11-13-2007

Ensemble Fuzzy Belief Intrusion Detection Design

Te-Shun Chou
Florida International University, tchou001@fiu.edu

DOI: 10.25148/etd.FI08081509

FLORIDA INTERNATIONAL UNIVERSITY

Miami, Florida

ENSEMBLE FUZZY BELIEF INTRUSION DETECTION DESIGN

A dissertation submitted in partial fulfillment of the

requirements for the degree of

DOCTOR OF PHILOSOPHY

in

ELECTRICAL ENGINEERING

by

Te-Shun Chou

2007

To: Interim Dean Amir Mirmiran
College of Engineering and Computing

This dissertation, written by Te-Shun Chou, and entitled Ensemble Fuzzy Belief Intrusion Detection Design, having been approved in respect to style and intellectual content, is referred to you for judgment.

We have read this dissertation and recommend that it be approved.

_____
Niki Pissinou

_____
Kia Makki

_____
Jean Andrian

_____
Yimin Zhu

_____
Kang K. Yen, Major Professor

Date of Defense: November 13, 2007

The dissertation of Te-Shun Chou is approved.

_____
Interim Dean Amir Mirmiran
College of Engineering and Computing

_____
Dean George Walker
University Graduate School

Florida International University, 2007

DEDICATION

To the Memory of My Father

and

To My Mother

ACKNOWLEDGMENTS

I would like to express my gratitude to all those who gave the possibility to complete this thesis.

First of all, there are no words enough to express my gratitude to my advisor, Dr. Kang K. Yen who not only inspired and guided me throughout my academic program but also served as a mentor in my life. Without his constant support and encouragement I could never have finished my Ph.D. program.

Besides my advisor, I would like to thank my dissertation committee: Dr. Niki Pissinou, Dr. Kia Makki, Dr. Jean Andrian, and Dr. Yimin Zhu for all their help. Especially for Dr. Niki Pissinou and Dr. Kia Makki, without their valuable suggestions upon my research and support toward me I would have never got through my Ph.D. program. Thanks are also due to Dr. Jinsong Zhang at the Applied Research Center where I started my study for Ph.D. degree.

Thanks to the Telecommunications and Information Technology Institute, I have not only been financially supported but also had a stimulating and fun environment in which to learn and grow for the last two years. I would like to express my appreciation to all members at The Telecommunications and Information Technology Institute. In particular, I would like to thank Zhiwen Wan and Zhaomin Mo whose friendship not only warm my heart but also offer solutions whenever I encountered technical difficulties. For his kind assistance with developing computer programs, I wish to thank in addition Jun Luo. My appreciation also goes to Pat Brammer, Maria Benincasa, and Daniela C. Madriz, for their kind administrative support.

I am indebted to Ya-Fen, my wife and life partner of over twenty years. It is her belief in and support of me that gave me the strength to step out and pursue my dream. Without her encouragement and faith, I would not have been able to bring out the very best of myself and find my way to achieving what I truly want. Thanks also to my lovely daughters, Annie and Jun, for loving and believing in me. You were at my side each day that I worked on this thesis.

Lastly, and most importantly, I wish to express my deepest gratitude to my late father, Peng-Hsiao Chou, without whom I would not be living my best life and to my mother, Hsueh-Lin Wang Chou, who always gives me one hundred percent unconditional support, encouragement, and love. This thesis is dedicated to my parents, especially in memoriam to my father who was always a role model in the journey of my life and will live in my heart forever.

ABSTRACT OF THE DISSERTATION

ENSEMBLE FUZZY BELIEF INTRUSION DETECTION DESIGN

by

Te-Shun Chou

Florida International University, 2007

Miami, Florida

Professor Kang K. Yen, Major Professor

With the rapid growth of the Internet, computer attacks are increasing at a fast pace and can easily cause millions of dollar in damage to an organization. Detecting these attacks is an important issue of computer security. There are many types of attacks and they fall into four main categories, Denial of Service (*DoS*) attacks, *Probe*, User to Root (*U2R*) attacks, and Remote to Local (*R2L*) attacks. Within these categories, *DoS* and *Probe* attacks continuously show up with greater frequency in a short period of time when they attack systems. They are different from the normal traffic data and can be easily separated from normal activities. On the contrary, *U2R* and *R2L* attacks are embedded in the data portions of the packets and normally involve only a single connection. It becomes difficult to achieve satisfactory detection accuracy for detecting these two attacks. Therefore, we focus on studying the ambiguity problem between normal activities and *U2R*/*R2L* attacks. The goal is to build a detection system that can *accurately* and *quickly* detect these two attacks.

In this dissertation, we design a two-phase intrusion detection approach. In the first phase, a correlation-based feature selection algorithm is proposed to advance the speed of detection. Features with poor prediction ability for the signatures of attacks and features

inter-correlated with one or more other features are considered redundant. Such features are removed and only indispensable information about the original feature space remains. In the second phase, we develop an ensemble intrusion detection system to achieve accurate detection performance. The proposed method includes multiple feature selecting intrusion detectors and a data mining intrusion detector. The former ones consist of a set of detectors, and each of them uses a fuzzy clustering technique and belief theory to solve the ambiguity problem. The latter one applies data mining technique to automatically extract computer users' normal behavior from training network traffic data. The final decision is a combination of the outputs of feature selecting and data mining detectors. The experimental results indicate that our ensemble approach not only significantly reduces the detection time but also effectively detect *U2R* and *R2L* attacks that contain degrees of ambiguous information.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

With the rapid growth of Internet based technology, applications of computer networks such as web service, file transfer, and voice IP are extensively used. In the meantime, the networks inevitably become as the targets of computer attacks and the attacks can easily cause millions of dollar damage to an organization. According to the annual report from the Computer Emergency Response Team (CERT) [1], only 8 computer security incidents were documented in 1988 but over 130,000 in 2003. Since 2004, CERT no longer publishes the number of incidents because the attacks against Internet-connected systems have become so commonplace. Not only are those attacks increasing in a fast pace, they are also becoming more sophisticated with the advance of technology. Consequently, to manage breaches of security has become an important issue for nowadays network infrastructures and has become an irreplaceable element in modern security systems. Today intrusion detection has caught researchers' attention greater than ever. Its development and improvement have been set with the highest priority by academia, government, research institutes, and industrial corporations.

Intrusion detection is a technology developed to discover breaches of security, attempted breaches, or open vulnerabilities that could lead to potential breaches [2]. An intrusion detection system is a security management system for computers and networks. It examines activities from computer users and identifies inappropriate, incorrect, or anomalous activities within computers or networks. The possible attempts to breach the

security could be either attacks from outside of the computer system or misuse conducts from inside of the computer system [3]. The former aims at gaining access to penetrate the system and the latter tries to exploit security vulnerabilities or mistreat their approved privileges. While there are many types of attacks, they fall into four main classes.

- *Denial of Service* (*DoS*) attacks: Attackers disrupt a host or network service in order to make legitimate users not be able to have an access to a machine;
- *Probe* attacks: Attackers use programs to automatically scan networks for gathering information or finding known vulnerabilities;
- *User to Root* (*U2R*) attacks: Local users get access to root access of a system without authorization and then exploit the machine's vulnerabilities; and
- *Remote to Local* (*R2L*) attacks: Unauthorized attackers gain local access from a remote machine and then exploit the machine's vulnerabilities.

During the past years, a large variety of techniques to the task of detecting the above mentioned intruders' activities have been proposed [6]-[12]. These techniques are mainly categorized into two groups: anomaly detection and misuse detection. Anomaly detection searches for intrusive activities by comparing network traffic to those established acceptable normal usage patterns learned from training data. If the pattern of observed data is different from those learned normal ones, the data is classified as an attack. This approach can successfully detect novel and unseen malicious occurrences from computer users. However, it suffers from a high volume of false alarms. Misuse detection involves the comparison of observed traffic data with a set of well defined rules that describe signatures of intrusions. If the signature of observed network traffic is not matched with any of predefined rules, it is declared as an attack. This approach can detect the

recognized attacks in an efficient way with high level of accuracy. However, it suffers from its inability of identifying attacks which differ from these predefined patterns. A minor variation of an attack may not be identified during the whole detection procedure.

1.1 Problem Statement

Using either anomaly detection or misuse detection techniques in the design of intrusion detection systems, a data set of network traffic is necessary to be collected in advance for analysis. It consists of a great amount of traffic records with various features such as the length of connection, the type of protocol, the network service and other information. Based on this set of data, misuse detection specifies well defined attack signatures and anomaly detection constructs acceptable user behavior. However, there are several problems in the collected database.

a) *Problem of Redundant Information*

Not every feature of the network traffic information we are monitoring is relevant to the intrusion detection task. Some features may be irrelevant to the signatures of attacks, and some features may be redundant since they are highly inter-correlated with one or more of the other features [4]. The detection speed becomes slow if unnecessary information is involved in the analysis.

b) *Problem of Uncertainty*

The collected data always enclose uncertainty when only limited amount of information about intrusive activities is available. It is difficult to completely collect intrusive behavior because new exploits are discovered at anytime and anywhere. The available data is always incomplete that only contains limited information.

c) *Problem of Ambiguity*

Within the four attack categories, *DoS* and *Probe* attacks continuously show up with large amounts in a short period of time when they attack systems. They are different from the normal traffic data and can be easily separated from normal activities. On the contrary, *U2R* and *R2L* attacks are embedded in the data portions of the packets and normally involve only a single connection (a data packet related to a particular service). Their patterns are similar to those of normal activities. The boundaries between those two attacks and the normal behavior are always unclear. When processing an intrusion detection task, the attacks may not be detected if the ambiguous behavior is not considered anomalous. On the other hand, if the ambiguity is considered anomalous, then system administrators may be alerted by false alarms, i.e., in cases where there is no attack [5].

Past research results [6]-[12] shown in Table 1.1 have indicated that it is difficult to achieve satisfactory detection accuracy while detecting *U2R* and *R2L* attacks. Therefore, our study will address the issue on how to *accurately* and *quickly* detect them in the network traffic. We expect that our system not only has the ability to correctly detect those two types of attacks but also achieve a minimum number of incorrect false alarms. More specifically, we are trying to solve four major problems.

- To select a subset of features from the network traffic to advance speed of detection
- To solve the ambiguity problem between *U2R*/*R2L* attacks and normal activities
- To solve the uncertainty problem associated with the available data which is always incomplete
- To build an accurate model with high detection rate but low false negative rate

Table 1.1. Detection Performances of Past Study

Ref.	Method	*DoS*		*Probe*		*U2R*		*R2L*	
		DR	FPR	DR	FPR	DR	FPR	DR	FPR
6	KDD cup winner	97.10%	0.30%	83.30%	0.60%	12.30%	0.00%	8.40%	0.01%
7	SOM map	95.10%	-	64.30%	-	22.90%	-	11.30%	-
8	Gaussian classifier	82.40%	0.90%	90.20%	11.30%	22.80%	0.50%	9.60%	0.10%
8	K-means clustering	97.30%	0.40%	87.60%	2.60%	29.80%	0.40%	6.40%	0.10%
8	Nearest cluster alg.	97.10%	0.30%	88.80%	0.50%	2.20%	0.00%	3.40%	0.01%
8	Radial basis	73.00%	0.20%	93.20%	18.80%	6.10%	0.04%	5.90%	0.30%
8	C4.5 decision tree	97.00%	0.30%	80.80%	0.70%	1.80%	0.00%	4.60%	0.01%
9	PN-rule	-	-	-	-	6.60%	-	10.70%	-
10	Linear GP	96.70%	-	85.70%	-	1.30%	-	9.30%	-
11	Online k-means	69.81%	5.00%	99.62%	5.00%	49.45%	5.00%	6.48%	5.00%
11	SVM	99.90%	5.00%	67.31%	5.00%	0.00%	5.00%	29.09%	5.00%
11	KMO+SVM	75.76%	5.00%	99.61%	5.00%	49.45%	5.00%	22.24%	5.00%
11	SVM	99.96%	10.00%	68.10%	10.00%	0.00%	10.00%	29.16%	10.00%
12	Backpropagation	97.23%	-	96.63%	-	87.71%	-	30.97%	-

* DR: Detection Rate, FPR: False Positive Rate

1.2 Research Hypotheses

This dissertation describes the work done with the following hypotheses.

- Feature selection technique can reduce the complexity of network traffic data and therefore increase the detection speed.
- Fuzzy clustering technique [13], [14] can solve the ambiguity problem between *U2R*/*R2L* attacks and normal activities.
- Dempster-Shafer theory [15], [16] can solve the uncertainty problem caused by limited information during the detection process.
- Multiple intrusion detectors can get better detection performance than that of individual one.

1.3 Proposed Approach

In the entire course of work, we develop a system using the intrusion detection benchmark data set *DARPA KDD99* [17], which not only includes a large quantity of network traffic but also collects a wide variety of attacks. It is a standardized data set for researchers to develop and test their intrusion detection systems as well as to compare their results with those of others. In the beginning of research, a correlation-based feature selection algorithm is proposed to advance both the detection speed and accuracy. Features with poor prediction ability to the signatures of attacks and features inter-correlated with one or more other features are considered redundant. Such features will be removed and the remaining ones contain indispensable information about the original feature space. Then, the selected features are incorporated with other feature subsets into the ensemble intrusion detection design. This design includes multiple feature selecting intrusion detectors and a data mining intrusion detector that act as anomaly detection and misuse detection, respectively. This ensemble approach is capable of further improving the detection performance.

Each feature selecting intrusion detector uses a subset of features to derive independent decision about an input network traffic data, then all the decisions from multiple ones are combined into a fused result. In the kernel of each detector, a developed machine learning algorithm is used to detect both known and novel *U2R* and *R2L* attacks. The problems of uncertainty and ambiguity caused by incomplete and imprecise information are solved during the intrusion detection procedure. Using the developed algorithm, we are considering the intrusion detection task as a decision making process rather than

identifying the vicious usages by their similarity to earlier defined recognized attacks. Specifically, fuzzy clustering technique is applied to maximize the intra-class of similar types of normality and abnormality, as well as to minimize the inter-class of dissimilar types of normality and abnormality. By the use of Dempster-Shafer theory, the input network traffic is identified by fusing evidences from clustering process. Also, the *k*-nearest neighbors (*k*-NN) technique [18] is employed to speed up the detection process.

The data mining intrusion detector uses data mining technique to automatically extract computer users' normal behavior from training data set. We combine the output of this detector with the result from multiple feature selecting intrusion detectors to derive an output, which is the final decision of the input network traffic. The data mining intrusion detector acts as a filtering mechanism to reduce the number of false alarms.

1.4 Contributions

- *Speed up the detection process*: Based on the concept of information theory, a feature selection algorithm is implemented. With the use of selected features, the detection time is reduced during the intrusion detection task.
- *Solve the uncertainty problem*: By using the developed machine learning algorithm, the uncertainty problem caused by deficient information and the ambiguity between *U2R*/*R2L* attacks and normal activities is included in the design.
- *Improve the detection performance*: The proposed ensemble intrusion detection model not only increases the detection rate but also reduces the number of false alarms compared to the results of past research.

1.5 Dissertation Organization

The dissertation is organized as follows. Chapter II includes a discussion of taxonomy schemes on grouping attacks into categories, the survey of the popular used intrusion detection approaches, and the overview of feature selection and multiple-classifier techniques. Chapter III presents the idea of correlation based feature selection technique. We afterwards describe our proposed feature selection algorithm. Chapter IV initially explains the importance of considering the problem of uncertainty during the detection process. We then present the intrusion detector, namely fuzzy belief *k*-NN classifier. Chapter V discusses the experimental results of our developed feature selection algorithm. Chapter VI describes the evaluation of the fuzzy belief *k*-NN classifier. Also, we employ the feature selection results from Chapter V to test the detection accuracy and detection speed of fuzzy belief *k*-NN classifier. Chapter VII presents the ensemble fuzzy belief selection model, which combines several fuzzy belief *k*-NN feature selecting classifiers with a data mining classifier. Chapter VIII draws the conclusions and lists future research directions.

CHAPTER II

LITERATURE REVIEW

The main concern of intrusion detection systems is to detect possible abnormal behavior from computer users. This chapter starts with the discussion of a variety of taxonomies for intrusions. *DARPA KDD99* intrusion detection benchmark data set used throughout our entire research is described. Next, we introduce two categories of systems and their related research work. The former category is host-based and network-based intrusion detection. The later one is knowledge-based and behavior-based intrusion detection. We then introduce feature selection techniques that have been applied to find informative feature subset from a network traffic data stream. At last, multiple-classifier intrusion detection systems are described.

2.1 Intrusion Categorization

The concept of detecting abnormal behavior of computer users was first introduced by Anderson in 1980 [3]. He published a paper, Computer Security Threat Monitoring and Surveillance, and defined that an attack was a specific formulation or execution of a plan to carry out a threat. He classified a threat as a deliberate unauthorized attempt to

- access information,
- manipulate information, or
- render a system unreliable or unusable.

Since then, a variety of taxonomy schemes on grouping attacks into categories have been proposed. For example, in 1987 Denning [19] classified abnormal patterns of system usage into eight categories. They are attempted break-in, masquerading or successful break-in, penetration by legitimate user, leakage by legitimate user, inference by legitimate user, trojan horse, virus, and denial-of-service. In 1988 Smaha [20] divided intrusions into six main types: attempted break-ins, masquerade attacks, penetration of the security control system, leakage, denial of service, and malicious use. Howard [21] summarized the variations of taxonomy of attacks on the Internet from 1989 to 1995 in one of the chapters in his PhD dissertation. Dekker [22] defined network security incident as an activity threat violated an explicit or implicit security policy and classified incidents into the probe, scan, account compromise, root compromise, packet sniffer, denial of service, exploitation of trust, malicious code, and Internet infrastructure attacks in 1997.

In 1999, Lincoln Laboratory at MIT created the *KDD99* data set, which is known as "DARPA Intrusion Detection Evaluation Data Set" [17]. The data set includes thirty-nine types of attacks that are classified into four main categories. They are *DoS*, *Probe*, *U2R*, and *R2L* attacks.

The first category of attacks is *DoS* attacks. In this type of attacks, attackers attempt to disrupt a host or network resource in order to make legitimate users not be able to access to the computer service. The victim machines can be web server, domain name system server, mail server, and so on. In the *DARPA KDD99* data set, there are many common forms of *DoS* attacks that are included. For example, *smurf* attack is one and takes over 70% of the attacks in the *DoS* category. By using the vulnerability of ICMP (Internet Control Message Protocol), the attack can cause a target system crash. The attacker can

send a large number of ICMP “echo request” packets to the broadcast address and every packet has a spoofed source address of the intended target system. Any machine in the subnets will respond by sending ICMP “echo reply” packets back to the target. If the number of the packets is more than the system can handle, the result is the spoofed system can no longer be able to service to the real ICMP requests. Another common way to fail a system is *neptune* attacks. Over 25% of *DoS* attacks are *neptune* in the data set. It is a SYN (Synchronize) flood attack that exists in TCP/IP (Transmission Control Protocol/Internet Protocol) implementation of a network. The attacker just simply rapidly sends out a large number of connection requests but never responds to any replies from the system. While the attacker continues to request new connections faster than the system can handle, the legitimate connection requests can never be accommodated. In the mean time, the system may run out of memory and even crash.

The second category of attacks is *Probe* attacks. By using programs to automatically scan a large amount network IP addresses, the attacker can explore vulnerabilities of the computers. Once any vulnerability is found, the attacker can thus gain the access to the system and start to gather information without authorization. The *DARPA KDD99* data set collects six scanning attacks of *Probe* attack category. They are *ipsweep*, *mscan*, *nmap*, *portsweep*, *saint*, and *satan*.

The third category of attacks is *U2R* attacks. The attacker pretends as a legitimate user of the system without authorization and then exploits the system’s vulnerabilities to get root access of that system. The *DARPA KDD99* data set consists of eight different types of *U2R* attacks. Among them, *buffer_overflow* attack is the most common one that starts with by feeding many data into a fix length buffer. When the volume of data exceeds the

size of the buffer that can hold, the extra information will overflow into other buffers and overwrite the instructions that suppose to be executed. The result may cause the system crash or make the system execute the attacker's program as if it is part of the system's original programs.

The forth category of attacks is *R2L* attacks. This type of attacks is that an unauthorized attacker through networks gains local access as a user of local machine and then exploits the machine's vulnerabilities. Totally fifteen types of *R2L* attacks are included in the *DARPA KDD99* data set. For example, the *ftp_write* attack is one that the attacker creates rhost file to make anonymous FTP (File Transfer Protocol) directory writable and finally obtains local login to the system. The *guess_passwd* is another one that the attacker tries to gain access to a user's account by repeatedly guessing the possible passwords. Any service that needs password to access possibly becomes an attacked target, for example, *rlogin*, *ssh*, *ftp*, *telnet*, *pop*, and *imap*.

2.2 Host-Based vs. Network-Based Intrusion Detection

Based on the sources of data, intrusion detection systems can be divided into two major classes, host-based and network-based. In the first kind of systems, the intrusion detection mechanism is installed on the local host/terminal. By examining the status of audit information on system's behavior, the system finds signs of intrusion and can then protect its own local machine. The audit information can be obtained from different sources such as system logs and activities, application logs, and target monitoring [23]. These logs could be Unix logs, NT/2000/XP logs, firewall logs, router logs, web server logs, and FTP logs. The intrusions can be critical file modifications, segmentation fault

errors recorded in logs, crashed services or extensive usage of the processors [24]. From the system point of view, all users are considered as local clients to the target environment.

Unlike the host-based intrusion detection system that only protects its own host machine by examining audit trail, network-based intrusion detection system protects the entire environment of the network by monitoring all the activities from both inside and outside of the network. By inspecting the traffic data that goes through the network, the possible intrusions can be identified. In general, the network traffic that needs to be monitored is quite large. For releasing each sensor's detection burden, the network based intrusion system deploys its sensors on different locations instead of one central point. With a good design of sensor placement on the network, the network-based intrusion detection system can efficiently monitor a large network environment.

Compared to the audit trial used by a host-based intrusion detection system, the data shown on network-based intrusion detection system always has less information about what exactly happens during the attack courses. However, the data of network-based intrusion detection system has a broader range of attacks because the background traffic is much wider than that of a host-based intrusion detection system. With the popular use of the Internet and the growth of larger network systems, more attention has been focused on the development of network-based intrusion detection systems. Also, current trend shows people incline to use both host-based and network-based information to design hybrid systems. These intrusion detection systems are capable of running detection on local host and monitor network traffic as well.

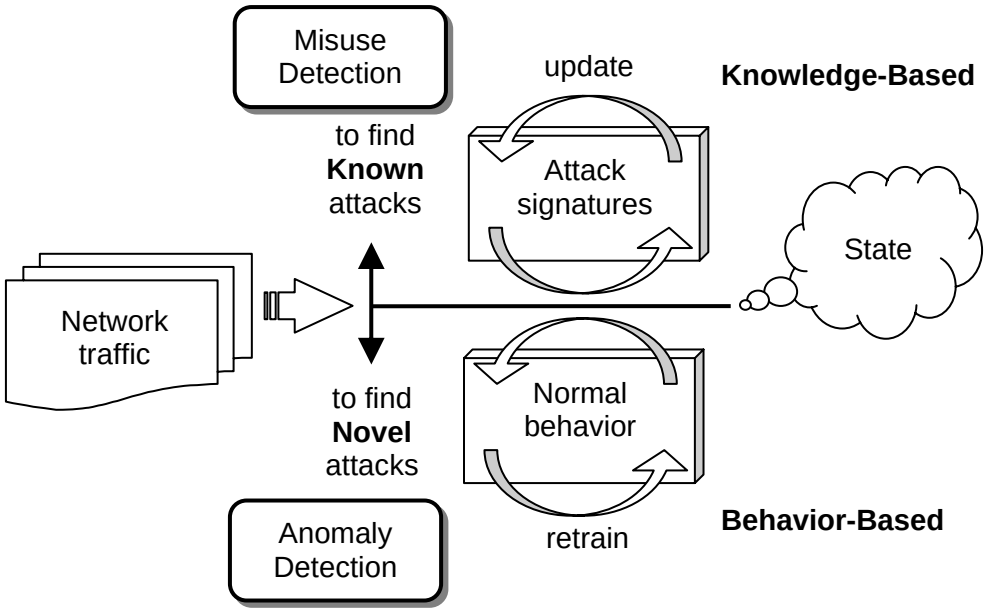


Figure 2.1. Comparison of Knowledge-Based and Behavior-Based Intrusion Detection

2.3 Knowledge-Based vs. Behavior-Based Intrusion Detection

Based on the use of detection technique, intrusion detection systems are categorized as knowledge-based and behavior-based intrusion detection systems as shown in Figure 2.1. Knowledge-based intrusion detection is typically realized by modeling known attack behavior with prior understanding about specific attacks and system vulnerabilities. This technique is to compare network traffic data being observed with well defined attack patterns for identifying the possible penetrations to a system. When the data is the same as one of the explicitly defined attack patterns, an alarm is raised. The defined attack patterns are frequently referred to as the signatures of intrusions. The signature could be a static string or a sequence of events. This type of detection method is called misuse detection [25].

Advantages of this type of approach are it is very efficient to detect known attacks and is very accurate with a very low false alarm rate. Since the attack patterns are comprehensively encoded in advance for matching against computer user activities on background traffic, the recognized attacks can be promptly identified. Once an attack is identified, the security administrator can quickly analyze the problem and make a correct action to prevent any breaches of security.

However, the main shortcoming of this approach is that it is only good for detecting known attacks. Once the attack pattern is slightly changed or a novel attack appears, the unseen attack will be considered as acceptable pattern and thus cannot be successfully detected. In addition, maintenance of the knowledge database is an extremely tedious and time-consuming task. It is very difficult to collect the required information of the known attacks since to label records of data as either normal or a specific type of attack requires careful analysis. Especially in this fast pace world, it is impossible to keep an intrusion detection system always up to date with all attacks and vulnerabilities information.

While knowledge-based intrusion detection is achieved by modeling known attack behavior, on the contrary, behavior-based intrusion detection also known as anomaly detection models normal or expected behavior of computer users. It looks for malicious activities by comparing the observed data with those acceptable behavior. If the data diverges from the learned normal behavior, an alarm is raised. Advantage of this approach is that novel and unseen attacks can be detected. Since it assumes any deviation from normal patterns is regarded as anomalous activities, the technique is not required to continuously keep up with hackers' techniques [26], [27]. Also, it is less dependent on target operating environments compared with the misuse detection technique.

The main drawback of this approach is it might have a high number of false alarms due to any deviations from the learned behaviors are treated as attacks. Since not every deviation is a real intrusion, the security administrator may spare precious time to take care of these false alarms and ignore the real anomalous activities.

2.4 Intrusion Detection Techniques

The popular intrusion detection techniques are reviewed in this section. We start with the expert systems based intrusion detection techniques. Expert systems are primary used in the design of misuse detection systems and most of them are rule-based systems. An expert system contains a set of predefined rules on the basis of knowledge of the intrusive activities. The inference engine then uses these rules to identify indications of known attacks from the background of network traffic. For example, SNORT [28] is a popular open-source network intrusion detection system of this kind. SNORT uses rules to describe attacks in which each rule uses a single line of text to explicitly describe the signatures of a certain attack. When monitoring the network traffic, SNORT compares traffic data with rule database and fires an alarm if a traffic matches SNORT's rule signatures. A sample SNORT rule [29] is shown in the following equation.

alert tcp !HOME_NET any –> HOME_NET any (flags: SF; msg: "SYN-FIN scan";) (2.1)

This rule fires an alert message "SYN-FIN scan" when an outsider attempts to make an internal home network TCP connection. For building such kind of rules, SNORT employs human knowledge to recognize those attempts of security breach. It provides a systematic search for attacks in the audit data, yet it will not flag alarm if the attack signatures are not described within the rules. For maintaining its up-to-date status, a

regular update is posted on the SNORT website. This update could be a very tedious and difficult task because the rules in the system must be reformulated by security professionals. In addition, the rule-based technique lacks flexibility in the rule-to-audit record representation [30]. Expert systems based approach suffers from its inability to identify attacks which differ from those predefined patterns. A minor variation of an attack itself or an attack sequence is possible to affect the rule comparison result and causes the attack never be found during the detection process.

Unlike the rule-based system that provides a set of predefined rules to identify indications of known attack activities, researchers also apply a variety of approaches to model the normal behavior of the protected system. In such approaches, neural networks and fuzzy logic are two well-known techniques in the development of intrusion detection systems. Neural network system acts as a computational model to process the network traffic information, which the system can be trained to perform intrusion detection tasks based on the traffic data provided. At the end of the training procedure, the neural network gains the knowledge that can extract the normal and attack signatures from the provided data automatically. With the ability to generalize rules from learned data, the neural network performs generalization of attacks and fault tolerance to imprecise and uncertain information.

Approaches using various neural network structures have been applied in building anomaly intrusion detection systems and the two most common architectures are the Self-Organizing Maps (SOM) [31] and the Multilayer Perceptrons (MLP) network [32]. A SOM uses unsupervised learning algorithm to group similar data to clusters in the input space. It is a data visualization technique that produces a low dimensional topological

map to help people understand the original high dimensional data. Once the neural network is trained, the map converges to a stationary distribution and shows a clear separation between normal and attacks. The output neurons are considered as the counts for normal and attacks. Next, future connections can be rapidly classified as normal or attacks by visualizing the histogram of the map. Examples can be found in the works of Kayacik et al. [7], Depren et al. [33], and DeLooze [34]. These researchers selected various subsets of features of *KDD99* data set to build different sizes of networks in order to simplify the complexity of the networks. Kayacik et al. built a hierarchical topological SOM maps for network intrusion detection. In the first layer, they selected six basic TCP features (length of the connection, protocol type, network service, status flag, total data bytes from source to destination, and total data bytes from destination to source) to build six 6×6 SOMs, each individual one was associated with each basic feature. The second layer integrated the information from the first level SOMs into a single view of the problem. Then in the third layer six SOMs were built for the second layer SOM neurons that demonstrated significant counts for both attack and normal connections. In each third layer SOM, 20×20 neurons were included. By using the same six basic features as Kayacik et al. did, Depren et al. built 15×15, 8×8, and 6×6 SOM maps for TCP, UDP (User Datagram Protocol) and ICMP traffic data, respectively. Each SOM structure was trained with the normal traffic data. When the training process was completed, any incoming anomalous traffic would be clustered outside of the normal clustering or inside the normal clustering with high quantization error. In the work of DeLooze, he created three 20×20 SOM maps using content, time, and connection features. Content features included number of total packets, acknowledgement packets, data bytes, retransmitted

packets, pushed packets, SYN and FIN packets flowing to or from the source and destination, and the status of completed, not completed and reset for each connection. Time features included the number of connections and the type of services to or from the source and destination within the last 5 seconds. Connection features were the duration of the connection, the service requested, and protocol used. Each feature space was used independently to detect anomalous behavior. Then the results of the individual SOMs were combined using the majority ensemble method (reports an attack if two of the three SOMs report an attack for a particular connection) and the belief ensemble method (reports an attack if any of the three SOMs reports an attack for a particular connection). Here, the problems shown in the above three works are how to select a feature space as the input to the network and how to configure a network with proper size. These two factors play important roles in the detection performance and the granularity of the network nodes, which training a SOM with a large amount of neurons needs long computational time and a SOM with a small volume of neurons may loss some important information. Also, the empirical nature of training parameters development is still an unresolved question, which the topology, learning rate and function, the number of training epochs, and initial weights of the network are decided by trial and error.

MLP uses a feedforward structure to solve classification problems by its supervised learning algorithm. The network weights are updated by using gradient-based optimization algorithm during the training period. When the network converges to the local minima of error, the output layer of the network will show the result when data is fed into the input layer. In the reported work of Faraoun and Boukelif [12], they applied *k*-Means algorithm to group the input data into a number of clusters. Having obtained the

clustering centers and their relative boundaries, the distances between the centers and input data were calculated and only the most discriminating samples that cover at maximum the region of each class were selected for the learning process. The selected samples were then presented to the MLP network for classifying four classes of attacks in the *KDD99* data set. Although they tried to improve the learning process by reducing the amount of training samples, the network still had a heavy computational burden because it was complex with 41, 30, and 5 neurons in input, hidden, and output layers, respectively. Also, the neural network suffers its "black box" nature that it does not provide explicit knowledge representation of its internal connections between layers. It is difficult to understand why a network event is classified as a normal or abnormal activity.

Fuzzy logic [35] is specifically designed to deal with imprecision of facts. With its capability of dealing with vagueness, there are several reasons it is a possible approach in the design of intrusion detection system. First, there is often no clear boundary between normal and abnormal of a computer user's activity [36]. Instead of a network traffic is either completely assigned to a member of normal category or a member of abnormal category, the traffic is possible classified into more than one categories. Fuzzy logic provides a dynamic decision boundary in the detection of intrusions. Second, the moment that we raise an alarm is often fuzzy. There would be too many alarms if we raise an alarm every time when we suspect an intrusion event occurs. At what degree of intrusion we should raise an alarm often depends on different situations. It depends on the degrees of intrusion and different circumstances [37]. With the dynamic decision boundary characteristic provided by fuzzy logic, the security officers can decide the best time to raise alarms according to the alarm threshold desired.

The work of Dickerson et al. [38] is an example of applying fuzzy logic to spot malicious activities against computers. In their work, they built an anomaly-based intrusion detection system named FIRE (Fuzzy Intrusion Recognition Engine). Initially the FIRE system applied data mining technique to TCP packet data to extract metrics. The metrics were chosen to reveal anomalies in the network traffic and formed the basis for the fuzzy inputs as well. For example, SDP was one of the metrics, that were composed of the IP source, IP destination, and the destination port fields. Once the system completed the metrics extraction, it used the historical data to calculate the ranges over the input space. All of the data were evaluated in terms of three fuzzy characteristics, COUNT, UNIQUENESS, and VARIANCE. Each input space had five fuzzy membership functions: LOW, MEDIUM-LOW, MEDIUM, MEDIUM-HIGH, and HIGH. With the use of metrics, three fuzzy characteristics, and five fuzzy membership functions, the authors then developed fuzzy inference rules to describe intrusions with their past experiences. For example, Equation 2.2 is a fuzzy inference rule to detect the port scan.

IF (COUNT OF SDPs = MEDIUM) AND (UNIQUENESS OF SDPs Observed = HIGH) THEN “Port Scan” = HIGH (2.2)

During the development of any fuzzy inference engine such as FIRE, the settings of fuzzy parameters, fuzzy characteristics and fuzzy membership functions, heavily depend on the experience of human experts. However, those settings are very critical to the result of fuzzy inference. It’s preferable if we can find a solution to automatically covert the professional’s expertise to a knowledge-based fuzzy inference machine. Hence, Dickerson et al. [39] improved the settings of five fuzzy membership functions by the use of fuzzy c-Means (FCM) clustering algorithm. They applied FCM to a data set and

obtained the clustering centers and membership grades. By the reference of clustering results, the extents and midpoints of the five fuzzy membership functions were determined. Methods such as neural networks [40], Support Vector Machine (SVM) [37], genetic algorithms [41], and data mining technique [42] had also been proposed to help the decision of the fuzzy parameters. All of them did improve the decision process of deriving necessary parameters of fuzzy logic, however more or less human expert's knowledge is still involved. Also, a large number of labeled data is needed during the design process.

With the fast growing of Internet and the large and complex network systems, it becomes impossible for a security officer to look for intrusive activities by manually analyzing the traffic data. Thus data mining technique has caught researchers' attention because it is capable of extracting useful information by sorting through a large amount of data. The intrusion detection is therefore treated as a data analysis process with a data-centric point of view [43]. The technique is defined as an information discovery task that looks for patterns in the network traffic data and can be applied to both misuse and anomaly intrusion detections. The data mining procedure from data collection to model computation can be made totally automatic. Unlike hand coding intrusion signatures into the systems [44], [45], data mining reduces the effort on manually analyzing and encoding intrusion patterns. However, it suffers from the degree of complexity if the raw data is formed by a great amount of data with a large number of features. For avoiding too many information included, feature selection or feature extraction technique is always applied to reduce the dimensionality of the original feature space.

Generally speaking, two data mining techniques, association rules and decision trees, are mostly applied to intrusion detection tasks. In the works of Lee and Stolfo [46], they created a framework: Mining Audit Data for Automated Models for Intrusion Detection (MADAM ID). Their idea was using frequent episode algorithm and association rules to compute patterns from system audit data and extract predictive features from the patterns. Then RIPPER classification algorithm [47] was applied to generate intrusion detection rules such as Equation 2.3, which represents the telnet connection and is a guessing password attack if the number of failed logins is greater than 4.

$$\text{guess:- failed\_logins} \geq 4 \tag{2.3}$$

Although the design process from extracting discriminative features to generating detection rules is totally automatic, the amount of labeled network traffic is usually large, which the expert-based labeling process is very tedious and time-consuming. In addition, labeling a large number of network traffic can possibly lead to errors [11].

The other typical data mining approach is associated with decision trees. Levin [6] used a data mining tool, Kernel Miner, to generate decision trees for classifying normal behavior and attacks in *KDD99* data set. By randomly choosing 10% data from the entire training data set, they constructed 218 decision trees for normal and four attack classes (*DoS, Probe*, *U2R*, and *R2L*). The result showed that the system achieved satisfactory detection rates on normal examples, *DoS* and *Probe* attacks, but failed on detecting *U2R* and *R2L* attacks. The system can only correctly detect 12.3% and 8.4% of *U2R* and *R2L* attacks, respectively, and misclassify most of them that belong to the new attack types not shown in the training set. Examples of using decision trees technique can also be found in the publications of Sabhnani and Serpen [9], [48]-[50]. In their works, they focused on

detecting both *U2R* and *R2L* attacks. They analyzed training and testing sets of *KDD99* data set through C4.5 decision trees algorithm [51] and concluded that no pattern classification algorithm or machine learning could be trained to successfully demonstrate misuse detection on both *U2R* and *R2L* attack categories. The reason is that not only these two attacks are content-based (embedded in the data portions of the packets) but also the testing set has extensive new types of attacks that are not correlated with attacks shown in the training set.

2.5 Feature Selection Techniques

For designing an intrusion detection system, a training set involving thousands of traffic connections is always required. In each traffic connection, it includes a number of features plus a class label of normal or a type of attack. By the use of misuse or anomaly detection technique, a model can be induced and used to classify future traffic into malicious activities or normal usage behavior.

Theoretically and ideally, the ability to discriminate attack from normal behavior should be performed better if more features are added during the detection process. However, the answer is sometimes negative. The reason is that some of the features may be irrelevant with poor prediction ability to the target class, and some of the features may be redundant due to they are highly inter-correlated with one of more of the other features [52]. Therefore, analysis of traffic features is a very critical step in the development of intrusion detection system. Of the large number of features included in the high dimensional data set, it is very important to have a good understanding which features are truly essential in detecting the attacks; which are less significant in only providing the

auxiliary information; and which are redundant that can be discarded. Based on the understanding of the significance of these features, irrelevant and redundant features can be discarded effectively. The remaining relevant features thus contain most significant information related to the given intrusion detection task. The feature selection result is helpful to speed up the detection time and to enhance the detection accuracy. The maximum overall performance can therefore be achieved.

Generally, the algorithms of feature selection are mainly divided into two categories, filter and wrapper, as defined in the work of John et al. [53]. Filter method operates without engaging any information of induction algorithm. By using prior knowledge such as features should have strong correlation with the target class or should uncorrelate to each other, filter method selects the best subset of features. Example is the work of Kayacık et al. [54]. They performed feature relevance analysis in the *KDD99* training set. In order to get feature relevance measure for attacks, they applied information gain to binary classification (normal and attack) and reported their chosen relevant features for normal behavior and part of the attacks. In their paper, they only reported the feature selection result but didn't demonstrate any evaluation of it. During the feature selection process, only the irrelevant features were considered to be removed. However, there was no description about the setting of threshold, which is critical in the elimination of the irrelevant features. Also, the information gain is possibly biased if feature with more values, i.e., the features with greater numbers of value will gain more information than those with fewer values even if the former ones are actually less informative than the latter ones.

On the other hand, wrapper method employs a predetermined induction algorithm to find a subset of features with the highest evaluation by searching through the space of feature subsets and evaluating quality of selected features. The process of feature selection acts like “wrapped around” an induction algorithm. Machine learning algorithms such as ID3 [55] and C4.5 [51] are commonly used as the induction algorithm. Since wrapper method includes a specific induction algorithm to optimize feature selection, it often provides an accurate classification result than that of filter approach. However, wrapper method is more time consuming than filter method due to it is strongly coupled with an induction algorithm with repeatedly calling the algorithm to evaluate the performance of each subset of features. It thus becomes unpractical to apply a wrapper method to select features from a large data set that contains numerous features and instances [56]. Furthermore, wrapper approach is required to re-execute its induction algorithm for selecting features from data set while the algorithm is replaced with a dissimilar one. It is less independent of any induction algorithms than filter is.

The work of Mukkamala and Sung [57] is an example of using wrapper method. With the use of *KDD99* data set, they applied both Support Vector Machines (SVM) and Support Vector Decision Function Ranking Method (SVDFRM) to rank important input features for the intrusion detection task. For each feature, it was deleted from the training and testing sets and the remaining ones were used to train the classifier. Then the classifier’s performance was compared with that of using full feature set. Finally, the importance of the feature was ranked according to a set of rules based on the performance comparison. Equation 2.4 shows one of the rules.

IF *accuracy* decreases AND *training time* increases AND *testing time* decreases (2.4)
THEN the feature is important

With its iterative search and evaluation procedure, the forty-one features were grouped into important features, secondary features, and unimportant features for normal, *DoS*, *Probe*, *U2R*, and *R2L* attacks. Table 2.1 shows their experimental results in which group 1 represents important features, group 2 represents secondary features, and group 3 represents unimportant features. This work used wrapper approach to execute its SVM-base induction algorithm. Without doubt, this process was computationally expensive to determine the final subset of features because the induction algorithms were iteratively executed on data sets that is with a large number of records and features.

Table 2.1. Experimental Results of the Work of Mukkamala and Sung

	SVM
Normal	{1,3,5,6,8-10,14,15,17,20-23,25-29,33,35,36,38,39,41}, <2,4,7,11,12,16,18,19,24,30,31,34,37,40>, (13,32)
Probe	{3,5,6,23,24,32,33}, <1,4,7-9,12-19,21,22,25-28,34-41>, (2,10,11,20,29,30,31,36,37)
DoS	{1,3,5,6,8,19,23-28,32,33,35,36,38-41}, <2,7,9-11,14,17,20,22,29,30,34,37>, (4,12,13,15,16,18,19,21,3)
U2R	{5,6,15,16,18,32,33}, <7,8,11,13,17,19-24,26,30,36-39>, (9,10,12,14,27,29,31,34,35,40,41)
R2L	{3,5,6,24,32,33}, <2,4,7-23,26-31,34-41>, (1,20,25,38)
	SVDFRM
Normal	{1-6,10,12,17,23,24,27,28,29,31-34,36,39}, <11-14,16,19,22,25,26,30,35,37,38, 40,41>, (7-9,15,18,20,21)
Probe	{1-6,23,24,29,32,33}, <10,12,22,28,34-36,38-41>, (7-9,11,13-21,25-27,30,31,37,40)
DoS	{1,5,6,23-26,32,36,38,39}, <2,3,4,10,12,29,33,34>, (7-9,11,13-22,27,28,30,31,35-37,40,41)
U2R	{1-6,12,23,24,32,33}, <4,10,13,14,17,22,27,29,31,34,36,37,39>, (7-9,11,15,16,18-21,25,26,28,30,35,38,40,41)
R2L	{1,3,5,6,32,33}, <2,4,10,12,22-24,29,31,34,36,37,38,40>, (7-9,11,13-21,25-28,30,35,39,41)

2.6 Multiple Classifiers Systems

Ensemble is to combine the outputs of a set of base classifiers together in a proper way when classifying input data. The fused result is expected to perform a better outcome than that of any individual base classifier within the ensemble. In the schemes of building an ensemble classifier, three distinct topologies are frequently engaged, they are cascading, parallel, and hierarchical structures [58]. Figure 2.2 illustrates these three basic frameworks. In the cascading structure, the output from the previous classifier is fed into the next one. By cascading all the classifiers together, the final result is obtained at the last classifier's output of the chain. While each previous classifier's output is the input of succeeding classifier, the latter classifier has difficulty to correct inaccuracy made by former one. In the parallel structure, the predictions of base classifiers are integrated to produce a fused output of the ensemble. The combination method is the key factor to decide if the result is successful or not. A careful choosing of combination methods can lead the ensemble classifier to a supreme performance, and on the contrary to poor consequence with an improper selection of combination methods. The hierarchical structure is a combination of cascading and parallel configurations. It is possible to alleviate both shortcomings of cascading and parallel structures and thus to achieve an optimal classification result.

The types of decision generated by the individual base classifier can be classified into three major categories: abstract form, rank level, and measurement level [59]. The abstract form is that a classifier only outputs a solitary class label for an input pattern. The rank level is that a classifier ranks a list of classes in accordance with the degrees of

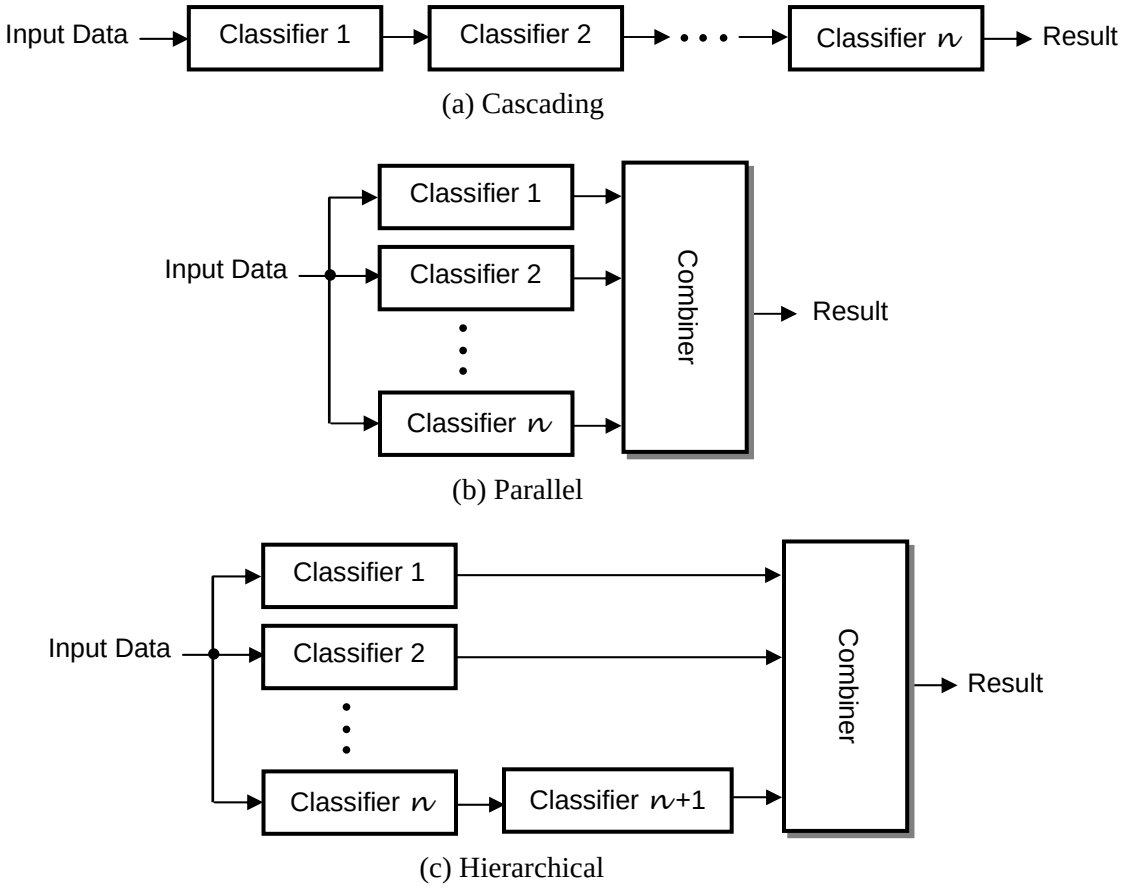


Figure 2.2. Three Topologies of Ensemble Classifiers

belief on classes the input pattern belongs to. The list is always sorted in descending way where the first and the last components are the highest and lowest ranked output classes, respectively. The measurement level is that the classifier assigns a level of confidence to each class for expressing the classifier's degree of belief for an input pattern. Among the combination methods that work with abstract form outputs, the popular methods are behavior knowledge space method, majority voting, weighted majority voting, naive bayes method. For measurement level outputs, the combination methods are Dempster-Shafer method, MAX, MIN, SUM, PROD, AVG, and MED methods that the ensemble

selects the maximum, minimum, summation, product, average, or median value of the combined classifiers as its output.

Example is the work of Giacinto and Roli [60]. In their research, they restricted the problem domain in the *ftp* service of the *KDD99* data set and selected 30 out of the 41 available features from the data set. They built three neural networks using 4 intrinsic features, 19 traffic features, and 7 content features, respectively. Also, they built one neural networks using 30 features for the sake of comparison. All of the networks were three layers fully-connected multi-layer networks, which each had 5 output neurons (for normal and four attack classes), a number of input neurons that equal to the number of features, and a hidden layer made up of 5 neurons for the networks using distinct features and 15 neurons for the network trained using all of the 30 features. For performing the ensemble operation, they carried out three fusion techniques: the majority voting rule, the average rule, and the belief function to combine the outputs from the networks trained on three distinct features together. The results showed that all of the fusion techniques improved the overall detection performance compared with those of individual classifiers and the classifier using 30 features. However, it also showed that the ensemble model did not improve the detection on unknown attacks in testing set, which had around 15% error rates. During the entire course of work, they only used 725 training connections and 7,436 testing connections but did not explain the reason. It explicitly hints that the neural networks could need a long time for training. The work of Mukkamala et al. [61] is another example using multiple classifiers approach. They used the *KDD99* data set and performed five-class (normal, *DoS*, *Probe*, *U2R*, and *R2L*) classification. They designed two ensemble models. One consisted of three multilayer feedforward neural networks and

the other was made up of neural networks, SVM and MARS (Multivariate Adaptive Regression Splines). By using the majority voting technique, individual base classifiers' outcomes of each ensemble model were combined together. The experimental results showed that each ensemble outcome outperformed that of its every base classifier. In one of their experiments, they fused three base classifiers' outputs with 48%, 0%, and 16% accuracies together and get 56% ensemble accuracy. However, Hansen and Salamon [62] had proved that multi-classifiers will only work when it is possible to build individual classifiers which are more than 50% accurate. But, in the paper they did not describe the input features of every individual base classifier. This is very important because base classifiers should be independent of each other, otherwise no improvement can be obtained through the combination.

CHAPTER III

CORRELATION-BASED FEATURE SELECTION

In an intrusion detection task, the quantity of network traffic data is enormous with a large amount of features. The objective of feature selection is to reduce the dimensionality of the original feature space with a way to select a subset of features. However, the problem is how to select the feature subset, which still can represent sufficient information about the original data set. For solving this problem, approaches based on information gain are employed in this dissertation in order to find the strength of predictive from features to targets and the strength of correlation between features themselves. As described in Section 2.5, the algorithms of feature selection are mainly divided into two categories, filter and wrapper. Since filter method is computational efficient than wrapper method, we choose it when the number of features is large. In the following, we will address aspects of feature selection based on filter method because the size of data collected from the network is always large which includes many traffic records with a number of various features. Our approach uses the concept of information theory to evaluate the worth of features and eliminate both irrelevant and redundant features. The approach is close to the Fast Correlation-Based Filter (FCBF) [63], however the difference is we treat the correlation between features in a global perspective. We measure the total amount of information associated with a feature as the summation of the inter-correlations to all of the rest of the features, but FCBF only considers on a feature of rest ones at a time. Figure 3.1 shows FCBF feature selection scheme.

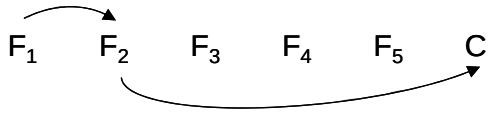


Figure 3.1. FCBF Feature Selection Scheme

First of all, the algorithm calculates the *symmetrical uncertainty* (*SU*) [65] value for each feature to the class *C*, selects the relevant features whose *SU* values are larger than a predefined threshold, and orders them in a descending order. In Figure 5, five features are selected as relevant features and ranked in descending order. In the second step, it processes the relevant features to remove the redundant ones. The algorithm starts from the left-most feature F_i and calculates its *SU* value with the remaining ones (the one F_j right next to F_i to the last one). If $SU_{i,j} \geq SU_{j,c}$, F_j is considered as a redundant feature and is removed from the list. After one round of removing features based on F_i, the algorithm takes the feature right next to F_i as the new reference to repeat the process. It stops until no more features can be selected. By using this technique, some so called redundant features can be removed quickly. However, FCBF may be tricked in a situation where the dependence between a pair of features is weak but the total inter-correlated strength of one feature to the others is strong. So the FCBF possibly will keep a feature that its information can be found in the remaining selected subset of features. In addition, FCBF requires adjusting a threshold for its feature selection procedure, while our algorithm does not.

In the following sections, we first introduce the theoretical framework which forms the base of our proposed approach in measuring the goodness between features, and between

features and classes. We then describe our proposed feature selection algorithm in Section 3.2.

3.1. Theoretical Framework

In information theory, *entropy* [64] is a measure of the amount of uncertainty about a source of messages. The entropy of variable Y before and after observing values of another variable X can be described by.

$$H(Y) = -\sum_i p(y_i)\log_2 p(y_i) \tag{3.1}$$

and

$$H(Y|X) = -\sum_j p(x_j)\sum_i p(y_i|x_j)\log_2 p(y_i|x_j) \tag{3.2}$$

Here $p(y_i)$ is the prior probabilities for all values of random variable Y and $p(y_i|x_j)$ is the conditional probability of y_i given x_j. By treating Y as classes and X as features in a data set, the entropy is 0, i.e., without any uncertainty at all if all members of a feature belong to the same class. On the other hand, members in a feature set are totally random to a class if the value of entropy is 1. The range of entropy is between 0 and 1.

The amount by which the entropy of Y decreases reflects additional information about Y provided by X. This is called *information gain* (or *mutual information*) [55] as shown in Equation 3.3.

$$\begin{aligned} I(Y;X) &= H(Y) - H(Y|X) \\ &= H(X) - H(X|Y) \end{aligned} \tag{3.3}$$

It measures how well a given variable separates instances into another variable. The function $I(Y; X)$ is symmetrical, i.e., the amount of information gained about Y after

observing X is equal to the information gained about X after observing Y. Symmetry is a desired property for correlation measurements between features. However, information gain is biased if feature with more values, which the features with greater numbers of values will gain more information than those with fewer values even if the former ones are actually less informative than the latter ones. Also, the range of information gain is not from 0 to 1. Its values should be normalized in order to ensure they are comparable and have the same affect. Therefore, we choose SU as our tool to find the strength of predictive from features to target classes and that of correlations between features themselves. Its definition is shown in the following equation.

$$SU(Y;X) = 2 \cdot \left[\frac{I(Y;X)}{H(X) + H(Y)} \right] \tag{3.4}$$

It averages the values of two uncertainty variables, compensates for information gain's bias toward features with more values, and normalizes its values to the range [0, 1]. A value of 1 indicates that knowing the value of either one completely predicts the value of the other and a value of 0 indicates that X and Y are independent each other. In addition, it still treats a pair of features symmetrically.

In the following study, we apply SU measure to calculate the correlation between features and target class. If a feature has a low SU to the target class, it implies that the feature has poor prediction ability to the class. On the other hand, the feature has strong prediction ability to the class if the SU is high. Once having all the symmetric uncertainties between features and the target class, features can be ranked in descending order according to their degrees of association to the target class Y. Those features which have the lowest ranks are considered as irrelevant features and will be filtered out.

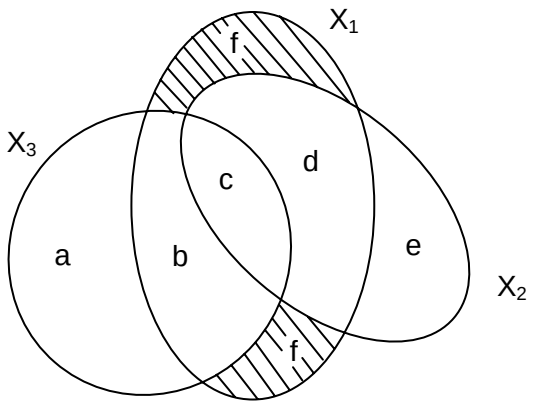


Figure 3.2. Illustration of Correlations of Three Features in Venn Diagram

Similarly, we apply *SU* measure to pairs of features. If the measure of mutual information between a pair of features is low, it represents these two features are independent to each other, i.e., knowing one feature cannot provide any information about the other. On the contrary, the two features are highly inter-correlated with each other if they have a high mutual information measure. This means that one feature contains similar information about the other and implies that knowing one feature can gain necessary information about the other. Under this circumstance, one of them can be considered as a redundant feature and can be discarded.

For a better understanding of the idea of redundant feature, we use Venn diagram to illustrate correlations among multiple features X_1, X_2 and X_3. As shown in Figure 3.2, $SU(X_1; X_3) = b + c$, $SU(X_1; X_2) = c$ and $SU(X_2; X_3) = c + d$. Obviously, some redundant information will be included if we choose all three features since the information included in $SU(X_1; X_3)$, $SU(X_1; X_2)$, and $SU(X_2; X_3)$ is $b + 3c + d$, which is greater than $b + c + d$ in $SU(X_1, X_2; X_3)$. Therefore, removing redundant features from the original feature space is necessary in order to discard needless information. The intrusion detection

processing time can therefore be reduced by the use of a subset of the original feature space.

In Figure 3.2, feature X_3 is highly inter-correlated with both features X_1 and X_2. By using Shannon's information-theoretic measure, we get the joint entropy:

$$\begin{aligned} H(X_1, X_2, X_3) &= (a + b + c + d + e) + f \\ &= H(X_1, X_2) + [H(X_3) + SU(X_1, X_2; X_3) - SU(X_1; X_3) \\ &\quad - SU(X_2; X_3)] \end{aligned} \tag{3.5}$$

The larger mutual information $SU(X_1; X_3)$ and $SU(X_2; X_3)$ are, the smaller area f will be. When f is very small, it represents that X_3 heavily depends on both X_1 and X_2. The measure of joint entropy $H(X_1, X_2, X_3)$ is approximately equal to $H(X_1, X_2)$. It implies that the total amount of information of X_1, X_2, and X_3 can be represented by the amount of information of X_1 and X_2. Feature X_3 is considered as a redundant feature and can be removed with only losing a little information of the original feature space. Finally, we select the feature that is neither an irrelevant feature nor a redundant feature and call this type of feature as "significant feature".

3.2 Feature Selection Algorithm

Table 3.1 describes our proposed feature selection algorithm. The algorithm mainly consists of two parts for achieving the goal of reducing dimensionality of the original feature space. In the first part (lines 1-5), the algorithm removes irrelevant features with poor prediction ability to target class. Given a data set with a number of input features and a target class, the algorithm first calculates the mutual information between features and class. The algorithm then ranks the features in descending order according to their

Table 3.1. Feature Selection Algorithm

1 // Remove irrelevant features
2 Input original data set D that includes features X and target class Y
3 For each feature X_i
    Calculate mutual information $SU(Y; X_i)$
4 Sort $SU(Y; X_i)$ in descending order
5 Put X_j whose $SU(Y; X_i) > 0$ into relevant feature set R_{XY}
6 // Remove redundant features
7 Input relevant feature set R_{XY}
8 For each feature X_j
    Calculate pairwise mutual information
    $SU(X_j; X_k)\ \forall j \neq k$
9 $S_{XX} = \Sigma(SU(X_j; X_k))$
10 Calculate means μ_R and μ_S of R_{XY} and S_{XX}, respectively.
    $w = \mu_S / \mu_R$
11 $R = w \cdot R_{XY} - S_{XX}$
12 Select X_j whose $R > 0$ into final set F

degrees of association to the target class. Once the input features' degrees of importance are ranked, those terms whose information measure are equal to zero are removed.

The second part of the algorithm (lines 6-12) eliminates redundant features that are inter-correlated with one or more of other features. It starts with calculating the inter-correlated strengths of each pair of features. The total amount of mutual information for each feature is acquired by adding all mutual information measures together that relate to that feature. For adjusting the discriminative power of mutual information performed on feature-to-feature and feature-to-class to the same level, we introduce factor w and its

value is equal to the mean of summation of feature-to-class information divided by the mean of summation of feature-to-feature information. By multiplying *w* to each feature-to-class measure, both feature-to-class and feature-to-feature reach to the same important rank. Finally, the differences of them are computed and we only keep those features whose values are greater than zero; which means the selected features are the most "significant features" that restrain indispensable information of the original feature space.

CHAPTER IV

FUZZY BELIEF *k*-NN ANOMALY DETECTION

This chapter presents an intrusion detection method named fuzzy believe *k*-NN anomaly detection. It is a combination of fuzzy clustering technique and Dempster-Shafer theory since both of them have merit of resolving the uncertainty problems caused by limited and ambiguous information during a decision process. Also, the *k*-NN technique is applied to speed up the detection process.

In the first part of this chapter, we describe the reason of choosing anomaly detection technique. Then, we explain the importance of considering the problems of uncertainty while designing intrusion detection models. Finally, we present our proposed fuzzy believe *k*-nearest neighbors anomaly detection design.

4.1 Anomaly Detection

As described in Section 2.3, two approaches are typically used for detecting intruders of the information from network traffic or system audit trail. They are misuse detection and anomaly detection. Misuse detection models known attack behavior and anomaly detection models normal behavior. The main drawback of misuse detection technique is that it cannot detect unknown intrusions. Whenever a novel attack is discovered, it is necessary to spend a number of hours or days on the development of this new attack signature and then to update it manually into the intrusion detection system. Maintenance of the knowledge database therefore becomes an extremely tedious task. Moreover,

misuse detection approach becomes impractical while the number and types of intrusions increase dramatically with the networks grow rapidly. Human analysis becomes insufficient to catch the growing speed of intrusions.

A misuse intrusion detection system can neither cover all intrusive behavior space nor include all normal behavior space. This is due to the fact that there is not only a large amount of vulnerabilities that already have been discovered [1] but also an unknown number of vulnerabilities that may be immediately exposed. So it is very difficult to model such behavior spaces completely and correctly in reality. Additionally, computer attacks are usually polymorphic [66]. Computer hackers in general use different approaches to exploit a same vulnerability. The attack codes may look different from the known signature but are functionally equivalent. For example, the Internet worms are polymorphic and spread automatically across networks by exploiting vulnerabilities [67]. These worms are able to mutate as they spread across the network by using self-encryption mechanisms or semantics-preserving code manipulation techniques. Hence, it is correspondingly difficult to generate all possible combinations to cover the variations of attacks using misuse detection technique. It is necessary to develop an efficient way that is able to identify different variations of a same type of attack.

To address the above problems, an obvious solution would be to develop intrusion detection systems using anomaly detection, which are totally orthogonal to misuse based models. The anomaly based models have been successfully implemented by modeling what is normal instead of what is anomalous. It is advantageous to distinguish any deviations from normal behavior. Consequently, unusual or abnormal patterns are possible to be discovered. Furthermore, it offers the ability to resist polymorphic attacks

at the moment that novel attacks are constantly being introduced to the networks today. The anomaly based models provide a much more feasible approach by the use of generalizing the signatures of attacks than generating a number of signatures that cover possible variations of attack as in misuse based models.

4.2 Handling of Uncertainty

Uncertainties exist in our daily life. Sometimes the uncertainty is totally random, e.g., the future state of weather and the occurrence of failure of our home appliances. In other occasions it happens due to lack of knowledge or unpredictable factors such as the trend of stock and whether a war is going to happen. Therefore, people generally classify uncertainties into two categories, aleatory uncertainty and epistemic uncertainty, based on their fundamental difference in nature. Aleatory uncertainty is also known as variability, random uncertainty, stochastic uncertainty, objective uncertainty, and irreducible uncertainty [68], [69]. It is caused by inherent random variations associated with the physical system or the environment under consideration. Examples can be found in the outcomes while rolling a dice, the location and time of occurrence of future earthquakes, and the variability of a machining operation. The random nature of aleatory uncertainty is inherent. The occurrence of an event is not predicable even a large quantity of past data is collected.

The second type of uncertainty is epistemic uncertainty. This uncertainty is also referred to as imprecision, reducible uncertainty, subjective uncertainty, parameter uncertainty, model form uncertainty, and state-of-knowledge uncertainty [68], [69]. On the contrary to aleatory uncertainty that uncertainty arises from the system itself, epistemic uncertainty is

an uncertainty that is due to a lack of knowledge or information of processes of the system or the environment. Since it is not caused by the inherent random variations of the system but by the incomplete information or knowledge, the uncertainty is possible to be reduced by including new knowledge or information about the system or environmental factors. Examples of epistemic uncertainty can be seen when there are insufficient experimental data to describe physical parameters of a new material, limited understanding of a physics phenomena, and imperfect measurement of a complex physical model.

Actually, epistemic uncertainty does happen in intrusion detection tasks. From the decision-based perspective, the goal of intrusion detection is to make decisions on whether future traffic data are malicious or normal. For effectively and precisely making the decisions, data are collected in advance for analysis in either misuse or anomaly detection case. However, the collected data always enclose uncertainty when only limited information about intrusive activities is available. In real world modern computer systems and networks, hackers constantly develop new attack codes to exploit security vulnerabilities of organizations everyday. Not only are these attacks becoming more numerous, they are also becoming more sophisticated. Accordingly, it is not realistic to cover all intrusive behavior space completely for the use of decision making in an intrusion detection system.

Uncertainty is also occurred due to ambiguous information about computer users' activities. The patterns generated from users' behavior cannot be specifically defined as normality and abnormality. In order to illustrate this type of uncertainty, let us consider the following example of a person who tries to access an account from a remote machine.

A user attempts to retrieve forgotten passwords when he/she logins his/her own account, this action is considered as a normal behavior. On the other hand, the action that a hacker attempts to access other people's accounts by guessing passwords is definitely an intrusive activity. Thus, uncertainty is involved during the process of classification. If the guessing passwords behavior of a hacker is considered as a normal activity, then the intrusion can never be detected. If the retrieving forgotten passwords behavior of a user is considered as an intrusive activity, then the system administrators may fire an alarm but actually there is no intrusion happened. Hence, uncertainty is necessary to be concerned in the imprecise available data set during the intrusion detection procedure.

4.3 Approach

Intrusion detection in fact is a classification task that classifies network traffics into normal usage category or attack category. In our work, the main goal is to identify *U2R* and *R2L* attacks from the *KDD99* intrusion detection benchmark data set. For successfully achieving the goal, we divide the development of an intrusion detection system into two phases: training phase and classification phase. In the training phase, decision rules are generated in accordance with the clustering result of provided training data. The rules are used for classifying future network traffic whether is a normal activity or an attack in the classification phase. Figure 4.1 depicts the general operation scheme of the proposed approach. The details are described as follows.

A. The Training Phase

Let us assume the available information in a given training set is from a network with N traffic connections, and each of them is composed of n distinct features with positive

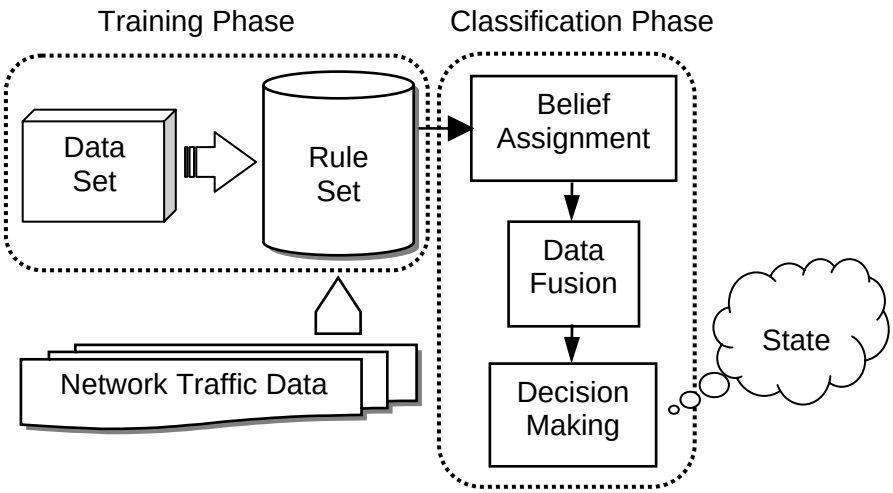


Figure 4.1. Intrusion Detection Scheme

numeric values. We denote the training set as *T*, the training traffic connection as *x*, and the set of features in each connection as *F*. Equations 4.1 and 4.2 denote *T* and *F*, respectively.

$$T = \{x_1, x_2, ..., x_N\} \tag{4.1}$$

and

$$F = \{f_1, f_2, ..., f_n\} \tag{4.2}$$

As described in the previous section, a training traffic connection sometimes could not be crisply defined as normality or abnormality. The boundary between normal activities and abnormal ones are always unclear. Crisp clustering algorithms cannot handle this ambiguity problem among network activities. Therefore, we decide to apply fuzzy c-Means (FCM) clustering technique developed by Dunn in 1973 [14] and improved by Bezdek in 1981 [13] to the following study. It allows one piece of data with gradual memberships to the clusters rather than completely assigning to just one cluster. By using this feature of FCM, the problem of ambiguity between attacks and normal activities can

be solved. The connection could be assigned to diverse classes with different degrees of memberships. We denote the set L as a number of p possible classes.

$$L = \{ l_1, l_2, ..., l_p \} \tag{4.3}$$

The clustering procedure is done by using iterative optimization technique to minimize an objective function J.

$$J = \sum_{i=1}^{N} \sum_{j=1}^{p} u_{ij}{}^{\sigma} \left\| x_i - c_j \right\|^2 \tag{4.4}$$

where the parameter σ is a weighting exponent on each fuzzy membership and has a value in the range [1, ∞). This parameter determines the amount of fuzziness in the classification process. When it is set to 1, the FCM approaches a hard c-Means algorithm, i.e., the membership grade assigning to cluster is either 0 or 1. As this parameter becomes larger, the fuzzier are the membership assignments to the clusters. Also, convergence of the algorithm tends to be slower as the value of σ increases. Normally, its value is in the range of 1.25 to 2 [70]. x_i is the i^{th} connection of the training set, c_j is the center of cluster j, and u_{ij} is the membership grade of x_i in the cluster j with a value between 0 and 1. || || denotes norm expressing the distance between any measured data and the cluster center. The membership grades u_{ij} and cluster centers c_j are updated by the following expressions,

$$c_j = \frac{\sum_{i=1}^{N} u_{ij}{}^{\sigma} x_i}{\sum_{i=1}^{N} u_{ij}{}^{\sigma}} \tag{4.5}$$

and

$$u_{ij} = \frac{1}{\sum_{k=1}^{p} \left(\frac{\|x_i - c_j\|}{\|x_i - c_k\|} \right)^{\frac{2}{\sigma - 1}}} \tag{4.6}$$

By iteratively updating the cluster centers and the membership grades for each training connection, FCM moves the cluster centers gradually to their correct values. Finally, the iteration stops when $max_{ij} \left| u_{ij}^{(k+1)} - u_{ij}^{(k)} \right| < \varepsilon$, where ε is a selected threshold for terminating the iteration process and *k* denotes the number of iterations.

The connection that lies "closer" to the center of a class has a higher membership grade to that class. On the contrary, the connection that lies "farther" away from the center of a class has a lower membership grade to that class. Training connections are grouped into *p* classes such that each connection has a certain membership grade to every class. The set of cluster centers *C* and membership partition matrix *U* are shown below.

$$C = \{c_1, c_2, ..., c_p\} \tag{4.7}$$

and

$$U = [u_{i1}, u_{i2}, ..., u_{ip}] \tag{4.8}$$

where *i* is the connection number of the training set and *p* is the number of possible classes. For each cluster center, it has a number of *n* values.

Within each row of *U*, the *p* membership grades are treated intuitively to be our degrees of confidence on *p* classes that a connection can belong to. Consequently, we can build *p* decision rules from a connection and each consists of a number of feature values *F*, a class label *l*, and a confidence value α.

$$R_U = \{r_U\} \quad where \quad r_U : \langle F, l \rangle, \ \alpha \tag{4.9}$$

The confidence values are in proportion to the correspondent membership grades that a connection belongs to certain classes. For a training connection, only portion of our belief is devoted to a certain class in a rule whereas the rest of beliefs are committed to other classes in other rules. The summation of the degrees of confidence on rules that generated from a training connection must be equal to 1. It is not possible that the connection can belong to any other classes except these *p* classes.

$$\sum_{j=1}^{p} \alpha_{ij} = 1 \tag{4.10}$$

where *i* is the connection number and *j* is the class number. Since the training set has *N* connections and each contains a number of *p* membership grades, totally *N* times *p* decision rules can therefore be generated.

In addition to the rules created from membership partition matrix *U*, a number of *p* rules are generated from the cluster centers. In each rule, the antecedent part includes *n* values of a cluster center and the corresponding class label. The degree of confidence is designated to 1 because we have full confidence that the cluster center should belong to that partitioned class without any doubt.

$$R_C = \{r_C\} \quad where \quad r_C : \langle c, l \rangle, \ \alpha = 1 \tag{4.11}$$

With Equations 4.9 and 4.11, totally (*N*+*1*)*p* rules are included in the decision rule set *R*. These rules will act as pieces of evidence to assign beliefs to an incoming connection in the decision making stage.

$$R = R_U \bigcup R_C \tag{4.12}$$

B. The Classification Phase

In the classification problem of intrusion detection, a complete prior knowledge regarding the probability distributions of attacks and normal behavior is not available. Also, the amount of traffic data for design is always limited. Hence, we decide to incorporate Dempster-Shafer theory into this phase because it does not require an assumption regarding the probability of the individual classes. It computes the probability that evidences support the attack or normal class. Also, this theory offers a solution for the mathematical representation of uncertainty. It is suitable for anomaly detection on unseen network traffic by using limited information on the uncertainty. With the combination of accumulative evidences from an insufficient amount of information, it is capable of making decision on a traffic whether it is normality or abnormality. In this phase, the pieces of evidences will be derived from the decision rules of the training phase.

Dempster-Shafer theory also known as *Evidence Theory* or *Theory of Believe Functions*, was introduced by Glenn Shafer in the late 1970s [16] based on the work of Arthur Dempster [15]. It starts by defining a sample space named *frame of discernment* (or simply *frame*), which is a finite set of mutually exclusive and exhaustive hypotheses in a problem domain under consideration. For adopting the theory into our intrusion detection design, we identify the set of class labels L as the *frame* of the problem domain. The possible subset A of L represent hypothesis that one could present evidence. The set of all possible subsets of L, including itself and the null set $\varnothing$, is called a *power set* and designated as 2^L. Assume v be an incoming traffic connection to be classified. To classify v means to assign it to one of the members in L, i.e., to assign v to a member of p classes: $v \in l_q$, $q = 1, 2, \ldots, p$.

A piece of evidence that influences our degree of belief on a hypothesis can be quantified by a *mass function* which is denoted as $m(\cdot)$. It is a mapping function and defined as m: $2^L \rightarrow [0, 1]$ such that

$$\sum_{A \subseteq L} m(A) = 1 \tag{4.13}$$

and

$$m(\varnothing) = 0 \tag{4.14}$$

where $A \subseteq L$ is called a *focal element* of m if $m(A) > 0$. The quantity $m(A)$ is defined as the hypothesis A's *basic probability assignment*. It can be interpreted as the portion of total belief to hypothesis A given the available evidence. For example, if $m(A) = 0.2$, then it means that one's belief committed to A is 20%. The left 80% beliefs are committed to other focal elements of *frame L*.

We treat the set of decision rules as pieces of evidence that alters our degrees of belief to which class v should belong while classifying it into the correct class. If the distance is large between v and a decision rule, it represents that v is "far" from the rule, i.e., the rule only has a little influence on v. On the other hand, we have stronger belief that v should belong to the same class of the rule if v is "close" to it, which means the distance has a smaller value. Here, distances from v to all decision rules are computed and the most informative rules are selected. By using this technique, the computational time is less than that of using the whole set of rules. Additionally, the weighted k-NN rule [70] is used to assign different weights to the selected rules.

$$w_i = \begin{cases} \dfrac{d(x_k, v) - d(x_i, v)}{d(x_k, v) - d(x_1, v)} & d(x_k, v) \neq d(x_1, v) \\ 1 & d(x_k, v) = d(x_1, v) \end{cases} \tag{4.15}$$

where x_i is the i^{th} rule, x_k and x_1 are the farthest and nearest rule of v, respectively, and d is the Euclidean distance between v and a rule. This weighting factor is used to give each decision rule a different amount of influence in a way that closer rule to v has larger influence. The factor is calculated such that the nearest neighbor of v has a weight value of 1 and the farthest k^{th} neighbor has a value of 0. Since the range of this factor is from 0 to 1, the resulting weights possibly have very similar values. Therefore, for further differentiating the rules' degrees of importance to v, the confidence value α is added to alter the degree of our belief on v.

$$m(l_q) = w_i \cdot \alpha_i \tag{4.16}$$

where i is the rule number and q is the corresponding class number of the i^{th} rule. Up to this stage, each rule creates a number of belief assignment indicating the degrees that v belongs to certain classes. If the value of m is large, it means that we have a strong belief that v belongs to the class of which m indicates. Otherwise v should belong to other classes if m is small. Nevertheless, we need to notice that a belief should also be designated to the *frame* (with every class labels). The reason is that only part of our beliefs is committed to single class for a given training connection, and the rest of our belief should be assigned to the *frame*. According to Dempster-Shafer theory, the summation of all mass functions inferred from one training connection is equal to 1.

Thus, the belief belonged to the *frame* becomes one minus the summation of beliefs of all of the single class.

$$m(L) = 1 - \sum_{i=1}^{p} m_i(l_q) \tag{4.17}$$

From the mass function given by Equation 4.16, the belief function *Bel* and plausibility function *Pl* can be derived to characterize certain hypotheses. They are shown in the following equations,

$$Bel(l_j) = m(l_j) \tag{4.18}$$

$$Pl(l_j) = 1 - Bel(\bar{l}_j) \tag{4.19}$$

where j is class number and $\bar{l}_j$ is the hypothesis "not l_j" with value between 0 and 1. Belief function is a measure of the total amount of belief that directly supports for a given hypothesis. The greater the support assigns to a hypothesis, the higher belief that the hypothesis is true. It can be regarded as a lower bound that indicates the impact of evidence of the hypothesis. Plausibility quantifies the extent to which one doubts the hypothesis. It shows the belief on the given hypothesis can only up to this value, which is an upper bound on the belief. The gap between them indicates the uncertainty about the hypothesis. It is a good reference in deciding whether more evidences are needed or not. Haralick and Shapiro [72] represent those various measurements over the interval unit as shown in Figure 4.2.

Now let us assume that the *frame* of the problem domain includes normal and attack classes. A network traffic connection is coming and the goal is to decide whether it is a normal activity or an attack by the use of belief and plausibility functions. Suppose we have two pieces of evidence regarding the connection and the mass functions are 0.1 and 0.2 for normal class and attack class, respectively. By using Equations 4.18 and 4.19, the belief and plausibility that support for normal class are 0.1 and 0.8 and for the attack class are 0.2 and 0.9, respectively. From the observation of the gap between belief and

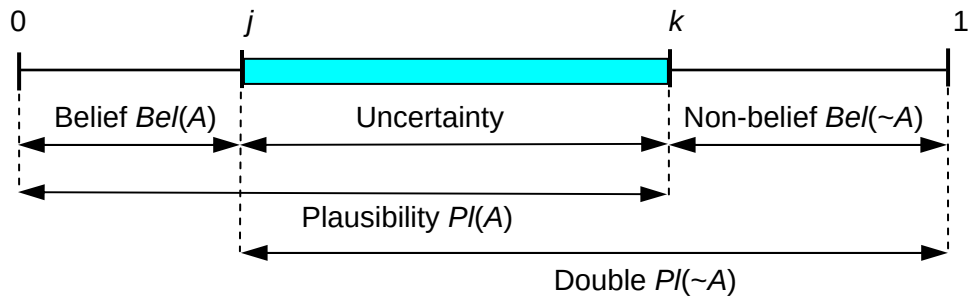


Figure 4.2. Functions of Belief and Plausibility

plausibility, it has a high degree of uncertainty. This indicates that more evidences are required to be incorporated so that we can make a better decision that the connection is a normal activity or an attack.

Generally speaking, the mass function is a piece of evidence that supports certain hypothesis concerning to the class member of a rule. When more evidences appear with same class label, those evidences can be integrated to generate a single belief function which represents the total support for the same class. *Dempster Rule of Combination* is applied here to combine all the beliefs induced from distinct pieces of information with same class label together. Using this combination rule, the final belief on every subset of class set can be obtained. In our case, a number of belief functions for single classes and one belief function for the class set will be generated.

Now assume that there are two mass functions m_1 and m_2 induced by distinct items of evidence X and Y. By using *Dempster Rule of Combination*, these two independent evidences can be fused into a single belief function Z that expresses the support of the hypotheses in both evidences. The combination result is called *orthogonal sum* of m_1 and m_2 and noted as $m = m_1 \oplus m_2$.

$$m(Z) = \frac{\sum_{X \bigcap Y = Z} m_1(X) \cdot m_2(Y)}{\sum_{X \bigcap Y \neq \varnothing} m_1(X) \cdot m_2(Y)} = \left(\sum_{X \bigcap Y = Z} m_1(X) \cdot m_2(Y) \right) \cdot k^{-1} \tag{4.20}$$

where

$$k^{-1} = \left(\sum_{X \bigcap Y \neq \varnothing} m_1(X) \cdot m_2(Y) \right)^{-1} = \left(1 - \sum_{X \bigcap Y = \varnothing} m_1(X) \cdot m_2(Y) \right)^{-1} \tag{4.21}$$

where the factor k^{-1} is the *renormalization constant*. Using the above equations, the final belief on single class and the *frame* are obtained. In an intrusion detection task, a number of p belief functions for single classes and one belief function for class set will be generated. For example, totally four final belief functions are obtained if there are three classes in the *frame*. There are three belief functions for single class and one belief function for the *frame*. They give fused allocations of belief and emphasize the agreement between multiple sources.

Let us continue with the previous example that we already have two pieces of evidence regarding a connection. The mass functions of corresponding evidences are 0.1 and 0.2 for normal class and attack class, respectively. Now assume that we have two more pieces of evidence regarding the same traffic connection and the correspondent mass functions are 0.3 and 0.6 for normal class and attack class, respectively. Table 4.1 shows the information of this example. The *frame* is

$$L = \{l_1, l_2\} = \{N, A\} \tag{4.22}$$

By using *Dempster Rule of Combination*, the above evidences can be aggregated into two fused belief functions *Bel*(*N*) and *Bel*(*A*). First, the *renormalization constant* factor k^{-1} is calculated in Equation 4.23. Then, individual fused mass functions can be obtained by using Equation 4.24. Equations 4.25 to 4.27 show the fused results.

Table 4.1. Connection Information of the Example

	$m_1(N) = 0.1$	$m_1(A) = 0.2$	$m_1(N, A) = 0.7$
$m_2(N) = 0.3$	$m(N) = 0.03$	$m(\mathrm{N} \cap \mathrm{A}) = 0.06$	$m(N) = 0.21$
$m_2(A) = 0.6$	$m(N \cap A) = 0.06$	$m(A) = 0.12$	$m(A) = 0.42$
$m_2(N, A) = 0.1$	$m(N) = 0.01$	$m(A) = 0.02$	$m(N, A) = 0.07$

* Normal and Attack are abbreviated as *N* and *A*, respectively.
Uncertainty between belief and plausibility is abbreviated as *U*.

$$
\begin{aligned}
k^{-1} &= \left(\sum_{N \cap A \neq \varnothing} m_1(N) \cdot m_2(A) \right)^{-1} \\
&= \left(1 - \sum_{N \cap A = \varnothing} m_1(N) \cdot m_2(A) \right)^{-1} \\
&= \left(1 - [m_1(N) \cap m_2(A) + m_1(A) \cap m_2(N)]\right)^{-1} \\
&= \left(1 - (0.06 + 0.06)\right)^{-1} \\
&= 1.14
\end{aligned}
\tag{4.23}
$$

$$m(l_q) = \left[m_1(l_q) \cdot m_2(l_q) + m_1(l_q) \cdot m_2(L) + m_1(L) \cdot m_2(l_q) \right] \cdot k^{-1} \tag{4.24}$$

$$m(N) = m_1 \oplus m_2(N) = (0.03 + 0.01 + 0.21) \cdot k^{-1} = 0.28 = Bel(N) \tag{4.25}$$

$$m(A) = m_1 \oplus m_2(A) = (0.12 + 002 + 0.42) \cdot k^{-1} = 0.64 = Bel(A) \tag{4.26}$$

$$m(L) = m(N, A) = m_1 \oplus m_2 (N, A) = [m_1(N, A) \cdot m_2(N, A)] \cdot k^{-1} = 0.07 \cdot k^{-1} = 0.08 \tag{4.27}$$

The two fused belief functions $m(N)$ and $m(A)$ express the total support of normal class and attack class, respectively. The plausibility and uncertainty functions for both normal and attack classes can be derived using Equations 4.28 to 4.31.

$$Pl(N) = 1 - Bel(\overline{N}) = 1 - Bel(A) = 1 - 0.64 = 0.36 \tag{4.28}$$

$$Pl(A) = 1 - Bel(\overline{A}) = 1 - Bel(N) = 1 - 0.28 = 0.72 \tag{4.29}$$

$$U(N) = Pl(N) - Bel(N) = 0.36 - 0.28 = 0.08 \tag{4.30}$$

$$U(A) = Pl(A) - Bel(A) = 0.72 - 0.64 = 0.08 \tag{4.31}$$

Table 4.2. Data Fusion Result

	$\{N\}$	$\{A\}$	$\{N, A\}$
m_1	0.1	0.2	0.7
Bel_1	0.1	0.2	1
Pl_1	0.8	0.9	1
m_2	0.3	0.6	0.1
Bel_2	0.3	0.6	1
Pl_2	0.4	0.7	1
m	0.28	0.64	0.08
Bel	0.28	0.64	1
Pl	0.36	0.72	1
U	0.08	0.08	
Bp	0.32	0.68	

The gap between belief and plausibility is 0.08. We can tell that uncertainty is reduced significantly after incorporating more evidences. We have stronger believe that the connection should be an attack.

At the data fusing level, each piece of evidence initializes the finite amount of belief to hypotheses of the *frame*. Part of the belief is allocated to the single class and part of it is allocated to the *frame*. To decide which class *v* should belong to, Equation 4.32 shows the *pignistic probability function* and it is applied to make the final decision.

$$Bp(l_q) = m(l_q) + \frac{m(L)}{p} \tag{4.32}$$

where q is the class number and p is the number of classes. The function quantifies our beliefs to individual classes with pignistic probability distribution. These probabilities distributed from zero to one and the summation of them equals to one. For making an optimal decision, v is assigned to a class with the highest pignistic probability. Continue with the example, the degrees of final belief on normal and attack classes are shown in Equation 4.33 and 4.44, respectively. Table 4.2 shows the computation result.

$$Bp(N) = m(N) + m(L)/2 = 0.28 + 0.08/2 = 0.32 \quad (4.33)$$

and

$$Bp(A) = m(A) + m(L)/2 = 0.64 + 0.08/2 = 0.68 \quad (4.34)$$

CHAPTER V

EVALUATION OF FEATURE SELECTION ALGORITHM

In order to test the effectiveness of our feature selection method and compare it with other methods, we test our method in a various sizes of data sets. In this chapter, the data sets involved during the experiment are first introduced. We then demonstrate the experimental methodology, followed by a discussion of the experiment results.

5.1 DARPA KDD99 Intrusion Detection Evaluation Data Set

In the beginning of the research of intrusion detection, the most preliminary step is to prepare a data set that is good for developing an intrusion detection algorithm and for future test. Some people built their own simulated network environment and collected data by using sniff software such as *tcpdump* [73]. Although they included attacks purposely, it is often difficult to build a large network with hundreds of computer to mimic the real network scenario. Also the results of the specific simulated network environment could not be compared with those from different networks. On the other hand, people can use an existing data set to design and test their intrusion detection systems and the most popular one is *DARPA KDD99 Intrusion Detection Evaluation data set* [74]. The data set, also known as "*DARPA Intrusion Detection Evaluation data set*", has been chosen for analyzing the performance of our proposed classifier. It was created by Lincoln Laboratory at MIT [17] and was used in *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 with the main objective of intrusion detection and report [74].

It is a tailor-designed data set for the research of intrusion detection and includes a wide variety of intrusions from a simulated network environment. Although some people criticize that the data set deliberates mix different types of legitimate traffic (different ports and client platforms) with attacks, it is still the most realistic and publicly available one with a full list of actual attacks [75]. Consequently, people have been using it for designing and evaluating their intrusion detection systems. Also, the best is they can compare their experimental results with those from others.

For acquiring the evaluation data set, Lincoln Labs built a Local Area Network (LAN) to simulate a typical U.S. Air Force LAN. The LAN was operated as if it were a true Air Force environment, but peppered it with multiple attacks. The victim machines subjected to these attacks ran Linux, SunOS™, and Solaris™ operating systems. The data set was acquired from raw *tcpdump* data for a length of nine weeks. It is made up of a large number of network traffic activities that include both normal and malicious connections. In the *KDD99* data set, three independent sets are included, they are “whole KDD”, “10% KDD”, and “corrected KDD”. In our experiment, “10% KDD” and “corrected KDD” are taken as our training and testing set, respectively. The training set contains a total of 22 training attack types, with an additional 17 types in the testing set. Totally 39 attack types are included and they fall into four main classes, *DoS*, *Probe*, *U2R*, and *R2L*. Table 5.1 summarizes the connection distributions of training and testing sets we have used during the entire work. Table 5.2 lists the attacks and note that the 22 types of attacks in training set are marked underline. Figure 5.1 shows the detailed distributions of *DoS, Probe*, *U2R*,

Table 5.1. Connection Distributions

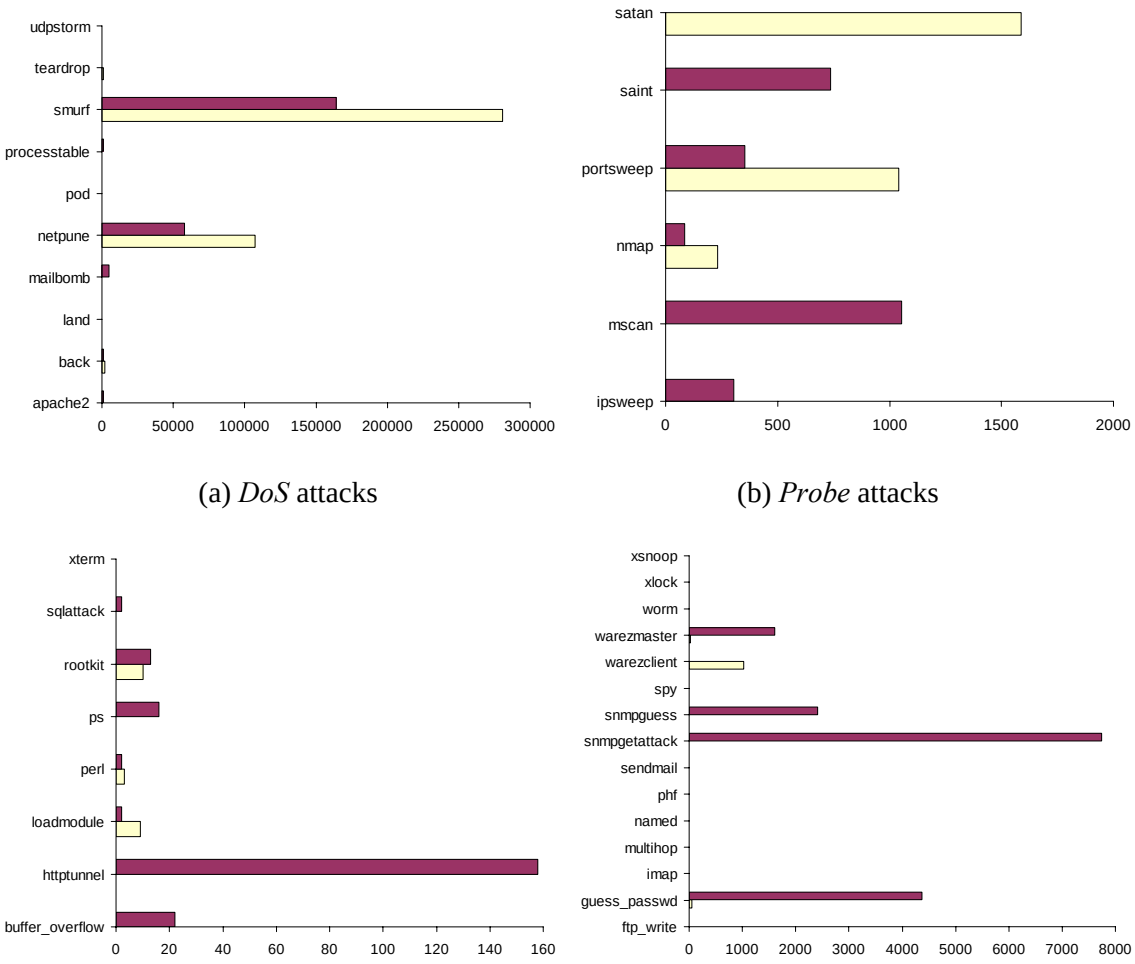
Data Set	Normal	DoS	R2L	U2R	Probe	Total
Training Set	97,277	391,458	1,126	52	4,107	494,020
Testing Set	60,593	229,853	16,189	228	4,166	311,029

Table 5.2. Thirty-Nine Attacks

DoS	R2L	U2R	Probe
apache2, back, land, mailbomb, netpune, pod, processtable, smurf, teardrop, udpstorm.	ftp_write, guess_passwd, imap, multihop, named, phf, sendmail, snmpgetattack, snmpguess, spy, warezclient, warezmaster, worm, xlock, xsnoop.	buffer_overflow, httptunnel, loadmodule, perl, ps, rootkit, sqlattack, xterm.	ipsweep, mscan, nmap, portsweep, saint, satan.

and *R2L* attacks. The signatures in *DoS* and *Probe* attacks in the testing set are very similar to those present in the provided training set. However, the types of attack of *U2R* and *R2L* attacks differ significantly between the training and the testing sets. In the testing set, over 80% *U2R* attacks and 60% *R2L* attacks are new to the training set.

Each connection is a sequence of TCP packets starting and ending at some well defined times. The set describes each connection in terms of 41 features plus a label of either normal or a type of attack. The content of these features are continuous, discrete, or symbolic with vary scales and ranges. These features can be classified into four classes, *basic*, *content*, *time-based*, and *host-based* features. Table 5.3 shows the detailed information of these 41 features. Features 1 to 9 are *basic features* that are derived from packet header without inspecting the payload. Features 10 to 22 are *content features* that are obtained by analyzing the payload of the original TCP packets. Features 23 to 31 are *time-based traffic features* that capture properties of connections in the past 2 seconds.



(a) *DoS* attacks (b) *Probe* attacks

(c) *U2R* attacks (d) *R2L* attacks

Figure 5.1. Distributions of Four *KDD99* Attack Categories
: Training Set : Testing Set

Features 32 to 41 are *host-based traffic features* that examine a number of connections using a window of 100 connections instead of a 2-second time window.

5.2 UCI Data Sets

In our experiment, six different sizes of data sets are chosen from the UCI Machine Learning Repository [76]. In the following we briefly describe the six data sets.

Table 5.3. Forty-One Features

No.	Feature	Description	Type
1	duration	length (no. of seconds) of the connection	continuous
2	protocol_type	type of the protocol	discrete
3	service	network service on the destination	discrete
4	flag	status flag of the connection	discrete
5	src_bytes	no. of data bytes from source to destination	continuous
6	dst_bytes	no. of data bytes from destination to source	continuous
7	land	1 if connection is from/to the same host/port; 0 otherwise	discrete
8	wrong_fragment	no. of wrong fragments	continuous
9	urgent	no. of urgent packets	continuous
10	hot	no. of “hot” indicators	continuous
11	num_failed_logins	no. of failed logins	continuous
12	logged_in	1 if successfully logged in; 0 otherwise	discrete
13	num_compromised	no. of “compromised” conditions	continuous
14	root_shell	1 if root shell is obtained; 0 otherwise	continuous
15	su_attempted	1 if “su root” command attempted; 0 otherwise	continuous
16	num_root	no. of “root” accesses	continuous
17	num_file_creations	no. of file creation operations	continuous
18	num_shells	no. of shell prompts	continuous
19	num_access_files	no. of operations on access control files	continuous
20	num_outbound_cmds	no. of outbound commands in an ftp session	continuous
21	is_host_login	1 if the login belongs to the “hot” list; 0 otherwise	discrete
22	is_guest_login	1 if the login is a “guest” login; 0 otherwise	discrete
23	count	no. of connections to the same host as the current connection in the past two seconds	continuous
24	srv_count	no. of connections to the same service as the current connection in the past two seconds	continuous
25	serror_rate	% of connections that have “SYN” errors	continuous
26	srv_serror_rate	% of connections that have “SYN” errors	continuous
27	rerror_rate	% of connections that have “REJ” errors	continuous
28	srv_rerror_rate	% of connections that have “REJ” errors	continuous
29	same_srv_rate	% of connections to the same service	continuous
30	diff_srv_rate	% of connections to different services	continuous
31	srv_diff_host_rate	% of connections to different hosts	continuous
32	dst_host_count	count of connections having the same destination host	continuous
33	dst_host_srv_count	count of connections having the same destination host and using the same service	continuous
34	dst_host_same_srv_rate	% of connections having the same destination host and using the same service	continuous
35	dst_host_diff_srv_rate	% of different services on the current host	continuous
36	dst_host_same_src_port_rate	% of connections to the current host having the same src port	continuous
37	dst_host_srv_diff_host_rate	% of connections to the same service coming from different hosts	continuous
38	dst_host_serror_rate	% of connections to the current host that have an S0 error	continuous
39	dst_host_srv_serror_rate	% of connections to the current host and specified service that have an S0 error	continuous
40	dst_host_rerror_rate	% of connections to the current host that have an RST error	continuous
41	dst_host_srv_rerror_rate	% of connections to the current host and specified service that have an RST error	continuous

Abalone: This data set is used for predicting the age of abalone from physical measurements. It includes totally 4,177 data records and each of them consists of eight features and one class label. Features include length (continuous), diameter (continuous), height (continuous), whole weight (continuous), shucked weight (continuous), viscera weight (continuous), shell weight (continuous), rings (integer). Classes are M (man), F (female), and I (infant).

Cmc: The data set is provided to study the problem of predicting the current contraceptive method choice (no use, long-term methods, or short-term methods) of a woman based on her demographic and socio-economic characteristics. It contains 1,473 data records and each has nine features. Features are wife's age (numerical), wife's education (categorical), husband's education (categorical), number of children ever born (numerical), wife's religion (binary), wife's now working (binary), husband's occupation (categorical), standard-of-living index (categorical), and media exposure (binary). Classes are 1 (no-use), 2 (long-term), and 3 (short-term).

Ionosphere: This is radar data that were collected by a system in Goose Bay, Labrador. It has 351 data records and each one has thirty four continuous features plus a class label. It is used for binary classification tasks and therefore the class label is either good or bad.

Pima: This data set collects information from patients who are all females over 21-year old of Pima Indian heritage. It includes 768 data records and has eight features which are number of times pregnant, plasma glucose concentration a 2 hours in an oral glucose tolerance test, diastolic blood pressure, triceps skin fold thickness, 2-hour serum insulin, body mass index, diabetes pedigree function, and age. Classes are 0 and 1.

Wdbc: For the study of breast tumor diagnosis, this data set provides 569 data records and each one comprises an ID number plus thirty features that describe characteristics of the cell nuclei presented in the image. The class label is M (malignant) or B (benign).

Wine: This data set includes 178 data records that represent the chemical analysis results of wines grown in the same region in Italy but derived from three different cultivars. Each data record has thirteen features, alcohol, malic acid, ash, alcalinity of ash, magnesium, total phenols, flavanoids, nonflavanoid phenols, proanthocyanins, color intensity, hue, OD280/OD315 of diluted wines, and praline, plus a class label (1, 2, or 3).

5.3 Experimental Methodology

A. Discretization of Features

In the six *UCI* data sets and *KDD99* data set, each record is composed by a set of features. The type of features is either discrete or continuous which the former is a qualitative scale and the latter is quantitative. For qualitative scales, the values are simply labels without any order involved. They could be symbolic or numeric values where are distinct and separated. Also, it is a form of categorical data that has no "numeric" meaning. By using the features of *KDD99* data set as an example, the value of feature *protocol_type* is one of the symbolic set {icmp, tcp, udp}. The numeric value of feature *logged_in* is 1 or 0 to represent the user successfully logged in the system or not. For quantitative scales, the data are characterized by numeric values within a finite interval. The distance between any two adjacent values is not necessary the same. Examples can be found in feature *duration* where it is given by numeric values to represent the lengths of record, and the values are within an interval [0, 58,329].

Since *SU* is calculated for discrete features only, all the continuous features in a given data set are necessary to be discretized prior to the feature selection analysis. Thus, we apply discretization method to transform continuous features to discrete ones prior to the analysis. For a numeric feature, cut points effectively decompose the range of continuous values into a number of intervals. These intervals can then be treated as categorical values of a discrete feature. In our work, *equal frequency binning* technique [77] is applied to each continuous feature individually. It is an unsupervised discretization method with no class information involved. It sorts the observed values of a continuous feature and then divides these values into a specified number of intervals. Each of the intervals has an approximate equal number of values. With the use of discretization of features, the complexity of every continuous feature is reduced as well.

B. Experimental Methodology

In order to evaluate the performance of our proposed feature selection algorithm on data sets, two representative feature selection algorithms, CFS [77] and FCBF [63], built on the top of *SU* are chosen. CFS method uses a correlation-based heuristic search algorithm to evaluate the worth of subsets of features. It considers good feature subsets that are highly correlated with the class, yet uncorrelated with one another. The heuristic algorithm measures the merit of feature subsets from pairwise feature correlations and then the subset with the highest merit found during the search is reported. Rather than scoring the worth of subsets of features of CFS approach, FCBF method measures correlations between features and classes and correlations between pairs of features as well. It then selects features which are highly correlated with the class to predict but are less correlated to any feature already selected. In addition, we apply two machine

learning algorithms, naive bayes and C4.5 algorithm, to evaluate the detection accuracy on the selected features for each feature selection algorithm.

The experiments are performed on the six *UCI* data sets and binary classification (normal/attack) of *KDD99* data set. Four new sets of data are generated from *KDD99* data set according to the normal class and four categories of attack (*DoS*, *Probe*, *U2R* and *R2L*). In each data set, connections with the same attack category and all the normal ones are included. For each set, we run our proposed approach and the other two feature selection algorithms CFS and FCBF, and record those features selected by each algorithm. Throughout the entire experiments, the threshold of FCBF is set to 0. We then apply C4.5 and naive bayes machine learning algorithms on each original full data set as well as each newly obtained data set that includes only those selected features from feature selection algorithms. By applying 10-fold cross-validation evaluation on each data set, classification accuracy of six *UCI* data sets and standard measurements, such as the *detection rate* (*DR*), *false positive rate* (*FPR*), and *overall classification rate* (*CR*), for evaluating the performance of intrusion detection tasks are reported. The denotations of *True Positive* (*TP*), *True Negative* (*TN*), *False Positive* (*FP*), and *False Negative* (*FN*) are defined as follows.

- *True Positive* (*TP*): The number of malicious records that are correctly identified.
- *True Negative* (*TN*): The number of legitimate records that are correctly classified.
- *False Positive* (*FP*): The number of records that were incorrectly identified as attacks however in fact they are legitimate activities.
- *False Negative* (*FN*): The number of records that were incorrectly classified as legitimate activities however in fact they are malicious.

Equations 5.1 to 5.3 describe *DR*, *FPR*, and *CR*, respectively.

$$DR = \frac{TP}{TP + FN} \tag{5.1}$$

$$FPR = \frac{FP}{TN + FP} \tag{5.2}$$

$$CR = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.3}$$

5.4 Experimental Results

Based on our developed feature selection algorithm, Tables 5.4 and 5.5 summarize the selected features from our approach as well as those selected by CFS and FCBF algorithms of *UCI* and *KDD99* data sets, respectively. Table 5.6 summarizes the classification accuracy of the six *UCI* data sets. Tables 5.7 and 5.8 summarize the percentages of DRs and FPRs performed on *KDD99* data set with C4.5 and naive bayes learning algorithms, respectively. For an intrusion detection task, abnormal activities are expected to be correctly identified and normal activities are anticipated not to be misclassified. Therefore, a higher DR and a lower FPR are desired. In addition, we show the results of *SU* measures of feature to class and feature to feature of *KDD99* data sets in Tables 5.9 and 5.10, respectively.

From Table 5.6, our approach shows higher averaged accuracies in comparison with the outcomes of CFS and FCBF feature selection algorithms. Especially in the abalone data set, we get the highest classification accuracy by using 2 out of 8 features performed on C4.5 learning algorithm, which is better than that of using full feature set. The averaged accuracies of Tables 5.7 and 5.8 also show that our approach outperform over CFS and

Table 5.4. Selected Features of *UCI* Data Sets

Data Set	Ours	CFS	FCBF
Abalone	3,8	2,3,6,8	8
Cmc	1,4	2,4	2,4
Ionosphere	1,5,6,8,9,16,33,34	1,33	1,33
Pima	2,5,6,8	2,5,6	2
Wdbc	1,3,4,6-8,11,13,14,21,23,24,26-28	8,21,23,24,28	24
Wine	1,7,10,11-13	1,7,10-13	1,2,4,5,7,10-13

Table 5.5. Selected Features of *KDD99* Data Sets

Data Set	Ours	CFS	FCBF
Normal-DoS	1-6,12,23,24,31,32,37	3,6,12,37	3,12,31,32
Normal-Probe	1-4,12,16,25,27-30,40	3,4,25,29	3,26,27,29
Normal-U2R	1-3,10,16	10	10,16
Normal-R2L	1-5,10,22	10	5,10,39

FCBF feature selection algorithms. The averaged DRs and the averaged FPRs of our experimental results are better than those of using full feature set performed on C4.5 and naive bayes, respectively.

In the *Normal-DoS* data set, the difference in DRs is very slight for all of the feature selection algorithms. With our approach, the DR is the same as that of using full feature set in C4.5 learning algorithm. In the *Normal-U2R* and *Normal-R2L* data sets, we have satisfactory DRs and FPRs. Though CFS and FCBF approaches achieve low FPRs, they have very poor DRs. In the *Normal-Probe* data set, both CFS and FCBF approaches fail to achieve an acceptable presentation on DRs while using naive bayes leaning algorithm, whereas our approach gains very high DRs performed on both leaning algorithms.

For any of the feature selection algorithms, FPRs are low because sufficient normal records present in any of those four data sets. As for the number of misclassification

attack records, our approach provides acceptable DRs on *Normal-DoS*, *Normal-Probe* and *Normal-R2L* data sets using both C4.5 and naive bayes learning algorithms. It is not only because each of the above data set supplies sufficient attack records but also most of the attacks have a same attack signature. For example, the *DoS* attack type includes near 400,000 data records distributed in 10 different attacks, which 99% of the attacks are *netpune* and *smurf* attacks. In the *Probe* attack category, 95% of attacks are *ipsweep*, *portsweep* and *satan* that are distributed in 4,107 attacks. As for *R2L* attack class, more than 90% of attacks are *warezclient* attack while 8 different kinds of attacks present. In contrast, the classification presented on *Normal-U2R* data set is satisfactory neither on full feature set approach nor on one of three feature selection algorithms. The *Normal-U2R* data set includes 52 attack records which are insufficient for learning on a classification algorithm. The experimental results have been published in [78].

Table 5.6. CRs of C4.5 and Naive Bayes Using Full and Selected Feature Sets of *UCI* Data Sets

Data Set	C4.5				Naive Bayes			
	Full Set	Ours	CFS	FCBF	Full Set	Ours	CFS	FCBF
Abalone	51.90	56.00	51.90	51.90	63.23	53.60	51.90	51.90
Cmc	63.68	54.65	52.89	52.89	53.36	52.61	52.27	52.27
Ionosphere	74.93	74.93	74.93	74.93	99.15	97.72	94.02	94.02
Pima	65.10	65.10	65.10	65.10	89.97	87.50	85.03	77.34
Wdbc	62.74	62.74	62.74	62.74	99.30	99.30	99.65	94.02
Wine	94.94	94.94	94.94	94.94	98.88	97.75	97.75	98.88
Average	68.88	68.06	67.08	67.08	83.98	81.41	80.10	78.07

Table 5.7. DRs of C4.5 and Naive Bayes Using Full and Selected Feature Sets of *KDD99* Data Sets

Data Set	C4.5				Naive Bayes			
	Full Set	Ours	CFS	FCBF	Full Set	Ours	CFS	FCBF
Normal-DoS	99.97	99.97	99.86	99.31	99.12	99.16	99.37	99.19
Normal-Probe	98.51	97.78	95.52	94.91	98.27	96.54	62.53	45.31
Normal-U2R	48.08	48.08	0	7.69	82.69	69.23	0	7.69
Normal-R2L	93.52	97.69	0	27.44	99.11	93.25	0	33.84
Average	85.02	85.88	48.85	57.34	94.80	89.55	40.48	46.51

Table 5.8. FPRs of C4.5 and Naive Bayes Using Full and Selected Feature Sets of *KDD99* Data Sets

Data Set	C4.5				Naive Bayes			
	Full Set	Ours	CFS	FCBF	Full Set	Ours	CFS	FCBF
Normal-DoS	0.04	0.03	2.19	7.58	0.01	0.01	2.76	7.77
Normal-Probe	0.02	0.38	0.36	0.36	1.29	0.87	0.15	0.10
Normal-U2R	0	0	0	0	0.63	0.50	0	0
Normal-R2L	0.01	0.01	0	0.02	1.31	0.49	0	0.08
Average	0.02	0.11	0.64	1.99	0.81	0.47	0.73	1.99

Table 5.9. *SU* Measure of Feature (F) to Class (C) of *KDD99* Data Set

Normal-DoS		Normal-Probe		Normal-U2R		Normal-R2L	
Feature	SU(F; C)	Feature	SU(F; C)	Feature	SU(F; C)	Feature	SU(F; C)
12	0.5939	29	0.2427	10	0.0552	10	0.2000
3	0.4638	27	0.2263	16	0.0131	22	0.1919
6	0.4578	25	0.2243	1	0.0037	3	0.0484
37	0.3639	4	0.2223	3	0.0033	39	0.0184
5	0.3423	30	0.1941	33	0.0011	38	0.0171
32	0.3352	28	0.1460	25	0.0010	33	0.0161
2	0.3284	38	0.1374	41	0.0009	4	0.0154
36	0.3126	40	0.1365	29	0.0008	5	0.0143
23	0.2698	41	0.1248	40	0.0007	1	0.0129
31	0.2418	12	0.1232	30	0.0006	40	0.0125
24	0.1699	3	0.1071	36	0.0006	37	0.0114
35	0.1344	35	0.0875	32	0.0005	36	0.0111
1	0.1211	2	0.0695	4	0.0005	6	0.0100
34	0.1158	37	0.0549	5	0.0005	2	0.0086
33	0.1104	26	0.0493	37	0.0004	23	0.0063
39	0.1100	34	0.0466	6	0.0004	24	0.0063
38	0.1028	23	0.0417	27	0.0004	35	0.0061
26	0.0898	5	0.0408	24	0.0003	32	0.0050
25	0.0893	33	0.0389	35	0.0003	12	0.0045
30	0.0891	39	0.0350	31	0.0003	31	0.0045
4	0.0743	36	0.0336	23	0.0002	41	0.0038
29	0.0670	6	0.0304	34	0.0002	34	0.0038
41	0.0397	32	0.0173	39	0.0002	30	0.0017
40	0.0119	31	0.0158	38	0.0002	29	0.0016
13	0.0022	24	0.0124	2	0.0002	26	0.0016
28	0.0014	1	0.0065	26	0.0002	25	0.0007
10	0	16	0.0023	12	0.0001	28	0.0004
27	0	7	0	28	0	27	0.0001
7	0	8	0	7	0	7	0
8	0	9	0	8	0	8	0
9	0	10	0	9	0	9	0
11	0	11	0	11	0	11	0
14	0	13	0	13	0	13	0
15	0	14	0	14	0	14	0
16	0	15	0	15	0	15	0
17	0	17	0	17	0	16	0
18	0	18	0	18	0	17	0
19	0	19	0	19	0	18	0
20	0	20	0	20	0	19	0
21	0	21	0	21	0	20	0
22	0	22	0	22	0	21	0

Table 5.10. *SU* Measure of Feature to Feature of *KDD99* Data Set

	1	2	3	4	5	6	10	12	13	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41
1	1	0.087	0.116	0.017	0.073	0.132	0.031	0.059	0.002	0.037	0.038	0.015	0.015	0.003	0.003	0.012	0.02	0.06	0.037	0.067	0.079	0.101	0.061	0.039	0.018	0.019	0.008	0.005
2	0	1	0.705	0.38	0.55	0.267	0.013	0.264	0.01	0.385	0.413	0.312	0.312	0.107	0.113	0.27	0.321	0.087	0.129	0.316	0.341	0.411	0.411	0.621	0.144	0.317	0.322	0.123	0.124
3	0	0	1	0.453	0.648	0.416	0.023	0.431	0.014	0.447	0.398	0.369	0.364	0.08	0.084	0.382	0.437	0.175	0.246	0.458	0.501	0.54	0.614	0.26	0.373	0.375	0.109	0.116
4	0	0	0	1	0.402	0.058	0.005	0.077	0.004	0.273	0.189	0.786	0.802	0.449	0.473	0.525	0.639	0.03	0.04	0.368	0.427	0.49	0.365	0.061	0.751	0.761	0.426	0.432
5	0	0	0	0	1	0.359	0.027	0.312	0.026	0.419	0.377	0.275	0.273	0.077	0.085	0.307	0.338	0.115	0.153	0.366	0.384	0.411	0.459	0.164	0.279	0.28	0.103	0.102
6	0	0	0	0	0	1	0.049	0.484	0.046	0.201	0.133	0.047	0.047	0.015	0.025	0.045	0.053	0.16	0.174	0.117	0.133	0.141	0.215	0.207	0.058	0.065	0.054	0.057
10	0	0	0	0	0	0	1	0.05	0.738	0.009	0.008	0.004	0.005	0.002	0.035	0.002	0.003	0.006	0.005	0.004	0.004	0.004	0.008	0.001	0.011	0.009	0.099	0.1
12	0	0	0	0	0	0	0	1	0.039	0.187	0.109	0.067	0.07	0.027	0.035	0.05	0.062	0.218	0.285	0.055	0.065	0.05	0.238	0.36	0.087	0.104	0.068	0.075
13	0	0	0	0	0	0	0	0	1	0.008	0.006	0.003	0.004	0.002	0.038	0.002	0.003	0.007	0.002	0.002	0.002	0.002	0.006	0.001	0.009	0.009	0.107	0.115
23	0	0	0	0	0	0	0	0	0	1	0.551	0.203	0.202	0.044	0.042	0.296	0.299	0.105	0.139	0.282	0.287	0.3	0.366	0.118	0.206	0.208	0.056	0.056
24	0	0	0	0	0	0	0	0	0	0	1	0.148	0.145	0.041	0.045	0.278	0.196	0.129	0.087	0.44	0.386	0.237	0.352	0.076	0.153	0.152	0.055	0.053
25	0	0	0	0	0	0	0	0	0	0	0	1	0.959	0.039	0.042	0.432	0.54	0.029	0.036	0.296	0.351	0.407	0.312	0.039	0.91	0.899	0.063	0.043
26	0	0	0	0	0	0	0	0	0	0	0	0	1	0.032	0.034	0.423	0.532	0.031	0.037	0.292	0.347	0.399	0.312	0.041	0.908	0.929	0.033	0.036
27	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0.906	0.061	0.095	0.01	0.013	0.051	0.054	0.069	0.059	0.047	0.033	0.03	0.766	0.754
28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0.064	0.099	0.025	0.013	0.054	0.057	0.075	0.063	0.045	0.038	0.033	0.777	0.801
29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0.564	0.022	0.032	0.432	0.483	0.427	0.303	0.032	0.417	0.41	0.074	0.075
30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0.027	0.038	0.383	0.447	0.526	0.349	0.039	0.517	0.51	0.11	0.11
31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0.113	0.036	0.039	0.034	0.09	0.132	0.033	0.037	0.018	0.023
32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0.067	0.066	0.076	0.449	0.457	0.059	0.057	0.038	0.047
33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0.794	0.481	0.31	0.071	0.295	0.292	0.064	0.065
34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0.565	0.362	0.053	0.348	0.342	0.067	0.067
35	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0.388	0.038	0.403	0.39	0.092	0.085
36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0.252	0.323	0.319	0.085	0.087
37	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0.05	0.063	0.056	0.085
38	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0.89	0.075	0.045
39	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0.039	0.046
40	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0.798
41	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

CHAPTER VI

EVALUATION OF FUZZY BELIEF INTRUSION DETECTION

In this chapter, we first test the feasibility of proposed fuzzy belief *k*-NN classifier in intrusion detection task, and compare its result with those of three other *k*-NN based classifiers: *k*-NN classifier [18], fuzzy *k*-NN classifier [79], and evidence-theoretic *k*-NN classifier [80]. Then we apply the experimental results of Chapter V to all four classifiers for observing the dissimilarities between applying full feature set and selected feature subsets.

6.1 Experimental Methodology

A. Data Preprocessing

Duplicated connections were removed from the original training and testing data sets. The new training set has 145,585 connections that are distributed as 87,831 normal connections, 54,572 *DoS* attacks, 2,131 *Probe* attacks, 52 *U2R* attacks, and 999 *R2L* attacks. The new testing set has 51,041 connections that are distributed as 47,913 normal connections, 23,568 *DoS* attacks, 2,682 *Probe* attacks, 215 *U2R* attacks, and 2,913 *R2L* attacks. In each connection, features represented by symbolic values and class labels are replaced by numeric values for the use of classifiers. For example, the values of icmp, tcp, and udp of feature *protocol_type* are replaced by values 1, 2, and 3, respectively. Class labels for normal connections, *U2R* attacks and *R2L* attacks are substituted by

values 1, 2, and 3, respectively. In addition, values of each feature are normalized between 0 and 1 in order to offer equal importance among features.

B. Experimental Methodology

In order to evaluate the detection performance of the proposed fuzzy belief *k*-NN classifier, three pattern classification algorithms are selected to compare with. One is *k*-NN classifier and the other two are fuzzy *k*-NN classifier and evidence-theoretic *k*-NN classifier that built on the base of *k*-NN rule.

The *k*-NN classifier is simple but effective in many pattern classification applications. For an input to be classified, *k* nearest training patterns are obtained based on the Euclidean distance measurement between the input and every training pattern. The input is then simply assigned to the class by majority voting, i.e., the input is classified to the most frequent class label among the *k* nearest training patterns. However, a major drawback of *k*-NN algorithm is that the precision of classification may decrease if all selected *k* nearest training patterns are equally important without considering the differences of distances [80]. To eliminate this drawback, fuzzy *k*-NN classifier assigns multiple membership grades to classes rather than a single class. By using the distance differences from the *k* nearest training patterns, the degrees of membership grades to classes are determined. As the evidence-theoretic *k*-NN classifier, it incorporates Dempster-Shafer theory to treat the *k* nearest training patterns of an input pattern as pieces of evidence to support certain hypotheses about the classes. By deriving evidences from both class labels and distances between input and *k* nearest training pattern pairs, these evidences are then combined into final beliefs with respect to each subset of the set of classes.

6.2 Experimental Results

Generally, a large amount of traffic records are essential for a classifier to be trained when using anomaly detection and the consequence is that a long computational time is required to reach a final decision. But unfortunately, the intrusion detection system has to perform its analysis as quick as possible, otherwise serious damages could happen and possibly cause millions of dollars loss. Therefore, only a small amount of connections are included in our experiments for training the classifiers. It not only speeds up the classification process but also simulates the uncertainty caused by lack of network traffic information.

The experiments are performed on the binary (normal/attack) classification. To minimize the inaccuracy and variation factor of experiment results, 10 trials are performed in every *U2R* and *R2L* detection task. In each trial, certain percentages of normal and attack connections are randomly selected from the training and testing sets. For detecting *U2R* attacks, the training and testing sets comprise 930 (878 normal and 52 *U2R*) and 694 (479 normal and 215 *U2R*) connections, respectively. For detecting *R2L* attacks, the training and testing sets include 977 (878 normal and 99 *R2L*) and 770 (479 normal and 291 *R2L*) connections, respectively.

To detect the attacks, training and testing are performed in each trial. In the training phase, the four classifier, *k*-NN, fuzzy *k*-NN, evidence-theoretic *k*-NN, and fuzzy belief *k*-NN, are constructed. The testing data are then fed into the trained classifiers to identify intrusions in the testing phase.

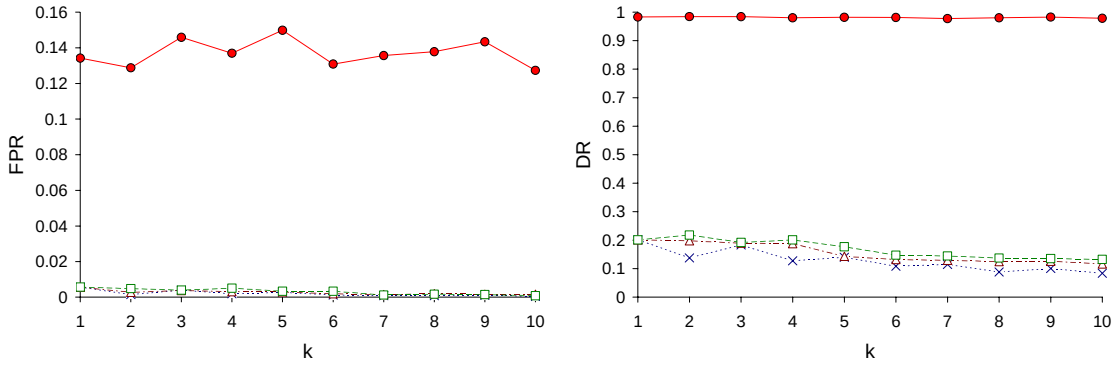
We evaluate the performances of the four classifiers using distinct values of *k* that ranges from 1 to 10. Tables 6.1 and 6.2 summarize the maximum, minimum, and averaged rates of *U2R* and *R2L* attacks, respectively. Figures 6.1 and 6.2 show FPRs, DRs, CRs, and Receiver Operating Characteristics (ROC) graphs of *U2R* and *R2L* attacks, respectively. A ROC graph is a plot with FPR on the *X* axis and DR on the *Y* axis. Since these four classifiers are discrete classifiers, each of them produces a single point representing the pair of FPR and DR in the ROC space.

The results show that DRs of our classifier are much higher than those of other three classifiers in detecting both *U2R* and *R2L* attacks. With our proposed classifier, a high averaged DR of 98.16% is achieved when detecting *U2R* attacks. By using only one nearest training connections for each testing connection, 98.33% DR has been reached. On the contrary, the other three classifiers can only reach around 15% DRs no matter fewer or more nearest training connections are applied. While detecting *R2L* attacks, the averaged DR of 67.88% has been achieved. However the other three classifiers provide only around 20% DRs. For the other three *k*-NN based classifiers, evidence-theoretic *k*-NN classifier has a slightly better performance over *k*-NN and fuzzy *k*-NN classifiers. It indicates that Dempster-Shafer theory of evidence-theoretic *k*-NN classifier provides a degree of influence while detecting attacks in network traffic.

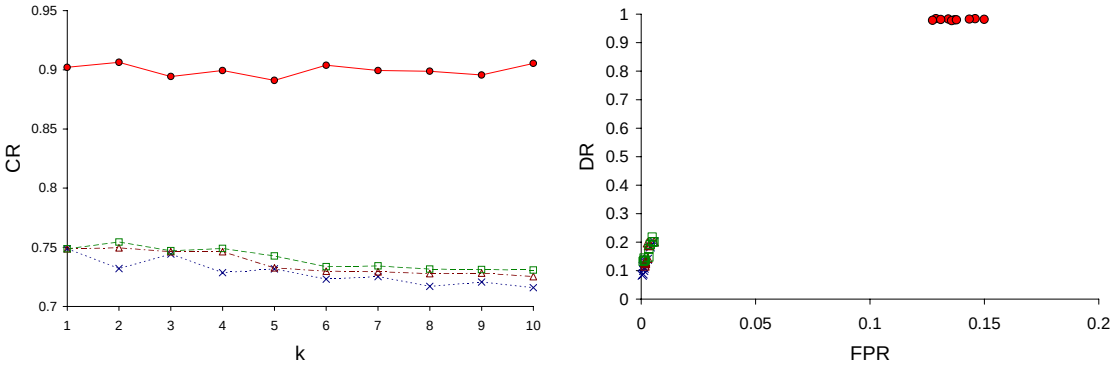
For identifying the normal connections in our classifier, the averaged FPRs are 13.71% and 11.58% for *U2R* and *R2L* attacks, respectively. Three classifiers have very low FPRs because they treat most network traffic data as normal connections no matter they are normal or malicious activities. The above observation can be explained from the ROC graphs.

Table 6.1. FPRs and DRs Performed on Four Classifiers of *Normal-U2R* Data Sets with *k* Ranging From 1 to 10

	FPR			DR		
	Max	Min	Avg.	Max	Min	Avg.
k-NN	0.56	0.04	0.20	20.14	8.37	12.87
Fuzzy *k*-NN	0.56	0.15	0.27	20.14	11.67	15.49
Evidence-Theoretic *k*-NN	0.56	0.08	0.31	21.81	13.26	16.87
Fuzzy Belief *k*-NN	14.99	12.73	13.71	98.47	97.77	98.16



(a) False Positive Rates (b) Detection Rates

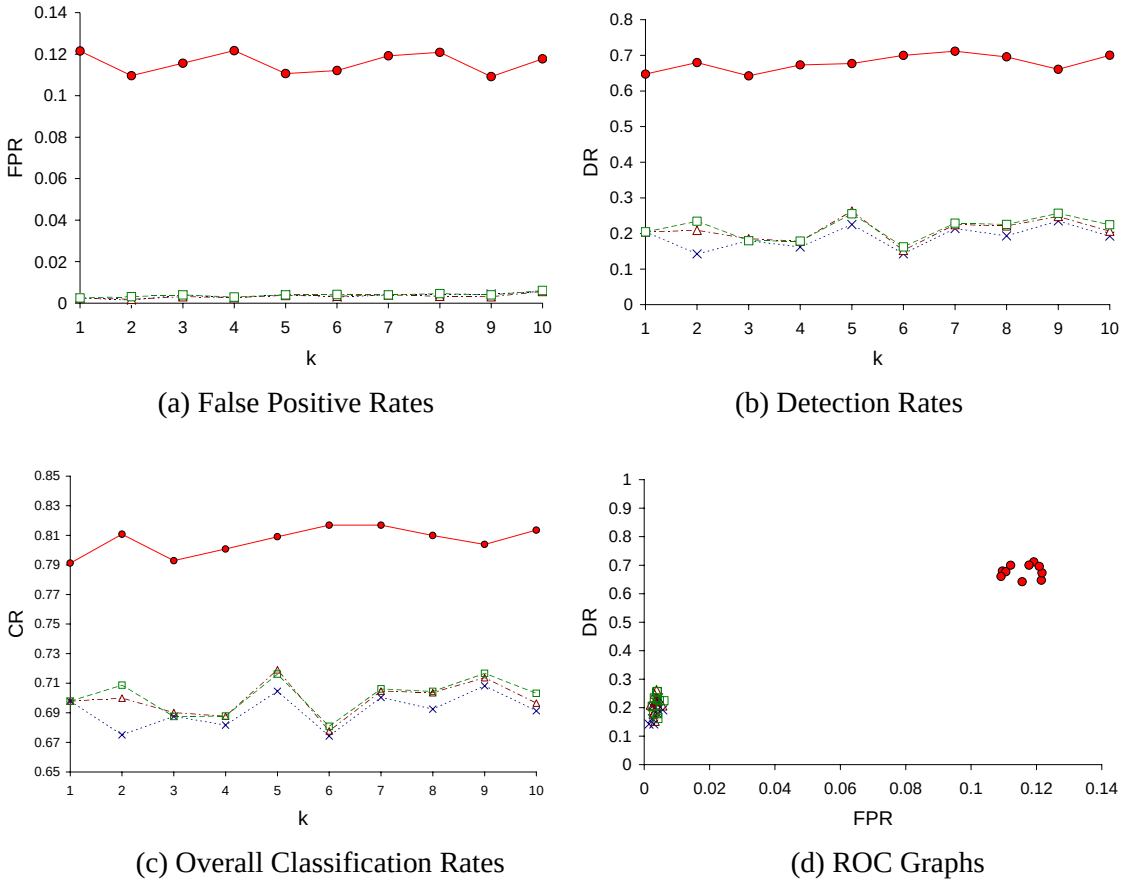


(c) Overall Classification Rates (d) ROC Graphs

Figure 6.1. ROC Graphs of Four Classifiers Performed on *Normal-U2R* Data Set
x: *k*-NN, Δ: Fuzzy *k*-NN, □: Evidence-Theoretic *k*-NN, •: Fuzzy Belief *k*-NN

Table 6.2. FPRs and DRs Performed on Four Classifiers of *Normal-R2L* Data Sets with *k* Ranging From 1 to 10

	FPR			DR		
	Max	Min	Avg.	Max	Min	Avg.
k-NN	0.56	0.15	0.36	23.51	14.26	18.91
Fuzzy *k*-NN	0.56	0.19	0.33	26.29	15.26	20.92
Evidence-Theoretic *k*-NN	0.61	0.25	0.39	25.67	16.22	21.50
Fuzzy Belief *k*-NN	12.17	10.92	11.58	71.17	64.23	67.88



(a) False Positive Rates

(b) Detection Rates

(c) Overall Classification Rates

(d) ROC Graphs

Figure 6.2. ROC Graphs of Four Classifiers Performed on *Normal-R2L* Data Set x: *k*-NN, Δ: Fuzzy *k*-NN, □: Evidence-Theoretic *k*-NN, ●: Fuzzy Belief *k*-NN

In a ROC graph, the point (0, 1) represents the classifier performs a perfect classification; it classifies all positive cases and negative cases correctly. On the contrary, the point (1, 0) represents the classifier in the worst case, i.e., it classifies all cases incorrectly. The lower left point (0, 0) represents the classifier never reports any false positive errors, while the upper right point (1, 1) has an opposite policy that the classifier predicts all cases to be positive. From both ROC graphs of *U2R* and *R2L* attacks, we can see points of *k*-NN, fuzzy *k*-NN, and evidence-theoretic *k*-NN classifiers are all gathering near point (0, 0), which indicate all of them seldom report false positive errors and correctly predict a few of attacks. However, all the points of fuzzy believe *k*-NN classifier are close to point (0, 1), which have higher DRs and have lower FPRs as well.

In summary, our classifier has a better performance compared with the other three classifiers. By including only a small portion of connections from the training set of *KDD99*, we achieve high DRs of identification of both *U2R* and *R2L* attacks.

Once we finish the feasibility test of fuzzy belief *k*-NN classifier, the results obtained in Chapter V can thus be provided to classifiers to identify traffic data in a shorter period of time. Tables 6.3 and 6.4 summarize the averaged FPRs and the averaged DRs performed on four classifiers with *k* ranging from 1 to 10 of *Normal-U2R* and *Normal-R2L* data sets, respectively.

In the comparison of four classifiers performed in different feature sets, *k*-NN, fuzzy *k*-NN, and evidence-theoretic *k*-NN classifiers have similar detection performances using either full feature set or one of selected feature subsets, which all the three *k*-NN based classifiers have poor detection performances. The maximum DRs in rows 1 to 3 of Tables 6.3 and 6.4 are 19.64% and 26.33% for *Normal-U2R* and *Normal-R2L* data sets,

Table 6.3. Averaged Rates Performed on Four Classifiers of *Normal-U2R* Data Set with k Ranging From 1 to 10

Classifier	Full Set		Ours		CFS		FCBF	
	FPR	DR	FPR	DR	FPR	DR	FPR	DR
k-NN	0.20	12.87	2.55	18.03	0.20	15.07	0.21	15.21
Fuzzy k-NN	0.27	15.49	2.50	18.47	0.19	14.52	0.20	15.27
Evidence-Theoretic k-NN	0.31	16.87	2.65	19.64	0.23	16.99	0.26	18.48
Fuzzy Belief k-NN	13.71	98.16	9.54	83.86	0.25	11.72	0.16	7.32

Table 6.4. Averaged Rates Performed on Four Classifiers of *Normal-R2L* Data Set with k Ranging From 1 to 10

Classifier	Full Set		Ours		CFS		FCBF	
	FPR	DR	FPR	DR	FPR	DR	FPR	DR
k-NN	0.36	18.91	3.68	20.41	0.26	14.42	3.05	19.31
Fuzzy k-NN	0.33	20.92	19.30	23.58	0.26	15.30	3.05	19.68
Evidence-Theoretic k-NN	0.39	21.50	8.76	26.33	0.27	15.84	4.50	26.23
Fuzzy Belief k-NN	11.58	67.88	9.86	69.82	0.18	7.78	0.19	7.98

respectively. With our proposed fuzzy belief k-NN classifier, the results of using three feature selection algorithms differ a lot, which our selected features provide much accurate DRs than those from CFS and FCBF. In the last row, the DRs of our approach reach 83.86% and 69.82% for *Normal-U2R* and *Normal-R2L* data sets, respectively. Both CFS and FCBF achieve low FPRs in the data sets because they treat most of the network traffic data as normal usages no matter the traffic are normal or malicious activities. For a better demonstration, Figure 6.3 shows the ROC graphs of four classifiers performed on *Normal-U2R* and *Normal-R2L* data sets using our selected feature subset with k ranging from 1 to 10. It shows the points of k-NN, fuzzy k-NN, and evidence-theoretic k-NN classifiers are all gathering near point (0, 0), which indicates that none of them can correctly identify attacks. However, all the points of fuzzy believe k-NN classifier are much closer to point (0, 1), which have higher DRs and have lower FPRs as well.

In Figure 6.4, we show the result of fuzzy belief *k*-NN classifier using full feature set and three feature subsets selected by our developed feature selection algorithm, CFS, and FCBF. For both data sets using features from CFS and FCBF, the diagrams show that all of the points are in the vicinity of (0, 0), which represents all the traffic are classified as normal activities and only a very few amount of attacks are correctly detected. In the left diagram, the DR with full feature set is higher than that of using our feature subset, however our selected features provide a better FPR result than that of using full feature set. In the right diagram, we notice that the points with our selected features are closer to point (0, 1) than those of using full feature set, which show that our selected features achieve better detection outcomes in both DR and FPR than those of using full feature set. In addition to the consideration of detection performance, we furthermore consider the detection processing time because an intrusion detection system has to perform its analysis as fast as possible before the attacks make any damage to the protected system. Consequently, we compare the computation time of fuzzy belief *k*-NN classifier using full feature set and our selected feature subset. Figure 6.5 illustrates the detection time on each testing connection of both data sets. The results show that we successfully reduce the computation time if our selected feature subset is used. Our approach only take about 25% of the time with full feature set in *Normal-U2R* and *Normal-R2L* data sets. The related works have been published in [81]-[84].

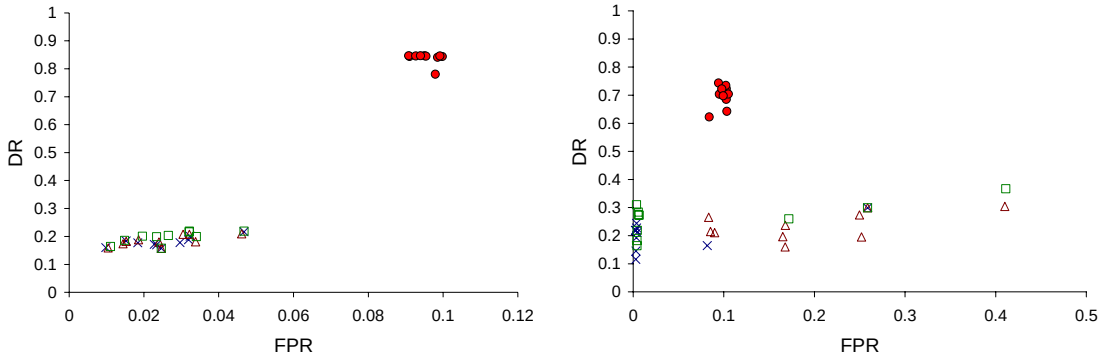


Figure 6.3. The Performance of Four Classifiers Using *Normal-U2R* (left) and *Normal-R2L* (right) Data Sets
x: *k*-NN, Δ: Fuzzy *k*-NN, □: Evidence-Theoretic *k*-NN, •: Fuzzy Belief *k*-NN

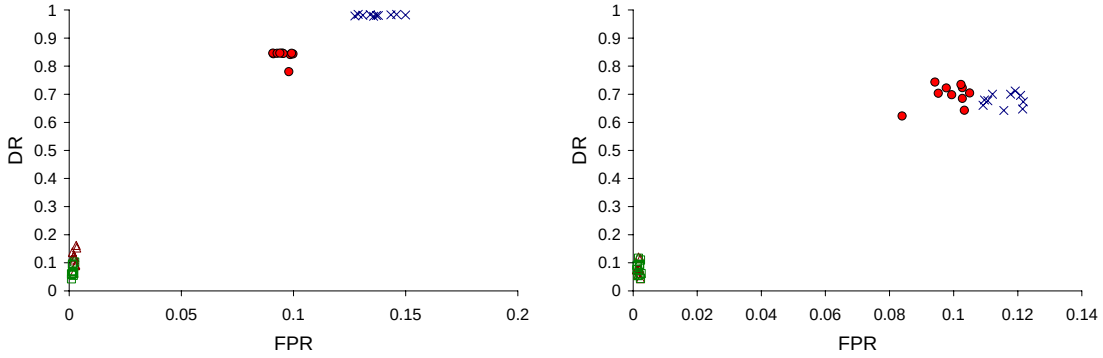


Figure 6.4. The Performance of Fuzzy Belief *k*-NN Classifier Using *Normal-U2R* (left) and *Normal-R2L* (right) Data Sets
x: Full Feature Set, Δ: CFS, □: FCBF, •: Ours

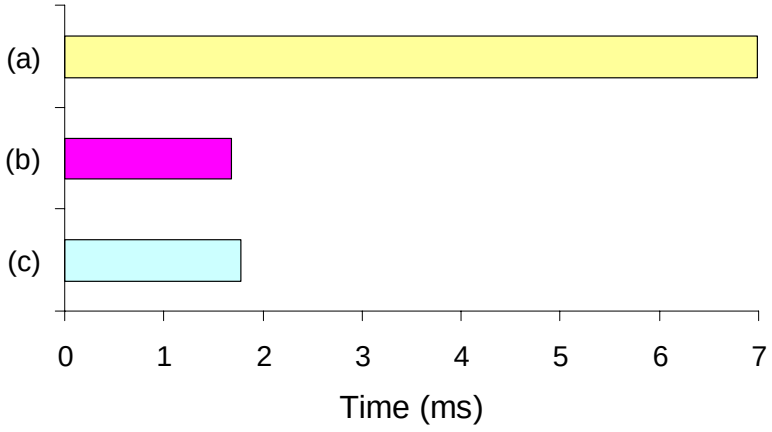


Figure 6.5. Detection Time of One Connection Using Fuzzy Belief *k*-NN Classifier
(a) With all 41 features
(b) With 5 selected features in *Normal-U2R* set
(c) With 7 selected features in *Normal-R2L* set

CHAPTER VII

ENSEMBLE INTRUSION DETECTION

An ensemble of classifiers is a set of base classifiers, whose classification decisions are combined together in some way to achieve a better performance than that of individual base classifier. In the intrusion detection task, we apply different feature subsets to base classifiers. We call it as *ensemble feature selection.* Also, we apply data mining technique to promote the FPR.

7.1 Ensemble Intrusion Detection Model

While designing an intrusion detection system, detection accuracy and speed are two important considerations. The system needs to perform a proper detection task with low FPR on normal computer usages and high DR on malicious activities. Also, this system has to perform its analysis as soon as possible before the attacks make any damage to the protected system. In the past, approaches to intrusion detection based on ensemble techniques have been investigated with the use of different feature subsets [60] or soft computing techniques [61] in every individual classifier. However, they only focused on improving DR in known and unknown intrusions but did not consider reducing the number of false alarms. Therefore, we propose an ensemble model that includes an ensemble feature selecting classifier and a data mining classifier to act as anomaly detection and misuse detection to improve the DR and FPR, respectively. The former consists of a set of base feature selecting classifiers and each uses partial feature space.

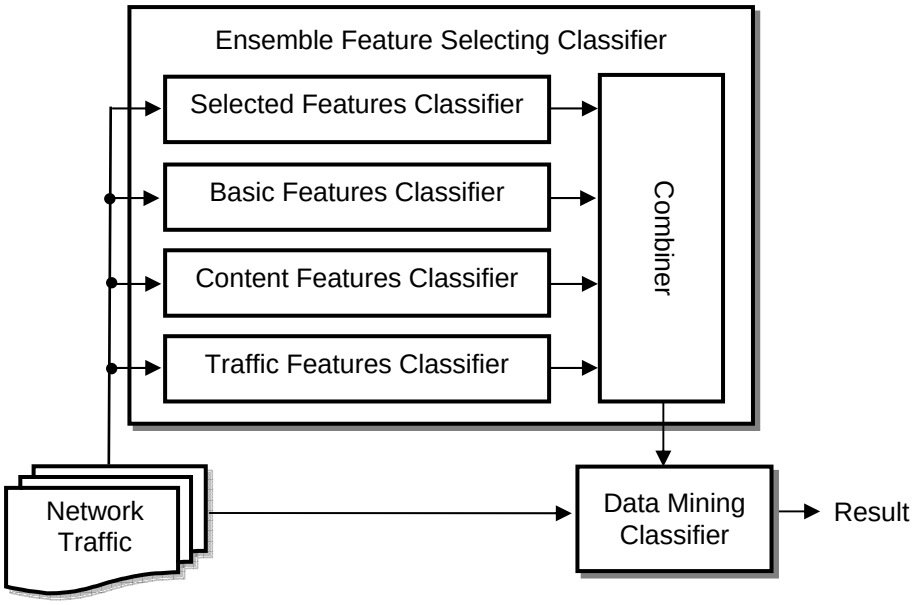


Figure 7.1. Ensemble Intrusion Detection Model

The latter applies data mining technique to look for patterns of normal activities. We believe that the overall performance of this ensemble architecture is better than that of each individual classifier. Also, DR, FPR, and detection speed are more accurate and faster than those of using full feature set. Figure 7.1 depicts our design.

A. Ensemble Feature Selection Classifier

In an ensemble classifier design, it is important to understand that individual base classifiers should be independent of each other. If the base classifiers provide similar outputs, then no significant improvement of the ensemble result can be obtained through the combination process. It is critical to notice the diversity among base classifiers in order to get effective and correct classification result. Hence, we decide to use *ensemble feature selection* approach to be our feature selecting classifier structure. By choosing dissimilar feature subsets for various base feature selecting classifiers, the diversity among these classifiers is expected to be maximized to achieve a better result. In our

design, we select four distinct subsets of features to be the foundation of the base classifiers. One subset consists of features selected by our feature selection algorithm. The other three are the partitions of the original 41 features that are 9 basic features (1 to 9), 13 content features (10 to 22), and 19 traffic features (23 to 41).

Besides the notability of multiplicity among the base classifiers, the right choice of a combination method is another important issue in creating a successful ensemble result. As described in Section 2.6, research has shown that there are many methods [59] available for combining the abstract form outputs of the base classifiers into an ensemble result. However, some of them in fact are not suitable for our ensemble intrusion detection design. For example, the behavior knowledge space method requires enough representative data sets to estimate high order distribution of classifiers' outputs. Otherwise overfitting is likely to occur, and the generalization error quickly increases [85]. In our designed intrusion detection model, we only use a very small amount of traffic data that is insufficient to offer behavior knowledge space method a representative number of observations. While the majority voting ensemble approach is used for integration, Hansen and Salamon [62] had proved that the ensemble only works if the CR of individual base classifier is higher than 50% accurate and independent of each other. However, we cannot guarantee that all the classification accuracies of our designed four base classifiers would satisfy the above requirement. Accordingly, naive bayes ensemble is selected to combine the decisions of base classifiers together. Based on its probabilistic approach, the evidences of base classifiers are computed and the most appropriate class can then be chosen. Its operation is explained as follows.

Let the possible classes of a system be $l_1, l_2, ..., l_p \in L$ and these p classes are mutually exclusive and exhaustive, i.e., the decision result of the system belongs to only one of the classes. The set $D = \{d_1, d_2, ..., d_m\}$ denotes a set of m base feature selecting classifiers and each of them is built from a feature subset of the feature space $F = \{f_1, f_2, ..., f_n\}$. The output of each base classifier o is an abstract form class label. Then the objective is to find the probability over a class member l conditional on the outcomes of m classifiers, h_1 through h_m.

$$P(h_1, ..., h_m \mid l_i) = \prod_{j=1}^{m} P(o_{ij} \mid l_i) \tag{7.1}$$

$$P(h_1, ..., h_m) = \sum_{i=1}^{p} P(h_1, ..., h_m \mid l_i) P(l_i) \tag{7.2}$$

Here, $P(h_1, ..., h_m \mid l_i)$ is the conditional probability of $h_1, ..., h_m$ given l_i and $P(o_{ij}|l_i)$ is the conditional probability of o_{ij} given l_i. The prior probability of each class is $P(l_i)$ and $P(h_1, ..., h_m)$ is the probability of $h_1, ..., h_m$.

Based on the Bayes theorem, we have

$$P(l_i \mid h_1, ..., h_m) = \frac{P(h_1, ..., h_m \mid l_i) P(l_i)}{P(h_1, ..., h_m)} \tag{7.3}$$

where i is the number of possible classes ranging from 1 to p. This posterior probability collects all evidences from base classifiers and integrates them together. Finally, the naive bayes classifier infers the state of system by choosing a class that achieves the highest posterior probability.

To illustrate how naive bayes ensemble method works, we assume there are 4 intrusion detection base feature selecting classifiers h_1, h_2, h_3, and h_4 that are built by features

Table 7.1. Intrusion Detection Accuracies of Four Base Feature Selecting Classifiers

	Classifier 1	Classifier 2	Classifier 3	Classifier 4
FPR	20%	10%	30%	10%
DR	70%	60%	50%	90%

selected by our feature selection algorithm, 9 basic features, 13 content features, and 19 traffic features, respectively. The intrusion detection task is a binary assignment, i.e., each base classifier assigns the network traffic data into either normal activity l_1 or attack l_2.

Suppose we have tested an amount of network traffic data and the distributions of the normal activities and attacks of the testing set are 60% and 40%, respectively. The FPRs and DRs of four base feature selecting classifiers are shown in Table 7.1. Here the goal is to classify a future network traffic data x into normal activity if $P(l_1 \mid h_1, h_2, h_3, h_4) > P(l_2 \mid h_1, h_2, h_3, h_4)$, else into attack category. Now assume a traffic data passes through these 4 base classifiers. The first, second and fourth classifiers identify it as a normal usage, however the third one recognizes it as an attack. Then,

$$P(h_1, h_2, h_3, h_4 \mid l_1) = \prod_{j=1}^{4} P(o_{1j} \mid l_1) = (1-0.2)\cdot(1-0.1)\cdot 0.3\cdot(1-0.1) = 0.1944 \quad (7.4)$$

$$P(h_1, h_2, h_3, h_4 \mid l_2) = \prod_{j=1}^{4} P(o_{2j} \mid l_2) = (1-0.7)\cdot(1-0.6)\cdot 0.5\cdot(1-0.9) = 0.006 \quad (7.5)$$

$$\begin{aligned} P(h_1, h_2, h_3, h_4) &= \sum_{i=1}^{2} P(h_1, h_2, h_3, h_4 \mid l_i) P(l_i) \\ &= 0.1944\cdot 0.6 + 0.006\cdot 0.4 = 0.11904 \end{aligned} \quad (7.6)$$

Hence, the joint posterior probabilities of $P(l_1 \mid h_1, h_2, h_3, h_4)$ and $P(l_2 \mid h_1, h_2, h_3, h_4)$ based on the above calculation and the prior probabilities of normal activities and attacks are

$$P(l_1 \mid h_1, h_2, h_3, h_4) = \frac{P(h_1, h_2, h_3, h_4 \mid l_1)P(l_1)}{P(h_1, h_2, h_3, h_4)} = \frac{0.1944 \cdot 0.6}{0.11904} = 0.9798 \quad (7.7)$$

$$P(l_2 \mid h_1, h_2, h_3, h_4) = \frac{P(h_1, h_2, h_3, h_4 \mid l_2)P(l_2)}{P(h_1, h_2, h_3, h_4)} = \frac{0.006 \cdot 0.4}{0.11904} = 0.0202 \quad (7.8)$$

We have 97.98% degree of confidence that the incoming traffic data x belongs to class l_1 which is greater than that of class l_2. We therefore conclude x is a normal computer user activity.

B. Data Mining Classifier

Having finished the process of ensemble classification, another important concern is the problem of false alarm rate. Due to the entire scope of both normal and attack behavior is covered during the training procedure, much study has shown that one of the most common problems of anomaly intrusion detection is too many false alarms might happen likely resulted from normal behavior. Hence, we suggest a two levels model that combines parallel and serial ensemble topologies together for getting a better quality of detection. In the second level data mining classifier, we utilize data mining technique to construct a filter to eliminate false alarms.

Data mining technique provides strategy to find useful information from large amount of data and induce inferences from those information. Here, we use C4.5 decision trees algorithm to extract patterns from training data. The goal is to find rules that represent

normal behavior of network traffic stream for our intrusion detection task. In this way, we can write decision rules as follows.

Rule: IF *conditions of features* THEN *the traffic is a normal behavior*

In each rule, the *antecedent* part consists of a number of conditions that are satisfied by the features. The *consequent* action of that rule is defined as the analyzed network traffic data is a normal behavior.

The data mining classifier compares the result of first level ensemble classifier with those well defined normal patterns, and the normal computer user activity to the system can be identified if the data is matched with one of these defined patterns. The data mining classifier has a higher priority to determine a traffic data is whether a normal behavior or not if it has a disagreement with the result of ensemble classifier.

7.2 Experimental Results

Continuing on the experiments of Chapters V and VI, we perform experiments over the *KDD99* data set. As shown in the previous section, we propose a two levels model, ensemble feature selecting classifier and data mining classifier, to classify network traffic into normal and malicious activities. In the experiments, we first implement the data mining classifier which uses C4.5 decision trees algorithm to extract rules from the training set.

Next, we implement the ensemble feature selecting classifier, which is constructed by four individual base classifiers and each models the system behavior from a single aspect of view point. By varying the feature subsets and maximizing the disagreement among four base classifiers, we use four distinct feature representations from the original feature

Table 7.2. DRs of Classifiers Performed on *Normal-U2R* Data Set

Classifier	Full Set		Ours		Basic		Content		Traffic	
	FPR	DR	FPR	DR	FPR	DR	FPR	DR	FPR	DR
k-NN	0.20	12.87	2.55	18.03	6.33	19.98	0.09	18.57	1.06	15.12
Fuzzy *k*-NN	0.27	15.49	2.50	18.47	5.94	26.40	0.09	26.40	0.99	15.85
Evidence-Theoretic *k*-NN	0.31	16.87	2.65	19.64	7.16	28.74	0.12	20.24	1.13	15.80
Fuzzy Belief *k*-NN	13.71	98.16	9.54	83.86	9.48	59.01	11.38	74.42	39.74	54.26

Table 7.3. DRs of Classifiers Performed on *Normal-R2L* Data Set

Classifier	Full Set		Ours		Basic		Content		Traffic	
	FPR	DR	FPR	DR	FPR	DR	FPR	DR	FPR	DR
k-NN	0.36	18.91	3.68	20.41	8.11	17.14	0.24	22.62	1.43	28.71
Fuzzy *k*-NN	0.33	20.92	19.30	23.58	21.48	24.39	0.25	24.39	1.41	28.80
Evidence-Theoretic *k*-NN	0.39	21.50	8.76	26.33	10.73	20.40	0.27	26.65	1.52	29.60
Fuzzy Belief *k*-NN	11.58	67.88	9.86	69.82	9.70	60.66	11.43	53.07	19.65	51.66

space, they are our feature selection result, the 9 basic features, the 13 content features, and the 19 traffic features. For each base classifier, fuzzy belief *k*-NN algorithm is used. Using the same experimental setup in Section 6.2, the experiments are performed on the binary (normal/attack) classification and 10 trials are performed for each *k* nearest neighbor experiment. For detecting *U2R* attacks, the training and testing sets include 930 (878 normal and 52 *U2R*) and 694 (479 normal and 215 *U2R*) connections, respectively. For detecting *R2L* attacks, the training and testing sets include 977 (878 normal and 99 *R2L*) and 770 (479 normal and 291 *R2L*) connections, respectively. The results of our experiments are summarized in Tables 7.2 and 7.3 for *Normal-U2R* and *Normal-R2L* data sets, respectively. For each of them, the tables show the averaged FPR, DR, and CR on the test sets of four base classifiers trained using full feature set and four diverse feature subsets. The averaged accuracies are reported with *k* ranging from 1 to 10 where the classification algorithm is run 10 times for each *k*.

The results show that three classifiers, *k*-NN, fuzzy *k*-NN and evidence-theoretic *k*-NN classifiers, performed in different feature sets have similar detection performances using either full feature set or one of feature subsets, which all of them have poor detection performances. In rows 1 to 3 of Tables 7.2 and 7.3, the maximum DRs are 28.74% and 29.60% for *Normal-U2R* and *Normal-R2L* data sets, respectively. These three classifiers have low FPRs because they treat most network traffic data as normal connections no matter these connections are normal or malicious activities. With our proposed fuzzy belief *k*-NN classifier, we achieve higher averaged accuracies in comparison with the outcomes of the other three *k*-NN based classifiers. Especially in the *Normal-R2L* data set, we get the highest classification accuracy by using 7 out of 41 features, which is better than that of using full feature set.

Having built individual base feature selecting base classifiers, we then proceed to generate the ensemble one by fusing the outputs of base classifiers together. In addition, we utilize data mining technique to look for patterns of normal activities in training set and then to produce a set of ten decision rules which cover 95% of normal behavior. With the combination of ensemble feature selecting classifier and data mining classifier, the final intrusion detection model is thus obtained. Figures 7.2 and 7.3 show the comparisons between fuzzy belief *k*-NN classifier using full feature set and ensemble intrusion detection model. Table 7.4 shows the decision rules.

With our proposed ensemble model, we achieve 95.45% and 87.33% averaged CRs for *Normal-U2R* and *Normal-R2L* data sets, respectively, which are higher than 89.97% and 80.66% of the single fuzzy belief *k*-NN classifier using full feature set. As shown in the ROC graphs, all the points of ensemble model are much closer to point (0, 1) than those

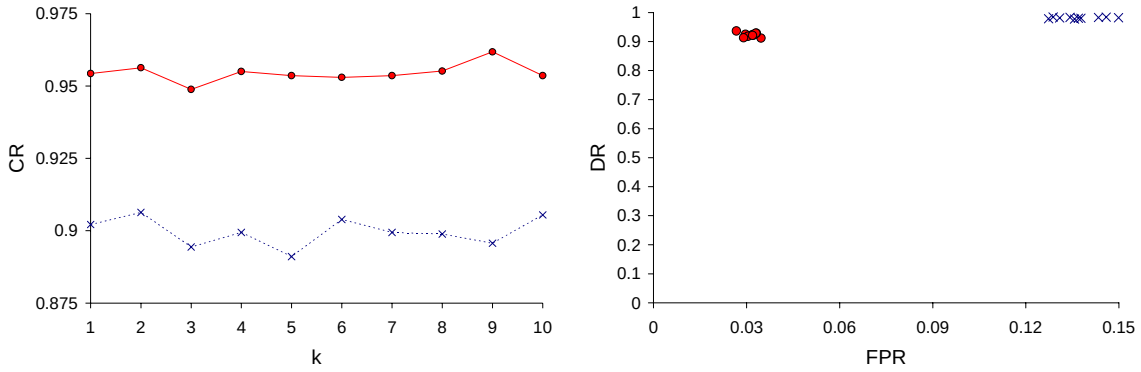


Figure 7.2. CRs (left) and ROC Graph (right) of *Normal-U2R* Data Set with *k* Ranging From 1 to 10
x: Fuzzy Belief *k*-NN Classifier Using Full Feature Set
•: Ensemble Intrusion Detection Model

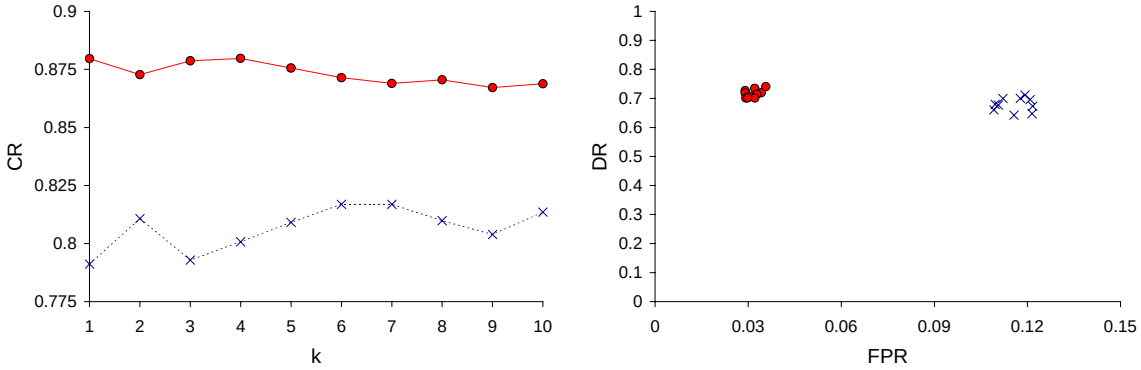


Figure 7.3. CRs (left) and ROC Graph (right) of *Normal-R2L* Data Set with *k* Ranging From 1 to 10
x: Fuzzy Belief *k*-NN Classifier Using Full Feature Set
•: Ensemble Intrusion Detection Model

of classifier using full feature set, which indicate the ensemble model not only has a high DR but also has a low FPR. To show the performance of ensemble model for specific intrusions, Table 7.5 describes the detailed DRs. The result shows that our model is capable of detecting most of the intrusions, especially 7 intrusions (*loadmodule*, *perl*, *sqlattack*, *imap*, *worm*, *xlock*, and *xsnoop*) are detected perfectly. In addition, we test the detection time on each testing connection and illustrate the result in Figure 7.4. The results show that we successfully reduce the detection time which our ensemble model

only needs 0.44 and 0.41 of that of classifier using full feature set in *Normal-U2R* and *Normal-R2L* data sets, respectively. The results have been published in [86].

Table 7.4. Decision Rules

Rule 1	Rule 2	Rule 3
IF *wrong_fragment* < 3 AND *num_compromised* < 1 AND *srv_serror_rate* < 0.06 AND *rerror_rate* < 0.06 AND *flag* = SF AND *hot* < 1 AND *protocol_type* = tcp AND *service* = http THEN normal connection	IF *wrong_fragment* < 3 AND *num_compromised* < 1 AND *srv_serror_rate* < 0.06 AND *rerror_rate* < 0.06 AND *flag* = SF AND *hot* < 1 AND *protocol_type* = tcp AND *service* = smtp THEN normal connection	IF *wrong_fragment* < 3 AND *num_compromised* < 1 AND *srv_serror_rate* < 0.06 AND *rerror_rate* >= 1 AND *hot* < 1 AND *service* = 20 AND *diff_srv_rate* < 0.01 AND *flag* = REJ THEN normal connection
Rule 4	**Rule 5**	**Rule 6**
IF *wrong_fragment* < 3 AND *num_compromised* < 1 AND *srv_serror_rate* < 0.06 AND *rerror_rate* < 0.06 AND *flag* = SF AND *hot* < 1 AND *protocol_type* = udp AND *dst_host_rerror_rate* < 0.01 AND *diff_srv_rate* < 0.01 AND *src_bytes* >= 30 AND *src_bytes* < 158 THEN normal connection	IF *wrong_fragment* < 3 AND *num_compromised* < 1 AND *srv_serror_rate* < 0.06 AND *rerror_rate* < 0.06 AND *flag* = SF AND *hot* < 1 AND *protocol_type* = udp AND *dst_host_rerror_rate* < 0.01 AND *diff_srv_rate* >= 0.5 THEN normal connection	IF *wrong_fragment* < 3 AND *num_compromised* < 1 AND *srv_serror_rate* < 0.06 AND *rerror_rate* < 0.06 AND *flag* = SF AND *hot* < 1 AND *protocol_type* = tcp AND *service* = 17 AND *dst_bytes* < 29 AND *duration* < 1 AND *src_bytes* >= 5 AND *src_bytes* < 30 THEN normal connection
Rule 7	**Rule 8**	**Rule 9**
IF *wrong_fragment* < 3 AND *num_compromised* < 1 AND *srv_serror_rate* < 0.06 AND *rerror_rate* < 0.06 AND *flag* = SF AND *hot* < 1 AND *protocol_type* = tcp AND *service* = ftp_data AND *dst_bytes* < 29 AND *duration* < 1 AND *src_bytes* >= 32 AND *src_bytes* < 246 THEN normal connection	IF *wrong_fragment* < 3 AND *num_compromised* < 1 AND *srv_serror_rate* < 0.06 AND *rerror_rate* < 0.06 AND *flag* = SF AND *hot* < 1 AND *protocol_type* = tcp AND *service* = ftp_data AND *dst_bytes* < 29 AND *duration* < 1 AND *src_bytes* >= 248 AND *src_bytes* < 334 THEN normal connection	IF *wrong_fragment* < 3 AND *num_compromised* < 1 AND *srv_serror_rate* < 0.06 AND *rerror_rate* < 0.06 AND *flag* = SF AND *hot* < 1 AND *protocol_type* = tcp AND *service* = ftp_data AND *dst_bytes* < 29 AND *duration* < 1 AND *src_bytes* >= 335 AND *src_bytes* < 644 THEN normal connection
Rule 10		
IF *wrong_fragment* < 3 AND *num_compromised* < 1 AND *srv_serror_rate* < 0.06 AND *rerror_rate* < 0.06 AND *flag* = SF AND *hot* < 1 AND *protocol_type* = tcp AND *service* = ftp_data AND *dst_bytes* < 29 AND *duration* < 1 AND *src_bytes* >= 726 THEN normal connection		

Table 7.5. Detection Performances of Ensemble Model

Normal-U2R		*Normal-R2L*	
FPR	3.13	FPR	3.15
DR	92.30	DR	71.66
buffer_overflow	97.00	ftp_write	93.75
loadmodule	100.00	guess_passwd	97.96
perl	100.00	imap	100.00
rootkit	50.77	multihop	92.96
httptunnel	98.92	phf	0.00
ps	77.19	warezmaster	74.61
sqlattack	100.00	named	77.40
xterm	67.08	sendmail	57.58
		snmpgetattack	0.00
		snmpguess	1.15
		worm	100.00
		xlock	100.00
		xsnoop	100.00

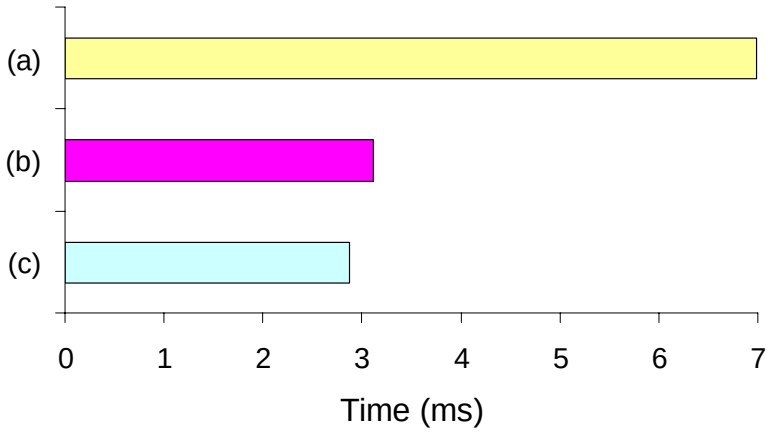


Figure 7.4. Detection Time on One Connection
(a) Full Feature Set
(b) Ensemble Model in *Normal-U2R* set
(c) Ensemble Model in *Normal-R2L* set

CHAPTER VIII

CONCLUSIONS

This research presents a solution to the problem of quickly and accurately detecting computer intrusions from network traffic data. The goal of our work has been set to detect attacks that attackers use illegal approaches to gain access to the target host and thus further to exploit the system's vulnerabilities. The proposed solution includes three major parts. They are feature selection algorithm, fuzzy belief machine learning algorithm, and ensemble intrusion detection model.

8.1 Feature Selection Algorithm

The first question in intrusion detection design is what features of a network traffic should be used to build the model. We start with our research on studying this problem and developing a correlation-based feature selection algorithm to select a set of most significant features that reserve vital information of traffic data. For evaluating the performance of our algorithm, six small and four large databases are used. We observe that the classification accuracy is improved by using our selected features from the original feature set, e.g., the CR of the abalone set performing on C4.5 algorithm with 2 out of 8 features, the DR of *Normal-DoS* set performing on Naive Bayes algorithm with 12 out of 41 features, and the DR of *Normal-R2L* set performing on C4.5 algorithm with 7 out of 41 features. We also observe that our algorithm has a superior performance compared with that of two participating correlation-based feature selection algorithms,

CFS and FCBF, in both small and large data sets. Especially in the *Normal-U2R* and *Normal-R2L* data sets, CFS can detect none of the attacks and FCBF can only detect around 7% and 30% of *U2R* and *R2L* attacks, respectively. On the other hand, the detection performance of our selected features is close to that of using full feature set. The result shows that our algorithm approach can be a practical feature selector to select informative features from data sets for classification tasks.

8.2 Fuzzy Belief Machine Learning Algorithm

We study the problems of uncertainty and ambiguity in audit network traffic data. The key idea is to imitate ambiguous of users' activities by fuzzy clustering technique, and to simulate uncertainty caused by limited information by incorporating only a small amount of network traffic data for analysis. With the use of Dempster-Shafer theory, we identify future network traffic by fusing evidences found in clustering development. Also, we employ *k*-NN technique to speed up the detection process. The experimental result shows that our approach is capable of detecting *U2R* attacks with an averaged DR of 98.16% using the full feature space. While detecting *R2L* attacks, we achieve an averaged DR of 67.88%. Compared with the past research results [6]-[12] having very low DRs, we successfully improve the detection on those two attacks that contain degrees of ambiguous information. However, we do have relatively high false alarm rates of 13.71% and 11.58% for *U2R* and *R2L* attacks, respectively. That is the stimulus that we incorporate data mining technique in the ensemble model, which we hope to further reduce the number of false alarms.

8.3 Ensemble Intrusion Detection Model

Having finished the development of feature selection and fuzzy belief machine learning algorithms, we then integrate them together to check whether our selected feature set is feasible to be applied to our developed detection method or not. The experimental result shows a DR of 83.86% is achieved in detecting *U2R* attacks, which drops a lot compared with the rate of using full feature set. It indicates that five features selected from the training set are not sufficient to cover all the attacks information that appears in the testing set. On the other hand, we get a DR of 69.82% in detecting *R2L* attacks by using only seven features, which is higher than that of full feature set. Also, the false alarm rates are reduced to 9.54% and 9.86% for *U2R* and *R2L* attacks, respectively.

For further improving the detection performance, we propose an ensemble intrusion detection model that consists of a set of feature selecting base fuzzy belief classifiers and a data mining classifier. The basic idea is using ensemble feature selection technique to promote the DR and data mining technique to reduce the number of false alarms. It is a combination of both anomaly detection and misuse detection. With our proposed ensemble model, we achieve 92.30% and 71.66% averaged DRs and 3.13% and 3.15% averaged false alarm rates in detecting *U2R* and *R2L* attacks, respectively. The false alarm rate is significant improved from around 10% to 3%. As for the CRs, the averaged rates are 95.45% and 87.33% for *U2R* and *R2L* attacks, respectively, which are higher than 89.97% and 80.66% if a single fuzzy belief machine learning algorithm with full feature set is used. This indicates that the detection performance is successfully improved through our proposed ensemble intrusion detection model.

8.4 Future Work

Up to now, this dissertation has developed an intrusion detection system based on ensemble of multiple base classifiers. However, there are several topics that deserve to be future studied.

- *False alarms*: In the design of an intrusion detection system, not only a high DR is necessary but also a low false alarms rate is required. Though it is not easy to control the false alarm rate because many unusual events sometimes are classified to hostile activities and most of these unusual events are actually normal behavior. In our research, we use data mining technique to extract decision rules from training set and achieve about 3% false alarms rate. We believe that this rate can be further reduced in the future if a more dedicated rule set can be built.
- *Respond to the intrusions*: In our work we focus on developing a detection method which can efficiently and effectively differentiate intrusive activities from large volume of network events. We believe that the response to the intrusions is also equally important. Once an intrusion is happened, it is necessary to properly present the alarm in order that system administrator can make proper and prompt decision. In a word, to find a method to integrate the intrusion detection system with the intrusion response system deserves further research.
- *Feature selection*: Feature selection plays an important role on both speed and accuracy of intrusion detection. It selects the most informative features that cover normal and intrusive activities by analyzing large quantity of network traffic data. In this dissertation we have developed a feature selection algorithm based on

symmetrical uncertainty measure to remove the worthless information from the original high dimensional database. However, we think there are still some issues that can be explored in order to get a better performance, e.g., relevant and redundant features analysis and discretization methods.

- *Multiple identification ability*: The *KDD99* data set includes four groups of attacks and each uses diverse skill to explore system's vulnerabilities. In our work we use binary classification technique to identify a network event as either normality or abnormality. The future research will be directed to upgrade the system with multiple identification ability, i.e., the system will be able to classify a network traffic data into normal activity or one of four attacks.
- *Real time attack detection*: We have made contributions in detecting both *U2R* and *R2L* attacks off-line in a publicly available intrusion detection database *DARPA KDD99*. However, the database we used is eight years old that is not enough to reflect the current status of Internet. In the future, the research can be expanded to the live Internet traffic based on our past achievement.

REFERENCES

1. CERT Coordination Center, Software Engineering Institute, Carnegie Mellon University. URL: http://www.cert.org (Last browsed in December 2007)

2. H. Debar, “An Introduction to Intrusion-Detection Systems,” Proceedings of Connect’2000, Doha, Qatar, April 2000.

3. J. P. Anderson, *Computer Security Threat Monitoring and Surveillance*, Technical Report, James P. Anderson Co., Fort Washington, PA, April 1980.

4. I. Guyon and A. Elisseeff, “An Introduction to Variable and Feature Selection,” Journal of Machine Learning Research, Volume 3, pp. 1157-1182, 2003.

5. K. Jones and R. S. Sielken, *Computer System Intrusion Detection: A Survey*, Technical Report, Computer University of Virginia, 2000.

6. I. Levin, “KDD-99 Classifier Learning Contest LLSoft’s Results Overview,” ACM SIGKDD Explorations Newsletter, Volume 1, issue 2, pp. 67-75, January 2000.

7. H. G. Kayacik, A. N. Zincir-Heywood, and M. I. Heywood, “On the Capability of an SOM Based Intrusion Detection System,” Proceedings of the International Joint Conference on Neural Networks, Volume 3, pp. 1808-1813, July 2003.

8. M. Sabhnani and G. Serpen, “Application of Machine Learning Algorithms to KDD Intrusion Detection Dataset within Misuse Detection Context,” Proceedings of the International Conference on Machine Learning, Models, Technologies and Applications, Las Vegas, NV, pp. 209-215, June 2003.

9. M. Sabhnani and G. Serpen, “Why Machine Learning Algorithms Fail in Misuse Detection on KDD Intrusion Detection Data Set,” Intelligent Data Analysis, Volume 8, Number 4, pp. 403-415, 2004.

10. D. Song, M. I. Heywood, and A. N. Zincir-Heywood, "Training Genetic Programming on Half a Million Patterns: An Example from Anomaly Detection," IEEE Transactions on Evolutionary Computation, 9(3), pp. 225-240, 2005.

11. S. Zhong, T. M. Khoshgoftaar, and N. Seliya, “Clustering-based Network Intrusion Detection,” International Journal of Reliability, Quality, and Safety Engineering, 2005.

12. K. M. Faraoun and A. Boukelif, “Neural Networks Learning Improvement using the k-Means Clustering Algorithm to Detect Network Intrusions,” International Journal of Computational Intelligence, Volume 3, Number 2, pp. 161-168, 2006.

13. J. C. Bezdek, *Pattern Recognition with Fuzzy Objective Function Algorithms*, Plenum Press, New York, 1981.

14. J. C. Dunn, "A Fuzzy Relative of the ISODATA Process and its Use in Detecting Compact Well-Separated Clusters," Journal of Cybernetics, Volume 3, pp. 32-57, 1973.

15. A. P. Dempster, "A Generalization of Bayesian Inference," Journal of the Royal Statistical Society, Series B, Volume 30, pp. 205-247, 1968.

16. G. Shafer, *A Mathematical Theory of Evidence, Princeton*, University Press, Princeton, NJ, 1976.

17. DARPA Intrusion Detection Evaluation, MIT Lincoln Laboratory.
URL: http://www.ll.mit.edu/IST/ideval/ (Last browsed in December 2007)

18. E. Fix and J. L. Hodges, "Discriminatory Analysis: Nonparametric Discrimination: Consistency Properties," Report Number 4, Project Number 21-49-004, USAF School of Aviation Medicine, Randolph Field, Texas, 1951.

19. E. Denning, "An Intrusion-Detection Model," IEEE Transactions on Software Engineering, Volume 13, Number 2, pp. 222-232, 1987.

20. S. E. Smaha, "Haystack: An Intrusion Detection System," Fourth Aerospace Computer Security Applications Conference, pp. 37-44, Austin, Texas, 1988.

21. J. D. Howard, *An Analysis of Security Incidents on the Internet 1989 – 1995*, Dissertation, Carnegie Mellon University, Pittsburgh, Pennsylvania, 1997.

22. M. Dekker, "Security of the Internet," The Froehlich/Kent Encyclopedia of Telecommunications, Volume 15, pp. 231-255, New York, 1997.

23. T. Crothers, *Implementing Intrusion Detection Systems, A Hands-On Guide for Securing the Network*, Wiley Publishing, Inc., 2003.

24. H. G. Kayacık, *Hierarchical Self Organizing Map Based IDS on KDD Benchmark*, Master Thesis, Dalhouse University, Halifax, Nova Scotia, Canada, 2003.

25. S. Kumar, *Classification and Detection of Computer Intrusions*, PhD Thesis, Purdue University, 1995.

26. J. Z. Lei and A. Ghorbani, "Network Intrusion Detection Using an Improved Competitive Learning Neural Network," 2nd Annual Conference on Communication Networks and Services Research, pp. 190-197, 2004.

27. S. Zanero and S. M. Savaresi, “Unsupervised Learning Techniques for an Intrusion Detection System,” Proceedings of the 14th ACM Symposium on Applied Computing, 2004.

28. Sourcefire Inc, URL: http://www.snort.org (Last browsed in December 2007)

29. S. Northcutt and J. Novak, *Network Intrusion Detection*, 2003.

30. J. Cannady, “Artificial Neural Networks for Misuse Detection,” Proceedings of the 1998 National Information Systems Security Conference, pp. 443-456, Arlington, VA., 1998.

31. T. Kohonen, “Automatic Formation of Topological Maps of Patterns in a Self-Organizing System,” Proceedings of 2nd Scandinavian Conference on Image Analysis, pp. 214-220, Helsinki, Finland, 1981.

32. D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning Internal Representations by Error Backpropagation,” Processing of Parallel Distributed, Volume 1, The MIT Press, pp. 318-362, 1986.

33. O. Depren, M. Topallar, E. Anarim and M. K. Ciliz, “An Intelligent Intrusion Detection System (IDS) for Anomaly and Misuse Detection in Computer Networks,” Expert Systems with Applications, Volume 29, Issue 4, pp. 713-722, November 2005.

34. L. L. DeLooze, “Attack Characterization and Intrusion Detection using an Ensemble of Self-Organizing Maps,” 2006 International Joint Conference on Neural Networks, pp. 2121-2128, Vancouver, BC, Canada, July, 2006.

35. L. A. Zadeh, “Fuzzy Sets,” Infomation Control, Volume 8, pp. 338-353, 1965.

36. S. M. Bridges and R. B. Vaughn, “Intrusion Detection via Fuzzy Data Mining,” Proceedings of the 12th Annual Canadian Information Technology Security Symposium, pp.109-122, Ottawa, Canada, June 2000.

37. J. T. Yao, S. L. Zhao, and L. V. Saxton, “A Study on Fuzzy Intrusion Detection,” Proceedings of SPIE, Data Mining, Intrusion Detection, Information Assurance, And Data Networks Security, Orlando, Florida, USA, pp. 23-30, 2005.

38. J. E. Dickerson and J. A. Dickerson, “Fuzzy Network Profiling for Intrusion Detection,” Proceedings of NAFIPS 19th International Conference of the North American Fuzzy Information Processing Society, pp. 301-306, Atlanta, July 2000.

39. J. E. Dickerson, J. Juslin, O. Koukousoula, and J. A. Dickerson, “Fuzzy Intrusion Detection,” IFSA World Congress and 20th North American Fuzzy Information

Processing Society International Conference, Volume 3, pp. 1506-1510, Vancouver, British Columbia, July 2001.

40. A. Abraham and R. Jain, "Soft Computing Models for Network Intrusion Detection Systems," Soft Computing in Knowledge Discovery: Methods and Applications, Studies in Fuzziness and Soft Computing, Springer Verlag, Chapter 16, Germany, 2004.

41. J. Gomez and D. Dasgupta, "Evolving Fuzzy Classifiers for Intrusion Detection," Proceedings of the IEEE Workshop on Information Assurance, West Point, NY June 2002.

42. G. Florez, S. M. Bridges, and R. B. Vaughn, "An Improved Algorithm for Fuzzy Data Mining for Intrusion Detection," Proceedings of the North American Fuzzy Information Processing Society Conference, CDROM Proceeding, New Orleans, LA, June, 2002.

43. W. Lee and S. J. Stolfo, "Data Mining Approaches for Intrusion Detection," Proceedings of the 7th USENIX Security Symposium, San Antonio, Texas, 1998.

44. S. Kumar and E. H. Spafford, "A Software Architecture to Support Misuse Intrusion Detection," Proceedings of the 18th National Conference on Information Security, pp. 194-204. 1995.

45. K. Ilgun, R. A. Kemmerer, and P. A. Porras, "State Transition Analysis: A Rule-Based Intrusion Detection Approach," IEEE Transactions on Software Engineering, Volume 21, Number 3, pp. 181-199, March 1995.

46. W. Lee and S. J. Stolfo, "A Framework for Constructing Features and Models for Intrusion Detection Systems," ACM Transactions on Information and System Security, Volume 3, Number 4, pp. 227-261, 2000.

47. W. W. Cohen, "Fast Effective Rule Induction," Proceedings of 12th International Conference on Machine Learning, San Mateo, CA, 1995.

48. M. Sabhnani and G. Serpen, "KDD Feature Set Complaint Heuristic Rules for R2L Attack Detection," Security and Management, pp. 310-316, 2003.

49. M. Sabhnani and G. Serpen, "Formulation of a Heuristic Rule for Misuse and Anomaly Detection for U2R Attacks in Solaris Operating System Environment," Security and Management, pp. 390-396, 2003.

50. G. Serpen and M. Sabhnani, "Measuring Similarity in Feature Space of Knowledge Entailed by Two Separate Rule Sets," Knowledge Based Systems, Volume 19, Number 1, pp. 67-76, 2006.

51. J. R. Quinlan, *C4.5: Programs for Machine Learning*, Morgan Kaufmann, 1993.

52. I. Guyon and A. Elisseeff, "An Introduction to Variable and Feature Selection," Journal of Machine Learning Research, Volume 3, pp. 1157-1182, 2003.

53. G. John, R. Kohavi, and K. Pfleger, "Irrelevant Features and the Subset Selection Problem," Proceedings ML-94, pp. 121-129, Morgan Kaufmann, 1994.

54. H. G. Kayacık, A. N. Zincir-Heywood, and M. I. Heywood, "Selecting Features for Intrusion Detection: A Feature Relevance Analysis on KDD 99 Intrusion Detection Datasets," Third Annual Conference on Privacy, Security and Trust, St. Andrews, New Brunswick, Canada, October 2005.

55. J. R. Quinlan, "Induction of Decision Trees," Machine Learning, Volume 1, pp. 81-106, 1986.

56. J. Biesiada and W. Duch, "Feature Selection for High-Dimensional Data: A Kolmogorov-Smirnov Correlation-Based Filter Solution," Proceedings of the 4th International Conference on Computer Recognition Systems, 2005.

57. S. Mukkamala and A. H. Sung, "Feature Selection for Intrusion Detection Using Neural Networks and Support Vector Machines", Journal of the Transportation Research Board of the National Academics, Transportation Research Record No 1822, pp. 33-39, 2003.

58. Y. Lu, "Knowledge Integration in a Multiple Classifier System," Application Intelligence, 6(2), pp. 75–86, 1996.

59. L. Xu, A. Krzyzak and C.Y. Suen, "Several Methods for Combining Multiple Classifiers and Their Applications in Handwritten Character Recognition," IEEE Transactions on System, Man and Cybernetics, SMC-22(3), pp. 418-435, 1992.

60. G. Giacinto and F. Roli, "Intrusion Detection in Computer Networks by Multiple Classifier Systems," 16th International Conference on Pattern Recognition, Volume 2, pp. 390-393, 2002.

61. S. Mukkamala, A. H. Sung, and A. Abraham, "Intrusion Detection Using an Ensemble of Intelligent Paradigms," Journal of Network and Computer Applications, Volume 28, Issue 2, pp. 167-182, 2005.

62. L. K. Hansen and P. Salamon, "Neural Network Ensembles," IEEE Transactions on Pattern Analysis Machine Intelligence, 12(10), pp. 993-1001, 1990.

63. L. Yu and H. Liu, “Feature Selection for High-Dimensional Data: A Fast Correlation-Based Filter Solution,” Proceedings of The Twentieth International Conference on Machine Leaning, pp. 856-863, Washington, D.C., August, 2003.

64. C. Shannon, “A Mathematical Theory of Communication,” Bell System Technical Journal, 1948.

65. W. H. Press, B. P. Flannery, S. A. Teukolski, and W. T. Vetterling, *Numerical Recipes in C*, Cambridge University Press, 1988.

66. S. Zanero and S. M. Savaresi, “Unsupervised Learning Techniques for an Intrusion Detection System,” Proceedings of the 14th ACM Symposium on Applied Computing, 2004.

67. C. Kruegel, E. Kirda, D. Mutz, W. Robertson, and G. Vigna, “Polymorphic Worm Detection Using Structural Information of Executables,” 8th Symposium on Recent Advances in Intrusion Detection, Lecture Notes in Computer Science, Springer Verlag, USA, September 2005.

68. J. M. Booker, M. C. Anderson, M. A. Meyer, “The Role of Expert Knowledge in Uncertainty Quantification (Are We Adding More Uncertainty or More Understanding?),” Seventh Army Conference on Applied Statistics, pp. 155-161, 2001.

69. W. L Oberkampf, J. C. Helton, C. A. Jos lyn, S. F. Wojtkiewicz, and S. Ferson, “Challenge Problems: Uncertainty in System Response Given Uncertain Parameters,” Reliability Engineering and System Safety, Volume 85, pp. 11-19, 2004.

70. T. J. Ross, *Fuzzy Logic with Engineering Applications*, 2nd Edition, John Wiley & Sons, Ltd., 2005.

71. S. A. Dudani, “The Distance-Weighted k-NN Rule,” IEEE Transactions on Systems, Man and Cybernetics, Volume 6, Number 4, pp. 325-327, 1976.

72. R. Haralick and L. Shapiro, *Computer and Robot Vision*, Volume 2, Addison-Wesley, 1993.

73. Tcpdump. URL: http://www.tcpdump.org/ (Last browsed in December 2007)

74. KDD’99 archive: The Fifth International Conference on Knowledge Discovery and Data Mining. URL: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html (Last browsed in December 2007)

75. U. Aickelin, J. Twycross, and T. Hesketh-Roberts, “Rule Generalization in Intrusion Detection Systems Using SNORT,” International Journal of Electronic Security and Digital Forensics, Volume 1, Number 1, pp. 101-116, May 2007.

76. C. L. Blake and C. J. Merz, UCI Repository of Machine Learning Databases, 1998. URL: http://mlearn.ics.uci.edu/MLRepository.html (Last browsed in December 2007)

77. M. Hall, *Correlation Based Feature Selection for Machine Learning*, Doctoral Dissertation, The University of Waikato, Department of Computer Science, 1999.

78. Te-Shun Chou, Kang K. Yen, Jun Luo, Niki Pissinou, and Kia Makki, “Correlation-Based Feature Selection for Intrusion Detection Design,” IEEE Military Communications Conference, Orlando, FL, October 2007.

79. M. Keller, M. R. Gray, and J. A. Givens Jr., “A Fuzzy k-Nearest Neighbor Algorithms,” Transactions on Systems, Man and Cybernetics, Volume SMC-15(4), pp. 580-585, 1985.

80. T. Denoeux, “A k-Nearest Neighbor Classification Rule Based on Dempster-Shafer Theory,” IEEE Transactions on Systems, Man and Cybernetics, Volume 25, Number 5, pp. 804-813, May 1995.

81. Te-Shun Chou, Kang K. Yen, Jun Luo, “Network Intrusion Detection Design Using Feature Selection of Soft Computing Paradigms,” International Journal of Computational Intelligence, Volume 4, Number 3, pp. 205-217, 2007.

82. Te-Shun Chou, Kang K. Yen, Niki Pissinou, and Kia Makki, “Fuzzy Belief Reasoning for Intrusion Detection Design,” IEEE The third International Conference on Intelligent Information Hiding and Multimedia Signal Processing, Kaohsiung, Taiwan, November 2007.

83. Te-Shun Chou, Kang K. Yen, and Jun Luo, “Feature Reduction and Fuzzy Belief Intrusion Detection Design,” The 11th World Multi-Conference on Systemics, Cybernetics and Informatics jointly with The 13th International Conference on Information Systems Analysis and Synthesis**,** pp. 262-267**,** Orlando, FL, July 2007.

84. Te-Shun Chou and Kang K. Yen, “Fuzzy Belief k-Nearest Neighbors Anomaly Detection of User to Root and Remote to Local Attacks,” 8th Annual IEEE SMC Information Assurance Workshop, pp. 207-213, West Point, NY, June 2007.

85. S. Raudys and F. Roli, “The Behavior Knowledge Space Fusion Method: Analysis of Generalization Error and Strategies for Performance Improvement,” Proceedings of International Workshop on Multiple Classifier Systems, pp. 55-64, Guildford, Surrey, June 2003.

86. Te-Shun Chou, Kang K. Yen, Niki Pissinou, and Kia Makki, “Ensemble of Multiple Classifiers in Network Intrusion Detection Design,” Computers & Security, 2007. (in review)

APPENDIX

Matlab Code List of Fuzzy Belief *k*-NN Classifier Evaluation

som_read_data.m

```
% Read data from an ascii file
function sData = som_read_data(filename, varargin)
error(nargchk(1, 3, nargin))          % check no. of input args is correct
dont_care = 'NaN';                    % default don't care string
comment_start   = '#';                % the char a SOM_PAK command line starts with
comp_name_line  = '#n';               % string denoting a special command line,
                                      % which contains names of each component
label_name_line = '#l';               % string denoting a special command line,
                                      % which contains names of each label
block_size = 1000;                    % block size used in file read
kludge = num2str(realmax, 100);       % used in sscanf
% open input file
fid = fopen(filename);
if fid < 0
  error(['Cannot open ' filename]);
end
% process input arguments
if nargin == 2
  if isstr(varargin{1})
    dont_care = varargin{1};
  else
    dim = varargin{1};
  end
elseif nargin == 3
  dim = varargin{1};
  dont_care = varargin{2};
end
 % if the data dimension is not specified, find out what it is
 if nargin == 1 | (nargin == 2 & isstr(varargin{1}))
   fpos1 = ftell(fid); c1 = 0;   % read first non-comment line
  while c1 == 0,
    line1 = strrep(fgetl(fid), dont_care, kludge);
    [l1, c1] = sscanf(line1, '%f ');
  end
  fpos2 = ftell(fid); c2 = 0;   % read second non-comment line
  while c2 == 0,
    line2 = strrep(fgetl(fid), dont_care, kludge);
    [l2, c2] = sscanf(line2, '%f ');
  end
  if (c1 == 1 & c2 ~= 1) | (c1 == c2 & c1 == 1 & l1 == 1)
    dim = l1;
    fseek(fid, fpos2, -1);
  elseif (c1 == c2)
    dim = c1;
    fseek(fid, fpos1, -1);
    warning on
    warning(['Automatically determined data dimension is ' num2str(dim) '. Is it correct?']);
  else
    error(['Invalid header line: ' line1]);
  end
end
% check the dimension is valid
if dim < 1 | dim ~= round(dim)
```

```
  error(['Illegal data dimension: ' num2str(dim)]);
end
% read data
sData = som_data_struct(zeros(1, dim), 'name', filename);
lnum = 0;   % data vector counter
data_temp = zeros(block_size, dim);
labs_temp = cell(block_size, 1);
comp_names = sData.comp_names;
label_names = sData.label_names;
form = [repmat('%g',[1 dim-1]) '%g%[^ \t]'];
limit = block_size;
while 1,
  li = fgetl(fid);   % read next line
  if ~isstr(li), break, end;   % is this the end of file?
  % all missing vectors are replaced by value realmax because
  % sscanf is not able to read NaNs
  li = strrep(li, dont_care, kludge);
  [data, c, err, n] = sscanf(li, form);
  if c < dim   % if there were less numbers than dim on the input file line
    if c == 0
      if strncmp(li, comp_name_line, 2)   % component name line?
    li = strrep(li(3:end), kludge, dont_care); i = 0; c = 1;
    while c
      [s, c, e, n] = sscanf(li, '%s%[^ \t]');
      if ~isempty(s), i = i + 1; comp_names{i} = s; li = li(n:end); end
    end
    if i ~= dim
      error(['Illegal number of component names: ' num2str(i) ...
             ' (dimension is ' num2str(dim) ')']);
    end
      elseif strncmp(li, label_name_line, 2)   % label name line?
    li = strrep(li(3:end), kludge, dont_care); i = 0; c = 1;
    while c
      [s, c, e, n] = sscanf(li, '%s%[^ \t]');
      if ~isempty(s), i = i + 1; label_names{i} = s; li = li(n:end); end
    end
      elseif ~strncmp(li, comment_start, 1)   % not a comment, is it error?
    [s, c, e, n] = sscanf(li, '%s%[^ \t]');
    if c
      error(['Invalid vector on input file data line ' ...
        num2str(lnum+1) ': [' deblank(li) ']']),
    end
      end
    else
      error(['Only ' num2str(c) ' vector components on input file data line ' ...
        num2str(lnum+1) ' (dimension is ' num2str(dim) ')']);
    end
  else
    lnum = lnum + 1;   % this was a line containing data vector
    data_temp(lnum, 1:dim) = data';   % add data to struct
    if lnum == limit   % reserve more memory if necessary
      data_temp(lnum+1:lnum+block_size, 1:dim) = zeros(block_size, dim);
      [dummy nl] = size(labs_temp);
      labs_temp(lnum+1:lnum+block_size,1:nl) = cell(block_size, nl);
```

```
    limit = limit + block_size;
   end
   % read labels
   if n < length(li)
    li = strrep(li(n:end), kludge, dont_care); i = 0; n = 1; c = 1;
    while c
   [s, c, e, n_new] = sscanf(li(n:end), '%s%[^ \t]');
   if c, i = i + 1; labs_temp{lnum, i} = s; n = n + n_new - 1; end
    end
   end
  end
end
% close input file
if fclose(fid) < 0, error(['Cannot close file ' filename]);
else fprintf(2, '\rdata read ok\n'); end
% set values
data_temp(data_temp == realmax) = NaN;
sData.data        = data_temp(1:lnum,:);
sData.labels      = labs_temp(1:lnum,:);
sData.comp_names  = comp_names;
sData.label_names = label_names;
return;
```

dataprocess.m

```
% Separate data to normal, dos, probe, u2r and r2l categories
function [all_normal_data, all_normal_labels, ...
            all_dos_data, all_dos_labels, ...
            all_probe_data, all_probe_labels, ...
            all_u2r_data, all_u2r_labels, ...
            all_r2l_data, all_r2l_labels] = dataprocess(data, labels)
% label has three groups
% type: different attacks
% category: 0:normal, 1:dos, 2:probe, 3:r2l ,4:u2r
% binary -- 0:normal, 1:attack
% find records in each category from set
normal = find(strcmp(labels(:,2), '0'));
dos = find(strcmp(labels(:,2), '1'));
probe = find(strcmp(labels(:,2), '2'));
u2r = find(strcmp(labels(:,2), '4'));
r2l = find(strcmp(labels(:,2), '3'));
% all records in each category -- data
all_normal_data = data(normal(1:size(normal)),:);
all_dos_data = data(dos(1:size(dos)),:);
all_probe_data = data(probe(1:size(probe)),:);
all_u2r_data = data(u2r(1:size(u2r)),:);
all_r2l_data = data(r2l(1:size(r2l)),:);
% all records in each category -- label
all_normal_labels = labels(normal(1:size(normal)),:);
all_dos_labels = labels(dos(1:size(dos)),:);
all_probe_labels = labels(probe(1:size(probe)),:);
all_u2r_labels = labels(u2r(1:size(u2r)),:);
all_r2l_labels = labels(r2l(1:size(r2l)),:);
```

transfer.m

```
% Randomly select records for normal, dos, probe, u2r and r2l category
function [transfer_data,transfer_labels,left_data,left_labels] = transfer(A,B,fetch_number)
% A, B: original data set
% fetch_number: number of select records
FETCH = zeros(1,fetch_number);
zero_array = find(FETCH==0);
while ~isempty(zero_array)
  rand('state', sum(100*clock));
  a = ceil(rand(1)*size(A,1));
  origial_array = find(FETCH==a);
  while ~isempty(origial_array)
      a = ceil(rand(1)*size(A,1));
      origial_array = find(FETCH==a);
  end
  FETCH(zero_array(1)) = a;
  zero_array = find(FETCH==0);
end
transfer_data = A(FETCH,:);
transfer_labels = B(FETCH,:);
index_A = 1:size(A,1);
index_B = 1:size(B,1);
left_data = A(setdiff(index_A,FETCH),:);
left_labels = B(setdiff(index_B,FETCH),:);
```

clust_normalize.m

```
% Normalization of features
function data=clust_normalize(data,method);
% method can be 'var'or 'range'
%   'var'      Variance is normalized to one (linear operation).
%   'range'  Values are normalized between [0,1] (linear operation).
data.Xold=data.X;
if strcmp(method,'range')
    data.min=min(data.X);
    data.max=max(data.X);
    array = (repmat(max(data.X),...
    size(data.X,1),1)-repmat(min(data.X),size(data.X,1),1));
    index = find(array==0);
    array(index) = 1;
    data.X=(data.X-repmat(min(data.X),size(data.X,1),1))./array;
elseif strcmp(method,'var')
   array = (repmat(std(data.X),size(data.X,1),1));
   index = find(array==0);
   array(index) = 1;
   data.X=(data.X-repmat(mean(data.X),size(data.X,1),1))./array;
   data.mean=mean(data.X);
   data.std=std(data.X);
else
    error('Unknown method given')
end
```

fselection.m

```
% Feature selection
function [traindata, testdata] = fs(data1, data2, f_s, w)
```

```
% f_s = 1: yes
% w: feature set
j = 1;
if f_s
  for i = 1:41
    if w(i) ~= 0
      traindata(:,j) = w(i).*data1(:,i);
      testdata(:,j) = w(i).*data2(:,i);
      j = j + 1;
    end
  end
else
   traindata = data1;
   testdata = data2;
end
```

celltonum.m

```
% Convert cell to number
function [train_labels_num, test_labels_num] = celltonum(train_labels, test_labels)
train_labels_num = zeros(size(train_labels,1),size(train_labels,2));
for i=1:1:size(train_labels,1)
   for j=1:1:size(train_labels,2)
      temp = cell2mat(train_labels(i,j));
      train_labels_num(i,j) = str2num(temp);
   end
end
% to add 1 because the class of fknn needs to start from 1, but the class
% label of train.labels starts from 0. same as test.labels
train_labels_num = train_labels_num+1;
test_labels_num = zeros(size(test_labels,1),size(test_labels,2));
for i=1:1:size(test_labels,1)
   for j=1:1:size(test_labels,2)
      temp = cell2mat(test_labels(i,j));
      test_labels_num(i,j) = str2num(temp);
   end
end
test_labels_num = test_labels_num+1;
```

fcm_order_binary.m

```
% For FCM to find clustering order (final_perms)
function [finalperms, fcm_labels] = fcm_order_binary(U, labels)
% permutation (all combinations) of classes
class = cell([2,1]);
class{1}='0';
class{2}='1';
fcmcluster = perms(class);
temp_current_no = 0;
final_perms = 0; % 1xN where N = no of class, e.g. 0 1
correct_no = 0;
for (j = 1:1:size(fcmcluster,1))
    temp_perms = fcmcluster(j,:);
    for (i=1:1:size(U,2))
        maxU = max(U(:,i));
        temp = U(:,i);
```

```
        index = find(temp == maxU);
        temp_fcmlabels(i) = temp_perms(index);
    end
    temp_correct = find(strcmp(temp_fcmlabels(:),labels(:,2)));
    temp_correct_no = length(temp_correct);
    if temp_correct_no > correct_no
      correct_no = temp_correct_no;
      final_perms = temp_perms;
      fcmlabels = temp_fcmlabels;
    end
end
for i = 1:1:size(final_perms,2)
   temp = cell2mat(final_perms(i));
   finalperms(i) = str2num(temp)+1;
end
for i = 1:1:size(fcmlabels,2)
   temp = cell2mat(fcmlabels(i));
   fcm_labels(i) = str2num(temp)+1;
end
```

fknn.m

```
% Fuzzy k-nearest neighbor classification algorithm
function [predicted,memberships, numhits] = fknn(data, labels, test, ...
            testlabels, k_values, info, fuzzy)
if nargin<7
   fuzzy = true;
end
num_train = size(data,1);
num_test  = size(test,1);
% scaling factor for fuzzy weights. see [1] for details
m = 2;
% convert class labels to unary membership vectors (of 1s and 0s)
max_class = max(labels); %original
temp = zeros(length(labels),max_class);
for i=1:num_train
   temp(i,:) = [zeros(1, labels(i)-1) 1 zeros(1,max_class - labels(i))];
end
labels = temp;
clear temp;
% allocate space for storing predicted labels
predicted = zeros(num_test, length(k_values));
% allocate space for 'numhits'. This will only be used if 'testlabels' is provided
numhits = zeros(length(k_values),1);
% will the memberships be stored? if yes, allocate space
store_memberships = false;
if nargout > 1,
   store_memberships=true;
   memberships = zeros(num_test, max_class, length(k_values));
end
% BEGIN kNN
% for each test point, do:
t0=clock;
tstart = t0;
for i=1:num_test
```

```
    distances = (repmat(test(i,:), num_train,1) - data).^2;
    % for efficiency, no need to take sqrt since it is a non-decreasing function
    if size(distances,2) ~= 1 % add
       distances = sum(distances');
    else %add
       distances = distances'; % add
    end % add
    % sort the distances
    [junk, indeces] = sort(distances);
    for k=1:length(k_values)
    neighbor_index = indeces(1:k_values(k));
    weight = ones(1,length(neighbor_index));
    if fuzzy,
       % originally, this weight calculation should be:
       % weight = distances(neighbor_index).^(-2/(m-1));
       % but since we didn't take sqrt above and the inverse 2th power
       % the weights are:
       % weight = sqrt(distances(neighbor_index)).^(-2/(m-1));
       % which is equaliavent to:
       weight = distances(neighbor_index).^(-1/(m-1));
       % set the Inf (infite) weights, if there are any, to  1.
       if max(isinf(weight))
       warning(['Some of the weights are Inf for sample: ' ...
          num2str(i) '. These weights are set to 1.']);
       weight(isinf(weight))=1;
       end
    end
    test_out = weight*labels(neighbor_index,:)/(sum(weight));
    if store_memberships, memberships(i,:,k) = test_out; end;
    % find predicted class (the one with the max. fuzzy vote)
    [junk, index_of_max] = max(test_out');
    predicted(i,k) = index_of_max;
    % compute current hit rate, if test labels are given
    if ~isempty(testlabels) && predicted(i,k)==testlabels(i)
       numhits(k) = numhits(k)+1;
    end
    end
    % print info
    if mod(i,info)==0
    elapsed = etime(clock, t0);
    fprintf(1,['%dth sample done.  Elapsed (from previous info): %.2f' ...
                ' sn.  Estimated left: %.2f sn.\n\tHit rate(s) so far:   '], ...
       i, elapsed, etime(clock, tstart)*((num_test-i)/i) );
    for k=1:length(k_values)
       fprintf(1,'%3d: %.3f\t',k_values(k), 100*numhits(k)/i);
    end
    fprintf(1,'\n');
    t0=clock; % start timer again
    end
end
```

knndsinit.m

```
% Initialise parameter gamma and alpha of the BPA
% Modifiy from Thierry Denoeux, http://www.hds.utc.fr/~tdenoeux/software.htm
```

```
% Last browsed in December 2007
function [gamm,alpha] = knndsinit1(x,S);
 [Napp,nent]=size(x);
M=max(S);
for i=1:M,
  ii=find(S==i);Nii=length(ii);
  D=zeros(1,Nii);
  for j=1:Nii
   D(1,j) = D(1,j) + sum(sqrt(sum(((ones(Nii,1)*x(ii(j),:))-x(ii,:))'.^2)'));
  end;
Dm(i) = sum(D)/(Nii*Nii - Nii);
end;
gamm = ones(1,M) ./ Dm;
gamm=gamm';
alpha=.95;
```

knndsval.m

```
% K-nearest neighbour classification rule based on Dempster-Shafer theory
% Modifiy from Thierry Denoeux, http://www.hds.utc.fr/~tdenoeux/software.htm
% Last browsed in December 2007
function [m,L] = knndsval(xapp,Sapp,K,gamm,alpha,loo,xtst);
 [Napp,nent]=size(xapp);
M=max(Sapp); % original
if loo,
   xtst=xapp;
end;
 [Ntst,nent]=size(xtst);
% Computation of the K-nearest neighbors in the training set
  dst=[];ist=[];
  for i = 1:Ntst,
    if size(xapp,2) ~= 1 % add
       dist=sum(((ones(Napp,1)*xtst(i,:))-xapp)'.^2)';
    else % add
       dist=(((ones(Napp,1)*xtst(i,:))-xapp)'.^2)'; %add
    end % add
    [dss,iss]=sort(dist);
    dst = [dst dss(1+loo:K+loo)]; % distance matrix: MxN  where M=k, N=number of testing records
                                  % each testing record has M nearest neighbors
    ist = [ist iss(1+loo:K+loo)]; % index matrix: MxN
                                  % each testing record has M nearest training record number
  end;
% Computation of the BPA
m = classdstst(alpha,gamm,xtst,dst,ist,Sapp,K);
[temp,L]=max(m(:,1:M)');
L=L';
function m = classdstst(alpha,gamm,xtst,ds,is,Sapp,K);
N= max(size(xtst));
M=max(Sapp); % original
m = [zeros(M,N);ones(1,N)];
cppv=zeros(N,1);
for i=1:N,
   for j=1:K,
     m1 = zeros(M+1,1);
     m1(Sapp(is(j,i))) = alpha*exp(-gamm(Sapp(is(j,i))).^2*ds(j,i));
```

```
    m1(M+1) = 1 - m1(Sapp(is(j,i)));
    m(1:M,i) = m1(1:M).*m(1:M,i) + m1(1:M)*m(M+1,i) + m(1:M,i)*m1(M+1);
    m(M+1,i) = m1(M+1) * m(M+1,i);
  end;
end;
m=m./(ones(M+1,1)*sum(m));
m=m';
```

myknn_binary.m

```
% Fuzzy Belief k-NN Classification Algorithm
% Modifiy from Thierry Denoeux, http://www.hds.utc.fr/~tdenoeux/software.htm
% Last browsed in December 2007
function [m,L1] = myknn(xapp,Sapp,K,U,loo,xtst,finalperms);
 [Napp,nent]=size(xapp);
if loo,
   xtst=xapp;
end;
 [Ntst,nent]=size(xtst);
% Computation of the K-nearest neighbours in the training set
  dst=[];ist=[];
  for i = 1:Ntst,
    if size(xapp,2) ~= 1 % add
       dist=sum(((ones(Napp,1)*xtst(i,:))-xapp)'.^2)';
    else % add
       dist=(((ones(Napp,1)*xtst(i,:))-xapp)'.^2)'; %add
    end % add
    [dss,iss]=sort(dist);
    dst = [dst dss(1+loo:K+loo)]; % distance matrix: MxN  where M=k, N=number of testing records
                                     % each testing record has M nearest neighbors
    ist = [ist iss(1+loo:K+loo)]; % index matrix: MxN
                                    % each testing record has M nearest training record number
  end;
N= max(size(xtst));
% M=max(Sapp);
M=2; % binay classification
m = [zeros(M,N);ones(1,N)];
cppv=zeros(N,1);
% distance weighted
dw(K,N) = 0;
for i = 1:N
    for j = 1:K
        if dst(j,i) == max(dst(:,i))
            dw(j,i) = 1;
        else
            dw(j,i) = (max(dst(:,i))-dst(j,i))/(max(dst(:,i))-min(dst(:,i)));
        end
    end
end
for i=1:N
    num=ones(M+1,1);
    for j=1:K
          m1 = zeros(M+1,1);
          m1(1:M) = U(1:M,ist(j,i))*dw(j,i);
          m1(M+1) = 1 - sum(m1);
```

```
        m(1:M,i) = m1(1:M).*m(1:M,i)+ m1(1:M)*m(M+1,i) + m(1:M,i)*m1(M+1);
        m(M+1,i) = m1(M+1) * m(M+1,i);
    end
end
m(3,:) = 1-m(1,:)-m(2,:);
m=m./(ones(M+1,1)*sum(m));
m=m';
 [temp,L]=max(m(:,1:M)');
L=L';
L1=zeros(size(L,1),1);
for i = 1:1:size(L,1)
    if L(i)==1
        L1(i)=finalperms(1);
    elseif L(i)==2
        L1(i)=finalperms(2);
    end
end
```

weight.m

```
% Weighting for myknn_binary_w.m
function w = weight(data,labels,m)
for i=1:size(data,1)
    if labels(i,2) == 1
       S(i,1) = 1;
       S(i,2) = 0;
    else
       S(i,1) = 0;
       S(i,2) = 1;
    end
end
w = ones(size(data,1),3);
for i = 1:size(data,1)-1
    w(i,1:2) = 2*(m(i,1:2).*w(i,1:2)-S(i,1:2)).*m(i,1:2);
    w(i+1,1:2) = w(i+1,1:2)+0.1*w(i,1:2);
end
w = w(size(w,1),:);
```

myknn_binary_w.m

```
% weighted fb_knn
% Modifiy from Thierry Denoeux, http://www.hds.utc.fr/~tdenoeux/software.htm
% Last browsed in December 2007
function [m,L1] = myknn(xapp,Sapp,K,U,loo,xtst,finalperms,w);
 [Napp,nent]=size(xapp);
if loo,
   xtst=xapp;
end;
 [Ntst,nent]=size(xtst);
% Computation of the K-nearest neighbours in the training set
 dst=[];ist=[];
  for i = 1:Ntst,
    if size(xapp,2) ~= 1 % add
       dist=sum(((ones(Napp,1)*xtst(i,:))-xapp)'.^2)';
    else % add
       dist=(((ones(Napp,1)*xtst(i,:))-xapp)'.^2)'; % add
```

```
    end % add
    [dss,iss]=sort(dist);
    dst = [dst dss(1+loo:K+loo)]; % distance matrix: MxN  where M=k, N=number of testing records
                                    % each testing record has M nearest neighbors
    ist = [ist iss(1+loo:K+loo)]; % index matrix: MxN
                                   % each testing record has M nearest training record number
  end;
N= max(size(xtst));
M=2; % binay classification
m = [zeros(M,N);ones(1,N)];
cppv=zeros(N,1);
% distance weighted
dw(K,N) = 0;
for i = 1:N
    for j = 1:K
        if dst(j,i) == max(dst(:,i))
            dw(j,i) = 1;
        else
            dw(j,i) = (max(dst(:,i))-dst(j,i))/(max(dst(:,i))-min(dst(:,i)));
        end
    end
end
for i=1:N
    num=ones(M+1,1);
    for j=1:K
          m1 = zeros(M+1,1);
          m1(1:M) = U(1:M,ist(j,i))*dw(j,i);
          m1(M+1) = 1 - sum(m1);
          m(1:M,i) = m1(1:M).*m(1:M,i)+ m1(1:M).*m(M+1,i) + m(1:M,i).*m1(M+1);
          m(M+1,i) = m1(M+1) * m(M+1,i);
     end
end
m(3,:) = 1-m(1,:)-m(2,:);
m=m./(ones(M+1,1)*sum(m));
m=m';
for i=1:N
    m(i,1:M+1)=m(i,1:M+1).*w;
end
m(3,:) = 1-m(1,:)-m(2,:);
 [temp,L]=max(m(:,1:M)');
L=L';
L1=zeros(size(L,1),1);
for i = 1:1:size(L,1)
    if L(i)==1
        L1(i)=finalperms(1);
    elseif L(i)==2
        L1(i)=finalperms(2);
    end
end
```

accuracy_binary.m

```
% Detection performance
function [correct_normal_rate, correct_attack_rate] = ...
    accuracy_binary(labels_num, predicted, normal_record_no, attack_record_no, attack_index, attack_label)
```

```
% normal
correct_normal = find(labels_num(:,2)==1 & predicted(:)==1);
correct_normal_no = length(correct_normal);
correct_normal_rate = correct_normal_no / normal_record_no;
n_a = find(labels_num(:,2)==1 & predicted(:)==attack_label);
n_a_no = length(n_a);
% attack
correct_attack = find(labels_num(:,2)==attack_index & predicted(:)==attack_label);
correct_attack_no = length(correct_attack);
correct_attack_rate = correct_attack_no / attack_record_no;
a_n = find(labels_num(:,2)==attack_index & predicted(:)==1);
a_n_no = length(a_n);
fprintf('\n');
```

ensemble_mv.m

```
% Ensemble using majority voting
function [e_result, predict] = ensemble_mv(A_predicted, B_predicted, C_predicted, test_label)
predict = [A_predicted B_predicted C_predicted test_label(:,3)];
e_result = zeros(size(A_predicted));
for i=1:size(e_result)
   if predict(i,1) == 1 && predict(i,2) == 1 && predict(i,3) == 1
     e_result(i) = 1;
   elseif predict(i,1) == 2 && predict(i,2) == 2 && predict(i,3) == 2
     e_result(i) = 2;
   elseif predict(i,1) == 1 && predict(i,2) == 1 && predict(i,3) == 2
     e_result(i) = 1;
   elseif predict(i,1) == 1 && predict(i,2) == 2 && predict(i,3) == 1
     e_result(i) = 1;
   elseif predict(i,1) == 2 && predict(i,2) == 1 && predict(i,3) == 1
     e_result(i) = 1;
   elseif predict(i,1) == 2 && predict(i,2) == 2 && predict(i,3) == 1
     e_result(i) = 2;
   elseif predict(i,1) == 2 && predict(i,2) == 1 && predict(i,3) == 2
     e_result(i) = 2;
   elseif predict(i,1) == 1 && predict(i,2) == 2 && predict(i,3) == 2
     e_result(i) = 2;
   end
end
predict = [predict e_result];
```

ensemble_avg.m

```
% Selects average value of the combined classifiers as the ensemble output
function [m, e_result, predict] = ensemble_avg(A_memberships, A1, A2, index, ...
                                               B_memberships, B1, B2, ...
                                               C_memberships, C1, C2, ...
                                               D_memberships, D1, D2, ...
                                               test_label)
e_memberships = zeros(size(A_memberships),2);
e_result = zeros(size(e_memberships),1);
A_memberships = [A_memberships(:,1) A_memberships(:,2) A_memberships(:,3)]';
B_memberships = [B_memberships(:,1) B_memberships(:,2) B_memberships(:,3)]';
C_memberships = [C_memberships(:,1) C_memberships(:,2) C_memberships(:,3)]';
D_memberships = [D_memberships(:,1) D_memberships(:,2) D_memberships(:,3)]';
M=2;
```

```
N = size(e_result);
m = [zeros(M,N);ones(1,N)];
for i=1:N
   m(1,i) = A_memberships(1,i)*A1+B_memberships(1,i)*B1+…
            C_memberships(1,i)*C1+D_memberships(1,i)*D1;
   m(2,i) = A_memberships(2,i)*A2+B_memberships(2,i)*B2+…
            C_memberships(2,i)*C2+D_memberships(2,i)*D2;
end
m=m./(ones(M+1,1)*sum(m));
for i=1:size(e_result)
   if (m(1,i) - m(2,i)) > 0
         e_result(i) = 1;
   else
         e_result(i) = 2;
   end
end
predict = e_result;
```

ensemble_m.m

```
% Ensemble using Dempster-Shafer theory
function [m, e_result, predict] = ensemble_m(A_memberships, A_predicted, index, ...
                                             B_memberships, B_predicted, ...
                                             C_memberships, C_predicted, ...
                                             D_memberships, D_predicted, ...
                                             test_label)
predict = [A_predicted B_predicted C_predicted D_predicted test_label(:,3)];
e_memberships = zeros(size(A_memberships),2);
e_result = zeros(size(e_memberships),1);
A_memberships = [A_memberships(:,1) A_memberships(:,2) A_memberships(:,3)]';
B_memberships = [B_memberships(:,1) B_memberships(:,2) B_memberships(:,3)]';
C_memberships = [C_memberships(:,1) C_memberships(:,2) C_memberships(:,3)]';
D_memberships = [D_memberships(:,1) D_memberships(:,2) D_memberships(:,3)]';
M=2;
N = size(e_result);
m = [zeros(M,N);ones(1,N)];
for i=1:N
   num=ones(M+1,1);
   for j=1:4
        m1 = zeros(M+1,1);
        if j==1
          m1(1:M) = A_memberships(1:M,i);
        elseif j==2
          m1(1:M) = B_memberships(1:M,i);
        elseif j==3
          m1(1:M) = C_memberships(1:M,i);
        elseif j==4
          m1(1:M) = D_memberships(1:M,i);
        end
        m1(M+1) = 1 - sum(m1);
        m(1:M,i) = m1(1:M).*m(1:M,i)+ m1(1:M)*m(M+1,i) + m(1:M,i)*m1(M+1);
        m(M+1,i) = m1(M+1) * m(M+1,i);
    end
end
m(3,:) = 1-m(1,:)-m(2,:);
```

```
m=m./(ones(M+1,1)*sum(m));
for i=1:size(e_result)
   if m(1,i) >= m(2,i)
      e_result(i) = 1;
   else
      e_result(i) = 2;
   end
end
predict = [predict e_result];
```

ensemble4.m

```
% Ensemble using naive bayes
function [result] = ensemble4(A1, A2, A_p, B1, B2, B_p, C1, C2, C_p, D1, D2, D_p, P1, P2)
predict = [A_p B_p C_p D_p];
P_1 = zeros(size(A_p));
P_2 = zeros(size(A_p));
for i = 1:size(A_p)
   if predict(i,:) == [1 1 1 1]
         P_1(i) = A1*B1*C1*D1*P1 / (A1*B1*C1*D1*P1 + (1-A2)*(1-B2)*(1-C2)*(1-D2)*P2);
   elseif predict(i,1) == [2 2 2 2]
         P_1(i) = (1-A1)*(1-B1)*(1-C1)*(1-D1)*P1 / ((1-A1)*(1-B1)*(1-C1)*(1-D1)*P1 +
A2*B2*C2*D2*P2);
   elseif predict(i,1) == [1 1 1 2]
         P_1(i) = A1*B1*C1*(1-D1)*P1 / (A1*B1*C1*(1-D1)*P1 + (1-A2)*(1-B2)*(1-C2)*D2*P2);
   elseif predict(i,1) == [1 1 2 1]
         P_1(i) = A1*B1*(1-C1)*D1*P1 / (A1*B1*(1-C1)*D1*P1 + (1-A2)*(1-B2)*C2*(1-D2)*P2);
   elseif predict(i,1) == [1 1 2 2]
         P_1(i) = A1*B1*(1-C1)*(1-D1)*P1 / (A1*B1*(1-C1)*(1-D1)*P1 + (1-A2)*(1-B2)*C2*D2*P2);
   elseif predict(i,:) == [1 2 1 1]
         P_1(i) = A1*(1-B1)*C1*D1*P1 / (A1*(1-B1)*C1*D1*P1 + (1-A2)*B2*(1-C2)*(1-D2)*P2);
   elseif predict(i,1) == [1 2 1 2]
         P_1(i) = A1*(1-B1)*C1*(1-D1)*P1 / (A1*(1-B1)*C1*(1-D1)*P1 + (1-A2)*B2*(1-C2)*D2*P2);
   elseif predict(i,:) == [1 2 2 1]
         P_1(i) = A1*(1-B1)*(1-C1)*D1*P1 / (A1*(1-B1)*(1-C1)*D1*P1 + (1-A2)*B2*C2*(1-D2)*P2);
   elseif predict(i,1) == [1 2 2 2]
         P_1(i) = A1*(1-B1)*(1-C1)*(1-D1)*P1 / (A1*(1-B1)*(1-C1)*(1-D1)*P1 + (1-A2)*B2*C2*D2*P2);
   elseif predict(i,:) == [2 1 1 1]
         P_1(i) = (1-A1)*B1*C1*D1*P1 / ((1-A1)*B1*C1*D1*P1 + A2*(1-B2)*(1-C2)*(1-D2)*P2);
   elseif predict(i,:) == [2 1 1 2]
         P_1(i) = (1-A1)*B1*C1*(1-D1)*P1 / ((1-A1)*B1*C1*(1-D1)*P1 + A2*(1-B2)*(1-C2)*D2*P2);
   elseif predict(i,:) == [2 1 2 1]
         P_1(i) = (1-A1)*B1*(1-C1)*D1*P1 / ((1-A1)*B1*(1-C1)*D1*P1 + A2*(1-B2)*C2*(1-D2)*P2);
   elseif predict(i,:) == [2 1 2 2]
         P_1(i) = (1-A1)*B1*(1-C1)*(1-D1)*P1 / ((1-A1)*B1*(1-C1)*(1-D1)*P1 + A2*(1-B2)*C2*D2*P2);
   elseif predict(i,:) == [2 2 1 1]
         P_1(i) = (1-A1)*(1-B1)*C1*D1*P1 / ((1-A1)*(1-B1)*C1*D1*P1 + A2*B2*(1-C2)*(1-D2)*P2);
   elseif predict(i,:) == [2 2 1 2]
         P_1(i) = (1-A1)*(1-B1)*C1*(1-D1)*P1 / ((1-A1)*(1-B1)*C1*(1-D1)*P1 + A2*B2*(1-C2)*D2*P2);
    elseif predict(i,:) == [2 2 2 1]
         P_1(i) = (1-A1)*(1-B1)*(1-C1)*D1*P1 / ((1-A1)*(1-B1)*(1-C1)*D1*P1 + A2*B2*C2*(1-D2)*P2);
   end
   P_2(i) = 1-P_1(i);
end
for i = 1:size(A_p)
```

```
    if P_1(i) >= P_2(i)
        result(i) = 1;
    else
        result(i) = 2;
    end
end
```

attack_train_binary.m

```
% The number of each attack in training set
function [ train_3_no, train_4_no, train_5_no,
           train_6_no,  train_7_no, train_8_no, train_9_no, train_10_no,   ...
           train_11_no, train_12_no, train_13_no, train_14_no, train_15_no,   ...
           train_16_no, train_17_no, train_18_no, train_19_no, train_20_no,   ...
           train_21_no, train_22_no, train_23_no, train_24_no, train_25_no,   ...
           train_26_no, train_27_no, train_28_no, train_29_no, train_30_no,   ...
           train_31_no, train_32_no, train_33_no, train_34_no, train_35_no,   ...
           train_36_no, train_37_no, train_38_no, train_39_no, train_40_no, train_41_no] =  ...
           attack_train_binary(train_labels_num)
% DOS
train_3_record = find(train_labels_num(:,1)==3);
train_3_no = length(train_3_record);
train_9_record = find(train_labels_num(:,1)==9);
train_9_no = length(train_9_record);
train_12_record = find(train_labels_num(:,1)==12);
train_12_no = length(train_12_record);
train_16_record = find(train_labels_num(:,1)==16);
train_16_no = length(train_16_record);
train_20_record = find(train_labels_num(:,1)==20);
train_20_no = length(train_20_record);
train_22_record = find(train_labels_num(:,1)==22);
train_22_no = length(train_22_record);
train_27_record = find(train_labels_num(:,1)==27);
train_27_no = length(train_27_record);
train_25_record = find(train_labels_num(:,1)==25);
train_25_no = length(train_25_record);
train_30_record = find(train_labels_num(:,1)==30);
train_30_no = length(train_30_record);
train_37_record = find(train_labels_num(:,1)==37);
train_37_no = length(train_37_record);
% PROBE
train_8_record = find(train_labels_num(:,1)==8);
train_8_no = length(train_8_record);
train_13_record = find(train_labels_num(:,1)==13);
train_13_no = length(train_13_record);
train_17_record = find(train_labels_num(:,1)==17);
train_17_no = length(train_17_record);
train_19_record = find(train_labels_num(:,1)==19);
train_19_no = length(train_19_record);
train_28_record = find(train_labels_num(:,1)==28);
train_28_no = length(train_28_record);
train_32_record = find(train_labels_num(:,1)==32);
train_32_no = length(train_32_record);
% R2L
```

```
train_5_record = find(train_labels_num(:,1)==5);
train_5_no = length(train_5_record);
train_6_record = find(train_labels_num(:,1)==6);
train_6_no = length(train_6_record);
train_7_record = find(train_labels_num(:,1)==7);
train_7_no = length(train_7_record);
train_11_record = find(train_labels_num(:,1)==11);
train_11_no = length(train_11_record);
train_15_record = find(train_labels_num(:,1)==15);
train_15_no = length(train_15_record);
train_21_record = find(train_labels_num(:,1)==21);
train_21_no = length(train_21_record);
train_23_record = find(train_labels_num(:,1)==23);
train_23_no = length(train_23_record);
train_24_record = find(train_labels_num(:,1)==24);
train_24_no = length(train_24_record);
train_29_record = find(train_labels_num(:,1)==29);
train_29_no = length(train_29_record);
train_33_record = find(train_labels_num(:,1)==33);
train_33_no = length(train_33_record);
train_34_record = find(train_labels_num(:,1)==34);
train_34_no = length(train_34_record);
train_35_record = find(train_labels_num(:,1)==35);
train_35_no = length(train_35_record);
train_38_record = find(train_labels_num(:,1)==38);
train_38_no = length(train_38_record);
train_39_record = find(train_labels_num(:,1)==39);
train_39_no = length(train_39_record);
train_40_record = find(train_labels_num(:,1)==40);
train_40_no = length(train_40_record);
% U2R
train_4_record = find(train_labels_num(:,1)==4);
train_4_no = length(train_4_record);
train_10_record = find(train_labels_num(:,1)==10);
train_10_no = length(train_10_record);
train_14_record = find(train_labels_num(:,1)==14);
train_14_no = length(train_14_record);
train_18_record = find(train_labels_num(:,1)==18);
train_18_no = length(train_18_record);
train_26_record = find(train_labels_num(:,1)==26);
train_26_no = length(train_26_record);
train_31_record = find(train_labels_num(:,1)==31);
train_31_no = length(train_31_record);
train_36_record = find(train_labels_num(:,1)==36);
train_36_no = length(train_36_record);
train_41_record = find(train_labels_num(:,1)==41);
train_41_no = length(train_41_record);
```

attack_test_binary.m

```
% The correct predict attack number in testing set
function [ total_3_no,  correct_3_no, total_4_no,  correct_4_no, total_5_no,  correct_5_no,  ...
           total_6_no,  correct_6_no, total_7_no,  correct_7_no, total_8_no,  correct_8_no,  ...
           total_9_no,  correct_9_no, total_10_no, correct_10_no, total_11_no, correct_11_no,  ...
           total_12_no, correct_12_no, total_13_no, correct_13_no, total_14_no, correct_14_no,  ...
```

```
            total_15_no, correct_15_no, total_16_no, correct_16_no, total_17_no, correct_17_no,  ...
            total_18_no, correct_18_no, total_19_no, correct_19_no, total_20_no, correct_20_no,  ...
            total_21_no, correct_21_no, total_22_no, correct_22_no, total_23_no, correct_23_no,  ...
            total_24_no, correct_24_no, total_25_no, correct_25_no, total_26_no, correct_26_no,  ...
            total_27_no, correct_27_no, total_28_no, correct_28_no, total_29_no, correct_29_no,  ...
            total_30_no, correct_30_no, total_31_no, correct_31_no, total_32_no, correct_32_no,  ...
            total_33_no, correct_33_no, total_34_no, correct_34_no, total_35_no, correct_35_no,  ...
            total_36_no, correct_36_no, total_37_no, correct_37_no, total_38_no, correct_38_no,  ...
            total_39_no, correct_39_no, total_40_no, correct_40_no, total_41_no, correct_41_no] = ...
  attack_test_binary(labels_num, ...
  predicted1,   predicted2,   predicted3,   predicted4, ...
  predicted5,   predicted6,   predicted7,   predicted8, ...
  predicted9,   predicted10, predicted11, predicted12, ...
  predicted13, predicted14, predicted15, predicted16, ...
  predicted17, predicted18, predicted19, predicted20, ...
  predicted21, predicted22, predicted23, ...
  predicted24, predicted25, predicted26, ...
  predicted27, predicted28, predicted29)
for i = 1:29
    if i == 1
        predicted = predicted1;
    elseif i == 2
        predicted = predicted2;
    elseif i == 3
        predicted = predicted3;
    elseif i == 4
        predicted = predicted4;
    elseif i == 5
        predicted = predicted5;
    elseif i == 6
        predicted = predicted6;
    elseif i == 7
        predicted = predicted7;
    elseif i == 8
        predicted = predicted8;
    elseif i == 9
        predicted = predicted9;
    elseif i == 10
        predicted = predicted10;
    elseif i == 11
        predicted = predicted11;
    elseif i == 12
        predicted = predicted12;
    elseif i == 13
        predicted = predicted13;
    elseif i == 14
        predicted = predicted14;
    elseif i == 15
        predicted = predicted15;
    elseif i == 16
        predicted = predicted16;
    elseif i == 17
        predicted = predicted17;
    elseif i == 18
```

```
        predicted = predicted18;
    elseif i == 19
        predicted = predicted19;
    elseif i == 20
        predicted = predicted20;
    elseif i == 21
        predicted = predicted21;
    elseif i == 22
        predicted = predicted22;
    elseif i == 23
        predicted = predicted23;
    elseif i == 24
        predicted = predicted24;
    elseif i == 25
        predicted = predicted25;
    elseif i == 26
        predicted = predicted26;
    elseif i == 27
        predicted = predicted27;
    elseif i == 28
        predicted = predicted28;
    elseif i == 29
        predicted = predicted29;
    end
% DOS
total_3_record = find(labels_num(:,1)==3);
total_3_no(i) = length(total_3_record);
correct_3_record = find(labels_num(:,1)==3 & predicted(:)~=1);
correct_3_no(i) = length(correct_3_record);
total_9_record = find(labels_num(:,1)==9);
total_9_no(i) = length(total_9_record);
correct_9_record = find(labels_num(:,1)==9 & predicted(:)~=1);
correct_9_no(i) = length(correct_9_record);
total_12_record = find(labels_num(:,1)==12);
total_12_no(i) = length(total_12_record);
correct_12_record = find(labels_num(:,1)==12 & predicted(:)~=1);
correct_12_no(i) = length(correct_12_record);
total_16_record = find(labels_num(:,1)==16);
total_16_no(i) = length(total_16_record);
correct_16_record = find(labels_num(:,1)==16 & predicted(:)~=1);
correct_16_no(i) = length(correct_16_record);
total_20_record = find(labels_num(:,1)==20);
total_20_no(i) = length(total_20_record);
correct_20_record = find(labels_num(:,1)==20 & predicted(:)~=1);
correct_20_no(i) = length(correct_20_record);
total_22_record = find(labels_num(:,1)==22);
total_22_no(i) = length(total_22_record);
correct_22_record = find(labels_num(:,1)==22 & predicted(:)~=1);
correct_22_no(i) = length(correct_22_record);
total_27_record = find(labels_num(:,1)==27);
total_27_no(i) = length(total_27_record);
correct_27_record = find(labels_num(:,1)==27 & predicted(:)~=1);
correct_27_no(i) = length(correct_27_record);
total_25_record = find(labels_num(:,1)==25);
```

```
total_25_no(i) = length(total_25_record);
correct_25_record = find(labels_num(:,1)==25 & predicted(:)~=1);
correct_25_no(i) = length(correct_25_record);
total_30_record = find(labels_num(:,1)==30);
total_30_no(i) = length(total_30_record);
correct_30_record = find(labels_num(:,1)==30 & predicted(:)~=1);
correct_30_no(i) = length(correct_30_record);
total_37_record = find(labels_num(:,1)==37);
total_37_no(i) = length(total_37_record);
correct_37_record = find(labels_num(:,1)==37 & predicted(:)~=1);
correct_37_no(i) = length(correct_37_record);
% PROBE
total_8_record = find(labels_num(:,1)==8);
total_8_no(i) = length(total_8_record);
correct_8_record = find(labels_num(:,1)==8 & predicted(:)~=1);
correct_8_no(i) = length(correct_8_record);
total_13_record = find(labels_num(:,1)==13);
total_13_no(i) = length(total_13_record);
correct_13_record = find(labels_num(:,1)==13 & predicted(:)~=1);
correct_13_no(i) = length(correct_13_record);
total_17_record = find(labels_num(:,1)==17);
total_17_no(i) = length(total_17_record);
correct_17_record = find(labels_num(:,1)==17 & predicted(:)~=1);
correct_17_no(i) = length(correct_17_record);
total_19_record = find(labels_num(:,1)==19);
total_19_no(i) = length(total_19_record);
correct_19_record = find(labels_num(:,1)==19 & predicted(:)~=1);
correct_19_no(i) = length(correct_19_record);
total_28_record = find(labels_num(:,1)==28);
total_28_no(i) = length(total_28_record);
correct_28_record = find(labels_num(:,1)==28 & predicted(:)~=1);
correct_28_no(i) = length(correct_28_record);
total_32_record = find(labels_num(:,1)==32);
total_32_no(i) = length(total_32_record);
correct_32_record = find(labels_num(:,1)==32 & predicted(:)~=1);
correct_32_no(i) = length(correct_32_record);
% R2L
total_5_record = find(labels_num(:,1)==5);
total_5_no(i) = length(total_5_record);
correct_5_record = find(labels_num(:,1)==5 & predicted(:)~=1);
correct_5_no(i) = length(correct_5_record);
total_6_record = find(labels_num(:,1)==6);
total_6_no(i) = length(total_6_record);
correct_6_record = find(labels_num(:,1)==6 & predicted(:)~=1);
correct_6_no(i) = length(correct_6_record);
total_7_record = find(labels_num(:,1)==7);
total_7_no(i) = length(total_7_record);
correct_7_record = find(labels_num(:,1)==7 & predicted(:)~=1);
correct_7_no(i) = length(correct_7_record);
total_11_record = find(labels_num(:,1)==11);
total_11_no(i) = length(total_11_record);
correct_11_record = find(labels_num(:,1)==11 & predicted(:)~=1);
correct_11_no(i) = length(correct_11_record);
total_15_record = find(labels_num(:,1)==15);
```

```
total_15_no(i) = length(total_15_record);
correct_15_record = find(labels_num(:,1)==15 & predicted(:)~=1);
correct_15_no(i) = length(correct_15_record);
total_21_record = find(labels_num(:,1)==21);
total_21_no(i) = length(total_21_record);
correct_21_record = find(labels_num(:,1)==21 & predicted(:)~=1);
correct_21_no(i) = length(correct_21_record);
total_23_record = find(labels_num(:,1)==23);
total_23_no(i) = length(total_23_record);
correct_23_record = find(labels_num(:,1)==23 & predicted(:)~=1);
correct_23_no(i) = length(correct_23_record);
total_24_record = find(labels_num(:,1)==24);
total_24_no(i) = length(total_24_record);
correct_24_record = find(labels_num(:,1)==24 & predicted(:)~=1);
correct_24_no(i) = length(correct_24_record);
total_29_record = find(labels_num(:,1)==29);
total_29_no(i) = length(total_29_record);
correct_29_record = find(labels_num(:,1)==29 & predicted(:)~=1);
correct_29_no(i) = length(correct_29_record);
total_33_record = find(labels_num(:,1)==33);
total_33_no(i) = length(total_33_record);
correct_33_record = find(labels_num(:,1)==33 & predicted(:)~=1);
correct_33_no(i) = length(correct_33_record);
total_34_record = find(labels_num(:,1)==34);
total_34_no(i) = length(total_34_record);
correct_34_record = find(labels_num(:,1)==34 & predicted(:)~=1);
correct_34_no(i) = length(correct_34_record);
total_35_record = find(labels_num(:,1)==35);
total_35_no(i) = length(total_35_record);
correct_35_record = find(labels_num(:,1)==35 & predicted(:)~=1);
correct_35_no(i) = length(correct_35_record);
total_38_record = find(labels_num(:,1)==38);
total_38_no(i) = length(total_38_record);
correct_38_record = find(labels_num(:,1)==38 & predicted(:)~=1);
correct_38_no(i) = length(correct_38_record);
total_39_record = find(labels_num(:,1)==39);
total_39_no(i) = length(total_39_record);
correct_39_record = find(labels_num(:,1)==39 & predicted(:)~=1);
correct_39_no(i) = length(correct_39_record);
total_40_record = find(labels_num(:,1)==40);
total_40_no(i) = length(total_40_record);
correct_40_record = find(labels_num(:,1)==40 & predicted(:)~=1);
correct_40_no(i) = length(correct_40_record);
% U2R
total_4_record = find(labels_num(:,1)==4);
total_4_no(i) = length(total_4_record);
correct_4_record = find(labels_num(:,1)==4 & predicted(:)~=1);
correct_4_no(i) = length(correct_4_record);
total_10_record = find(labels_num(:,1)==10);
total_10_no(i) = length(total_10_record);
correct_10_record = find(labels_num(:,1)==10 & predicted(:)~=1);
correct_10_no(i) = length(correct_10_record);
total_14_record = find(labels_num(:,1)==14);
total_14_no(i) = length(total_14_record);
```

```
correct_14_record = find(labels_num(:,1)==14 & predicted(:)~=1);
correct_14_no(i) = length(correct_14_record);
total_18_record = find(labels_num(:,1)==18);
total_18_no(i) = length(total_18_record);
correct_18_record = find(labels_num(:,1)==18 & predicted(:)~=1);
correct_18_no(i) = length(correct_18_record);
total_26_record = find(labels_num(:,1)==26);
total_26_no(i) = length(total_26_record);
correct_26_record = find(labels_num(:,1)==26 & predicted(:)~=1);
correct_26_no(i) = length(correct_26_record);
total_31_record = find(labels_num(:,1)==31);
total_31_no(i) = length(total_31_record);
correct_31_record = find(labels_num(:,1)==31 & predicted(:)~=1);
correct_31_no(i) = length(correct_31_record);
total_36_record = find(labels_num(:,1)==36);
total_36_no(i) = length(total_36_record);
correct_36_record = find(labels_num(:,1)==36 & predicted(:)~=1);
correct_36_no(i) = length(correct_36_record);
total_41_record = find(labels_num(:,1)==41);
total_41_no(i) = length(total_41_record);
correct_41_record = find(labels_num(:,1)==41 & predicted(:)~=1);
correct_41_no(i) = length(correct_41_record);
end
```

main.m

```
clc;
clear all;
close all;
% READING DATA
t0 = clock;
kdd_train = som_read_data('kdd_org_Data_allnumber.txt');
kdd_test = som_read_data('kdd_test_Data_allnumber.txt');
t = etime(clock,t0);
fprintf('\nreading data time =%6.2f sec\n', t);
% SEPARATE DATA to normal, dos, probe, u2r and r2l categories
[all_normal_train.data, all_normal_train.labels, ...
 all_dos_train.data,    all_dos_train.labels, ...
 all_probe_train.data,  all_probe_train.labels, ...
 all_u2r_train.data,    all_u2r_train.labels, ...
 all_r2l_train.data,    all_r2l_train.labels] = dataprocess(kdd_train.data, kdd_train.labels);
 [all_normal_test.data, all_normal_test.labels, ...
 all_dos_test.data,    all_dos_test.labels, ...
 all_probe_test.data,  all_probe_test.labels, ...
 all_u2r_test.data,    all_u2r_test.labels, ...
 all_r2l_test.data,    all_r2l_test.labels] = dataprocess(kdd_test.data, kdd_test.labels);
% TRAINING
% original – 494,020
% normal:97,277, dos:391,458, probe:4,107, u2r:52, r2l:1,126
% no duplicate – 145,585
% normal:87,831, dos:54,572, probe:2,131, u2r:52, r2l:999
% selected no. of training records in each category
normal_train_record_no = 878;
dos_train_record_no = 0;
probe_train_record_no = 0;
```

```
u2r_train_record_no = 0;
r2l_train_record_no = 99;
train_record_no = normal_train_record_no + dos_train_record_no + probe_train_record_no +
u2r_train_record_no + r2l_train_record_no;
% TESTING
% original – 311,029
% normal:60,593, dos:229,853, probe:4,166, u2r:228, r2l:16,189
% no duplicate – 77,291
% normal:47,913, dos:23,568, probe:2,682, u2r:215, r2l:2,913
% selected no. of testing records in each category
normal_test_record_no = 479;
dos_test_record_no = 0;
probe_test_record_no = 0;
u2r_test_record_no = 0;
r2l_test_record_no = 291;
test_record_no = normal_test_record_no + dos_test_record_no + probe_test_record_no +
u2r_test_record_no + r2l_test_record_no;
% attack_index: normal = 1, dos = 2, probe = 3, r2l = 4, u2r = 5
attack_index = 4;
% attack_label: normal = 1, dos = 2, probe = 3, r2l = 4, u2r = 5 for knn, fknn, etknn
attack_label = 4;
% attack_label: 2 for myknn
attack_mylabel = 2; % fix
attack_train_record_no = r2l_train_record_no;
attack_test_record_no = r2l_test_record_no;
nn = 2; % number of nearest neighbors
iter_no = 3; % number of iteration
for k = 1:nn
for iter = 1:iter_no
% RANDOMLY SELECT records
% randomly select records for normal, dos, probe, u2r and r2l category
[normal_train.data,normal_train.labels,normal_left.data,normal_left.labels] =
transfer(all_normal_train.data,all_normal_train.labels,normal_train_record_no);
[dos_train.data,dos_train.labels,dos_left.data,dos_left.labels] =
transfer(all_dos_train.data,all_dos_train.labels,dos_train_record_no);
[probe_train.data,probe_train.labels,probe_left.data,probe_left.labels] =
transfer(all_probe_train.data,all_probe_train.labels,probe_train_record_no);
[u2r_train.data,u2r_train.labels,u2r_left.data,u2r_left.labels] =
transfer(all_u2r_train.data,all_u2r_train.labels,u2r_train_record_no);
[r2l_train.data,r2l_train.labels,r2l_left.data,r2l_left.labels] =
transfer(all_r2l_train.data,all_r2l_train.labels,r2l_train_record_no);
 [normal_test.data,normal_test.labels,normal_left.data,normal_left.labels] =
transfer(all_normal_test.data,all_normal_test.labels,normal_test_record_no);
[dos_test.data,dos_test.labels,dos_left.data,dos_left.labels] =
transfer(all_dos_test.data,all_dos_test.labels,dos_test_record_no);
[probe_test.data,probe_test.labels,probe_left.data,probe_left.labels] =
transfer(all_probe_test.data,all_probe_test.labels,probe_test_record_no);
[u2r_test.data,u2r_test.labels,u2r_left.data,u2r_left.labels] =
transfer(all_u2r_test.data,all_u2r_test.labels,u2r_test_record_no);
[r2l_test.data,r2l_test.labels,r2l_left.data,r2l_left.labels] =
transfer(all_r2l_test.data,all_r2l_test.labels,r2l_test_record_no);
% FINAL data set
mytrain.data = [normal_train.data;dos_train.data;probe_train.data;u2r_train.data;r2l_train.data];
mytrain.labels = [normal_train.labels;dos_train.labels;probe_train.labels;u2r_train.labels;r2l_train.labels];
```

```
mytest.data = [normal_test.data;dos_test.data;probe_test.data;u2r_test.data;r2l_test.data];
mytest.labels = [normal_test.labels;dos_test.labels;probe_test.labels;u2r_test.labels;r2l_test.labels];
% NORMALIZATION of features
data_train.X = mytrain.data;
data_train = clust_normalize(data_train,'range');
mytrain.data = data_train.X;
data_test.X = mytest.data;
data_test = clust_normalize(data_test,'range');
mytest.data = data_test.X;
% FEATURE SELECTION
fs = 1;
% BASIC: 1-9
        % 1  2  3  4  5  6  7  8  9  10
w_e = [  1; 1; 1; 1; 1; 1; 0; 0; 0; 1; ...   %   1-10
            0; 1; 0; 0; 0; 0; 1; 0; 0; 0; ...   % 11-20
            0; 0; 1; 1; 0; 0; 1; 1; 1; 0; ...   % 21-30
            1; 1; 1; 1; 0; 1; 0; 0; 1; 0; 0];   % 31-41
% CONTENT: 10-22
        % 1  2  3  4  5  6  7  8  9  10
w_f = [  0; 0; 0; 0; 0; 0; 0; 0; 0; 1; ...   %   1-10
            1; 1; 1; 1; 1; 1; 1; 1; 1; 1; ...   % 11-20
            1; 1; 0; 0; 0; 0; 0; 0; 0; 0; ...   % 21-30
            0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0];   % 31-41
% TRAFFIC: 23-41
        % 1  2  3  4  5  6  7  8  9  10
w_g = [  0; 0; 0; 0; 0; 0; 0; 0; 0; 0; ...   %   1-10
            0; 0; 0; 0; 0; 0; 0; 0; 0; 0; ...   % 11-20
            0; 0; 1; 1; 1; 1; 1; 1; 1; 1; ...   % 21-30
            1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1];   % 31-41
% All: 1-41
        % 1  2  3  4  5  6  7  8  9  10
w_a = [  1; 1; 1; 1; 1; 1; 1; 1; 1; 1; ...   %   1-10
            1; 1; 1; 1; 1; 1; 1; 1; 1; 1; ...   % 11-20
            1; 1; 1; 1; 1; 1; 1; 1; 1; 1; ...   % 21-30
            1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1];   % 31-41
% Ours
% dos    : 1,2,3,4,5,6,12,23,24,31,32,37
% probe : 1,2,3,4,12,16,25,27,28,29,30,40
% u2r    : 1,2,3,10,16
% r2l    : 1,2,3,4,5,10,22
        % 1  2  3  4  5  6  7  8  9  10
w_b = [  1; 1; 1; 1; 1; 0; 0; 0; 0; 1; ...   %   1-10
            0; 0; 0; 0; 0; 0; 0; 0; 0; 0; ...   % 11-20
            0; 1; 0; 0; 0; 0; 0; 0; 0; 0; ...   % 21-30
            0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0];   % 31-41
% CFS
% dos    : 3,6,12,37
% probe : 3,4,25,29
% u2r    : 10
% r2l    : 10
        % 1  2  3  4  5  6  7  8  9  10
w_c = [  0; 0; 0; 0; 0; 0; 0; 0; 0; 1; ...   %   1-10
            0; 0; 0; 0; 0; 0; 0; 0; 0; 0; ...   % 11-20
            0; 0; 0; 0; 0; 0; 0; 0; 0; 0; ...   % 21-30
```

```
            0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0];  % 31-41
% FCBF
% dos    : 3,12,31,32
% probe : 3,26,27,29
% u2r    : 10,16
% r2l    : 5,10,39
        % 1  2  3  4  5  6  7  8  9  10
w_d = [  0; 0; 0; 0; 1; 0; 0; 0; 0; 1; ...    %   1-10
            0; 0; 0; 0; 0; 0; 0; 0; 1; 0; ...    % 11-20
            0; 0; 0; 0; 0; 0; 0; 0; 0; 0; ...    % 21-30
            0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0];   % 31-41
 [train_data_a, test_data_a] = fselection(mytrain.data, mytest.data, fs, w_a);
[train_data_b, test_data_b] = fselection(mytrain.data, mytest.data, fs, w_b);
[train_data_c, test_data_c] = fselection(mytrain.data, mytest.data, fs, w_c);
[train_data_d, test_data_d] = fselection(mytrain.data, mytest.data, fs, w_d);
[train_data_e, test_data_e] = fselection(mytrain.data, mytest.data, fs, w_e);
[train_data_f, test_data_f] = fselection(mytrain.data, mytest.data, fs, w_f);
[train_data_g, test_data_g] = fselection(mytrain.data, mytest.data, fs, w_g);
% CONVERTION
% convert cell (mytrain.labels,mytest.labels) to number (train_labels_num,test_labels_num)
% reason: fknn can only take number
[train_labels_num,test_labels_num] = celltonum(mytrain.labels,mytest.labels);
% for myknn
% CLUSTERING
% fuzzy cmeans
class_no = 2;
[center_a,U_a,objFcn] = fcm(train_data_a,class_no);  % NxM, N = no of class, M = no of training records
[center_b,U_b,objFcn] = fcm(train_data_b,class_no);
[center_c,U_c,objFcn] = fcm(train_data_c,class_no);
[center_d,U_d,objFcn] = fcm(train_data_d,class_no);
[center_e,U_e,objFcn] = fcm(train_data_e,class_no);
[center_f,U_f,objFcn] = fcm(train_data_f,class_no);
[center_g,U_g,objFcn] = fcm(train_data_g,class_no);
% for myknn
% for FCM to find clustering order (final_perms)
[finalperms_a, fcm1_labels_a] = fcm_order_binary(U_a, mytrain.labels);
[finalperms_b, fcm1_labels_b] = fcm_order_binary(U_b, mytrain.labels);
[finalperms_c, fcm1_labels_c] = fcm_order_binary(U_c, mytrain.labels);
[finalperms_d, fcm1_labels_d] = fcm_order_binary(U_d, mytrain.labels);
[finalperms_e, fcm1_labels_e] = fcm_order_binary(U_e, mytrain.labels);
[finalperms_f, fcm1_labels_f] = fcm_order_binary(U_f, mytrain.labels);
[finalperms_g, fcm1_labels_g] = fcm_order_binary(U_g, mytrain.labels);
% CLASSIFICATION of k-NN, fuzzy k-NN, evidence-theoretic k-NN and my-knn
% k-NN
tic;
[knn_predicted_a,  knn_memberships_a,  knn_numhits_a] = fknn(train_data_a, train_labels_num(:,2),
test_data_a, test_labels_num(:,2), k, 0, 0);
t_knn_a = toc; tic;
[knn_predicted_b,  knn_memberships_b,  knn_numhits_b] = fknn(train_data_b, train_labels_num(:,2),
test_data_b, test_labels_num(:,2), k, 0, 0);
t_knn_b = toc; tic;
[knn_predicted_c,  knn_memberships_c,  knn_numhits_c] = fknn(train_data_c, train_labels_num(:,2),
test_data_c, test_labels_num(:,2), k, 0, 0);
t_knn_c = toc; tic;
```

```
[knn_predicted_d,  knn_memberships_d,  knn_numhits_d] = fknn(train_data_d, train_labels_num(:,2),
test_data_d, test_labels_num(:,2), k, 0, 0);
t_knn_d = toc; tic;
[knn_predicted_e,  knn_memberships_e,  knn_numhits_e] = fknn(train_data_e, train_labels_num(:,2),
test_data_e, test_labels_num(:,2), k, 0, 0);
t_knn_e = toc; tic;
[knn_predicted_f,  knn_memberships_f,  knn_numhits_f] = fknn(train_data_f, train_labels_num(:,2),
test_data_f, test_labels_num(:,2), k, 0, 0);
t_knn_f = toc; tic;
[knn_predicted_g,  knn_memberships_g,  knn_numhits_g] = fknn(train_data_g, train_labels_num(:,2),
test_data_g, test_labels_num(:,2), k, 0, 0);
t_knn_g = toc;
% fuzzy k-NN
tic;
 [fknn_predicted_a, fknn_memberships_a, fknn_numhits_a] = fknn(train_data_a, train_labels_num(:,2),
test_data_a, test_labels_num(:,2), k, 0);
t_fknn_a = toc; tic;
[fknn_predicted_b, fknn_memberships_b, fknn_numhits_b] = fknn(train_data_b, train_labels_num(:,2),
test_data_b, test_labels_num(:,2), k, 0);
t_fknn_b = toc; tic;
[fknn_predicted_c, fknn_memberships_c, fknn_numhits_c] = fknn(train_data_c, train_labels_num(:,2),
test_data_c, test_labels_num(:,2), k, 0);
t_fknn_c = toc; tic;
[fknn_predicted_d, fknn_memberships_d, fknn_numhits_d] = fknn(train_data_d, train_labels_num(:,2),
test_data_d, test_labels_num(:,2), k, 0);
t_fknn_d = toc; tic;
[fknn_predicted_e, fknn_memberships_e, fknn_numhits_e] = fknn(train_data_e, train_labels_num(:,2),
test_data_e, test_labels_num(:,2), k, 0);
t_fknn_e = toc; tic;
[fknn_predicted_f, fknn_memberships_f, fknn_numhits_f] = fknn(train_data_f, train_labels_num(:,2),
test_data_f, test_labels_num(:,2), k, 0);
t_fknn_f = toc; tic;
[fknn_predicted_g, fknn_memberships_g, fknn_numhits_g] = fknn(train_data_g, train_labels_num(:,2),
test_data_g, test_labels_num(:,2), k, 0);
t_fknn_g = toc;
% evidence-theoretic k-NN
[gamm_a, alpha_a] = knndsinit(train_data_a, train_labels_num(:,2));
tic;
[etknn_memberships_a, etknn_predicted_a] = knndsval(train_data_a,
train_labels_num(:,2),k,gamm_a,alpha_a,0,test_data_a);
t_etknn_a = toc;
[gamm_b, alpha_b] = knndsinit(train_data_b, train_labels_num(:,2));
tic;
[etknn_memberships_b, etknn_predicted_b] = knndsval(train_data_b,
train_labels_num(:,2),k,gamm_b,alpha_b,0,test_data_b);
t_etknn_b = toc;
[gamm_c, alpha_c] = knndsinit(train_data_c, train_labels_num(:,2));
tic;
[etknn_memberships_c, etknn_predicted_c] = knndsval(train_data_c,
train_labels_num(:,2),k,gamm_c,alpha_c,0,test_data_c);
t_etknn_c = toc;
[gamm_d, alpha_d] = knndsinit(train_data_d, train_labels_num(:,2));
tic;
```

```
[etknn_memberships_d, etknn_predicted_d] = knndsval(train_data_d,
train_labels_num(:,2),k,gamm_d,alpha_d,0,test_data_d);
t_etknn_d = toc;
[gamm_e, alpha_e] = knndsinit(train_data_e, train_labels_num(:,2));
tic;
[etknn_memberships_e, etknn_predicted_e] = knndsval(train_data_e,
train_labels_num(:,2),k,gamm_e,alpha_e,0,test_data_e);
t_etknn_e = toc;
[gamm_f, alpha_f] = knndsinit(train_data_f, train_labels_num(:,2));
tic;
[etknn_memberships_f, etknn_predicted_f] = knndsval(train_data_f,
train_labels_num(:,2),k,gamm_f,alpha_f,0,test_data_f);
t_etknn_f = toc;
[gamm_g, alpha_g] = knndsinit(train_data_g, train_labels_num(:,2));
tic;
[etknn_memberships_g, etknn_predicted_g] = knndsval(train_data_g,
train_labels_num(:,2),k,gamm_g,alpha_g,0,test_data_g);
t_etknn_g = toc;
tic;
[myknn_memberships_a, myknn_predicted_a] = myknn_binary(train_data_a,
train_labels_num(:,2),k,U_a,0,test_data_a,finalperms_a);
t_myknn_a = toc; tic;
[myknn_memberships_b, myknn_predicted_b] = myknn_binary(train_data_b,
train_labels_num(:,2),k,U_b,0,test_data_b,finalperms_b);
t_myknn_b = toc; tic;
[myknn_memberships_c, myknn_predicted_c] = myknn_binary(train_data_c,
train_labels_num(:,2),k,U_c,0,test_data_c,finalperms_c);
t_myknn_c = toc; tic;
[myknn_memberships_d, myknn_predicted_d] = myknn_binary(train_data_d,
train_labels_num(:,2),k,U_d,0,test_data_d,finalperms_d);
t_myknn_d = toc; tic;
[myknn_memberships_e, myknn_predicted_e] = myknn_binary(train_data_e,
train_labels_num(:,2),k,U_e,0,test_data_e,finalperms_e);
t_myknn_e = toc; tic;
[myknn_memberships_f, myknn_predicted_f] = myknn_binary(train_data_f,
train_labels_num(:,2),k,U_f,0,test_data_f,finalperms_f);
t_myknn_f = toc; tic;
[myknn_memberships_g, myknn_predicted_g] = myknn_binary(train_data_g,
train_labels_num(:,2),k,U_g,0,test_data_g,finalperms_g);
t_myknn_g = toc;
train_data_a_w = train_data_a;
test_data_a_w = test_data_a;
mytrain.labels_a_w = mytrain.labels;
train_labels_num_a_w = train_labels_num;
U_a_w = U_a;
finalperms_a_w = finalperms_a;
[m_a_w,L_a_w] =
myknn_binary(train_data_a_w,train_labels_num_a_w(:,2),k,U_a_w,0,train_data_a_w,finalperms_a_w);
a_w = weight(train_data_a_w,train_labels_num_a_w,m_a_w);
tic;
[myknn_memberships_a_w,myknn_predicted_a_w] =
myknn_binary_w(train_data_a_w,train_labels_num_a_w(:,2),k,U_a_w,0,test_data_a_w,finalperms_a_w,a_
w);
t_myknn_a_w = toc;
```

```
train_data_b_w = train_data_b;
test_data_b_w = test_data_b;
mytrain.labels_b_w = mytrain.labels;
train_labels_num_b_w = train_labels_num;
U_b_w = U_b;
finalperms_b_w = finalperms_b;
[m_b_w,L_b_w] =
myknn_binary(train_data_b_w,train_labels_num_b_w(:,2),k,U_b_w,0,train_data_b_w,finalperms_b_w);
b_w = weight(train_data_b_w,train_labels_num_b_w,m_b_w);
tic;
[myknn_memberships_b_w,myknn_predicted_b_w] =
myknn_binary_w(train_data_b_w,train_labels_num_b_w(:,2),k,U_b_w,0,test_data_b_w,finalperms_b_w,b
_w);
t_myknn_b_w = toc;
train_data_c_w = train_data_c;
test_data_c_w = test_data_c;
mytrain.labels_c_w = mytrain.labels;
train_labels_num_c_w = train_labels_num;
U_c_w = U_c;
finalperms_c_w = finalperms_c;
[m_c_w,L_c_w] =
myknn_binary(train_data_c_w,train_labels_num_c_w(:,2),k,U_c_w,0,train_data_c_w,finalperms_c_w);
c_w = weight(train_data_c_w,train_labels_num_c_w,m_c_w);
tic;
[myknn_memberships_c_w,myknn_predicted_c_w] =
myknn_binary_w(train_data_c_w,train_labels_num_c_w(:,2),k,U_c_w,0,test_data_c_w,finalperms_c_w,c_
w);
t_myknn_c_w = toc;
train_data_d_w = train_data_d;
test_data_d_w = test_data_d;
mytrain.labels_d_w = mytrain.labels;
train_labels_num_d_w = train_labels_num;
U_d_w = U_d;
finalperms_d_w = finalperms_d;
[m_d_w,L_d_w] =
myknn_binary(train_data_d_w,train_labels_num_d_w(:,2),k,U_d_w,0,train_data_d_w,finalperms_d_w);
d_w = weight(train_data_d_w,train_labels_num_d_w,m_d_w);
tic;
[myknn_memberships_d_w,myknn_predicted_d_w] =
myknn_binary_w(train_data_d_w,train_labels_num_d_w(:,2),k,U_d_w,0,test_data_d_w,finalperms_d_w,d
_w);
t_myknn_d_w = toc;
fprintf('k = %d\n', k);
fprintf('\n KNN');
[knn_correct_normal_rate_a, knn_correct_attack_rate_a] = ...
   accuracy_binary(test_labels_num, knn_predicted_a, normal_test_record_no, attack_test_record_no,
attack_index, attack_label);
[knn_correct_normal_rate_b, knn_correct_attack_rate_b] = ...
   accuracy_binary(test_labels_num, knn_predicted_b, normal_test_record_no, attack_test_record_no,
attack_index, attack_label);
[knn_correct_normal_rate_c, knn_correct_attack_rate_c] = ...
   accuracy_binary(test_labels_num, knn_predicted_c, normal_test_record_no, attack_test_record_no,
attack_index, attack_label);
[knn_correct_normal_rate_d, knn_correct_attack_rate_d] = ...
```

```
    accuracy_binary(test_labels_num, knn_predicted_d, normal_test_record_no, attack_test_record_no,
attack_index, attack_label);
[knn_correct_normal_rate_e, knn_correct_attack_rate_e] = ...
    accuracy_binary(test_labels_num, knn_predicted_e, normal_test_record_no, attack_test_record_no,
attack_index, attack_label);
[knn_correct_normal_rate_f, knn_correct_attack_rate_f] = ...
    accuracy_binary(test_labels_num, knn_predicted_f, normal_test_record_no, attack_test_record_no,
attack_index, attack_label);
[knn_correct_normal_rate_g, knn_correct_attack_rate_g] = ...
    accuracy_binary(test_labels_num, knn_predicted_g, normal_test_record_no, attack_test_record_no,
attack_index, attack_label);
fprintf('\n FKNN');
[fknn_correct_normal_rate_a, fknn_correct_attack_rate_a] = ...
    accuracy_binary(test_labels_num, fknn_predicted_a, normal_test_record_no, attack_test_record_no,
attack_index, attack_label);
[fknn_correct_normal_rate_b, fknn_correct_attack_rate_b] = ...
    accuracy_binary(test_labels_num, fknn_predicted_b, normal_test_record_no, attack_test_record_no,
attack_index, attack_label);
[fknn_correct_normal_rate_c, fknn_correct_attack_rate_c] = ...
    accuracy_binary(test_labels_num, fknn_predicted_c, normal_test_record_no, attack_test_record_no,
attack_index, attack_label);
[fknn_correct_normal_rate_d, fknn_correct_attack_rate_d] = ...
    accuracy_binary(test_labels_num, fknn_predicted_d, normal_test_record_no, attack_test_record_no,
attack_index, attack_label);
[fknn_correct_normal_rate_e, fknn_correct_attack_rate_e] = ...
    accuracy_binary(test_labels_num, fknn_predicted_e, normal_test_record_no, attack_test_record_no,
attack_index, attack_label);
[fknn_correct_normal_rate_f, fknn_correct_attack_rate_f] = ...
    accuracy_binary(test_labels_num, fknn_predicted_f, normal_test_record_no, attack_test_record_no,
attack_index, attack_label);
[fknn_correct_normal_rate_g, fknn_correct_attack_rate_g] = ...
    accuracy_binary(test_labels_num, fknn_predicted_g, normal_test_record_no, attack_test_record_no,
attack_index, attack_label);
fprintf('\n ETKNN');
[etknn_correct_normal_rate_a, etknn_correct_attack_rate_a] = ...
    accuracy_binary(test_labels_num, etknn_predicted_a, normal_test_record_no, attack_test_record_no,
attack_index, attack_label);
[etknn_correct_normal_rate_b, etknn_correct_attack_rate_b] = ...
    accuracy_binary(test_labels_num, etknn_predicted_b, normal_test_record_no, attack_test_record_no,
attack_index, attack_label);
[etknn_correct_normal_rate_c, etknn_correct_attack_rate_c] = ...
    accuracy_binary(test_labels_num, etknn_predicted_c, normal_test_record_no, attack_test_record_no,
attack_index, attack_label);
[etknn_correct_normal_rate_d, etknn_correct_attack_rate_d] = ...
    accuracy_binary(test_labels_num, etknn_predicted_d, normal_test_record_no, attack_test_record_no,
attack_index, attack_label);
[etknn_correct_normal_rate_e, etknn_correct_attack_rate_e] = ...
    accuracy_binary(test_labels_num, etknn_predicted_e, normal_test_record_no, attack_test_record_no,
attack_index, attack_label);
[etknn_correct_normal_rate_f, etknn_correct_attack_rate_f] = ...
    accuracy_binary(test_labels_num, etknn_predicted_f, normal_test_record_no, attack_test_record_no,
attack_index, attack_label);
[etknn_correct_normal_rate_g, etknn_correct_attack_rate_g] = ...
```

```
    accuracy_binary(test_labels_num, etknn_predicted_g, normal_test_record_no, attack_test_record_no,
attack_index, attack_label);
fprintf('\n MYKNN');
[myknn_correct_normal_rate_a, myknn_correct_attack_rate_a] = ...
    accuracy_binary(test_labels_num, myknn_predicted_a, normal_test_record_no, attack_test_record_no,
attack_index, attack_mylabel);
[myknn_correct_normal_rate_b, myknn_correct_attack_rate_b] = ...
    accuracy_binary(test_labels_num, myknn_predicted_b, normal_test_record_no, attack_test_record_no,
attack_index, attack_mylabel);
[myknn_correct_normal_rate_c, myknn_correct_attack_rate_c] = ...
    accuracy_binary(test_labels_num, myknn_predicted_c, normal_test_record_no, attack_test_record_no,
attack_index, attack_mylabel);
[myknn_correct_normal_rate_d, myknn_correct_attack_rate_d] = ...
    accuracy_binary(test_labels_num, myknn_predicted_d, normal_test_record_no, attack_test_record_no,
attack_index, attack_mylabel);
[myknn_correct_normal_rate_e, myknn_correct_attack_rate_e] = ...
    accuracy_binary(test_labels_num, myknn_predicted_e, normal_test_record_no, attack_test_record_no,
attack_index, attack_mylabel);
[myknn_correct_normal_rate_f, myknn_correct_attack_rate_f] = ...
    accuracy_binary(test_labels_num, myknn_predicted_f, normal_test_record_no, attack_test_record_no,
attack_index, attack_mylabel);
[myknn_correct_normal_rate_g, myknn_correct_attack_rate_g] = ...
    accuracy_binary(test_labels_num, myknn_predicted_g, normal_test_record_no, attack_test_record_no,
attack_index, attack_mylabel);
fprintf('\n MYKNN-W');
[myknn_correct_normal_rate_a_w, myknn_correct_attack_rate_a_w] = ...
accuracy_binary(test_labels_num, myknn_predicted_a_w, normal_test_record_no, attack_test_record_no,
attack_index, attack_mylabel);
[myknn_correct_normal_rate_b_w, myknn_correct_attack_rate_b_w] = ...
accuracy_binary(test_labels_num, myknn_predicted_b_w, normal_test_record_no, attack_test_record_no,
attack_index, attack_mylabel);
[myknn_correct_normal_rate_c_w, myknn_correct_attack_rate_c_w] = ...
accuracy_binary(test_labels_num, myknn_predicted_c_w, normal_test_record_no, attack_test_record_no,
attack_index, attack_mylabel);
[myknn_correct_normal_rate_d_w, myknn_correct_attack_rate_d_w] = ...
accuracy_binary(test_labels_num, myknn_predicted_d_w, normal_test_record_no, attack_test_record_no,
attack_index, attack_mylabel);
A_memberships = myknn_memberships_b;
A_predicted = myknn_predicted_b;
B_memberships = myknn_memberships_e;
B_predicted = myknn_predicted_e;
C_memberships = myknn_memberships_f;
C_predicted = myknn_predicted_f;
D_memberships = myknn_memberships_g;
D_predicted = myknn_predicted_g;
E_memberships = myknn_memberships_e;
E_predicted = myknn_predicted_e;
F_memberships = myknn_memberships_a;
F_predicted = myknn_predicted_a;
G_memberships = etknn_memberships_b;
G_predicted = etknn_predicted_b;
G_membership(:,1) = G_memberships(:,1);
G_membership(:,2) = G_memberships(:,attack_label);
G_membership(:,3) = G_memberships(:,attack_label+1);
```

```
H_memberships = etknn_memberships_d;
H_predicted = etknn_predicted_d;
H_membership(:,1) = H_memberships(:,1);
H_membership(:,2) = H_memberships(:,attack_label);
H_membership(:,3) = H_memberships(:,attack_label+1);
I_memberships = etknn_memberships_c;
I_predicted = etknn_predicted_c;
I_membership(:,1) = I_memberships(:,1);
I_membership(:,2) = I_memberships(:,attack_label);
I_membership(:,3) = I_memberships(:,attack_label+1);
   if finalperms_b == [2 1]
     A_membership(:,1) = A_memberships(:,2);
     A_membership(:,2) = A_memberships(:,1);
     A_membership(:,3) = A_memberships(:,3);
   else
     A_membership = A_memberships;
   end
   if finalperms_e == [2 1]
     B_membership(:,1) = B_memberships(:,2);
     B_membership(:,2) = B_memberships(:,1);
     B_membership(:,3) = B_memberships(:,3);
    else
     B_membership = B_memberships;
   end
   if finalperms_f == [2 1]
     C_membership(:,1) = C_memberships(:,2);
     C_membership(:,2) = C_memberships(:,1);
     C_membership(:,3) = C_memberships(:,3);
   else
     C_membership = C_memberships;
   end
   if finalperms_g == [2 1]
     D_membership(:,1) = D_memberships(:,2);
     D_membership(:,2) = D_memberships(:,1);
     D_membership(:,3) = D_memberships(:,3);
   else
     D_membership = D_memberships;
   end
   if finalperms_e == [2 1]
     E_membership(:,1) = E_memberships(:,2);
     E_membership(:,2) = E_memberships(:,1);
     E_membership(:,3) = E_memberships(:,3);
   else
     E_membership = E_memberships;
   end
   if finalperms_a == [2 1]
     F_membership(:,1) = F_memberships(:,2);
     F_membership(:,2) = F_memberships(:,1);
     F_membership(:,3) = F_memberships(:,3);
   else
     F_membership = F_memberships;
   end
index = attack_mylabel;
fprintf('\n ENSEMBLE');
```

```
% ensemble: majority voting
[ensemble_predicted_mv, predict_mv] = ensemble_mv(A_predicted, ...
                                                  C_predicted, ...
                                                  D_predicted, ...
                                                  test_labels_num);
[ensemble_correct_normal_rate_mv, ensemble_correct_attack_rate_mv] = ...
  accuracy_binary(test_labels_num, ensemble_predicted_mv, normal_test_record_no,
attack_test_record_no, attack_index, attack_mylabel);
A1 = myknn_correct_normal_rate_b;
A2 = myknn_correct_attack_rate_b;
A_p = myknn_predicted_b;
B1 = myknn_correct_normal_rate_e;
B2 = myknn_correct_attack_rate_e;
B_p = myknn_predicted_e;
C1 = myknn_correct_normal_rate_f;
C2 = myknn_correct_attack_rate_f;
C_p = myknn_predicted_f;
D1 = myknn_correct_normal_rate_g;
D2 = myknn_correct_attack_rate_g;
D_p = myknn_predicted_g;
% ensemble: average
[ensemble_memberships_avg, ensemble_predicted_avg, predict_avg] = ensemble_avg(A_membership, …
                                                  A1, A2, index, ...
                                                  B_membership, B1, B2, ...
                                                  C_membership, C1, C2, ...
                                                  D_membership, D1, D2, ...
                                                  test_labels_num);
% ensemble: dempster-shafer
[ensemble_memberships_m, ensemble_predicted_m, predict_m] = ensemble_m(A_membership, …
                                                  A_predicted, index, ...
                                                  B_membership, B_predicted, ...
                                                  C_membership, C_predicted, ...
                                                  D_membership, D_predicted, ...
                                                  test_labels_num);
[ensemble_correct_normal_rate_m, ensemble_correct_attack_rate_m] = ...
  accuracy_binary(test_labels_num, ensemble_predicted_m, normal_test_record_no, attack_test_record_no,
attack_index, attack_mylabel);
% ensemble: naive bayes
% prior probability
if attack_index == 4
  P1 = normal_test_record_no / (normal_test_record_no + r2l_test_record_no); % normal distribution
  P2 = r2l_test_record_no / (normal_test_record_no + r2l_test_record_no); % attack distribution
elseif attack_index == 5
  P1 = normal_test_record_no / (normal_test_record_no + u2r_test_record_no);
  P2 = u2r_test_record_no / (normal_test_record_no + u2r_test_record_no);
end
tic;
[ensemble_predicted_bayes] = ensemble4(A1, A2, A_p, B1, B2, B_p, C1, C2, C_p, D1, D2, D_p, P1, P2);
t_ensemble_bayes = toc;
[ensemble_correct_normal_rate_bayes, ensemble_correct_attack_rate_bayes] = ...
  accuracy_binary(test_labels_num, ensemble_predicted_bayes, normal_test_record_no,
attack_test_record_no, attack_index, attack_mylabel);
% normal
myknn_predicted_a_tree = myknn_predicted_a;
```

```
myknn_predicted_b_tree = myknn_predicted_b;
myknn_predicted_c_tree = myknn_predicted_c;
myknn_predicted_d_tree = myknn_predicted_d;
myknn_predicted_e_tree = myknn_predicted_e;
myknn_predicted_f_tree = myknn_predicted_f;
myknn_predicted_g_tree = myknn_predicted_g;
ensemble_predicted_tree_m = ensemble_predicted_m;
ensemble_predicted_tree_mv = ensemble_predicted_mv;
ensemble_predicted_tree_bayes = ensemble_predicted_bayes;
% use reduced training set 145585 (41 features, normal-attacks) to generate rules
tic;
for i=1:size(myknn_predicted_a,1)
if attack_index == 4 || attack_index == 5
% 55229
if data_test.Xold(i,8) < 3 & ...
   data_test.Xold(i,13) < 1 & ...
   data_test.Xold(i,26) < 0.06 & ...
   data_test.Xold(i,27) < 0.06 & ...
   data_test.Xold(i,4) == 10 & ...
   data_test.Xold(i,10) < 1 & ...
   data_test.Xold(i,2) == 2 & ...
   data_test.Xold(i,3) == 20
      myknn_predicted_a_tree(i) = 1;
      myknn_predicted_b_tree(i) = 1;
      myknn_predicted_c_tree(i) = 1;
      myknn_predicted_d_tree(i) = 1;
      myknn_predicted_e_tree(i) = 1;
      myknn_predicted_f_tree(i) = 1;
      myknn_predicted_g_tree(i) = 1;
      ensemble_predicted_tree_m(i) = 1;
      ensemble_predicted_tree_mv(i) = 1;
      ensemble_predicted_tree_bayes(i) = 1;
% 9541
elseif data_test.Xold(i,8) < 3 & ...
   data_test.Xold(i,13) < 1 & ...
   data_test.Xold(i,26) < 0.06 & ...
   data_test.Xold(i,27) < 0.06 & ...
   data_test.Xold(i,4) == 10 & ...
   data_test.Xold(i,10) < 1 & ...
   data_test.Xold(i,2) == 2 & ...
   data_test.Xold(i,3) == 49
      myknn_predicted_a_tree(i) = 1;
      myknn_predicted_b_tree(i) = 1;
      myknn_predicted_c_tree(i) = 1;
      myknn_predicted_d_tree(i) = 1;
      myknn_predicted_e_tree(i) = 1;
      myknn_predicted_f_tree(i) = 1;
      myknn_predicted_g_tree(i) = 1;
      ensemble_predicted_tree_m(i) = 1;
      ensemble_predicted_tree_mv(i) = 1;
      ensemble_predicted_tree_bayes(i) = 1;
% 4611
elseif data_test.Xold(i,8) < 3 & ...
   data_test.Xold(i,13) < 1 & ...
```

```
    data_test.Xold(i,26) < 0.06 & ...
    data_test.Xold(i,27) >= 1 & ...
    data_test.Xold(i,10) < 1 & ...
    data_test.Xold(i,3) == 20 & ...
    data_test.Xold(i,30) < 0.01 & ...
    data_test.Xold(i,4) == 2
        myknn_predicted_a_tree(i) = 1;
        myknn_predicted_b_tree(i) = 1;
        myknn_predicted_c_tree(i) = 1;
        myknn_predicted_d_tree(i) = 1;
        myknn_predicted_e_tree(i) = 1;
        myknn_predicted_f_tree(i) = 1;
        myknn_predicted_g_tree(i) = 1;
        ensemble_predicted_tree_m(i) = 1;
        ensemble_predicted_tree_mv(i) = 1;
        ensemble_predicted_tree_bayes(i) = 1;
% 9167
elseif data_test.Xold(i,8) < 3 & ...
    data_test.Xold(i,13) < 1 & ...
    data_test.Xold(i,26) < 0.06 & ...
    data_test.Xold(i,27) < 0.06 & ...
    data_test.Xold(i,4) == 10 & ...
    data_test.Xold(i,10) < 1 & ...
    data_test.Xold(i,2) == 3 & ...
    data_test.Xold(i,40) < 0.01 & ...
    data_test.Xold(i,30) < 0.01 & ...
    data_test.Xold(i,5) >= 30 & ...
    data_test.Xold(i,5) < 158
        myknn_predicted_a_tree(i) = 1;
        myknn_predicted_b_tree(i) = 1;
        myknn_predicted_c_tree(i) = 1;
        myknn_predicted_d_tree(i) = 1;
        myknn_predicted_e_tree(i) = 1;
        myknn_predicted_f_tree(i) = 1;
        myknn_predicted_g_tree(i) = 1;
        ensemble_predicted_tree_m(i) = 1;
        ensemble_predicted_tree_mv(i) = 1;
        ensemble_predicted_tree_bayes(i) = 1;
% 1212
elseif data_test.Xold(i,8) < 3 & ...
    data_test.Xold(i,13) < 1 & ...
    data_test.Xold(i,26) < 0.06 & ...
    data_test.Xold(i,27) < 0.06 & ...
    data_test.Xold(i,4) == 10 & ...
    data_test.Xold(i,10) < 1 & ...
    data_test.Xold(i,2) == 3 & ...
    data_test.Xold(i,40) < 0.01 & ...
    data_test.Xold(i,30) >= 0.5
        myknn_predicted_a_tree(i) = 1;
        myknn_predicted_b_tree(i) = 1;
        myknn_predicted_c_tree(i) = 1;
        myknn_predicted_d_tree(i) = 1;
        myknn_predicted_e_tree(i) = 1;
        myknn_predicted_f_tree(i) = 1;
```

```
        myknn_predicted_g_tree(i) = 1;
        ensemble_predicted_tree_m(i) = 1;
        ensemble_predicted_tree_mv(i) = 1;
        ensemble_predicted_tree_bayes(i) = 1;
% 107
elseif data_test.Xold(i,8) < 3 & ...
    data_test.Xold(i,13) < 1 & ...
    data_test.Xold(i,26) < 0.06 & ...
    data_test.Xold(i,27) < 0.06 & ...
    data_test.Xold(i,4) == 10 & ...
    data_test.Xold(i,10) < 1 & ...
    data_test.Xold(i,2) == 2 & ...
    data_test.Xold(i,3) == 17 & ...
    data_test.Xold(i,6) < 29 & ...
    data_test.Xold(i,1) < 1 & ...
    data_test.Xold(i,5) >= 5 & ...
    data_test.Xold(i,5) < 30
        myknn_predicted_a_tree(i) = 1;
        myknn_predicted_b_tree(i) = 1;
        myknn_predicted_c_tree(i) = 1;
        myknn_predicted_d_tree(i) = 1;
        myknn_predicted_e_tree(i) = 1;
        myknn_predicted_f_tree(i) = 1;
        myknn_predicted_g_tree(i) = 1;
        ensemble_predicted_tree_m(i) = 1;
        ensemble_predicted_tree_mv(i) = 1;
        ensemble_predicted_tree_bayes(i) = 1;
% 611
elseif data_test.Xold(i,8) < 3 & ...
    data_test.Xold(i,13) < 1 & ...
    data_test.Xold(i,26) < 0.06 & ...
    data_test.Xold(i,27) < 0.06 & ...
    data_test.Xold(i,4) == 10 & ...
    data_test.Xold(i,10) < 1 & ...
    data_test.Xold(i,2) == 2 & ...
    data_test.Xold(i,3) == 17 & ...
    data_test.Xold(i,6) < 29 & ...
    data_test.Xold(i,1) < 1 & ...
    data_test.Xold(i,5) >= 32 & ...
    data_test.Xold(i,5) < 246
        myknn_predicted_a_tree(i) = 1;
        myknn_predicted_b_tree(i) = 1;
        myknn_predicted_c_tree(i) = 1;
        myknn_predicted_d_tree(i) = 1;
        myknn_predicted_e_tree(i) = 1;
        myknn_predicted_f_tree(i) = 1;
        myknn_predicted_g_tree(i) = 1;
        ensemble_predicted_tree_m(i) = 1;
        ensemble_predicted_tree_mv(i) = 1;
        ensemble_predicted_tree_bayes(i) = 1;
% 88
elseif data_test.Xold(i,8) < 3 & ...
    data_test.Xold(i,13) < 1 & ...
    data_test.Xold(i,26) < 0.06 & ...
```

```
    data_test.Xold(i,27) < 0.06 & ...
    data_test.Xold(i,4) == 10 & ...
    data_test.Xold(i,10) < 1 & ...
    data_test.Xold(i,2) == 2 & ...
    data_test.Xold(i,3) == 17 & ...
    data_test.Xold(i,6) < 29 & ...
    data_test.Xold(i,1) < 1 & ...
    data_test.Xold(i,5) >= 248 & ...
    data_test.Xold(i,5) < 334
        myknn_predicted_a_tree(i) = 1;
        myknn_predicted_b_tree(i) = 1;
        myknn_predicted_c_tree(i) = 1;
        myknn_predicted_d_tree(i) = 1;
        myknn_predicted_e_tree(i) = 1;
        myknn_predicted_f_tree(i) = 1;
        myknn_predicted_g_tree(i) = 1;
        ensemble_predicted_tree_m(i) = 1;
        ensemble_predicted_tree_mv(i) = 1;
        ensemble_predicted_tree_bayes(i) = 1;
% 628
elseif data_test.Xold(i,8) < 3 & ...
    data_test.Xold(i,13) < 1 & ...
    data_test.Xold(i,26) < 0.06 & ...
    data_test.Xold(i,27) < 0.06 & ...
    data_test.Xold(i,4) == 10 & ...
    data_test.Xold(i,10) < 1 & ...
    data_test.Xold(i,2) == 2 & ...
    data_test.Xold(i,3) == 17 & ...
    data_test.Xold(i,6) < 29 & ...
    data_test.Xold(i,1) < 1 & ...
    data_test.Xold(i,5) >= 335 & ...
    data_test.Xold(i,5) < 644
        myknn_predicted_a_tree(i) = 1;
        myknn_predicted_b_tree(i) = 1;
        myknn_predicted_c_tree(i) = 1;
        myknn_predicted_d_tree(i) = 1;
        myknn_predicted_e_tree(i) = 1;
        myknn_predicted_f_tree(i) = 1;
        myknn_predicted_g_tree(i) = 1;
        ensemble_predicted_tree_m(i) = 1;
        ensemble_predicted_tree_mv(i) = 1;
        ensemble_predicted_tree_bayes(i) = 1;
% 2310
elseif data_test.Xold(i,8) < 3 & ...
    data_test.Xold(i,13) < 1 & ...
    data_test.Xold(i,26) < 0.06 & ...
    data_test.Xold(i,27) < 0.06 & ...
    data_test.Xold(i,4) == 10 & ...
    data_test.Xold(i,10) < 1 & ...
    data_test.Xold(i,2) == 2 & ...
    data_test.Xold(i,3) == 17 & ...
    data_test.Xold(i,6) < 29 & ...
    data_test.Xold(i,1) < 1 & ...
    data_test.Xold(i,5) >= 726
```

```
        myknn_predicted_a_tree(i) = 1;
        myknn_predicted_b_tree(i) = 1;
        myknn_predicted_c_tree(i) = 1;
        myknn_predicted_d_tree(i) = 1;
        myknn_predicted_e_tree(i) = 1;
        myknn_predicted_f_tree(i) = 1;
        myknn_predicted_g_tree(i) = 1;
        ensemble_predicted_tree_m(i) = 1;
        ensemble_predicted_tree_mv(i) = 1;
        ensemble_predicted_tree_bayes(i) = 1;
end
end
end
t_tree = toc;
 [myknn_correct_normal_rate_a_tree, myknn_correct_attack_rate_a_tree] = ...
    accuracy_binary(test_labels_num, myknn_predicted_a_tree, normal_test_record_no,
attack_test_record_no, attack_index, attack_mylabel);
[myknn_correct_normal_rate_b_tree, myknn_correct_attack_rate_b_tree] = ...
    accuracy_binary(test_labels_num, myknn_predicted_b_tree, normal_test_record_no,
attack_test_record_no, attack_index, attack_mylabel);
[myknn_correct_normal_rate_c_tree, myknn_correct_attack_rate_c_tree] = ...
    accuracy_binary(test_labels_num, myknn_predicted_c_tree, normal_test_record_no,
attack_test_record_no, attack_index, attack_mylabel);
[myknn_correct_normal_rate_d_tree, myknn_correct_attack_rate_d_tree] = ...
    accuracy_binary(test_labels_num, myknn_predicted_d_tree, normal_test_record_no,
attack_test_record_no, attack_index, attack_mylabel);
[myknn_correct_normal_rate_e_tree, myknn_correct_attack_rate_e_tree] = ...
    accuracy_binary(test_labels_num, myknn_predicted_e_tree, normal_test_record_no,
attack_test_record_no, attack_index, attack_mylabel);
[myknn_correct_normal_rate_f_tree, myknn_correct_attack_rate_f_tree] = ...
    accuracy_binary(test_labels_num, myknn_predicted_f_tree, normal_test_record_no,
attack_test_record_no, attack_index, attack_mylabel);
[myknn_correct_normal_rate_g_tree, myknn_correct_attack_rate_g_tree] = ...
    accuracy_binary(test_labels_num, myknn_predicted_g_tree, normal_test_record_no,
attack_test_record_no, attack_index, attack_mylabel);
[ensemble_correct_normal_rate_tree_m, ensemble_correct_attack_rate_tree_m] = ...
    accuracy_binary(test_labels_num, ensemble_predicted_tree_m, normal_test_record_no,
attack_test_record_no, attack_index, attack_mylabel);
[ensemble_correct_normal_rate_tree_mv, ensemble_correct_attack_rate_tree_mv] = ...
    accuracy_binary(test_labels_num, ensemble_predicted_tree_mv, normal_test_record_no,
attack_test_record_no, attack_index, attack_mylabel);
[ensemble_correct_normal_rate_tree_bayes, ensemble_correct_attack_rate_tree_bayes] = ...
    accuracy_binary(test_labels_num, ensemble_predicted_tree_bayes, normal_test_record_no,
attack_test_record_no, attack_index, attack_mylabel);
fprintf('\n KNN');
knn_normal_rate_a(iter)   = knn_correct_normal_rate_a;
knn_attack_rate_a(iter)   = knn_correct_attack_rate_a;
knn_normal_rate_b(iter)   = knn_correct_normal_rate_b;
knn_attack_rate_b(iter)   = knn_correct_attack_rate_b;
knn_normal_rate_c(iter)   = knn_correct_normal_rate_c;
knn_attack_rate_c(iter)   = knn_correct_attack_rate_c;
knn_normal_rate_d(iter)   = knn_correct_normal_rate_d;
knn_attack_rate_d(iter)   = knn_correct_attack_rate_d;
knn_normal_rate_e(iter)   = knn_correct_normal_rate_e;
```

```
knn_attack_rate_e(iter)   = knn_correct_attack_rate_e;
knn_normal_rate_f(iter)   = knn_correct_normal_rate_f;
knn_attack_rate_f(iter)   = knn_correct_attack_rate_f;
knn_normal_rate_g(iter)   = knn_correct_normal_rate_g;
knn_attack_rate_g(iter)   = knn_correct_attack_rate_g;
knn_rate_a(iter) = (knn_correct_normal_rate_a*normal_test_record_no +
knn_correct_attack_rate_a*attack_test_record_no)/test_record_no;
knn_rate_b(iter) = (knn_correct_normal_rate_b*normal_test_record_no +
knn_correct_attack_rate_b*attack_test_record_no)/test_record_no;
knn_rate_c(iter) = (knn_correct_normal_rate_c*normal_test_record_no +
knn_correct_attack_rate_c*attack_test_record_no)/test_record_no;
knn_rate_d(iter) = (knn_correct_normal_rate_d*normal_test_record_no +
knn_correct_attack_rate_d*attack_test_record_no)/test_record_no;
knn_rate_e(iter) = (knn_correct_normal_rate_e*normal_test_record_no +
knn_correct_attack_rate_e*attack_test_record_no)/test_record_no;
knn_rate_f(iter) = (knn_correct_normal_rate_f*normal_test_record_no +
knn_correct_attack_rate_f*attack_test_record_no)/test_record_no;
knn_rate_g(iter) = (knn_correct_normal_rate_g*normal_test_record_no +
knn_correct_attack_rate_g*attack_test_record_no)/test_record_no;
time_knn_a(iter) = t_knn_a;
time_knn_b(iter) = t_knn_b;
time_knn_c(iter) = t_knn_c;
time_knn_d(iter) = t_knn_d;
time_knn_e(iter) = t_knn_e;
time_knn_f(iter) = t_knn_f;
time_knn_g(iter) = t_knn_g;
fprintf('\n FKNN');
fknn_normal_rate_a(iter)  = fknn_correct_normal_rate_a;
fknn_attack_rate_a(iter)  = fknn_correct_attack_rate_a;
fknn_normal_rate_b(iter)  = fknn_correct_normal_rate_b;
fknn_attack_rate_b(iter)  = fknn_correct_attack_rate_b;
fknn_normal_rate_c(iter)  = fknn_correct_normal_rate_c;
fknn_attack_rate_c(iter)  = fknn_correct_attack_rate_c;
fknn_normal_rate_d(iter)  = fknn_correct_normal_rate_d;
fknn_attack_rate_d(iter)  = fknn_correct_attack_rate_d;
fknn_normal_rate_e(iter)  = fknn_correct_normal_rate_e;
fknn_attack_rate_e(iter)  = fknn_correct_attack_rate_e;
fknn_normal_rate_f(iter)  = fknn_correct_normal_rate_f;
fknn_attack_rate_f(iter)  = fknn_correct_attack_rate_f;
fknn_normal_rate_g(iter)  = fknn_correct_normal_rate_g;
fknn_attack_rate_g(iter)  = fknn_correct_attack_rate_g;
fknn_rate_a(iter) = (fknn_correct_normal_rate_a*normal_test_record_no +
fknn_correct_attack_rate_a*attack_test_record_no)/test_record_no;
fknn_rate_b(iter) = (fknn_correct_normal_rate_b*normal_test_record_no +
fknn_correct_attack_rate_b*attack_test_record_no)/test_record_no;
fknn_rate_c(iter) = (fknn_correct_normal_rate_c*normal_test_record_no +
fknn_correct_attack_rate_c*attack_test_record_no)/test_record_no;
fknn_rate_d(iter) = (fknn_correct_normal_rate_d*normal_test_record_no +
fknn_correct_attack_rate_d*attack_test_record_no)/test_record_no;
fknn_rate_e(iter) = (fknn_correct_normal_rate_e*normal_test_record_no +
fknn_correct_attack_rate_e*attack_test_record_no)/test_record_no;
fknn_rate_f(iter) = (fknn_correct_normal_rate_f*normal_test_record_no +
fknn_correct_attack_rate_f*attack_test_record_no)/test_record_no;
```

```
fknn_rate_g(iter) = (fknn_correct_normal_rate_g*normal_test_record_no +
fknn_correct_attack_rate_g*attack_test_record_no)/test_record_no;
time_fknn_a(iter) = t_fknn_a;
time_fknn_b(iter) = t_fknn_b;
time_fknn_c(iter) = t_fknn_c;
time_fknn_d(iter) = t_fknn_d;
time_fknn_e(iter) = t_fknn_e;
time_fknn_f(iter) = t_fknn_f;
time_fknn_g(iter) = t_fknn_g;
fprintf('\n ETKNN');
etknn_normal_rate_a(iter) = etknn_correct_normal_rate_a;
etknn_attack_rate_a(iter) = etknn_correct_attack_rate_a;
etknn_normal_rate_b(iter) = etknn_correct_normal_rate_b;
etknn_attack_rate_b(iter) = etknn_correct_attack_rate_b;
etknn_normal_rate_c(iter) = etknn_correct_normal_rate_c;
etknn_attack_rate_c(iter) = etknn_correct_attack_rate_c;
etknn_normal_rate_d(iter) = etknn_correct_normal_rate_d;
etknn_attack_rate_d(iter) = etknn_correct_attack_rate_d;
etknn_normal_rate_e(iter) = etknn_correct_normal_rate_e;
etknn_attack_rate_e(iter) = etknn_correct_attack_rate_e;
etknn_normal_rate_f(iter) = etknn_correct_normal_rate_f;
etknn_attack_rate_f(iter) = etknn_correct_attack_rate_f;
etknn_normal_rate_g(iter) = etknn_correct_normal_rate_g;
etknn_attack_rate_g(iter) = etknn_correct_attack_rate_g;
etknn_rate_a(iter) = (etknn_correct_normal_rate_a*normal_test_record_no +
etknn_correct_attack_rate_a*attack_test_record_no)/test_record_no;
etknn_rate_b(iter) = (etknn_correct_normal_rate_b*normal_test_record_no +
etknn_correct_attack_rate_b*attack_test_record_no)/test_record_no;
etknn_rate_c(iter) = (etknn_correct_normal_rate_c*normal_test_record_no +
etknn_correct_attack_rate_c*attack_test_record_no)/test_record_no;
etknn_rate_d(iter) = (etknn_correct_normal_rate_d*normal_test_record_no +
etknn_correct_attack_rate_d*attack_test_record_no)/test_record_no;
etknn_rate_e(iter) = (etknn_correct_normal_rate_e*normal_test_record_no +
etknn_correct_attack_rate_e*attack_test_record_no)/test_record_no;
etknn_rate_f(iter) = (etknn_correct_normal_rate_f*normal_test_record_no +
etknn_correct_attack_rate_f*attack_test_record_no)/test_record_no;
etknn_rate_g(iter) = (etknn_correct_normal_rate_g*normal_test_record_no +
etknn_correct_attack_rate_g*attack_test_record_no)/test_record_no;
time_etknn_a(iter) = t_etknn_a;
time_etknn_b(iter) = t_etknn_b;
time_etknn_c(iter) = t_etknn_c;
time_etknn_d(iter) = t_etknn_d;
time_etknn_e(iter) = t_etknn_e;
time_etknn_f(iter) = t_etknn_f;
time_etknn_g(iter) = t_etknn_g;
fprintf('\n MYKNN');
myknn_normal_rate_a(iter) = myknn_correct_normal_rate_a;
myknn_attack_rate_a(iter) = myknn_correct_attack_rate_a;
myknn_normal_rate_b(iter) = myknn_correct_normal_rate_b;
myknn_attack_rate_b(iter) = myknn_correct_attack_rate_b;
myknn_normal_rate_c(iter) = myknn_correct_normal_rate_c;
myknn_attack_rate_c(iter) = myknn_correct_attack_rate_c;
myknn_normal_rate_d(iter) = myknn_correct_normal_rate_d;
myknn_attack_rate_d(iter) = myknn_correct_attack_rate_d;
```

```
myknn_normal_rate_e(iter) = myknn_correct_normal_rate_e;
myknn_attack_rate_e(iter) = myknn_correct_attack_rate_e;
myknn_normal_rate_f(iter) = myknn_correct_normal_rate_f;
myknn_attack_rate_f(iter) = myknn_correct_attack_rate_f;
myknn_normal_rate_g(iter) = myknn_correct_normal_rate_g;
myknn_attack_rate_g(iter) = myknn_correct_attack_rate_g;
myknn_rate_a(iter) = (myknn_correct_normal_rate_a*normal_test_record_no +
myknn_correct_attack_rate_a*attack_test_record_no)/test_record_no;
myknn_rate_b(iter) = (myknn_correct_normal_rate_b*normal_test_record_no +
myknn_correct_attack_rate_b*attack_test_record_no)/test_record_no;
myknn_rate_c(iter) = (myknn_correct_normal_rate_c*normal_test_record_no +
myknn_correct_attack_rate_c*attack_test_record_no)/test_record_no;
myknn_rate_d(iter) = (myknn_correct_normal_rate_d*normal_test_record_no +
myknn_correct_attack_rate_d*attack_test_record_no)/test_record_no;
myknn_rate_e(iter) = (myknn_correct_normal_rate_e*normal_test_record_no +
myknn_correct_attack_rate_e*attack_test_record_no)/test_record_no;
myknn_rate_f(iter) = (myknn_correct_normal_rate_f*normal_test_record_no +
myknn_correct_attack_rate_f*attack_test_record_no)/test_record_no;
myknn_rate_g(iter) = (myknn_correct_normal_rate_g*normal_test_record_no +
myknn_correct_attack_rate_g*attack_test_record_no)/test_record_no;
time_myknn_a(iter) = t_myknn_a;
time_myknn_b(iter) = t_myknn_b;
time_myknn_c(iter) = t_myknn_c;
time_myknn_d(iter) = t_myknn_d;
time_myknn_e(iter) = t_myknn_e;
time_myknn_f(iter) = t_myknn_f;
time_myknn_g(iter) = t_myknn_g;
fprintf('\n MYKNN-W');
myknn_normal_rate_a_w(iter) = myknn_correct_normal_rate_a_w;
myknn_attack_rate_a_w(iter) = myknn_correct_attack_rate_a_w;
myknn_normal_rate_b_w(iter) = myknn_correct_normal_rate_b_w;
myknn_attack_rate_b_w(iter) = myknn_correct_attack_rate_b_w;
myknn_normal_rate_c_w(iter) = myknn_correct_normal_rate_c_w;
myknn_attack_rate_c_w(iter) = myknn_correct_attack_rate_c_w;
myknn_normal_rate_d_w(iter) = myknn_correct_normal_rate_d_w;
myknn_attack_rate_d_w(iter) = myknn_correct_attack_rate_d_w;
myknn_rate_a_w(iter) = (myknn_correct_normal_rate_a_w*normal_test_record_no +
myknn_correct_attack_rate_a_w*attack_test_record_no)/test_record_no;
myknn_rate_b_w(iter) = (myknn_correct_normal_rate_b_w*normal_test_record_no +
myknn_correct_attack_rate_b_w*attack_test_record_no)/test_record_no;
myknn_rate_c_w(iter) = (myknn_correct_normal_rate_c_w*normal_test_record_no +
myknn_correct_attack_rate_c_w*attack_test_record_no)/test_record_no;
myknn_rate_d_w(iter) = (myknn_correct_normal_rate_d_w*normal_test_record_no +
myknn_correct_attack_rate_d_w*attack_test_record_no)/test_record_no;
time_myknn_a_w(iter) = t_myknn_a_w;
time_myknn_b_w(iter) = t_myknn_b_w;
time_myknn_c_w(iter) = t_myknn_c_w;
time_myknn_d_w(iter) = t_myknn_d_w;
fprintf('\n ENSEMBLE');
ensemble_normal_rate_m(iter) = ensemble_correct_normal_rate_m;
ensemble_attack_rate_m(iter) = ensemble_correct_attack_rate_m;
ensemble_normal_rate_mv(iter) = ensemble_correct_normal_rate_mv;
ensemble_attack_rate_mv(iter) = ensemble_correct_attack_rate_mv;
ensemble_normal_rate_bayes(iter) = ensemble_correct_normal_rate_bayes;
```

```
ensemble_attack_rate_bayes(iter) = ensemble_correct_attack_rate_bayes;
ensemble_rate_m(iter) = (ensemble_correct_normal_rate_m*normal_test_record_no +
ensemble_correct_attack_rate_m*attack_test_record_no)/test_record_no;
ensemble_rate_mv(iter) = (ensemble_correct_normal_rate_mv*normal_test_record_no +
ensemble_correct_attack_rate_mv*attack_test_record_no)/test_record_no;
ensemble_rate_bayes(iter) = (ensemble_correct_normal_rate_bayes*normal_test_record_no +
ensemble_correct_attack_rate_bayes*attack_test_record_no)/test_record_no;
time_ensemble_bayes(iter) = t_ensemble_bayes;
fprintf('\n TREE');
myknn_normal_rate_a_tree(iter) = myknn_correct_normal_rate_a_tree;
myknn_attack_rate_a_tree(iter) = myknn_correct_attack_rate_a_tree;
myknn_normal_rate_b_tree(iter) = myknn_correct_normal_rate_b_tree;
myknn_attack_rate_b_tree(iter) = myknn_correct_attack_rate_b_tree;
myknn_normal_rate_c_tree(iter) = myknn_correct_normal_rate_c_tree;
myknn_attack_rate_c_tree(iter) = myknn_correct_attack_rate_c_tree;
myknn_normal_rate_d_tree(iter) = myknn_correct_normal_rate_d_tree;
myknn_attack_rate_d_tree(iter) = myknn_correct_attack_rate_d_tree;
myknn_normal_rate_e_tree(iter) = myknn_correct_normal_rate_e_tree;
myknn_attack_rate_e_tree(iter) = myknn_correct_attack_rate_e_tree;
myknn_normal_rate_f_tree(iter) = myknn_correct_normal_rate_f_tree;
myknn_attack_rate_f_tree(iter) = myknn_correct_attack_rate_f_tree;
myknn_normal_rate_g_tree(iter) = myknn_correct_normal_rate_g_tree;
myknn_attack_rate_g_tree(iter) = myknn_correct_attack_rate_g_tree;
myknn_rate_a_tree(iter) = (myknn_correct_normal_rate_a_tree*normal_test_record_no +
myknn_correct_attack_rate_a_tree*attack_test_record_no)/test_record_no;
myknn_rate_b_tree(iter) = (myknn_correct_normal_rate_b_tree*normal_test_record_no +
myknn_correct_attack_rate_b_tree*attack_test_record_no)/test_record_no;
myknn_rate_c_tree(iter) = (myknn_correct_normal_rate_c_tree*normal_test_record_no +
myknn_correct_attack_rate_c_tree*attack_test_record_no)/test_record_no;
myknn_rate_d_tree(iter) = (myknn_correct_normal_rate_d_tree*normal_test_record_no +
myknn_correct_attack_rate_d_tree*attack_test_record_no)/test_record_no;
myknn_rate_e_tree(iter) = (myknn_correct_normal_rate_e_tree*normal_test_record_no +
myknn_correct_attack_rate_e_tree*attack_test_record_no)/test_record_no;
myknn_rate_f_tree(iter) = (myknn_correct_normal_rate_f_tree*normal_test_record_no +
myknn_correct_attack_rate_f_tree*attack_test_record_no)/test_record_no;
myknn_rate_g_tree(iter) = (myknn_correct_normal_rate_g_tree*normal_test_record_no +
myknn_correct_attack_rate_g_tree*attack_test_record_no)/test_record_no;
fprintf('\n ENSENBLE + TREE');
ensemble_normal_rate_tree_m(iter) = ensemble_correct_normal_rate_tree_m;
ensemble_attack_rate_tree_m(iter) = ensemble_correct_attack_rate_tree_m;
ensemble_normal_rate_tree_mv(iter) = ensemble_correct_normal_rate_tree_mv;
ensemble_attack_rate_tree_mv(iter) = ensemble_correct_attack_rate_tree_mv;
ensemble_normal_rate_tree_bayes(iter) = ensemble_correct_normal_rate_tree_bayes;
ensemble_attack_rate_tree_bayes(iter) = ensemble_correct_attack_rate_tree_bayes;
ensemble_rate_tree_m(iter) = (ensemble_correct_normal_rate_tree_m*normal_test_record_no +
ensemble_correct_attack_rate_tree_m*attack_test_record_no)/test_record_no;
ensemble_rate_tree_mv(iter) = (ensemble_correct_normal_rate_tree_mv*normal_test_record_no +
ensemble_correct_attack_rate_tree_mv*attack_test_record_no)/test_record_no;
ensemble_rate_tree_bayes(iter) = (ensemble_correct_normal_rate_tree_bayes*normal_test_record_no +
ensemble_correct_attack_rate_tree_bayes*attack_test_record_no)/test_record_no;
time_tree(iter) = t_tree;
%attacks in training set
[train_3_no, train_4_no, train_5_no, train_6_no, train_7_no, train_8_no, train_9_no, train_10_no, ...
 train_11_no, train_12_no, train_13_no, train_14_no, train_15_no, train_16_no, train_17_no, …
```

```
train_18_no, train_19_no, train_20_no, train_21_no, train_22_no, train_23_no, train_24_no, …
train_25_no, train_26_no, train_27_no, train_28_no, train_29_no, train_30_no, train_31_no, …
train_32_no, train_33_no, train_34_no, train_35_no, train_36_no, train_37_no, train_38_no, …
train_39_no, train_40_no, train_41_no ] = attack_train_binary(train_labels_num);
train_3_number(iter,:)  = train_3_no;   train_4_number(iter,:) = train_4_no;
train_5_number(iter,:)  = train_5_no;   train_6_number(iter,:) = train_6_no;
train_7_number(iter,:)  = train_7_no;   train_8_number(iter,:) = train_8_no;
train_9_number(iter,:)  = train_9_no;  train_10_number(iter,:) = train_10_no;
train_11_number(iter,:) = train_11_no; train_12_number(iter,:) = train_12_no;
train_13_number(iter,:) = train_13_no; train_14_number(iter,:) = train_14_no;
train_15_number(iter,:) = train_15_no; train_16_number(iter,:) = train_16_no;
train_17_number(iter,:) = train_17_no; train_18_number(iter,:) = train_18_no;
train_19_number(iter,:) = train_19_no; train_20_number(iter,:) = train_20_no;
train_21_number(iter,:) = train_21_no; train_22_number(iter,:) = train_22_no;
train_23_number(iter,:) = train_23_no; train_24_number(iter,:) = train_24_no;
train_25_number(iter,:) = train_25_no; train_26_number(iter,:) = train_26_no;
train_27_number(iter,:) = train_27_no; train_28_number(iter,:) = train_28_no;
train_29_number(iter,:) = train_29_no; train_30_number(iter,:) = train_30_no;
train_31_number(iter,:) = train_31_no; train_32_number(iter,:) = train_32_no;
train_33_number(iter,:) = train_33_no; train_34_number(iter,:) = train_34_no;
train_35_number(iter,:) = train_35_no; train_36_number(iter,:) = train_36_no;
train_37_number(iter,:) = train_37_no; train_38_number(iter,:) = train_38_no;
train_39_number(iter,:) = train_39_no; train_40_number(iter,:) = train_40_no;
train_41_number(iter,:) = train_41_no;
% attacks in testing set
      [ total_3_no,  correct_3_no, total_4_no,  correct_4_no, total_5_no,  correct_5_no,  ...
        total_6_no,  correct_6_no, total_7_no,  correct_7_no, total_8_no,  correct_8_no,  ...
        total_9_no,  correct_9_no, total_10_no, correct_10_no, total_11_no, correct_11_no,  ...
       total_12_no, correct_12_no, total_13_no, correct_13_no, total_14_no, correct_14_no,  ...
       total_15_no, correct_15_no, total_16_no, correct_16_no, otal_17_no, correct_17_no,  ...
       total_18_no, correct_18_no, total_19_no, correct_19_no, total_20_no, correct_20_no,  ...
       total_21_no, correct_21_no, total_22_no, correct_22_no, total_23_no, correct_23_no,  ...
       total_24_no, correct_24_no, total_25_no, correct_25_no, total_26_no, correct_26_no,  ...
       total_27_no, correct_27_no, total_28_no, correct_28_no, total_29_no, correct_29_no,  ...
       total_30_no, correct_30_no, total_31_no, correct_31_no, total_32_no, correct_32_no,  ...
       total_33_no, correct_33_no, total_34_no, correct_34_no, total_35_no, correct_35_no,  ...
       total_36_no, correct_36_no, total_37_no, correct_37_no, total_38_no, correct_38_no,  ...
       total_39_no, correct_39_no, total_40_no, correct_40_no, total_41_no, correct_41_no]= ...
       attack_test_binary(test_labels_num, ...
                    knn_predicted_a,    knn_predicted_b,    knn_predicted_c,    knn_predicted_d, ...
                    fknn_predicted_a,   fknn_predicted_b,   fknn_predicted_c,   fknn_predicted_d, ...
                    etknn_predicted_a,  etknn_predicted_b,  etknn_predicted_c,  etknn_predicted_d, ...
                    myknn_predicted_a,  myknn_predicted_b,  myknn_predicted_c,  myknn_predicted_d, ...
                    myknn_predicted_e,  myknn_predicted_f,  myknn_predicted_g, ...
                    myknn_predicted_a_w, myknn_predicted_b_w, myknn_predicted_c_w, …
                    myknn_predicted_d_w, ...
                    ensemble_predicted_m, ensemble_predicted_mv, ensemble_predicted_bayes, ...
                    ensemble_predicted_tree_m, ensemble_predicted_tree_mv, …
                    ensemble_predicted_tree_bayes);
total_3_number(iter,:) = total_3_no;   correct_3_number(iter,:)  = correct_3_no;
total_4_number(iter,:) = total_4_no;   correct_4_number(iter,:)  = correct_4_no;
total_5_number(iter,:) = total_5_no;   correct_5_number(iter,:)  = correct_5_no;
total_6_number(iter,:) = total_6_no;   correct_6_number(iter,:)  = correct_6_no;
total_7_number(iter,:) = total_7_no;   correct_7_number(iter,:)  = correct_7_no;
```

```
total_8_number(iter,:) = total_8_no;   correct_8_number(iter,:)  = correct_8_no;
total_9_number(iter,:) = total_9_no;   correct_9_number(iter,:)  = correct_9_no;
total_10_number(iter,:) = total_10_no; correct_10_number(iter,:) = correct_10_no;
total_11_number(iter,:) = total_11_no; correct_11_number(iter,:) = correct_11_no;
total_12_number(iter,:) = total_12_no; correct_12_number(iter,:) = correct_12_no;
total_13_number(iter,:) = total_13_no; correct_13_number(iter,:) = correct_13_no;
total_14_number(iter,:) = total_14_no; correct_14_number(iter,:) = correct_14_no;
total_15_number(iter,:) = total_15_no; correct_15_number(iter,:) = correct_15_no;
total_16_number(iter,:) = total_16_no; correct_16_number(iter,:) = correct_16_no;
total_17_number(iter,:) = total_17_no; correct_17_number(iter,:) = correct_17_no;
total_18_number(iter,:) = total_18_no; correct_18_number(iter,:) = correct_18_no;
total_19_number(iter,:) = total_19_no; correct_19_number(iter,:) = correct_19_no;
total_20_number(iter,:) = total_20_no; correct_20_number(iter,:) = correct_20_no;
total_21_number(iter,:) = total_21_no; correct_21_number(iter,:) = correct_21_no;
total_22_number(iter,:) = total_22_no; correct_22_number(iter,:) = correct_22_no;
total_23_number(iter,:) = total_23_no; correct_23_number(iter,:) = correct_23_no;
total_24_number(iter,:) = total_24_no; correct_24_number(iter,:) = correct_24_no;
total_25_number(iter,:) = total_25_no; correct_25_number(iter,:) = correct_25_no;
total_26_number(iter,:) = total_26_no; correct_26_number(iter,:) = correct_26_no;
total_27_number(iter,:) = total_27_no; correct_27_number(iter,:) = correct_27_no;
total_28_number(iter,:) = total_28_no; correct_28_number(iter,:) = correct_28_no;
total_29_number(iter,:) = total_29_no; correct_29_number(iter,:) = correct_29_no;
total_30_number(iter,:) = total_30_no; correct_30_number(iter,:) = correct_30_no;
total_31_number(iter,:) = total_31_no; correct_31_number(iter,:) = correct_31_no;
total_32_number(iter,:) = total_32_no; correct_32_number(iter,:) = correct_32_no;
total_33_number(iter,:) = total_33_no; correct_33_number(iter,:) = correct_33_no;
total_34_number(iter,:) = total_34_no; correct_34_number(iter,:) = correct_34_no;
total_35_number(iter,:) = total_35_no; correct_35_number(iter,:) = correct_35_no;
total_36_number(iter,:) = total_36_no; correct_36_number(iter,:) = correct_36_no;
total_37_number(iter,:) = total_37_no; correct_37_number(iter,:) = correct_37_no;
total_38_number(iter,:) = total_38_no; correct_38_number(iter,:) = correct_38_no;
total_39_number(iter,:) = total_39_no; correct_39_number(iter,:) = correct_39_no;
total_40_number(iter,:) = total_40_no; correct_40_number(iter,:) = correct_40_no;
total_41_number(iter,:) = total_41_no; correct_41_number(iter,:) = correct_41_no;
end % iter
fprintf('\n KNN');
knn_average_normal_rate_a(k)   = 1- sum(knn_normal_rate_a) / iter_no;
knn_average_attack_rate_a(k)   = sum(knn_attack_rate_a) / iter_no;
knn_average_normal_rate_b(k)   = 1- sum(knn_normal_rate_b) / iter_no;
knn_average_attack_rate_b(k)   = sum(knn_attack_rate_b) / iter_no;
knn_average_normal_rate_c(k)   = 1- sum(knn_normal_rate_c) / iter_no;
knn_average_attack_rate_c(k)   = sum(knn_attack_rate_c) / iter_no;
knn_average_normal_rate_d(k)   = 1- sum(knn_normal_rate_d) / iter_no;
knn_average_attack_rate_d(k)   = sum(knn_attack_rate_d) / iter_no;
knn_average_normal_rate_e(k)   = 1- sum(knn_normal_rate_e) / iter_no;
knn_average_attack_rate_e(k)   = sum(knn_attack_rate_e) / iter_no;
knn_average_normal_rate_f(k)   = 1- sum(knn_normal_rate_f) / iter_no;
knn_average_attack_rate_f(k)   = sum(knn_attack_rate_f) / iter_no;
knn_average_normal_rate_g(k)   = 1- sum(knn_normal_rate_g) / iter_no;
knn_average_attack_rate_g(k)   = sum(knn_attack_rate_g) / iter_no;
knn_average_rate_a(k) = sum(knn_rate_a) / iter_no;
knn_average_rate_b(k) = sum(knn_rate_b) / iter_no;
knn_average_rate_c(k) = sum(knn_rate_c) / iter_no;
knn_average_rate_d(k) = sum(knn_rate_d) / iter_no;
```

```
knn_average_rate_e(k) = sum(knn_rate_e) / iter_no;
knn_average_rate_f(k) = sum(knn_rate_f) / iter_no;
knn_average_rate_g(k) = sum(knn_rate_g) / iter_no;
time_average_knn_a(k) = sum(time_knn_a) / iter_no;
time_average_knn_b(k) = sum(time_knn_b) / iter_no;
time_average_knn_c(k) = sum(time_knn_c) / iter_no;
time_average_knn_d(k) = sum(time_knn_d) / iter_no;
time_average_knn_e(k) = sum(time_knn_e) / iter_no;
time_average_knn_f(k) = sum(time_knn_f) / iter_no;
time_average_knn_g(k) = sum(time_knn_g) / iter_no;
fprintf('\n FKNN');
fknn_average_normal_rate_a(k)  = 1- sum(fknn_normal_rate_a) / iter_no;
fknn_average_attack_rate_a(k)  = sum(fknn_attack_rate_a) / iter_no;
fknn_average_normal_rate_b(k)  = 1- sum(fknn_normal_rate_b) / iter_no;
fknn_average_attack_rate_b(k)  = sum(fknn_attack_rate_b) / iter_no;
fknn_average_normal_rate_c(k)  = 1- sum(fknn_normal_rate_c) / iter_no;
fknn_average_attack_rate_c(k)  = sum(fknn_attack_rate_c) / iter_no;
fknn_average_normal_rate_d(k)  = 1- sum(fknn_normal_rate_d) / iter_no;
fknn_average_attack_rate_d(k)  = sum(fknn_attack_rate_d) / iter_no;
fknn_average_normal_rate_e(k)  = 1- sum(fknn_normal_rate_e) / iter_no;
fknn_average_attack_rate_e(k)  = sum(fknn_attack_rate_e) / iter_no;
fknn_average_normal_rate_f(k)  = 1- sum(fknn_normal_rate_f) / iter_no;
fknn_average_attack_rate_f(k)  = sum(fknn_attack_rate_f) / iter_no;
fknn_average_normal_rate_g(k)  = 1- sum(fknn_normal_rate_g) / iter_no;
fknn_average_attack_rate_g(k)  = sum(fknn_attack_rate_g) / iter_no;
fknn_average_rate_a(k) = sum(fknn_rate_a) / iter_no;
fknn_average_rate_b(k) = sum(fknn_rate_b) / iter_no;
fknn_average_rate_c(k) = sum(fknn_rate_c) / iter_no;
fknn_average_rate_d(k) = sum(fknn_rate_d) / iter_no;
fknn_average_rate_e(k) = sum(fknn_rate_e) / iter_no;
fknn_average_rate_f(k) = sum(fknn_rate_f) / iter_no;
fknn_average_rate_g(k) = sum(fknn_rate_g) / iter_no;
time_average_fknn_a(k) = sum(time_fknn_a) / iter_no;
time_average_fknn_b(k) = sum(time_fknn_b) / iter_no;
time_average_fknn_c(k) = sum(time_fknn_c) / iter_no;
time_average_fknn_d(k) = sum(time_fknn_d) / iter_no;
time_average_fknn_e(k) = sum(time_fknn_e) / iter_no;
time_average_fknn_f(k) = sum(time_fknn_f) / iter_no;
time_average_fknn_g(k) = sum(time_fknn_g) / iter_no;
fprintf('\n ETKNN');
etknn_average_normal_rate_a(k) = 1- sum(etknn_normal_rate_a) / iter_no;
etknn_average_attack_rate_a(k) = sum(etknn_attack_rate_a) / iter_no;
etknn_average_normal_rate_b(k) = 1- sum(etknn_normal_rate_b) / iter_no;
etknn_average_attack_rate_b(k) = sum(etknn_attack_rate_b) / iter_no;
etknn_average_normal_rate_c(k) = 1- sum(etknn_normal_rate_c) / iter_no;
etknn_average_attack_rate_c(k) = sum(etknn_attack_rate_c) / iter_no;
etknn_average_normal_rate_d(k) = 1- sum(etknn_normal_rate_d) / iter_no;
etknn_average_attack_rate_d(k) = sum(etknn_attack_rate_d) / iter_no;
etknn_average_normal_rate_e(k) = 1- sum(etknn_normal_rate_e) / iter_no;
etknn_average_attack_rate_e(k) = sum(etknn_attack_rate_e) / iter_no;
etknn_average_normal_rate_f(k) = 1- sum(etknn_normal_rate_f) / iter_no;
etknn_average_attack_rate_f(k) = sum(etknn_attack_rate_f) / iter_no;
etknn_average_normal_rate_g(k) = 1- sum(etknn_normal_rate_g) / iter_no;
etknn_average_attack_rate_g(k) = sum(etknn_attack_rate_g) / iter_no;
```

```
etknn_average_rate_a(k) = sum(etknn_rate_a) / iter_no;
etknn_average_rate_b(k) = sum(etknn_rate_b) / iter_no;
etknn_average_rate_c(k) = sum(etknn_rate_c) / iter_no;
etknn_average_rate_d(k) = sum(etknn_rate_d) / iter_no;
etknn_average_rate_e(k) = sum(etknn_rate_e) / iter_no;
etknn_average_rate_f(k) = sum(etknn_rate_f) / iter_no;
etknn_average_rate_g(k) = sum(etknn_rate_g) / iter_no;
time_average_etknn_a(k) = sum(time_etknn_a) / iter_no;
time_average_etknn_b(k) = sum(time_etknn_b) / iter_no;
time_average_etknn_c(k) = sum(time_etknn_c) / iter_no;
time_average_etknn_d(k) = sum(time_etknn_d) / iter_no;
time_average_etknn_e(k) = sum(time_etknn_e) / iter_no;
time_average_etknn_f(k) = sum(time_etknn_f) / iter_no;
time_average_etknn_g(k) = sum(time_etknn_g) / iter_no;
fprintf('\n MYKNN');
myknn_average_normal_rate_a(k) = 1- sum(myknn_normal_rate_a) / iter_no;
myknn_average_attack_rate_a(k) = sum(myknn_attack_rate_a) / iter_no;
myknn_average_normal_rate_b(k) = 1- sum(myknn_normal_rate_b) / iter_no;
myknn_average_attack_rate_b(k) = sum(myknn_attack_rate_b) / iter_no;
myknn_average_normal_rate_c(k) = 1- sum(myknn_normal_rate_c) / iter_no;
myknn_average_attack_rate_c(k) = sum(myknn_attack_rate_c) / iter_no;
myknn_average_normal_rate_d(k) = 1- sum(myknn_normal_rate_d) / iter_no;
myknn_average_attack_rate_d(k) = sum(myknn_attack_rate_d) / iter_no;
myknn_average_normal_rate_e(k) = 1- sum(myknn_normal_rate_e) / iter_no;
myknn_average_attack_rate_e(k) = sum(myknn_attack_rate_e) / iter_no;
myknn_average_normal_rate_f(k) = 1- sum(myknn_normal_rate_f) / iter_no;
myknn_average_attack_rate_f(k) = sum(myknn_attack_rate_f) / iter_no;
myknn_average_normal_rate_g(k) = 1- sum(myknn_normal_rate_g) / iter_no;
myknn_average_attack_rate_g(k) = sum(myknn_attack_rate_g) / iter_no;
myknn_average_rate_a(k) = sum(myknn_rate_a) / iter_no;
myknn_average_rate_b(k) = sum(myknn_rate_b) / iter_no;
myknn_average_rate_c(k) = sum(myknn_rate_c) / iter_no;
myknn_average_rate_d(k) = sum(myknn_rate_d) / iter_no;
myknn_average_rate_e(k) = sum(myknn_rate_e) / iter_no;
myknn_average_rate_f(k) = sum(myknn_rate_f) / iter_no;
myknn_average_rate_g(k) = sum(myknn_rate_g) / iter_no;
time_average_myknn_a(k) = sum(time_myknn_a) / iter_no;
time_average_myknn_b(k) = sum(time_myknn_b) / iter_no;
time_average_myknn_c(k) = sum(time_myknn_c) / iter_no;
time_average_myknn_d(k) = sum(time_myknn_d) / iter_no;
time_average_myknn_e(k) = sum(time_myknn_e) / iter_no;
time_average_myknn_f(k) = sum(time_myknn_f) / iter_no;
time_average_myknn_g(k) = sum(time_myknn_g) / iter_no;
fprintf('\n MYKNN-W');
myknn_average_normal_rate_a_w(k) = 1- sum(myknn_normal_rate_a_w) / iter_no;
myknn_average_attack_rate_a_w(k) = sum(myknn_attack_rate_a_w) / iter_no;
myknn_average_normal_rate_b_w(k) = 1- sum(myknn_normal_rate_b_w) / iter_no;
myknn_average_attack_rate_b_w(k) = sum(myknn_attack_rate_b_w) / iter_no;
myknn_average_normal_rate_c_w(k) = 1- sum(myknn_normal_rate_c_w) / iter_no;
myknn_average_attack_rate_c_w(k) = sum(myknn_attack_rate_c_w) / iter_no;
myknn_average_normal_rate_d_w(k) = 1- sum(myknn_normal_rate_d_w) / iter_no;
myknn_average_attack_rate_d_w(k) = sum(myknn_attack_rate_d_w) / iter_no;
myknn_average_rate_a_w(k) = sum(myknn_rate_a_w) / iter_no;
myknn_average_rate_b_w(k) = sum(myknn_rate_b_w) / iter_no;
```

```
myknn_average_rate_c_w(k) = sum(myknn_rate_c_w) / iter_no;
myknn_average_rate_d_w(k) = sum(myknn_rate_d_w) / iter_no;
time_average_myknn_a_w(k) = sum(time_myknn_a_w) / iter_no;
time_average_myknn_b_w(k) = sum(time_myknn_b_w) / iter_no;
time_average_myknn_c_w(k) = sum(time_myknn_c_w) / iter_no;
time_average_myknn_d_w(k) = sum(time_myknn_d_w) / iter_no;
fprintf('\n ENSEMBLE');
ensemble_average_normal_rate_m(k) = 1- sum(ensemble_normal_rate_m) / iter_no;
ensemble_average_attack_rate_m(k) = sum(ensemble_attack_rate_m) / iter_no;
ensemble_average_normal_rate_mv(k) = 1- sum(ensemble_normal_rate_mv) / iter_no;
ensemble_average_attack_rate_mv(k) = sum(ensemble_attack_rate_mv) / iter_no;
ensemble_average_normal_rate_bayes(k) = 1- sum(ensemble_normal_rate_bayes) / iter_no;
ensemble_average_attack_rate_bayes(k) = sum(ensemble_attack_rate_bayes) / iter_no;
ensemble_average_rate_m(k) = sum(ensemble_rate_m) / iter_no;
ensemble_average_rate_mv(k) = sum(ensemble_rate_mv) / iter_no;
ensemble_average_rate_bayes(k) = sum(ensemble_rate_bayes) / iter_no;
time_average_ensemble_bayes(k) = sum(time_ensemble_bayes) / iter_no;
fprintf('\n TREE');
myknn_average_normal_rate_a_tree(k) = 1- sum(myknn_normal_rate_a_tree) / iter_no;
myknn_average_attack_rate_a_tree(k) = sum(myknn_attack_rate_a_tree) / iter_no;
myknn_average_normal_rate_b_tree(k) = 1- sum(myknn_normal_rate_b_tree) / iter_no;
myknn_average_attack_rate_b_tree(k) = sum(myknn_attack_rate_b_tree) / iter_no;
myknn_average_normal_rate_c_tree(k) = 1- sum(myknn_normal_rate_c_tree) / iter_no;
myknn_average_attack_rate_c_tree(k) = sum(myknn_attack_rate_c_tree) / iter_no;
myknn_average_normal_rate_d_tree(k) = 1- sum(myknn_normal_rate_d_tree) / iter_no;
myknn_average_attack_rate_d_tree(k) = sum(myknn_attack_rate_d_tree) / iter_no;
myknn_average_normal_rate_e_tree(k) = 1- sum(myknn_normal_rate_e_tree) / iter_no;
myknn_average_attack_rate_e_tree(k) = sum(myknn_attack_rate_e_tree) / iter_no;
myknn_average_normal_rate_f_tree(k) = 1- sum(myknn_normal_rate_f_tree) / iter_no;
myknn_average_attack_rate_f_tree(k) = sum(myknn_attack_rate_f_tree) / iter_no;
myknn_average_normal_rate_g_tree(k) = 1- sum(myknn_normal_rate_g_tree) / iter_no;
myknn_average_attack_rate_g_tree(k) = sum(myknn_attack_rate_g_tree) / iter_no;
myknn_average_rate_a_tree(k) = sum(myknn_rate_a_tree) / iter_no;
myknn_average_rate_b_tree(k) = sum(myknn_rate_b_tree) / iter_no;
myknn_average_rate_c_tree(k) = sum(myknn_rate_c_tree) / iter_no;
myknn_average_rate_d_tree(k) = sum(myknn_rate_d_tree) / iter_no;
myknn_average_rate_e_tree(k) = sum(myknn_rate_e_tree) / iter_no;
myknn_average_rate_f_tree(k) = sum(myknn_rate_f_tree) / iter_no;
myknn_average_rate_g_tree(k) = sum(myknn_rate_g_tree) / iter_no;
fprintf('\n TREE + ENSEMBLE');
ensemble_average_normal_rate_tree_m(k) = 1- sum(ensemble_normal_rate_tree_m) / iter_no;
ensemble_average_attack_rate_tree_m(k) = sum(ensemble_attack_rate_tree_m) / iter_no;
ensemble_average_normal_rate_tree_mv(k) = 1- sum(ensemble_normal_rate_tree_mv) / iter_no;
ensemble_average_attack_rate_tree_mv(k) = sum(ensemble_attack_rate_tree_mv) / iter_no;
ensemble_average_normal_rate_tree_bayes(k) = 1- sum(ensemble_normal_rate_tree_bayes) / iter_no;
ensemble_average_attack_rate_tree_bayes(k) = sum(ensemble_attack_rate_tree_bayes) / iter_no;
ensemble_average_rate_tree_m(k) = sum(ensemble_rate_tree_m) / iter_no;
ensemble_average_rate_tree_mv(k) = sum(ensemble_rate_tree_mv) / iter_no;
ensemble_average_rate_tree_bayes(k) = sum(ensemble_rate_tree_bayes) / iter_no;
time_average_tree(k) = sum(time_tree) / iter_no;
sum_train_3_number(k,:) = sum(train_3_number(:,:));
sum_train_4_number(k,:) = sum(train_4_number(:,:));
sum_train_5_number(k,:) = sum(train_5_number(:,:));
sum_train_6_number(k,:) = sum(train_6_number(:,:));
```

```
sum_train_7_number(k,:) = sum(train_7_number(:,:));
sum_train_8_number(k,:) = sum(train_8_number(:,:));
sum_train_9_number(k,:) = sum(train_9_number(:,:));
sum_train_10_number(k,:) = sum(train_10_number(:,:));
sum_train_11_number(k,:) = sum(train_11_number(:,:));
sum_train_12_number(k,:) = sum(train_12_number(:,:));
sum_train_13_number(k,:) = sum(train_13_number(:,:));
sum_train_14_number(k,:) = sum(train_14_number(:,:));
sum_train_15_number(k,:) = sum(train_15_number(:,:));
sum_train_16_number(k,:) = sum(train_16_number(:,:));
sum_train_17_number(k,:) = sum(train_17_number(:,:));
sum_train_18_number(k,:) = sum(train_18_number(:,:));
sum_train_19_number(k,:) = sum(train_19_number(:,:));
sum_train_20_number(k,:) = sum(train_20_number(:,:));
sum_train_21_number(k,:) = sum(train_21_number(:,:));
sum_train_22_number(k,:) = sum(train_22_number(:,:));
sum_train_23_number(k,:) = sum(train_23_number(:,:));
sum_train_24_number(k,:) = sum(train_24_number(:,:));
sum_train_25_number(k,:) = sum(train_25_number(:,:));
sum_train_26_number(k,:) = sum(train_26_number(:,:));
sum_train_27_number(k,:) = sum(train_27_number(:,:));
sum_train_28_number(k,:) = sum(train_28_number(:,:));
sum_train_29_number(k,:) = sum(train_29_number(:,:));
sum_train_30_number(k,:) = sum(train_30_number(:,:));
sum_train_31_number(k,:) = sum(train_31_number(:,:));
sum_train_32_number(k,:) = sum(train_32_number(:,:));
sum_train_33_number(k,:) = sum(train_33_number(:,:));
sum_train_34_number(k,:) = sum(train_34_number(:,:));
sum_train_35_number(k,:) = sum(train_35_number(:,:));
sum_train_36_number(k,:) = sum(train_36_number(:,:));
sum_train_37_number(k,:) = sum(train_37_number(:,:));
sum_train_38_number(k,:) = sum(train_38_number(:,:));
sum_train_39_number(k,:) = sum(train_39_number(:,:));
sum_train_40_number(k,:) = sum(train_40_number(:,:));
sum_train_41_number(k,:) = sum(train_41_number(:,:));
sum_total_3_number(k,:) = sum(total_3_number(:,:));
sum_correct_3_number(k,:) = sum(correct_3_number(:,:));
k_correct_3_rate(k,:) = sum_correct_3_number(k,:)/sum_total_3_number(k,:);
sum_total_4_number(k,:) = sum(total_4_number(:,:));
sum_correct_4_number(k,:) = sum(correct_4_number(:,:));
k_correct_4_rate(k,:) = sum_correct_4_number(k,:)/sum_total_4_number(k,:);
sum_total_5_number(k,:) = sum(total_5_number(:,:));
sum_correct_5_number(k,:) = sum(correct_5_number(:,:));
k_correct_5_rate(k,:) = sum_correct_5_number(k,:)/sum_total_5_number(k,:);
sum_total_6_number(k,:) = sum(total_6_number(:,:));
sum_correct_6_number(k,:) = sum(correct_6_number(:,:));
k_correct_6_rate(k,:) = sum_correct_6_number(k,:)/sum_total_6_number(k,:);
sum_total_7_number(k,:) = sum(total_7_number(:,:));
sum_correct_7_number(k,:) = sum(correct_7_number);
k_correct_7_rate(k,:) = sum_correct_7_number(k,:)/sum_total_7_number(k,:);
sum_total_8_number(k,:) = sum(total_8_number(:,:));
sum_correct_8_number(k,:) = sum(correct_8_number(:,:));
k_correct_8_rate(k,:) = sum_correct_8_number(k,:)/sum_total_8_number(k,:);
sum_total_9_number(k,:) = sum(total_9_number(:,:));
```

```
sum_correct_9_number(k,:) = sum(correct_9_number(:,:));
k_correct_9_rate(k,:) = sum_correct_9_number(k,:)/sum_total_9_number(k,:);
sum_total_10_number(k,:) = sum(total_10_number(:,:));
sum_correct_10_number(k,:) = sum(correct_10_number(:,:));
k_correct_10_rate(k,:) = sum_correct_10_number(k,:)/sum_total_10_number(k,:);
sum_total_11_number(k,:) = sum(total_11_number(:,:));
sum_correct_11_number(k,:) = sum(correct_11_number(:,:));
k_correct_11_rate(k,:) = sum_correct_11_number(k,:)/sum_total_11_number(k,:);
sum_total_12_number(k,:) = sum(total_12_number(:,:));
sum_correct_12_number(k,:) = sum(correct_12_number(:,:));
k_correct_12_rate(k,:) = sum_correct_12_number(k,:)/sum_total_12_number(k,:);
sum_total_13_number(k,:) = sum(total_13_number(:,:));
sum_correct_13_number(k,:) = sum(correct_13_number(:,:));
k_correct_13_rate(k,:) = sum_correct_13_number(k,:)/sum_total_13_number(k,:);
sum_total_14_number(k,:) = sum(total_14_number(:,:));
sum_correct_14_number(k,:) = sum(correct_14_number(:,:));
k_correct_14_rate(k,:) = sum_correct_14_number(k,:)/sum_total_14_number(k,:);
sum_total_15_number(k,:) = sum(total_15_number(:,:));
sum_correct_15_number(k,:) = sum(correct_15_number(:,:));
k_correct_15_rate(k,:) = sum_correct_15_number(k,:)/sum_total_15_number(k,:);
sum_total_16_number(k,:) = sum(total_16_number(:,:));
sum_correct_16_number(k,:) = sum(correct_16_number(:,:));
k_correct_16_rate(k,:) = sum_correct_16_number(k,:)/sum_total_16_number(k,:);
sum_total_17_number(k,:) = sum(total_17_number(:,:));
sum_correct_17_number(k,:) = sum(correct_17_number(:,:));
k_correct_17_rate(k,:) = sum_correct_17_number(k,:)/sum_total_17_number(k,:);
sum_total_18_number(k,:) = sum(total_18_number(:,:));
sum_correct_18_number(k,:) = sum(correct_18_number(:,:));
k_correct_18_rate(k,:) = sum_correct_18_number(k,:)/sum_total_18_number(k,:);
sum_total_19_number(k,:) = sum(total_19_number(:,:));
sum_correct_19_number(k,:) = sum(correct_19_number(:,:));
k_correct_19_rate(k,:) = sum_correct_19_number(k,:)/sum_total_19_number(k,:);
sum_total_20_number(k,:) = sum(total_20_number(:,:));
sum_correct_20_number(k,:) = sum(correct_20_number(:,:));
k_correct_20_rate(k,:) = sum_correct_20_number(k,:)/sum_total_20_number(k,:);
sum_total_21_number(k,:) = sum(total_21_number(:,:));
sum_correct_21_number(k,:) = sum(correct_21_number(:,:));
k_correct_21_rate(k,:) = sum_correct_21_number(k,:)/sum_total_21_number(k,:);
sum_total_22_number(k,:) = sum(total_22_number(:,:));
sum_correct_22_number(k,:) = sum(correct_22_number(:,:));
k_correct_22_rate(k,:) = sum_correct_22_number(k,:)/sum_total_22_number(k,:);
sum_total_23_number(k,:) = sum(total_23_number(:,:));
sum_correct_23_number(k,:) = sum(correct_23_number(:,:));
k_correct_23_rate(k,:) = sum_correct_23_number(k,:)/sum_total_23_number(k,:);
sum_total_24_number(k,:) = sum(total_24_number(:,:));
sum_correct_24_number(k,:) = sum(correct_24_number(:,:));
k_correct_24_rate(k,:) = sum_correct_24_number(k,:)/sum_total_24_number(k,:);
sum_total_25_number(k,:) = sum(total_25_number(:,:));
sum_correct_25_number(k,:) = sum(correct_25_number(:,:));
k_correct_25_rate(k,:) = sum_correct_25_number(k,:)/sum_total_25_number(k,:);
sum_total_26_number(k,:) = sum(total_26_number(:,:));
sum_correct_26_number(k,:) = sum(correct_26_number(:,:));
k_correct_26_rate(k,:) = sum_correct_26_number(k,:)/sum_total_26_number(k,:);
sum_total_27_number(k,:) = sum(total_27_number(:,:));
```

```
sum_correct_27_number(k,:) = sum(correct_27_number(:,:));
k_correct_27_rate(k,:) = sum_correct_27_number(k,:)/sum_total_27_number(k,:);
sum_total_28_number(k,:) = sum(total_28_number(:,:));
sum_correct_28_number(k,:) = sum(correct_28_number(:,:));
k_correct_28_rate(k,:) = sum_correct_28_number(k,:)/sum_total_28_number(k,:);
sum_total_29_number(k,:) = sum(total_29_number(:,:));
sum_correct_29_number(k,:) = sum(correct_29_number(:,:));
k_correct_29_rate(k,:) = sum_correct_29_number(k,:)/sum_total_29_number(k,:);
sum_total_30_number(k,:) = sum(total_30_number(:,:));
sum_correct_30_number(k,:) = sum(correct_30_number(:,:));
k_correct_30_rate(k,:) = sum_correct_30_number(k,:)/sum_total_30_number(k,:);
sum_total_31_number(k,:) = sum(total_31_number(:,:));
sum_correct_31_number(k,:) = sum(correct_31_number(:,:));
k_correct_31_rate(k,:) = sum_correct_31_number(k,:)/sum_total_31_number(k,:);
sum_total_32_number(k,:) = sum(total_32_number(:,:));
sum_correct_32_number(k,:) = sum(correct_32_number(:,:));
k_correct_32_rate(k,:) = sum_correct_32_number(k,:)/sum_total_32_number(k,:);
sum_total_33_number(k,:) = sum(total_33_number(:,:));
sum_correct_33_number(k,:) = sum(correct_33_number(:,:));
k_correct_33_rate(k,:) = sum_correct_33_number(k,:)/sum_total_33_number(k,:);
sum_total_34_number(k,:) = sum(total_34_number(:,:));
sum_correct_34_number(k,:) = sum(correct_34_number(:,:));
k_correct_34_rate(k,:) = sum_correct_34_number(k,:)/sum_total_34_number(k,:);
sum_total_35_number(k,:) = sum(total_35_number(:,:));
sum_correct_35_number(k,:) = sum(correct_35_number(:,:));
k_correct_35_rate(k,:) = sum_correct_35_number(k,:)/sum_total_35_number(k,:);
sum_total_36_number(k,:) = sum(total_36_number(:,:));
sum_correct_36_number(k,:) = sum(correct_36_number(:,:));
k_correct_36_rate(k,:) = sum_correct_36_number(k,:)/sum_total_36_number(k,:);
sum_total_37_number(k,:) = sum(total_37_number(:,:));
sum_correct_37_number(k,:) = sum(correct_37_number(:,:));
k_correct_37_rate(k,:) = sum_correct_37_number(k,:)/sum_total_37_number(k,:);
sum_total_38_number(k,:) = sum(total_38_number(:,:));
sum_correct_38_number(k,:) = sum(correct_38_number(:,:));
k_correct_38_rate(k,:) = sum_correct_38_number(k,:)/sum_total_38_number(k,:);
sum_total_39_number(k,:) = sum(total_39_number(:,:));
sum_correct_39_number(k,:) = sum(correct_39_number(:,:));
k_correct_39_rate(k,:) = sum_correct_39_number(k,:)/sum_total_39_number(k,:);
sum_total_40_number(k,:) = sum(total_40_number(:,:));
sum_correct_40_number(k,:) = sum(correct_40_number(:,:));
k_correct_40_rate(k,:) = sum_correct_40_number(k,:)/sum_total_40_number(k,:);
sum_total_41_number(k,:) = sum(total_41_number(:,:));
sum_correct_41_number(k,:) = sum(correct_41_number(:,:));
k_correct_41_rate(k,:) = sum_correct_41_number(k,:)/sum_total_41_number(k,:);
end % nn
final_train_3_number = sum(sum_train_3_number(:,:));
final_train_4_number = sum(sum_train_4_number(:,:));
final_train_5_number = sum(sum_train_5_number(:,:));
final_train_6_number = sum(sum_train_6_number(:,:));
final_train_7_number = sum(sum_train_7_number(:,:));
final_train_8_number = sum(sum_train_8_number(:,:));
final_train_9_number = sum(sum_train_9_number(:,:));
final_train_10_number = sum(sum_train_10_number(:,:));
final_train_11_number = sum(sum_train_11_number(:,:));
```

```
final_train_12_number = sum(sum_train_12_number(:,:));
final_train_13_number = sum(sum_train_13_number(:,:));
final_train_14_number = sum(sum_train_14_number(:,:));
final_train_15_number = sum(sum_train_15_number(:,:));
final_train_16_number = sum(sum_train_16_number(:,:));
final_train_17_number = sum(sum_train_17_number(:,:));
final_train_18_number = sum(sum_train_18_number(:,:));
final_train_19_number = sum(sum_train_19_number(:,:));
final_train_20_number = sum(sum_train_20_number(:,:));
final_train_21_number = sum(sum_train_21_number(:,:));
final_train_22_number = sum(sum_train_22_number(:,:));
final_train_23_number = sum(sum_train_23_number(:,:));
final_train_24_number = sum(sum_train_24_number(:,:));
final_train_25_number = sum(sum_train_25_number(:,:));
final_train_26_number = sum(sum_train_26_number(:,:));
final_train_27_number = sum(sum_train_27_number(:,:));
final_train_28_number = sum(sum_train_28_number(:,:));
final_train_29_number = sum(sum_train_29_number(:,:));
final_train_30_number = sum(sum_train_30_number(:,:));
final_train_31_number = sum(sum_train_31_number(:,:));
final_train_32_number = sum(sum_train_32_number(:,:));
final_train_33_number = sum(sum_train_33_number(:,:));
final_train_34_number = sum(sum_train_34_number(:,:));
final_train_35_number = sum(sum_train_35_number(:,:));
final_train_36_number = sum(sum_train_36_number(:,:));
final_train_37_number = sum(sum_train_37_number(:,:));
final_train_38_number = sum(sum_train_38_number(:,:));
final_train_39_number = sum(sum_train_39_number(:,:));
final_train_40_number = sum(sum_train_40_number(:,:));
final_train_41_number = sum(sum_train_41_number(:,:));
final_total_3_number = sum(sum_total_3_number(:,:));
final_correct_3_number = sum(sum_correct_3_number(:,:));
final_correct_3_rate = final_correct_3_number./final_total_3_number;
final_total_4_number = sum(sum_total_4_number(:,:));
final_correct_4_number = sum(sum_correct_4_number(:,:));
final_correct_4_rate = final_correct_4_number./final_total_4_number;
final_total_5_number = sum(sum_total_5_number(:,:));
final_correct_5_number = sum(sum_correct_5_number(:,:));
final_correct_5_rate = final_correct_5_number./final_total_5_number;
final_total_6_number = sum(sum_total_6_number(:,:));
final_correct_6_number = sum(sum_correct_6_number(:,:));
final_correct_6_rate = final_correct_6_number./final_total_6_number;
final_total_7_number = sum(sum_total_7_number(:,:));
final_correct_7_number = sum(sum_correct_7_number(:,:));
final_correct_7_rate = final_correct_7_number./final_total_7_number;
final_total_8_number = sum(sum_total_8_number(:,:));
final_correct_8_number = sum(sum_correct_8_number(:,:));
final_correct_8_rate = final_correct_8_number./final_total_8_number;
final_total_9_number = sum(sum_total_9_number(:,:));
final_correct_9_number = sum(sum_correct_9_number(:,:));
final_correct_9_rate = final_correct_9_number./final_total_9_number;
final_total_10_number = sum(sum_total_10_number(:,:));
final_correct_10_number = sum(sum_correct_10_number(:,:));
final_correct_10_rate = final_correct_10_number./final_total_10_number;
```

```
final_total_11_number = sum(sum_total_11_number(:,:));
final_correct_11_number = sum(sum_correct_11_number(:,:));
final_correct_11_rate = final_correct_11_number./final_total_11_number;
final_total_12_number = sum(sum_total_12_number(:,:));
final_correct_12_number = sum(sum_correct_12_number(:,:));
final_correct_12_rate = final_correct_12_number./final_total_12_number;
final_total_13_number = sum(sum_total_13_number(:,:));
final_correct_13_number = sum(sum_correct_13_number(:,:));
final_correct_13_rate = final_correct_13_number./final_total_13_number;
final_total_14_number = sum(sum_total_14_number(:,:));
final_correct_14_number = sum(sum_correct_14_number(:,:));
final_correct_14_rate = final_correct_14_number./final_total_14_number;
final_total_15_number = sum(sum_total_15_number(:,:));
final_correct_15_number = sum(sum_correct_15_number(:,:));
final_correct_15_rate = final_correct_15_number./final_total_15_number;
final_total_16_number = sum(sum_total_16_number(:,:));
final_correct_16_number = sum(sum_correct_16_number(:,:));
final_correct_16_rate = final_correct_16_number./final_total_16_number;
final_total_17_number = sum(sum_total_17_number(:,:));
final_correct_17_number = sum(sum_correct_17_number(:,:));
final_correct_17_rate = final_correct_17_number./final_total_17_number;
final_total_18_number = sum(sum_total_18_number(:,:));
final_correct_18_number = sum(sum_correct_18_number(:,:));
final_correct_18_rate = final_correct_18_number./final_total_18_number;
final_total_19_number = sum(sum_total_19_number(:,:));
final_correct_19_number = sum(sum_correct_19_number(:,:));
final_correct_19_rate = final_correct_19_number./final_total_19_number;
final_total_20_number = sum(sum_total_20_number(:,:));
final_correct_20_number = sum(sum_correct_20_number(:,:));
final_correct_20_rate = final_correct_20_number./final_total_20_number;
final_total_21_number = sum(sum_total_21_number(:,:));
final_correct_21_number = sum(sum_correct_21_number(:,:));
final_correct_21_rate = final_correct_21_number./final_total_21_number;
final_total_22_number = sum(sum_total_22_number(:,:));
final_correct_22_number = sum(sum_correct_22_number(:,:));
final_correct_22_rate = final_correct_22_number./final_total_22_number;
final_total_23_number = sum(sum_total_23_number(:,:));
final_correct_23_number = sum(sum_correct_23_number(:,:));
final_correct_23_rate = final_correct_23_number./final_total_23_number;
final_total_24_number = sum(sum_total_24_number(:,:));
final_correct_24_number = sum(sum_correct_24_number(:,:));
final_correct_24_rate = final_correct_24_number./final_total_24_number;
final_total_25_number = sum(sum_total_25_number(:,:));
final_correct_25_number = sum(sum_correct_25_number(:,:));
final_correct_25_rate = final_correct_25_number./final_total_25_number;
final_total_26_number = sum(sum_total_26_number(:,:));
final_correct_26_number = sum(sum_correct_26_number(:,:));
final_correct_26_rate = final_correct_26_number./final_total_26_number;
final_total_27_number = sum(sum_total_27_number(:,:));
final_correct_27_number = sum(sum_correct_27_number(:,:));
final_correct_27_rate = final_correct_27_number./final_total_27_number;
final_total_28_number = sum(sum_total_28_number(:,:));
final_correct_28_number = sum(sum_correct_28_number(:,:));
final_correct_28_rate = final_correct_28_number./final_total_28_number;
```

```
final_total_29_number = sum(sum_total_29_number(:,:));
final_correct_29_number = sum(sum_correct_29_number(:,:));
final_correct_29_rate = final_correct_29_number./final_total_29_number;
final_total_30_number = sum(sum_total_30_number(:,:));
final_correct_30_number = sum(sum_correct_30_number(:,:));
final_correct_30_rate = final_correct_30_number./final_total_30_number;
final_total_31_number = sum(sum_total_31_number(:,:));
final_correct_31_number = sum(sum_correct_31_number(:,:));
final_correct_31_rate = final_correct_31_number./final_total_31_number;
final_total_32_number = sum(sum_total_32_number(:,:));
final_correct_32_number = sum(sum_correct_32_number(:,:));
final_correct_32_rate = final_correct_32_number./final_total_32_number;
final_total_33_number = sum(sum_total_33_number(:,:));
final_correct_33_number = sum(sum_correct_33_number(:,:));
final_correct_33_rate = final_correct_33_number./final_total_33_number;
final_total_34_number = sum(sum_total_34_number(:,:));
final_correct_34_number = sum(sum_correct_34_number(:,:));
final_correct_34_rate = final_correct_34_number./final_total_34_number;
final_total_35_number = sum(sum_total_35_number(:,:));
final_correct_35_number = sum(sum_correct_35_number(:,:));
final_correct_35_rate = final_correct_35_number./final_total_35_number;
final_total_36_number = sum(sum_total_36_number(:,:));
final_correct_36_number = sum(sum_correct_36_number(:,:));
final_correct_36_rate = final_correct_36_number./final_total_36_number;
final_total_37_number = sum(sum_total_37_number(:,:));
final_correct_37_number = sum(sum_correct_37_number(:,:));
final_correct_37_rate = final_correct_37_number./final_total_37_number;
final_total_38_number = sum(sum_total_38_number(:,:));
final_correct_38_number = sum(sum_correct_38_number(:,:));
final_correct_38_rate = final_correct_38_number./final_total_38_number;
final_total_39_number = sum(sum_total_39_number(:,:));
final_correct_39_number = sum(sum_correct_39_number(:,:));
final_correct_39_rate = final_correct_39_number./final_total_39_number;
final_total_40_number = sum(sum_total_40_number(:,:));
final_correct_40_number = sum(sum_correct_40_number(:,:));
final_correct_40_rate = final_correct_40_number./final_total_40_number;
final_total_41_number = sum(sum_total_41_number(:,:));
final_correct_41_number = sum(sum_correct_41_number(:,:));
final_correct_41_rate = final_correct_41_number./final_total_41_number;
% write statistics of ATTACK to a file
fid = fopen('d:\attack.txt','w');
fprintf(fid,'Number of attacks in training set\n');
fprintf(fid,'Number of attacks in testing set\n');
fprintf(fid,'Number of attacks be correctly detected\n');
fprintf(fid,'Detection rates\n');
fprintf(fid,'\nDOS\n\n');
fprintf(fid,'3\n');
fprintf(fid,'%d\t',final_total_3_number);      fprintf(fid,'\n');
fprintf(fid,'%d\t',final_correct_3_number);    fprintf(fid,'\n');
fprintf(fid,'%.2f\t',final_correct_3_rate);    fprintf(fid,'\n');
fprintf(fid,'9\n');
fprintf(fid,'%d\t',final_total_9_number);      fprintf(fid,'\n');
fprintf(fid,'%d\t',final_correct_9_number);    fprintf(fid,'\n');
fprintf(fid,'%.2f\t',final_correct_9_rate);    fprintf(fid,'\n');
```

```
fprintf(fid,'12\n');
fprintf(fid,'%d\t',final_total_12_number);     fprintf(fid,'\n');
fprintf(fid,'%d\t',final_correct_12_number);   fprintf(fid,'\n');
fprintf(fid,'%.2f\t',final_correct_12_rate);   fprintf(fid,'\n');
fprintf(fid,'16\n');
fprintf(fid,'%d\t',final_total_16_number);     fprintf(fid,'\n');
fprintf(fid,'%d\t',final_correct_16_number);   fprintf(fid,'\n');
fprintf(fid,'%.2f\t',final_correct_16_rate);   fprintf(fid,'\n');
fprintf(fid,'20\n');
fprintf(fid,'%d\t',final_total_20_number);     fprintf(fid,'\n');
fprintf(fid,'%d\t',final_correct_20_number);   fprintf(fid,'\n');
fprintf(fid,'%.2f\t',final_correct_20_rate);   fprintf(fid,'\n');
fprintf(fid,'25\n');
fprintf(fid,'%d\t',final_total_25_number);     fprintf(fid,'\n');
fprintf(fid,'%d\t',final_correct_25_number);   fprintf(fid,'\n');
fprintf(fid,'%.2f\t',final_correct_25_rate);   fprintf(fid,'\n');
fprintf(fid,'27\n');
fprintf(fid,'%d\t',final_total_27_number);     fprintf(fid,'\n');
fprintf(fid,'%d\t',final_correct_27_number);   fprintf(fid,'\n');
fprintf(fid,'%.2f\t',final_correct_27_rate);   fprintf(fid,'\n');
fprintf(fid,'30\n');
fprintf(fid,'%d\t',final_total_30_number);     fprintf(fid,'\n');
fprintf(fid,'%d\t',final_correct_30_number);   fprintf(fid,'\n');
fprintf(fid,'%.2f\t',final_correct_30_rate);   fprintf(fid,'\n');
fprintf(fid,'37\n');
fprintf(fid,'%d\t',final_total_37_number);     fprintf(fid,'\n');
fprintf(fid,'%d\t',final_correct_37_number);   fprintf(fid,'\n');
fprintf(fid,'%.2f\t',final_correct_37_rate);   fprintf(fid,'\n');
fprintf(fid,'\nPROBE\n\n');
fprintf(fid,'8\n');
fprintf(fid,'%d\t',final_total_8_number);      fprintf(fid,'\n');
fprintf(fid,'%d\t',final_correct_8_number);    fprintf(fid,'\n');
fprintf(fid,'%.2f\t',final_correct_8_rate);    fprintf(fid,'\n');
fprintf(fid,'13\n');
fprintf(fid,'%d\t',final_total_13_number);     fprintf(fid,'\n');
fprintf(fid,'%d\t',final_correct_13_number);   fprintf(fid,'\n');
fprintf(fid,'%.2f\t',final_correct_13_rate);   fprintf(fid,'\n');
fprintf(fid,'17\n');
fprintf(fid,'%d\t',final_total_17_number);     fprintf(fid,'\n');
fprintf(fid,'%d\t',final_correct_17_number);   fprintf(fid,'\n');
fprintf(fid,'%.2f\t',final_correct_17_rate);   fprintf(fid,'\n');
fprintf(fid,'19\n');
fprintf(fid,'%d\t',final_total_19_number);     fprintf(fid,'\n');
fprintf(fid,'%d\t',final_correct_19_number);   fprintf(fid,'\n');
fprintf(fid,'%.2f\t',final_correct_19_rate);   fprintf(fid,'\n');
fprintf(fid,'28\n');
fprintf(fid,'%d\t',final_total_28_number);     fprintf(fid,'\n');
fprintf(fid,'%d\t',final_correct_28_number);   fprintf(fid,'\n');
fprintf(fid,'%.2f\t',final_correct_28_rate);   fprintf(fid,'\n');
fprintf(fid,'32\n');
fprintf(fid,'%d\t',final_total_32_number);     fprintf(fid,'\n');
fprintf(fid,'%d\t',final_correct_32_number);   fprintf(fid,'\n');
fprintf(fid,'%.2f\t',final_correct_32_rate);   fprintf(fid,'\n');
fprintf(fid,'\nR2L\n\n');
```

```
fprintf(fid,'5\n');
fprintf(fid,'%d\t',final_total_5_number);      fprintf(fid,'\n');
fprintf(fid,'%d\t',final_correct_5_number);    fprintf(fid,'\n');
fprintf(fid,'%.2f\t',final_correct_5_rate);    fprintf(fid,'\n');
fprintf(fid,'6\n');
fprintf(fid,'%d\t',final_total_6_number);      fprintf(fid,'\n');
fprintf(fid,'%d\t',final_correct_6_number);    fprintf(fid,'\n');
fprintf(fid,'%.2f\t',final_correct_6_rate);    fprintf(fid,'\n');
fprintf(fid,'7\n');
fprintf(fid,'%d\t',final_total_7_number);      fprintf(fid,'\n');
fprintf(fid,'%d\t',final_correct_7_number);    fprintf(fid,'\n');
fprintf(fid,'%.2f\t',final_correct_7_rate);    fprintf(fid,'\n');
fprintf(fid,'11\n');
fprintf(fid,'%d\t',final_total_11_number);     fprintf(fid,'\n');
fprintf(fid,'%d\t',final_correct_11_number);   fprintf(fid,'\n');
fprintf(fid,'%.2f\t',final_correct_11_rate);   fprintf(fid,'\n');
fprintf(fid,'15\n');
fprintf(fid,'%d\t',final_total_15_number);     fprintf(fid,'\n');
fprintf(fid,'%d\t',final_correct_15_number);   fprintf(fid,'\n');
fprintf(fid,'%.2f\t',final_correct_15_rate);   fprintf(fid,'\n');
fprintf(fid,'21\n');
fprintf(fid,'%d\t',final_total_21_number);     fprintf(fid,'\n');
fprintf(fid,'%d\t',final_correct_21_number);   fprintf(fid,'\n');
fprintf(fid,'%.2f\t',final_correct_21_rate);   fprintf(fid,'\n');
fprintf(fid,'23\n');
fprintf(fid,'%d\t',final_total_23_number);     fprintf(fid,'\n');
fprintf(fid,'%d\t',final_correct_23_number);   fprintf(fid,'\n');
fprintf(fid,'%.2f\t',final_correct_23_rate);   fprintf(fid,'\n');
fprintf(fid,'24\n');
fprintf(fid,'%d\t',final_total_24_number);     fprintf(fid,'\n');
fprintf(fid,'%d\t',final_correct_24_number);   fprintf(fid,'\n');
fprintf(fid,'%.2f\t',final_correct_24_rate);   fprintf(fid,'\n');
fprintf(fid,'29\n');
fprintf(fid,'%d\t',final_total_29_number);     fprintf(fid,'\n');
fprintf(fid,'%d\t',final_correct_29_number);   fprintf(fid,'\n');
fprintf(fid,'%.2f\t',final_correct_29_rate);   fprintf(fid,'\n');
fprintf(fid,'33\n');
fprintf(fid,'%d\t',final_total_33_number);     fprintf(fid,'\n');
fprintf(fid,'%d\t',final_correct_33_number);   fprintf(fid,'\n');
fprintf(fid,'%.2f\t',final_correct_33_rate);   fprintf(fid,'\n');
fprintf(fid,'34\n');
fprintf(fid,'%d\t',final_total_34_number);     fprintf(fid,'\n');
fprintf(fid,'%d\t',final_correct_34_number);   fprintf(fid,'\n');
fprintf(fid,'%.2f\t',final_correct_34_rate);   fprintf(fid,'\n');
fprintf(fid,'35\n');
fprintf(fid,'%d\t',final_total_35_number);     fprintf(fid,'\n');
fprintf(fid,'%d\t',final_correct_35_number);   fprintf(fid,'\n');
fprintf(fid,'%.2f\t',final_correct_35_rate);   fprintf(fid,'\n');
fprintf(fid,'38\n');
fprintf(fid,'%d\t',final_total_38_number);     fprintf(fid,'\n');
fprintf(fid,'%d\t',final_correct_38_number);   fprintf(fid,'\n');
fprintf(fid,'%.2f\t',final_correct_38_rate);   fprintf(fid,'\n');
fprintf(fid,'39\n');
fprintf(fid,'%d\t',final_total_39_number);     fprintf(fid,'\n');
```

```
fprintf(fid,'%d\t',final_correct_39_number);   fprintf(fid,'\n');
fprintf(fid,'%.2f\t',final_correct_39_rate);   fprintf(fid,'\n');
fprintf(fid,'40\n');
fprintf(fid,'%d\t',final_total_40_number);     fprintf(fid,'\n');
fprintf(fid,'%d\t',final_correct_40_number);   fprintf(fid,'\n');
fprintf(fid,'%.2f\t',final_correct_40_rate);   fprintf(fid,'\n');
fprintf(fid,'\nU2R\n\n');
fprintf(fid,'4\n');
fprintf(fid,'%d\t',final_total_4_number);      fprintf(fid,'\n');
fprintf(fid,'%d\t',final_correct_4_number);    fprintf(fid,'\n');
fprintf(fid,'%.2f\t',final_correct_4_rate);    fprintf(fid,'\n');
fprintf(fid,'10\n');
fprintf(fid,'%d\t',final_total_10_number);     fprintf(fid,'\n');
fprintf(fid,'%d\t',final_correct_10_number);   fprintf(fid,'\n');
fprintf(fid,'%.2f\t',final_correct_10_rate);   fprintf(fid,'\n');
fprintf(fid,'14\n');
fprintf(fid,'%d\t',final_total_14_number);     fprintf(fid,'\n');
fprintf(fid,'%d\t',final_correct_14_number);   fprintf(fid,'\n');
fprintf(fid,'%.2f\t',final_correct_14_rate);   fprintf(fid,'\n');
fprintf(fid,'18\n');
fprintf(fid,'%d\t',final_total_18_number);     fprintf(fid,'\n');
fprintf(fid,'%d\t',final_correct_18_number);   fprintf(fid,'\n');
fprintf(fid,'%.2f\t',final_correct_18_rate);   fprintf(fid,'\n');
fprintf(fid,'22\n');
fprintf(fid,'%d\t',final_total_22_number);     fprintf(fid,'\n');
fprintf(fid,'%d\t',final_correct_22_number);   fprintf(fid,'\n');
fprintf(fid,'%.2f\t',final_correct_22_rate);   fprintf(fid,'\n');
fprintf(fid,'26\n');
fprintf(fid,'%d\t',final_total_26_number);     fprintf(fid,'\n');
fprintf(fid,'%d\t',final_correct_26_number);   fprintf(fid,'\n');
fprintf(fid,'%.2f\t',final_correct_26_rate);   fprintf(fid,'\n');
fprintf(fid,'31\n');
fprintf(fid,'%d\t',final_total_31_number);     fprintf(fid,'\n');
fprintf(fid,'%d\t',final_correct_31_number);   fprintf(fid,'\n');
fprintf(fid,'%.2f\t',final_correct_31_rate);   fprintf(fid,'\n');
fprintf(fid,'36\n');
fprintf(fid,'%d\t',final_total_36_number);     fprintf(fid,'\n');
fprintf(fid,'%d\t',final_correct_36_number);   fprintf(fid,'\n');
fprintf(fid,'%.2f\t',final_correct_36_rate);   fprintf(fid,'\n');
fprintf(fid,'41\n');
fprintf(fid,'%d\t',final_total_41_number);     fprintf(fid,'\n');
fprintf(fid,'%d\t',final_correct_41_number);   fprintf(fid,'\n');
fprintf(fid,'%.2f\t',final_correct_41_rate);   fprintf(fid,'\n');
fclose(fid);
% write COMPUTATION TIME to a file
% TIME
fid = fopen('d:\time.txt','w');
fprintf(fid,'KNN');
fprintf(fid,'\nknn_a\t');
fprintf(fid,'%f\t',time_average_knn_a);
fprintf(fid,'\nknn_b\t');
fprintf(fid,'%f\t',time_average_knn_b);
fprintf(fid,'\nknn_c\t');
fprintf(fid,'%f\t',time_average_knn_c);
```

```
fprintf(fid,'\nknn_d\t');
fprintf(fid,'%f\t',time_average_knn_d);
fprintf(fid,'\nknn_e\t');
fprintf(fid,'%f\t',time_average_knn_e);
fprintf(fid,'\nknn_f\t');
fprintf(fid,'%f\t',time_average_knn_f);
fprintf(fid,'\nknn_g\t');
fprintf(fid,'%f\t',time_average_knn_g);
fprintf(fid,'\n\nFKNN');
fprintf(fid,'\nfknn_a\t');
fprintf(fid,'%f\t',time_average_fknn_a);
fprintf(fid,'\nfknn_b\t');
fprintf(fid,'%f\t',time_average_fknn_b);
fprintf(fid,'\nfknn_c\t');
fprintf(fid,'%f\t',time_average_fknn_c);
fprintf(fid,'\nfknn_d\t');
fprintf(fid,'%f\t',time_average_fknn_d);
fprintf(fid,'\nfknn_e\t');
fprintf(fid,'%f\t',time_average_fknn_e);
fprintf(fid,'\nfknn_f\t');
fprintf(fid,'%f\t',time_average_fknn_f);
fprintf(fid,'\nfknn_g\t');
fprintf(fid,'%f\t',time_average_fknn_g);
fprintf(fid,'\n\nETKNN');
fprintf(fid,'\netknn_a\t');
fprintf(fid,'%f\t',time_average_etknn_a);
fprintf(fid,'\netknn_b\t');
fprintf(fid,'%f\t',time_average_etknn_b);
fprintf(fid,'\netknn_c\t');
fprintf(fid,'%f\t',time_average_etknn_c);
fprintf(fid,'\netknn_d\t');
fprintf(fid,'%f\t',time_average_etknn_d);
fprintf(fid,'\netknn_e\t');
fprintf(fid,'%f\t',time_average_etknn_e);
fprintf(fid,'\netknn_f\t');
fprintf(fid,'%f\t',time_average_etknn_f);
fprintf(fid,'\netknn_g\t');
fprintf(fid,'%f\t',time_average_etknn_g);
fprintf(fid,'\n\nMYKNN');
fprintf(fid,'\nmyknn_a\t');
fprintf(fid,'%f\t',time_average_myknn_a);
fprintf(fid,'\nmyknn_b\t');
fprintf(fid,'%f\t',time_average_myknn_b);
fprintf(fid,'\nmyknn_c\t');
fprintf(fid,'%f\t',time_average_myknn_c);
fprintf(fid,'\nmyknn_d\t');
fprintf(fid,'%f\t',time_average_myknn_d);
fprintf(fid,'\nmyknn_e\t');
fprintf(fid,'%f\t',time_average_myknn_e);
fprintf(fid,'\nmyknn_f\t');
fprintf(fid,'%f\t',time_average_myknn_f);
fprintf(fid,'\nmyknn_g\t');
fprintf(fid,'%f\t',time_average_myknn_g);
fprintf(fid,'\n\nMYKNN-W');
```

```
fprintf(fid,'\nmyknn_a_w\t');
fprintf(fid,'%f\t',time_average_myknn_a_w);
fprintf(fid,'\nmyknn_b_w\t');
fprintf(fid,'%f\t',time_average_myknn_b_w);
fprintf(fid,'\nmyknn_c_w\t');
fprintf(fid,'%f\t',time_average_myknn_c_w);
fprintf(fid,'\nmyknn_d_w\t');
fprintf(fid,'%f\t',time_average_myknn_d_w);
fprintf(fid,'\n\nENSEMBLE');
fprintf(fid,'\nt_ensemble_bayes\t');
fprintf(fid,'%f\t',time_average_ensemble_bayes);
fprintf(fid,'\n\nTREE');
fprintf(fid,'\nt_tree\t');
fprintf(fid,'%f\t',time_average_tree);
fclose(fid);
% write CLASSIFICATION RATE to a file
fid = fopen('d:\result.txt','w');
fprintf(fid,'KNN');
fprintf(fid,'\nknn_average_normal_rate_a\t');
fprintf(fid,'%f\t',knn_average_normal_rate_a);
fprintf(fid,'\nknn_average_normal_rate_b\t');
fprintf(fid,'%f\t',knn_average_normal_rate_b);
fprintf(fid,'\nknn_average_normal_rate_c\t');
fprintf(fid,'%f\t',knn_average_normal_rate_c);
fprintf(fid,'\nknn_average_normal_rate_d\t');
fprintf(fid,'%f\t',knn_average_normal_rate_d);
fprintf(fid,'\nknn_average_normal_rate_e\t');
fprintf(fid,'%f\t',knn_average_normal_rate_e);
fprintf(fid,'\nknn_average_normal_rate_f\t');
fprintf(fid,'%f\t',knn_average_normal_rate_f);
fprintf(fid,'\nknn_average_normal_rate_g\t');
fprintf(fid,'%f\t',knn_average_normal_rate_g);
fprintf(fid,'\n');
fprintf(fid,'\nknn_average_attack_rate_a\t');
fprintf(fid,'%f\t',knn_average_attack_rate_a);
fprintf(fid,'\nknn_average_attack_rate_b\t');
fprintf(fid,'%f\t',knn_average_attack_rate_b);
fprintf(fid,'\nknn_average_attack_rate_c\t');
fprintf(fid,'%f\t',knn_average_attack_rate_c);
fprintf(fid,'\nknn_average_attack_rate_d\t');
fprintf(fid,'%f\t',knn_average_attack_rate_d);
fprintf(fid,'\nknn_average_attack_rate_e\t');
fprintf(fid,'%f\t',knn_average_attack_rate_e);
fprintf(fid,'\nknn_average_attack_rate_f\t');
fprintf(fid,'%f\t',knn_average_attack_rate_f);
fprintf(fid,'\nknn_average_attack_rate_g\t');
fprintf(fid,'%f\t',knn_average_attack_rate_g);
fprintf(fid,'\n');
fprintf(fid,'\nknn_average_rate_a\t');
fprintf(fid,'%f\t',knn_average_rate_a);
fprintf(fid,'\nknn_average_rate_b\t');
fprintf(fid,'%f\t',knn_average_rate_b);
fprintf(fid,'\nknn_average_rate_c\t');
fprintf(fid,'%f\t',knn_average_rate_c);
```

```
fprintf(fid,'\nknn_average_rate_d\t');
fprintf(fid,'%f\t',knn_average_rate_d);
fprintf(fid,'\nknn_average_rate_e\t');
fprintf(fid,'%f\t',knn_average_rate_e);
fprintf(fid,'\nknn_average_rate_f\t');
fprintf(fid,'%f\t',knn_average_rate_f);
fprintf(fid,'\nknn_average_rate_g\t');
fprintf(fid,'%f\t',knn_average_rate_g);
fprintf(fid,'\n\nFKNN');
fprintf(fid,'\nfknn_average_normal_rate_a\t');
fprintf(fid,'%f\t',fknn_average_normal_rate_a);
fprintf(fid,'\nfknn_average_normal_rate_b\t');
fprintf(fid,'%f\t',fknn_average_normal_rate_b);
fprintf(fid,'\nfknn_average_normal_rate_c\t');
fprintf(fid,'%f\t',fknn_average_normal_rate_c);
fprintf(fid,'\nfknn_average_normal_rate_d\t');
fprintf(fid,'%f\t',fknn_average_normal_rate_d);
fprintf(fid,'\nfknn_average_normal_rate_e\t');
fprintf(fid,'%f\t',fknn_average_normal_rate_e);
fprintf(fid,'\nfknn_average_normal_rate_f\t');
fprintf(fid,'%f\t',fknn_average_normal_rate_f);
fprintf(fid,'\nfknn_average_normal_rate_g\t');
fprintf(fid,'%f\t',fknn_average_normal_rate_g);
fprintf(fid,'\n');
fprintf(fid,'\nfknn_average_attack_rate_a\t');
fprintf(fid,'%f\t',fknn_average_attack_rate_a);
fprintf(fid,'\nfknn_average_attack_rate_b\t');
fprintf(fid,'%f\t',fknn_average_attack_rate_b);
fprintf(fid,'\nfknn_average_attack_rate_c\t');
fprintf(fid,'%f\t',fknn_average_attack_rate_c);
fprintf(fid,'\nfknn_average_attack_rate_d\t');
fprintf(fid,'%f\t',fknn_average_attack_rate_d);
fprintf(fid,'\nfknn_average_attack_rate_e\t');
fprintf(fid,'%f\t',fknn_average_attack_rate_e);
fprintf(fid,'\nfknn_average_attack_rate_f\t');
fprintf(fid,'%f\t',fknn_average_attack_rate_e);
fprintf(fid,'\nfknn_average_attack_rate_g\t');
fprintf(fid,'%f\t',fknn_average_attack_rate_g);
fprintf(fid,'\n');
fprintf(fid,'\nfknn_average_rate_a\t');
fprintf(fid,'%f\t',fknn_average_rate_a);
fprintf(fid,'\nfknn_average_rate_b\t');
fprintf(fid,'%f\t',fknn_average_rate_b);
fprintf(fid,'\nfknn_average_rate_c\t');
fprintf(fid,'%f\t',fknn_average_rate_c);
fprintf(fid,'\nfknn_average_rate_d\t');
fprintf(fid,'%f\t',fknn_average_rate_d);
fprintf(fid,'\nfknn_average_rate_e\t');
fprintf(fid,'%f\t',fknn_average_rate_e);
fprintf(fid,'\nfknn_average_rate_f\t');
fprintf(fid,'%f\t',fknn_average_rate_f);
fprintf(fid,'\nfknn_average_rate_g\t');
fprintf(fid,'%f\t',fknn_average_rate_g);
fprintf(fid,'\n\nETKNN');
```

```
fprintf(fid,'\netknn_average_normal_rate_a\t');
fprintf(fid,'%f\t',etknn_average_normal_rate_a);
fprintf(fid,'\netknn_average_normal_rate_b\t');
fprintf(fid,'%f\t',etknn_average_normal_rate_b);
fprintf(fid,'\netknn_average_normal_rate_c\t');
fprintf(fid,'%f\t',etknn_average_normal_rate_c);
fprintf(fid,'\netknn_average_normal_rate_d\t');
fprintf(fid,'%f\t',etknn_average_normal_rate_d);
fprintf(fid,'\netknn_average_normal_rate_e\t');
fprintf(fid,'%f\t',etknn_average_normal_rate_e);
fprintf(fid,'\netknn_average_normal_rate_f\t');
fprintf(fid,'%f\t',etknn_average_normal_rate_f);
fprintf(fid,'\netknn_average_normal_rate_g\t');
fprintf(fid,'%f\t',etknn_average_normal_rate_g);
fprintf(fid,'\n');
fprintf(fid,'\netknn_average_attack_rate_a\t');
fprintf(fid,'%f\t',etknn_average_attack_rate_a);
fprintf(fid,'\netknn_average_attack_rate_b\t');
fprintf(fid,'%f\t',etknn_average_attack_rate_b);
fprintf(fid,'\netknn_average_attack_rate_c\t');
fprintf(fid,'%f\t',etknn_average_attack_rate_c);
fprintf(fid,'\netknn_average_attack_rate_d\t');
fprintf(fid,'%f\t',etknn_average_attack_rate_d);
fprintf(fid,'\netknn_average_attack_rate_e\t');
fprintf(fid,'%f\t',etknn_average_attack_rate_e);
fprintf(fid,'\netknn_average_attack_rate_f\t');
fprintf(fid,'%f\t',etknn_average_attack_rate_f);
fprintf(fid,'\netknn_average_attack_rate_g\t');
fprintf(fid,'%f\t',etknn_average_attack_rate_g);
fprintf(fid,'\n');
fprintf(fid,'\netknn_average_rate_a\t');
fprintf(fid,'%f\t',etknn_average_rate_a);
fprintf(fid,'\netknn_average_rate_b\t');
fprintf(fid,'%f\t',etknn_average_rate_b);
fprintf(fid,'\netknn_average_rate_c\t');
fprintf(fid,'%f\t',etknn_average_rate_c);
fprintf(fid,'\netknn_average_rate_d\t');
fprintf(fid,'%f\t',etknn_average_rate_d);
fprintf(fid,'\netknn_average_rate_e\t');
fprintf(fid,'%f\t',etknn_average_rate_e);
fprintf(fid,'\netknn_average_rate_f\t');
fprintf(fid,'%f\t',etknn_average_rate_f);
fprintf(fid,'\netknn_average_rate_g\t');
fprintf(fid,'%f\t',etknn_average_rate_g);
fprintf(fid,'\n\nMYKNN');
fprintf(fid,'\nmyknn_average_normal_rate_a\t');
fprintf(fid,'%f\t',myknn_average_normal_rate_a);
fprintf(fid,'\nmyknn_average_normal_rate_b\t');
fprintf(fid,'%f\t',myknn_average_normal_rate_b);
fprintf(fid,'\nmyknn_average_normal_rate_c\t');
fprintf(fid,'%f\t',myknn_average_normal_rate_c);
fprintf(fid,'\nmyknn_average_normal_rate_d\t');
fprintf(fid,'%f\t',myknn_average_normal_rate_d);
fprintf(fid,'\nmyknn_average_normal_rate_e\t');
```

```
fprintf(fid,'%f\t',myknn_average_normal_rate_e);
fprintf(fid,'\nmyknn_average_normal_rate_f\t');
fprintf(fid,'%f\t',myknn_average_normal_rate_f);
fprintf(fid,'\nmyknn_average_normal_rate_g\t');
fprintf(fid,'%f\t',myknn_average_normal_rate_g);
fprintf(fid,'\n');
fprintf(fid,'\nmyknn_average_attack_rate_a\t');
fprintf(fid,'%f\t',myknn_average_attack_rate_a);
fprintf(fid,'\nmyknn_average_attack_rate_b\t');
fprintf(fid,'%f\t',myknn_average_attack_rate_b);
fprintf(fid,'\nmyknn_average_attack_rate_c\t');
fprintf(fid,'%f\t',myknn_average_attack_rate_c);
fprintf(fid,'\nmyknn_average_attack_rate_d\t');
fprintf(fid,'%f\t',myknn_average_attack_rate_d);
fprintf(fid,'\nmyknn_average_attack_rate_e\t');
fprintf(fid,'%f\t',myknn_average_attack_rate_e);
fprintf(fid,'\nmyknn_average_attack_rate_f\t');
fprintf(fid,'%f\t',myknn_average_attack_rate_f);
fprintf(fid,'\nmyknn_average_attack_rate_g\t');
fprintf(fid,'%f\t',myknn_average_attack_rate_g);
fprintf(fid,'\n');
fprintf(fid,'\nmyknn_average_rate_a\t');
fprintf(fid,'%f\t',myknn_average_rate_a);
fprintf(fid,'\nmyknn_average_rate_b\t');
fprintf(fid,'%f\t',myknn_average_rate_b);
fprintf(fid,'\nmyknn_average_rate_c\t');
fprintf(fid,'%f\t',myknn_average_rate_c);
fprintf(fid,'\nmyknn_average_rate_d\t');
fprintf(fid,'%f\t',myknn_average_rate_d);
fprintf(fid,'\nmyknn_average_rate_e\t');
fprintf(fid,'%f\t',myknn_average_rate_e);
fprintf(fid,'\nmyknn_average_rate_f\t');
fprintf(fid,'%f\t',myknn_average_rate_f);
fprintf(fid,'\nmyknn_average_rate_g\t');
fprintf(fid,'%f\t',myknn_average_rate_g);
fprintf(fid,'\n\nMYKNN-W');
fprintf(fid,'\nmyknn_average_normal_rate_a_w\t');
fprintf(fid,'%f\t',myknn_average_normal_rate_a_w);
fprintf(fid,'\nmyknn_average_normal_rate_b_w\t');
fprintf(fid,'%f\t',myknn_average_normal_rate_b_w);
fprintf(fid,'\nmyknn_average_normal_rate_c_w\t');
fprintf(fid,'%f\t',myknn_average_normal_rate_c_w);
fprintf(fid,'\nmyknn_average_normal_rate_d_w\t');
fprintf(fid,'%f\t',myknn_average_normal_rate_d_w);
fprintf(fid,'\n');
fprintf(fid,'\nmyknn_average_attack_rate_a_w\t');
fprintf(fid,'%f\t',myknn_average_attack_rate_a_w);
fprintf(fid,'\nmyknn_average_attack_rate_b_w\t');
fprintf(fid,'%f\t',myknn_average_attack_rate_b_w);
fprintf(fid,'\nmyknn_average_attack_rate_c_w\t');
fprintf(fid,'%f\t',myknn_average_attack_rate_c_w);
fprintf(fid,'\nmyknn_average_attack_rate_d_w\t');
fprintf(fid,'%f\t',myknn_average_attack_rate_d_w);
fprintf(fid,'\n');
```

```
fprintf(fid,'\nmyknn_average_rate_a_w\t');
fprintf(fid,'%f\t',myknn_average_rate_a_w);
fprintf(fid,'\nmyknn_average_rate_b_w\t');
fprintf(fid,'%f\t',myknn_average_rate_b_w);
fprintf(fid,'\nmyknn_average_rate_c_w\t');
fprintf(fid,'%f\t',myknn_average_rate_c_w);
fprintf(fid,'\nmyknn_average_rate_d_w\t');
fprintf(fid,'%f\t',myknn_average_rate_d_w);
fprintf(fid,'\n\nENSEMBLE');
fprintf(fid,'\nensemble_average_normal_rate_m\t');
fprintf(fid,'%f\t',ensemble_average_normal_rate_m);
fprintf(fid,'\nensemble_average_normal_rate_mv\t');
fprintf(fid,'%f\t',ensemble_average_normal_rate_mv);
fprintf(fid,'\nensemble_average_normal_rate_bayes\t');
fprintf(fid,'%f\t',ensemble_average_normal_rate_bayes);
fprintf(fid,'\n');
fprintf(fid,'\nensemble_average_attack_rate_m\t');
fprintf(fid,'%f\t',ensemble_average_attack_rate_m);
fprintf(fid,'\nensemble_average_attack_rate_mv\t');
fprintf(fid,'%f\t',ensemble_average_attack_rate_mv);
fprintf(fid,'\nensemble_average_attack_rate_bayes\t');
fprintf(fid,'%f\t',ensemble_average_attack_rate_bayes);
fprintf(fid,'\n');
fprintf(fid,'\nensemble_average_rate_m\t');
fprintf(fid,'%f\t',ensemble_average_rate_m);
fprintf(fid,'\nensemble_average_rate_mv\t');
fprintf(fid,'%f\t',ensemble_average_rate_mv);
fprintf(fid,'\nensemble_average_rate_bayes\t');
fprintf(fid,'%f\t',ensemble_average_rate_bayes);
fprintf(fid,'\n\nTREE');
fprintf(fid,'\nmyknn_average_normal_rate_a_tree\t');
fprintf(fid,'%f\t',myknn_average_normal_rate_a_tree);
fprintf(fid,'\nmyknn_average_normal_rate_b_tree\t');
fprintf(fid,'%f\t',myknn_average_normal_rate_b_tree);
fprintf(fid,'\nmyknn_average_normal_rate_c_tree\t');
fprintf(fid,'%f\t',myknn_average_normal_rate_c_tree);
fprintf(fid,'\nmyknn_average_normal_rate_d_tree\t');
fprintf(fid,'%f\t',myknn_average_normal_rate_d_tree);
fprintf(fid,'\nmyknn_average_normal_rate_e_tree\t');
fprintf(fid,'%f\t',myknn_average_normal_rate_e_tree);
fprintf(fid,'\nmyknn_average_normal_rate_f_tree\t');
fprintf(fid,'%f\t',myknn_average_normal_rate_f_tree);
fprintf(fid,'\nmyknn_average_normal_rate_g_tree\t');
fprintf(fid,'%f\t',myknn_average_normal_rate_g_tree);
fprintf(fid,'\n');
fprintf(fid,'\nmyknn_average_attack_rate_a_tree\t');
fprintf(fid,'%f\t',myknn_average_attack_rate_a_tree);
fprintf(fid,'\nmyknn_average_attack_rate_b_tree\t');
fprintf(fid,'%f\t',myknn_average_attack_rate_b_tree);
fprintf(fid,'\nmyknn_average_attack_rate_c_tree\t');
fprintf(fid,'%f\t',myknn_average_attack_rate_c_tree);
fprintf(fid,'\nmyknn_average_attack_rate_d_tree\t');
fprintf(fid,'%f\t',myknn_average_attack_rate_d_tree);
fprintf(fid,'\nmyknn_average_attack_rate_e_tree\t');
```

```
fprintf(fid,'%f\t',myknn_average_attack_rate_e_tree);
fprintf(fid,'\nmyknn_average_attack_rate_f_tree\t');
fprintf(fid,'%f\t',myknn_average_attack_rate_f_tree);
fprintf(fid,'\nmyknn_average_attack_rate_g_tree\t');
fprintf(fid,'%f\t',myknn_average_attack_rate_g_tree);
fprintf(fid,'\n');
fprintf(fid,'\nmyknn_average_rate_a_tree\t');
fprintf(fid,'%f\t',myknn_average_rate_a_tree);
fprintf(fid,'\nmyknn_average_rate_b_tree\t');
fprintf(fid,'%f\t',myknn_average_rate_b_tree);
fprintf(fid,'\nmyknn_average_rate_c_tree\t');
fprintf(fid,'%f\t',myknn_average_rate_c_tree);
fprintf(fid,'\nmyknn_average_rate_d_tree\t');
fprintf(fid,'%f\t',myknn_average_rate_d_tree);
fprintf(fid,'\nmyknn_average_rate_e_tree\t');
fprintf(fid,'%f\t',myknn_average_rate_e_tree);
fprintf(fid,'\nmyknn_average_rate_f_tree\t');
fprintf(fid,'%f\t',myknn_average_rate_f_tree);
fprintf(fid,'\nmyknn_average_rate_g_tree\t');
fprintf(fid,'%f\t',myknn_average_rate_g_tree);
fprintf(fid,'\n\nTREE + ENSEMBLE');
fprintf(fid,'\nensemble_average_normal_rate_tree_m\t');
fprintf(fid,'%f\t',ensemble_average_normal_rate_tree_m);
fprintf(fid,'\nensemble_average_normal_rate_tree_mv\t');
fprintf(fid,'%f\t',ensemble_average_normal_rate_tree_mv);
fprintf(fid,'\nensemble_average_normal_rate_tree_bayes\t');
fprintf(fid,'%f\t',ensemble_average_normal_rate_tree_bayes);
fprintf(fid,'\n');
fprintf(fid,'\nensemble_average_attack_rate_tree_m\t');
fprintf(fid,'%f\t',ensemble_average_attack_rate_tree_m);
fprintf(fid,'\nensemble_average_attack_rate_tree_mv\t');
fprintf(fid,'%f\t',ensemble_average_attack_rate_tree_mv);
fprintf(fid,'\nensemble_average_attack_rate_tree_bayes\t');
fprintf(fid,'%f\t',ensemble_average_attack_rate_tree_bayes);
fprintf(fid,'\n');
fprintf(fid,'\nensemble_average_rate_tree_m\t');
fprintf(fid,'%f\t',ensemble_average_rate_tree_m);
fprintf(fid,'\nensemble_average_rate_tree_mv\t');
fprintf(fid,'%f\t',ensemble_average_rate_tree_mv);
fprintf(fid,'\nensemble_average_rate_tree_bayes\t');
fprintf(fid,'%f\t',ensemble_average_rate_tree_bayes);
fclose(fid);
fprintf('\nDONE\n\n');
beep;
```

VITA

TE-SHUN CHOU

1984-1989	B.S., Electronic Engineering Feng Chia University Taichung, Taiwan, R.O.C.
1990-1992	M.S., Electrical Engineering Florida International University Miami, FL
1992-1994	Elan Microelectronics Corporation, Taiwan, R.O.C.
1994-1999	The Overseas Chinese Institute of Technology, Taiwan, R.O.C.
1999-2002	University of British Columbia, Vancouver, BC, Canada
2003-2005	Porter Electronics Ltd., Richmond, BC, Canada
2005-2007	Doctoral Candidate, Electrical Engineering Florida International University Miami, FL

PUBLICATIONS AND PRESENTATIONS

Te-Shun Chou, Kang K. Yen, Niki Pissinou, and Kia Makki. (2007). *Ensemble of Multiple Classifiers in Network Intrusion Detection Design*. Computers & Security. (in review)

Te-Shun Chou, Kang K. Yen, Jun Luo. (2007). *Network Intrusion Detection Design Using Feature Selection of Soft Computing Paradigms*. International Journal of Computational Intelligence, Volume 4, Number 3, pp. 205-217.

Te-Shun Chou, Kang K. Yen, Niki Pissinou, and Kia Makki. (November, 2007). *Fuzzy Belief Reasoning for Intrusion Detection Design*. IEEE The third International Conference on Intelligent Information Hiding and Multimedia Signal Processing, Kaohsiung, Taiwan.

Te-Shun Chou, Kang K. Yen, Jun Luo, Niki Pissinou, and Kia Makki. (October, 2007). *Correlation-Based Feature Selection for Intrusion Detection Design*. IEEE Military Communications Conference, Orlando, FL.

Te-Shun Chou, Kang K. Yen, Liwei An, Niki Pissinou, and Kia Makki. (October, 2007). *Fuzzy Belief Pattern Classification of Incomplete Data*. IEEE International Conference on Systems, Man and Cybernetics, pp. 535-540, Montreal, Quebec, Canada.

Te-Shun Chou, Kang K. Yen, and Jun Luo. (July, 2007). *Feature Reduction and Fuzzy Belief Intrusion Detection Design*. The 11th World Multi-Conference on Systemics, Cybernetics and Informatics jointly with The 13th International Conference on Information Systems Analysis and Synthesis**,** pp. 262-267**,** Orlando, FL.

Te-Shun Chou and Kang K. Yen. (June, 2007). *Fuzzy Belief k-Nearest Neighbors Anomaly Detection of User to Root and Remote to Local Attacks*. 8th Annual IEEE SMC Information Assurance Workshop, pp. 207-213, West Point, NY.

Edward T. Lee and Te-Shun Chou. (2001). *Fuzzy Monotone Functions and Applications*. Kybernetes, Volume 30, Number 1&2, pp. 84-97.

Te-Shun Chou and Sidney S. Fels. (March, 2001). *Hand Modeling for Adaptive Interfaces, Techniques for Geometry and Prediction of Hand Models*. BC Advanced Systems Institute Exchange, Poster, Vancouver, BC.

Te-Shun Chou and Edward T. Lee. (May, 1998). *Fuzzy Monotone Functions and Applications*. IEEE World Congress on Computational Intelligence, pp. 829-834, Anchorage, AK.

Edward T. Lee and Te-Shun Chou. (1995). *Fuzzy Threshold Functions and Applications*. Kybernetes, Volume 24, Number 7, pp. 99-122.

Te-Shun Chou, Jyn-Guo Hwang, and Rong-Da Chung. (September, 1994). *Two New Strategies of Membership Function Circuit and Defuzzifier for a Fuzzy Logic Controller*. Second National Conference on Fuzzy Theory and Applications, pp. 172-175, Taipei, Taiwan.

Te-Shun Chou and Jyn-Guo Hwang. (September, 1994). *An architecture of a Cascadable Digital Fuzzy Processor by Using Rule Break Up Method*. Second National Conference on Fuzzy Theory and Applications, pp. 176-180, Taipei, Taiwan.

Te-Shun Chou, Jyn-Guo Hwang, and Lih-Chiou Lin. (August, 1993). *Design of a Fuzzy Logic Processor With Active Rules Methods*. Forth VLSI Design/CAD Workshop, pp. 156-160, Taipei, Taiwan.