators are applied on all the selected chromosomes.

At the end of each generation, the fitness value (DB index and number of features) is calculated for each chromosome and sorted according to the fitness value using NSGA algorithm. Before selecting the best chromosomes from the current population, one percent of the best chromosomes of old population are moved to the new population.

Table 6.4 List of parameters used for the feature selection

No	*Parameter*	*Value*	*Description*
1	Selection probability	0.90	The probability of selecting the best chromosome
2	Crossover probability	0.80	The probability of applying crossover
3	Mutation probability	0.01	The probability of applying mutation
4	Elitism Rate	1%	Keeping 1% of the old generation

5	Terminating condition	1000	Number of generations

The other parameter used in GA is population size. The test is performed on different values of population size to get the best result. But, the results as indicated in the table 6.5 are almost similar for most of the sizes.

Table 6.5 Results of the feature selection algorithm

Population size	*Chromosome*	*Fitness Value*
10	11111111111011111111111111111110101011111	0.050496757
20	11111111101011111111101111111110101011111	0.049394194
30	11111111101011111111101111111110101011111	0.049394194
50	11111111101011111111101111111110101011111	0.049394194

The fitness values obtained are the same with the population size of greater than twenty. The best chromosome at the end of the stopping criterion shows only six unwanted features or features that have insignificant effect in the clustering process. These six attributes are 10, 12, 22, 32, 34 and 36th attribute of KDD dataset (See Appendix I, table A for the name of each attribute). Since there is no change in the selection of the best features / alleles and fitness value, there is no need to further test the system with a population size of greater than fifty.

6.2.4 SOM Result

SOM model is used to map large dimensional data to two-dimensional lattice to observe the clustering of each group. The two dimensional lattice is composed of 50x50 neurons (2500 nodes). In this work, SOM is trained and tested with 2 different set of input attributes. The first set (dataset 1) constitutes the original 41 features of KDD dataset and the second set (dataset 2) comprises of 35 features, the output of the above mentioned

feature selection algorithm. Initially, random values are assigned for each neuron of SOM model, each characterized by 41 or 35 features.

The results corresponding to both the datasets are presented in section 6.3, with a detailed discussion of the results corresponding to dataset 2 in the following sections.

As clearly stated in section 5.4, the distance between each input vector and neuron is calculated and the smallest distance is awarded to update the weight of the node. Not only the winner node but also the weight vectors of the neighborhood nodes are updated. The radius of the neighbors and the corresponding amount of weight update decreases with time. Time decreases from one to zero when number of iteration increases. To find the neighbors, a Gaussian function is used in this work.

All the parameters used in SOM is listed in table 6.6

Table 6.6 List of the parameters used for the SOM model

No	Parameter	Value	Description
1	Size of the Map	50X50	The dimension of the two-dimensional map. That is, the number of nodes in the grid is 2500 and each is characterized by 41 attributes
2	Radius	60	Initial radius of influence (should start with a large neighborhood)
3	Number of Iterations	1000	There is no other criterion to stop the iteration. Maximum number of iterations is the stopping criterion

The input for the SOM network is the connection data that is characterized by thirty-five features (dataset 2) and the output is neuron weighting. The SOM model is trained using the entire training set and it is iterated till stopping criterion. Weight sample of one of the SOM's output node after 1000 iterations is shown below in table 6.7.

Table 6.7 Weight sample of a node

1.401298464324817E-45,1.6562596928301796E-25, 0.1692308783531189, 0.3999999761581421, 1.1593818497555664E-31, 1.401298464324817E-45, 0.0, 1.401298464324817E-45, 0.0, 0.0, 1.401298464324817E-45, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.035419680178165436, 0.9999993443489075, 1.0, 1.401298464324817E-45, 1.401298464324817E-45, 0.14867159724235535, 0.0599580779671669, 1.401298464324817E-45, 1.0, 0.06990597397089005, 2.013312559896633E-25, 1.0, 1.0, 1.401298464324817E-45, 1.401298464324817E-45

Fig 6.1 shows the final graphical output of SOM model, where the 35-dimensional input data is reduced to 2-dimensional output data. The graphical output further demonstrates clustering efficiency of SOM model. The graph shows 5 distinct clusters, of which green color represents the normal connection data and other colors represent four different types of attacks.

However, the clusters show a lot of false mapping of the patterns. In order to fine tune the clustering, the SOM output is provided as input for the GA model as the next step.

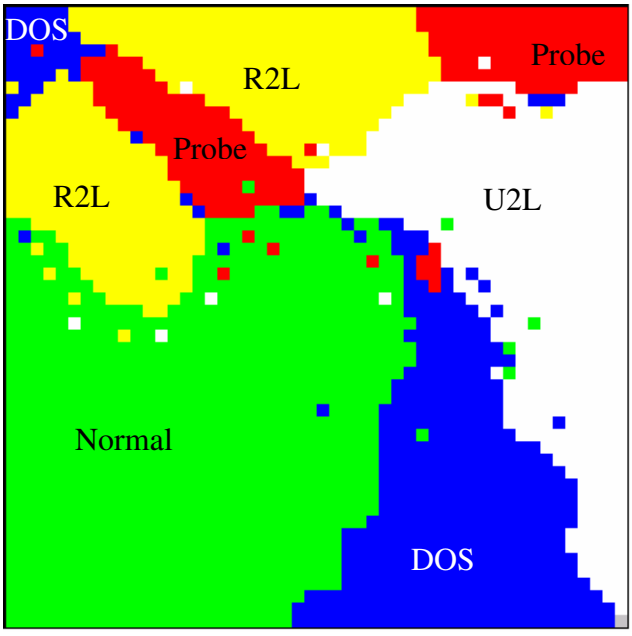

Fig 6.1 Graphical output of Feature Selection + SOM system

6.2.5 Clustering Result of GA

For GA clustering algorithm that is described in section 5.4, the parameters listed in table 6.8 are used. The output of SOM (2500 neurons, each consist of a weight vector of length 35) is given as input for the GA. Initially, GA selects strings of centroids randomly from the input population (i.e. 2500 vectors). It recalculates the centroid by assigning all the neurons in each cluster. And then crossover and mutation operators are applied on the generation for a number of iterations. Since there is no stopping criterion in this clustering problem, 1000 iterations or generations are applied and the results are observed.

Table 6.8 list of parameters for GA clustering model

No	Parameter	Value	Description
1	Selection probability	0.90	The probability of selecting the best chromosome
2	Crossover probability	0.80	The probability of applying crossover
3	Mutation probability	0.01	The probability of applying mutation
4	Elitism rate	0.01	Keeping 1 present of the old generation
5	Number of population	100	Population size
6	Number of generation	1000	Maximum number of generation
7	Number of neurons	2500	Total number of neurons or nodes

In this clustering algorithm, five clusters are considered. One is for normal connection and the other four are for different types of attack categories as stated in appendix I table F. The final centroid of one of the clusters is presented below as a sample.

Table 6.9 Sample data for one of the five cluster centers

3.682555E-8,2.797231E-4,0.1525597,0.2648102,6.355545E-8,2.8041238E-6,0.0,3.899797E-9,0.0,5.563645E-29,0.007905403,0.0,0.0,1.4E-45,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.012308926,0.01233133,0.014094444,0.97096956,0.9692063,0.49073228,0.11140214,0.067904964,0.66151386,0.37639266,0.24231087,0.014234214,0.014116358,0.95926356,0.93729854

The result of SOM+GA shows 94.3 % detection rate and 2.93% false alarm rate. From the result, one can understand that the introduction of GA model clusters the attacks and normal connections with a better detection rate and false alarm rate. The graphical output of the SOM+GA clustering model is shown in fig 6.2. Though there are some false mappings, the number is much lesser than the output of SOM model. So, GA improved the clustering and reduced the number of false mapping of the patterns.

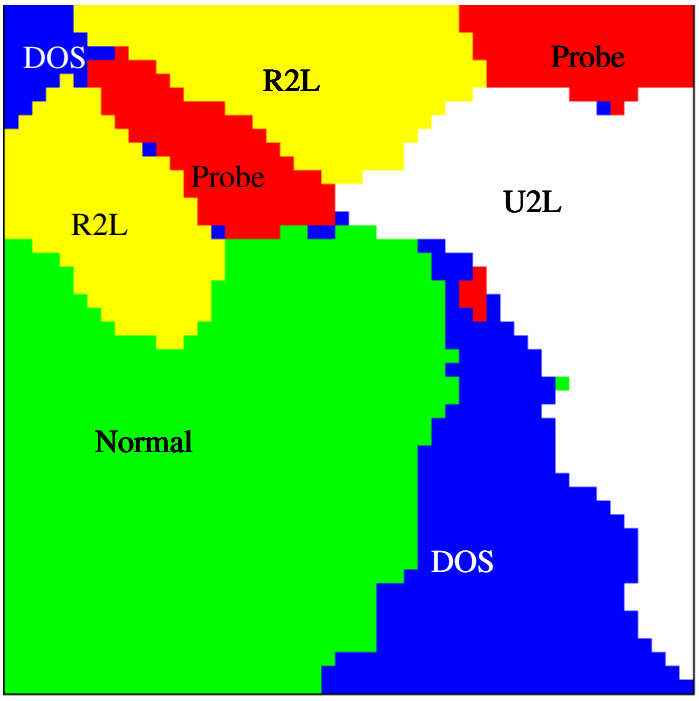

Fig 6.2 Graphical output of the Feature Selection + SOM + GA system

From the graphical output one simply observes the same cluster (same color) in different locations.This may be due to the various attack types under the same category (refer appendix I, Table B) which may have different input characteristics.

6.3 Performance Result

Performance of the detector is evaluated in terms of the false alarm rate and detection rates which are calculated as follows. False alarm rate, called false position rate, is connections considered as attack by the system but those are not attacks.

$$FalsePositiveRate = \frac{NumberofFalsePositive}{TotalNumberofNormalConnections} \quad 6.1$$

$$DetectionRate = \frac{NumberofFalseNegative}{TotalNumberofAttachConnections} \quad 6.2$$

The number of false positive is the number of normal connections, but labeled as attacks. The number of False Negative is the number of attack connections, but labeled as normal connections.

The results for each possible scenario are observed as follows.

Table 6.6 Result of the integrated model

No	Models	Detection Rate In Percentage	False Alarm Rate in Percentage
1	SOM	92.21	3.81
2	SOM + GA	93.85	3.02
3	Feature selection + SOM	92.09	3.42
4	Feature selection + SOM + GA	94.3	2.93

Without the feature selection approach, SOM + GA combination is better than SOM approach by 1.64% of detection rate and 0.79 % of false alarm rate. With feature selection, the output of SOM+GA is better compared to SOM by 2.21% of detection rate and 0.49% of false alarm rate. The improvement from 93.85% to 94.3% in detection rate is mainly due to the application of feature selection approach. Thus, the results of this work proved the feasibility of integrated approach in IDS so as to reduce the percentage of false alarms and increase the percentage of detection rate.

CHAPTER SEVEN

CONCLUSIONS AND RECOMMENDATIONS

7.1 Introduction

This chapter summarizes the work by highlighting the major conclusions obtained. Besides, it points out future research directions that could be done as a continuation of this research.

7.2 Conclusion

The purpose of this study was to design and test IDS using visualization and integration techniques. Various intrusion detection algorithms are reviewed and the one that is meant to produce a better result is selected for the purpose of this research. As a result, an integrated method is developed to improve the performance of the detection rate and to reduce the false alarm rate.

For the purpose of reducing and visualizing large amounts of data, SOM is used. The research demonstrated the same.

SOM and GA are integrated to realize IDS. The integration has brought about 94.3% of detection rate with 2.93% of false alarm rate which is a better achievement as compared to the previous research endeavor in [22]. From the experimental results of paper [22], their scheme archived about 90% detection rate and less than 5 % false alarm rate.

7.3 Recommendations

The research attempted to integrate SOM and GA for intrusion detection. The following works can be done as a continuation of this attempt.

1. The algorithms are tested on an ideal setting. They have to be tested on an actual setting to make the necessary modifications and make it usable.
2. Conduct similar researches with a different technique and compare the results.
3. Launching a big project to develop efficient full-fledged IDS.
4. Explore different techniques to further improve the feature selection as it is helpful to achieve an enhanced result.
5. The dataset used has a small amount of U2R, R2L and Probe type of attacks. Around 75% of the data used is of Dos attack. Even if that is usual in the real case, using proportional amount of data for each cluster may lead to better results.
6. The integrated SOM and GA clustering model do not have a feedback mechanism. Adding a feedback mechanism to such IDS for the purpose of improving performance is an area that needs further exploration.

Bibliography

1. KDD CUP 1999 Data. The UCI KDD Archive Information and Computer Science, University of California, Irvine. http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html
2. D. L. Davies and W. Bouldin. A cluster separation measure. IEEE PAMI, 1929
3. Lou Mendelsohn, Preprocessing Data for Neural Network, http://www.tradertech.com/preprocessing_data.asp
4. Declan McGrath, Introduction to Network Security
5. Y.S. Kim, W.N. Street, and F. Menczer. Feature selection in unsupervised learning via evolutionary search. In Proc. 6th ACM SINGKDD International Conference on Knowledge Discovery and Data Mining, 2000.
6. Barton P Miller, David Koski, Cjin Pheow Lee, Vivekananda Maganty, Ravi Murthy, Ajitkumar Natarajan, Jeff Steidl. Fuzz Revisited: A Re-examination of the Reliability of UNIX Utilities and Services. Computer Sciences Department, University of Wisconsin, 1995.
7. Teuvo Kohonen. *Self-Organizing Maps*. Springer, Berlin, Heidelberg, 1995.
8. C. A. Murthy and N.Chowdhury, "In search of optimal clusters using genetic algorithms," Pattern Recognition Letters, 1996
9. N.Srinivas and K. Deb. Multiobjective optimization using nondominated sorting in genetic algorithms. Evolutionary computing,1995.
10. Stephen E. Hansen and E. Todd Atkins. Automated system monitoring and nofication with swatch. In Proceedings of the seventh Systems Administration Conference,Montery, CA, November 1993
11. Ulf Lindqvist and P.A. Porras. Detecting computer and network misuse through the Production-Based Expert System Toolset. In Proceeding of the 1999 Symposium of Security and Privacy, Oakland, California, May 19999
12. A. Mounji, Languages and Tools for Rule-Based Distributed Intrution Detection. PhD thesis, Facult'e Universitaier Notre de la Paix de Namur, Belgium, September 1997
13. James Cannady, Artificial Neural Networks for Misuse Detection, School of computer and Information Sciences, Nova Southeastern University

14. Wenke Lee, Salvatore J. Stolfo, and Kui W. Mok, A data mining framework for building intrusion detection models. In Proceedings of the 1999 IEEE Symposium on Security and Privacy,1999
15. Harold S. Javitz and Alfonso Valdes. The NIDES statistical component: Description and justification. Technical report, SRI Computer Science Labratory, Menlo Park, CA, March 1994
16. Dae-Ki Kang, Doug Fuller , and Vasant Honavar, Learning Classifiers for Misuse Detection Using a Bag of System Calls Representaion, Artificial Intelligence Lab, Iowa State Universtiy
17. Anup K. Ghosh and Aaron Schwartzbard, A study in using neural networks for anomaly and misuse detection, USENIX association,1999
18. C. Ko, M. Ruschitizka, and K. Levitt. Execution monitoring of security-critical programs in distributed systes: A specification-based approach. In Proceedings of the 1997 IEEE Symposium on Secuirty and Privacy, Oakland, California, 1997
19. Luc Girardin, AN eye on network intruder - administrator shootouts, In Proceedings of the workshop on intrusion detection and network monitoring, Santa Clara, California, April 1999.
20. Soon Tee Teoh, Kwan-Liu Ma, S. Felix Wu, Xiaoliang Zhao. A Visual Technique for Internet Anomaly Detection, University of California
21. Soon Tee Teoh, T.J. Jankun-Kelly, Kwan-Liu Ma, S.Felix Wu. Visual Data Analysis for Detecting Flaws and Intruders in Computer Network Systems, University of California
22. Chavat Jirapummin Naruemon Wattanapongsakorn and Prasert Kanthamanon, Hybrid Neural Networks for Intrusion detection System, King Mongkut's University of Technology Thonburi.
23. Tom Fawcett and Foste Provost. Combining data mining and machine learning for effective user profiling. In Proceedings of the Second Interantional Conference on Knowledge Discovery and Data Mining(KDD-96),1996
24. D.E. Goldberg, Genetic Algorithms: Search, Optimization and Machine Learning. Reading, MA: Ad-dison-Wesley, 1989.
25. Binh Viet Nguyen, Self Organizing Map (SOM) for Anomaly Detection

26. Wei Li, Mississippi State Univeristy, using Genetic Algorithm for Network Intrusion detection
27. Dipankar Dasgupta and Fabio A. Gonzalez, An Intelligent Decision Support System for Intrusion detection and Respons
28. Gary Stein, Bing CHen, Annie S. Wu and Kien A. Hua. Decision Tree Classifier For Network Intrusion Detection With GA-based Feature Selection, Univeristy of Central Florida,
29. William H. Hsu, Michael Welge, Jie Wu, and Ting-Hao Yang, Genetic Algorithms for Selection and Partitioning of Attributes in Large-Scale Data Mining Problems
30. Gisele L. Pappa, Alex A. Freitas, Celso A.A Kaestner,A Multi objective Genetic Algorithm for Attribute Selection
31. Mahehkumar Sabhanani, Gursel Serpen. Application of Machine Learning Algorithms to KDD Intrusion Detection Dataset within Misuse Detection Context, EECS Department, University of Toledo,USA
32. K. Krishna and N. M. Murty, ”Genetic K means algorithm,” *IEEE Trans. Syst., Man, Cybern. B*, vol.
33. Robert F. Erbacher, Kenneth L. Walker, Deborah A. Frincke, Misuse Detection in Large-Scale Systems
34. M. Morita, R. Sabourin, F. Bortolozzi and C. Y. Suen, Unsupervised Feature Selection Using Multi-Objective Genetic Algorithms for Handwritten word Recognition, Ecole de Technologie Superieure, Canada

Appendix I KDD Dataset

Table A Attribute list of the KDD 99 dataset

Attribute No	Attribute Name	Data Type
1	Duration	Continuous
2	protocol_type	Symbolic
3	Service	Symbolic
4	Flag	Symbolic
5	src_bytes	Continuous
6	dst_bytes	Continuous
7	Land	Continuous
8	Wrong_fragment	Continuous
9	Urgent	Continuous
10	Hot	Continuous
11	num_failed_logins	Continuous
12	Logged_in	Continuous
13	num_compromised	Continuous
14	root_shell	continuous
15	su_attempted	continuous
16	num_root	continuous
17	num_file_creations	continuous
18	num_shells	continuous
19	num_access_files	continuous
20	num_outbound_cmds	continuous
21	is_host_login	continuous
22	is_guest_login	continuous
23	Count	continuous
24	srv_count	continuous
25	serror_rate	continuous
26	srv_serror_rate	continuous
27	rerror_rate	continuous
28	srv_rerror_rate	continuous
29	same_srv_rate	continuous
30	diff_srv_rate	continuous
31	srv_diff_host_rate	continuous
32	Dst_host_count	continuous
33	Dst_host_srv_count	continuous
34	Dst_host_same_srv_rate	continuous
35	Dst_host_diff_srv_rate	continuous
36	Dst_host_same_src_port_rate	continuous
37	Dst_host_srv_diff_host_rate	continuous
38	Dst_host_serror_rate	continuous
39	Dst_host_srv_serror_rate	continuous
40	Dst_host_rerror_rate	continuous
41	Dst_host_srv_rerror_rate	continuous

42	Attack type	symbolic

Table B Attack and category list of the KDD 99 dataset

Attack	Category	Avalability of attcks
Apache2	Dos	Testing
Back	Dos	Training and Testing
Buffer-overflow	U2r	Training and Testing
Ftp_Write.	R21	Training and Testing
Guess-passwd.	R21	Training and Testing
Httptunnel.	R21	Testing
Imap.	R21	Training and Testing
Ipsweep.	Probe	Training and Testing
Land.	Dos	Training and Testing
Loadmodule	U2r	Training and Testing
Mailbomb.	Dos	Testing
Mscan.	Probe	Testing
Multihop.	R21	Training and Testing
Named	R21	Testing
Neptune.	Dos	Training and Testing
Nmap.	Probe	Training and Testing
Normal.	Normal	Training and Testing
Perl.	U2r	Training and Testing
phf.	R21	Training and Testing
Pod.	Dos	Training and Testing
Portsweep.	Probe	Training and Testing
Processtable.	Dos	Testing
Ps.	U2r	Testing
Rootkit	U2r	Training and Testing
Saint.	Probe	Testing
Satan.	Probe	Training and Testing
Sendmail.	R21	Testing
Smurf.	Dos	Training and Testing
Snmpgetattack.	R21	Testing
Snmpguess.	R21	Testing
Spy	R21	Training and Testing
Sqlattack.	U2r	Testing
Teardrop.	Dos	Training and Testing
Udpstom.	Dos	Testing
Warezclient.	R21	Training and Testing
Warezmaster.	R21	Training and Testing
Worm.	R21	Testing
Xlock.	R21	Testing
Xsnoop.	R21	Testing

Xterm.	U2r	Testing

Table C Enumeration of the alphanumeric Service feature

Value	Assigned
http	0
Smtp	1
Finger	2
Domain u	3
Auth	4
telnet	5
ftp	6
Eco i	7
Ntp u	8
Ecr i	9
Other	10
Private	11
Pop 3	12
ftp data	13
Rje	14
Time	15
Mtp	16
Link	17
Remote job	18
Gopher	19
Ssh	20
Name	21
Whois	22
Domain	23
Login	24
Imap4	25
Daytime	26
Ctf	27
nntp	28
Shell	29
IRC	30
Nnsp	31
http 443	32
Exec	33
Printer	34
Efs	35
Courier	36
Uucp	37

Continued on next page

Value	Assigned
Klogin	38
Kshell	39
Echo	40
Discard	41
Systat	42
Supdup	43
Isso tsap	44
Hostnames	45
Csnet ns	46
Pop 2	47
Sunrpc	48
Uucp path	49
Netbios ssn	51
Netbios dgm	52
Sql net	53
Vmnet	54
Bgp	55
Z39 50	56
Ldap	57
Netstat	58
Urp i	59
X11	60
Urp i	61
Pm dump	62
Tftp u	63
Tim i	64
Red i	65

Table D Enumeration of the alphanumeric Protocol feature

Value	Assigned
Tcp	0
Udp	1
icmp	2

Table E Enumeration of the Bro Flag feature

Value	Assigned
SF	0
S1	1
REJ	2
S2	3
S0	4
S3	5
RSTO	6
RSTR	7
RSTOS0	8
OTH	9
SH	10

Table F Enumeration of the attack category

Value	Assigned
Normal	0
DOS	1
Probe	2
R2L	3
U2L	4

Appendix II Preprocessing Code

```
// required imports
import java.io.*;

class ipp {
  static final int Imin=0;
  static final int Imax=1;
  public static void main(String[] args)
  {
    // constant for the space between the line number and the content
    final String SPACER = "\t";
    float[] min= new float[42];
    float[] max= new float[42];
    float[] row= new float[42];

    // declare the file stream objects to use
    BufferedReader inFile = null;
    PrintWriter outFile = null;

  if (args.length != 3)
    {
      System.err.println("The correct number of commandline arguments weren't given.");
      System.err.println("Usage:   java FileNumber <sourcefile> <numeratedfile>
<normalizedfile>");

      System.exit(1);
    }
    try
    {
       inFile = new BufferedReader(new FileReader(args[0]));
    }
    catch (FileNotFoundException e)
    {
       System.err.println("The file " + args[0] + " wasn't found.");

      System.exit(2);
    }

    try
    {
      outFile = new PrintWriter(new BufferedWriter(new FileWriter(args[1])));
    }
    catch (IOException e)
    {
        System.err.println("An I/O error occured when trying to open the file");
        System.err.println("for writing.  The program cannot continue.");
        System.exit(3);
    }
    boolean finished = false;  // flag used to signify end of file
```

```
String inData = null;      // the line of data read from the source
int lineNo = 1;            // the current line number
String [] attribute;
// while the finished flag isn't set...
while (!finished)
{
  try
  {
    inData = inFile.readLine();
  }
  catch (IOException e)
  {
    System.err.println("An I/O error occured when trying to read the file.");
    System.err.println("The program cannot continue.");
    System.exit(3);
  }
if (inData != null)
  {
   try{
    attribute=inData.split(",");
 //Numberate the second attribute

    if (attribute[1].equalsIgnoreCase("tcp"))
      attribute[1]="0";
    else if (attribute[1].equalsIgnoreCase("udp"))
      attribute[1]="1";
    else if (attribute[1].equalsIgnoreCase("icmp"))
      attribute[1]="2";
    else
      System.out.println("On line " + lineNo + " " + attribute[1] + " unkown category");

  //Numerate the third attribute
    if ( attribute[2].equalsIgnoreCase("http"))
      attribute[2]="0";
    else if ( attribute[2].equalsIgnoreCase("smtp"))
      attribute[2]="1";
    else if ( attribute[2].equalsIgnoreCase( "finger"))
      attribute[2]="2";
    else if ( attribute[2].equalsIgnoreCase( "domain_u"))
      attribute[2]="3";
    else if ( attribute[2].equalsIgnoreCase( "auth"))
      attribute[2]="4";
    else if ( attribute[2].equalsIgnoreCase( "telnet"))
      attribute[2]="5";
    else if ( attribute[2].equalsIgnoreCase( "ftp"))
      attribute[2]="6";
    else if ( attribute[2].equalsIgnoreCase( "eco_i"))
      attribute[2]="7";
    else if ( attribute[2].equalsIgnoreCase( "ntp_u"))
      attribute[2]="8";
    else if ( attribute[2].equalsIgnoreCase( "ecr_i"))
```

```
  attribute[2]="9";
else if ( attribute[2].equalsIgnoreCase( "other"))
  attribute[2]="10";
else if ( attribute[2].equalsIgnoreCase( "private"))
  attribute[2]="11";
else if ( attribute[2].equalsIgnoreCase( "pop_3"))
  attribute[2]="12";
else if ( attribute[2].equalsIgnoreCase( "ftp_data"))
  attribute[2]="13";
else if ( attribute[2].equalsIgnoreCase( "rje"))
  attribute[2]="14";
else if ( attribute[2].equalsIgnoreCase( "time"))
  attribute[2]="15";
else if ( attribute[2].equalsIgnoreCase( "mtp"))
  attribute[2]="16";
else if ( attribute[2].equalsIgnoreCase( "link"))
  attribute[2]="17";
else if ( attribute[2].equalsIgnoreCase( "remote_job"))
  attribute[2]="18";
else if ( attribute[2].equalsIgnoreCase( "gopher"))
  attribute[2]="19";
else if ( attribute[2].equalsIgnoreCase( "ssh"))
  attribute[2]="20";
else if ( attribute[2].equalsIgnoreCase( "name"))
  attribute[2]="21";
else if ( attribute[2].equalsIgnoreCase( "whois"))
  attribute[2]="22";
else if ( attribute[2].equalsIgnoreCase( "domain"))
  attribute[2]="23";
else if ( attribute[2].equalsIgnoreCase( "login"))
  attribute[2]="24";
else if ( attribute[2].equalsIgnoreCase( "imap4"))
  attribute[2]="25";
else if ( attribute[2].equalsIgnoreCase( "daytime"))
  attribute[2]="26";
else if ( attribute[2].equalsIgnoreCase( "ctf"))
  attribute[2]="27";
else if ( attribute[2].equalsIgnoreCase( "nntp"))
  attribute[2]="28";
else if ( attribute[2].equalsIgnoreCase( "shell"))
  attribute[2]="29";
else if ( attribute[2].equalsIgnoreCase( "IRC"))
  attribute[2]="30";
else if ( attribute[2].equalsIgnoreCase( "nnsp"))
  attribute[2]="31";
else if ( attribute[2].equalsIgnoreCase( "http_443"))
  attribute[2]="32";
else if ( attribute[2].equalsIgnoreCase( "exec"))
  attribute[2]="33";
else if ( attribute[2].equalsIgnoreCase( "printer"))
  attribute[2]="34" ;
```

```
else if ( attribute[2].equalsIgnoreCase( "efs"))
  attribute[2]="35";
else if ( attribute[2].equalsIgnoreCase( "courier"))
  attribute[2]="36" ;
else if ( attribute[2].equalsIgnoreCase( "uucp"))
  attribute[2]="37";
else if ( attribute[2].equalsIgnoreCase( "klogin"))
  attribute[2]="38";
else if ( attribute[2].equalsIgnoreCase( "kshell"))
  attribute[2]="39";
else if ( attribute[2].equalsIgnoreCase( "echo"))
  attribute[2]="40";
else if ( attribute[2].equalsIgnoreCase( "discard"))
  attribute[2]="41";
else if ( attribute[2].equalsIgnoreCase( "systat"))
  attribute[2]="42";
else if ( attribute[2].equalsIgnoreCase( "supdup"))
  attribute[2]="43";
else if ( attribute[2].equalsIgnoreCase( "iso_tsap"))
  attribute[2]="44";
else if ( attribute[2].equalsIgnoreCase( "hostnames"))
  attribute[2]="45";
else if ( attribute[2].equalsIgnoreCase( "csnet_ns"))
  attribute[2]="46";
else if ( attribute[2].equalsIgnoreCase( "pop_2"))
  attribute[2]="47";
else if ( attribute[2].equalsIgnoreCase( "sunrpc"))
  attribute[2]="48";
else if ( attribute[2].equalsIgnoreCase( "uucp_path"))
  attribute[2]="49";
else if ( attribute[2].equalsIgnoreCase( "netbios_ns"))
  attribute[2]="50";
else if ( attribute[2].equalsIgnoreCase( "netbios_ssn"))
  attribute[2]="51";
else if ( attribute[2].equalsIgnoreCase( "netbios_dgm"))
  attribute[2]="52";
else if ( attribute[2].equalsIgnoreCase( "sql_net"))
  attribute[2]="53";
else if ( attribute[2].equalsIgnoreCase( "vmnet"))
  attribute[2]="54";
else if ( attribute[2].equalsIgnoreCase( "bgp"))
  attribute[2]="55";
else if ( attribute[2].equalsIgnoreCase( "Z39_50"))
  attribute[2]="56";
else if ( attribute[2].equalsIgnoreCase( "ldap"))
  attribute[2]="57";
else if ( attribute[2].equalsIgnoreCase( "netstat"))
  attribute[2]="58";
else if ( attribute[2].equalsIgnoreCase( "urh_i"))
  attribute[2]="59";
else if ( attribute[2].equalsIgnoreCase( "X11"))
```

```
      attribute[2]="60";
    else if ( attribute[2].equalsIgnoreCase( "urp_i"))
      attribute[2]="61";
    else if ( attribute[2].equalsIgnoreCase( "pm_dump"))
      attribute[2]="62";
    else if ( attribute[2].equalsIgnoreCase( "tftp_u"))
      attribute[2]="63";
    else if ( attribute[2].equalsIgnoreCase( "tim_i"))
      attribute[2]="64";
    else if ( attribute[2].equalsIgnoreCase( "red_i"))
      attribute[2]="65";
    else
     System.out.println("On line " + lineNo + " " + attribute[2] + " unkown category");

  //Numerate the fourth attribute
   if ( attribute[3].equalsIgnoreCase( "SF"))
      attribute[3]="0";
   else if ( attribute[3].equalsIgnoreCase( "S1"))
      attribute[3]="1";
   else if ( attribute[3].equalsIgnoreCase( "REJ" ))
      attribute[3]="2";
   else if ( attribute[3].equalsIgnoreCase( "S2"))
      attribute[3]="3";
   else if ( attribute[3].equalsIgnoreCase( "S0" ))
      attribute[3]="4";
   else if ( attribute[3].equalsIgnoreCase( "S3" ))
      attribute[3]="5";
   else if ( attribute[3].equalsIgnoreCase( "RSTO" ))
      attribute[3]="6";
   else if ( attribute[3].equalsIgnoreCase( "RSTR"))
      attribute[3]="7";
   else if ( attribute[3].equalsIgnoreCase( "RSTOS0" ))
      attribute[3]="8";
   else if ( attribute[3].equalsIgnoreCase( "OTH" ))
      attribute[3]="9";
   else if ( attribute[3].equalsIgnoreCase( "SH"))
      attribute[3]="10";
   else
    System.out.println("On line " + lineNo + " " + attribute[3] + " unkown category");

  //Numerate the output field
   if ( attribute[41].equalsIgnoreCase( "multihop."))
    attribute[41]="3"; //"1" ;
   else if ( attribute[41].equalsIgnoreCase( "neptune."))
    attribute[41]="1";  //"3" ;
   else if ( attribute[41].equalsIgnoreCase( "nmap." ))
    attribute[41]="2"; //"4";
   else if ( attribute[41].equalsIgnoreCase( "normal."))
    attribute[41]="0"; //"5" ;
   else if ( attribute[41].equalsIgnoreCase( "perl."))
    attribute[41]="4"; //"6" ;
```

```
        else if ( attribute[41].equalsIgnoreCase( "phf."))
         attribute[41]="3"; //"7" ;
        else if ( attribute[41].equalsIgnoreCase( "pod." ))
         attribute[41]="1"; //"8";
        else if ( attribute[41].equalsIgnoreCase( "portsweep."))
         attribute[41]="2"; // "9";
        else if ( attribute[41].equalsIgnoreCase( "rootkit."))
         attribute[41]="4"; //"12" ;
        else if ( attribute[41].equalsIgnoreCase( "satan."))
         attribute[41]="2"; //"14" ;
        else if ( attribute[41].equalsIgnoreCase( "smurf."))
         attribute[41]="1"; //"16" ;
        else if ( attribute[41].equalsIgnoreCase( "spy." ))
         attribute[41]="3"; //"19";
        else if ( attribute[41].equalsIgnoreCase( "teardrop." ))
         attribute[41]="1"; //"21";
        else if ( attribute[41].equalsIgnoreCase( "warezclient."))
         attribute[41]="3"; //"23" ;
        else if ( attribute[41].equalsIgnoreCase( "warezmaster."))
         attribute[41]="3"; //"24" ;
        else if ( attribute[41].equalsIgnoreCase( "back."))
         attribute[41]="1"; //"30";
        else if ( attribute[41].equalsIgnoreCase( "buffer_overflow."))
         attribute[41]="4"; //"31";
        else if ( attribute[41].equalsIgnoreCase( "ftp_write."))
         attribute[41]="3"; //"32";
        else if ( attribute[41].equalsIgnoreCase( "guess_passwd."))
         attribute[41]="3"; //"33";
        else if ( attribute[41].equalsIgnoreCase( "imap."))
         attribute[41]="3"; //"35";
        else if ( attribute[41].equalsIgnoreCase( "ipsweep."))
         attribute[41]="2"; //"36";
        else if ( attribute[41].equalsIgnoreCase( "land."))
         attribute[41]="1"; //"37";
        else if ( attribute[41].equalsIgnoreCase( "loadmodule."))
         attribute[41]="4"; //"38";
        else
         System.out.println("On line " + lineNo + " " + attribute[41] + " unkown category");
      //Convert the attribute string values to float value
        for(int loop=0;loop<attribute.length;loop++)
         {
          try{
          row[loop] = Double.valueOf(attribute[loop].trim()).floatValue();
            }
          catch(NumberFormatException e){
                                    System.err.println("On line " + lineNo + " attribute " +
attribute[loop] + " NumberFormatException error");
                    }
                  catch(ArrayIndexOutOfBoundsException e){
                                    System.err.println("On line " + lineNo + " attribute " +
attribute[loop] + " NumberFormatException error");
```

```
                    }
                }
  //Initialize min and max
                if(lineNo==1)
                {
                  for(int loop=0;loop<attribute.length;loop++)
                    {
                      min[loop]=row[loop];
                      max[loop]=row[loop];
                    }
                }
  //Keep min for max for each attribute
                 minimum(row,min);
                 maximum(row,max);
                 inData=join(attribute,",");
                 outFile.println(inData);
                 lineNo++;
               }
               catch(NullPointerException e)
                 {
                 System.err.println("On line " + lineNo + " NullPointerException error");
                 }
             }
             else
             {
               finished = true;
             }
          }
          try
          {
            inFile.close();
          }
          catch (IOException e)
          {
            System.err.println("There was an error closing the destination file.");
            System.exit(3);
          }
          System.out.println(lineNo);
          outFile.close();
           for (int loop=0,n=min.length; loop<n; loop++)
             {
                System.out.println("(" + min[loop] + "," + max[loop] + ")");
               }

          //Normalize the data from second file
               try
          {
             inFile = new BufferedReader(new FileReader(args[1]));
          }
          catch (FileNotFoundException e)
          {
```

```
        System.err.println("The file " + args[0] + " wasn't found.");

       System.exit(2);
     }
     try
     {
       outFile = new PrintWriter(new BufferedWriter(new FileWriter(args[2])));
     }
     catch (IOException e)
     {
         System.err.println("An I/O error occured when trying to open the file");
         System.err.println("for writing.  The program cannot continue.");
         System.exit(3);
     }
   finished=false;
    lineNo=0;
   while (!finished)
     {
       try
       {
         inData = inFile.readLine();
       }
       catch (IOException e)
       {
         System.err.println("An I/O error occured when trying to read the file.");
         System.err.println("The program cannot continue.");
         System.exit(3);
       }
     if (inData != null)
       {
        try{
             attribute=inData.split(",");

        for(int loop=0;loop<attribute.length;loop++)
         {
        try{
          row[loop] = Double.valueOf(attribute[loop].trim()).floatValue();
            }
          catch(NumberFormatException e){
                                        System.err.println("On line " + lineNo + " atrribute " +
attribute[loop] + " NumberFormatException error");
                       }
                    catch(ArrayIndexOutOfBoundsException e){
                                        System.err.println("On line " + lineNo + " attribute " +
attribute[loop] + " NumberFormatException error");
}                    }
          //Normalized
        for(int loop=0;loop<row.length;loop++)
          {
          Double nValue= new Double(normalized(row[loop],min[loop],max[loop]));
          attribute[loop]= nValue.toString();
```

```
            }
             inData=join(attribute,",");
             outFile.println(inData);
             lineNo++;

            }
          catch(NullPointerException e)
            {
             System.err.println("On line " + lineNo + " NullPointerException error");
            }
         }
         else
         {
           finished = true;
}}}
public static String join(String[] input, String delim)
{
            StringBuffer result = new StringBuffer();

            for(int loop=0,n=input.length;loop<n;loop++)
            {
              result.append(input[loop]);
              if (loop < n-1)
              {
               result.append(delim);
              }
  }
return result.toString();
}
public static void minimum(float[] attr, float[] min)
{
            for(int loop=0;loop<attr.length;loop++)
              if(attr[loop]<min[loop])
       min[loop]=attr[loop];
}
public static void maximum(float[] attr, float[] max)
{
            for(int loop=0;loop<attr.length;loop++)
              if(attr[loop]>max[loop])
       max[loop]=attr[loop];
}

public static float normalized(float value, float min, float max)
{
  if (max==0) max=1;

  return (Imin + (Imax-Imin)*(value-min)/(max-min));

}}
```

Appendix III Feature Selection Code

```
// required imports
import java.io.*;
import java.util.Random;
class chromosome{
public int size;
 public boolean[] alleles;
 public float fitness;
private Random Rand = new Random();
 public chromosome(int size){
            this.fitness=0;
            this.size=size;
        alleles=new boolean[size];
            for ( int j=0; j<size; j++){
                        //if(j==9||j==11||j==21||j==31||j==33||j==35)
     //if(j==6||j==8||j==10||j==13||j==14||j==15||j==16||j==17||j==18||j==19||j==20 )

//if(j==6||j==8||j==10||j==12||j==13||j==14||j==15||j==16||j==17||j==18||j==19||j==20||j==21||j==23||
j==25||j==26||j==28||j==30||j==40)

//if(j==8||j==10||j==13||j==14||j==15||j==16||j==17||j==18||j==19||j==20||j==21||j==23||j==26||j==28
||j==30||j==40)
      //  alleles[j] = false;//Rand.nextBoolean();
      //else
      //  alleles[j] = true;
      alleles[j] = Rand.nextBoolean();
     }
            }

 public void copy(chromosome chrom){
            for(int i=0;i<chrom.size;i++)
             alleles[i]=chrom.alleles[i];
            size=chrom.size;
            fitness=chrom.fitness;
            }

 public void flip(int bit){
   alleles[bit]=!(alleles[bit]);
  }
}

class genetic{

    private int noFeatures;
    private int popSize;
            private int noCluster;

            private Random Rand = new Random();
```

```
public genetic(int noFeatures,int popSize,int noCluster)
{
  this.noFeatures=noFeatures;
  this.popSize=popSize;
  this.noCluster=noCluster;

}

public chromosome tournamentSelect ( chromosome[] population, float prob){

   int first = Rand.nextInt(popSize);
   int second= Rand.nextInt(popSize);

   while( first == second  ){
       second=Rand.nextInt(popSize);
   }

   float invert = Rand.nextFloat();

   System.out.println("Tournament selection " + first + " = " + population[first].fitness + "
and " + second + " = " + population[second].fitness);

   chromosome temp = new chromosome(noFeatures);

   if ( population[first].fitness < population[second].fitness ){
     if ( invert < prob )
       temp.copy(population[first]);

      else
       temp.copy(population[second]);



   } else{
    if ( invert < prob )
       temp.copy(population[second]);
    else
       temp.copy(population[first]);
   }
  return temp;
}


public boolean crossover ( chromosome first, chromosome second, float prob ){

  float p = Rand.nextFloat();
  if ( p > prob ) {return false;}

  int fbit = Rand.nextInt(first.size - 2);

  while( fbit == 0){
```

```
                fbit = Rand.nextInt(first.size - 2);
            }
  System.out.println("Crossover starts : " + fbit);
  for (int i=fbit; i<first.size; i++){
    boolean tmp    = first.alleles[i];
    first.alleles[i]  = second.alleles[i];
    second.alleles[i] = tmp;
  }
  return true;
}

public boolean mutate ( chromosome indi, float prob ){
  for (int i=0; i<indi.size; i++){
          float p=Rand.nextFloat();
     if( p < prob ){
      indi.flip(i);
      System.out.println(i + " : ");
      }
   }
  return true;
}


private float [] floatValue(String [] attribute)
{
      float[] floatValue;
      floatValue= new float[attribute.length];
      for (int loop=0;loop<attribute.length; loop++)
       {

        try{
           floatValue[loop] = Double.valueOf(attribute[loop].trim()).floatValue();
          }
        catch(NumberFormatException e){
                                        System.err.println(" NumberFormatException error");
                        }
      }
    return floatValue;
}

private float max(float[] x)
{
           float max=x[0];
           for(int loop=0;loop<x.length;loop++)
            if(max<x[loop])
              max=x[loop];
  return max;
}

private float distance(float[] x,float[] y,chromosome chrom)
{   float distance=0;
```

```
            for(int loop=0;loop<noFeatures;loop++)
             if(chrom.alleles[loop])
              distance+=(x[loop]-y[loop])*(x[loop]-y[loop]);

             return  (float)Math.sqrt(distance);

}

public float [][] centroid(BufferedReader inFile)
{
 int cluster;
 String[] attribute;
 String inData=null;
 float [][] centroid= new float[noCluster][noFeatures];
 float []   counter = new float[noCluster];
 float[] row= new float[noFeatures+1];
 boolean finished=false;

 while (!finished)
                {
                  try
                  {
                    inData = inFile.readLine();
                  }
                  catch (IOException e)
                  {
                    System.err.println("An I/O error occured when trying to read the file.");
                    System.err.println("The program cannot continue.");
                    System.exit(3);
                  }

                  if (inData != null)
                   {

                    attribute=inData.split(",");
                    row=floatValue(attribute);

                    cluster=(int)row[41];

                    for(int j=0;j<noFeatures;j++)
                      centroid[cluster][j]+=row[j];

                    counter[cluster]++;


                   }
                  else{
                    finished=true;
                  }

                }
```

```
                for(int k=0;k<noCluster;k++)
                for(int j=0;j<noFeatures;j++)
                  centroid[k][j]/=counter[k];


 return centroid;
}


private float [] sWithin(BufferedReader inFile,chromosome chrom,float [][] centroid)
{
  boolean finished=false;
  int cluster=0;
  String inData=null;


  String[] attribute;
  float[] row= new float[noFeatures+1];
  float[] sWithin=new float[noCluster];
  float[] counter= new float[noCluster];

  for(int loop=0;loop<noCluster;loop++){
            sWithin[loop]=0f;
            counter[loop]=0f;
  }
  while (!finished)
                  {
                    try
                    {
                      inData = inFile.readLine();
                    }
                    catch (IOException e)
                    {
                      System.err.println("An I/O error occured when trying to read the file.");
                      System.err.println("The program cannot continue.");
                      System.exit(3);
                    }

                    if (inData != null)
                     {


                      attribute=inData.split(",");
                      row=floatValue(attribute);

                      cluster=(int)row[41];



               sWithin[cluster]+=distance(row,centroid[cluster],chrom);
```

```
                    counter[cluster]++;

                  }
                  else{
                    finished=true;
                  }

     }


      for(int loop=0;loop<noCluster;loop++)
       {sWithin[loop]/=counter[loop];

       }


    return sWithin;
}

private float DBIndex(float[] swithin,float[][] centroid,chromosome chrom)
{

          float [] DB = new float[noCluster];
          float DBIndex=0f;
          int noSeleFeatures=0;
          noSeleFeatures=noSeleFeatures(chrom);
          //System.out.println(noSeleFeatures);
          for(int i=0;i<noCluster;i++){

            for(int j=0;j<noCluster;j++){
                    if(i==j){ DB[i]=0f; continue;};
                    float ds = distance(centroid[i],centroid[j],chrom);
              DB[j]= (swithin[i]+swithin[j])/ds;
             }
             DBIndex+=max(DB);

   }

          DBIndex=DBIndex/(noSeleFeatures*noCluster);
          return DBIndex;
}

private int noSeleFeatures(chromosome chrom)
{

          int noSeleFeatures=0;
          for(int loop=0;loop<chrom.size;loop++)
           if(chrom.alleles[loop])
             noSeleFeatures++;
 return noSeleFeatures;
}
```

```
public float [][] fitfun(String fileName,chromosome[] chrom,float[][] center, int npop)
        {
                BufferedReader inFile = null;
                float[][] fit= new float[2][npop];
                int noSelectedFeatures=0;
                float IDB=0f;
                float[] swithin = new float[noCluster];

                for(int i=0;i<npop;i++){

                noSelectedFeatures=noSeleFeatures(chrom[i]);

                inFile=openFile(fileName);

                swithin=sWithin(inFile,chrom[i],center);
                IDB=DBIndex(swithin,center,chrom[i]);

           closeFile(inFile);

                fit[0][i]=(float)noSelectedFeatures;
                fit[1][i]=IDB;
           }

  return fit;
        }

public float [] fastNonDominatedSort(float [][] fit){

                int [] n = new int[fit[0].length];
                float [] newFitnessRank= new float[fit[0].length];
                int Rank=1;
                int p,q;
                int counter=0;

                for(p=0;p<fit.length;p++){
                   newFitnessRank[p]=0;
                   n[p]=0;
                  }

                for(p=0;p<fit[0].length;p++){
                 for(q=0;q<fit[0].length;q++)
     if((fit[0][p]>fit[0][q] && fit[1][p]>=fit[1][q])||
               (fit[0][p]>=fit[0][q] && fit[1][p]>fit[1][q])) //if p is dominated by q
           n[p]++;

                  if(n[p]==0){
                          newFitnessRank[p]=(float)Rank;
                          counter++;
```

```
                        }
                    }
                     Rank++;

                    while(counter!=0){
                      int x=0;
                      for(int i=0;i<counter;i++)
                      for(p=0;p<fit[0].length;p++){
                                n[p]=n[p]-1;
                                if(n[p]==0){
                                        newFitnessRank[p]=(float)Rank;
                                        x++;
                                 }
                               }
                               counter=x;
                               Rank++;
                      }

                               for(int i=0;i<fit[0].length;i++)
                                        System.out.print(newFitnessRank[i] + " , ");

    return newFitnessRank;
   }

  public float [] weighting(float[] IDBIndex,float[] nfeature)
  {
          float weight1=0.0017f;
          float weight2=0.9983f;
          float [] newfitness= new float[IDBIndex.length];
          for(int i=0;i<IDBIndex.length;i++){
            newfitness[i]=IDBIndex[i]*weight2+nfeature[i]*weight1;
          }
         return newfitness;
          }

private BufferedReader openFile(String fileName)
 {
              BufferedReader inFile=null;

              try
              {
                 inFile = new BufferedReader(new FileReader(fileName));
              }
              catch (FileNotFoundException e)
              {
                 System.err.println("The file " + fileName + " wasn't found.");

                System.exit(2);
              }
           return inFile ;
 }
```

```
private void closeFile(BufferedReader inFile)
 {

    try
     {
       inFile.close();
     }
     catch (IOException e)
     {
       System.err.println("There was an error closing the destination file.");
       System.exit(3);
     }
 }


}

class fselection{

          final static int   NFEATURES   =41;
          final static int   NCLUSTER    =5;

          final static int   NPOP        = 10;
          final static int   MAXGEN      = 100;

   final static float PROB_TOUR   = 0.90f;
   final static float PROB_CROSS  = 0.80f;
   final static float PROB_MUTATE = 0.01f;
   final static float ELIT_RATE   = 0.1f;

  public static void main(String[] args)
  {

     // declare the file stream objects to use
     BufferedReader inFile = null;
     PrintWriter outFile = null;
     chromosome[] pop=new chromosome[NPOP];
     chromosome[] newpop=new chromosome[NPOP];

     genetic ga= new genetic(NFEATURES,NPOP,NCLUSTER);
     fselection fs = new fselection();
     float [][] centroid= new float[NCLUSTER][NFEATURES];

    if (args.length != 1)
     {
       System.err.println("The correct number of commandline arguments weren't given.");
       System.err.println("Usage:   java FileNumber <sourcefile> <numeratedfile>
<normalizedfile>");

       System.exit(1);
     }
```

```
    inFile=fs.openFile(args[0]);
    centroid=ga.centroid(inFile);
    fs.closeFile(inFile);

for(int i=0;i<NCLUSTER;i++){
for(int j=0;j<NFEATURES;j++)
  System.out.print(centroid[i][j] + ",");
System.out.println();
}
  for(int i=0;i<NPOP;i++){
     pop[i]= new chromosome(NFEATURES);
    // newpop[i]=new chromosome(NFEATURES);
    }

    for(int i=0;i<NPOP;i++){
    // pop[i]= new chromosome(NFEATURES);
    newpop[i]=new chromosome(NFEATURES);

    }
  System.out.println("Initial Population");

      float[][] bothFitness;
      float[] fitness;
      bothFitness=ga.fitfun(args[0],pop,centroid,NPOP);
      fitness= bothFitness[1];
      //fitness=ga.fastNonDominatedSort(bothFitness);
      //fitness=ga.weighting(bothFitness[1],bothFitness[0]);

      for ( int i=0; i<NPOP; i++)
          pop[i].fitness=fitness[i];

      fs.printpop(pop,NPOP);


     for (int ngen=0;  ngen < MAXGEN;  ngen++ ){

      for ( int i=0; i<NPOP; i+=2){

        newpop[i] =  ga.tournamentSelect ( pop,  PROB_TOUR);
        newpop[i+1] = ga.tournamentSelect ( pop,  PROB_TOUR);

        ga.crossover ( newpop[i], newpop[i+1], PROB_CROSS);

        ga.mutate ( newpop[i], PROB_MUTATE );
        ga.mutate ( newpop[i+1], PROB_MUTATE );

      }

      float best=1.0f;
      float sum =0.0f;
```

```
        System.out.println("Unsorted Previous population");
        fs.printpop(pop,NPOP);

        fs.sort(pop,NPOP);
        System.out.println("Sorted Previous population");
        fs.printpop(pop,NPOP);

        bothFitness=ga.fitfun(args[0],newpop,centroid,NPOP);
        fitness=bothFitness[1];
        //fitness= ga.fastNonDominatedSort(bothFitness);
        //fitness=ga.weighting(bothFitness[1],bothFitness[0]);

        for ( int i=0; i<NPOP; i++)
            newpop[i].fitness=fitness[i];

        System.out.println("Unsorted New population");
        fs.printpop(newpop,NPOP);

        fs.sort(newpop,NPOP);
        System.out.println("Sorted New population");
        fs.printpop(newpop,NPOP);

        int esize =(int) (ELIT_RATE * (float)NPOP);
        System.out.println("The value of esize "+ esize);

        for ( int i=esize; i<NPOP; i++)
            pop[i].copy(newpop[i-esize]);

        for ( int i=0; i<NPOP; i++){
           if ( pop[i].fitness < best )
             best = pop[i].fitness;

           sum += pop[i].fitness;
        }

            System.out.println("Final population");
        fs.printpop(pop,NPOP);

        System.out.println("Generation " + ngen + " best fitness is " + best + " the average fitness is "
+ sum/NPOP);

    }

 }

 void sort(chromosome chr[],int npop)
 {
            for (int i=0; i<npop-1; i++ )
              for (int j=i+1; j<npop; j++ )
                 if(chr[i].fitness>chr[j].fitness)
```

```
                {
                    chromosome tmp= new chromosome(chr[i].size);

                    tmp.copy(chr[i]);
                    chr[i].copy(chr[j]);
                    chr[j].copy(tmp);

                }
            }
void printpop( chromosome chr[], int npop){
  String output="";
  for (int i=0; i<npop; i++ ){
  for (int j=0; j<chr[i].size; j++ ){
    if(chr[i].alleles[j])
      output=output + "1" ;
    else
      output=output + "0" ;
    }
       System.out.println( output + "--> " + chr[i].fitness ); output="";
    }
}

BufferedReader openFile(String fileName)
{
                BufferedReader inFile=null;

                try
                {
                   inFile = new BufferedReader(new FileReader(fileName));
                }
                catch (FileNotFoundException e)
                {
                   System.err.println("The file " + fileName + " wasn't found.");

                  System.exit(2);
                }
             return inFile ;
}
void closeFile(BufferedReader inFile)
{
    try
     {
       inFile.close();
     }
     catch (IOException e)
     {
       System.err.println("There was an error closing the destination file.");
       System.exit(3);
     }}}
```

Appendix IV Self-Organized Map Code

```
import java.applet.*;
import java.util.*;
import java.lang.*;

public class fpoint {

  private final int NFEATURES=41;
  public float[] x;



  public fpoint()
   {
    x=new float[NFEATURES];
    for(int loop=0;loop<x.length;loop++)
     x[loop]=0f;
   }

  public fpoint(float[] x1)
   {
    x=new float[NFEATURES];
    for(int loop=0;loop<x.length;loop++)
          x[loop]=x1[loop];
   }

  public float sum(){
           float sum=0f;
   for(int loop=0;loop<x.length;loop++)
          sum+=x[loop];
   return sum;
          }

  public float dist() {
            float dist=0f;
            for(int loop=0;loop<x.length;loop++)
             x[loop]*=x[loop];
            for(int loop=0;loop<x.length;loop++)
             dist+=x[loop];
    return (float)Math.sqrt(dist);
  }

  public void set(float[] x1) {
    for(int loop=0;loop<x.length;loop++)
          x[loop]=x1[loop];

  }

  public void normalize()
```

```
{
  float d=dist();
  for(int loop=0;loop<x.length;loop++)
        x[loop]/=d;
}

/*
ipoint fti() {
  ipoint i= new ipoint(Math.round(x), Math.round(y), Math.round(z));
  return i;
}
*/

public fpoint sub(fpoint m) {
  fpoint ret = new fpoint();
  for(int loop=0;loop<x.length;loop++)
        ret.x[loop]=x[loop]-m.x[loop];

  return ret;
}

public fpoint sub(int m) {
  fpoint ret = new fpoint();
  for(int loop=0;loop<x.length;loop++)
        ret.x[loop]=x[loop]-(float)m;

  return ret;
}

public fpoint sub(float m) {
  fpoint ret = new fpoint();
  for(int loop=0;loop<x.length;loop++)
        ret.x[loop]=x[loop]-m;

  return ret;
}

public fpoint add(fpoint m) {
  fpoint ret = new fpoint();
  for(int loop=0;loop<x.length;loop++)
   ret.x[loop]=x[loop]+m.x[loop];

  return ret;
}

public fpoint mult(fpoint m) {
  fpoint ret = new fpoint();
  for(int loop=0;loop<x.length;loop++)
   ret.x[loop]=x[loop]*m.x[loop];

  return ret;
```

```
}

public fpoint add(int m) {
  fpoint ret = new fpoint();
  for(int loop=0;loop<x.length;loop++)
        ret.x[loop]=x[loop]+(float)m;

  return ret;
}

public fpoint add(float m)
{
  fpoint ret = new fpoint();
  for(int loop=0;loop<x.length;loop++)
        ret.x[loop]=x[loop]+m;

  return ret;
}

public fpoint mult(int m)
{
  fpoint ret = new fpoint();
  for(int loop=0;loop<x.length;loop++)
        ret.x[loop]=x[loop]*(float)m;

  return ret;
}

public fpoint mult(float m) {
  fpoint ret = new fpoint();
  for(int loop=0;loop<x.length;loop++)
        ret.x[loop]=x[loop]*m;

  return ret;
}

public fpoint mult(double m) {
  fpoint ret = new fpoint();
  for(int loop=0;loop<x.length;loop++)
  ret.x[loop]=(float)(x[loop]*m);

  return ret;
}

public fpoint div(int m) {
  fpoint ret = new fpoint();
  if (m!=0) {
     for(int loop=0;loop<x.length;loop++)
        ret.x[loop]=x[loop]/m;
          }
  else{
```

```
        for(int loop=0;loop<x.length;loop++)
          ret.x[loop]=0;
        }
    return ret;
  }

  public fpoint div(float m) {
        fpoint ret = new fpoint();
    if (m!=0) {
      for(int loop=0;loop<x.length;loop++)
          ret.x[loop]=x[loop]/m;
           }
    else{
       for(int loop=0;loop<x.length;loop++)
          ret.x[loop]=0f;
        }
    return ret;
  }

  public float pipe(fpoint m) {
            float pipe=0f;
            for(int loop=0;loop<x.length;loop++)
              pipe+=x[loop]*m.x[loop];

    return pipe;
  }

  public void print() {

           for(int loop=0;loop<x.length;loop++)
    System.out.println(" x " + loop + " = " +x[loop]);
    System.out.println("\n");
  }
}
public class ipoint {
  public int x,y,z;

  public ipoint() {
    x=0; y=0; z=0;
    }
  public ipoint(int x1, int y1, int z1) {
    x=x1; y=y1; z=z1;
    }

  public float dist() {
    return (float)Math.sqrt(x*x+y*y+z*z);
  }

  public void set(int x1, int y1, int z1) {
    x=x1;
    y=y1;
```

```
    z=z1;
  }

  public ipoint sub(ipoint m) {
    ipoint ret = new ipoint();
    ret.x=x-m.x;
    ret.y=y-m.y;
    ret.z=z-m.z;
    return ret;
  }

  public ipoint sub(int m) {
    ipoint ret = new ipoint();
    ret.x=x-m;
    ret.y=y-m;
    ret.z=z-m;
    return ret;
  }

  public ipoint add(ipoint m) {
    ipoint ret = new ipoint();
    ret.x=x+m.x;
    ret.y=y+m.y;
    ret.z=z+m.z;
    return ret;
  }

  public ipoint add(int m) {
    ipoint ret = new ipoint();
    ret.x=x+m;
    ret.y=y+m;
    ret.z=z+m;

    return ret;
  }

  public ipoint mult(int m) {
    ipoint ret = new ipoint();
    ret.x=x*m;
    ret.y=y*m;
    ret.z=z*m;
    return ret;
  }

  public ipoint mult(float m) {
    ipoint ret = new ipoint();
    ret.x=Math.round(x*m);
    ret.y=Math.round(y*m);
    ret.z=Math.round(z*m);

    return ret;
```

```
  }

  public ipoint div(int m) {
    ipoint ret = new ipoint();
    if (m!=0) {
      ret.x=x/m;
      ret.y=y/m;
      ret.z=z/m;
    }
    else
      ret.x=ret.y=ret.z=0;
    return ret;
  }

  public ipoint div(float m) {
    ipoint ret = new ipoint();
    if (m!=0.0f) {
      ret.x=Math.round(x/m);
      ret.y=Math.round(y/m);
      ret.z=Math.round(z/m);
    }
    else
      ret.x=ret.y=ret.z=0;

    return ret;
  }

  public int pipe(ipoint m) {
    return (x*m.x+y*m.y+z*m.z);
  }

  public void print() {
    System.out.println(" x = " + x);
    System.out.println(" y = " + y);
    System.out.println(" z = " + z);
    System.out.println("\n");
  }
};
// required imports
import java.io.*;
import java.util.Random;

class sofm{
  java.util.Random r = new java.util.Random();

  //Weight values of the SOM
  public fpoint v_weights[][]   = new fpoint [50][50];

  //Randomly selected values
  fpoint v_samples[] = new fpoint [50];
```

```
//Dimensions of the Map (multiplied by 4 in paint(Graphics))
int HEIGHT     = 50;
int WIDTH      = 50;

//Initial radius of influence (should start large)
int RADIUS     = 60;

public sofm()
{
          for (int loop=0; loop<HEIGHT; loop++)
    for (int loop2=0; loop2<WIDTH; loop2++)
                v_weights[loop][loop2]= new fpoint();
          }

//Calculates the Euclidean distance

public float get_dist(fpoint imap, fpoint iactual) {
  fpoint d= new fpoint();
  d = imap.sub(iactual);

  d.set(d.mult(d).x);

  return ((float)Math.sqrt(d.sum()));
}
//Calculates the Euclidean distance without the square root, saves time

public float fget_dist(ipoint imap, ipoint iactual) {
  ipoint d= new ipoint();
  d = imap.sub(iactual);
  d.set(d.x*d.x,d.y*d.y,d.z*d.z);

  return ((float)Math.sqrt(d.x+d.y+d.z));
}
 //initialize weights
 public void init_v_weights() {
  for (int loop=0; loop<HEIGHT; loop++)
    for (int loop2=0; loop2<WIDTH; loop2++)
     for(int loop3=0;loop3<v_weights[loop][loop2].x.length;loop3++)
          v_weights[loop2][loop].x[loop3] = ((float)(r.nextInt(500)))/100.0f;
   }

ipoint getBestMatch(fpoint input) {
  int match_amt=0;
  int WH=WIDTH*HEIGHT;
  float max_dist=0.0f;
  ipoint match_list[] = new ipoint[WIDTH*HEIGHT];
  for (int loop=0; loop<WIDTH*HEIGHT; loop++)
    match_list[loop] = new ipoint();
  max_dist=get_dist(input,v_weights[0][0]);

  for (int loop=0; loop<HEIGHT; loop++)
```

```
      for (int loop2=0; loop2<WIDTH; loop2++) {
        float t_dist=get_dist(input,v_weights[loop2][loop]);
        if (t_dist<max_dist) {
          max_dist = t_dist;
          match_list[0].set(loop2,loop,0);
          match_amt=1;
          }
        else if (t_dist==max_dist && match_amt<(WH))
          match_list[match_amt++].set(loop2,loop,0);
    }
    return match_list[r.nextInt(match_amt)];
    }//getBestMatch

public void scaleNeighbors(ipoint loc, fpoint factual, float t2) {
    int R2 = Math.round(((float)(RADIUS)*(1.0f-t2))/2.0f);
    ipoint outer  = new ipoint(R2,R2,0);
    ipoint center = new ipoint(0,0,0);
    float d_normalize = fget_dist(center,outer);

    for (int loop=-R2; loop<R2; loop++)
      for (int loop2=-R2; loop2<R2; loop2++)
        if ((loop+loc.y)>=0 && (loop+loc.y)<HEIGHT && (loop2+loc.x)>=0 &&
(loop2+loc.x)<WIDTH) {

          //Get distance from center point and normalize it
          outer.set(loop2,loop,0);
          float distance = fget_dist(outer,center);
          distance/= d_normalize;

          //Get how much to scale it by
          float t=(float)(Math.exp(-1.0f*(Math.pow(distance,2.0f))/0.15f));

          //Amount a neuron can learn decreases with time
          //The 4 is chosen and the +1 is to avoid divide by 0's
          t/=(t2*4.0f+1.0f);

          //Scale it with the parametric equation
          fpoint temp = (factual.mult(t)).add(v_weights[loc.x+loop2][loc.y+loop].mult(1.0f-t));
          v_weights[loc.x+loop2][loc.y+loop] = temp;
          }
  }//scale neighbors

}
class som{

   final static int    NFEATURES    =41;

   public static void main(String[] args)
   {
    final int    MAX_ITER     =100;
    final float T_INC=1.0f/(float)MAX_ITER;
```

```
   float t=0;
   fpoint r_sample;
   ipoint bmu_loc = new ipoint();
   sofm s = new sofm();

   //declare the in and out file names
   String inData="";
   String outData="";
    // declare the file stream objects to use
    BufferedReader inFile = null;
    PrintWriter outFile = null;

    if (args.length != 2)
    {
      System.err.println("The correct number of commandline arguments weren't given.");
      System.err.println("Usage:   java FileNumber <sourcefile> <numeratedfile>
<normalizedfile>");

      System.exit(1);
    }
    try
    {
      outFile = new PrintWriter(new BufferedWriter(new FileWriter(args[1])));
    }
    catch (IOException e)
    {
        System.err.println("An I/O error occured when trying to open the file");
        System.err.println("for writing.  The program cannot continue.");
        System.exit(3);
    }
   String[] attribute= new String[NFEATURES+1];
   float [] row = new float[NFEATURES];

   //Initialize the weight first
   s.init_v_weights();

  while(t<1.0f) {

     try
    {
       inFile = new BufferedReader(new FileReader(args[0]));
    }
    catch (FileNotFoundException e)
    {
       System.err.println("The file " + args[0] + " wasn't found.");

      System.exit(2);
    }
     boolean finished=false;
     int loop=0;
     while (!finished)
```

```
            {
              try
              {
                inData = inFile.readLine();
              }
              catch (IOException e)
              {
                System.err.println("An I/O error occured when trying to read the file.");
                System.err.println("The program cannot continue.");
                System.exit(3);
              }
              if (inData != null)
               {
                try{
                 attribute=inData.split(",");
                 row=floatValue(attribute);

                 //Get a random sample
                 r_sample=new fpoint(row);

                  //Find its best matching unit
                  bmu_loc  = s.getBestMatch(r_sample);
                  //Scale the neighbors according to t
                  s.scaleNeighbors(bmu_loc,r_sample,t);
            }
          catch(NullPointerException e)
           {
            System.out.println("Null Pointer Exception Error");
            System.exit(0);
                   }
           loop++;
            if(loop%1000==0) System.out.println(loop);
                }
        else{
                  finished=true;
                }
            }
          try
         {
     inFile.close();
         }
    catch (IOException e)
         {
     System.err.println("There was an error closing the destination file.");
     System.exit(3);
     }
    System.out.println(t);
    t+=T_INC;
  }
      //Output the file
for(int loop=0;loop<s.v_weights.length;loop++){
```

```
   for(int loop2=0;loop2<s.v_weights[loop].length;loop2++){
   for(int loop3=0;loop3<s.v_weights[loop][loop2].x.length;loop3++)   {

      Double nValue= new Double(s.v_weights[loop][loop2].x[loop3]);
      attribute[loop3]= nValue.toString();
      }
      inData=join(attribute,",");
      outFile.println(inData);
}      }
        outFile.close();

}
private static String join(String[] input, String delim)
{
           StringBuffer result = new StringBuffer();

           for(int loop=0,n=input.length;loop<n;loop++)
           {
             result.append(input[loop]);
             if (loop < n-1)
             {
              result.append(delim);
             }
   }

return result.toString();
}
static float [] floatValue(String [] attribute)
 {
      float[] floatValue= new float[NFEATURES];

      for (int loop=0; loop<NFEATURES; loop++)
       {
        try{
           floatValue[loop] = Double.valueOf(attribute[loop].trim()).floatValue();
          }
        catch(NumberFormatException e){
                                        System.err.println(" NumberFormatException error");
                        }
      }
    return floatValue;
 }}
```

Appendix V GA Clustering Code

```
// required imports
import java.io.*;
import java.util.Random;

class genes{

 public int size;

 public float[] alleles;

 genes(int size){

          this.size=size;
   alleles= new float[size];
          for ( int j=0; j<size; j++)
    alleles[j] = 0f;
          }

 public void set(float[] value)
 {
          for (int j=0; j<size; j++ )
            alleles[j]=value[j];

          }
}

class chrom{

 public int size;

 public genes[] gene;
 public float[] dist;
 public int[] count;
 public int[] clusterCategory;
 public float fitness;

 chrom(int size,int gsize){

          this.size=size;
   gene= new genes[size];
   dist= new float[size];
   count = new int[size];
   fitness=0;
          for ( int j=0; j<size; j++)
    gene[j] = new genes(gsize);

   for ( int j=0; j<size; j++)
    dist[j] = 0f;
   for ( int j=0; j<size; j++)
```

```
    count[j] = 0;
        }

 public void set(chrom value)
 {
        for (int i=0; i<size; i++ )
          gene[i].set(value.gene[i].alleles);

 }
 public boolean compare(genes a, genes b)
 {
           for(int loop=0;loop<a.size;loop++){
            if(a.alleles[loop]!=b.alleles[loop])
              return false;

             }
        return true;
 }
}
class gen{

        //public chrom [] pop;

   private int noFeatures;
   private int popSize;
        private int noCluster;


        private Random Rand = new Random();

        gen(int noFeatures,int popSize,int noCluster)
        {
         this.noFeatures=noFeatures;
         this.popSize=popSize;
         this.noCluster=noCluster;

         //pop= new chrom[popSize];

         //for(int i=0;i<popSize;i++)
         //    pop[i]=new chrom(noCluster, noFeatures);
        }
public chrom tournamentSelect ( chrom[] pop, float prob){

          int first = Rand.nextInt(popSize-1);
          int second= Rand.nextInt(popSize-1);
          while( first == second  ){
              second=Rand.nextInt(popSize-1);
          }

          float invert = Rand.nextFloat();

     chrom temp = new chrom(noCluster,noFeatures);
```

```
            //System.out.println("In tournament selection value of fitness is "+ first +"=" + x +
" & " + second + " " +y);
            if ( pop[first].fitness > pop[second].fitness ){
              if ( invert < prob )
                temp.set(pop[first]);
              else
                temp.set(pop[second]);
            } else{
              if ( invert < prob )
                temp.set(pop[second]);
              else
                temp.set(pop[first]);
            }
           return temp;
          }

public boolean crossover ( chrom first, chrom second, float prob ){
           float p = Rand.nextFloat();
           if ( p > prob ) return false;
           int r= Rand.nextInt(noCluster-1);

           int loop=0;
           while(first.compare(first.gene[r],second.gene[r])&& loop<=noCluster){

                    r= Rand.nextInt(noCluster-1);
                    loop++;

                  }
             genes tmp= new genes(noFeatures);
             tmp.set(first.gene[r].alleles);
             first.gene[r].set(second.gene[r].alleles);
             second.gene[r].set(tmp.alleles);

           return true;
          }

public boolean mutate ( chrom indi, float prob, float[][] data){

           float p=Rand.nextFloat();
           boolean check=false;
           do{
            int r= Rand.nextInt(noCluster-1);
            int d= Rand.nextInt(data.length-1);

            if( p > prob ) return false;

            genes g= new genes(noFeatures);
            g.set(data[d]);

            check=false;
            for(int loop=0;loop<noCluster;loop++){
```

```
            if(r!=loop){
              if(indi.compare(g,indi.gene[loop]))
                check=true;
             }
            }
            if(!check) indi.gene[r].set(g.alleles);

           }while(check);

           return true;
          }
public float fitfun(chrom pop)
{
    float fit=0f;

    for(int loop=0;loop<noCluster;loop++)
     {
      if(pop.count[loop]==0)
        return 0;

      fit+=pop.dist[loop];
     }
    return 1/fit;

}
public void initPop(chrom[] pop, float[][] data)
{
 for(int loop=0;loop<popSize;loop++)
 for(int loop2=0;loop2<noCluster;loop2++)
  {
                    int s=Rand.nextInt(data.length-1);

                    for(int loop3=0;loop3<noFeatures;loop3++)
                      pop[loop].gene[loop2].alleles[loop3]=data[s][loop3];
          }
}


public chrom clusterGeneration(chrom pop1, float[][] data)
{
 pop1.clusterCategory =new int[data.length];
 float[] d= new float[noCluster];

              for(int loop=0;loop<data.length;loop++)
          {
            for(int loop2=0;loop2<noCluster;loop2++)
                                d[loop2]=distance(pop1.gene[loop2].alleles,data[loop]);

                                int minIdx=minIndex(d);
                                pop1.clusterCategory[loop]=minIdx;
                                pop1.dist[minIdx]+=d[minIdx];
                                pop1.count[minIdx]++;
```

```
                                }

                        return pop1;

  }

public chrom clusterCenter(float[][] data, int [] cluster)
{

   chrom clusterCenter=new chrom(noCluster,noFeatures);
   int[] counter= new int[noCluster];

   for(int loop=0;loop<noCluster;loop++)
            counter[loop]=0;

           for(int loop=0;loop<data.length;loop++)
           {
                    for(int loop2=0;loop2<noFeatures;loop2++)
                      clusterCenter.gene[cluster[loop]].alleles[loop2]+=data[loop][loop2];

                    counter[cluster[loop]]++;

            }

           for(int loop=0;loop<noCluster;loop++)
           for(int loop2=0;loop2<noFeatures;loop2++)
             clusterCenter.gene[loop].alleles[loop2]/=counter[loop];
return clusterCenter;
}
public int minIndex(float[] x)
{
   float min=x[0];
   int minIndex=0;
   for(int loop=0;loop<x.length;loop++)
   if(x[loop]<min)
     {
           min=x[loop];
           minIndex=loop;
           }
   return minIndex;
}

public float distance(float[] x,float[] y)
{   float distance=0f;
           for(int loop=0;loop<x.length;loop++)
            distance+=(x[loop]-y[loop])*(x[loop]-y[loop]);

            return  (float)Math.sqrt(distance);

 }}
class cluster{
```

```
        final static int    NFEATURES   =41;
        final static int    NCLUSTER    =5;

        final static int    NPOP        = 100;
        final static int    MAXGEN      = 100;
        final static int    NNEURONS    =2500;

  final static float PROB_TOUR   = 0.90f;
  final static float PROB_CROSS  = 0.80f;
  final static float PROB_MUTATE = 0.01f;
  final static float ELIT_RATE   = 0.01f;
 public static void main(String[] args)
 {

   // declare the file stream objects to use
   BufferedReader inFile = null;
   PrintWriter outFile = null;
   gen ga= new gen(NFEATURES,NPOP,NCLUSTER);
   chrom[] pop=new chrom[NPOP];
   chrom[] newpop=new chrom[NPOP];

if (args.length != 2)
   {
     System.err.println("The correct number of commandline arguments weren't given.");
     System.err.println("Usage:   java FileNumber <sourcefile>");

     System.exit(1);
   }

   try
   {
      inFile = new BufferedReader(new FileReader(args[0]));
   }
   catch (FileNotFoundException e)
   {
      System.err.println("The file " + args[0] + " wasn't found.");

     System.exit(2);
   }
   try
   {
     outFile = new PrintWriter(new BufferedWriter(new FileWriter(args[1])));
   }
   catch (IOException e)
   {
       System.err.println("An I/O error occured when trying to open the file");
       System.err.println("for writing.  The program cannot continue.");
       System.exit(3);
   }

  float [][] neurons= new float[NNEURONS][NFEATURES];
  boolean finished=false;
```

```
    int idx=0;
    String inData="";
    String[] attribute;

    //Read data from the file

     while (!finished)
                {
                  try
                  {
                    inData = inFile.readLine();
                  }
                  catch (IOException e)
                  {
                    System.err.println("An I/O error occured when                trying to read the
file.");
                    System.err.println("The program cannot continue.");
                    System.exit(3);
                  }

                  if (inData != null)
                   {

                    attribute=inData.split(",");
                    neurons[idx]=floatValue(attribute);

                    idx++;
                    }
                   else{
                     finished=true;
              }       }

                System.out.println("No of Neurons " +  idx);

    for(int i=0;i<NPOP;i++){
      pop[i]= new chrom(NCLUSTER,NFEATURES);
      newpop[i]= new chrom(NCLUSTER,NFEATURES);
     }

//Initial Step
    ga.initPop(pop,neurons);
    //The procedures of Generating Clusters
    //Step1
    //Generating new cluster
    for(int i=0;i<NPOP;i++)
     pop[i]=ga.clusterGeneration(pop[i],neurons);

    //Compute the centroid of each population
    for(int i=0;i<NPOP;i++)
    pop[i]= ga.clusterCenter(neurons, pop[i].clusterCategory);
      for (int ngen=0;  ngen < MAXGEN;  ngen++ ){
```

```
//Step2
                //Generating new cluster
                for(int i=0;i<NPOP;i++)
                pop[i]=ga.clusterGeneration(pop[i],neurons);

                //calculate fitness
                for ( int i=0; i<NPOP; i++)
     pop[i].fitness=ga.fitfun(pop[i]);

   //System.out.println("Generation " + ngen + " Fitness value= "+ ga.fitfun(pop[0]));

for ( int loop=0; loop<NPOP; loop+=2){


    newpop[loop]  =  ga.tournamentSelect (pop,PROB_TOUR);
    newpop[loop+1] = ga.tournamentSelect ( pop,PROB_TOUR);

ga.crossover ( newpop[loop], newpop[loop+1], PROB_CROSS);
ga.mutate ( newpop[loop], PROB_MUTATE,neurons );
ga.mutate ( newpop[loop+1], PROB_MUTATE,neurons );

   }

float best=1.0f;
float sum =0.0f;
sort(pop,NPOP);

for ( int i=0; i<NPOP; i++)
     newpop[i].fitness=ga.fitfun(newpop[i]);

sort(newpop,NPOP);

int esize =(int)(ELIT_RATE * (float)(NPOP));

for ( int i=esize; i<NPOP; i++)
    pop[i].set(newpop[i-esize]);
for ( int i=0; i<NPOP; i++){
    if ( pop[i].fitness < best )
      best = pop[i].fitness;

    sum += pop[i].fitness;
}
  System.out.println("Generation " + ngen + " best fitness is " + best + " the average fitness
is " + sum/NPOP);



        for(int loop=0;loop<50;loop++){
        for(int loop2=0;loop2<50;loop2++)
          System.out.print(pop[0].clusterCategory[loop*50 + loop2]+ ",");
          System.out.println("");
        }
```

```
    if(ngen!=MAXGEN-1)
        {
         //Compute the centroid of each population
         for(int i=0;i<NPOP;i++)
                    pop[i]= ga.clusterCenter(neurons, pop[i].clusterCategory);

          }      }
   for(int loop=0;loop<50;loop++){
        for(int loop2=0;loop2<50;loop2++)
          outFile.print(pop[0].clusterCategory[loop*50 + loop2] +",");
          outFile.println("");
        }
 for (int i=0; i<NPOP; i++ ){
  outFile.println("Population " + i + " Fitness " + ga.fitfun(pop[i]));
  for (int j=0; j<NCLUSTER; j++ ){
  for (int k=0; k<NFEATURES; k++ ){
    outFile.print( pop[i].gene[j].alleles[k]);
   if(k!=NFEATURES-1)
    outFile.print(",");
   }
    outFile.println("");
  }
 }
       printpop(pop);


}
static void sort(chrom pop[],int npop)
{
          for (int i=0; i<npop-1; i++ )
            for (int j=i+1; j<npop; j++ )
              if(pop[i].fitness>pop[j].fitness)
               {
                    chrom tmp= new chrom(NCLUSTER,NFEATURES);
                    tmp.set(pop[i]);
                    pop[i].set(pop[j]);
                    pop[j].set(tmp);
               }

          }
static float [] floatValue(String [] attribute)
{
     float[] floatValue;
     floatValue= new float[attribute.length-1];
     for (int idx=0,n=attribute.length-1; idx<n; idx++)
      {

       try{
          floatValue[idx] = Double.valueOf(attribute[idx].trim()).floatValue();
        }
       catch(NumberFormatException e){
                                    System.err.println(" NumberFormatException error");
```

```
                        }
        }

     return floatValue;
 }

static void printpop( chrom[] pop){
   //for (int i=0; i<NPOP; i++ ){
   System.out.println("Population 0 " + " Fitness " + pop[0].fitness);
   for (int j=0; j<NCLUSTER; j++ ){
   for (int k=0; k<NFEATURES; k++ )
      System.out.print( pop[0].gene[j].alleles[k] + " ");
    System.out.println(" ");
   }
  //}

}

}
```